データベースシステム 106－3
（1996. 1. 24）

相互結合網構成の並列情報サーバ上のファイル管理

味松 康行　　横田 治夫
mimatsu@jaist.ac.jp　　yokota@jaist.ac.jp

北陸先端科学技術大学院大学 情報科学研究科
〒923-12 石川県能美郡辰口町旭台 15

我々は高信頼並列ディスクシステムを構成するためにDR-netを提案してきた。DR-netを用いて並列情報サーバを実現しようとしたとき、DR-net上にファイルシステムを構築する必要がある。DR-netには複数の外部インタフェースがあるため、複数のアクセス要求が並列に出されてもファイルシステムの整合性が失われないようにすることが重要である。また、多数のディスクを均等に使用するために、各ディスクの使用量を均等化する何らかの機構が備わっていることが望ましい。本稿では、ファイルシステムを構成する2つの制御方式について評価を行なう。実験機を用いた評価実験により、それぞれの方式におけるファイルアクセス要求のレスポンスタイムおよびディスク使用量のばらつきを計測する。実験結果は、並列制御方式が集中制御方式よりも短いレスポンスタイムを示すこと、またファイルサイズが大きい場合には、各ディスクの使用量が均等化されることなどを示している。

File Control in a Parallel Information Server with an Interconnection Network

Yasuyuki MIMATSU　　Haruo YOKOTA
mimatsu@jaist.ac.jp　　yokota@jaist.ac.jp

School of Information Science, Japan Advanced Institute of Science and Technology
15 Asahidai, Tatsunokuchi Nomi-gun Ishikawa 923-12, Japan

We have proposed DR-nets, Data-Reconstruction networks, to construct highly reliable parallel disk systems. When we use DR-nets for a parallel information server, we must provide a file system on DR-nets. DR-nets have a number of interface nodes, so the file system of DR-nets must care to keep consistency of information of files and disks, even if a number of requests come in parallel. Furthermore, the file system should take care to keep the balance of the number of used blocks within all disks. We evaluate two file system configurations using a DR-net prototype. The results indicate that the distributed configuration has quicker response time than centralized one, and it keeps balanced number of used blocks within all disks when file size becomes large.

1 はじめに

近年、プロセッサの処理速度向上にともない、ディスクシステムに対する性能向上の要求が高まっている。また、大容量二次記憶の利用には、ディスクシステムの信頼性向上が不可欠である。しかし、プロセッサやメモリのような半導体製品とは異なり、ディスクシステムには機械的な動作が伴うため、単体ディスクにおける性能向上には限界がある。そこで、複数のディスク装置を並列に動作させることが考えられる。データを複数のディスクに分割して格納し、並列にアクセスすることにより性能の向上が期待できる。

また信頼性に関しても、多数のディスク装置を用いる並列ディスクシステムにおいては、より多くの注意を払う必要がある。RAID として知られる冗長ディスクアレイは、多数のディスクを並列に動作させることにより二次記憶装置の性能を向上させ、さらに冗長情報を用いた信頼性の向上を図っている [1][2]。

しかし、非常に多数のディスクを用いた場合に RAID は必ずしも十分な性能、信頼性を提供するとは限らないように思われる。多数のディスク装置をつなぐ 1 本のバスがボトルネックとなり、性能を低下させることが考えられる。信頼性に関しても、1 つのパリティグループ内では単一のディスク故障を仮定しているが、ディスクの数が増えた場合には、単一故障の前提は必ずしも適当ではないとも考えられる。

我々は、RAID で用いられているパリティ計算の手法を相互接続ネットワークに適用し、上記の問題を解決する方法を提案してきた [3][4][5]。データ再構築ネット (Data-Reconstruction Networks:DR-net) では、ディスクは相互接続ネットワークの各ノードに接続され、その相互接続ネットワークのサブネットでパリティグループを形成する。各ノードは 1 本のバスではなく、相互にネットワークで接続されているためバスボトルネックは存在しない。また、任意の数の外部インタフェースを持つことが可能であり、それらを通してディスクアクセス要求を並列に処理することが可能である [6]。さらに、ディスク故障が存在する際のデータ再構築計算は局所的 なサブネット内で行なわれるため、ネットワークのサイズが大きくなった場合のデータ再構築の速さも劣化しない。

DR-net では、2 種類のパリティグループをネットワーク上に重ね合わせて配置することにより、高い信頼性を実現することができる。5×5 の 2 次元トーラスネットワークを用いた構成では、いかなる 2 つのディスク故障に対しても全てのデータを再構築することが可能であり、最大 9 つのディスク故障に対しても対応できることがわかっている [5]。また、冗長情報の割合の等しい RAID6 と比較すると、パリティ分散をしない構成の DR-net は RAID6 をしのぐ信頼性を持つことが示されている [7]。

近年の WWW に見られるように、多数のユーザを対象として多くの情報を提供するサービスが普及しつつある。また、将来的にはヴィデオ・オン・デマンド等に代表されるような、さまざまなマルチメディア情報を多数のクライアントに提供できる並列情報サーバの構成が望まれている。このようなサーバでは、膨大な情報を安定して提供できることが重要であり、DR-net のような大容量・高信頼の二次記憶が不可欠である。

DR-net を用いてこのようなサーバを実現する場合、DR-net にファイルシステムを構築する必要がある。DR-net を用いたファイルシステムを考えるとき、複数のインタフェースノードから同時にアクセスしても、ファイルシステムとして整合性を保てることが重要である。例えば、新たに空きディスクブロックを必要とするような要求が並行して複数出されても、それぞれの要求が別個の空きブロックを獲得することが保証されなければならない。また、多数のディスクを並列に動作させ、効率良くファイルアクセスするためには、使用中のディスクブロックが一部のディスクに偏るのは避けることが望ましい。

ファイルアクセスを要求するクライアント側でこれらの制約を満たすように工夫することも考えられるが、ディスクの均一な利用やファイルシステムの整合性を保ちつつ効率的なアクセスを実現することは困難である。また、各データノード毎にファイルシステムを構築し、NFS, AFS, DFS[8] 等の分散ファイルシステムを用いてインタフェースノードで統合することも考えられるが、この方法では 1 つのファイルを複数のディスクに分散させることができない。ここでは、DR-net のディスクノードにアクセス管理やファイル分散の制御を行なう機能を付加することによってファイルシステムを実現することを考える。

制御のひとつの方法としては、全てのアクセスを 1 つのノードに集め、逐次的に処理することが考えられる。この方法では、容易に排他制御が可能であり、また、全ディスクの使用ディスク量をそのノードに保持することで、各ディスクの使用率を均一にすることも可能である。しかし、要求が集まるノードがボトルネックになり易く、しかも要求が並列に処理されないために性能的に問題があると思われる。

一方、制御を全てのノードに分散して並列に管理する方法も考えられる。その場合には処理速度向上が望めるが、ファイル管理情報へのアクセスを排他制御する必要がある。本稿では、ファイルシステムとしての整合性を保ちつつ、アクセス要求を各ノードに分散させて並列処理するための制御方法を採り入れ、実験により集中方式との比較を行なう。

以降の本稿の構成を述べる。2節では、並列ディスクシステム DR-net の構成と特徴の概要について述べる。3節では、DR-net 上のファイルシステムを構築する際に考慮すべき点などを説明する。4節では、ファイルシステムの整合性を保ち、均一なディスク使用量を実現する集中および並列の 2 つの制御方式について説明する。5節で、DR-net の実験機に実装した 2 つの制御方式の評価結果およびそれに対する考察を述べる。6節で、まとめおよび今後の課題などを述べる。

2 DR-net の概要

DR-net を構成するネットワークトポロジは各種考えられるが [5]、以下では単純な例として 5×5 の 2 次元トーラスネットワークを考える。その他のネットワークトポロジの場合には、パリティグループの形状等も変更する

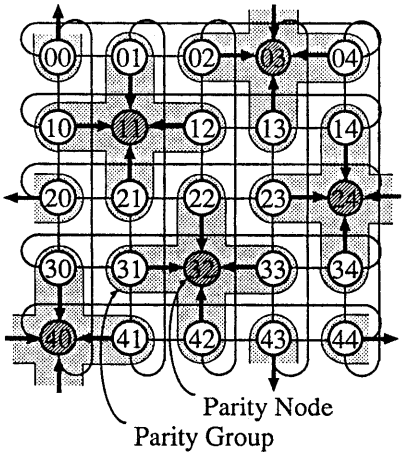

図 1: FPG(First Parity Groups) の構成

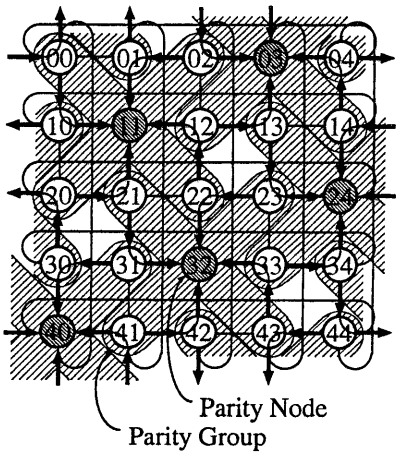

図 2: SPG(Second Parity Groups) の構成

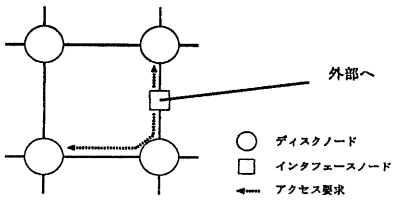

図 3: 外部インタフェースノード

op	i	ファイルネーム

op = ファイルアクセスの種別および必要なパラメータ
(ディレクトリ作成, ファイル削除など)

i = ディレクトリのiノード番号

図 4: ファイルアクセス要求のパケット

ことになる。

パリティグループはデータを保持する複数のデータノードと、それらのデータのパリティを保持するパリティノードから構成される。データノードにデータを書き込む場合にはパリティの更新が伴う。パリティ計算の際に、パリティグループ内での通信を局所的に行なうことが重要であるから、各データノードはなるべくパリティノードの近傍に配置されていることが望ましい。我々は、2種類のパリティグループとして、パリティノードを中心とする十字型と斜め十字型を選んだ。これらを5×5のトーラス上に配置したのが図1, 2である。十時型のパリティグループを FPG(first parity groups)、斜め十字型のパリティグループを SPG(second parity groups) と呼ぶ。

故障ディスクが保持するデータに対して読み出し要求があった場合、そのノードが属するパリティグループ内の他の4つのノードで保持されているデータおよびパリティの排他的論理和を計算し、データを再構築する。故障ディスクに対する書き込みは、そのノードが属するパリティグループのパリティを更新することで実現される。

DR-net では外部とデータを送受信するために、ノード間リンク部にインタフェースノードを設置する (図3)。外部から DR-net へのアクセスは、インタフェースノードを通じて各データノードにアクセス要求パケットを送信することで実現する。インタフェースノードはディスクを持たず、どのパリティグループにも属さない。インタフェースノードにパケットが到着すると自分宛以外のものに対しては何も処理せずに通過させ、単なるリンクのように振舞う。

原理的にはインタフェースノードの数に制限はなく、どのリンク部にもいくつでもインタフェースノードを挿入することが可能である。設置されたどのインタフェースノードからも全てのディスクの全てのデータブロックにアクセスすることができる。多くのインタフェースノードを持つことで、外部とのリンクのバンド幅拡大が期待できる [6]。

3 ファイルシステムの構成

DR-net を用いてファイルシステムを構築するとき、様々な構成が考えられるが、ここでは UNIX と同様な構成を用いた [9][10]。階層的なディレクトリ構造を持ち、ファイルはいずれかのディレクトリの中に収められる。各ディスクにはiノード、データブロック、およびそれらが割り当て済みか空いているかを管理するためのビットマップが保持されている。各iノード、データブロックにはシステム全体で静的に一意に決まる番号 (*num*) が付けられており、それらの番号から当該iノードやディスクブロック

の格納されているディスクおよびディスク内のページ番号などを得ることができる関数mapが定義され、全てのノードが保持している。

$$\mathrm{map}(num) = \text{ノード,ディスク,ページ}$$

ファイルにアクセスするためには、図4のように、アクセスの種類(ディレクトリ作成、ファイル削除など)、ファイルが存在するディレクトリのiノード番号、ファイルネームを指定する。ディレクトリのiノード番号を指定するのは、ファイルの追加や消去の際には親ディレクトリの更新が必要なためである。2階層以上の下のディレクトリにあるファイルをアクセスする際には、カレントディレクトリの直接のサブディレクトリのiノード番号を問い合わせることで1つずつ階層を下り、ファイルの存在するディレクトリのiノード番号を得てから行なう。また、ルートディレクトリのiノード番号は初期状態から決められており、全てのインタフェースノードが知っている。

DR-netは複数の外部インタフェースを持つことができるため、複数のファイルアクセス要求を並行に処理することが考えられる。その際には、各iノードやブロック管理のビットマップの更新は排他的に行なわれなければならない。例として、2つのクライアントA, Bが、それぞれファイルf1, f2を作成するために、あるディスクに新しくデータブロックを確保しようとするとき、

1. クライアントAがディスクからビットマップを読む。
2. クライアントBも同様にディスクのビットマップを読む。
3. Aはブロックbが空いていることを確認する。
4. Bもブロックbが空いていることを確認する。
5. Aはf1にブロックbを割り当てる。
6. Bもf2にブロックbを割り当てる。

このような状況では、2つの別個なファイルf1, f2に同一のデータブロックが割り当てられてしまい、ファイルシステムとしての整合性が保たれない。整合性を保つためには、複数のファイルアクセス要求処理で共有する情報、つまり

(A) 管理ビットマップ(ディスクブロックやiノードの割り当て/解放)

(B) ディレクトリのiノードおよびディレクトリエントリ(ファイルの追加/削除)

にアクセスする際の複数のアクセス要求間の排他制御が必要となる。

4 制御方式

排他制御の方式としては、ある1つのノードでアクセスを集中的に管理し、排他制御を行なう集中制御方式と、全てのノードで制御する並列制御方式が考えられる。本節では、これらの方式について説明する。

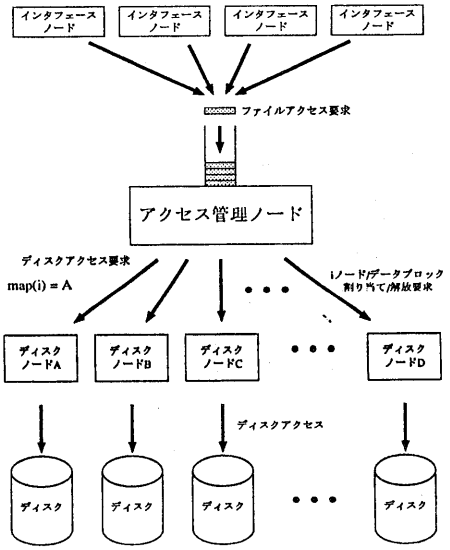

図 5: 集中制御方式の構成

4.1 集中制御方式

集中制御方式のシステムは、複数のインタフェースノード、1つのアクセス管理ノード、複数のディスクノードからなる(図5)。インタフェースノードはクライアントからのファイルアクセス要求を受けとり、アクセス管理ノードに渡す。また、アクセス管理ノードから返される結果をクライアントに送る。アクセス管理ノードは、集められた全てのアクセス要求をキューに入れ、逐次に処理する。処理の際にはディスクノードにディスクアクセス要求、iノード/ディスクブロックの割り当て/解放要求を送り、結果を受けとる。また、アクセス管理ノードは各ディスクの使用量を保持しており、新たなディスクブロックやiノードを必要とするときには、使用量の少ないディスクを持つディスクノードに割り当て要求を出す。ディスクノードは、アクセス管理ノードからディスクブロック/iノードの割り当て/解放要求が来ると、管理ビットマップを更新し、必要ならば割り当てたブロックの番号などをアクセス管理ノードに返す。また、ディスクブロックの読み出し/書き込み要求を受けとり、ディスクアクセスを行なう。割り当て/解放要求とディスクアクセス要求は並行に処理することができる。例として、新たなファイルの作成の過程を示す(ここでは、指定したディレクトリが存在しない場合などのエラー処理は省略してある)。

1. アクセス管理ノードがキューから要求を取り出す。
2. アクセス管理ノードは、ファイルを作成するディレクトリのiノードを読み出すため、iノードを保持するディスクのディスクノードに読み出し要求を送る。
3. 受けとったディスクノードがiノードを読み出し、アクセス管理ノードに送る。
4. アクセス管理ノードは最後のディレクトリブロック

の読み出し要求を送る。

5. 受けとったディスクノードがディスクブロックを読み出し、アクセス管理ノードに送る。

6. アクセス管理ノードは、作成するファイルのiノードおよびデータブロックを割り当てるディスクを決める。各ディスクの使用量を比較し、少ないものから割り当てる。各ディスクノードに割り当て要求を送る。

7. 受けとったディスクノードはビットマップを読み出し、割り当て可能なブロックを探す。割り当てるブロックが決まったら、ビットマップを更新し、割り当てたブロックやiノード番号をアクセス管理ノードに返す。

8. アクセス管理ノードは、確保された各データブロックの番号を確保したiノード内に書き込むため、ディスクアクセスを要求する。受けとったディスクノードがディスクに書き込む。

9. アクセス管理ノードは、ディレクトリに新たなエントリを作成し、ディレクトリブロックやiノードの書き込みを要求する。受けとったディスクノードはディスクに書き込む。

10. 全てのディスクアクセスが完了したら、要求の送信元に完了を通知し、ステップ1に戻る。

この方式では、各ファイルアクセスを逐次処理することにより前述の(A), (B)の排他的アクセス実現している。(A)については、1つのファイルアクセス要求に対して、各ディスクノードは高々1回の割り当て/解放要求しか受けとらないので、排他制御の問題は生じない(1つのディスクで複数のブロック/iノードを割り当て/解放する際にも、要求は1回だけ送られる)。また、1つのファイルアクセスに関係するディレクトリは1つだけなので、(B)に関する問題も生じない。

4.2 並列制御方式

集中制御方式では、一つのアクセス管理ノードに負荷が集中してボトルネックになり易く、複数のインタフェースノードを持つというDR-netの利点を生かせない。また、各ファイルアクセスが逐次処理されるのでアクセス要求間の並列性も生かされない。そこで、複数のファイルアクセスを並列に処理できるようにした方式が並列制御方式である(図6)。並列制御方式ではアクセス管理ノードは存在せず、全てのディスクノードがアクセス管理ノードの処理も兼ねる。

インタフェースノードに送られたファイルアクセス要求は、アクセス管理ノードではなく、ディスクディスクノードに直接送られる。アクセス要求パケットに指定されたディレクトリのiノード番号から、インタフェースノードがmap関数によりそのiノードが格納されているディスクを求め、そのディスクノードへ送られる。

ディスクノードは、集中制御方式でアクセス管理ノードが行なった処理と同じことを行なうが、他のディスクノードからディスクアクセス要求や、ブロック割り当て要求などを受けとった場合には、並行してそれらの処理も行なう。iノード/ディスクブロックの割り当ては、各ディスクノードが独自に全ディスクノードに対しラウンドロビンで割り当てることによりなるべく分散させる。この方式では、要求を受けとったディスクノードはそれぞれ並列に処理を進めることが可能であるため、集中制御方式よりも高い性能が期待できる。

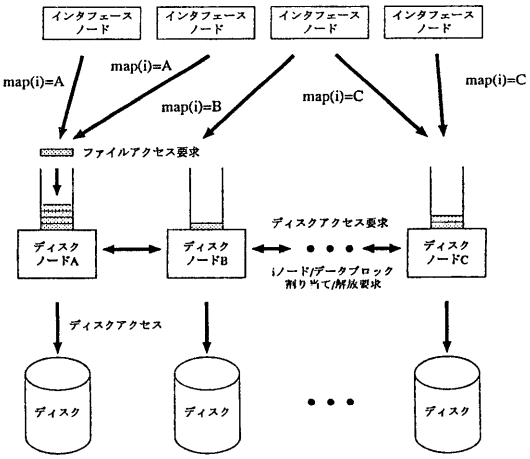

図 6: 並列制御方式の構成

この方式では、前述の(A), (B)の排他制御を次のように行なっている。(A)については、各ディスクノードは1つの割り当て/解放要求を処理している間は、他の要求のためにビットマップを読んだり更新したりしないことで排他制御を行なう。つまり、各ノードにおいて、前節のステップ7にあるビットマップの読み出し・割り当て/解放ブロック(またはiノード)の決定・ビットマップの更新という一連の動作がアトミックな動作として処理される。従って、ビットマップを読んでから書き込むまでの間に、他のアクセスの影響でビットマップの状態が変化してしまうようなことは起こらず、常に整合性は保たれる。(B)については、同一のディレクトリを更新するようなアクセス要求は1つのディスクノードに集められ、逐次処理することで実現されるため、集中制御方式の場合と同様に問題は生じない。

5 評価実験

2つの制御方式を比較するために、実験機を用いた評価実験を行なった。

5.1 実験システム

実験機は5×5トーラスネットワーク構成で、各ディスクノードはトランスピュータT805および2.5″小型ハードディスクからなる[11][12][13]。ディスクの平均シーク時間は17ms, 平均回転待ち時間は8.3ms, 最大転送速度は4MB/s, ディスクのページ長は4KBである。また、4つの

T805を外部インタフェースノードとして設置した。設置場所は図1のノード02と12の間に一台、同様に10と20、23と33、31と41間に一台ずつ設置した。本来ならばこれらのインタフェースノードに外部から要求を送信するが、実験ではインタフェースノードで直接要求パケットを生成させた。また、集中制御方式ではノード00をアクセス管理ノードとした。T805の制御プログラムはOccam2で記述した。

各ディスクは、データブロックゾーン(約20000ページ)、iノードゾーン(500ページ)およびそれぞれの管理用ビットマップ(各1ページ)を持つ。iノードは実装を単純にするため、間接ブロックを用いていない。

5.2 実験内容

実験では、4つのインタフェースノードでそれぞれファイルのアクセス要求を送信させ、処理の平均レスポンスタイムを計測した。また、ある程度アクセスを繰り返した時点で各ディスクの使用容量を計測し、使用容量の分散の度合を調べた。アクセス要求は次の3種類である。

- 存在するディレクトリに新たなファイルを作成する(サイズおよびディレクトリはランダムに指定)
- 作成されたファイルの削除
- 新たなディレクトリの作成

各要求の発生比率は、ファイルの作成が60%、ファイルの削除およびディレクトリの作成がそれぞれ20%である。各インタフェースは、1つの要求の結果が帰ってくるまでは次の要求を出さず、また、実験はルートディレクトリのみが存在する初期状態から開始した。

5.3 評価結果

5.3.1 ファイルサイズが大きいとき

この実験では、作成するファイルサイズを0〜2.7MBの間でランダムに指定し、各インタフェースノードからアクセス要求を500回送った。その結果を表1に示す。表中のMAXおよびMINは、全てのディスクでもっとも使用量が多かったものの使用量、およびもっと少なかったものの使用量である。この差が小さいものほど各ディスクを均等に使っていることになる。

表1から分かるように、集中制御方式ではもっとも使用量の多かったディスクともっとも使用量の小さかったディスクの差が4KB、つまり1ページしかない。これは、アクセス管理ノードにおける各ディスクへの使用量分散処理が非常に効果的であることを示している。一方、並列制御方式は集中制御方式に比べると差が大きいが、各ディスクの合計使用量との比率を考えれば、それほど大きな差ではない。アクセス管理ノードのようにグローバルな情報を持つ存在がなく、各ディスクノードが独立して使用量を分散させているにもかかわらず、全体として分散がうまくいっている理由として、平均ファイルサイズが大きいことがあげられる。大きいファイルサイズ(平均1.35MB)では、1回のファイル作成処理に全てのディスクが使用され、一部のディスクだけに格納されるということは少ない。このため、ファイルの作成・削除を繰り返しても全てのディスクの使用量がほぼ均等に増減するだけで、大きなばらつきは生じにくくなるものと思われる。

	集中	並列
レスポンスタイム(s)	3.39	1.74
ディスク使用量MAX (KB)	40892	44308
ディスク使用量MIN (KB)	40888	44224
ディスク使用量の分散	0.64	354

表1: ファイルサイズ0〜2.7MB, 500回試行の結果

	集中	並列
レスポンスタイム(s)	2.70	1.19
ディスク使用量MAX (KB)	1500	1604
ディスク使用量MIN (KB)	1496	1476
ディスク使用量の分散	0.64	884

表2: ファイルサイズ0〜96KB, 500回試行の結果

	集中	並列
ディスク使用量MAX (KB)	6484	6548
ディスク使用量MIN (KB)	6476	6248
ディスク使用量の分散	10.7	5540

表3: ファイルサイズ0〜96KB, 2000回試行の結果

レスポンスタイムについてみると、集中制御方式に比べて並列制御方式は1.9倍の速度向上である。このことから、並列制御により複数の要求が並列に処理され、速度向上の効果があることが確認できる。並列制御では最大でインタフェースノード数(実験では4)に等しい要求が並列に処理される。しかし、並列に処理する場合でも同一ディスクへのアクセスは逐次処理されるため、ファイルサイズが大きい場合、つまり1つのアクセス要求の処理で多くのディスクを使用する場合にはディスクアクセス要求が競合し、並列化の効果が小さくなったと思われる。

5.3.2 ファイルサイズが小さいとき

ファイルサイズを0から96KBまでの間でランダムに指定し、各インタフェースノードから500回の要求を送った場合の結果を表2に示す。最大96KB(24ページ)のファイルサイズでは、1回のアクセス要求で25個全てのディスクを使用することはない。従って、各ディスクでの使用量のばらつきが大きくなるものと思われる。並列制御方式のディスク使用量のばらつきを見ると、ファイルサイズが大きいときよりもばらつきが大きくなっており、このことが確認できる。しかし、集中制御ではファイルサイズが小さい場合でもアクセス管理ノードによる分散が効果的に行なわれており、ばらつきが小さく抑えられていることが分かる。試行回数を2000回に増やした場合の結果を表3に示す。試行回数を増やしても同様の傾向が見られることが分かる。

	最大	最小	分散
ファイルサイズ大 (500 回)	24	13	213
ファイルサイズ小 (500 回)	26	11	267
ファイルサイズ小 (2000 回)	76	46	940

表 4: ディレクトリの i ノード数のばらつき

また、レスポンスタイムに関しては、集中制御に比べて並列制御では約 2.3 倍の速度向上を示している。これは、ファイルサイズが小さいために複数のアクセス要求で競合するディスクが少なくなった結果と思われる。

5.3.3 ディスクノードの負荷分散

並列制御方式では、ディレクトリの i ノードを持っているディスクのディスクノードがファイルアクセス要求を処理する。従って、ディスクブロックの使用量が同じでも、ディレクトリの i ノードをたくさん持っているディスクノードは負荷が重くなると考えられる。各ディスクでのディレクトリの i ノード数のばらつきを示したものが表 4 である。この結果から、ディスク間でかなりばらつきがあることが分かる。これは、i ノード数の分散は行なっているが、ファイルの i ノードとディレクトリの i ノードの区別をしていないため、ディレクトリの i ノードが偏ってしまう可能性があることと、1 つの i ノードが複数のディスクにまたがって格納されることはないため、ファイルサイズが小さい場合のディスク使用量同様の理由で、ばらつきが大きくなったと考えられる。

6 おわりに

DR-net を用いた 2 種類のファイルシステムの構成について報告した。DR-net には複数の外部インタフェースノードがあり、それぞれが並列にファイルアクセス要求を出せるため、ファイルシステムの管理情報へのアクセスは排他制御する必要がある。また、より多くのディスクを並列に動作させ性能向上を図るという点から、全ディスクの使用量をなるべく均等にすることが望ましい。

1 つのアクセス管理ノードに全ての要求を集める集中制御方式は、ディスク使用量の均等化に優れているが、1 つのアクセス管理ノードがボトルネックになることと、アクセス要求間の並列性がいかされないために、レスポンスタイムが長くなる。一方、並列制御方式では、処理の負荷が複数のディスクノードに分散され、複数のアクセス要求を並列に処理することが可能であるため、レスポンスタイムが短くなる。また、ファイルサイズが大きいときには、各ディスク使用量は比較的均等に保てる。

実験では、並列制御方式における各ディスクノードの負荷の目安となるディレクトリの i ノード数にかなりばらつきがあることが明らかになった。今後は、ディレクトリの i ノードをなるべく全ディスクに均等に分散させる研究が必要である。また、レスポンスタイムだけでなく、スループットの計測や i ノードをある程度キャッシュしたときの性能に関しても評価を行ないたい。

今回はディスク故障がない場合について調査したが、ディスク故障が発生した場合についても研究する必要がある。ディスク故障が存在すると、そのディスクの内容にアクセスするためにはパリティグループ内でデータの再構築処理を行なう必要があるため、アクセス速度が低下する。従って、1 つのファイルが全て故障ディスクの中に保持されていれば、そのファイルのどの部分へのアクセスも遅くなる。しかし、ファイルが多くのディスクに分散されていれば、故障ディスクに保持されている部分にアクセスするときのみ速度が低下し、残りの部分へのアクセスは通常とほぼ同じ速度で行なわれるため、ファイル全体で平均すればさほど速度が低下しないと思われる。また、故障が発生したことを検出した場合には、故障ディスクへの新たなブロック割り当てを他のノードへ振り替えることで、負荷を軽減することなども考えられる。これらについても、研究してみたい。

参考文献

[1] D. A. Patterson, G. Gibson and R. H. Katz: "A Case for Redundant Arrays of Inexpensive Disks(RAID)", Proc. of ACM SIGMOD Conference, pp. 109–116 (1988).

[2] G. A. Gibson: "Performance and Reliability in Redundant Arrays of Inexpensive Disks", Technical report, UCB, Computer Science Division (EECS), University of California, Berkeley, California 94720 (1989).

[3] 横田："RAID のネットワーク上への展開と信頼性向上", 信学技報 CPSY 93-11, 電子情報通信学会 (1993).

[4] 横田："データ再構築ネット (DR-net) における不均衡対策", 信学技報 FTS 93-20, 電子情報通信学会 (1993).

[5] H. Yokota: "DR-nets: Data-Reconstruction Networks for Highly Reliable Parallel-Disk Systems", Proc. of 2nd Workshop on I/O in Parallel Computer Systems, pp. 105 – 116 (1994). (Also in ACM Computer Arch. News, Vol. 22, No. 4, pp. 41–46, Sep, 1994).

[6] 味松, 横田："ネットワーク結合並列ディスクの外部インタフェース", 信学技法 CPSY 95-34, 電子情報通信学会 (1995).

[7] 味松, 横田："並列ディスクシステムのパリティグループの構成の変化と信頼性の比較", 信学技法 FTS 95-34, 電子情報通信学会 (1995).

[8] P. Honeyman: "Distributed file systems", Distributed Computing: Implementation and Management Strategies (Ed. by R. Khanna), Prentice Hall, pp. 27 – 44 (1994).

[9] A. S. Tanenbaum: "MINIX オペレーティングシステム", アスキー (1989).

[10] M. K. McKusick and W. N. Joy: "A Fast File System for UNIX", ACM Trans. on Computer Systems, **12**, 3, pp. 181 – 197 (1984).

[11] S. Tomonaga and H. Yokota: "An Implementation of a Highly Reliable Parallel-Disk System using Transputers", Proc. of the 6th Transputer/Occam Intn'l Conf., IOS Press (1994).

[12] H. Yokota and S. Tomonaga: "The Performance of a Highly Reliable Parallel Disk System", Proc. of the World Transputer Congress '94 (Eds. by A. D. Gloria, M. R. Jane and D. Maini), IOS Press, pp. 147–160 (1994).

[13] 横田, 友永："高信頼並列ディスクプロトタイプのアクセス性能", 信学技報 FTS 94-37, 電子情報通信学会 (1994).