C3–4

自律ディスクを用いたWebサーバーの構成

花井知広
hanai@de.cs.titech.ac.jp
東京工業大学 工学部 情報工学科

横田治夫
yokota@cs.titech.ac.jp
東京工業大学 学術国際情報センター

1 はじめに

今日、Webの急速な普及によって、Webサーバーにはますます高いスループットとアベイラビリティが要求されるようになってきている。同時に、コンテンツ容量の増加に伴いストレージ管理にも工夫が必要となっている。

我々は、ストレージシステムにおける負荷分散、故障対策、障害回復などの高度な機能を実現するためのアプローチとして、ディスク装置内の制御用プロセッサとキャッシュ用のメモリを利用する自律ディスクを提案してきた。自律ディスクは、ネットワークに複数接続されクラスタを構成する。

本稿では、その一つのアプリケーションとして、自律ディスクを用いてクラスタ化したWebサーバーを構成する手法を検討する。負荷分散・耐故障処理をクラスタ内で自律的に行う、高性能・高信頼なWebサーバーの実現を目指す。

2 自律ディスクの概要

2.1 自律ディスクの特徴

ここではまず自律ディスクについて説明する。図1に標準的な自律ディスクのクラスタの例を示す。自律ディスクはネットワーク環境でクラスタを構成することを前提としている。ホストはデータにアクセスするためにクラスタ内の任意のディスクに要求を発する。クラスタ内のディスクはディスク間の局所的な通信を行うことで、協力してホストからの要求に対処する。このような前提のもとで、自律ディスクは以下のような性質を持つ[1]。

データ分散 データは分割され、クラスタ内の複数のディスクに分散される。

ホストからの均質的なアクセス クラスタ内のいかなるディスクに格納されているデータに対するアクセスの要求も、クラスタ内の全てのディスクで受け付けられる。このために各ディスクはデータ配置に関する情報を含んだ分散ディレクトリ構造を持っている。

同時実行制御 データが複数のホストから同時にアクセスされた際の同時実行制御は対象データが格納されている場所で行われる。

偏り制御 クラスタ内のディスク間でデータ分散の偏りや負荷の偏りが生じた場合は、その偏りは検出され、偏り制御プロセスが起動される。

耐故障性 クラスタ内のディスクの故障やディスクコントローラのソフトウエアバグが発生した場合、それらの障害は自動的にマスクされる。そのために、クラスタ内にプライマリデータとバックアップデータを持ち、バックアップはログを介して非

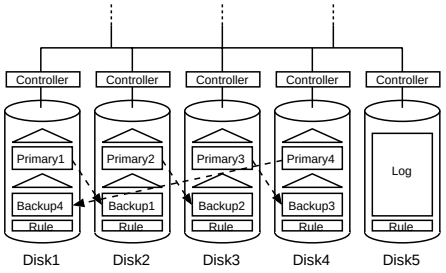

図1: 自律ディスク

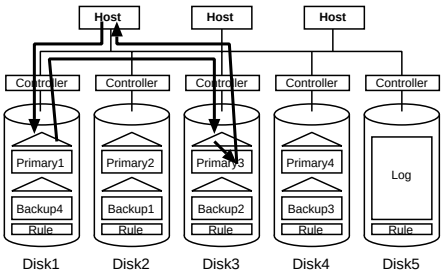

図 2: 自律ディスクからの読み出し

同期に行われる。

異種性 上記の性質の実現方法や自律ディスク自体の情報を外部から隠蔽するように、自律ディスク–ホスト間のインターフェース及び自律ディスク–自律ディスク間のアクセスはストリームインターフェースを持つ。

これらの性質の主な長所は、ホストからの透過性とシステムのスケーラビリティである。データ分散の仕方、偏り制御の方法、耐故障性、異種性等に関してホストは関与しないため、分散ディスク制御に対するホストのオーバーヘッドをかなり減少させることができる。

2.2 自律ディスクの読み出し時の処理

自律ディスクからデータを読み出す際には以下の処理が行われる。この様子を図 2 に示す。

1. ホストがクラスタ内の任意のディスクに接続し、要求するストリーム ID を含めた Retrieve コマンドを送信する。

2. Retrieve コマンドを受信したディスクは分散ディレクトリをトラバースする。そのディスクには要求されたストリームが無いとわかった場合は、そのストリームを含んでいると思われるディスクにコマンドを転送する。

3. リクエストされたストリームが分割して格納されていた場合は、リクエストされたストリーム ID で示されるストリームは、そのサブストリームの識別子のリストである。その場合はサブストリームに対する新たなリクエストコマンドが発行される。

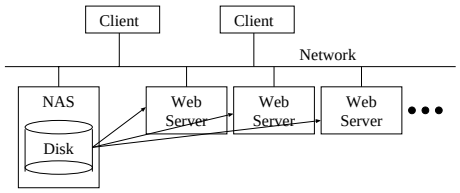

図 3: NAS を用いた Web サーバークラスタ

4. ディスクがリクエストされたストリームを含んでいる場合は、そのディスクがデータを要求しているホストに接続し、ストリームを送信する。

5. ホストがストリームを受信する。ストリームが分割して格納されていた場合は、複数のディスクからデータを受信する。

サイズの大きなストリームは複数のディスクに分割して格納されているので、読み出しを各ディスクで並列に行うことができ、スループットが向上する [2]。また、ディスクに故障が発生し検出された場合には、ホストに透過的にバックアップのディスクからストリームを読み出すことができる [3]。

3 クラスタ化 Web サーバー

Web サーバーに高いスループットとアベイラビリティが要求される場合は、クラスタ化 Web サーバーを構成することが一般的である。この際に一般的にもっとも性能がよい構成方法は、全てのドキュメントを各ノードがそれぞれ持つ方法である。この方法は耐故障性が非常に高いが、必要なディスク容量が非常に大きくなるという欠点がある。また、ディスク容量が不足した場合に全てのノードのディスクを増設しなければならないため、ディスク容量を増加させることは困難である。さらに、コンテンツを各ディスクで同期して更新することも困難である。

これらの問題を解消するために、図 3 のように NAS などを用いてドキュメントを集中管理する方法もある。しかしこの方法ではストレージ部分がボトルネックになる可能性が高く、拡張性にも限界がある。

そこで、ドキュメントを各ノードに分散して配置すれば同期更新とデータの分散配置を両立させることができる。しかし、従来の手法ではドキュメントの追加・

削除・時間的なアクセス偏りの変化などが発生した場合には、ノード間の負荷を均衡化するためにシステム管理者がドキュメントの再配置を行う必要がある。このため、管理コストが大きく、負荷の偏りに対応するまでに時間がかかるといった問題がある。

4 自律ディスクを用いた Web サーバーの構成

2 章で述べた自律ディスクを用いてクラスタ化 Web サーバーを構成することで、以下の利点を得られると考えられる。

- 自律ディスクがドキュメントを各ノードに均等に分散して配置するので、ノード間の負荷の偏りを減少させることができる。また、特定のドキュメントにアクセスが集中した場合にも動的な偏り除去が行われ、負荷が分散される。これにより、管理者はドキュメントの配置を考える必要がなくなり、アクセス偏りの時間的な変化に対して迅速に対応できる。
- ディスクが故障した場合にも、自律ディスクが持つバックアップ機構を利用してサービスを継続することができる。また、ダーティリードを許可する場合はバックアップディスクからも読み出すことができ、さらに負荷を分散することができる。

Web サーバーを構成する上で、以下のような条件を設ける。

- CGI などを用いて動的にドキュメントを生成することは考えず、静的ドキュメントのみを扱う。
- クラスタ内の各ノードが HTTP リクエストを受け、コンテンツをサービスする。
- HTTP リクエストはクラスタ内の各ノードに均等に与えられるものとする。これはラウンドロビン DNS 等の手法を用いることで実現できる。

4.1 構成要素

自律ディスクを用いた Web サーバーは以下の要素から構成される。その構成を図 4 に示す。

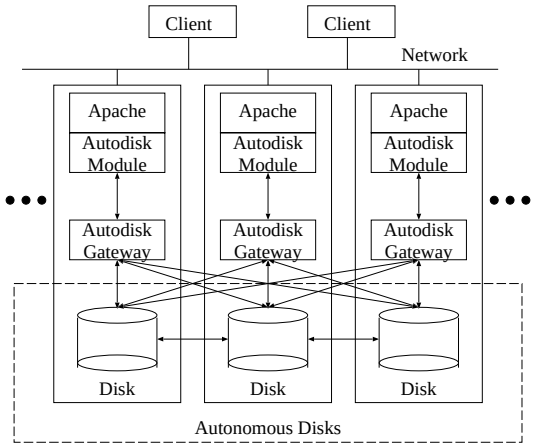

図 4: 自律ディスクを用いた Web サーバー

Apache Web Sever HTTP のハンドリングを行う [4]。Apache は起動時に複数の子プロセスを生成し、それぞれの子プロセスが平行して HTTP リクエストを処理するアーキテクチャを持っている。また、C や Perl を用いてモジュールを作成し追加することで、容易に機能の拡張が行えるようになっている。この Apache に自律ディスクゲートウェイと通信する機能を、C モジュールとして組み込む。以下このモジュールを Apache 自律ディスクモジュールと呼ぶ。設定ファイルで指定したディレクトリ以下へのアクセスを自律ディスクへのドキュメント要求とみなし、処理を行う。

自律ディスクゲートウェイ Apache 自律ディスクモジュールが自律ディスクにアクセスするためのインターフェースである。詳しくは次節で述べる。

自律ディスク ドキュメントを格納し、そのディレクトリ情報を持つ。現在、Java で模擬実装が行われている [5]。

4.2 自律ディスクゲートウェイ

自律ディスクからデータを受信するときには、ホストはサーバーソケットを開いて自律ディスクからの接続を待たなければならない。Apache では複数のプロセスが平行して HTTP リクエストを処理しているが、複数プロセスが同時に同一ポートのサーバーソケットを開くことは出来ないので、サーバーソケットを開くタ

イミングをプロセス間で調整する必要が生じる。また、他のプロセスが受信するべきデータを横取りして受信してしまう可能性もある。加えて、現状の自律ディスクは Java を用いて実装されており、通信にオブジェクトのシリアライズ機構を用いている。しかし、Apache の仕様のために Apache 自律ディスクモジュールを Java を用いて実装することは困難である。

そこで Apache 自律ディスクモジュールと自律ディスクの間に自律ディスクゲートウェイを置き、Apache 自律ディスクモジュールは自律ディスクと直接通信せずに自律ディスクゲートウェイを通して自律ディスクからドキュメントを得ることにする。これによって、自律ディスクからのデータを受信するのは自律ディスクゲートウェイのみであるので、Apache 自律ディスクモジュールがサーバーソケットを開く必要はなくなる。さらに、自律ディスクゲートウェイを Java で実装することにより後者の問題も解決できる。

自律ディスクゲートウェイはその他にも以下のような特徴を持つ。

- ドキュメントが複数のディスクに分割して格納されていた場合は複数回に分けてドキュメントを受信する必要があるので、それらを 1 つにまとめた上で Apache 自律ディスクモジュールに送信する。
- 自律ディスクゲートウェイが、同時に 2 つ以上の Apache 自律ディスクモジュールから同一のドキュメントをリクエストされた場合は、負荷を減らすために自律ディスクに対しては 1 回だけリクエストを行う。
- 自律ディスク内のどのディスクにドキュメントをリクエストするかはルールによって選択することができる。選択肢として以下のような方法が考えられる。
  - 全ノードからランダムに選んで送る。
  - 自ノードのディスクにのみ送る。
  - 自ノード以外のディスクをランダムに選んで送る。
  - 右 (左) 隣のノードに送る。
  - 隣接する n ノードからランダムに選んで送る。

  負荷を分散させるためにはランダムにノードを選ぶ方法がよいが、通信量を減らすためにはその自律ディスクゲートウェイと同じ位置にあるノードにリクエストを出したほうがよいと考えられる。

4.3 リクエスト処理の流れ

Web サーバーに対する HTTP リクエストは以下の流れで処理される。

1. Apache は HTTP リクエストを受けると、URL から自律ディスク内のドキュメントを要求しているかどうかを判定する。要求しているならば Apache 自律ディスクモジュールにリクエストの処理を任せる。
2. Apache 自律ディスクモジュールは URL から自律ディスク内でのストリーム ID を取り出し、自律ディスクゲートウェイにドキュメントをリクエストする。
3. 自律ディスクゲートウェイは Apache 自律ディスクモジュールから自律ディスク内のドキュメントのストリーム ID を受け取り、自律ディスクに Retrieve コマンドを送信する。送信先ディスクは管理者の設定したルールによって選択される。
4. 自律ディスクは Retrieve コマンドを受け、ドキュメントを読み出して自律ディスクゲートウェイに送信する。
5. 自律ディスクからリクエストしたドキュメントを受け取り、Apache 自律ディスクモジュールに送信する。
6. Apache 自律ディスクモジュールは自律ディスクゲートウェイからドキュメントを受信し、HTTP リクエスト元に送信する。その後、自律ディスクゲートウェイへの接続を切断する。

4.4 自律ディスクゲートウェイの構成と実装

図 5 のようなアーキテクチャに基づいて自律ディスクゲートウェイの実装を Java を用いて行った。

各要素は次のような動作をする。

Apache からのリクエスト受信部 Apache 自律ディスクモジュールからの接続を受けてリクエストを解

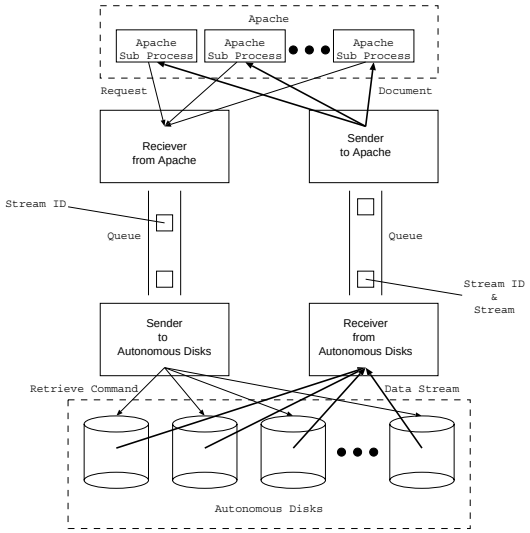

図 5: 自律ディスクゲートウェイのアーキテクチャ

析し、要求するストリーム ID を得る。そのストリーム ID をキューに入れる。ストリーム ID と Apache との接続を組にして、テーブルに加える。

自律ディスクへのリクエスト送信部 キューからストリーム ID を取り出し、自律ディスクに Retrieve コマンドを送信する。送信先ディスクは管理者の設定したルールによって選択される。

自律ディスクからのストリーム受信部 自律ディスクからの接続を待ち、要求したストリームを受信する。ストリームが分割されて到着した場合はそれらを 1 つのストリームに結合する。ストリーム受信後、ストリーム ID とストリーム自身を組にしてキューへ入れる。

Apache へのデータ送信部 キューからストリーム ID とストリームを取り出し、ストリーム ID に対応する Apache との接続をテーブルより得て、ストリームを Apache へ送信する。

これら 4 つの部分は互いに非同期に動作させるために異なるスレッドとして動作する。自律ディスクからのストリーム受信部と Apache へのデータ送信部は並列にドキュメントを扱うためにさらに複数のスレッドで構成される。

また、Apache 自律ディスクモジュールについても C を用いて実装を行った。

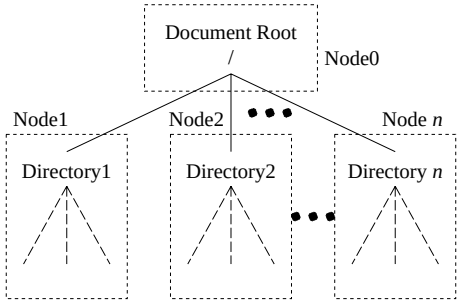

図 6: 比較対象システムでのドキュメントの分散配置

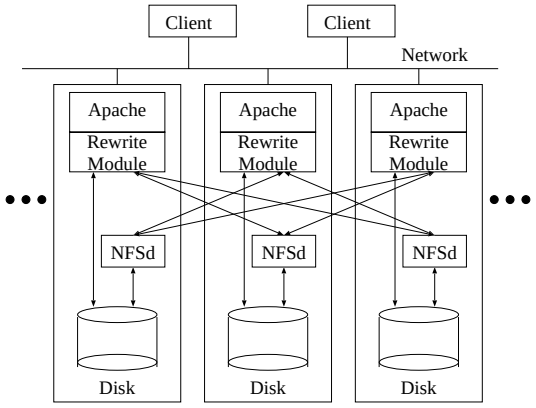

図 7: NFS を用いた Web サーバー

5 性能評価

5.1 比較対象

性能を測定する上で、比較対象としてドキュメントをクラスタ内に分散配置する以下ような 2 つのシステムを考える。これらのシステムでは図 6 のようにディレクトリごとにドキュメントを各ノードに分散させてデータを配置する。

5.1.1 NFS を用いたドキュメントの分散配置

このシステムではドキュメントの読み出しに NFS プロトコルを用いる。その構成を図 7 に示す。この構成では Web サーバーに対する HTTP リクエストは以下の流れで処理される。rewrite モジュールとはルールを用いて URL を内部的に書き換える Apache のモジュールである。

1. Apache は HTTP リクエストを受けると、URL か

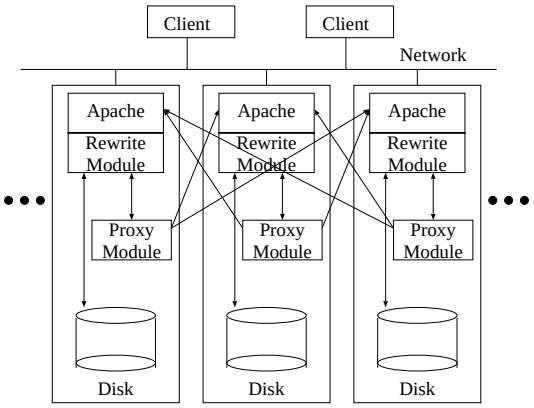

図 8: rewrite モジュールの Proxy Throughput 機能を用いた Web サーバー

らクラスタ内のドキュメントを要求しているかどうかを判定する。要求しているならば rewrite モジュールに URL を渡す。

2. rewrite モジュールはマッピングが書かれた設定ファイルに従って URL に含まれるディレクトリ名からそのドキュメントを持つノードを求める。

3. ドキュメントが自ノードにある場合は Apache がそのドキュメントをクライアントに送信する。他のノードにある場合はドキュメントを NFS プロトコルを用いて読み出し、クライアントに送信する。

5.1.2 Apache の rewrite モジュールの Proxy Throughput 機能を用いたドキュメントの分散配置

この手法はドキュメントの読み出しに Apache の rewrite モジュールと proxy モジュールを用いるものである。簡単のために以下この手法を APT と呼ぶ。その構成を図 8 に示す。proxy モジュールはプロキシー機能を実現する Apache のモジュールである。この構成では Web サーバーに対する HTTP リクエストは以下の流れで処理される。

1. Apache は HTTP リクエストを受けると、URL からクラスタ内のドキュメントを要求しているかどうかを判定する。要求しているならば rewrite モジュールに URL を渡す。

表 1: 実験環境

ノード数	5 台
クライアント台数	1 台
CPU	Intel Pentium III 933 MHz
メモリ	PC133 SDRAM 256MB
HDD	Seagate Barracuda II 20.4GB 7200rpm
OS	Linux 2.2.17
ネットワーク	1000BASE-SX
Java 環境	IBM JDK 1.3.0
Apache	Version 1.3.22

2. rewrite モジュールはマッピングが書かれた設定ファイルに従ってディレクトリ名からそのドキュメントを持つノードを求める。

3. ドキュメントが自ノードにある場合は Apache がそのドキュメントをクライアントに送信する。他のノードにある場合はそのノード名と URL を proxy モジュールに渡す。

4. proxy モジュールは渡された情報を元にドキュメントを持つノードに HTTP リクエストを行い、ドキュメントを得てクライアントに送信する。

5.2 性能測定

Apache 自律ディスクモジュールと自律ディスクゲートウェイの実装を行った上で、PC 上に Java を用いて実装された自律ディスクを用いて Web サーバーを構成し、性能を測定した。測定に使用した環境は表 1 の通りである。ホスト側のネットワークがボトルネックになってしまうので、クラスタ–クライアント間にもネットワークにギガビットイーサネットを用いる。

測定の方法は以下の通りである。

- クラスタ内の各ノードに対して HTTP リクエストを発行する。リクエストは一定の同時接続数で合計 10000 回を連続して発行する。

- 一定サイズのドキュメントをクラスタ全体で 256 個格納する。

- どのノードにリクエストが行われるかはランダムである。
- どのドキュメントをリクエストするかもランダムである。
- これらの条件で測定するためにベンチマークソフトウェアとして http_load[6] を使用する。リクエストごとのレスポンスタイムを出力するようにプログラムを変更している。
- 自律ディスクを用いた手法の場合、自律ディスクゲートウェイにおいてリクエスト先ディスクを選択するルールは、全ノードからランダムに選ぶ方法とする。

これらの条件で以下の測定を行った。

実験 1 自律ディスクゲートウェイにおけるオーバーヘッドの測定

実験 2 ドキュメントサイズの変化に対するスループットの変化

実験 3 ノード数の変化に対するスループットの変化

実験 4 同時接続数の変化に対するスループットの変化

実験 5 各手法におけるレスポンスタイム分布

5.2.1 自律ディスクゲートウェイにおけるオーバーヘッドの測定

自律ディスクゲートウェイにおいて以下の値を測定した。その結果を表 2 に示す。これはドキュメントサイズが 128KB、同時接続数が 15 の時のものである。

- Apache からリクエストを受けて自律ディスクへリクエストを送信するのにかかった時間 (リクエスト処理時間)
- 自律ディスクからのストリームを受信し Apache へデータを送信するのにかかった時間 (ドキュメント処理時間)

これにより、自律ディスクゲートウェイ全体でのオーバーヘッドは 10.34ms となる。リクエスト処理時間は十分に短く、問題ない。ドキュメント処理時間はリクエスト処理時間に比べ長くなっているが、これは自律ディスクから分割された受信したストリームを再構成していることを考えると、妥当な値だと考えられる。

表 2: 自律ディスクゲートウェイにおけるオーバーヘッド

リクエスト処理時間 (平均)	1.15ms
ドキュメント処理時間 (平均)	9.19ms

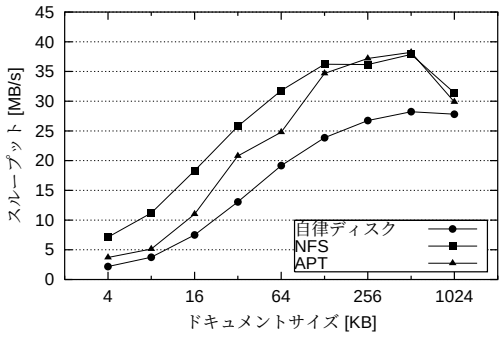

図 9: ドキュメントサイズとスループット

5.2.2 ドキュメントサイズの変化に対するスループットの変化

ドキュメントサイズに対するスループットの変化を図 9 に示す。これは同時接続数が 15 の時のものである。各手法ともドキュメントサイズが大きくなるにつれてスループットが向上している。これはドキュメントサイズが大きくなるにつれて接続の確立や HTTP ハンドリングにかかる時間がドキュメントの転送時間に比べて相対的に短くなるからであると考えられる。

NFS を用いた手法と APT を用いた手法では、ドキュメントサイズが 1024KB の時にスループットが低下している。これは、ドキュメントサイズが大きくなるにつれて各ノードでのディスクキャッシュが効かなくなるためだと考えられる。ドキュメントサイズが小さい場合は NFS が APT を上回っている。APT ではドキュメントが他のノードにある場合に新たな HTTP リクエストが生成されるため、リクエストを解析するオーバーヘッドが存在する。APT ではドキュメントサイズが小さい時にこのオーバーヘッドのためにスループットが低下していると考えられる。

自律ディスクを用いた手法では、28MB/s あたりでスループットが飽和している。これは次の実験の結果

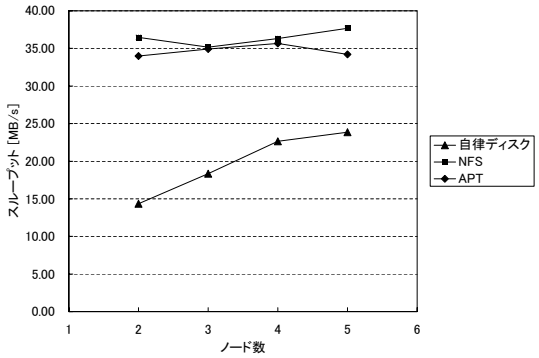

図 10: ノード数とスループット (ドキュメントサイズ =128KB)

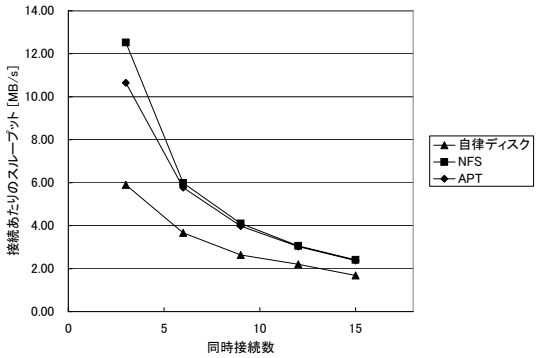

図 11: 同時接続数と 1 接続あたりのスループット (ドキュメントサイズ=128KB)

とあわせると、CPU 能力やメモリなどがボトルネックになっていると予想される。

5.2.3 ノード数の変化に対するスループットの変化

ノード数の変化に対するスループットの変化を、図 10 に示す。これはドキュメントサイズが 128KB、同時接続数が 15 の時のものである。

他の手法がノード数にかかわらずほぼ一定であるのに比べ、自律ディスクを用いた手法ではノード数が増加するにつれてスループットも増加している。測定中に各ノードの CPU 負荷を計測したところほぼ飽和しており、CPU 能力がボトルネックになっていることが判明した。このため、ノード数が増えるにつれて 1 ノードあたりの負荷が減少し、スループットが向上したと考えられる。

5.2.4 同時接続数の変化に対するスループットの変化

同時接続数の変化に対する 1 接続あたりのスループットの変化を図 11 に示す。これはドキュメントサイズが 128KB の時のものである。

いずれの手法も同時接続数が多くなるに従って負荷が大きくなり、1 接続あたりのスループットが低下している。自律ディスクを用いた手法は同時接続数が少ない時はその他の手法に大きな差をつけられている。しかし同時接続数が多くなるに従ってその他の手法のスループットが急激に低下しているため、同時接続数が多く負荷が高い状況では自律ディスクとその他の手法との差が非常に小さくなっている。

5.2.5 各手法におけるレスポンスタイム分布

自律ディスクを用いた手法におけるレスポンスタイム分布を図 12 に、NFS を用いた手法におけるレスポンスタイム分布を図 13 に、APT におけるレスポンスタイム分布を図 13 にそれぞれ示す。これはドキュメントサイズが 128KB、同時接続数が 15 の時のものである。

レスポンスタイム分布は NFS を用いた場合がもっとも良く、自律ディスクを用いた場合はかなりのばらつきが生じてしまっている。これは、自律ディスクを用いた場合は各ノードで CPU 能力が不足しているため、リクエストが処理されるまでに時間がかかってしまっているためであると考えられる。また、Java のガベージコレクションが頻繁に発生し、その時点で処理中のリクエストに処理の遅れが生じていることも予想される。

5.3 性能評価のまとめ

行った性能評価では自律ディスクを用いた手法は他の手法に及ばない結果となった。その理由として、自律ディスクを用いた手法では、自律ディスクゲートウェイでの処理と自律ディスク内部での分散ディレクトリのトラバース処理のオーバーヘッドが存在することがあげられる。また、実装上の理由もあると考えられる。現在、自律ディスクと自律ディスクゲートウェイの実

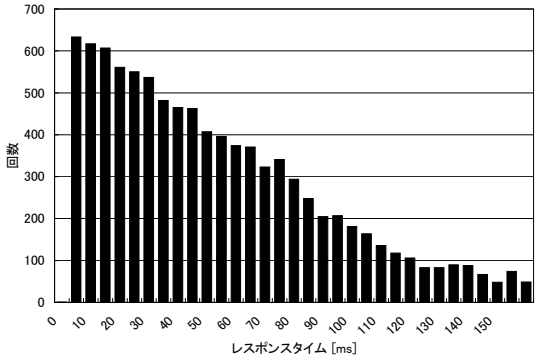

図 12: 自律ディスクを用いた手法におけるレスポンスタイム分布

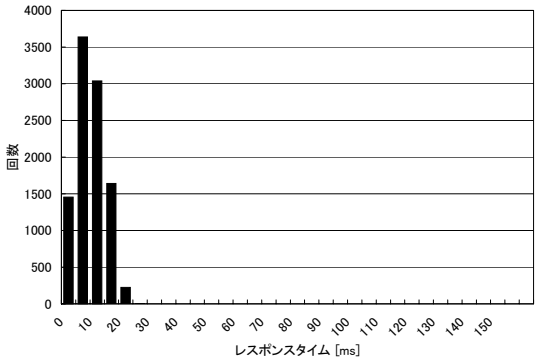

図 13: NFS を用いた手法におけるレスポンスタイム分布

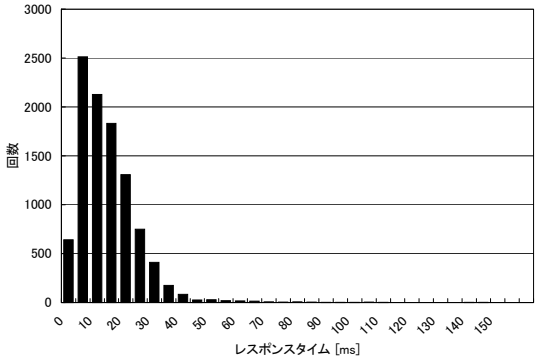

図 14: APT におけるレスポンスタイム分布

装は Java を用いて行っており、CPU 能力、メモリ使用量などの点でオーバーヘッドが大きい。そのため、C++などで実装を行えば性能はさらに向上すると思われる。

これらの点を考慮すると、負荷分散・耐故障機能を持たない NFS を用いた手法や APT に対して約 2/3 の性能が得られたことは良好な結果と考えられる。また、ノード数を増やすことで性能をさらに向上させることができると考えられる。

6 まとめと今後の課題

6.1 まとめ

本稿では自律ディスクを用いてクラスタ化 Web サーバーを構成する手法について述べた。自律ディスクが持つ特徴を利用することで、負荷分散・耐故障処理を自律的に行うクラスタ化 Web サーバーを構成できることを示した。これによってクラスタ化 Web サーバーにおいて、管理コストを減少させることができると考えられる。

また、本手法の中心である自律ディスクゲートウェイと Apache 自律ディスクモジュールの実装を行った上で基本性能を測定した。NFS を用いた手法や Apache の Proxy Throughput 機能を利用した手法によってクラスタ化 Web サーバーを構成した場合にも同様に性能を測定し、本研究での手法との比較を行った。その結果、Java での試作にもかかわらず、負荷分散・耐故障機能を持たない NFS や Proxy Throughput 機能を利用した場合の約 2/3 の性能が得られ、システムの持つ機能および実装上の問題を考慮すると良好な結果が得られた。本手法では管理コストが従来の手法に比べて減少していることから、本手法が大規模なクラスタ化 Web サーバーの構成に有効であると考えられる。

6.2 今後の課題

本稿では実験システムの都合上、最大ノード数が 5 ノードであったので、さらに多数のノードを持つシステム上において本手法の有効性を示す必要がある。また、アクセスが特定のドキュメントに集中する場合、アクセスパターンに時間的な変化がある場合などの負荷に偏りがある場合の性能の測定はまだ行われていない。

自律ディスクはその性質上、ドキュメントの更新を従来のストレージシステムより効率的に扱えるという特徴を持っている。そこで、Web サーバーとして動作中にドキュメントの更新が与える影響についても調べる必要がある。

謝辞

本研究の一部は、文部科学省科学研究費補助金基盤研究 (12680333) および情報ストレージ研究推進機構 (SRC) の助成により行なわれた。

参考文献

[1] Haruo Yokota. Autonomous Disks for Advanced Database Applications. In *Proc. of International Symposium on Database Applications in Non-Traditional Environments (DANTE'99)*, pp. 441–448, Nov. 1999.

[2] Michael Borgwardt and Haruo Yokota. Flexible storage methods in autonomous disks. 情処学会研究会報告, データベースシステム DBS-125-17. 情報処理学会, 2001.

[3] 阿部 亮太 , 横田治夫. 自律ディスクにおける故障時の動作とそのルール処理の実装. 第 12 回データ工学ワークショップ論文集, DEWS2001 2B–3. 電子情報通信学会データ工学研究専門委員会, 2001.

[4] The Apache Software Foundation. Apache http server. http://httpd.apache.org/.

[5] 安部洋平 , 横田治夫. Java による耐故障ネットワークディスクのルール処理の実装. 第 11 回データ工学ワークショップ論文集, DEWS2000 3B–2. 電子情報通信学会データ工学研究専門委員会, 2000.

[6] ACME Laboratories. http_load - multiprocessing http test client. http://www.acme.com/software/http_load/.