マルチキャストによるクラスタWebサーバの構築

Multicast Based Cluster Web Server

瀧ヶ平 健* S. Sedukhin†
T. Takighira and S. Sedukhin

概要

近年 Web サーバには性能だけでなく信頼性や拡張性、費用対効果が求められている。従来の Web サーバではこれらの要求をすべて満たすことは難しかったが、クラスタリング技術によるクラスタ Web サーバはこれらを満たすことができる。しかし既存のクラスタ Web サーバシステムにはその設計上の問題から信頼性を低下させている物がある。そこで本論文では信頼性の向上に重点をおいたクラスタ Web サーバシステムを提案する。このシステムではクライアントからのリクエストをクラスタ内にマルチキャストすることによりリクエストの分散処理の負荷と機能を全 Web サーバ上に分散した。また、システム全体を統括する部分を作らずに自律している各 Web サーバが他 Web サーバと協調することによりクラスタ Web サーバ全体を管理するするように設計した。これにより階層構造や集中制御される部分を持たないシステム構成となりボトルネックやシステム上の弱点を排除した、信頼性の高いシステムを設計することができた。

1 はじめに

近年のインターネット人口の爆発的な増加に伴い Web サーバは常により高い性能を求められている。利用者からのアクセスが日々増加し、Web サーバの処理能力を超えようとする場合には素早く Web サーバの性能を向上させることも必要とされる。また近年、特に商用利用の場において、Web サーバは単に利用者に何らかのデータを供給するだけに留まらず、より重要なサービスの提供を行うようになってきた。これにより Web サーバには高い性能だけでなく高い信頼性も要求されるようになってきている。しかしこれらの要求を従来通りの Web サーバで満たすのは非常に困難であるか、または非常に費用対効果の悪い物になってしまう。

この様な状況を背景として現在 Web サーバのクラスタ化が注目されている。本論文でのクラスタ化とは「何らかの手法により利用者からの Web サーバへの多数のアクセス要求を複数の Web サーバに分散して処理する」もの全てを指す。クラスタ化により上述した近年の Web サーバに対する要求を比較的安価に満たすことができる。以下、複数の Web サーバをクラスタ化する為のシステムをクラスタシステムと呼び、クラスタ化した複数の Web サーバの集合をクラスタサーバと呼ぶ。またクラスタサーバを構成する複数の Web サーバのそれぞれをクラスタノードと呼ぶ。

クラスタサーバの処理能力は複数のクラスタノードの処理能力を合わせたものであるので、安価な Web サーバを複数使用して非常に高い性能をもつクラスタサーバを構築することができる。またクラスタサーバの性能の向上が必要になった場合は新たな Web サーバをクラスタノードとしてクラスタサーバに追加するだけで容易・安価に性能を向上させることができる。

信頼性については、通常はクラスタシステムにはクラスタノードの監視機能が含まれており、クラスタノードの１台または数台が障害で停止してもクラスタシステムが停止したクラスタノードをクラスタサーバから外し、クラスタサーバを再構築する。その為例え個々のクラスタノードの信頼性が低くとも、クラスタ

*会津大学大学院理工学研究科情報システム学専攻
Graduate Department of Information Systems, Graduate School of Computer Science and Engineering, The University of Aizu
†会津大学大学院理工学研究科
Graduate School of Computer Science and Engineering, The University of Aizu

サーバ全体としての信頼性は非常に高いものとなる。またクラスタサーバ全体としての停止を伴わずにその一部を停止する事が可能という特徴はシステムの保守作業等に非常に有効である。

既に様々な手法を用いた多くのクラスタシステムが提案・開発されている。本論文では第２章でこれら既存のクラスタシステムをその手法により分類する。第３章では我々の提案であるマルチキャストを利用したクラスタシステムを説明する。第４章ではそのプロトタイプの実装と性能計測について解説する。第５章で本論文をまとめ、第６章で今後の課題について述べる。

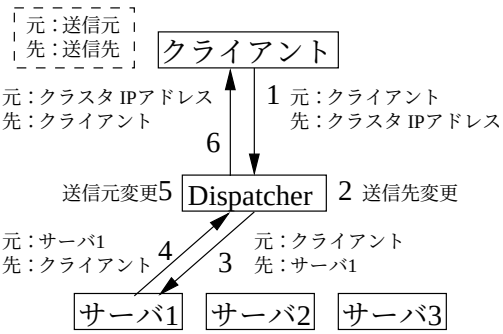

図 1: 全送信集中制御方式

2　関連研究

クラスタシステムで最も有名なものは Round Robin DNS (RR-DNS) であろう。RR-DNS はクラスタシステムの最も初期の試みの一つであり、現在でも最も一般的な方式である。RR-DNS の DNS サーバ上では１つのクラスタサーバの名前に対し複数の IP アドレスが登録されている。この DNS サーバはクライアントからのクラスタサーバの名前解決要求を受け取った際に、これら複数の IP アドレスの中の１つをラウンドロビン方式で選び、クライアントに返す。これにより各クライアントはそれぞれ異なる IP アドレスを得て、それぞれ異なるサーバにアクセスするためクライアントリクエストの処理の負荷が複数のサーバ上に分散される。

RR-DNS は非常に単純で強力な方式であるが、負荷分散が均一にならず片寄ってしまいやすいという欠点があった。これは主に名前解決要求の結果が他の DNS サーバやクライアント自身にキャッシュされてしまう為である [1] [2]。また、キャッシュされてしまうことにより、クラスタノードが障害で停止した場合に対処できない問題もあった。

その後均一な負荷分散、障害耐性を実現すべく今日までに様々な方式が提案・実装された。以下ではそれらをそれぞれ異なった特徴をもつ４種の方式に分類する。

2.1　全送信集中制御方式

Magic Router [3]、Local Director [4]、Virtual Server via NAT [5] がこの方式を使用している。全送信集中制御とはデータの流れ、クライアントからサーバへ、サーバからクライアントへ、その両方が一点で集中制御されているということである（図 1)。クラスタシステムをインストールされたルータがこの集中制御を行う。以後このルータを Dispatcher と呼ぶ。

この方式ではクラスタ IP アドレスと呼ばれる１つの IP アドレスが使われる。クラスタサーバはネットワーク上で仮想的に一台の Web サーバと見なされ、クラスタ IP アドレスはこの仮想の Web サーバに対して与えられる。クライアントはクラスタサーバにアクセスする際に、このクラスタ IP アドレスに向けてリクエストを送る。そしてクラスタ IP アドレスに送られたリクエストは Dispatcher に到達するようネットワークが設定されている。Dispatcher はリクエストを受け取るとクラスタノードの中から１台を選び、そのリクエストの宛先アドレスを選んだクラスタノード固有の IP アドレスに書き換える。この書き換えにより負荷分散が行われる。書き換え後、そのリクエストはクラスタノードに送られる。クラスタノードはリクエストを処理してクライアントに返答する。このクラスタノードからの返答も Dispatcher を通過するようネットワークが設定されている。Dispatcher はクラスタノードからの返答を受け取るとその送信元アドレスをクラスタ IP アドレスに書き換え、クライアントに送る。これによりクライアント側から見れば、クラスタ IP アドレスにリクエストを送り、クラスタ IP アドレスより返答を得る事になる。

この方式では Dispatcher 以外は特殊なハードウェア/ソフトウェアを用意する必要はない。クラスタノードは通常の Web サーバがそのまま利用できる。しかし全ての負荷分散制御を Dispatcher 上で行う為に Dispatcher がボトルネックになりやすい。また Dispatcher に障害が発生した場合クラスタ Web サーバ全体が使用不能になる。以後このようにシステムの一部分にも関わらずその部分の障害がシステム全体の停止を引き起こすようなシステム上の弱点を critical part と呼ぶ。

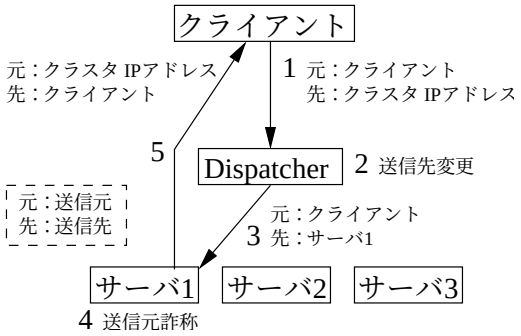

図 2: 半送信集中制御方式

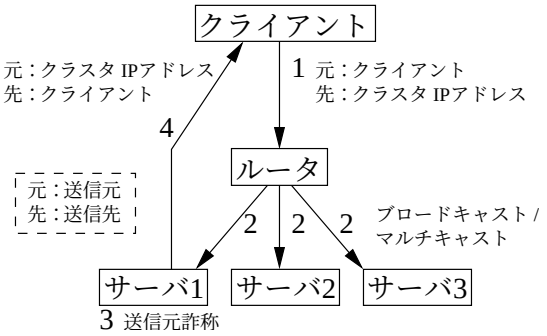

図 3: リクエスト拡散方式

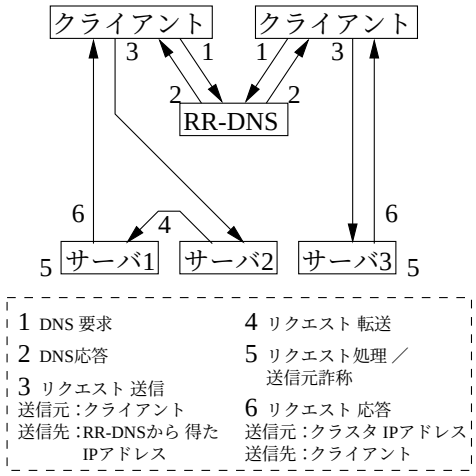

図 4: ２段階分散方式

2.2 半送信集中制御方式

TCP Router [2]、ONE-IP (routing based) [6] 、Virtual Server via TUN/DR [5] がこの方式を取っている。

この方式は全送信集中制御方式を改善した物と見ることができる（図 2)。全送信集中制御方式との主な違いはクラスタノードからクライアントへの返信は Dispatcher による制御を必要としない点である。返信はクラスタノード自身により送信元アドレスをクラスタ IP アドレスに設定された後クライアントへ向け送信される。通常は Web サーバからクライアントへのデータ転送量の方がその逆に比べ格段に多い為、この部分の処理をクラスタノードで行うことで Dispatcher の負荷を相当下げることが出来る。よってこの方式では Dispatcher はボトルネックになりにくいが、依然として critical part である。また、上述したように各クラスタノードはクライアントへの応答に際し送信元アドレスに自身の固有 IP アドレスでなくクラスタ IP アドレスを設定しなければならず、その為の何らかの機構／設定が各クラスタノードに必要である。

2.3 リクエスト拡散方式

ONE-IP (broadcast based dispatching) [6]、Convoy Cluster [7] 等がこの方式を使用している。

この方式はクライアントからクラスタサーバへの全てのリクエストがマルチキャスト／ブロードキャストにより全てのクラスタノードに伝えられるのが特徴である（図 3)。この方式でもクラスタ IP アドレスが使われ、クライアントはこのアドレスにリクエストを送る。リクエストはクラスタシステムによりマルチキャスト／ブロードキャストされ全てのクラスタノードで受信されるが、そのうちの１台のクラスタノードが選ばれてその処理にあたる。選ばれたクラスタノードはリクエストを処理し、結果をクライアントに返す。その際、送信元にはクラスタ IP アドレスが指定される。

この方式ではボトルネックや critical part になる部分はない。しかし、全てのリクエストパケットをマルチキャスト／ブロードキャストすることによりネットワーク帯域が圧迫され、各クラスタノードはそれらのパケット全てを受信しなければならない為、その処理により性能が低下する。また、リクエストの拡散にマルチキャストやブロードキャストを利用しているためにクラスタノードのネットワーク上での設置位置がある程度限定される。

2.4 ２段階分散方式

Distributed Packet Rewriting [8]、SWEB [9] 等がこの方式を取っている。

この方式では一旦 RR-DNS によりクライアントからのリクエストを各クラスタノードに分散する。（図 4）その結果各クラスタノードの負荷に偏りが生じた場合はクラスタシステムがリクエストをクラスタノード間で転送し、負荷が均一になるよう調整する。どのサーバで処理されたにせよ、クライアントに返される

リクエストの結果には送信元としてクラスタ IP アドレスが指定される。

この方式ではボトルネックや critical part になる部分は無い。しかしリクエストの転送は高価な処理であり、転送処理によりサーバの性能は低下し、応答速度は遅くなる。また RR-DNS による欠点も受け継いでいる。

2.5 考察

本章では既存のクラスタシステムの基本的な手法を解説した。これら以外にもクライアント側で負荷分散をするものや RR-DNS を改善した物などがある。しかし負荷分散をクライアント側で行う方式は一般にクライアントに好まれず、クライアントの環境に依存するので難しい。RR-DNS を改善した手法では名前解決の結果がキャッシュされる問題を解決することが困難である。また、この方式では負荷バランス調整の対象となるのが今後アクセスしてくるクライアントのみで、調整は長期的な緩やかなものになってしまう。よってこれらは本章では取り上げなかった。

本章で解説した手法の内幾つかには設計上の問題点によるボトルネックや critical part が存在した。こうした欠点はその部分の処理能力を増やしたりバックアップを用意することにより改善する事は出来るが根本的な解決にはなりにくい。クラスタサーバでは性能は新たにノードを追加することによって比較的簡単に向上させることが出来るため、基本設計における信頼性の確保は性能の向上よりも重要と我々は考える。次章では我々の考える信頼性の高いクラスタシステムを解説する。

3 マルチキャストを利用した分散管理型クラスタ Web システム

本章では信頼性の向上に重点をおいたクラスタシステムを提案する。我々はこのシステムを Hive システムと名付けた。また Hive システムによるクラスタサーバを Hive サーバ、Hive サーバに属するクラスタノードを Hive ノードと呼ぶ。

Hive システムでは全体のリクエスト処理性能よりも、ボトルネックや critical part を排除し信頼性の高い分散管理型のクラスタシステムを構築することを重視する。その為にシステム全体を統括する部分を作らず、システムの各部分が自立して自身を管理し他の部分と協調することにより、全体としてクラスタサーバとして動作するように設計する。

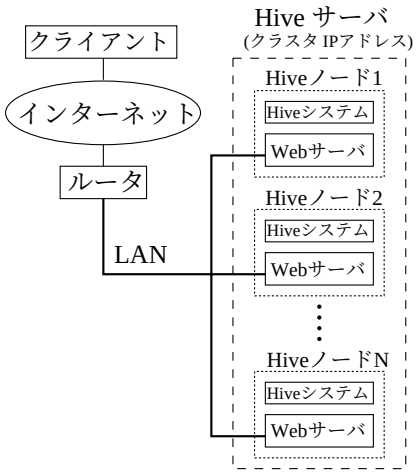

図 5: システム概要

3.1 システム概要

システム構成は単純で、複数の Web サーバとその全ての Web サーバ上で実行される Hive システムのプログラムから成る（図 5）。Hive システムによりそれぞれの Web サーバは Hive ノードとなり、複数の Hive ノードから Hive サーバが構築される。全ての Web サーバは同一のネットワークセグメントに接続されていなければならない。また Hive サーバはクラスタ IP アドレスを持つ。クライアントはこのクラスタ IP アドレスに宛ててリクエストを送信し、クラスタ IP アドレスより返信を得る。

Hive システムはリクエスト分散システムと負荷バランス調整システムの 2 つの部分より成る。リクエスト分散システムによりリクエストは複数の Hive ノード上に分散され、そして処理される。各 Hive ノードの負荷に片寄りが生じた場合、負荷バランス調整システムによりそれを検知し調整する。

3.2 リクエスト分散システム

Hive サーバではリクエストはマルチキャストにより全 Hive ノードによって受信される。その後に全 Hive ノードは各リクエストに対し割り当て確認を行い、その結果一台の Hive ノードだけがそのリクエストに反応しクライアントに応答を返す。その他の Hive ノードはその要求は無視する（図 6）。

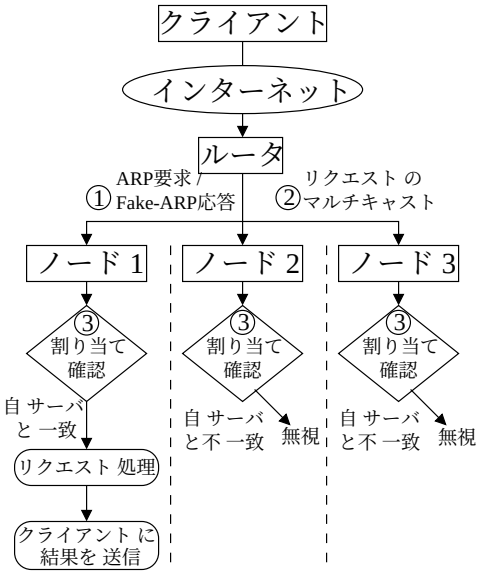

図 6: リクエスト分散システム

3.2.1 リクエストのマルチキャスト

Hive サーバに最も近いルータはリクエストを受け取ると、クラスタＩＰアドレスを持つマシンの MAC アドレス（ハードウェアアドレス）を得るために、適切なネットワークセグメントに対し ARP 要求をブロードキャストする。全 Hive ノードはこのセグメント上に設置されているため、全 Hive ノードがこの ARP 要求を受け取る。この ARP 要求に対し、オペレーティングシステム内の正規の ARP システムからではなく、Hive システム内の Fake-ARP 機能により ARP 応答が返される。但し、この ARP 応答によりルータに返されるのは通常のユニキャスト MAC アドレスではなく、ある一つのマルチキャスト MAC アドレスである。このマルチキャスト MAC アドレスは一定時間ルータ上にキャッシュされ、その間は新たに APR 要求を発生させること無く繰り返し参照される。ルータは Fake-ARP 応答によって得たマルチキャスト MAC アドレスにリクエストを送る。つまり、リクエストを含む IP パケットをマルチキャストする事になる。各 Hive ノードはこのマルチキャスト MAC アドレスへのパケットを受信するように設定されているので、全 Hive ノードがこのパケットを受信することになる。

3.2.2 割り当て確認

各 Hive ノードはルータによりマルチキャストされたクラスタ IP アドレス宛てのパケットを受信すると直ちにそのパケットに対し割り当て確認を行う（図 7）。

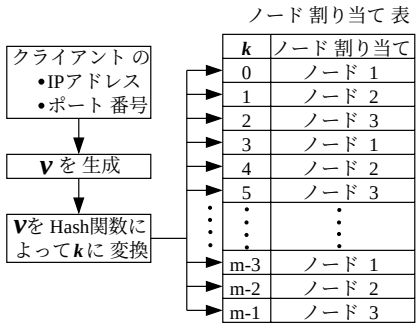

図 7: 割り当て確認

まず各 Hive ノードは該当パケットを受信するとパケットからクライアントの IP アドレスとポート番号を読み出す。例えば、IP アドレス＝ 100.200.100.200、ポート番号＝ 5000 とする。

この２つからあるアルゴリズムを用いて新しい値 v を作り出す。どのようなアルゴリズムでも構わないがなるべく Hive ノードに負担を掛けず、かつ v がある程度大きな値になるようなものにする。例として、ここでは単純に IP アドレスとポート番号を足す事にする。つまり、$v = 100200100200 + 5000 = 100200105200$ となる。

次にあるハッシュ関数により v を 0 〜 m の値 k に変換する。このハッシュ関数もなるべく Hive ノードに負担を掛けない物を選ぶ。ここで m は負荷分散の粒度を決める値で Hive ノードの予定最大数の数倍から数十倍の値を指定する。m が大きいほど負荷分散の細かい調整が可能になる。ここでは Hive ノードが最大で１０台を予定し、最大数の１０倍の１００を m の値として指定することにする。また、ハッシュ関数としては mod 関数を使うことにする。つまり $k = v \bmod m = 100200105200 \bmod 100 = 0$ である。

次に k とノード割り当て表を用いてどの Hive ノードがこのリクエストを処理するか決定する。ノード割り当て表は負荷バランス調整システムにより管理され、リクエスト分散システムはただ読み出すだけである。それぞれの k の値に対し１台の Hive ノードが割り当てられている。先ほど得た k の値に対応する Hive ノードを調べ、それがもしその Hive ノード自身であればこのリクエストを処理する。そうでなければこのリクエストを無視する。例えば今 Hive ノード１が割り当て確認しているとする。先ほど得た k ＝ 0 を使いノード割り当て表（図 7 参照）を見ると k ＝ 0 には Hive ノード１自身が割り当てられている。つまり Hive ノード１自身でこのリクエストを処理しなければならない。

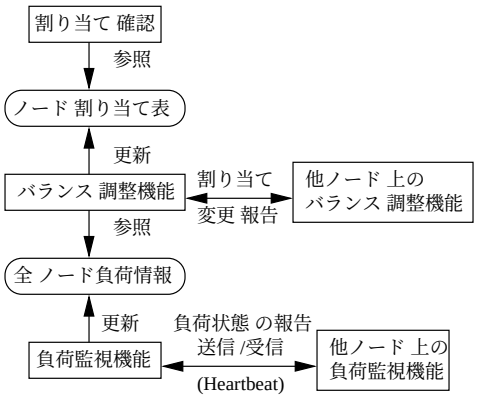

図 8: 各機能の依存関係

3.2.3 サーバからの応答

割り当て確認の結果そのリクエストが自身に割り当てられていた場合、Hive ノードはそのリクエストを処理してクライアントに結果を返す。その際、送信元 IP アドレスとしてクラスタ IP アドレスを指定しなければならない。これによりクライアントはクラスタ IP アドレスに対しリクエストを送り、クラスタ IP アドレスから応答を得る事になる。

3.3 負荷バランス調整システム

Hive サーバから最大の性能を引き出すには全 Hive ノードの負荷を常に均等な状態に保つ必要がある。Hive システムでは負荷バランス調整システムがこの処理を行う。負荷バランス調整システムは全 Hive ノード上で常に実行されており、負荷監視機能とバランス調整機能の 2 つの機能から構成される。負荷監視機能は他の Hive ノードから負荷状況報告を定期的に受けとり、それをまとめてバランス調整機能に提供する。また自 Hive ノードの負荷状況を定期的に他全 Hive ノードに報告する。バランス調整機能は負荷監視機能からの報告を元に Hive サーバ内の負荷バランスを監視し、負荷が片寄った、ある Hive ノードに障害が起きた、新しい Hive ノードが追加された、等の問題が起きたときノード割り当て表を変更することにより問題に対処する。この 2 つの機能を含む各機能の依存関係を図 8 に示す。

3.3.1 負荷監視機能

各 Hive サーバ上の負荷監視機能は一定時間（1 秒〜数秒）毎に自 Hive ノードの負荷を調べ、結果を全 Hive ノードへマルチキャストする。つまり全ての Hive ノードは一定時間毎に他全 Hive ノードからの負荷の報告を受け取る。負荷監視機能はこの報告をまとめ、バランス調整機能に提供する。また、この負荷報告は Hive ノードの障害検出も兼ねている。これは Heartbeat [10] と呼ばれる。ある Hive ノードからある一定回数（5 回等）連続して負荷報告が無かった場合、全 Hive ノードはその Hive ノードが障害で停止したと見なし、バランス調整機能に報告する。同様に新規 Hive ノードが Hive サーバに追加された時、または障害で停止していた Hive ノードが回復した時も負荷監視機能はそれを検出し、バランス調整機能に報告する。

3.3.2 バランス調整機能

バランス調整機能は負荷監視機能より常に Hive サーバ内の状態に関する情報を得ている。バランス調整機能は常にそれを監視し、Hive サーバ内の負荷バランスが片寄った場合、ある Hive ノードに障害が起きた場合、新規 Hive ノード（または障害による停止から回復した Hive ノード）が Hive サーバに追加された場合にそれらに対処するべく処理を開始する。

負荷バランスが片寄っている場合はバランス調整機能は負荷の大きい Hive ノードから負荷の小さい Hive ノードへリクエスト処理の負担を移動させる。具体的にはノード割り当て表を書き換える事により行う。1 台の Hive ノード（例えばその時点で一番負荷が低い Hive ノード）上のバランス調整機能がノード割り当て表で負荷が大きい Hive ノードに割り当てられている k を探し、その k の内幾つかを負荷が小さい Hive ノードに再割り当てする。同時にこのノード割り当て表の変更内容を全 Hive ノードにマルチキャストする。全 Hive ノード上のバランス調整機能はそれに基づき自 Hive ノード上のノード割り当て表を変更する。これにより全 Hive ノード上のノード割り当て表は常に同じ内容を保持する。

ある Hive ノードに障害が起きた場合、全 Hive ノード上のバランス調整機能はノード割り当て表から障害の起きた Hive ノードに割り当てられていた全ての k を検索し、共通のアルゴリズムによりそれらの新しい割り当てノードを調べる。それが自ノードであった場合割り当て表の変更を行い、変更内容を全 Hive ノードにマルチキャストする。

新規 Hive ノードの追加、または障害から回復した Hive ノードがあった場合、その Hive ノードは単に負荷が非常に低い Hive ノードとして扱われる。つまり

負荷バランスが片寄る事になる。よって通常の負荷バランスが片寄った場合と同じ処理が行われる。

以上のようにバランス調整機能はノード割り当て表を書き換えることにより Hive サーバ内の負荷バランスを調整する。ノード割り当て表の書き換え内容はマルチキャストにより全 Hive ノードに連絡され、全ノードがほぼ同時に同じ書き換え作業を行う。よって全 Hive サーバは常に同じ内容のノード割り当て表を保持する。

3.4 Hive サーバの特徴

Hive サーバは本論文第 2 章の分類ではリクエスト拡散方式に分類されるシステムである。同方式に分類される他システムと比較した本システムの特徴を以下に挙げる。

- Hive サーバでは同じクライアントからの違うポート番号を用いた同時アクセスは別々に扱われる。よってクライアントが例えば 3 個のポートを同時に使用して Hive サーバにアクセスした場合最大で 3 台の Hive ノードにより同時にリクエストが処理される。これにより Hive サーバからの素早い応答が期待できる。

- 各 Hive ノードの性能に違いがあっても、サーバ割り当て表上での割当数が性能に応じて調整されて適切な負荷が与えられる。負荷が片寄ってしまった場合でも同様にサーバ割り当て表を変更することにより片寄りを修正することができる。

- ある Hive ノードがダウンした場合や逆に新しく追加される場合でも現在接続中のクライアントとのコネクションは一部を除いて影響を受けない。影響を受けるのは、Hive ノードのダウンの場合はダウンした Hive ノードと接続していた全コネクションのみである。Hive ノードが追加される場合は、追加後の負荷調整の為に割り当てノードが変更される k を通じて（割り当て確認の際その k を得て）アクセスしていたコネクションのみである。

- Hive ノードはマルチキャスト MAC アドレスを利用しているので複数のマルチキャスト MAC アドレスを利用した今後の機能拡張が可能である。

ファイルのサイズ	割合
500 B	35%
5 KB	50%
50 KB	14%
500 KB	0.9%
5 MB	0.1%

表 1: 各ファイルのダウンロードされる割合

4 プロトタイプの実装と性能計測

4.1 プロトタイプの実装

Hive システムのプロトタイプを Linux 2.2.14 + Apache 1.3.19 の環境で実装した。リクエスト分散システムは Network Interface Card (NIC) ドライバ上に実装した。NIC は 3Com 905B を選択し、3Com 905B 用ドライバプログラムである 3c59x.c 上にリクエスト分散システム用のコードを書き足した。追加されたコードにより全割り当て確認処理、Fake-ARP 処理が NIC の割り込み処理の Top half で行われる。負荷バランス調整システムはユーザ空間プログラムとして実装され、リクエスト分散システムに指示を送り、マルチキャストメッセージを通じて他ノード上の Hive システムと負荷情報／割り当て変更報告をやり取りする。

負荷監視機能の 1 回の負荷報告（heartbeat）のデータ量は 100 bytes 程であり、100 台の Hive ノードによる Hive サーバで毎秒負荷報告を行ったとしても毎秒の総データ量が 10,000 bytes (80Kbits) である。この場合使用するネットワーク帯域は、100Mbps イーサネットの場合で全体の 0.1 % 以下となり十分に小さい。

4.2 性能計測環境

ベンチマークツール WebStone2.5 を用いてプロトタイプシステムの性能拡張性についての計測を以下の環境で行った。

Web サーバ用として 366MHz CPU, 256 MB memory, 3COM 3c905B NIC のマシンを 4 台、クライアント用として SUN SPARC, 440 MHz CPU, 256 MB memory のマシンを 3 台使用した。全てのクライアント／Web サーバマシンは 1 台の 100Mbps イーサネットスイッチングハブ（NETGEAR FS509）に接続されている。Web サーバ用のマシンには Linux 2.2.14 + Apache 1.3.19 を標準設定でインストールした。但

し Apache のアクセスログは記録しないように設定した。Web サーバ用マシンの NIC ドライバは Hive サーバとして計測の際には Hive サーバ用のドライバに取り替えられる。クライアントマシンには WebStone を標準設定でインストールした。

WebStone の設定は標準のものをそのまま使用した。filelist には filelist.standard を使用し、ダウンロードされるのは全て静的なファイルである。filelist.standard による各ファイルのダウンロード率を表 1 に示す。計測に使用した WebStone のダウンロード用ファイルは総計で 6MB 以下であり、Web サーバマシンのメモリに全て読み込まれる量である。よって計測中 HDD からの読み出しはほぼ無かった。

WebStone により実クライアントマシン 3 台上に仮想クライアントを最大で計１５クライアント構築し計測を行った。今回は実験環境の都合上１５クライアント以上での計測はできなかった。仮想クライアントは３台の実クライアントマシン上になるべく同じ数になるように自動的に振り分けられる。よって仮想クライアントの数が 1 クライアントの時は実クライアントマシンが 1 台だけ、2 クライアントの時は 2 台だけが使用される。計測は Hive ノード数、仮想クライアント数を変えた各パターンで 5 分間の計測を 4 回行い、その平均を結果として記録した。

負荷監視機能による負荷報告は 1 秒 1 回とし、5 回連続して負荷報告が無かった場合に障害が起こったと見なされるよう設定した。サーバ割り当て表の大きさ（k の数）は１００とした。

4.3 結果の予測

Hive サーバではクライアントからの全てのリクエストパケットに対し全てのサーバが割り当て確認を行っており、この割り当て確認により Hive サーバの性能が低下する。つまりクラスタ化の為のオーバーヘッドによる性能低下である。Hive サーバのノード数を増やして Hive サーバの性能を向上させ、Hive サーバ全体の受信リクエストパケットの数が増加した場合には、このオーバーヘッドもそれに比例して増加する。増加したオーバーヘッドは複数の Hive ノードに分散されるのではなく、各 Hive ノードのそれぞれの性能を直接低下させる。

ここで、オーバーヘッドによる性能の低下率を x とし、通常の Web サーバとしてのサーバのリクエスト処理性能を T、ノードが N 台あるときの各 Hive ノードのリクエスト処理性能を T'、とすると以下の式が

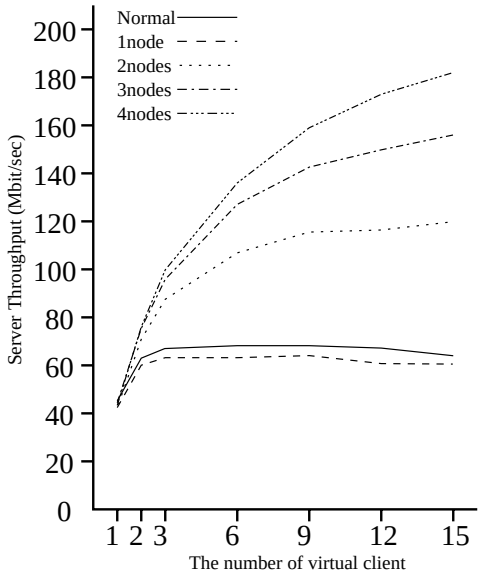

図 9: サーバスループット

得られる。

$$T' = T - NT'x$$

よって

$$T' = \frac{T}{1 + Nx}$$

Hive サーバ全体の性能 S はノードを N 台分集めたものであるから、

$$\begin{aligned} S &= NT' \\ &= \frac{NT}{1 + xN} \end{aligned} \tag{1}$$

となる。

4.4 性能計測結果

性能計測の結果を図 9、図 10、図 11 に示す。図 9 は Hive サーバからのスループット、図 10 はリクエストをサーバに送り要求ファイルを受信するまでの時間（レスポンスタイム）であり WebStone による計測結果である。図 11 は計測実行中のサーバマシンの CPU 使用率を top コマンドにより常に計測し、そのおおよその平均値を記録したものである。図中 *Normal* とは Hive ノードではない通常の Web サーバとしての実験結果、*1 node*〜*4 nodes* はそれぞれのノード数での Hive サーバによる実験結果である。

図 9 中、*Normal* に比べ *1 node* が常に少し低くなっている。この低下率が性能結果予測で述べた x である（プロトタイプシステムでは Hive ノードが例え 1

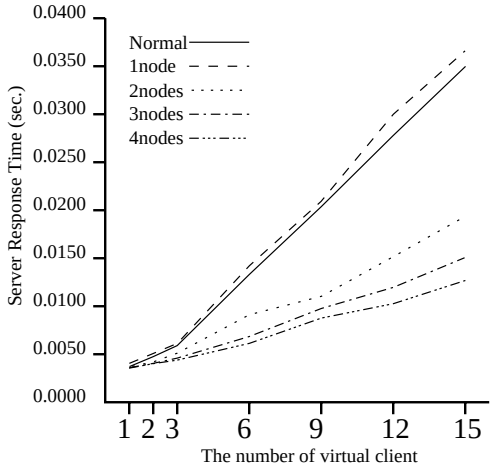

図 10: レスポンスタイム

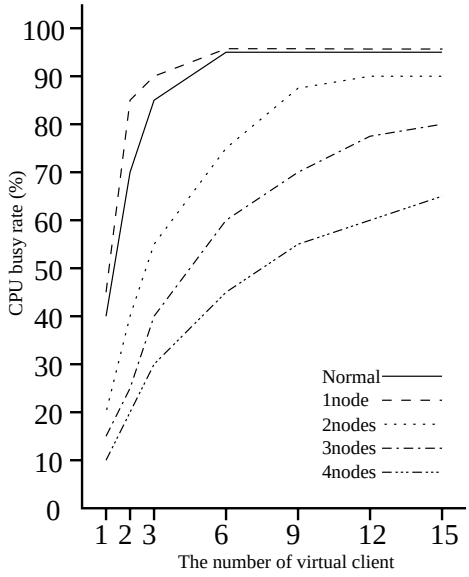

図 11: CPU 使用率

台でも割り当て確認を行う)。*Normal*、*1 node* ともに仮想クライアント数が 3 を超えた辺りから結果が横ばいになり、性能限界に達したと見ることができる。図 11 の CPU 使用率グラフからもそれが分かる。*Normal*、*1 node* のスループット最大計測値はそれぞれ約 68Mbit/sec., 65Mbit/sec. である。よって、式 (1) において $T = 68$、$x = 0.055$ とすると、予想される *2 nodes*、 *3 nodes*、*4 nodes* の性能はそれぞれ約 123 Mbit/sec.、 175 Mbit/sec.、223 Mbit/sec. である。

図 9 中の *2 nodes* の結果を見ると仮想クライアント数が 9 を超えた辺りで結果が横ばいになっており、予想値（123Mbit/sec）に近い値が得られている。CPU 使用率グラフからも性能限界に近い事が分かる。*3 nodes* の場合は予想値 (175Mbit/sec) よりも低い値が得られた。また、*4 nodes* の場合は、予想値（223Mbit/sec）よりかなり低い値となった。*3 nodes*、*4 nodes*、共にスループットグラフ上で横ばいには達していないものの性能の上昇率を見る限り予想値には達しないであろう。しかし一方で図 11 の CPU 使用率を見ると共にまだ余裕を残している。各 Hive ノード上の CPU に割り当て処理の負担を掛けることによりクラスタを維持する Hive サーバにおいて、CPU 使用率に余裕があるという事はサーバはまだ余力を残していると言える。つまり何らかの原因でサーバに十分なリクエストを与える事が出来ていないと考える。しかし今回の実験ではその原因を特定は出来なかった。正確な計測には 1Gbit/sec 等の高帯域なネットワークと、より多くの実クライアントマシンが必要であると思われる。

また信頼性については、性能計測中にサーバマシンがダウンすることが何度もあったが、予定通り負荷分散システムにより負荷が再分配され Hive サーバ全体としては継続してサービスの提供を続けており、図らずもその高い信頼性を（不本意な形で）確認できた。

5 まとめ

本論文では、まず既存のクラスタ Web サーバをその基本的手法により分類した。その際各手法においてボトルネックやシステム上の弱点になりやすい部分に注意を払った。クラスタ Web サーバは Web サーバを追加することにより簡単に性能を向上させることができる。よってクラスタ Web システムの基本設計においてはボトルネックやシステム上の弱点を除去し、信頼性を確保する事が重要だと考えた。

そこで信頼性の高い分散管理型のクラスタ Web サーバの提案として Hive サーバを構築した。Hive サーバは複数の Web サーバとその Web サーバ上で実行される Hive システムのみで構成される。Hive サーバに属する各 Hive ノードはそれぞれ自律し、他 Hive ノードと情報を交換する事により共同で Hive サーバを管理していく。そのため Hive サーバは全体を統括する部分が無く、どの Hive ノードに障害が起きようとも Hive サーバ全体に影響を与えることはない。一台でも Hive ノードが稼働している限り Hive サーバとして機能しクライアントにサービスを提供する。

性能拡張性の計測実験の結果は Hive ノード数が 2 の場合は予想されるスループット性能を達成できたが、３、４の場合は予想より低い値が計測された。しかし計測中のＣＰＵ使用率の結果からは、Hive ノード数

が３、４の場合 Hive サーバはまだかなりの性能の余裕を残しているように読み取れた。何らかの原因で今回の実験では Hive サーバに十分なリクエストを与えることが出来なかったと思われる。正確な計測は今後の課題とする。

また、Hive サーバの信頼性については計測実験を通じて非常に高い物であることが確認できた。

6　今後の課題

現在 Hive システムでは数台の Web サーバ上に１つのクラスタ IP アドレスによる１つのクラスタ Web サーバを構築しているが、これを複数のクラスタ IP アドレスによる複数の Hive サーバに拡張することを予定している。つまり M 個の実 Web サーバ上に構築された Hive サーバ内に N 個の仮想 Hive サーバを構築する。これによる利点は、例えばそれぞれが２台の Hive ノードから成る２つの Hive サーバよりも、４台の Hive ノードから成る Hive サーバ上で２つの仮想 Hive サーバを構築した方が高速な応答が期待でき、信頼性も高まる点である。

また、Hive システムに RR-DNS を組み入れる事も考えている。Hive システムでは設計上全ての Hive ノードが全てのリクエストパケットを受信しなければならない。この制約により、大規模な Hive サーバを構築しクライアントからの多数のリクエストを処理する場合にリクエストの割り当て確認処理により Hive サーバ全体の性能が本来の性能より低下する。この問題の解決法として、まず RR-DNS を用意し同一セグメント上の複数の IP アドレスを同じ名前に対して登録する。Hive ノードの NIC はハードウェア的に受信するマルチキャストパケットを選別出来るものを使用する。上記の仮想 Hive サーバを Hive サーバ内に複数構築し、それぞれの仮想 Hive サーバに RR-DNS に登録した IP アドレスを与え、RR-DNS によりそれらに対し負荷分散を行う。各 Hive ノードはどれか１つの仮想 Hive サーバに所属するようにし、複数に跨って所属しないようにする。よって各 Hive ノードは所属する仮想 Hive サーバに割り当てられた IP アドレス宛てのパケットのみを受信すればよいことになる。これにより上記仮想 Hive サーバを２つ設置すれば各 Hive ノードが受信するリクエストパケットの数は１／２、４つなら１／４になり、各 Hive ノードの割り当て処理の負担を減らす事が出来る。

参考文献

[1] V. Cardellini, M. Colajanni, P. S. Yu, Dynamic load balancing on Web-server systems, *IEEE Internet computing,* Vol. 3, No. 3, pp. 28-39, May-June,1999.

[2] D. M. Dias, W. Kish, R. Mukherjee, R. Tewari, A Scalable and Highly Available Web Server, *Proc. 41st IEEE Computer Soc. Int'l Conf., COMPCON '96,* Feb. 1996, pp. 85-92.

[3] E. Anderson, D. Patterson, E. Brewer, The Magicrouter, an Application of Fast Packet Interposing, May 1996, http://www.cs.berkeley.edu/~eanders/projects/magicrouter/.

[4] Cisco Local Director, Cisco Systems, Inc., 1997, http://www.cisco.com/warp/public/cc/pd/cxsr/400/tech/index.shtml.

[5] W. Zhang and et al. Linux virtual server project. 1998, http://www.linuxVirtualServer.org/.

[6] O. P. Damani, P. E. Chung, Y. Huang, C. Kintala, Y. Wang, ONE-IP: Techniques for Hosting a Service on a Cluster of Machines, *J. Computer Networks and ISDN systems,* Vol. 29, Sept. 1997, pp. 1,019-1,027.

[7] Valence™ Research, Inc., Convoy Cluster™ Software, 1998.

[8] A. Bestavros, M. Crovella, J. Liu, D. Martin, Distributed Packet Rewriting and its Application to Scalable Web Server Architectures, *Proc. of ICNP'98: The 6th IEEE International Conference on Network Protocols,* (Austin, TX), October 1998.

[9] D. Andresen, T. Yang, and O. H. Ibarra, Toward a Scalable Distributed WWW Server on Workstation Clusters, *The Journal of Parallel and Distributed Computing* vol. 42, pp. 91-100, September, 1997.

[10] A. Robertson, Linux-HA Heartbeat System Design, http://linux-ha.org/comm/.