バージョン管理用差分情報のアクセス頻度に着目した分散データ配置

Distributed Data Allocation Considering Access Frequency of Differential Information for File Versions

中野 真那♡ 小林 大♡ 渡邊 明嗣♡
上原 年博♢ 田口 亮♢
横田 治夫♠

Mana NAKANO Dai KOBAYASHI
Akitsugu WATANABE
Toshihiro UEHARA Ryo TAGUCHI
Haruo YOKOTA

本稿では，並列分散ストレージシステム上でファイルのバージョン管理を行いながら，システムの負荷分散と容量分散を行う方法について検討する．最新のバージョンとその時点までの更新差分情報を保存して過去のバージョンへのアクセスを行うバージョン管理法を前提とし，バージョンが古くなるほど, そのアクセス頻度が最新のバージョンに対して低くなる点に着目して，アクセス負荷分散は最新バージョンのアクセス頻度に基づいて行い，データ格納量の平坦化をアクセス頻度が低く細かい粒度の容量調整が可能な差分情報で行う．このための分散ディレクトリ構造を提案し，シミュレーションによってその負荷分散と容量分散に対する効果を評価する．

In this paper, a method of allocating data used in a version management mechanism on a distributed parallel storage system is discussed to balance both access frequency and data amount on each storage device. We assume that the version management mechanism keeps the latest version and a number of differential information sets to access any previous versions. In the mechanism, usages of the latest version and differential information are different, and the access frequency for an aged version is tend to be lower than that for the latest version. Considering these facts, we make the access frequency distribution be adjusted by the placement of the latest versions, while the data amount distribution by the placement of the differential information sets whose size is enough small to adjust the subtle difference of data amount. We propose a distributed directory structure for the method, and evaluate its effect on the access frequency and data amount distribution.

1. はじめに

近年コンピュータ内に蓄積されるデータ量は著しく増加し, 性能や信頼性の面から大容量ストレージは並列分散構成される. これは, 多数のストレージ装置（ディスク）をネットワークで結合し, データを各ストレージ装置に分割して格納する. このようなシステムでは, データへのアクセス分布が偏り一部のディスクにリクエストが集中すると, レスポンスタイムが極端に遅くなることが知られている [9]. 処理に必要なリソースが確保できなくなるためにシステムのスループット低下も起こる. 場合によってはシステム障害につながるため, ストレージ装置間のアクセス量を均衡化する必要がある. また, リソースを効率よく使用するためには, 装置間のデータ格納量も均一にすることが望ましい. このように, 分散ストレージ上のデータ配置は, 各ディスク装置のアクセス負荷と格納データ量が均等になるように決定する必要がある. 一般にアクセス負荷分散とデータ格納量分散（容量分散）を両立させることは困難であるが, レポジトリ内のデータに対するアクセス頻度の傾向やデータサイズに注目した配置を行うことで両方をある程度満足させることができる.

システムに配置されるデータの種類には様々なものが考えられるが, ここではバージョン管理に利用されるデータに注目する. バージョニング手法はこれまでに多数提案されており, そこで用いられるデータを利用することでディスクのアクセス量やデータ格納量を均等にできると考えられる.

ソフトウェアや CAD の開発環境, 論文の作成などといった繰り返しファイルに変更が発生する作業では, ファイルのバージョン管理を行って変更内容を記録し, 後から特定のバージョンを取り出したり, いつどんな修正を加えたのかを調査可能にすることが必要とされる [4, 7]. 変更履歴の管理をするために差分を利用する方法が多数提案されている. これは基準となるファイルとファイルの更新差分を保管しておき, 基準ファイルに差分を適用することによって任意の時点の状態にファイルを戻すことを可能にする [1–3, 6, 8, 10]. RCS [10],SCCS [8],CVS [2] などはテキストファイルのバージョン管理を行うツールで,Reverse-delta [10] で管理される. これは基準ファイルとしてファイルの最新版を持ち, 更新のたびに前のバージョンに戻るための差分情報を新しく保存する. 本稿では, 最新版のファイルと更新ごとに作成される差分情報を保存する形でバージョン管理が行われる際に, バージョン管理のための情報が置かれるレポジトリを並列分散ストレージ上に効率よく実現する方法について検討する.

以下に本稿の構成を示す. 2. では用語の定義を行いバージョン管理方法とその特性について分析する. 3. ではその分析に基づいたアプローチを示し, 4. で配置法の要件とそれに適したアクセス構造を述べる. 5. ではデータ配置のアルゴリズムを示し,6. では提案手法に基づいて配置を行う際のアクセス量とデータ量の分散の様子をシミュレーションを行うことで評価する. 最後に 7. で成果についてまとめる.

2. データ特性の分析

2.1 用語の定義

以下に用語の定義を行う.

ファイルのある時点の状態のことをバージョンと呼ぶ. バージョン管理対象のファイルのことをバージョンファイルと呼ぶ. レポジトリに置かれるデータは, バージョンファイルの最新版が書き込まれている最新版オブジェクトと, 毎回の更新差分情報が書き込まれている差分オブジェクトである. 差分オブジェクトはファイル名とバージョン番号で識別される. また, バージョンファイルに対する更新が行われるたびに, 差分オブジェクトが作られることを想定する. レポジトリ上でのデータ管理単位は個々のオブジェクトとする.

2.2 オブジェクトの特性

まず, オブジェクトのアクセス量について考える. Reverse-delta では, ファイルの古いバージョンを取り出すときに, 最新版オブジェクトと必要なバージョンまでの間に存在する差分オブジェクトすべてが必要である. よって, 第 n 版を示

♡ 学生会員 東京工業大学 大学院 情報理工学研究科 計算工学専攻
{mana,daik,aki}@de.cs.titech.ac.jp
♢ NHK 放送技術研究所
{uehara.t-jy,taguchi.r-cs}@nhk.or.jp
♠ 正会員 東京工業大学 学術国際情報センター
yokota@cs.titech.ac.jp

す差分オブジェクトへのアクセス量は, 第 1 版から第 n 版までのオブジェクトへのアクセス量の合計値となる. 一般的に, 古いバージョンに対するアクセス量は新しいバージョンへのアクセス量に対して低いと考えられるので, 差分オブジェクトが古くなるとそのアクセス量は最新版オブジェクトのアクセス量に比較して極端に小さくなる. ただし, 古い差分オブジェクトはアクセスされることが少なくなっても, そのバージョンを読み出すときのために適切にアクセス可能な状態を保つ必要がある. また, ソフトウェアの安定版のように, 読み出しのみの目的でアクセスが多く発生するバージョンに対してはスナップショットが存在すると考えられるので, ここでは, 古いバージョンの読み出し時に差分オブジェクトのアクセスが必要となる, 最新版とスナップショットの間のバージョンのアクセス傾向について考える.

次に, オブジェクトのデータサイズについて考える. レポジトリに保存されるオブジェクト数とデータ総量はバージョン数が増えることによって増大する. バージョンファイルの更新時に追加されるオブジェクトの大きさは更新内容によってばらつくが, バージョンファイルに対して修正が繰り返される状況では, 差分オブジェクトのデータ量は最新版オブジェクトに比べてある程度小さいと考えることができる.

バージョンファイルの性質をまとめる. (性質 1) 差分オブジェクトは, バージョンファイルのバージョンの数だけ存在する. (性質 2-1) 古いバージョンの読み出しには, 最新版オブジェクトとそのバージョンまでの差分オブジェクトのすべてが必要である. (性質 2-2) 作成時点の近い差分オブジェクトは同時にアクセスされる可能性が高い. (性質 3-1) バージョンファイルのアクセス時には, その最新版オブジェクトは毎回アクセスされる. (性質 3-2) 古い差分オブジェクトのアクセス量は最新版オブジェクトのアクセス量に対して急激に減少する. (性質 3-3) 差分オブジェクトは最新版オブジェクトに比べてサイズが小さい. (性質 3-4) 差分オブジェクトのアクセス量は, 時間とともに低下する.

3. アプローチ

ここでは, ストレージ上のファイルは Btree を用いた分散ディレクトリで管理され, 値域分割により適当なディスク装置に配置されるものとする. 性質 3-4 からアクセス傾向は時間とともに変動するので, オブジェクトが作られてからの経過時間とアクセス頻度を考慮した動的な配置決定法が必要である. まず, アクセス頻度の高いファイルの最新版と, 更新のたびに作成される差分情報に効率よくアクセスが可能で, アクセス量やデータ格納量についても管理可能なアクセス構造を提案する. また, このアクセス構造を用いる場合に, それぞれのファイルの更新頻度, アクセス頻度, ファイルサイズを考慮したディスクへのデータ配置アルゴリズムについても検討を行い, 動的な偏り除去操作でのファイル移動基準にバージョンのアクセス量を用いることで各ディスク装置にアクセス負荷やデータ格納量を分散させる手法を提案する.

4. 提案ディレクトリ構造

4.1 構造の要件

性質 3-1 から性質 3-4 で示したように, 最新版オブジェクトと差分オブジェクトを同様に扱うディレクトリ構造は, バージョン管理には適さない. ファイル更新により, ディレクトリに含まれるオブジェクトは増加する. Btree によるディレクトリでは, 含まれるオブジェクトが多くなったときのアクセスパス長の増加量はさほど大きくないが, ディレクトリ内部ノードの数が多くなるため, 木全体をメモリ上にのせることが困難になる. これにより, ディレクトリ検索の速度が低下するが, ディレクトリに含まれるオブジェクトのほとんどはアクセスが少ないと思われる差分オブジェクトである. オブジェクトをすべて均一に扱ってしまうと, よくアクセスされるオブジェクトへのアクセスコストが高くなる.

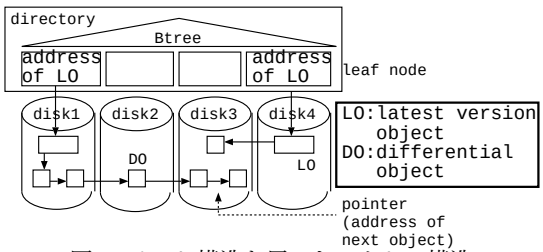

図 1: リスト構造を用いたアクセス構造
Fig.1 Access structure with lists

また, データ格納量の偏りはバージョンファイルの追加や, バージョンファイルの更新で差分オブジェクトが追加されることによって生じ, アクセス量の偏りは, バージョンファイル間や, 差分オブジェクト間にアクセス偏りが存在することによって発生することから, 値域分割によるデータ配置ではデータマイグレーションを行ってもこのどちらかの偏りが除去されない可能性が高い.

性質 2-1,2-2,3-1 をふまえると, 作成されてからの経過時間が短いオブジェクトにはより低コストでアクセスできるディレクトリ構造が望ましい. また, 性質 3-2,3-4 からアクセス頻度の変動を考慮して配置決定を行える必要もある.

4.2 アクセス構造

図 1 に提案ディレクトリ構造のイメージを示す. 最新版オブジェクトはファイル名をキーとした Btree によるディレクトリ管理を行う. ディスクへの配置方法は値域分割とする. 最新版オブジェクトから差分オブジェクトへはポインタを用意する. 差分オブジェクトはバージョンファイルごとに最新版オブジェクトを始点として新しいものから順にポインタでつなぐ. ポインタはディスク上に置かれ, 次の差分オブジェクトのあるディスク装置とディスク上のアドレスを示す. 差分オブジェクトへのアクセスは, 最新版オブジェクトからリストを逐次たどる. このディレクトリ構造の Btree インデックス部には最新版オブジェクトのみが含まれるため, 通常の Btree ディレクトリに比べて小さくアクセスパスも短くなる. また, この小さな Btree をメモリ上に置くことにより, アクセス頻度の大きな新しいオブジェクトに対して高速なアクセスが可能になる. さらに, この構造ではファイルの更新が繰り返されても, 最新版オブジェクトへのアクセスパスには影響がない. 最新版オブジェクトと差分オブジェクトを一つの Btree で管理するディレクトリ構造と比較すると, 古いバージョンのアクセス量が少ないほどこの手法が有利になる. また, これは Btree によるインデクス構造の拡張であるので, バージョン管理対象でないファイルに対しては差分オブジェクトへのリンクを作らずに通常の Btree インデクスとして用いることで, 版管理を行うファイルと行わないファイルを同時に管理することができる.

5. データ配置

5.1 データ配置の方針

一つのバージョンファイルの差分オブジェクトが複数のディスク装置上に配置されている場合, これらのすべての差分オブジェクトを読み出すには, オブジェクトの読み出しコストにディスク移動数 × ディスク間通信時間が上乗せされた時間がかかる. このため同時に読み出し要求をされる差分オブジェクト同士を同じディスク装置に配置することが望ましい. 性質 2-2 と性質 3-1 から, ファイル更新によって新しく作成された差分オブジェクトは, 最新版オブジェクトと同じディスクに配置するものとする. また, アクセス量を分散させるためにはアクセスの集中するディスクからアクセス

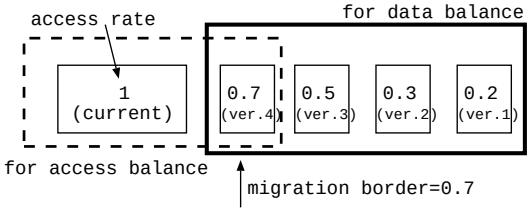

図 2: 差分オブジェクトと移動境界
Fig.2 Differential objects and a migration border

量の多い新しいオブジェクトを選択する. 性質 3-2,3-4 から, 新しいオブジェクトを移動の対象とする. 最新版オブジェクトは値域分割により各ディスクに配置されるため, オブジェクトの移動先は論理的に隣接するディスク装置である. 容量分散の移動対象は古い差分オブジェクトとする. アクセス負荷分散によってアクセス量が均衡しているときにディスク装置のアクセスバランスに影響を与えないために性質 4-2 によりアクセス頻度の低い古い差分オブジェクトを動かす.

元のディスクに残すオブジェクトと, 移動するオブジェクトを決定するために移動境界を定義する. 移動境界はオブジェクトで, 最新版オブジェクトに対するアクセス頻度の割合により指定される. この移動境界より新しいオブジェクト, または古いオブジェクトを移動させることにより, アクセス量とデータ格納量の均衡化を行う. 図 2 の例では 0.7 を移動境界としている.

5.2 アクセス負荷の均等化アルゴリズム

アクセス量の多い新しいオブジェクトを移し, アクセス負荷を分散させる. 移動先は論理的に隣接したディスクである.

1 一定時間間隔で, 各ディスク装置のアクセス負荷とデータ格納量を調べ, ディスクのアクセス量が, 隣接ディスクのアクセス量の一定割合を超えていたら以下を実行する.
2 移動先を隣接ディスクのうちアクセス量が少ないディスクとする.
3 移動境界を指定し, 各バージョンファイルに対して移動させるオブジェクトを決定する.
4 バージョンファイルごとにオブジェクトを移動させる. 移動先のディスクとの間に偏りがなくなるまで繰り返す.

5.3 データ格納量の均等化アルゴリズム

古いほうの差分オブジェクトを移すことによって容量分散を行う. 移動先は格納量の少ないディスクから選択する.

1 一定時間間隔で, 各ディスク装置のアクセス負荷とデータ格納量を調べ, ディスクのデータ格納量が, 隣接ディスクの格納量の一定割合を超えていたら以下を実行する.
2 移動先をシステム内のデータ格納量が少ないほうのディスクとする.
3 移動境界を指定し, 各バージョンファイルに対して移動させるオブジェクトを決定する.
4 バージョンファイルごとにオブジェクトを移動させる. 移動先のディスクとの間にデータ格納量の偏りがなくなるまで繰り返す.

6. 評価実験

6.1 コスト見積もり

各動作で必要なアクセスコストを示す.

Retrieve(filename, version) は,filename と version を指定してファイルの読み込みを行う. 必要なオブジェクトの読み出しコスト＋ディスク間通信量が必要コストである. Update（filename）は,filename を指定して最新版オブジェクトを更新し, 差分オブジェクトの生成を行う. 新しく作られた差分オブジェクトは, 最新版オブジェクトと同じディスクに置かれる. 必要コストは, 最新版オブジェクト書き込みコスト + 差分オブジェクト書き込みコストとする. Migrate(filename, destination) は,filename を指定してファイルを読み出し,destination ディスクに送信, 書き込みを行う. 必要なオブジェクトの読み出しコスト + ディスク間通信量 + オブジェクトの書き込みコストが必要コストである. また, これらのコストには, 移動境界アクセスコストが上乗せされる. 移動境界が n%のときのアクセスコストは, アクセス量が線形減少の場合は, （ディスク内の差分オブジェクトの数）× n/100 × 差分オブジェクトのサイズ + 最新版オブジェクトのサイズとなる. アクセス量が指数減少の場合は, （ディスク内の差分オブジェクトの数）×$\log n$× 差分オブジェクトのサイズ + 最新版オブジェクトのサイズとなる.

6.2 シミュレーション

前章で述べた配置戦略に基づくシミュレータを構築した. 各バージョンファイルへの Retrieve,Update 要求を zipf 分布 [11] に従って発生させて, 一定時間ごとに容量分散とアクセス負荷分散アルゴリズムを実行し. システムの各ディスク装置の格納データ量の標準偏差を測定した.

はじめにマイグレーションを行う場合と行わない場合の偏り状況を比較する. また, 最新版オブジェクトと差分オブジェクトを一つの Btree で管理するディレクトリ構造（Btree ディレクトリと呼ぶ）との比較も行う. Btree ディレクトリでは値域分割によりデータを各ディスクに配置し, 値域を変更することにより隣接ディスク間でデータの配置変更を行う. データ格納量の偏りの様子を図 3 に示す. マイグレーションを行わない方は標準偏差が増加するのに対し, 提案手法ではデータの偏りが低い値に保たれることから, ディスクのデータ格納量を均一にできることが分かる. また,Btree ディレクトリでも標準偏差が上昇する. これは, 値域分割でデータを管理することにより偏り除去の際のデータの移動先が隣接ディスクのみになるためである. 偏りが大きいときはそれを除去しきれず, グラフのように変動する.

図 4 は移動境界とファイルへの要求の比率を変化させたときの様子を表す. これにより, データ格納量を均等にするためには移動境界を高くするほうがよいことがわかる. 同様に, 移動境界を高く設定することでアクセス負荷をより均等にすることができた. 図は省略する.

データマイグレーションを行うときは, 不必要なマイグレーションを発生させないために現在の偏りを悪化させないことが重要である. 移動境界を高く取ると, データ格納量の均衡化のために移動できるオブジェクトが多くなるので, データ格納量を細かく調節することが可能になる. また, アクセス負荷の偏りを調節するときにも, 特にアクセス量の多いオブジェクトを選択して移動することになるので, 移動するデータ量に対して効率よくアクセス負荷を分散させることができ, データ格納量の分布への影響が小さく済む. また, ファイルへの更新要求が読み出し要求に比べて多い場合ではデータ格納量の偏りが大きくなる. ファイル更新によるオブジェクトの増加に均等化が追いついていないことが理由として考えられるが, 更新率がさらに高い場合では格納量の均等化のためのマイグレーションとアクセス量の均等化のためのマイグレーションが相互に起きてしまい悪化する場合がある. これに対しては, オブジェクトの増加速度に合わせてマイグレーションの優先度を切り替えるなどの手法が必要だと考えられる.

7. 結論と今後の課題

本稿では分散ストレージ上でレポジトリを構築する際に, 各ファイルをディスク装置上に効率よく配置する方法について検討を行った.

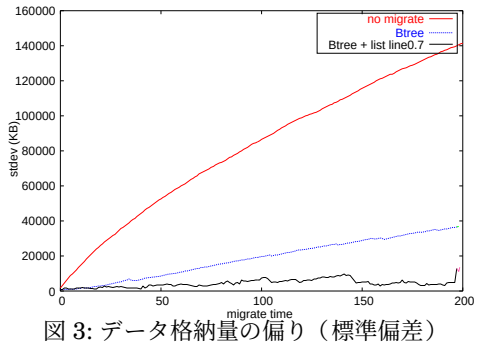

図 3: データ格納量の偏り（標準偏差）
Fig.3 Distribution of data amount (STDEV)

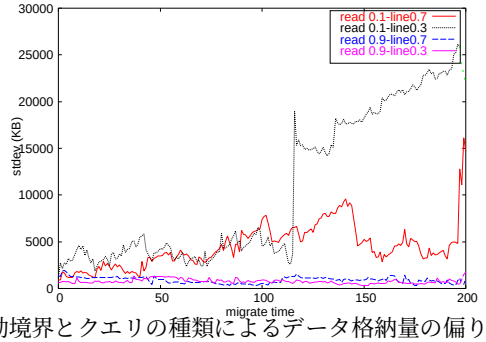

図 4: 移動境界とクエリの種類によるデータ格納量の偏り比較（標準偏差）
Fig.4 Relationship between query and migration border on distribution of data amount (STDEV)

差分オブジェクトのアクセス頻度が古くなるほど急激に低下する点に注目し, ファイルの管理構造として, 最新版オブジェクトの Btree インデクス管理と, 差分オブジェクトのリスト構造管理を併用し, アクセス頻度の高いオブジェクトに効率よくアクセス可能な方法を提案した.

さらに, この構造では差分オブジェクトを任意の場所に配置することが可能なため, オブジェクトのアクセス頻度の低下度合いを用いて容量分散とアクセス負荷分散を行うアルゴリズムを提案した. また, このアルゴリズムを用いて各ディスクに配置を行ったときの分散度合いをシミュレーションにより実験し, パラメータを変えたときの容量分散とアクセス負荷分散の様子を観察した. 移動境界を高く設定することにより, アクセス量やデータ格納量を効率よく分散させることが可能なことが分かった.

今後の課題としては, 配置戦略にファイルの更新速度をパラメータとして組み込み, 極端にアクセス量の高くなるファイルの扱いを考えることがあげられる. また, もっと多数のファイルを含むシステムでの動作についても調べる必要がある. ディスク装置のアクセスコストを削減するために, ディスク装置上におけるポジショニングコストも考慮した分散方式も考える必要がある.

さらに一般的なデータに対しても適用可能なデータ配置方法を考えるためには, 異なる差分管理方式を用いたときの効率のよいオブジェクト配置方法を考える必要がある. 多次元インデクス構造を利用したアクセス構造 [1,5] では時間経過とアクセスコストの増加関係が提案手法とは異なるので, 上で述べたような異なるアクセスモデルが適すると考えられる. これ以外にも, ある時間区間に存在する差分オブジェクトのすべてにアクセスするのではなく, その一部のオブジェクトを選択して古いバージョンの構築が可能なバージョン管理システム [3] に対しても適切な配置方法を考える必要がある.

[謝辞]

本研究の一部は, 独立行政法人科学技術振興機構戦略的創造研究推進事業 CREST, 情報ストレージ研究推進機構 (SRC), 文部科学省科学研究費補助金特定領域研究 (16016232) および東京工業大学 21 世紀 COE プログラム「大規模知識資源の体系化と活用基盤構築」の助成により行なわれた.

[文献]

[1] B. Becker, S. Gschwind, T. Ohler, B. Seeger, and P. Widmayer. An asymptotically optimal multiversion b-tree. *VLDB J.*, 5(4):264–275, 1996.
[2] B. Berliner. Cvs ii: Parallelizing software development. *In Proceedings of the Winter 1990 USENIX Conference*, 1990.
[3] S.-Y. Chien, V. J. Tsotras, and C. Zaniolo. Efficient management of multiversion documents by object referencing. In *The VLDB Journal*, pages 291–300, 2001.
[4] D.B.Leblang. The cm challenge: Configuration management that works, configuration management, ed. *Wiley Co.*, pages 1–38, 1994.
[5] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.
[6] J. MacDonald. File system support for delta compression, 2000.
[7] E. C. R.H. Katz. Managing change in computer-aided design databases. *Proc. of VLDB Conf., Brighton, England,*, Sep. 1987.
[8] M. Rochkind. The source code control system. *IEEE Trans Software Eng SE-1*, pages 364–370, 1975.
[9] H. Simitci. *Storage Network Performance Analysis*. Wiley Technology Publishing, 2003.
[10] W. F. Tichy. RCS — a system for version control. *Software — Practice and Experience*, 15(7):637–654, 1985.
[11] G. K. Zipf. *Human Behavior and the Principle of Least-Effort*. Addison-Wesley, Cambridge, MA, 1949.

中野 真那　Mana NAKANO
東工大大学院・情報理工・計算工・修士課程在学中. 日本データベース学会学生会員.
小林 大　Dai KOBAYASHI
平 17 東工大大学院・情報理工・計算工・修士課程了. 同大大学院・情報理工・計算工・博士後期課程在学中. 日本データベース学会学生会員.
渡邊 明嗣　Akitsugu WATANABE
平 14 東工大大学院・情報理工・計算工・博士前期課程了. 同大大学院・情報理工・計算工・博士後期課程在学中. 日本データベース学会学生会員.
上原 年博　Toshihiro UEHARA
昭 56 慶應義塾大・工・電気工・修士課程了. 昭 59 より NHK 放送技術研究所. 電子情報通信学会, 映像情報メディア学会各会員.
田口 亮　Ryo TAGUCHI
平 6 慶應義塾大大学院・理工・計測工・修士課程了. 同年より NHK 放送技術研究所. 映像情報メディア学会会員.
横田 治夫　Haruo YOKOTA
昭 55 東工大・工・電物卒. 昭 57 同大大学院・情報・修士課程了. 同年富士通 (株). 同年 6 月 (財) 新世代コンピュータ技術開発機構研究所. 昭 61(株) 富士通研究所. 平 4 北陸先端大・情報・助教授. 平 10 東工大・情報理工・助教授. 平 13 東工大・学術国際情報センター・教授. 工博. 日本データベース学会, 電子情報通信学会, 情報処理学会, 人工知能学会, IEEE, ACM 各会員.