ストレージシステムにおけるデータ一貫性を保証したアーカイブ取得方式

出口　彰†　　大森　匡†　　星　　守†

† 電気通信大学大学院情報システム学研究科　〒 182–8585 東京都調布市調布ヶ丘 1–5–1
E-mail: †{deguchi,omori}@hol.is.uec.ac.jp

あらまし　現在，データストレージに関するメタデータを DBMS にて一元管理し，データストレージを利用する際に，この DBMS を通してデータアクセスを行うことで，分散配置されたデータを共有するというデータ管理形態が注目されている．本稿では，この様な環境におけるデータ一貫性を保証したアーカイブ手法に関して議論する．この様な環境において既存のアーカイブ手法によってデータ一貫性を保証するためには，ストレージを停止してダンプを行うか，データとメタデータの操作に対するログを取り続ける事が必要とされる．しかし，ログを取り続ける手法では，ストレージに関するログ量が膨大になりアーカイブには適していない．本稿では，スナップショットの前後一定区間に限ってログを保持するようにログを縮約してリストアする体系を提案し，その原理，手続きを述べる．

キーワード　アーカイブ，リストア，トランザクション，データ一貫性，ストレージシステム

A Novel Archiving Algorithm for Storage Systems under Transactional Consistency

Akira DEGUCHI†, Tadashi OHMORI†, and Mamoru HOSHI†

† Graduate School of Information Systems, The University of Electro-Communications
1–5–1 Chofugaoka, Chofu-shi, Tokyo, 182–8585 Japan
E-mail: †{deguchi,omori}@hol.is.uec.ac.jp

Abstract　In recent few years, a data management concept for sharing distributed data by centralized metadata management about data-storages using DBMS has attracted increasing interests. Data accesses of users are efficiently supported by this concept. In this paper, we describe an algorithm for transactional archiving in this environment. In order to assure transactional consistency by traditional archiving technique in this environment, it is neccesary to take a dump by stopping all storages or to continue acquiring logs for data-storages as well as these metadata. However, the latter way should need a huge amount of logs. To solve this obstacle, we propose a new archiving algorithm for storage systems with much less amount of logs.

Key words　Archive, Restore, Transaction, Data Consistency, Storage Systems

1. はじめに

現在の大規模データ管理システムの分野においては，ファイルシステムなどの一般のデータストレージに置かれたデータ (以後，外部データ) に関するメタデータをデータベース管理システム (DBMS) にて一元管理し，利用時にこの DBMS を通して利用することで，データ共有を行うというデータ管理形態が注目されている．DB2 の DATA-Link 機能 [2] [3] や Oracle Files [8] などが好例である．本研究では，この様な環境におけるデータ一貫性を保証したアーカイブとリストアの手法を提案する．

この様なデータ管理形態の事例として，ニュースサーバなどが考えられる．例えば，ニュース記事自体が通常のファイルとしてファイルシステムにて管理され，その記事間のリンク構造，ファイルパスやニュースの属性などをメタデータとしてデータベースで管理する場合が考えられる．この様にデータベース技術を利用することで非常に高機能なデータ管理を実現することが可能となる．

また，上記の様な複数のリソースが連携することでデータを管理している環境において，従来のアーカイブ技術，すなわちスナップショットと差分ダンプの併用によってアーカイブが行われる場合には，リソース間の同期を取ること，トランザクショ

ンを考慮することができない．このため，アーカイブデータをリストアした際に，“実行したはずの更新操作がデータに反映されていない”，“存在しないはずのデータが存在している” などの異常現象が発生してしまう可能性がある．

そこで，本研究ではトランザクションの概念を利用し，リストア時にトランザクションの境界を保証することで，ACID 性を有したアーカイブとリストアの手法を提案する．

以下，2 節でメタデータを利用するデータ管理形態と，その環境おけるアーカイブを概説するとともに，本研究のアプローチを示す．そして，3 節では本研究で提案するアーカイブ，リストア手法を示し，4 節でその手続を与える．最後，5 節でまとめる．

2. 問題設定と本研究のアプローチ

2.1 メタデータサーバを有するデータ管理形態

前節で述べたデータ管理形態のアーキテクチャを図 1 に示す．図右側に示すデータストレージにおいてファイルなどのデータが管理され，それらのデータに関するメタデータがメタデータサーバ内部の DBMS において一元管理される．利用者は，このメタデータサーバへの読み出し・更新操作を介してデータストレージへの読み出し・更新を行う．この様にメタデータを一元管理することによって，データモデルを定義し高機能なデータ管理ができ，利用者は適切なビューを通して必要とするデータへ効率的にアクセスすることが可能となる．

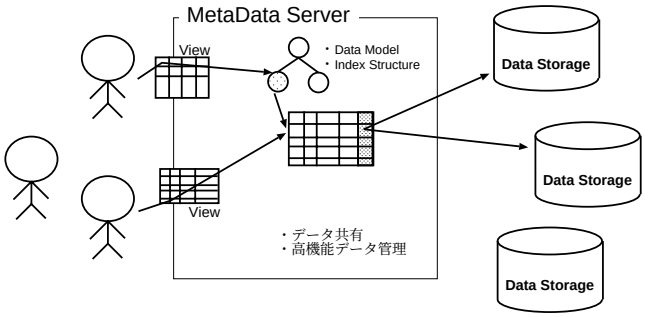


図 1　アーキテクチャのモデル

2.2 複数リソースのアーカイブとデーター貫性

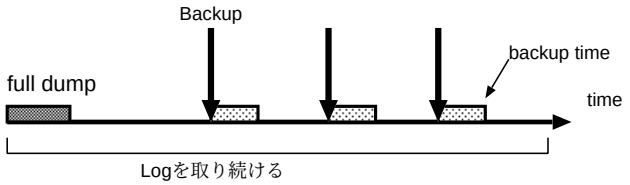
図 1 に示す様に，複数個のリソースの連携によってデータ管理を行っている環境において，トランザクションレベルでの一貫性を保証したアーカイブを行うためには，トランザクションを定義しログを取り続けるか，全てのストレージを停止してスナップショットを作成しアーカイブを行う必要がある．しかし，後者は近年の情報システムにおける高可用性への要求から考えて実用的ではない．また，今後さらにシステム停止許容時間が短くなることが予想される．そこで，本研究では図 1 に示すアーキテクチャを利用してログを取り続ける手法を実現する．同図に示すリソースに加えて，グローバルログマネージャとアーカイブシステムを追加する．このグローバルログマネージャは，メタデータサーバにおけるメタデータ操作に対応するログと，外部データ操作に対応するログを一元管理する．そして，アーカイブシステムは，メタデータサーバとデータストレージが定期的にバックアップ機能やスナップショットによってアーカイブしたデータを管理する．これによって，メタデータとデータなど，複数のリソースにアクセスを行う一連の操作であっても，それらを一つのトランザクションとして定義し，各データ更新について更新前イメージ，更新後イメージをトランザクションの UNDO，REDO ログとして保持することで，従来のデータベース技術によって一貫状態を保証することが可能となる．しかし，ファイルなどの外部データ更新に対して更新前イメージと更新後イメージを常に保持する場合，保持するログ量が非常に膨大になってしまう．

2.3 本研究のアプローチ

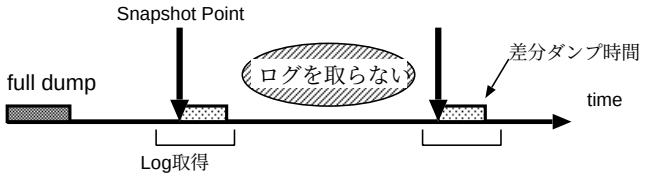
図 2(a) にログを取得し続ける手法を示す．この手法の具体的な事例として，データベースシステムに全てのデータとそのメタデータを格納し，定期的にバックアップ機能を利用しアーカイブを行う場合が考えられる [7] [8]．この場合，一貫性を保証するためにログを取得し続けるために保持するログの量が膨大になってしまいアーカイブには適していない．

そこで，本研究では図 2(b) に示すように，ストレージに関するログ取得を行う区間を一定に限定する手法を提案する．そして，この区間の任意点でのデータ一貫性を保証したリストアを実現する．また，本研究では，メタデータサーバのログは通常通り取得し続けるものとし，ログ取得の限定はストレージに関するログに限って行う．以下，ログ取得の限定という場合にはストレージ側に限っているものとする．

このログ取得の限定は，グローバルログマネージャによってオンラインで取得し続けたログを，定期的にアーカイブのリストア用にバックエンドで変換することによって実現する．これによって，オンラインでのリカバリを実現し，アーカイブのリストアも効率的に行う事が可能となる．



(a) データベースを利用しログを取得し続ける



(b) 本研究のアプローチ

図 2　データベースを利用する手法と本研究のアプローチ

従来，データベースではログを取り続けることによってトランザクションレベルでの一貫性を保証してきた．よって，上記の様にログ取得を限定した状態においてデータ一貫性を保証す

ることは自明な問題ではなく，本研究ではトランザクションのリカバリの技法に基づきこれを解決する．

2.4 前提条件

図 1 に示すアーキテクチャにおいて，スナップショットと差分ダンプの併用によって，外部データストレージを定期的にアーカイブする．このスナップショットはファイル単位で一貫した状態がアーカイブされることを仮定する．この時，トランザクションはファイル単位でロックを獲得し，グローバルログマネージャで保持するログをファイルに対する操作単位で取得すると仮定をおく事によって，ファイル単位で一貫したアーカイブデータをこのログを利用し REDO/UNDO 可能となる．

3. ログの縮約の原理

3.1 ログの縮約とその操作

前述したログ取得の限定と，リストア用へのログの変換を「ログの縮約」として導入する．図 3 に示す様に，定期的にスナップショットと差分ダンプの併用によってアーカイブが行われている時，データストレージの操作に対応するログ取得を同図に示すように一定区間 (以後，ログ取得区間) に限定する．

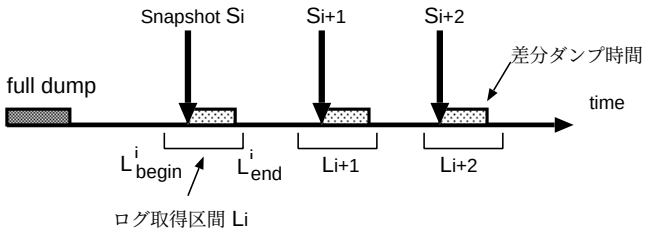


図 3　ログ取得の限定

データ一貫性を保証したトランザクション境界でリストアするための REDO/UNDO 処理は，リストアに利用されるアーカイブデータに基づいて行われる．よって，ログ取得区間 L_i は定期的に取得されているスナップショットポイント S_i を含むような区間であるとする．また，ログ取得区間 L_i を表現するために，メタデータサーバがオンラインで，ログ中にログ取得区間開始を示す L^i_{begin} とログ取得区間終了を示す L^i_{end} を書き出す．

次にログの縮約の具体的な操作を示す．図 4 に示すような形式でデータ操作に対応するログレコードが追加されてくる時に，ログの縮約とはログレコード中の外部データ更新に対する更新前イメージ，更新後イメージを切り捨てることであり，これをそれぞれ「UNDO ログを捨てる」,「REDO ログを捨てる」と呼ぶ．

3.2 REDO/UNDO 不能状態

アーカイブデータをリストアする際にログを利用し REDO/UNDO 処理を行う事で，トランザクションの境界で一貫性を保証することが可能である．しかし，本研究で提案するようにログ取得を図 3 に示すログ取得区間のみに限定する事によって，ログレコード中の更新前，更新後イメージが存在しないために REDO/UNDO 処理を行う事ができない場合が原理的に存在してしまう．

LSN	TRID	Resource Name	Operation	Object	UNDOLog	REDOLog
1	1	FS1	update	ObjectA	ObjectA.old	ObjectA.new
2	1	FS1	update	ObjectB	ObjectB.old	ObjectB.new
3	1	FS1	update	ObjectC	ObjectC.old	ObjectC.new
4	1	FS1	commit			

データサイズが大きい
ログ縮約の対象となる

図 4　ログ縮約の具体的操作

図 5 には，ログ取得区間にまたがっているトランザクションとログ取得区間開始，終了の関係に基づいたトランザクションの実行パターンを示している．この例において，トランザクションの開始を tx_{begin}，完了を tx_{end} とする時，左から二つ目の例では $tx_{begin} < L_{begin} < S$ という関係になっている．リストアが要求された際に，このトランザクションを REDO する場合は S を起点として REDO 可能であるが，UNDO はログ縮約によって更新前イメージが存在しないために実現不能である．同様に，左から三つ目の例では，$S < L_{end} < tx_{end}$ の関係にあり UNDO は可能であるが REDO 不能である．さらに，右端の例は REDO/UNDO の両方が実現不能となってしまう場合である．

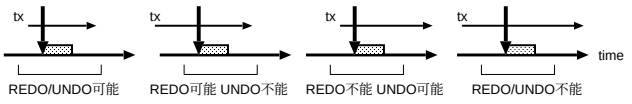


図 5　REDO/UNDO 不能状態

3.3 必要とされるログ

3.3.1 一区間で考える場合

ログ取得区間における任意点でのデータ一貫性を保証したリストアを可能とする時，リストア時に REDO/UNDO 処理の対象となるトランザクションはログ取得区間内において何らかの操作を実行しているトランザクションとなる．すなわち，条件

$$tx_{begin} < L_{end} \ \wedge \ L_{begin} < tx_{end} \qquad (1)$$

を満たすトランザクションのみである(注1)．

アーカイブをリストアする時，データベース分野のリカバリの原理に従い，利用者によって指定されるリストアポイント (R) にアクティブであるトランザクション ($tx_{begin} < R < tx_{end}$) を UNDO することで取り消し，リストアポイントまでにコミットしているトランザクション ($tx_{end} < R$) を REDO することで，トランザクションの境界においてリストアすると仮定する [1]．これに従う限り，いかなるリストアポイント R(ただし，$L_{begin} < R < L_{end}$) が指定された場合でも，$L_{end} < tx_{end}$ の関係にあるトランザクションが REDO されることはない．そして，外部データストレージをリストアする場合には，スナップショットを利用して取得されたアーカイブデータを起点とし

(注1)：この条件は，ログレコード中の単調増加な数である LSN で比較するために等号が成立する場合は考慮する必要がない．

てログの順方向スキャンによって REDO 処理が行われる．このため，図 6 に示す様に REDO ログは S から L_{end} までの区間のみ保持すれば十分である．

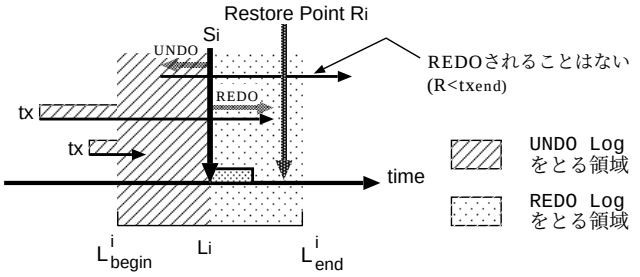


図 6　ログ取得区間を設ける時に必要とされるログ

また，$tx_{begin} < L_{begin}$ の関係にあるトランザクションは UNDO される場合があり，条件 (1) を満たすトランザクションは，ログ取得区間外である tx_{begin} から L_{begin} にも UNDO ログを保持する必要がある．これによって，前節で示した REDO/UNDO 不能状態を解決することが可能となる．そして，REDO 処理同様にリストア時にアーカイブデータを起点としてログの逆方向スキャンによって UNDO 処理が行われる．このため，図 6 に示す様にログ取得区間内における UNDO ログは L_{begin} から S までの区間のみ保持すればよい．よって，UNDO ログに関しては条件 (1) を満たすトランザクションが tx_{begin} から S までの UNDO ログを保持すれば十分であるといえる．

3.3.2　二区間以上で考える場合

スナップショットを利用したアーカイブが定期的に取得されていることと，長期トランザクションの存在から，二つ以上のログ取得区間にまたがって実行されるトランザクションが存在する場合がある．二区間以上にまたがるトランザクションを，$tx_{begin} < L^i_{end} \ \wedge \ L^{i+1}_{begin} < tx_{end}$ を満たすような i が少なくとも一つは存在するようなトランザクションであると定義する．

この時，この二区間以上にまたがるトランザクションのログに関して，tx_{begin} から L^i_{end} の区間は，図 6 にも示したとおりスナップショット S_i からみると REDO ログが必要であり，S_{i+1} からみると UNDO ログが必要となるために，REDO/UNDO 両方のログを保持する必要がある．

このことを，手続化するために形式的に定義し REDO/UNDO 処理に必要とされるログが何であるかを示す．そのために，以下の用語を定義する．

- スナップショットポイント S_i から S_{i+1} までの区間を I_i と定義する．
- トランザクション開始ポイントを示す T_{begin} に対応するログレコードを $\mathcal{T}_{begin}$ と定義する．
- トランザクション完了ポイントを示す T_{end} に対応するログレコードを $\mathcal{T}_{end}$ と定義する．
- ログ取得区間開始ポイントを示す L_{begin} に対応するログレコードを $\mathcal{L}_{begin}$ と定義する．
- ログ取得区間終了ポイントを示す L_{end} に対応するログレコードを $\mathcal{L}_{end}$ と定義する．
- スナップショット作成完了ポイントである S に対応するログレコードを $\mathcal{S}$ と定義する．
- あるログレコード *Log* に対して，*LSN(Log)* はログレコード *Log* に割り当てられた LSN 値であると定義する．

条件 (1) を満たすトランザクション T を考える．この T を構成するアクションの内オペレーションが *write* であるアクションに対応するログレコードを $\mathcal{W}$ とおくとき，次に示す条件に従い，スナップショットポイント S_i に対して更新操作のログレコードからなる集合 A_i, B_i, C_i を定義する．

- A_i : $LSN(\mathcal{S}_i) < LSN(\mathcal{W}) \ \wedge \ LSN(\mathcal{W}) < LSN(\mathcal{L}^i_{end})$ を満たす更新操作に対応するログレコードの集合．
- B_i : $LSN(\mathcal{L}^i_{end}) < LSN(\mathcal{W}) \ \wedge \ LSN(\mathcal{T}_{begin}) < LSN(\mathcal{L}^i_{end}) \ \wedge \ LSN(\mathcal{W}) < LSN(\mathcal{L}^{i+1}_{begin})$ を満たす更新操作に対応するログレコードの集合．
- C_i : $LSN(\mathcal{T}_{begin}) < LSN(\mathcal{S}_i) \wedge LSN(\mathcal{W}) < LSN(\mathcal{S}_i) \wedge LSN(\mathcal{S}_{i-1}) < LSN(\mathcal{W}) \ \wedge \ LSN(\mathcal{L}^i_{begin}) < LSN(\mathcal{T}_{end})$ を満たす更新操作に対応するログレコードの集合．

区間 I_i に注目し，集合 A_i, B_i, C_{i+1} に含まれるログレコードに対応する更新操作が取り得る区間の関係を図 7 に示す．この時，A_i, B_i, C_{i+1} に含まれるログレコードが，それぞれどのような種類のログ (REDO/UNDO ログ) を保持するべきであるかを次に示す．

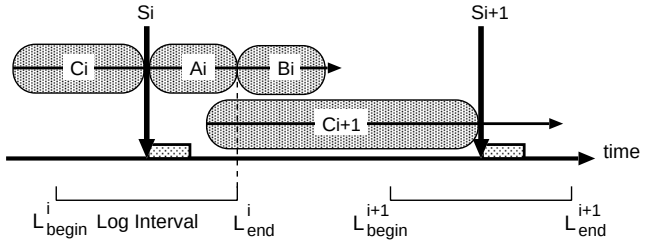


図 7　ログ縮約のための区間分割

- 集合 A_i のみに含まれる更新操作のログレコードに関しては REDO ログ のみを保持する．
- 集合 B_i のみに含まれる更新操作のログレコードに関しては REDO/UNDO ログ を共に保持しない．
- 集合 C_{i+1} のみに含まれる更新操作のログレコードに関しては UNDO ログ のみを保持する．
- 集合 $A_i \cap C_{i+1}$ に含まれる更新操作のログレコードに関しては REDO/UNDO ログ を共に保持する．
- 集合 $B_i \cap C_{i+1}$ に含まれる更新操作のログレコードに関しては UNDO ログ のみを保持する．

以上のログによって各区間の REDO/UNDO 処理は実現される (証明略) [6].

3.4　以前の差分ダンプデータとログの利用

データ一貫性を保証するために必要とされるログをさらに削減可能であることを示す．スナップショットを定期的に取得し

ているという前提から，あるトランザクションを UNDO した場合のデータイメージが，以前のアーカイブデータを利用し生成可能である場合がある．その場合には，そのトランザクションの UNDO ログを削減することが可能である．

図 8 に示す場合は，UNDO ログを保持する必要がない．例えば，図 8(b) の場合では，トランザクション T_2 の *write fileA* に関して UNDO ログを保持する必要があるとしてきたが，T_2 を正しく UNDO したときの *fileA* のデータイメージは，直近のスナップショット S_i によってアーカイブされた *fileA* のデータイメージと等価である．よって，T_2 を UNDO するための *write fileA* に関する UNDO ログを保持する必要はないことがいえる．同様にして，図 8(a) の場合には，直近のスナップショットと，その直後の REDO ログを利用する事で，トランザクション T_2 を正しく UNDO したときの *fileA* のデータイメージを生成可能であるために，UNDO ログを保持する必要はない．

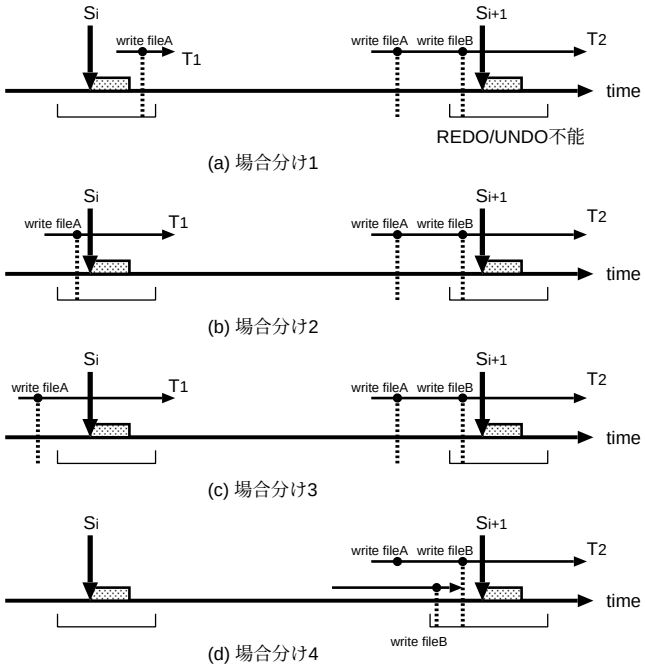


図 8　UNDO 不能状態でのログと差分データの利用

一方，図 9 に示すような場合は，スナップショット S_{i+1} にまたがっているトランザクション T_2 が UNDO される可能性があり，かつ，T_2 は *fileA* にアクセスしている．そして，*fileA* は前回のスナップショット S_i 以後に別のトランザクション T_1 によってアクセスされ，既にその REDO ログがログ縮約によって切捨てられているため，スナップショット S_i からデータを生成することができず，T_2 は *fileA* に関して UNDO ログを取得する必要がある．

また，図 10 に示すように区間 I_{i-1} において更新されたデータは，差分ダンプの性質から考えて必ずスナップショット S_i によって取得される．よって，UNDO ログを取得するかは直近のスナップショットまでの区間におけるデータへのアクセス状況から判定可能である．これを手続きとして実現するために次に示すリストを利用する．

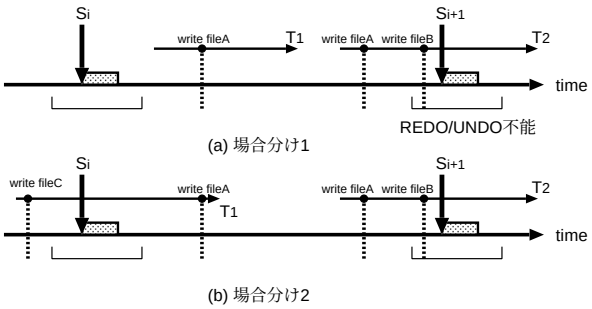


図 9　以前の差分データとログが利用できない場合

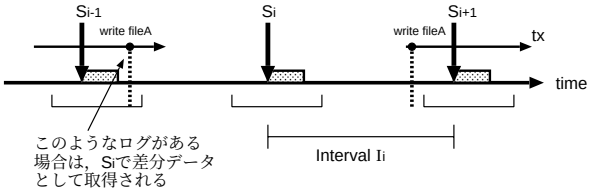


図 10　探索するログと差分データの範囲

- OperationInt[i] ： 区間 I_i において更新され，かつ，REDO ログを保持していない様なファイルと，その更新操作を実行したトランザクション識別番号の組から成るリスト
(<*filename*,TRID | *false* >) ：<>はリスト
TRID は，唯一ファイルを更新したトランザクションのトランザクション識別番号を示す．そして，相異なる二つ以上のトランザクションによってファイルが更新された時には，UNDO イメージが生成不能となる．この場合を *false* で示す．
- ValueLogInt[i] : 区間 I_i において更新され，かつ，REDO ログを保持している様なファイルと，そのログレコードの LSN のリスト
(<*filename*,<LSN >>) : <>はリスト

OperationInt[i] では，直近のスナップショットと REDO ログによる UNDO イメージの生成可能性を判定する．直近スナップショットからファイル f に対する更新操作 w に関する UNDO イメージが生成可能であるための条件は，以下に示す三つの条件の何れかが成立する場合であり，かつ，その場合に限る．

(i)　区間 I_i において，更新操作 w がファイル f を更新した最初の更新操作である．

(ii)　区間 I_i において，更新操作 w がファイル f を更新し，REDO ログを保持しない最初の更新操作である．

(iii)　上記 (i),(ii) の条件のどちらかを満たす更新操作を実行したトランザクションと同値の TRID を持ち，かつ，同一ファイルの 2 回目以降の更新操作である．

条件 (i),(ii) は OperationInt[i] に<f,TRID >が含まれるかどうかによって判定可能である (リストに含まれないことを，後述の手続きでは ϕ で示している)．そして，条件 (iii) は<f,TRID >の TRID と同値の TRID であるかによって判定可能である．

ログの縮約を実行するとき区間 I_i がスキャンされる．この時，更新操作に対応するログレコードに関して OperationInt[i] を生成することによって，直近までのスナップショットと REDO

ログによって，あるトランザクションを UNDO した時のファイルイメージを生成可能であるかを判定可能となる．また，これと同時に ValueLogInt[i] によって，以前の差分ダンプデータと REDO ログを利用してトランザクションの UNDO イメージを生成可能である場合には，その生成経路を示すことが可能となる．

4. ログ縮約の手続きとリストアの手続き

4.1 ログ縮約の手続き

ここでは，前節までのログ取得区間を限定することと，直近のスナップショットからの UNDO イメージの生成可能性を判定し UNDO ログを削減する手法を手続きとして与える．手順は，グローバルログマネージャから切り出してきたログ縮約の対象となるログ列を 2 回スキャンすることによって実現される．

a） 1 回目のスキャン

1 回目のスキャンでは，条件 (1) を満たすトランザクションのトランザクション識別番号 TRID，開始時点 T_{begin}，終了時点 T_{end} から構成されるリスト <TRID, T_{begin}, T_{end}> と，ログ取得区間を示す L_{begin},L_{end} から構成されるリスト $<L_{begin}, L_{end}>$ を作成する．

b） 2 回目のスキャン

2 回目のスキャンでは，更新操作 (write, create, 等) に注目しログを順方向スキャンする．この時，OperationInt[i]，ValueLogInt[i] を更新するとともに，1 回目のスキャンによって作成されたリストを元に，更新操作に対応するログレコードが，前述した集合 A_i, B_i, C_{i+1} のいずれに含まれるかを判定し，それぞれの集合において保持する必要があるログの種類に応じてログを削減していく．以下に，その手続きを示す．

```
if(更新オペレーションである)
  if(1 回目のスキャンのリストに含まれる)
    更新操作が Ai,Bi,Ci+1 のどれに含まれるかを判定
    if(A ∧ C の時)
      fname_trid = OperationInt[i].get(filename);
      if(fname_trid == φ || fname_trid.id == 自分)
        REDO のみ保持  // A に含まれるため
         // UNDO は ValueLogInt[i] の内容から生成可能である
      else // 再現不能の時 (fname_trid.id != 自分)
        REDO/UNDO 保持 // A に含まれ，C に含まれるため
      end if
      ValueLogInt[i] へ <filename,LSN> を追加
                  // REDO ログは保持したから
    else if(B ∧ C の時 || C のみの時){    // C のみと同様になる
      fname_trid = OperationInt[i].get(filename);
      if(fname_trid == φ || fname_trid.id == 自分)
        REDO/UNDO 不要
         //UNDO は ValueLogInt[i] の内容から生成可能である
      else
        UNDO ログのみ保持する
      end if
      PutOperationInt(filename,TRID)
           // REDO ログを保持しないから
    else if(A の時)     // Snapshot の右側
      REDO ログのみ保持する
      ValueLogInt[i] へ <filename,LSN> を追加
           // REDO ログは保持したから
    else if(B の時)
      REDO/UNDO 不要
      PutOperationInt(filename,TRID)
           // REDO ログを保持しないから
    end if
  else  //1 回目のリストに含まれない
    REDO/UNDO の両方が不要
       // ログを取得する必要がないトランザクション
    PutOperationInt(filename,TRID)
  end if
end if
if(操作がスナップショット)
  ValueLogInt[i+1]  を初期化
  OperationInt[i+1] を初期化
end if

procedure putOperationInt(filename,TRID)
  fname_trid = OperationInt[i].get(filename)
  if(fname_trid == φ)  // まだファイルがリストに含まれない
    OperationInt[i] = <filename,TRID>
  else
    if(fname_trid.id == false) // 既に生成不能
      ;
    else if(fname_trid.id == TRID) // 自分がアクセスしている
      ;
    else if(fname_trid.id ≠ TRID) // 自分以外がアクセスしている
      fname_trid.id = false
  end if
end
```

4.2 ログ縮約の具体例

本節では，本稿で提案したログ縮約のアルゴリズムが正確に動作するかを確認するために，具体的なログデータに適用することで検証する．

図 12 に縮約前のログデータを示す．このログデータは，(LSN, TRID, ResourceName, Operation, Object, UNDO Log, REDOLog, prevLSN) の形式であると仮定する．これを，時間軸に基づいて図示したものが図 11 となっている．

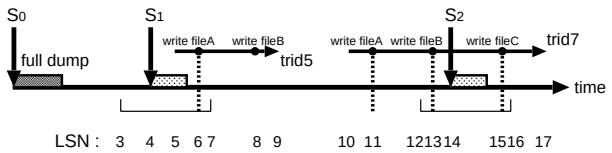


図 11　以前のスナップショットと REDO ログの利用

図 13 には，図 12 のログを縮約した出力を示す．本稿で提案したログの縮約によって保持する必要がない REDOLog 列，UNDOLog 列における REDO/UNDO ログを”null” で示している．そして，スナップショットと REDO ログによって生成可能である UNDO イメージは LSN で示す．例えば，縮約後のログである図 13 における LSN=11 であるログレコードの UNDOLog 列には，ログレコードを生成したトランザクションを正しく UNDO した時のファイル *fileA* のファイルイメージを直前のスナップショット (LSN=4) と REDO ログ (LSN=6) によって生成可能であることを”4+6” と示している．

4.3 リストアの手続き

本節では，外部データストレージに対して，スナップショットを利用して取得したアーカイブデータを，リストア用に縮約したログを利用してトランザクションレベルで一貫した状態でリストアするための手続きを与える．

ここに示す手続きは，従来のデータベース分野において利用されてきたリカバリの諸原理に基づいている [1]．そのリストア処理は以下に示す四つのステップから構成される．

```
0,0,MDS,Lbegin,null,null,null,0
1,1,lv01,snapshot,null,null,null,0
2,2,MDS,Lend,null,null,null,1
3,3,MDS,Lbegin,null,null,null,2
4,4,lv01,snapshot,null,null,null,3
5,5,lv01,begin,null,null,null,4
6,5,lv01,update,fileA,Before,After,5
7,6,MDS,Lend,null,null,null,6
8,5,lv01,update,fileB,Before,After,6
9,5,lv01,commit,null,null,null,8
10,7,lv01,begin,null,null,null,9
11,7,lv01,update,fileA,Before,After,10
12,8,MDS,Lbegin,null,null,null,11
13,7,lv01,update,fileB,Before,After,11
14,9,lv01,snapshot,null,null,null,13
15,7,lv01,update,fileC,Before,After,13
16,10,MDS,Lend,null,null,null,15
17,7,lv01,commit,null,null,null,15
```

図 12　ログ縮約前

```
0,0,MDS,Lbegin,null,null,null,0
1,1,lv01,snapshot,null,null,null,0
2,2,MDS,Lend,null,null,null,1
3,3,MDS,Lbegin,null,null,null,2
4,4,lv01,snapshot,null,null,null,3
5,5,lv01,begin,null,null,null,4
6,5,lv01,update,fileA,null,After,5
7,6,MDS,Lend,null,null,null,6
8,5,lv01,update,fileB,null,null,6
9,5,lv01,commit,null,null,null,8
10,7,lv01,begin,null,null,null,9
11,7,lv01,update,fileA,4+6,null,10
12,8,MDS,Lbegin,null,null,null,11
13,7,lv01,update,fileB,Before,null,11
14,9,lv01,snapshot,null,null,null,13
15,7,lv01,update,fileC,null,After,13
16,10,MDS,Lend,null,null,null,15
17,7,lv01,commit,null,null,null,15
```

縮約によって不要であると判断された

前回のスナップショット(LSN=4)とREDOログ(LSN=6)から生成可能であると判断された

生成不能なので取得した

図 13　ログ縮約後

a）　RESTORE Phase

利用者によってリストアポイント R_{i_0} が指定されると，それに対応する S_i が決定される．そして，スナップショット $S_i(i = 0, \ldots, i)$ によるアーカイブデータをアーカイブシステムからリストアする(注2)．

b）　Analysis Phase

指定されたリストアポイント R_{i_0} までにコミットし，その効果をリストアに反映すべきトランザクションのリストと，未コミットであり，その効果を全て取り消した状態にすべきトランザクションのリストを作成する．

また，以降のフェーズである REDO，UNDO Phase のログスキャン開始ポイント (LSN によって示される) を決定する．REDO Phase 開始 LSN は，R_{i_0} を含むログ取得区間 L_i に含まれる全てのリソースのスナップショット作成ポイント $S_{i_j}(j = 0, \ldots, n, n$ はリソース数) と R_{i_0} の最小値とする．同様に，UNDO Phase 開始 LSN は，S_{i_j} と R_{i_0} の最大値とする．

c）　REDO Phase

REDO Phase 開始 LSN から，R_{i_0} までのログをスキャンする．この時，Analysis Phase で REDO すると判定されたトランザクションのリストに含まれ，かつ，そのログを生成したリソースのスナップショット S_{i_j} より LSN が大きい時のみ REDO ログを適用することで REDO 処理を行う．

d）　UNDO Phase

UNDO Phase 開始 LSN からログを逆順にスキャンする．この時，Analysis Phase で UNDO すると判定されたトランザクションのリストに含まれ，かつ，そのログを生成したリソースのスナップショット S_{i_j} より LSN が小さい時のみ UNDO ログを適用することで UNDO 処理を行い，UNDO すると判定されたトランザクションの UNDO が全て完了した時点でログスキャンは終了する．

以上に示したデータストレージのリストアとともに，これと同期したトランザクションの境界によってメタデータサーバもリストアされる．メタデータサーバのリストアは，従来のデータベースのリカバリ機能によって行われる [1].

4.4　計算コストの見積り

S_i から L^i_{end} に出現する更新操作に対応するログレコード数を V とする．ValueLogInt[i] は，ファイルのリストに加えて UNDO イメージの生成経路を示すための LSN を記憶するために図 14(a) に示すように，各々のファイルに対して LSN からなるリストが存在する．S_i から L^i_{end} に V_v 種類のファイルが出現するとした場合，記憶する LSN の数は，$V - V_v$ 個となる．よって，記憶するデータ数は，$V_v + (V - V_v) = V$ となるために，$O(V)$ となる．L^i_{end} から S_{i+1} に出現し更新されたファイルが F 種類であるとする．OperationInt[i] は F に依存し，$O(F)$ となる (図 14).

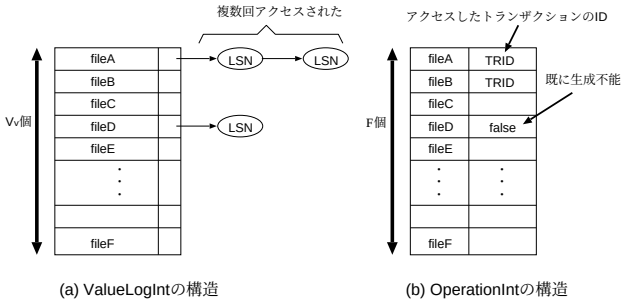


図 14　直近スナップショットのデータを利用可能かを判定するためのデータ構造

4.5　縮約されるログ量に関する予備的検証

図 15 に多少複雑なログ列を示す．この例を通して実際に本稿で示したログの縮約によってどのくらいのログが削減可能であるかを簡単に検証する．同図に示す，ログ列では 11 個のトランザクションが発生し，更新操作の合計が 23 個であり，これらは，全て異なるファイルを更新しているとする．このとき，ログを取得し続けたとすると REDO/UNDO ログが合計 46 箇所取得されることになる．このログ列を縮約したとき，最終的に保持する必要があるログは，REDO ログが 7 箇所であり，UNDO ログが 0 箇所となった．よって，REDO/UNDO ログ取得の回数によって考えた時，$7/46 = 15.2\%$(8 割以上削減される) にまでログが縮約された．さらに，この例ではほとんどのトランザクションがログ取得区間にまたがっている．実際の，

(注2)：正確にはフルダンプを取得したスナップショットからでよい．本研究ではフルダンプは最初の 1 回 $(i = 0)$ のみと仮定する．

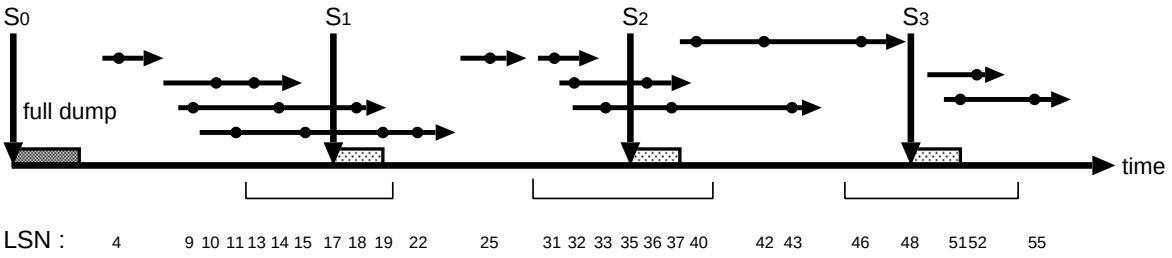


図 15　多少複雑な場合

システムではログ取得区間外にアクセスされるトランザクションの方が多いと考えられるために，さらにログは削減されるものと考えている．

以上のことから，ログの縮約による効果が十分に期待できるログ列もあるといえる．

5. ま と め

本稿では，メタデータサーバを通してメタデータサーバ外部に配置されるデータストレージに格納されたデータをアクセスするというデータ管理形態を対象に，その効率的なアーカイブとリストア機構を提案した．特に，従来のアーカイブシステムでは実現されていなかったトランザクションレベルでの一貫性を保証したアーカイブを実現した．

データベース技術を利用し，ログを取り続ける手法では保持するログ量が膨大になってしまうという問題に対して，ログ取得を一定区間に限定するようにログの縮約を行う手法を提案した．特に，その一定区間に限定した時に発生する問題を示し，データ一貫性を保証するために必要とされるログが何であるかということを，ログ取得区間，トランザクション開始，完了ポイント，及びトランザクションの更新操作ログレコードの関係から導出し示した．

さらに，スナップショットを定期的に取得しているという性質から考え，直近のスナップショットや REDO ログを利用することで，アーカイブのリストア用に保持するべきログ量を削減する手法を示した．そして，これらを 2 回の順方向ログスキャンによって実現するための手続きを示すとともに，縮約後のログ列を利用してトランザクションの境界において複数のリソースを同期した状態でリストアするための手続きも示した．これらの手続きを実装し，いくつかの具体的なログ列に対して適用することで，本研究で示した手法が利用可能であることを確認した．

今後の課題として，本稿で提案した手法が効率的に利用可能な場合や，効果が期待できないログ列がどのようなものかを同定する．また，実際の共有データ環境に利用した場合にどの程度のログ削減が期待できるかを検証する．

文　献

[1] Jim Gray, Andreas Reuter, “Transaction Processing: Concepts and Techniques”, Morgan Kaufmann 1993.

[2] Neeraj Mittal, Hui-I Hsiao, “Database Managed External File Update”, Proceedings of the 17th IEEE International Conference on Data Engineering, pp.557-564, 2001.

[3] Suparna Bhattacharya, Karen Brannon, Hui-I Hsiao, C.Mohan, Inderpal Narang, Mahadevan Subramanian, “Coordinating Backup/Recovery and Data Consistency Between Database and File Systems”, Proceedings of the 2002 ACM SIGMOD International Conference On Management of Data, pp.500-511, 2002.

[4] IBM Corporation, “IBM Storage Tank^{TM}A Distributed Storage System”, IBM White paper prepared by D.A.Pease, R.M.Rees, D.L.Plantenberg, R.A.Becker-Szendy, R.Anantha narayanan, M.Sivan-Zimet, C.J.Sullivan, R.C.Burns, D.D.E. Long, January 24, 2002.

[5] Avaki Corporation, “Avaki Data Grid 3.0 Conceptual Overview”, Avaki White paper, December, 2002.

[6] 出口　彰，“ストレージシステムにおけるデータ一貫性を保証したアーカイブとリストア機構に関する研究”，電気通信大学情報システム学研究科 修士論文，January，2004.

[7] Oracle Corporation，“Oracle Internet File System 開発者リファレンス”，2003.
http://otn.oracle.co.jp/document/products/ifs/index.html

[8] Oracle Corporation，“Oracle Collaboration Suite テクニカル・ホワイト・ペーパー”，Oracle white paper，2003.
http://otn.oracle.co.jp/products/cs/index.html

[9] Ron Weiss，“How Oracle Database 10g Revolutionizes Availability and Enables the Grid”，Oracle white paper Paper40164，2003.
http://otn.oracle.com/tech/grid/collateral/10gAvailability.pdf