データベース・システム 52－7
（１９８６・３・１０）

マルチビットマップ表現による コミットメント制御

小寺　誠、横山　恭子、吉田　誠
（沖電気工業株式会社）

１．　まえがき

ローカルネットワークの普及とワークステーションに発展により、これらを組み合せた分散処理システムの開発が盛んになってきている。このようなシステムでは、リモートからのファイルアクセス要求を処理し、ファイルスペースの共有制御をおこなう共有ファイルシステムが重要な役割を果している。共有ファイルシステムにおいては、格納されたデータのconsistency を保つことが必須の条件であり、処理のatomicity 保証の単位としてtransaction の概念を持つに至っている。

トランザクションは、処理のatomicity を保証するためにふたつの主要な制御をもっている。ひとつは、複数依頼者からの処理依頼を論理的には直列に実行することを保証する共有制御であり、いまひとつはシステムの障害に対してファイルが回復可能（recoverable ）であることを保証するコミットメント制御である[1] 。

一般にコミットメント制御の概念を持つファイルシステムでは、データ領域の拡張・削除を含む更新処理は、直ちに共有ファイル上には反映されない。つまり、トランザクションがコミットするまでは、当該のトランザクションでのデータ領域の拡張、あるいは削除処理は有効でなくするようなファイル領域管理が要求されるわけである。しかも、領域管理の部分はcritical region であることが多く、この部分でのオーバヘッドはシステム全体の処理効率を低下させる。

本稿では、Multi-Bit-Mapを用いることにより、極めて小さいオーバヘッドで上記の処理を実現する制御方式[9] 、及び当該方式の実システムへの適用に関する報告をおこなう。

２．　トランザクションモデル

本稿で用いるトランザクションの概念について概説する。

システム内には複数のトランザクション（Ｔ１…Ｔｎ）が並行して存在する。トランザクションの処理は、データのＲＥＡＤ／ＷＲＩＴＥ、ＬＯＣＫ／ＵＮＬＯＣＫ、あるいはその他の必要な計算から構成されているが、ファイル領域のコミットメント制御を論じる上で、ここではファイル領域に対するＡＬＬＯＣＡＴＥ操作とＤＥＡＬＬＯＣＡＴＥ操作に着目する。各トランザクションの処理は、開始要求（ＳＴＡＴ－ＴＲＡＮＳＡＣＴＩＯＮ）で開始され、ＡＬＬＯＣＡＴＥ／ＤＥＡＬＬＯＣＡＴＥ操作の系列によって構成される。

トランザクションの終了は、二相コミットに基づいて実行される。第一の段階としてＰＲＥＣＯＭＭＩＴが発行され、これに引き続いて終了要求（ＣＯＭＭＩＴ／ＲＯＬＬＢＡＣＫ）が発行される。ＣＯＭＭＩＴは、トランザクションの正常な終了を要求し、当該のトランザクションが発行したすべてのＡＬＬＯＣＡＴＥ／ＤＥＡＬＬＯＣＡＴＥ操作の結果を有効なものとする。一方、ＲＯＬＬＢＡＣＫは、ＰＲＥＣＯＭＭＩＴのfail、もしくはなんらかの障害が発生した場合の処理要求であり、当該のトランザクションを構成する操作系列の結果は、すべて無効となる。また、ＣＲＡＳＨをシステムの障害として定義する。ＣＲＡＳＨが発生すると、実行中のトランザクションを構成する操作系列のすべてが無効となる。

共有ファイルＦは、分割された複数個（ｍ個）の部分（ブロック）の集合から構成され、以下のように表現される。

$$F = \{ b_1 , \cdots b_m \}$$

トランザクションでのＡＬＬＯＣＡＴＥ／ＤＥＡＬＬＯＣＡＴＥ操作は、すべてこのブロックを単位として実行されるものとする。

３．コミットメント制御の状態遷移による記述

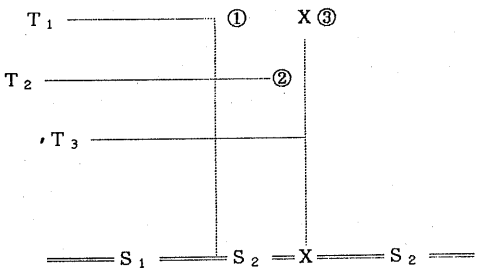
３．１　定常状態

ある時点において各ブロックは、割当て済（allocated ）、または未割当て（free）のいずれかの状態のみをとり得る。これらのブロックの状態の集合が、その時点でのファイルの定常状態であり、ブロックの状態を　ｓ（b_k）　で表すと、

$$S(F) = \{ s(b_1) , \cdots s(b_m) \}$$

として表現することができる。

ファイルの定常状態は、ＣＯＭＭＩＴ／ＲＯＬＬＢＡＣＫ／ＣＲＡＳＨのうち、ＣＯＭＭＩＴの実行によってのみ遷移し、他の処理によっては変化してはならない。この様子を示したのが図－１である。



①　トランザクションＴ$_1$のＣＯＭＭＩＴ
②　トランザクションＴ$_2$のＲＯＬＬＢＡＣＫ
③　ＣＲＡＳＨの発生

Ｓ$_1$ ～Ｓ$_2$ ：共有ファイルの定常状態

図－１：ファイルの定常状態の遷移

３．２　中間状態

定常状態においては各ファイルブロックの状態は、allocated 、または、freeのいずれかのみである。トランザクションＴｉでＡＬＬＯＣＡＴＥ操作の対象となったブロックは、ＣＯＭＭＩＴにより、freeからallocated への遷移をatomicに起こす。ＤＥＡＬＬＯＣＡＴＥ操作の場合は、この逆である。

これをさらに分解して考察する。トランザクションＴｉでＡＬＬＯＣＡＴＥ操作の対象となったブロックb_k の定常状態ｓ（b_k）は、Ｔｉが終了するまではfreeを保持しなければならない。しかし、同時にシステム内に存在する他のトランザクションＴｊにも、b_k をＡＬＬＯＣＡＴＥ操作の対象とすることを許すならば、ＴｉとＴｊのＣＯＭＭＩＴで、b_k が二重に割当てられたことになり、明らかなinconsistency が生じる。つまり、ブロックb_k は、定常状態はfreeでありながら、システム内のすべてのトランザクションから見るとallocated であり、かつトランザクションＴｉに対してallocated であるという中間状態、言いかえれば割当てられようとしている状態に遷移しなければならない。

同様にトランザクションＴｉ中でＤＥＡＬＬＯＣＡＴＥ操作の対象となったブロックb_k は、Ｔｉに対してfreeの状態となるが、ＴｉのＣＯＭＭＩＴ以前に他のトランザクションにb_k へのＡＬＬＯＣＡＴＥ操作を許すならば、ＴｉのＲＯＬＬＢＡＣＫが不可能になる。従ってb_k は、トランザクションからはfreeでありながら、一方で定常状態はallocated であるという中間状態、すなわち解放されようとしている状態に遷移することを要求される。

定常状態がfreeで、トランザクションＴｉにはallocated であるようなブロックの状態を、Ｔｉに対してintending-to-allocate 状態にあると呼び[6] 、一方定常状態がallocated でかつトランザクションＴｉにとってfreeであるようなブロックの状態を、トランザクションＴｉに対してintending-to-free である[6] とする。ただし、ＡＬＬＯＣＡＴＥ処理の対象となり得るブロックは、freeであり、かつintending-to-allocate でないものであり、ＤＥＡＬＬＯＣＡＴＥ操作の対象となり得るブロックは、allocated であり、かつintending-to-free でないものである。

以上から自明なように、ファイルＦは、定常状態Ｓ（Ｆ）のほかに、システム内の全トランザクションが認識する中間状態（intending state）Ｃ（Ｆ）と、トランザクションごとに対応した中間状態Ｉｉ（Ｆ）を持つことになる。これらの中間状態に対応するブロックの状態をそれぞれ、

ｃ（b_k）、i_i（b_k）　とすればＣ（Ｆ）とＩｉ（Ｆ）はＳ（Ｆ）と同じように以下のように表現することができる。

$$C(F) = \{c(b_1), \cdots c(b_m)\}$$

$$Ii(F) = \{i_i(b_1), \cdots i_i(b_m)\}$$

この中間状態は、ＡＬＬＯＣＡＴＥ／ＤＥＡＬＬＯＣＡＴＥ、及びＰＲＥＣＯＭＭＩＴの処理によって逐次に遷移する。Ｓ（Ｆ）は、トランザクションＴｉのＣＯＭＭＩＴで、この中間状態に依存して、次の定常状態 Ｓ’（Ｆ） へ遷移することになる（図－２参照）。

３．３　状態のMulti-Bit-Map表現

前節で示したファイルの状態の表現方法について記述する。

すでに述べたように、定常状態Ｓ（Ｆ）は、

Fを構成するブロックb_kの定常状態の集合であり、また中間状態C（F）、及びIi（F）はやはりブロックb_kの中間状態の集合であって、以下のように表現することができた。

$$S(F) = \{s(b_1), \cdots s(b_m)\}$$

$$C(F) = \{c(b_1), \cdots c(b_m)\}$$

$$Ii(F) = \{i_i(b_1), \cdots i_i(b_m)\}$$

各ブロックの定常状態はfreeであるか、またはallocated であるから、S（F）は、共有ファイルを構成するブロックの集合 $\{b_1, b_2 \cdots b_m\}$ の各要素を１ビットで表現したビットマップに置きかえることが可能である。この定常状態を表現したビットマップがＳＡＭ（Stable Allocation Map）である。

まったく同様に、中間状態も、ブロックの中間状態（intending-to-free、intending-to-alloacte）をそれぞれ１ビットで表現したビットマップに置きかえることができる。全トランザクションの認識する中間状態を表現するのがＣＯＭ（Common Map）であり、トランザクションごとに対応する中間状態を示すビットマップがＴＡＭ（Transaction Allocation Map）である。ある時点でのシステム内のトランザクションの状態は、定常状態を表現するビットマップＳＡＭと、中間状態を表現するＣＯＭ、システム中に存在するトランザクションに対応する中間状態を表現する複数のＴＡＭにより一意に表現されることになる。

３．４　状態遷移の考察

トランザクションの状態は上記のように、Multi-Bit-Mapを用いることで表現できることを示した。ＳＡＭ、ＣＯＭ、ＴＡＭはそれぞれ共有ファイルの定常状態、中間状態、トランザクションごとの中間状態を示している。一方、あるブロックに対応するこれら三種類のビットマップのビットの組み合わせは、ある時点でのブロックの遷移状態を示しているものとして捉えることができる。以下では、トランザクションＴｉ中の各処理（ＳＴＡＲＴ－ＴＲＡＮＳＡＣＴＩＯＮ、ＡＬＬＯＣＡＴＥ、ＤＥＡＬＬＯＣＡＴＥ、ＰＲＥＣＯＭＭＩＴ、ＣＯＭＭＩＴ，ＲＯＬＬＢＡＣＫ）を要因とするブロックの状態の遷移を記述する。

（１）　ＳＴＡＲＴ－ＴＲＡＮＳＡＣＴＩＯＮ

図－２に示すように、トランザクションＴｉの開始時には、中間状態Ｉｉ（Ｆ）と定常状態は一致しているものと見ることができ、ＳＴＡＲＴ－ＴＲＡＮＳＡＣＴＩＯＮでは、Ｉｉ（Ｆ）を、Ｓ（Ｆ）のcopyとして生成すればよいことがわかる。すなわち、ＳＡＭをcopyしてＴｉに対応するＴＡＭを生成すればよい。

（２）　ＡＬＬＯＣＡＴＥ操作

すでに述べたようにfreeであり、intending-to-allocateでないブロックb_kを選択し、対応するＩｉ（ｆ）の$i_i(b_k)$をintening-to-allocateとする。

（３）　ＤＥＡＬＬＯＣＡＴＥ操作

ＡＬＬＯＣＡＴＥ操作と同様に、allocatedであり、intending-to-free でないようなブロックb_kを選択し、対応するＩｉ（Ｆ）の$i_i(b_k)$をintending-to-free とする。

（４）　ＰＲＥＣＯＭＭＩＴ

ＰＲＥＣＯＭＭＩＴをＳＡＭ、ＣＯＭ、ＴＡＭの三種類のビットマップの、各ブロックに対応するビットごとに対するビットマップ演算として定義し、表－１に示す。

	ＳＡＭ	ＣＯＭ	ＴＡＭ	ＴＡＭ（遷移後）
①	0	0	0	0
②	0	0	1	0
③	0	1	0	0
④	0	1	1	1
⑤	1	0	0	0
⑥	1	0	1	0
⑦	1	1	0	0
⑧	1	1	1	1

表－１：ＰＲＥＣＯＭＭＩＴによる遷移
（ブロックb_kに対応する）

これら三種類のビットマップのビット組み合わせが意味するファイルの状態は以下のようになる。

①: b_k はfreeである。

②: b_k はＴｉ開始時にはallocated。Ｔｉ処理中にfreeへ遷移した。

③: b_k はＴｉ開始時にはfree。現在他のトランザクションにintending-to-allocate 。

④: b_k はＴｉにintending-to-allocate 。

⑤: b_k はＴｉにintending-to-free 。

⑥: b_k はＴｉ開始時にはallocated 。現在他のトランザクションにintending-to-free 。

⑦: b_k はＴｉ開始時にはfree。Ｔｉ処理中にallocated へ遷移した。

⑧: b_k はallocaterd である。

表－１に示された状態の遷移を記述する論理演算は、

ＴＡＭ＝ＣＯＭ　ａｎｄ　ＴＡＭ

と表現される。この論理式は、当該のトランザクションにintending-to-allocate であるブロックのみがprecommitting-to-allocate の状態に遷移し得ることを示している。

（５）　ＲＯＬＬＢＡＣＫ

同様に三種類のビットマップのビットに対する論理演算を定義する。

	ＳＡＭ	ＣＯＭ	ＴＡＭ	ＣＯＭ（遷移後）
①	0	0	0	0
②	0	0	1	0
③	0	1	0	1
④	0	1	1	0
⑤	1	0	0	1
⑥	1	0	1	0
⑦	1	1	0	1
⑧	1	1	1	1

表－２：ＲＯＬＬＢＡＣＫによる遷移
（ブロックb_k に対応する）

①: 表－１の①と同じ。

②: b_k は、Ｔｉの開始時にはallocated 。Ｔｉの処理中にfreeに遷移した。

③: b_k は、他のトランザクションに対してintending-to-allocate になった。

④: b_k は、Ｔｉに対してprecommitting-to-allocate 。

⑤: b_k は、Ｔｉに対してprecommitting-to-freeである。

⑥: b_k は、他のトランザクションに対して、intending-to-free である。

⑦: b_k は、Ｔｉの開始時にはfree。Ｔｉの処理中にallocated へ遷移した。

⑧: 表－１の⑧と同じ。

表－２から、状態の遷移を記述する論理演算を導くと、

$$COM = (COM \text{ and } SAM) \text{ or } \overline{(TAM \text{ and } (COM \text{ or } SAM))}$$

となる。第一項は、b_k の定常状態がallocated でかつintending-to-free でないことを示し、第二項は、b_k がＴｉによりintending-to-free となっていたことを示している。

（６）　ＣＯＭＭＩＴ

ＰＲＥＣＯＭＭＩＴと同様に三種類のビットマップ上のブロックb_k に対応するビットへの論理演算としてＣＯＭＭＩＴ処理を記述し、表－に示す。前述のように、ファイルの定常状態はＣＯＭＭＩＴによってのみ遷移するものであるから、遷移の対象となるのは定常状態を示すＳＡＭでなければならない。

	ＳＡＭ	ＣＯＭ	ＴＡＭ	ＳＡＭ（遷移後）
①	0	0	0	0
②	0	0	1	0
③	0	1	0	0
④	0	1	1	1
⑤	1	0	0	0
⑥	1	0	1	1
⑦	1	1	0	1
⑧	1	1	1	1

表－３：ＣＯＭＭＩＴによる遷移
（ブロックb_k に対応する）

①: 表－１の①と同じ。

②: b_k は、ＴｉのＰＲＥＣＯＭＭＩＴ時にはallocated 。ＴｉのＣＯＭＭＩＴ前にfreeへ遷移

した。

③: b_k は、他のトランザクションに対して intending-to-allocate になった。

④: b_j は、Ｔｉに対してprecommitting-to-allocate である。

⑤: b_k は、Ｔｉに対してprecommitting-to-freeである。

⑥: b_k は、他のトランザクションに対して、intending-to-free である。

⑦: b_k は、ＴｉのＰＲＥＣＯＭＭＩＴ時にはfree。ＴｉのＣＯＭＭＩＴ前にallocated へ遷移した。

⑧: 表－１の⑧と同じ。

表－３に示された状態の遷移を記述する論理演算は、

ＳＡＭ＝（ＴＡＭ　ａｎｄ　ＣＯＭ）　ｏｒ
（ＳＡＭ　ａｎｄ
（ＣＯＭ　ｏｒ　ＴＡＭ））

式の第一項は、ブロック b_k を、ＴｉがＡＬＬＯＣＡＴＥ操作の対象としたことを意味する。第二項は、ＴｉがＤＥＡＬＬＯＣＡＴＥ操作の対象としなかったことをそれぞれ意味している。

以上のように本項では、共有ファイルシステムにおけるトランザクションの状態を、ブロックの定常状態、中間状態、トランザクションごとの中間状態の組み合わせで表現し、それらの状態遷移に着目することで、極めてオーバヘッドの小さなコミットメント制御方式のモデルを提案した。

３．５　処理例

本稿では、分散システムの実現を意識して、二相コミットの概念を含めた処理方式の検討を行ってきた。ここで、該方式のサブセットとして、集中型の共有ファイルシステムで、ＡＬＬＯＣＡＴＥ操作のみをサポートした例を図－３に示し、以下に処理を格段階ごとに概説する。ただし、簡単のために四個のブロックの管理を行うものとする。

(step1) ‥トランザクション T_1 のＳＴＡＲＴ－ＴＲＡＮＳＡＣＴＩＯＮに伴う TAM_1 の生成

(step2) ‥ T_1 のＡＬＬＯＣＡＴＥ操作により（１個のブロックの割り当て要求）ＣＯＭと TAM_1 の対応するビットがセットされる。

(step3) ‥トランザクション T_2 のＳＴＡＲＴ－ＴＲＮＳＡＣＴＩＯＮに伴う TAM_2 の生成

(step4) ‥ T_2 のＡＬＬＯＣＡＴＥ操作により（２個のブロックの割り当て要求）ＣＯＭと TAM_2 の対応するビットがセットされる。

(step5) ‥ T_2 のＣＯＭＭＩＴ（表－３参照）

(step5) の直後にＣＲＡＳＨが発生し、ＣＯＭとＴＡＭの保持する情報が失われた場合には、ＳＡＭには T_2 がコミットした状態のみが残される。この例は、トランザクションのatomicity 保証が効果的に実現されていることを示している。

４．　実システムへの適用

著者らは以下の理念に基づき、ローカルネットワーク上に存在する各種アプリケーションから利用されるネットワーク上の大容量ファイリングシステム…ファイルサーバ…を開発した[9]（図－４。）。

■　オフィスに散在する可変長・大容量データの効率的格納、及び敏速なアクセスの実現。
■　共有利用のための基本機能実現。
■　統合環境支援。

領域管理方式の決定にあたっては、データアクセス速度、ファイルスペースの割当て・解放処理オーバヘッド、及びファイルスペース利用効率の三点を考慮しなければならない[11]。その一方、共有ファイルシステムとしての性格上、共有制御、及びコミットメント制御の概念を含んだトランザクション処理が必須の機能となり、これらの両制御をいかに効率よく実現するかの考察が必要となった。

４．１　ファイルサーバのファイル構造

ファイル構造の詳細について本稿では触れない。大雑把な構造として、システム依存のシステムデータ領域と、ユーザに利用されるユーザデータ領域とから構成され、両者がリンクされた形をとっている。

ただし、注意しておかなければならないことは、本システムがデータベース同様、トランザクションの機能を持つものの、性質は多少異なって

いる点である。通常のデータベースが定型で、比較的小さな単位のデータを扱うのに対して、ファイルサーバは、本来不定長で、かつ大規模なサイズのデータを取りあつかえることを主眼として設計されたシステムである。このために、本システムの基本的な領域管理機構として、可変長のファイル領域の処理を可能とするメカニズムが要求された。

可変長データの効率的格納と高速なアクセスを実現するため、著者らは以下に示す二点の考察のもとに、Buddy algorithm[2, 3] に基づくファイル領域管理方式…Buddy方式を採用した。

■ 高速なデータアクセス実現のためには可変長で、かつ連続的なファイルスペース割当てを可能としなければならない。Buddy algorithm は、この要求を満足する領域割当て方式を提供する。

■ 一般的にBuddy algorithmは、処理速度に優越し利用効率においては若干劣るとされる。しかし、基数をフィボナッチ数列にとるFibonacci-Buddy を用いたシミュレーションによれば、全体としてのフラグメンテーションは最悪でも30%を越えず[8] 、実用上充分な利用効率を得られる事が実証された。

4．2　コミットメント制御方式

コミットメント制御の基本的な考えかたは次のように要約することができる。

データの更新要求を直ちに実際のファイルに反映させるのではなく、一時的に別のあき領域に書き込んでおき、トランザクション処理の終了とともに実際の更新をおこなう[4] もので、この考えかたを実現する方法としては、shadow方式とlog 方式とがある。

本システムで採用したのが、shadow方式である。データは、current とshadowの二つのversion を持ち、更新処理はcurrent version にのみ反映させておき、トランザクションのコミットとともに、shadow versionを破棄することで、容易にコミットメント制御を実現することができる。

しかし、この方式は、Buddy方式に基づく動的な基本的領域管理方式と組み合せた場合には、処理時間の点でオーバヘッドを生じた。Buddy方式は、データのアクセス速度、ファイル領域の利用効率という点で満足できる方式である。しかし、shadow方式を実現するためには、更新処理時にcurrent version を新たに割当てなければならず、トランザクションコミット時にはshadow version を解放しなければならない。動的な領域管理を実現するためにBuddy方式では、ブロックの割当て・解放にともなってブロックを分割、あるいは統合しなければならず、これらの処理は二次記憶へのアクセスの頻度を高め、処理速度を低下させるという欠点を招いた。

4．3　Multi-Bit-Map方式との結合

Buddy方式に基づく動的な領域管理方式のもとでは、ブロックのサイズと個数は常に動的である。このために、このままではMulti-Bit-Map によるコミットメント制御をそのまま適用することはできない。ファイルサーバシステムの特性として、どのようなサイズのブロックがそれぞれ幾つ必要であるかを予め厳密に定めておくことができない。また、強いてそのようなことをおこなえば、大容量で可変長のデータを扱うという設計目標に反することになる。

このような問題を解決し、かつ動的な領域の割当て・解放にともなうオーバヘッドを軽減するために、Pre-allocate 方式を導入する。これは、処理要求頻度の高い特定サイズの領域を予め固定的に割り当てておく手法である[5] 。本システムでは、システム依存の領域をこのPre-allocate 方式の処理対象領域とし、Pre-allocate された当該ブロックを、本稿で述べたMulti-Bit-Mapによってコミットメント制御の対象とした。

システム依存領域を対象としたのは、この領域のサイズが固定であり、かつ最も処理の要求頻度が高かったこと。また、Buddy方式では、ブロックのサイズが小さくなるほど、割当て・解放にともなうオーバヘッドが増すという結果が得られていたことによる。

本ファイルサーバシステムには、本稿で述べた制御のサブセットを適用したのみである。しかし、本方式を適用しなかった場合に比較すると、処理効率の著しい向上が見られ、Multi-Bit-Map表現に基づくコミットメント制御方式が極めて有効なものであることを実証した。

5．　あとがき

本稿では、トランザクション状態をMulti-Bit-Map を用いて表現し、そのコミットメント制御を状態遷移を表す論理演算としてモデル化することで、オーバヘッドの少ない効率的な制御方式が実現可能であることを示した。また、実シス

テムにそのサブセットをインプリメントしたのみでも顕著な効果が期待出来ることをあわせて示した。

著者らは現在のファイルサーバシステムをマルチクライエント・マルチトランザクション処理をサポートする分散共有ファイルシステムに拡張して、Multi-Bit-Map表現によるコミットメント制御をより有効に機能させることで、負荷及び機能分散による一層の効率向上を計画している。

謝辞: 最後に、本研究をまとめるにあたり、御指導頂いた熊白部長、疋田課長ならびに関係者各位に深謝致します。

6. 参考文献

[1] Svabodoba, File Servers for Network-Based Distributed Systems, Comp. Survey, Vol. 16, No. 4, 1984

[2] Peterson, Buddy Systems, CACM V01. 20, No. 6, Jun. 1977

[3] Shen, A Weighted Buddy Method for Dynamic Storage Allocation, CACM, Vol. 17, No. 1 9, 1974

[4] Brown, The Alpine File System, ACM TOCS, Vol. 3, No. 4, Nov. 1985

[5] Morgolin, Analysis of free-storage algorithm, IBM system journal, Vol 10, No4, 1971

[6] Michell, A Comparison of Two Network-Based File Servers, CACM, Vol. 25, No. 4, Apr. 1982

[7] Lampson, Distributed Systems:Architecture and Implementation:An Advanced Cource, Springer-Verlag , 1983

[8] 水町他、大量データの格納に関するBuddy法の評価、第２９回情報処理学会全国大会

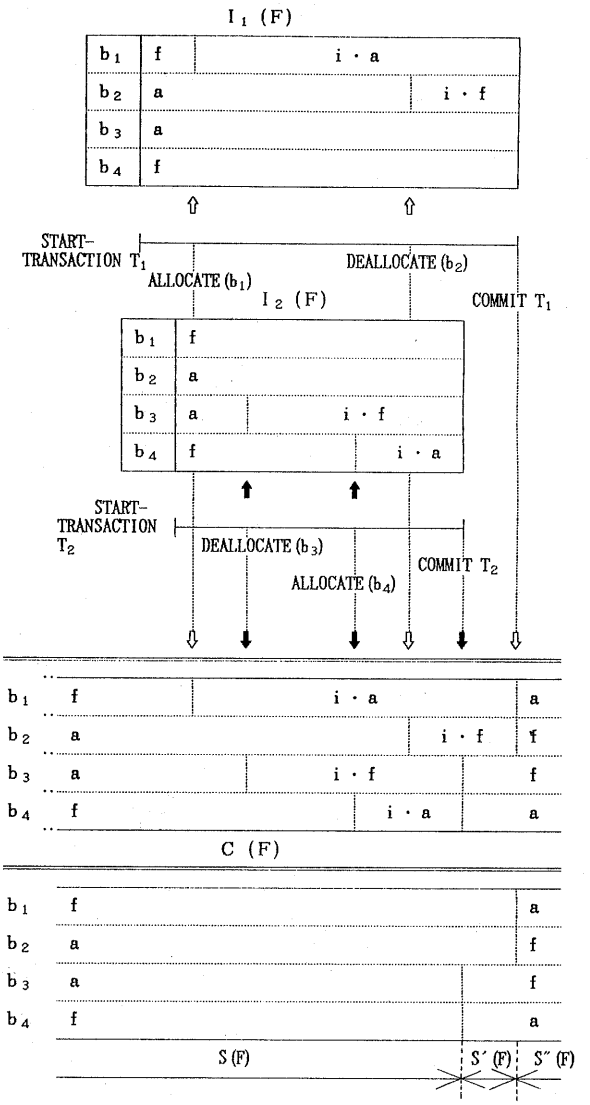
[9] 吉田他、オフィス環境におけるファイルサーバシステムの実際、マルチメディア通信と分散処理研究会（昭和６１年２月）

[10] 吉田他、ファイルサーバシステムの開発（１） 構想と構成、第３２回情報処理学会全国大会発表予定

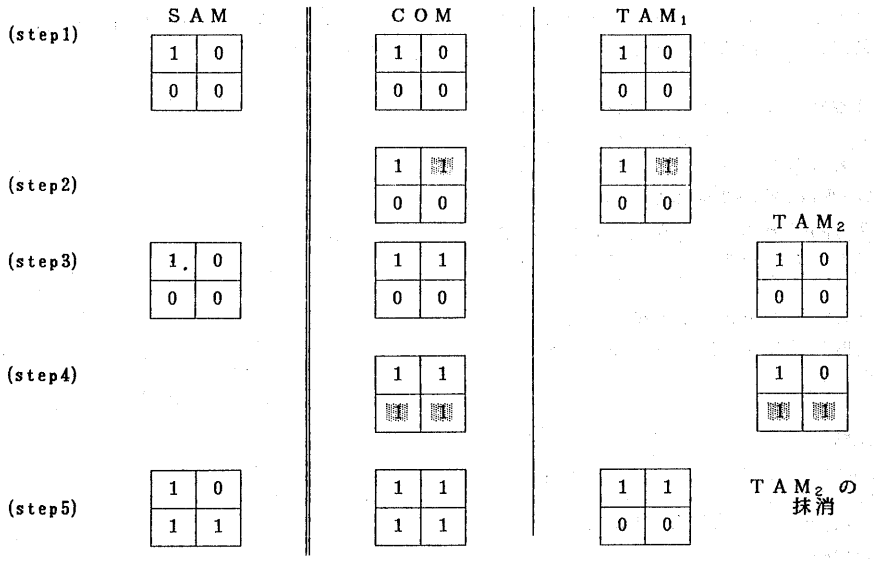
[11] 横山他、ファイルサーバシステムの開発（２） コミットメント制御方式、第３２回情報処理学会全国大会発表予定

[12] 小寺他、ファイルサーバシステムの開発（３） 共有制御方式、第３２回情報処理学会全国大会発表予定

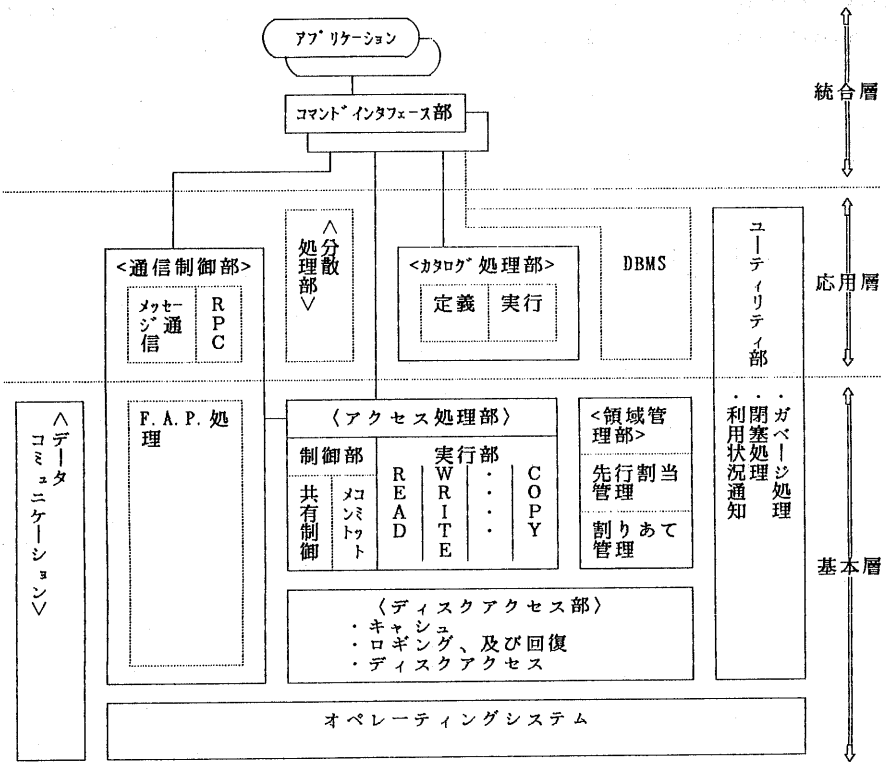
[13] 吉田他、ファイルサーバシステムの開発（４） 評価と展望、第３２回情報処理学会全国大会発表予定



図－２：定常状態と中間状態



図－３：Multi-Bit-Mapの適用例



図－４：　ファイルサーバシステムの構成