Shared Data Base に於ける障害回復モデル

日本ユニバック総合研究所研究開発部
飯塚重人・千葉恭弘

0. はじめに

データ・ベース・マネヅメント・システムにおける障害回復の処理は，共用データ・ベース（Shared data base）の要請が現実のものとなってきている現状では，必須のシステム機能となっている。近年のデータ・ベース・システムのモデル化の動向では，いわば論理的なデータ構造や関係概念の定式化にその興味の大半が占められており，障害回復の問題は形式的モデル化の背後に押しやられている感がある。一方，現実にインプリメントされている多くのシステムでは，かなり便宜的な方法で障害回復機能を実現しているのが現状であろう。

本稿では この障害回復の問題に焦点を当て，障害回復の1つの手法を形式化するモデルを構成し，回復プロセスの概念を明確にしたいと思う。このモデルは，日本ユニバック総合研究所で開発された汎用データ・ベース・システム FORIMS においてインプリメントされている。

＊ モデルのフレームワークはユニバック総研の飯塚重人研究員によって構成されたが，74年春 米国出張の折，宿泊ホテルの火災により逝去した。

1. データ・ベースの障害回復機能

データ・ベースにおける障害回復の問題は大別して次の2つのケースに分けられよう。

(1) エラー・リカバリー（error recovery）

過去のプロセスの間で，何らかのエラーが発生し，それによってその後のプロセスが影響を受け，現在のデータ・ベースが不整合の状態に陥ってしまった。不整合の状態が発生した時点が判明している場合，通常とられる回復の手法はその時点より前の最も新しいデータ・ベースのチェック・アウト・ダンプをロードし，audit file 上の after image を用いてエラー発生時直前の状態を回復する。その後は，エラーを生じたプロセスを除き，他のプロセスを再実行させる。

(2) ダウン・リカバリー（down recovery）

マシン・ダウンや実行中のプログラムの fatal なエラーのために，正常にプロセスが終了しなかった。この場合 データ・ベースは不整合の状態に陥てしまったと考え，ダウン時点で実行中であったプロセスを再実行可能な状態にするようにデータ・ベースを回復する。通常は，現状のデータ・ベースに audit file 上の before image を適用し，適当な修復時点まで戻す。

最近 Edelberg[1]は，(1)のエラー・リカバリーに関して，プロセスとブロックの結合性に基づいた1つのモデルを提示したが，これは我々のファイルの結合性に基づいたモデルと同様なアプローチである。ここでは，このEdelbergの

モデルの outline を紹介した後で，我々のモデルを提示する。なお，FORIMS では，after image を用いた restore 方式によるエラー・リカバリーと，before image を用いた roll-back 方式によるダウン・リカバリーの両者をインプリメントしているが，本稿のモデルは後者の方式に関係している。

2. Edelberg のデータ・ベース汚染伝播モデル (data base contamination)

Edelberg の multi-process データ・ベース・モデルは次のような前提を置く。

- プロセスが一度 active になると処理の終了まで実行される。
- 各プロセスは相互に独立で，active プロセスの実行スケジュールを支配するような制約はない。
- データ・ベースは幾つかのブロックから構成され，この分割は permanent とする。
- プロセスとデータ・ベース間のすべての interaction は，データ・ベース・マネジメント・システムによって処理される。

プロセスの集合を $P=\{p_1, p_2, \cdots, p_m\}$，データ・ベースを $B=\{b_1, b_2, \cdots, b_n\}$ プロセスとブロック間のデータ移送(transfer)列を $T=(\tau_1, \tau_2, \cdots, \tau_k)$ とする。データ移送にはプロセスからブロックへの移送 (e.g. 書き込み) とブロックからプロセスへの移送(e.g. 読み出し) の4つがあり，各々次のように表わす。

$\tau=(p_i, t, b_j)$ ：時間 t に於けるプロセス p_i からブロック b_j への移送
$\tau=(b_j, t, p_i)$ ：時間 t に於けるブロック b_j からプロセス p_i への移送

1つのデータ移送は開始から終了まで続けて行なわれ，分割されることはない。また1つのブロックに関するデータ移送は1時点では最高1つとするような更新のためのロック機能を前提とする。データ移送列 T は各移送の開始時間 t によって順序付けられている。

このような前提に立脚して，Edelberg はデータ・ベースにおけるエラーが伝播していく過程を定義する。

1つの移送列 $T=(\tau_1, \tau_2, \cdots, \tau_k)$ とその部分列 $T'=(\tau_1', \tau_2', \cdots, \tau_g') \subseteq T$ を考える。ここで $\tau_i=(x_i, t_i, y_i)$ $i=1,2,\cdots,g$ $(g \geq 1)$ とする。
ある移送 τ_r が他の移送 τ_s に先行 (predate) するとは次のような条件を満す T' が存在する場合である (これを $\tau_r < \tau_s$ として表わす)。

① $\tau_1'=\tau_r$ かつ $\tau_g'=\tau_s$
② $t_i < t_{i+1}$ for $1 \leq i < g$
③ $y_i = x_{i+1}$ for $1 \leq i < g$

すなわち"先行"の条件とは2つの移送の間でプロセスとブロック間に結合性が存在することである。τ_r が τ_s に先行すれば τ_s は τ_r に後続 (postdate) するという ($\tau_r < \tau_s \Rightarrow \tau_s > \tau_r$ ；$<$ は T の移送集合上での半順序関係)。
あるエラーが伝播していくような移送列 T' は必ず②と③の条件を満す。便宜的にエラーを伝播させる移送 τ_r を汚染者(contaminator)と呼ぶ。汚染者増殖の規則として次の rule をとる (ただし over estimate の可能性があるが無視する)。

④ τ_r が汚染者で $\tau_r < \tau_s$ であれば τ_s も汚染者である。今，汚染源 (contaminator-seed) の集合を S，$\bar{S}$ を S の移送とそれに後続するすべての移送 (i.e. S によって生成される $<$ の transitive closure) とする。$\bar{S}$ の移送に含まれているプロセスもしくはブロックを感染者 (contaminated) と呼ぶ。感染者のうち，$\bar{S}$ のどの移送にも先行されていないような移送に含まれるプロセスも

しくはブロックを1次感染者(contaminant)と呼ぶ。

(1) Algorithm X：汚染者追跡(Contamination Tracking)
移送列Tと汚染源Sを与え，プロセスp_i，ブロックb_jが感染者であるかどうかを判定し，ブロックが感染した時間t_jを求める。
◎移送列Tより順次に移送Tをとり出し次の判定を行う。
ⓐ $T \in S$ & $T=(p_i, t, b_j)$ の時
プロセスp_iが 非感染者であれば p_iを1次感染者とする。
ブロックb_jが 非感染者であれば b_jを感染者とし，感染時間をtとする。
ⓑ $T \in S$ & $T=(b_j, t, p_i)$ の時
プロセスp_iが 非感染者であれば p_iを感染者とする。
ブロックb_jが 非感染者であれば b_jを1次感染者とし，感染時間をtとする。
ⓒ $T \notin S$ & $T=(p_i, t, b_j)$ の時
プロセスp_iが 感染者(1次感染者も含む)でブロックb_jが非感染者であれば
ブロックb_jを感染者とし，感染時間をtとする。
ⓓ $T \notin S$ & $T=(b_j, t, p_i)$ の時
ブロックb_jが感染者(1次感染者も含む)でプロセスp_iが非感染者であれば
プロセスp_iを感染者とする。

(2) Algorithm R：汚染回復(Recovery from Contamination)
Algorithm Xによって決定されたプロセスとブロックのstatusを用いて，rerunもしくはrejectすべきプロセスの決定，修復すべきブロックとその時間を決める。なお感染していないブロックの感染時間は∞に初期化する。
◎移送列Tより順次に移送Tをとり出し次の判定を行う。
ⓐ $T=(p_i, t, b_j)$ の時
プロセスp_iが感染者で 移送の時間tがブロックb_jの感染時間より早ければ，ブロックb_jを感染者とし，感染時間を移送時間tとする。
ⓑ $T=(b_j, t, p_i)$ の時
ブロックb_jの感染時間が移送の時間tより早く，プロセスp_iが非感染者であれば，プロセスp_iを新たに感染者とし フラッグをたてる。
すべての移送が判定された時，もしフラッグがたっているならばもう1度最初のステップからくりかえす。これは判定の途中で新たにプロセスが感染者となったので 最初から伝播過程を再評価するためである。
最終的に決定したプロセスとブロックのstatusにより次のような回復手段がとられる。
◎プロセス
1次感染者 → プロセスp_iは reject
感染者 → プロセスp_iは rerun
非感染者 → プロセスp_iは normalに終了した
◎ブロック
1次感染者 → ブロックb_jは 感染時間t_jより前のイメージで restore
感染者 → ブロックb_jは 感染時間t_jのイメージで restore
非感染者 → ブロックb_jは 回復の必要なし

3. FORIMS 障害回復モデル

3.1 モデルの目的

データ・ベースへの障害が発見された際、回復の一般的な手続きとしては、
① どの時点まで回復させるか（回復時点（recovery point）の決定）。
② どんな方法で回復させるか（回復処理（recovery process）の適用）。
③ 無効となったプロセスを決める（rejected process の決定）。
④ 無効となったプロセスの再実行（rejected process rerun）。
などが考えられる。ここでは障害回復とは次の要請を満たすactionであるとする。

(1) データ・ベースを整合な状態に戻す。
(2) データ・ベースの回復時点とは、プロセスの再実行可能であるような時点である。従ってプロセス実行途中での状態は不整合であるとする。
(3) エラーによって（直接）影響されない部分は修復しない。

このような要請を満たすような障害回復の方法を検討し、回復時点の決定を定式化し、障害回復アルゴリズムを構成するのが本稿の目的である。

3.2 モデルの前提

以下の議論では、次のような要請がモデルに課せられる。

(1) プロセスは、データ・ベース・マネジメント・システム（DBMS）を介してのみデータ・ベースをアクセスできる。従ってデータ・ベースに施した修正は、すべてトランザクション・ジャーナルTJ（audit trail file）に記録される。
(2) プロセスの実行は、データ・ベースの内容によってのみ規定される。すなわち、データ・ベースを介してのみ他のプロセスと通信できる。
(3) 回復処理はエラー発生時のデータ・ベースとTJだけで行われる。
(4) 1つのプロセスは複数個のファイルを更新する。
(5) 1つのファイルは複数個のプロセスによって更新される。

データ・ベースは幾つかのファイルから構成されるものとする。ファイルの集まりを $F=\{f_1, f_2, \cdots, f_n\}$、プロセスの集まりを $P=\{p_1, p_2, \cdots, p_m\}$、時系列を $T=\{t_1, t_2, \cdots\}$ とする。モデルを構成する準備として、これらの集合上で定義される次の3つの情報を考える。

$$INT = \{\langle p, t, e\rangle \mid p\in P,\ t\in T,\ e\in\{0,1\}\}$$
$$CON = \{\langle p, f\rangle \mid p\in P,\ f\in F\}$$
$$TJ = \{\langle p, f, a, b, t\rangle \mid p\in P,\ f\in F,\ t\in T,\ a\in ADR,\ b\in V\}$$

ここでADRはfileの番地の集合、Vは値集合である。

INTは各々のプロセスの実行開始及び終了時刻の情報を伝えるもので $e=0$ の時はプロセスの開始を、$e=1$ の時は終了を意味する。CONはプロセスが更新したファイルの情報を伝える。TJはtransaction journalの項目を定義するもので、「プロセス p がファイル f の a 番地の内容 b を時刻 t に修正した」ことを意味する。この項目によってデータ・ベースを時刻 t に戻すには、ファイル f の番地 a の内容を b にすればよい。この b を 'before image' という。

これらの3種類の情報があればデータ・ベースの利用状況が再現できるが、上述の要請(3)でTJだけで障害回復を実行しなければならないので、INTと

CONをいかにTJ上に埋め込むかが問題となる。

3.3 回復時点，無効プロセス，修復ファイルの定義

回復時点（recovery point）は次のように定義される。

$RP = \{ <f, t> \mid f \in RF, t \in T \}$

ここで $RF \subseteq F$ であってRFは修復されるべきファイルである。

RP内の要素は関数的依存 functionally dependent である。すなわち

$<f, t_1>, <f, t_2> \in RP$ ならば $t_1 = t_2$

エラー発生後，修復ファイルへ作用したプロセスは無効とみなされる。この無効とすべきプロセスRJ（process to be rejected）は次のように定義される。

$RJ = \{ p \mid p \in P, \exists f \in F; <f, t_1> \in RP \ \& <p, f> \in CON \ \& <p, t_2, a> \in INT \ \& \ t_1 < t_2 \}$

すなわち，ファイルfを時刻t_1の状態に戻したので，その後の時刻に実行開始したプロセスは無効とする。

修復ファイルRF，回復時点RPおよび無効プロセスRJが与えられた時には次のような回復処理のアルゴリズムが考えられる。

(1) Transaction Journal / TJの項目を時刻tの降順に並べる。
(2) TJから修正項目$<p, f, a, b, t>$を取り出す。なければ終る。
(3) $f \notin RF$ ならば (2)へ行く。$f \in RF$であれば (4)へ進む。
(4) RPをサーチし，$<f, t'> \in RP$があった場合
　$t' < t$ ならば (7)へ行く
(5) $RF = RF - \{f\}$
(6) $RF = \emptyset$ ならば 終る。そうでなければ (2)へ行く。
(7) ファイルfの番地aの内容をbにする。 (2)へ行く。

実際には，上記のアルゴリズムで段階(1)は不要である。なんとなれば，TJの修正項目は時刻tに従って作成されるので，TJを順編成ファイルとし，段階(2)で逆読みすることにより(1)を省略することができる。

問題は このアルゴリズムではRF，RP，RJをgivenとしている点であるが，実際には，この決定メカニズムが重要である。整合的な回復処理を構成するために，RP，RF，RJの決定を定式化し，回復処理の中にその決定を組み込むことを検討する。

3.4 修復ファイル，回復時点，無効プロセスの決定

修復ファイルRF，回復時点RP，無効プロセスRJの決定について幾つかの方法が考えられる。ここでは3つの方法を提示し，それぞれの問題点を検討する。

(1) プロセス中断型

- 処理が完了しないうちに実行が中断されたプロセスは無効プロセスである。
- 無効プロセスが更新していたファイルは修復ファイルである。
- ファイルfの回復時点は，そのファイルを更新している無効プロセスの開始時間のうち最も早い時刻とする。

これらの定義は次のように記述される。

$RJ1 = \{p \in P \mid \exists <p,t,\emptyset> \in INT \ \& \ \neg\exists <p,t',1> \in INT\}$
$RF1 = \{f \in F \mid \exists p \in RJ1 ; <p,f> \in CON\}$
$RP1 = \{<f,t> \mid f \in RF1, \exists p \in RJ1 ;$
$(<p,f>, <p',f> \in CON \ \& \ <p,t,\emptyset>, <p',t',\emptyset> \in INT) \rightarrow t < t'\}$

この決定方法は 複数プロセスのデータ ベース同時共同利用という状況においては難点が発生する。たとえば 2つのプロセス p_1, p_2 が ファイル f を共同利用していて、その処理時間が重なっていた場合、p_1 が正常に終了した後で、p_2 がエラーとなり中断されてしまったとする。決定方法(1)による無効プロセスは p_2, 修復ファイルは f, 回復時点は p_2 の開始時点となる。しかし p_1 と p_2 の処理が重なっていた時間に於ける p_1 の効果はこの回復処理によってすべて打ち消されてしまうことになる。この点を改良する次の方法が考えられる。

(II) 無効プロセス波及型

- 処理が完了しないプロセス及びそれによって更新されているファイルは回復の対象となる。 ····· (I)と同じ
- 修復ファイルを通じて無効プロセスと時間的に結合しているプロセスも無効プロセスとする。 ····· 改良点
- ファイル f の修復時点は、そのファイルを更新している無効プロセスの開始時間のうち最も早い時刻とする。 ···· (I)と同じ

これらの定義は次のように記述される。

$RJ2^{(0)} = RJ1$
$RF2 = RF1$
$RJ2^{(i)} = \{p \in P \mid \exists f \in RF2, \exists p' \in RJ2^{(i-1)}$
$(<p,f>, <p',f> \in CON \ \& <p,t,1>, <p',t',\emptyset> \in INT) \rightarrow t' < t\}$
$RJ2 = \bigcup_i RJ2^{(i)}$

$RP2 = \{<f,t> \mid f \in RF2, \exists p \in RJ2 ;$
$(<p,f>, <p',f> \in CON \ \& <p,t,\emptyset>, <p',t',\emptyset> \in INT) \rightarrow t < t'\}$

この決定方法にも問題が発生する。すなわち1つの無効プロセスが複数個のファイルを更新していた場合、片方だけが修復され、他は修復の対象とならないケースが発生する。これは p が $RJ1$ の元であれば、問題とはならないが、$p \in RJ2 - RJ1$ である場合に発生する可能性がある。実際には 1つのプロセスが複数のファイルを更新していた場合、各々のファイル処理は独立である場合もありえよう。しかしこの依存関係の有無の確定にはプロセスの内容自体を検討して reference 関係を確認しなければならないという困難さを生じる。従って我々はモデルに課された要請を追加して(6)として次のものを掲げよう。

(6) 1つのプロセスで処理される複数のファイルの内容は結果的に互の内容に依存しあっている。

この要請の追加により、1つの無効プロセスが複数ファイルを更新している場合には、そのプロセスで更新しているすべてのファイルが修復されなければならない。この点を考慮する次の方法が規定される。

(III) 修復ファイル波及型

新たに設けられた要請(6)により修復ファイル，無効プロセス，回復時点は次のように定義される。

$$RJ3^{(0)} = RJ1$$
$$RF3^{(0)} = RF1$$
$$RJ3^{(i)} = \{p \in P \mid \exists f \in RF3^{(i-1)}, \exists p' \in RJ3^{(i-1)};\ (\langle p,f\rangle, \langle p',f\rangle \in CON \ \&\ \langle p,t,1\rangle, \langle p';t';\emptyset\rangle \in INT) \rightarrow t' < t\}$$
$$RF3^{(i)} = \{f \in F \mid \exists p \in RJ3^{(i)};\ \langle p,f\rangle \in CON\}$$

$$RJ3 = \bigcup_i RJ3^{(i)}$$
$$RF3 = \bigcup_i RF3^{(i)}$$

$$RP3 = \{\langle f,t\rangle \mid f \in RF3,\ \exists p \in RJ3;\ (\langle p,f\rangle, \langle p',f\rangle \in CON\ \&\ \langle p,t,\emptyset\rangle, \langle p';t';\emptyset\rangle \in INT) \rightarrow t < t'\}$$

このようにして得られたRJ3, RF3, RP3はすでにあげたすべての要請を満たしている。

3.5 障害回復アルゴリズム

前節で無効プロセス，修復ファイル及び回復時点の決定が定式化された。ファイルfが修復ファイルであるか否かは，回復処理の過程で，そのファイルへの最初の修正項目$\langle p,f,a,b,t\rangle$が得られる直前までに分ればよい。一方，プロセスp，ファイルfがRJ，RFに入るか否かは，CONおよびINTに依るのであるが，これらの情報が具体的に得られるのは，プロセスの開始時刻あるいは終了時刻である。つまりその時刻にだけ注目すればよいことが分る。TJに最後に記録されたプロセス終了時刻をt_0とすると，TJの逆読みによって，次の3つに場合分けができる。

(1) TJ項目$\langle p,f,a,b,t\rangle$で $t_0 < t$ の時
$p \in RJ1$（未終了プロセス）$\subseteq RJ3$ 故に修復項目である。

(2) $t_0 = t$ の時
t_0において終了していないプロセスはすべて無効プロセスである。
t_0で終了したプロセスp_0が無効であるか否かは，すでに定まった無効プロセスのうち，t_0で実行中のプロセスと更新ファイルを共用しているか否かによる。（$RJ3^{(i)}$の定義より）

(3) $t_0 > t$ の時
tで終了したプロセスp_iがある場合，この有効性は(2)と同様に，すでに確定した無効プロセスがtの時点でp_iと更新ファイルを共用しているか否かに依存する。

ここで実際のTJの設計に有効となる修復類の概念を導入する。時刻tにおいて，プロセスpが更新しているファイル群を考慮して次のように定義する。

$$f \approx_t g \iff \exists p \in P;\ \langle p,f\rangle, \langle p,g\rangle \in CON\ \&\ \langle p,t_0,\emptyset\rangle, \langle p,t_1,1\rangle \in INT \rightarrow t_0 < t \le t_1$$

関係$\approx_t$を用い，ファイルの集合Fの中での類列を考える。

$$f \sim_t g \iff f \approx_t g \ \text{or}\ \exists h \in F;\ f \approx_t h\ \&\ g \approx_t h$$

これは時刻tにおいて，何らかのプロセスを介して結ばれているファイルの集

まりを示している。関係 $\sim_t$ は 反射，対称，推移律を満たしているので同値関係であり，これによりFは素な同値類に分割される。ファイル f が属する類を g_t で表わす。 時刻 t においてfが修復されるならば，$g(\sim_t f)$ もまた修復されなければならない。とくに時刻 t において修復されるファイルの類を修復類と呼ぶ。

さて，相前後する2つのプロセス終了点 t_1, t_0 ($t_1 < t_0$) を考える。t_1 における修復ファイルを RF_1，t_0 におけるそれを RF_0 とすると，ファイル f が t_1 での修復ファイルであるためには，次の条件が満たされなければならない。すなわち

$$f \in RF_0 \quad or \quad \exists p \in P[t_1], \exists g \in RF_0 ; \langle p, f\rangle, \langle p, g\rangle \in CON$$

ここで $P[t_1]$ とは t_1 で実行中のすべてのプロセスを表わす。

この条件は，各プロセス終了時毎にファイルの類別（同値関係 $\sim_t$ による）を与えておけば，無効プロセスがどれであるかを知らなくても，順次に修復ファイルを決定できることを示している。TJ作成時点ではどのプロセスの終了時刻 t_i が最後終了時刻 t_0 となるかは分らないので，各時刻 t_i 毎にその時刻における実行プロセス名を記録する必要がある。（修復の目的からは上述のファイルの類別の際実行プロセスが関与しているか否かの情報を類毎に与えておけばよい。）

ファイルの類別におけるファイル間の連結性に着目して，各類を germ（芽ばえ）と名付ける。1つのファイルが何らかのプロセスで更新されると，そのファイルは1つのgermに所属する。ファイル f_1 が他のファイル f_2 と共に1つのプログラムで更新され，f_2 がすでに何らかのgermに属していると，f_1 もそのgermに属する。germの誕生日は，そこに属するファイルの処理開始された時刻のうち最も早いものとして定義され，各germは所属するファイルを更新するプロセスが1つもなくなった時点で消滅する。従ってエラー発生による修復ファイルは，エラー発生時に存在していたgermの要素であり，各germはその誕生日まで roll-back される必要がある。ファイルの類別に基づいたこの障害回復手法を GERM (General Erorr Recovery Model) と名付ける。

3.6 GERM method

INTとCONをTJの中に埋め込むために，次の3つのTJ項目が作成される。

(a) プロセス開始項目 (PSI) $= \langle \emptyset, p, t, DUM(t), GERMS\rangle$

$\emptyset$：PSIタイプ指示，p：プロセス名，t：プロセス開始時刻

DUM(t)：時刻 t でどのgermにも属さないファイル名，GERMS：時刻 t におけるすべてのgerm

ここで $GERMS = \{G_i(t)\}$, $i \neq j \rightarrow G_i(t) \cap G_j(t) = \emptyset$, $\cup G_i(t) \cup DUM(t) = F$

(b) プロセス終了項目 (PEI) $= \langle 1, p, t, GERMS\rangle$

1：PEIタイプ指示，t：プロセス終了時刻

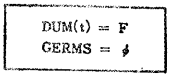
(c) before-image 項目 (BII) $= \langle 2, p, f, a, b, t\rangle$

2：BIIタイプ指示，f：更新ファイル名，a：ファイル内の番地，b：a の before-image

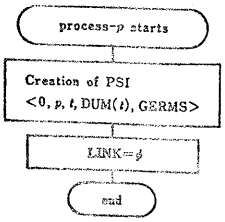
t：更新時刻

これらの項目は次のようなタイミングで作成される。PSIは任意のプロセスが処理を開始した時（データ・ベースの open 時点），PEIはプロセスが処理を終了した時（データ・ベースの close 時点），BIIはプロセスがファイルを更新した時点でシステムの logging module により作成される。各時点におけるDUMとGERMSは，multi-thread の状況でシステムによって maintenance される。これらの項目の作成と保守は次のコントロール・フローによって示される。

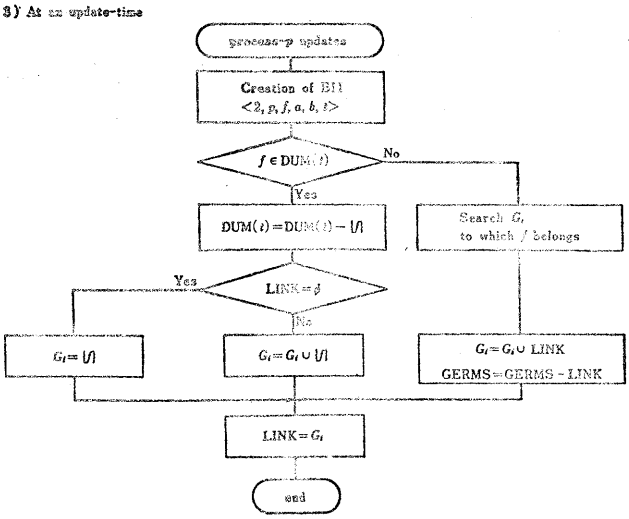
(1) At the System Generation Time (DB Load)



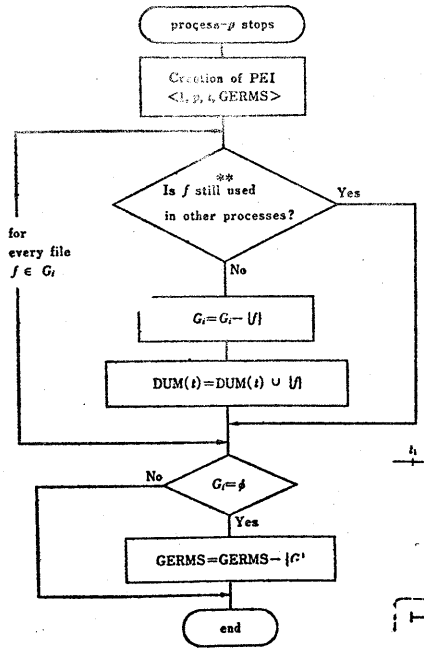
(2) At a process-starting-time



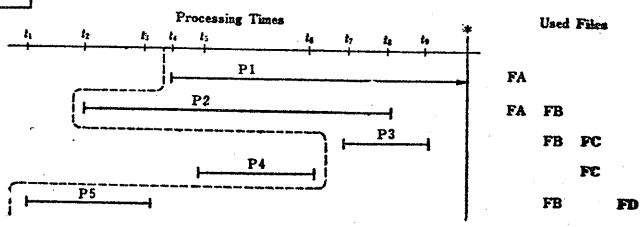
(3) At an update-time



(4) At a process-ending-time



サンプル・ケース



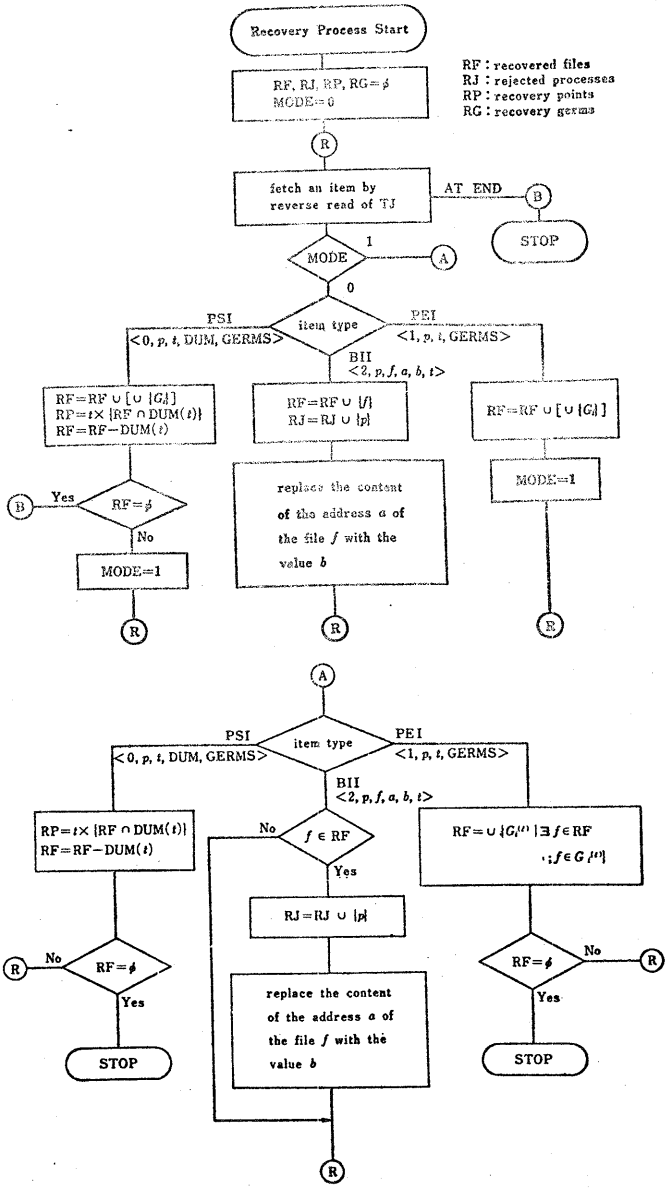
or this check, it may be necessary to prepare a reference counter for each file which designates how many processes refer to it.

time / initial	action / DB Load	TJ-item	DUM(t) FA, FB, FC, FD	GERMS 0
t_1	P5 open DB	<0, P5, t_1, {FA, FB, FC, FD}, 0>		
t_1'	P5 update FB	<2, P5, FB, fb_1, b_1, t_1'>	FA FC FD	G_1={FB}
t_2	P2 open DB	<0, P2, t_2, {FA, FC, FD}, G_1>		
t_2'	P5 update FD	<2, P5, FD, fd_1, b_2, t_2'>	FA FC	G_1=FB, FD}
t_2''	P2 update FB	<2, P2, FB, fb_2, b_2, t_2''>	FA FC	G_1={FB, FD}
t_3	P5 close DB	<1, P5, t_3, G_1>	FA FC FD	G_1={FB}
t_3'	P2 update FA	<2, P2, FA, fa_1, b_4, t_3'>	FC FD	G_1={FA, FB}
t_4	P1 open DB	<0, P1, t_4, {FC, FD}, G_1>		
t_5	P4 open DB	<0, P4, t_5, {FC, FD}, G_1>		
t_5'	P4 update FC	<2, P4, FC, fc_1, b_5, t_5'>	FD	G_1={FA, FB} G_2={FC}
t_5''	P1 update FA	<2, P1, FA, fa_2, b_6, t_5''>	FD	G_1={FA, FB} G_2={FC}
t_6	P4 close BB	<1, P4, t_6, {G_1, G_2}>	FC FD	G_1={FA, FB}
t_7	P3 open DB	<0, P3, t_7, {FC, FD}, G_1>		
t_7'	P3 update FC	<2, P3, FC, fc_2, b_7, t_7'>	FD	G_1={FA, FB} G_2={EC}
t_7''	P3 update FB	<2, P3, FB, fb_3, b_8, t_7''>	FD	G_1=G_1∪G_2={FA, FB, FC}
t_8	P2 close DB	<1, P2, t_8, G_1>	FD	G_1={FA, FB, FC}
t_9	P3 close DB	<1, P3, t_9, G_1>	FB FC FD	G_1={FA}
t_9'	P1 update FA	<2, P1, FA, fa_3, b_9, t_9'>	FB FC FD	G_1={FA}
t^*	Error occurred			

図1 障害回復プロセス

回復プロセスは次のフローで示される。

またこのアルゴリズムをサンプル・ケースに適用した際のTJ項目の作成やGERMSの推移は，前頁の図1に示される。



4. 結語

我々の障害回復の方法は，時間的に結合される複数プロセスによって共有されている更新ファイルを修復類に類別したGermの概念に立脚している。一方，Edelbergのモデルは，過去の汚染源からプロセスとブロックの結合性を通じて汚染伝播を追跡するというアプローチをとる。Germの増殖の過程と，感染者の増殖の過程が同じ性質を持つものと思える。FORIMSではTJの量の問題から，log情報には読み出しの情報を含んでいないが，過重回復を減少させるためにはこの点の検討が必要であろう。

参考文献

[1] Edelberg, M., "Data Base Contamination and Recovery", Proc. ACM-SIGFIDET, 197