データベースシステム　122－65
（2000. 7. 28）

自律ディスクの基本処理のオーバヘッド解析

阿部 亮太　　横田 治夫
ryota@de.cs.titech.ac.jp　　yokota@de.cs.titech.ac.jp

東京工業大学
大学院 情報理工学研究科 計算工学専攻
〒152-8552 東京都目黒区大岡山 2-12-1

あらまし　我々は、ディスク装置中の制御用のプロセッサやキャッシュ用の大容量メモリを有効活用し、装置内で高度な機能 (負荷分散、故障対策、障害回復等) を実現するアプローチとして、自律ディスクを提案してきた。本稿では、PC 上の JAVA で試作した模擬自律ディスクの基本処理におけるコンポーネントの実行時間を計測し、オーバーヘッドを解析した。その結果、自律ディスクの機能を実現する上での有効性を示している。また、JAVA のキューの実装によるレスポンスタイムが大きいこと、スループットとしては、並列動作の効果がでていること等が判明した。更に、負荷分散のためのマイグレーションでは、自律ディスクの効果が十分期待できる事を示している。

キーワード　自律ディスク、SAN、NAS、耐故障性、負荷分散

Overhead Analysis of Basic Processing in Autonomous Disks

Ryota ABE　　Haruo YOKOTA
ryota@de.cs.titech.ac.jp　　yokota@de.cs.titech.ac.jp

Graduate School of Information Science and Engineering,
Tokyo Institute of Technology
2-12-1 Oookayama, Meguro,
Tokyo 152-8552, Japan

Abstract

Abstract　We have proposed *autonomous disks* to realize high functionalities (balancing load, tolerating faults, and recovering failures),by utilizing a control processor and a large amount of memory for the disk cache in a disk drive. In this paper, we emulate autonomous disks using Java on PCs, and analyze results of their overhead. The analysis results indicate that queues implemented in JAVA considerably increase response time, but parallel accesses are effective for throughput. Moreover, they also suggest that the autonomous disks are suitable to migration for balancing load.

key words　autonomous disks, SAN,NAS,fault tolerance,load balance

1 はじめに

近年、半導体技術の向上により、ディスク装置中にも飛躍的に高性能のプロセッサと大容量のメモリが搭載されるようになり、ディスク制御やキャッシュ処理だけでなく、より効率的にデータを処理するためにも利用されるようにっている。これは、カリフォルニア大学バークレイ校のIDisk[1]や、カーネギーメロン大学のActive Disk[2]、カリフォルニア大学サンタバーバラ校とメリーランド大学共同のActive Disk[3]といった高機能化ディスクとして提案され、ディスクのプロセッサやメモリ上でアプリケーションを実行するというものである。

また、ストレージ・セントリックな構成として、SAN(Storage Area Network)が注目されている。SANは、ホスト間とのLANとは別にホストと上記のような高機能化ディスク間にネットワークを設ける記憶エリアネットワークのことであり、コンピュータと2次記憶装置の間、あるいは2次記憶装置間を直接接続し、コンピュータからの記憶装置の読み書きや、2次記憶間でのデータの移動を行うための専用ネットワークである。あるいは、LANを介してディスクを接続するNAS(Network Attached Storage)も注目されている。

我々は、このNASあるいはSANに接続するディスクとして、高機能化ディスクを更に発展させた自律ディスクを提案してきた[4]。自律ディスクは、ディスク装置内のプロセッサとメモリにより、ディスクの故障対策、障害回復、負荷分散等を行うものである。我々は、まず、ネットワークの利用効率を中心にコストを見積もり、従来のディスクを用いた場合との比較を行い[5]、実際にPC上に自律ディスクの機能を実装し、その有効性を確かめると共に、実行時間を計測した[6]。見積もりにより従来ディスクを用いた場合と比較して自律ディスクの有効性は示せたが、PC上の実装では思った以上にレスポンスタイムが長く、オーバヘッドが大きいことが分かった。そこで、本稿では、PC上の模擬自律ディスクの改良を行うとともに、コンポーネント単位の時間を計測し、そオーバヘッドの解析を行う。

2 自律ディスク

自律ディスクは、それぞれのディスク装置が単にデータを格納するだけでなく、ネットワークに接続された自律したノードとして動作するものとして考えており、以下の機能を持つ事が要求されている。

- 負荷分散
- アクセス制御
- 故障対策
- 障害回復
- 異種のデータの格納
- ディスクの動作記述に対する柔軟性

さらに自律ディスクは、以上のような機能に加えて、ホストとのインターフェースとしてストリームをベースとしており、より効率的な制御を行うことを目的としている。ストリームインターフェースには、ストリームの検索(Search)、挿入(Insert)、削除(Delete)、トラバース(Traverse)などが挙げられ、これらの機能をECAルールを用いて記述し、ホストからの要求の処理や負荷分散、故障時の動作などを実現している。ユーザがルールを記述する事で、ディスクにユーザレベルの柔軟性を取り入れ、動的に状況に対応する事を可能としている。

[5]におけるコスト見積もりでは、同一機能を実現しようとした場合の従来のディスクと比較して、ホストから1つ1つの操作を指示する必要がない分、効率的であることが示され、Insert(挿入)処理では自律ディスクは2倍以上の有効性を示した。また、挿入するストリーム数が増えたとしても1ストリームにかかる時間は、ほぼ変わらないことがわかった。

3 評価実験

[6]ではPC上のJAVAで実装した模擬自律ディスクにおいて、基本処理の一つであるInsertにかかる処理時間を計測を行った。実装に十分時間が使えなかったこともあるが、得られた性能は十分とは言えなかった。性能が十分でない理由がどこにあるのか、実装の改良を行うと伴に、コンポーネント単位の時間を計測し、オーバヘッドの解析を行う。

3.1 実験環境

実験にはLANに接続されたPC(Pentium II 450MHz、メモリ126MB、Linux2.2)4台を使用し、2台をデータディスク、故障対策のためのログディスク1台の場合(多重化度1)と信頼性を上げるためのログを多重化した2台の場合(多重化度2)を想定している(図1)。

ログディスクは分散させるより、集中させたほうがパフォーマンスが良いことが[6]で分かっているため、ここではログ集中方式を取り入れ、WALプロトコルを用いている。ホスト側には、同性能でメモリ256MB、WindowsNT

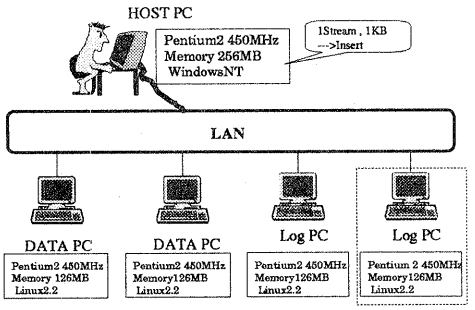

図 1: 実験環境

を用いる。レスポンスタイムの計測においては、1ストリームのサイズは1KBとし、Insertするストリーム数を増やしたときの処理時間から1 Insertにかかる処理時間を算出する。

3.2 Insertのモデル

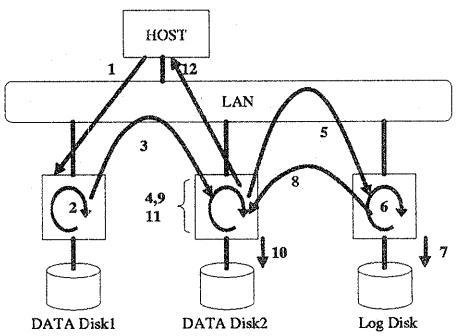

図 2: Insertの例

この実験におけるInsertの1例を図2に示す。この例では、HOSTがLAN経由でデータディスク1に、あるデータをInsertする要求を発したのだが、目的のデータはデータディスク2に格納すべきものであったとする。

まずHOSTはデータをディスク1に送る(1)。この要求を受け取ったディスク1はディレクトリとルールによりにディスク2の識別子を獲得し(2)、データを転送する(3)。ディスク2では、ルールによりデータのInsertを認め(4)、先にログ情報をログディスクへ送る(5)。ログディスクでは、ログ情報を処理し(6)書き込み(7)、それが成功した場合、ディスク2へレスポンスを返す(8)。ディスク2ではInsertを行い(9,10)、HOSTへレスポンスを返す(11,12)。

3.3 レスポンスタイム

まず我々は、1 Insertにおけるレスポンスタイムを計測した。その結果、1Insertあたりのレスポンスタイムは、以下のようになっている。

- データディスク1台・ログディスク1台の場合

  ····· **223ms**

- データディスク2台・ログディスク1台の場合

  ····· **451ms**

この結果は、当然の事ながら十分とは言えない。むしろかなり遅いと言える。そこで、まず、このレスポンスタイムのオーバヘッドの解析を行う。

3.4 コンポーネント計測

試作システムの1Insertにおける処理時間の解析のためにInsertをコンポーネントごとに分割し、実験、処理時間の計測を行う事で、Insertの処理時間との比較をする。実装に関しては、試作システムと同じくJAVAを用いている。コンポーネントとしての、通信時間、ディスクI/O時間、ルール処理時間の評価を行う。

3.4.1 通信実験

- **通信のみの実験**

  ノード間の通信時間の計測のために、JAVAで2ノード間の通信のみを行う実装を行った。ポートを見に行くだけのため、要求を受け取るホストは何の処理も実行しない。表1は、測定結果より得られた2ノード間の片道の通信時間である。

Linux–> Linux 通信	Windows NT –> Linux
約 0.32 ms	約 1.7ms

表 1: PC間の通信時間 (通信のみ)

比較対象として我々のLAN内でのLinux間のPingが約0.3msである事を考えるとLinux間通信の場合、良い結果が出ていると言える。自律ディスク間の通信に関してはJAVAによる実装ががボトルネックになることはないと思われる。

- **通信 & 処理 の実験**

2ノード間で一方のホストが要求を出し、もう一方のホストがその要求を受け取り、処理し、レスポンスを返すという実験を行う。表2は、通信の往復にかかる時間に加えて要求を受けた側での処理時間が加わっている。

Linux-> Linux 通信	Windows NT -> Linux
約 1.27 ms	約 1.58ms

表 2: PC 間の通信&処理時間

この結果と通信のみの結果を比較することで、JAVAで実装されたディスク上でのリクエストの処理時間の見積もりを行うことができる。すなわち表1、2より、
1.27ms-0.32ms × 2=0.63ms
がルール処理にかかる時間と見積もる事ができる。

- キューを用いた場合の通信実験

我々の試作システムでは、複数の要求を受け付けるためにキューを用いている。図3のとおり、リクエストを受け取ったら、キューにため、処理をしホストに返すだけである。データディスクが一台の場合と2台の場合の計測を行った。通信実験のため、ディスクアクセスはしていない。結果は、表3に示してある。

データディスク 1 台	212ms
データディスク 2 台	438ms

表 3: キューを用いた場合の通信&処理時間

よって、表1、2、3よりキューでの処理時間は、

215 − 1.58 = 213.32ms となり、他のコンポーネントの処理時間と比較してかなり大きいといえ、ボトルネックになっていると思われる。

3.4.2 ディスクI/O実験

ディスクのI/Oを計測するため、1ノード上でディスクにファイルを書き込むプログラムを実装し、ストリーム(データ)数ごとの処理時間を計測した。結果は表4に示してある。

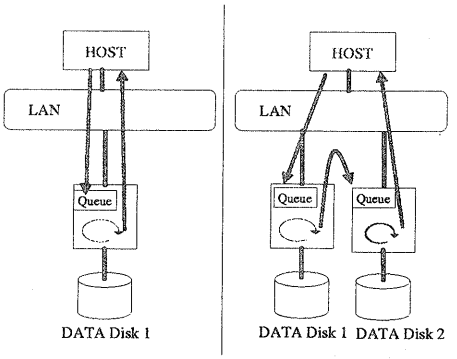

図 3: キューを用いた場合の通信実験

3.5 レスポンスタイムのオーバヘッド解析

コンポーネントごとの処理時間をまとめると表5のようになる。

ホストディスク間通信	1.80ms
ディスク間通信	0.32ms
キュー処理時間	213ms
ルール処理時間	0.63ms
データディスクアクセス時間	1.23ms
ログディスクアクセス時間	2.67ms

表 5: コンポーネント処理時間

この結果をもとに1 Insertにおけるオーバヘッド解析を行う。

それぞれのコンポーネント処理時間を図2で示したInsertに当てはめると図4のようになり、1Insertにかかる時間は438.08msと見積もることができる。ここでは、ログディスクにキューを用いていない。

この見積もりでも分かるように、試作システムでのInsertのオーバヘッドの大部分は、キューでの処理時間となっており、ボトルネックとなっている。しかし、キューを実装することで、複数の要求を扱うことが出来るようになり、スループットを上げる事が出来るのである。以下で、試作システムのスループットについて言及する。

Stream 数	100	500	1000	2000	3000	4000	5000
合計時間 (ms)	96	473	1099	2370	3641	6760	13507
1Stream の平均時間 (ms)	0.96	0.946	1.099	1.185	1.214	1.69	2.67

表 4: Disk I/O 時間

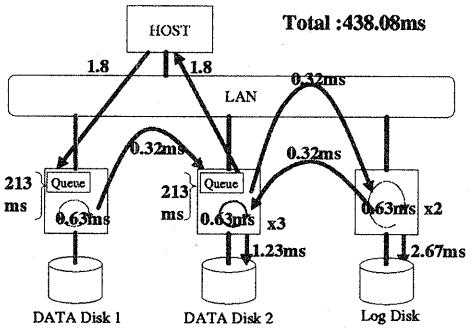

図 4: コンポーネントを元にした Insert のモデル

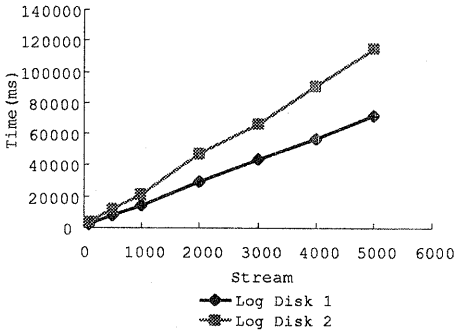

図 5: Insert におけるストリーム数と時間

3.6 複数要求の同時処理

3.3 の結果で 1 Insert のレスポンスタイムは 423ms であったが、実際には複数要求の処理は並列に行われており、スループットは上がる。ここでは、まず複数要求に対する平均処理時間を計測し、次にストリームサイズを変化させた場合のスループットを計測する。

3.6.1 複数要求に対する平均処理時間

我々は、複数 Insert の平均処理時間を計測するため、3.1 で示した環境のもとでログ多重化度 1 とログ多重化度 2 の場合の実験を行った。測定結果は、表 6、7 に示す。

これより、PC 上での模擬自律ディスクの 1Insert あたりの平均処理時間は、多重化度 1 の場合は約 14ms、多重化度 2 の場合は 22.5ms であり、ストリーム数と処理時間の変化を図 5、6 で示している。図 6 では、ストリーム数が増えても 1Insert 平均処理時間は変わらないことが分かる。また、この値はレスポンスタイム 425ms と比較しても、格段に良くなっている。これは、キューにより複数の要求を受け付けることができ、それぞれのリクエストがオーバラップしてパイプライン的に走っているためである。また、これは、[6] で示された結果よりは性能が上がっている。

3.6.2 1 ストリームサイズを変化させた時のスループット

ここでは、1 ストリームのサイズを変化させた場合の結果を示す (表 8)。ストリーム数は 500 とし、ストリームサイズは、2、4、8、16、32KB の場合を計測した。

逐次処理の場合、レスポンスタイムからストリームサイズが 1KB の時、1000/432 = 2.31KB/s のスループットにしかならないが、表 8 からもわかるように実際にはストリームサイズが 1KB の時でも 70KB/s のスループットが出ており、ストリームサイズが増えることにより 121KB/s まで上がり、かなり良くなっているといえる。

また表 8、9 をグラフに表すと図 7 となり多重化度を上げると性能が落ちることもわかる。

データディスクを増やすなどをすることにより、スループットは更に上がる事が予測される。しかし、ディスク単体のスループット (数 MB/s) と比較すると、まだ、かなりの差があり、キューを使った実装の改善、さらには Java での実装の善し悪しも含め、今後、検討する必要がある。

4 考察

4.1 キュー処理がない場合の見積もり

今後、キュー処理の時間がどの程度改善できるか分からないが、キューの処理時間がほとんどない場合を仮定

Stream 数	100	500	1000	2000	3000	4000	5000
合計時間 (ms)	2063	7051	13649	29293	42351	56331	71663
1Stream の平均時間	20.6	14.1	14.0	14.6	14.1	14.1	14.3

表 6: 複数 Insert 要求に対する平均処理時間 (多重化度 1)

Stream 数	100	500	1000	2000	3000	4000	5000
合計時間 (ms)	3084	11297	21180	47408	66205	90740	114910
1Stream の平均時間	30.8	22.6	21.2	23.7	22.1	22.7	23.0

表 7: 複数 Insert 要求に対する平均処理時間 (多重化度 2)

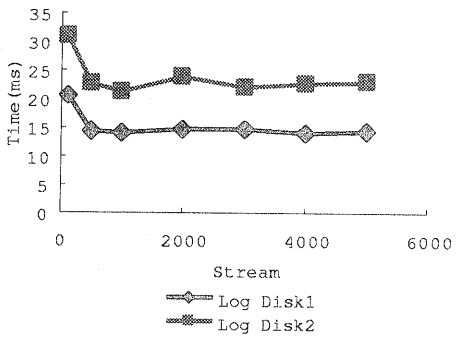

図 6: 1 ストリームあたりの Insert 時間

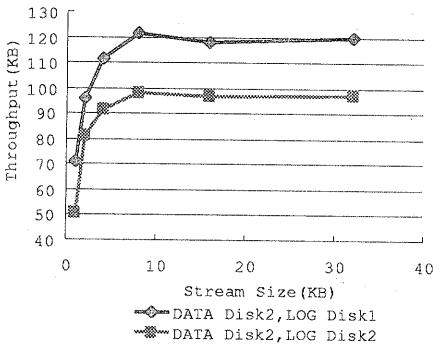

図 7: ストリームサイズとスループット

して、コストの見積もりを行うと図 8 になり、処理時間は約 12ms となる。

これは、3.6.1 の複数同時要求に対する平均レスポンスタイムに近い。このことから、キュー処理がスループットに対しては、あまり影響をしていないことが分かる。

4.2 従来のディスクとの比較

我々は、自律ディスクと従来のディスクのコストの比較を行うためにディレクトリが Fat-Btree であり、キュー処理にほとんど時間がかからない場合の見積もりを行っており、表 9 に示すような式を立てている [5]。

これは、従来のディスクはホスト主導型の通信のため、ホストとディスクの通信が頻繁に起こり、ネットワークの通信がオーバヘッドを上げていることを示している。

これより、実装ではディレクトリ構造に Fat-Btree でなくレンジパーティションを用いているが、従来のディスクが図 2 と同じ操作を行った場合、4 章の解析結果を利用し

従来のディスク	自律ディスク
$(2H+8)Chd(D)$	$2Chc(d)$
	$+\left\{\left(\sum_{k=1}^{N-1} 1-p(h)\right)+3\right\}\times Ccc(D)$
$+(H+4)Cdacc$	$+(H+4)Cdacc$
$+(H+4)Crw(D)$	$+(H+4)Crw(D)$

表 10: 自律ディスクと従来のディスクのコスト見積もり

てコストを見積もると図 9 のようになり、1 Insert にかかる時間は、23.82ms と見積もることができる。

[5] のコストの見積もりでは、ディスクアクセス時間やルール処理時間 (従来ディスクでは、ホストでの処理となる) は等しいとしているため、ネットワークを介した通信の差がこの値である。

しかし、ここでの見積もりは、キュー処理、ルール処理の時間を考慮していない。実際には、キュー処理がかなりの部分を占めており、ボトルネックとなっている。事実、初期の自律ディスクの仮定では、ディスクの CPU は、ホストでは使われなくなった一段レベルの低い CPU を使用

Streamサイズ	1KB	2 KB	4KB	8KB	16KB	32KB
時間 (ms)	7051	10435	17916	32937	69500	133482
スループット (KB/秒)	70.8	95.8	111.6	121.4	117.9	119.8

表 8: Insert におけるストリームサイズとスループット (多重化度 1)

Streamサイズ	1KB	2 KB	4KB	8KB	16KB	32KB
時間 (ms)	11297	12297	21862	40779	82678	164817
スループット (KB/秒)	50.6	81.3	91.5	98.09	96.8	97.1

表 9: Insert におけるストリームサイズとスループット (多重化度 2)

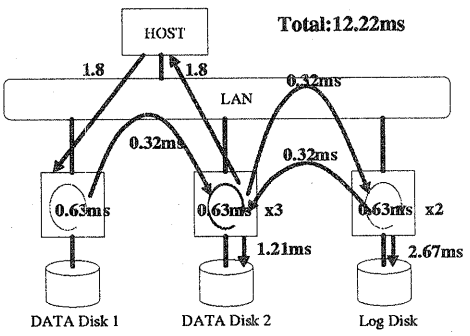

図 8: キュー処理のない場合のコスト見積もり

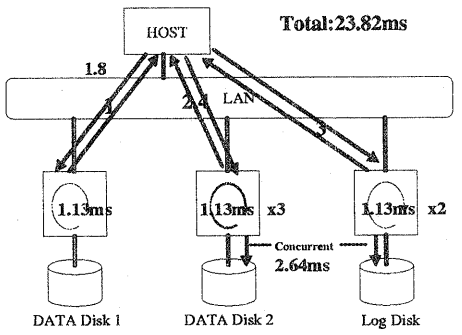

図 9: 従来のディスクのコスト見積もり

することで安価に、かつリソースの再利用を実現することを目的としている。その場合、ディスクでの処理時間は、ホストでの処理時間よりオーバヘッドが高いものとなる事も考えられる。本稿では、ディスクもホストも同じ CPU を用いての実験だったが、今後は CPU パワーの違う状況での測定を行い、従来のディスクとの比較を行うことを考えている。

4.3 マイグレーション

我々は、自律ディスクが持つ機能の一つとして、データや負荷が偏った場合の自律的なデータのマイグレーションを考えている。マイグレーションのイメージを自律ディスクと従来ディスクの場合を図 10 に示す。

従来のディスクは、マイグレーションもホスト主導であり、負荷分散を行いたい時には、ホストからの指示に従って行われる。それに対して、自律ディスクの場合、ルールにより負荷分散が自動的に発火されるため、ホストが関わる事はない。さらに複数の要求がないのでキューの必要もない。ここでは、3 章の解析結果をもとに、マイグレーションのコストの見積もり結果も表 10 に示す。

	自律ディスク	従来のディスク
ホストディスク通信	なし	4 回× 1.8ms
ディスク間通信	1 回× 0.32ms	なし
ディスク I/O 時間	1 回× 2.64ms	1 回× 2.64ms
ルール処理時間	2 回× 1.13ms	2 回× 1.13ms
計	5.22ms	12.1ms

表 11: マイグレーションのコスト見積もり

表 10 より、マイグレーションの場合は、さらに自律ディスクと従来ディスクの差が大きくなる事がわかる。今後、試作した模擬自律ディスク上で実際に計測する予定である。

5 まとめ、今後の課題

本稿では、自律ディスクの Insert に関してのオーバヘッドを測定することで、JAVA で実装された自律ディスクのコストの解析を行った。ルール処理の時間やログを多重化した場合の時間、スループット等を考えた場合、まだ

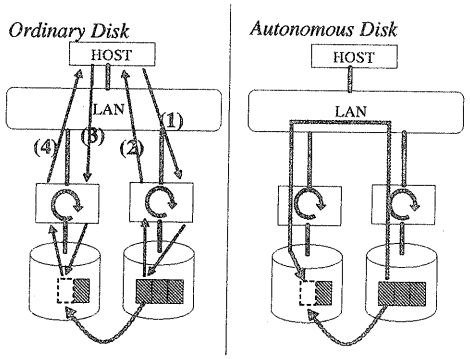

図 10: マイグレーション

一般的に考えて自律ディスクは遅いといえる。今回の結果での最大のボトルネックであるキュー処理に関しては、今後改善していく予定である。ただ、自律ディスクはスピードのみを求めているのではなく、多くの機能を持たせる事でホストのリライアビリティを上げることも目的としている。マイグレーションは、その一例といえる。現在の試作システムでは、ディレクトリ構造にレンジパーティションを用いている。今後、自律ディスクにマイグレーションの機能を持たせるには、ディレクトリを tree 構造にしなければならない。我々は、キュー処理問題とともに、まず Btree の実装をし、その後 Fat-Btree までに発展させるつもりである。

謝辞

本研究の一部は、情報ストレージ研究推進機構 (SRC) の助成により行われた。

参考文献

[1] Kimberly K.Keeton David A. Patterson, and Joseph M. Hellerstein. A case for Intelligent Disks (IDisks). SIGMOD Record, 27(3): 42-52.Sep.1998.

[2] Erik Riedel, Garth Gibson, and Chritos Faloutsos. Active Storage for Large-Scale Data Mining and Multimedia. In Proc. of the 24th VLDB Conf. ,pages 62-73,1998

[3] Anurag Acharya, Mustafa Uysal, and Joel Saltz, Active Disks: Programming Model, Algorithms and Evaluation. In Proc. of the 8th ASPLOS Conf., Oct. 1998.

[4] Haruo YOKOTA. Autonomous Disks for Advanced Database Applications. In Proc. of DANTE'99, pages. 441-448, NOV. 1999.

[5] 阿部 亮太, 横田 治夫. 自律ディスクと従来のディスクのコスト比較. 第 11 回電子情報通信学会データ工学ワークショップ論文集,DEWS2000 3B-1,2000.

[6] 安部 洋平, 横田 治夫. Java による対故障ネットワーク接続ディスクのルール処理の実装. 第 11 回電子情報通信学会データ工学ワークショップ論文集,DEWS2000 3B-2,2000.