分散システム 第10-11回
― フォールトトレラント性 ―

大連理工大学・立命館大学 国際情報ソフトウェア学部

大森 隆行

講義内容

- フォールトトレラント性
  - 概説
    - 基本概念
    - 障害モデル
  - 冗長性による障害の隠蔽
    - プロセス多重化
    - ビザンチン将軍問題
  - 高信頼クライアントサーバ間通信
  - 分散コミット
  - エラーからの回復
    - 安定ストレージ
    - チェックポイント作り

フォールトトレラント性

- [JIS-X0014:1999]によるフォールトトレランス(fault tolerance)の定義
  - 障害または誤りが存在しても、要求された機能を遂行し続けることのできる、機能単位の能力
  - フォールトトレランスを達成するための性質
    → フォールトトレラント性、耐故障性
- 分散システムでは部分的に障害が発生する
  - 単一システムの障害は全体をダウンさせることが多い
  - いかにシステムを動かしたまま障害から回復させるか(→障害透過性)
- 故障に強いシステム構築のために
  - プロセスの回復力(resilience) - プロセスの多重化
  - 高信頼クライアントサーバ間通信
  - 分散コミット
  - エラーからの回復

フォールトトレラント性

- フォールトトレラント性を支える性質
  - 可用性(availability)
    - 利用しようとした瞬間に利用できるか
  - 信頼性(reliability)
    - 障害を起こすことなく実行し続けられるか
  - 安全性(safety)
    - システムが正常に稼働しなくても重大な問題が生じないか
      - 原発やロケットの制御システム等では特に高い安全性が必要
  - 保守性(maintainability)
    - 故障からの回復が容易か
      - 可用性にも影響

フォールトトレラント性

- 可用性と信頼性の指標
  - MTTR(Mean Time To Recovery)
    - 平均修復時間(修復に要する時間の平均)
    - MTTRが小さいほど可用性が高い
  - MTBF(Mean Time Between Failures)
    - 平均故障間隔
    - MTBFが大きいほど信頼性が高い
- (例1) 1時間に1回1ミリ秒ダウン
  - 可用性: 1-(0.001/3600)=99.9999%以上
  - 信頼性(MTBF): 約1時間
- (例2) 1年のうち1回2週間ダウン
  - 可用性: 1-(14/365)=約96%
  - 信頼性(MTBF): 351日

MTTR=故障時間/故障回数

MTBF=稼働時間/故障回数

可用性(%)=稼働時間/総時間

フォールトトレラント性

- 基本概念 (cf. JIS X0014-1999)

※これらの用語の
定義は本により様々

  - 誤り (error)
    - 計算、観測もしくは測定された値または状態と、
      真の、指定されたもしくは論理的に正しい値または状態との間の相違
      → 外部的な振る舞いとして、正しく動かないこと
  - 故障 (failure)
    - 要求された機能を遂行する、機能単位の能力がなくなること
      → 外部的な振る舞いとして、正しく動かなくなること
  - 障害 (fault)、欠陥(defect)、バグ(bug)
    - 要求された機能を遂行する機能単位の能力の、
      縮退または喪失を引き起こす、異常な状態
      → 障害は内部的に間違いがあるという状態
      → 障害が原因で故障や誤り(外部的な振る舞いとして観測可能な状況)が発生

Fault tolerant = 障害に耐えられる

→ 障害があったとしても、いかに故障や誤りを防止できるか

障害(fault)

障害の種類	**説明**
クラッシュ障害	サーバの停止。直前まで正常に動作
欠落障害 　受信欠落 　送信欠落	サーバが要求の応答に失敗 サーバがメッセージの受信に失敗 サーバがメッセージの送信に失敗
タイミング障害	サーバが指定時間内に応答できない
応答障害 　値障害 　状態遷移障害	サーバの応答が不正確 応答の値が不正確 サーバが正しい制御の流れから逸脱
任意障害 (ビザンチン障害)	サーバが任意の時間に任意の応答を生成

障害

- クラッシュ障害
  - ダウンしたサーバにはそれ以上問い合わせできない
  - 解消するにはリブートするしかない
- 欠落障害(オミッション障害)
  - 受信欠落障害→サーバが要求を受けられない
  - 送信バッファあふれ→送信欠落障害
  - サーバソフトの問題→ハング(hang)

障害

- タイミング障害
  - データの供給が早すぎて処理しきれない
  - 一般的には送信サーバ側のレスポンスの方がずっと遅いので、性能障害の方が発生しやすい
- 応答障害
  - 値障害　(例) 検索サーバが検索キーワードと関係ないページを返す
  - 処理が規定されていないメッセージを受信
→状態遷移障害
- 任意障害(ビザンチン障害)
  - 任意のタイミングに任意の値を返答。
それが正しいかどうか判別が困難

確認問題

- 以下の説明に合う語句を答えよ。
  - (1) 障害が発生しても機能を遂行し続けられる性質
  - (2) 障害が発生してもユーザにそれを気付かせないという性質
- (1)を支える性質についての説明である。
  それぞれの説明に合うものを語群から選んで答えよ。
  - システムが正常に稼働しなくても重大な問題が生じないか
  - 故障からの回復が容易か
  - 利用しようとした瞬間に利用できるか
  - 障害を起こすことなく実行し続けられるか

語群：可用性、信頼性、安全性、保守性

- 次の説明に合う障害の種類はどれか、最も適切なものを語群から選んで答えよ。
  - サーバが必要な応答に失敗した。
  - サーバがプログラムのエラーによりダウンした。
  - あるプロセスが任意のタイミングにランダムな値を返すようになった。

語群：クラッシュ障害、欠落障害、タイミング障害、応答障害、任意障害

確認問題

- 1年間(8760時間)に3回(合計12時間)故障したシステムがあるとする。
  このシステムのこの期間の以下の指標の値をそれぞれ計算せよ。
  - 稼働時間
  - MTTR
  - MTBF
  - 可用性(稼働率)

講義内容

- フォールトトレラント性
  - 概説
    - 基本概念
    - 障害モデル
  - 冗長性による障害の隠蔽
    - プロセス多重化
    - ビザンチン将軍問題
  - 高信頼クライアントサーバ間通信
  - 分散コミット
  - エラーからの回復
    - 安定ストレージ
    - チェックポイント作り

冗長性による障害の隠蔽

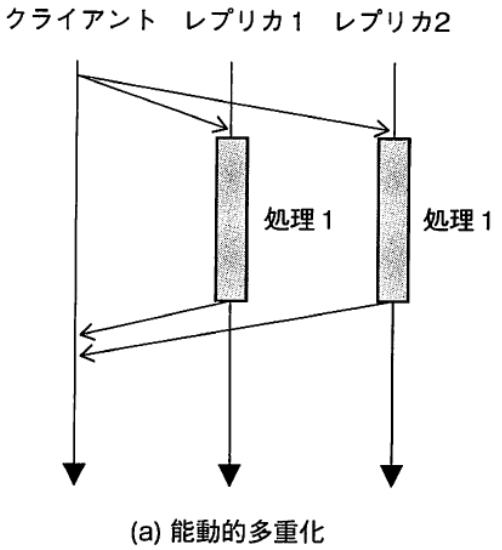
- 情報冗長性
  - 間違ったビットを修復するために余分なビットを付加
  - (例) ハミング符号
- 時間冗長性
  - 必要であれば動作をやり直す
  - (例) トランザクション処理
- 物理冗長性
  - 装置やプロセスを多重化(ソフトウェアでも行われる)
    - 一つが故障しても他でサービスを提供可能かどうか
    - (例) RAID (Redundant Array of Inexpensive Disks)

これらにより、高い回復力(resilience)を実現

プロセス多重化

- 能動的多重化
  - クライアントからのメッセージは全レプリカに送付
    - プロセスの状態変更がクライアントからのメッセージによって決まることが必要
    - メッセージに基づく命令の実行について、全順序性と原子性が満足される必要

複数のレプリカを一つのグループ(group)にしてそのメンバに要求送信

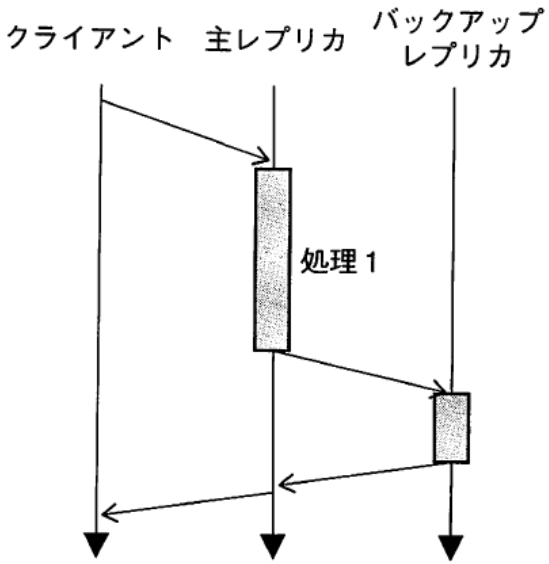


(a) 能動的多重化

    - 全順序性：すべてのレプリカが同じ命令を同じ順序で実行するという性質
    - 原子性：命令が実行される場合には、すべての正常なレプリカで実行されるという性質

プロセス多重化

- 受動的多重化
  - クライアントからのメッセージは主レプリカが処理し、他のレプリカはそのバックアップを行う(プライマリバックアップアプローチ)
    - 主レプリカの故障に弱いが、バックアップが故障してもサービスに影響を与えない



(b) 受動的多重化

プロセス間の合意

- 合意問題(agreement problem)
  - 多重化したプロセス間で異なる結果が出る場合にいかに合意を得るか
  - プロセスが故障→通常とは違う結果
    複数故障すると…
- 合意が必要な例
  - コーディネータの選出、分散トランザクションをコミットするかどうか、タスクの配分方法 等
- 分散合意アルゴリズム
  - 故障していないプロセス同士が、ある課題に対して有限回のステップで合意に至る問題を扱う

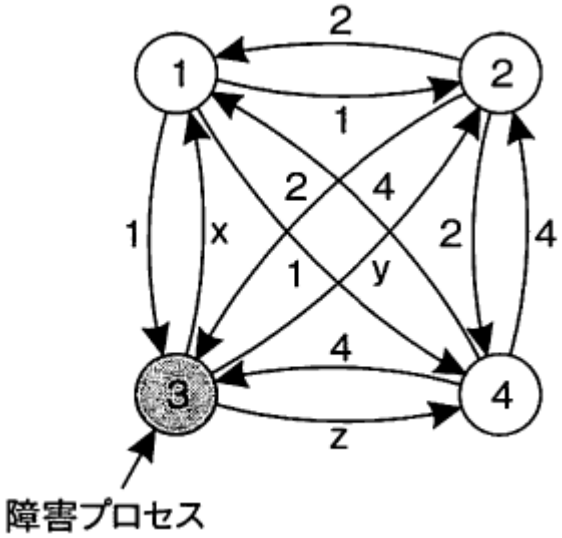
障害システムにおける合意
- ビザンチン将軍問題 -

- 前提
  - 通信は完全であるが，プロセスは完全でない
  - 同盟軍はn人の将軍が軍を率いており、
    このうち、m人は裏切り者である
  - 将軍は自分の軍の人数を知っており、
    他の軍の将軍に伝えることができる
  - 裏切り者の将軍は嘘を言うかも知れない

→裏切り者以外の将軍は他の軍の人数を合意できるか？

障害システムにおける合意
- ビザンチン将軍問題 -

- STEP1: 各将軍は自分の軍隊の人数を他に知らせる　　将軍→プロセス
- STEP2: 各将軍からの結果をベクトル形式でまとめる
- STEP3: STEP2で得たベクトルを他の将軍に知らせる
- STEP4: 受け取ったベクトルのi番目を検査し、過半数であれば値を採用



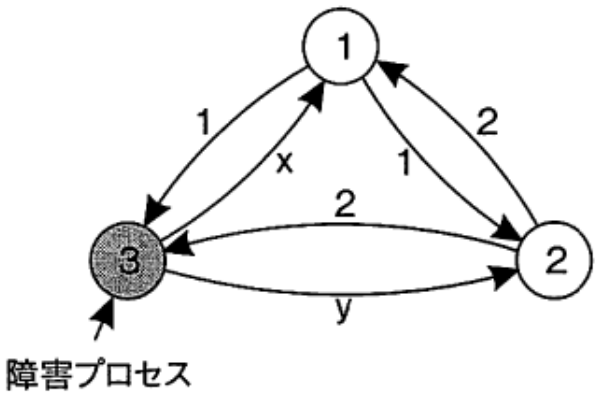
STEP2
プロセス1は(1,2,x,4)を獲得
プロセス2は(1,2,y,4)を獲得
プロセス3は(1,2,3,4)を獲得
プロセス4は(1,2,z,4)を獲得

STEP3
プロセス1　プロセス2　プロセス4
(1,2,y,4) (1,2,x,4) (1,2,x,4)
(a,b,c,d) (e,f,g,h) (1,2,y,4)
(1,2,z,4) (1,2,z,4) (i,j,k,l)

プロセス1,2,4に
関しては過半数一致
→合意が取れた

障害システムにおける合意
- ビザンチン将軍問題 -

- m個の障害プロセスを持つシステムにおいて、
  合意には2m+1個の正しく機能するプロセスが必要
  （Lamport, 1982）



STEP2
プロセス1は(1,2,x)を獲得
プロセス2は(1,2,y)を獲得
プロセス3は(1,2,3)を獲得

STEP3
プロセス1　プロセス2
(1,2,y)　(1,2,x)
(a,b,c)　(d,e,f)

合意できない

確認問題

- 以下の各文に最も関係する冗長性の種類を選択肢から選んで答えよ。
  - サーバの応答がなかったため、要求を繰り返し送信する。
  - 間違ったビットの修復のためにハミング符号を追加する。
  - ハードディスク故障に備えて、RAIDを組む。

    選択肢：情報冗長性、時間冗長性、物理冗長性
- ビザンチン将軍問題において、1つの障害プロセスがあるとき、プロセス間で合意に達するためには全体で最低いくつのプロセスが必要か。

講義内容

- フォールトトレラント性
  - 概説
    - 基本概念
    - 障害モデル
  - 冗長性による障害の隠蔽
    - プロセス多重化
    - ビザンチン将軍問題
  - 高信頼クライアントサーバ間通信
  - 分散コミット
  - エラーからの回復
    - 安定ストレージ
    - チェックポイント作り

高信頼クライアントサーバ間通信

- 通信路についても、プロセスと同様の障害モデルを適用可能
  - クラッシュ障害、欠落障害、タイミング障害、ビザンチン障害
- 高信頼point-to-point通信の例：TCP
では、これらの障害に対する隠蔽が図られる
  - シーケンス番号を含む確認応答と
それに基づく再送制御で欠落障害に対処
  - コネクションのクラッシュ障害
(＝TCPコネクションの突然の切断)に
対してはTCP内部だけでの対応は困難

RPCにおける障害隠蔽

高レベルな通信の例として
ここではRPCに着目

- 障害が起こりうるケース
  - クライアントがサーバの位置を特定できない
  - クライアントからサーバへの要求メッセージが喪失
  - サーバが要求を受けた後にクラッシュ
  - サーバからクライアントへの応答メッセージが喪失
  - クライアントで要求メッセージ送信後に障害が発生

RPCにおける障害隠蔽

- クライアントがサーバの位置を特定できない
  - 例外(exception)を発生
    → 透過性は失われる
- サーバへの要求メッセージが喪失
  - ACK(応答・確認通知)が返るまでに一定時間が過ぎたら再送処理を行う
- サーバがクラッシュ
  - クライアント側からはメッセージ喪失と区別不可能
  - 最低1回セマンティクス：サーバの再起動を待ち、処理をやり直す。複数回実行されることもある
  - 最大1回セマンティクス：すぐに障害の発生を通知(実際は遅れているだけかもしれない)
  - 何の保証もしない(実装は一番楽)

(次頁に続く)

RPCにおける障害隠蔽

- サーバがクラッシュ(続き)
  - 厳密1回セマンティクス：RPCの実行を必ず1回だけ保証する
    → 実現のためのクライアント側の方策は存在しない
- 応答メッセージの喪失
  - 再送処理が可能
  - 何度も実行するとまずい場合
    (繰り返し等価(冪等, idempotent)でない場合)
    → 何回目の要求かを示すビットを要求メッセージに組み込む等の対策

RPCにおける障害隠蔽

orphan[N] 孤児

- クライアントのクラッシュ
  - 要求がオーファン(orphan)になる
    - 要求を出したクライアントはクラッシュ
  - 共有ファイルをロックしたままorphanになると
    - オーファン駆除：要求のログをクライアント側で残しておき、再起動後にログをチェックしてオーファンを処分
    - 一定時間内の処理が完了しなければ時間切れにする
  - 上記のような対応をしても、
    - (RPCの管理のための)ディスクへの負担が大きい
    - オーファン自身が新たな追跡困難なRPC (孫オーファン)を発行
    - オーファンの突然の削除→ファイルのロックが残ることがある

  等の問題がある

確認問題

- RPCにおいてサーバで障害があった際、クライアントが「再起動を待って再試行する」という方策をとった場合、どのようなセマンティクスが実現されるか。以下の選択肢から選んで答えよ。

  - a. 最低1回セマンティクス
  - b. 最大1回セマンティクス
  - c. 厳密1回セマンティクス
  - d. 何の保証もなし

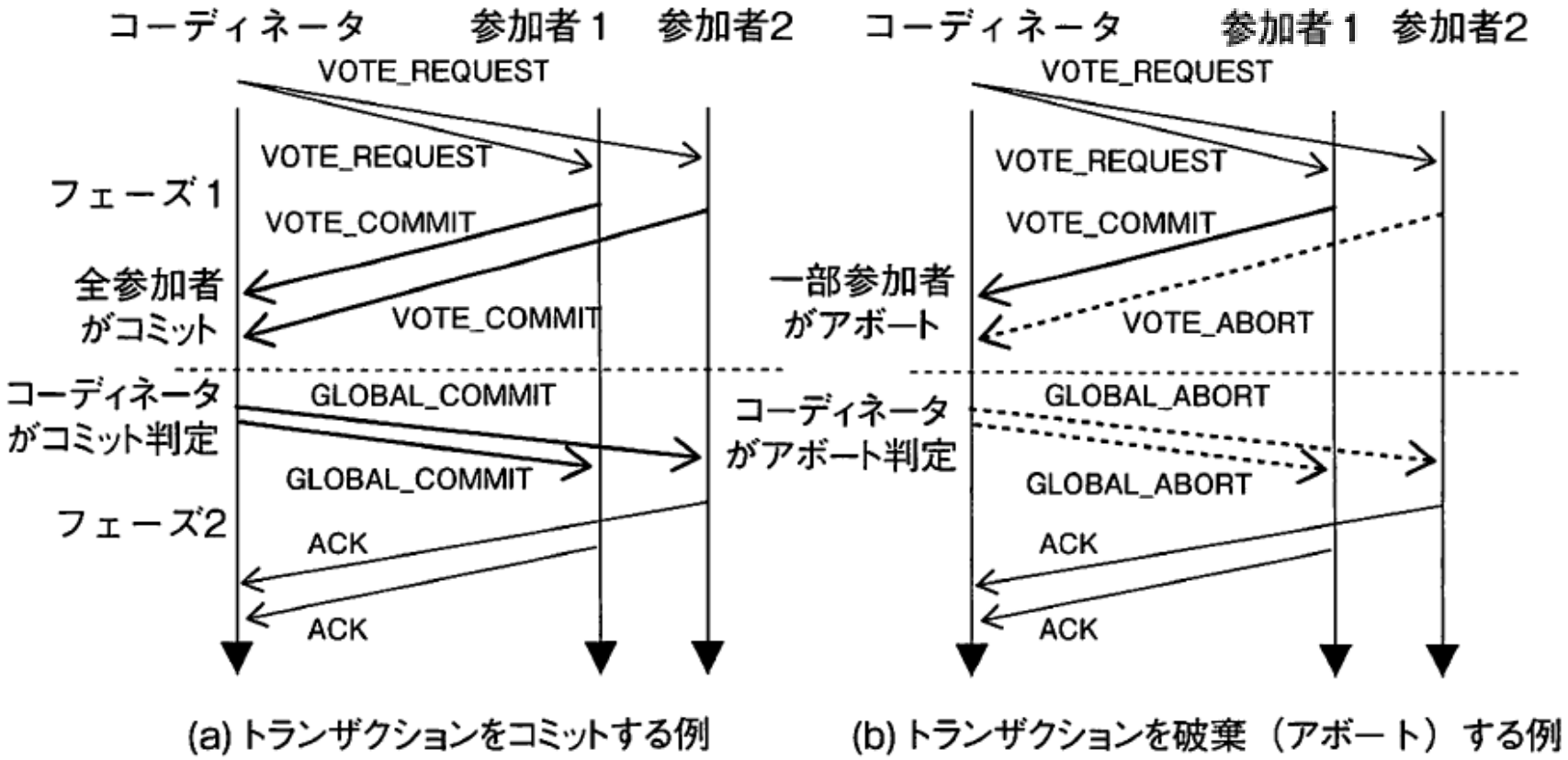
講義内容

- フォールトトレラント性
  - 概説
    - 基本概念
    - 障害モデル
  - 冗長性による障害の隠蔽
    - プロセス多重化
    - ビザンチン将軍問題
  - 高信頼クライアントサーバ間通信
  - 分散コミット
  - エラーからの回復
    - 安定ストレージ
    - チェックポイント作り

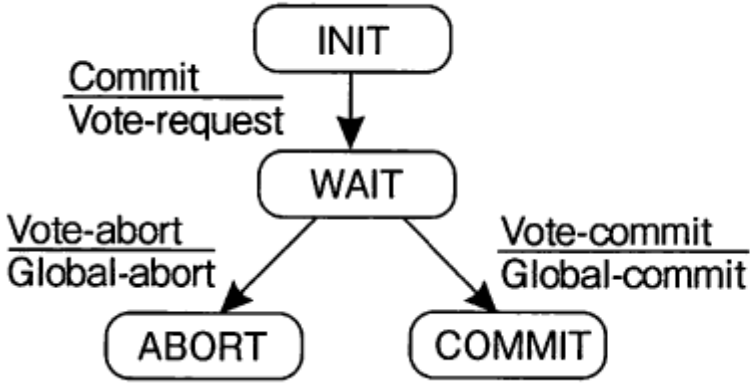
分散コミット

- 分散コミット
  - 原子コミット：グループ内のすべてのメンバによって行われるか、あるいは全く行われない、となるような処理
    - (例) 複製されたデータベースの処理
    - 通常、コーディネータにより主導される(他のメンバは参加者と呼ばれる)
  - 2相コミット：原子コミットを実現するための手段の一つ
    - (詳細は次ページ)

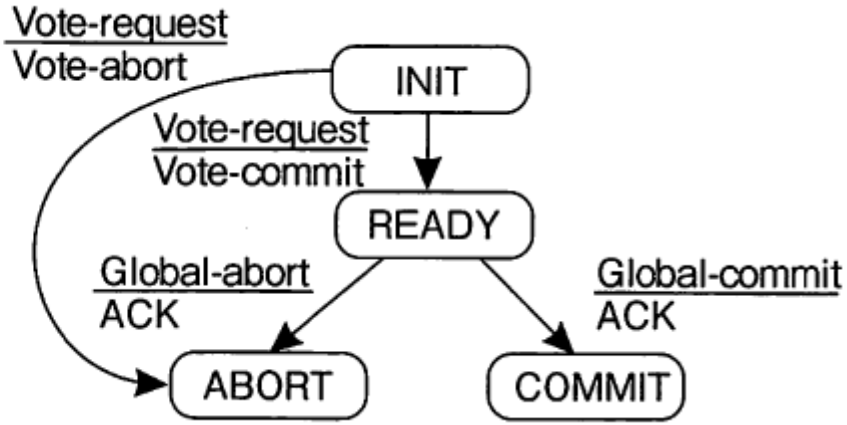
2相コミット (2PC: Two-phase commit protocol)



2相コミットにおける状態遷移



コーディネータの状態遷移



参加者の状態遷移

- ブロッキング状態：処理を待っている状態
  - (1) 参加者がコーディネータに要求を出した後で、コーディネータからVOTE_REQUESTが来るのを待っている(INIT)
  - (2) コーディネータが参加者の返答を待っている(WAIT)
  - (3) 参加者がVOTE_COMMITを送った後、コーディネータの判断を待っている(READY)

2相コミットにおける障害対策

- 参加者のINITの状態でタイムアウト
  - VOTE_ABORTをコーディネータに送信
- コーディネータのWAIT状態でタイムアウト
  - GLOBAL_ABORTを全参加者に送信
- 参加者のREADY状態でタイムアウト
  - 他の参加者に問い合わせる
    - 他の参加者がCOMMITであればCOMMIT
    - 他の参加者がABORTであればABORT
    - 他の参加者がREADY→判断不能(さらに他の参加者に聞く)

確認問題

- 以下の問いに答えよ。解答は選択肢から選ぶこと。
  - 2相コミットにおいて、コーディネータがVOTE_REQUESTを送信後、一部の参加者からの応答がタイムアウトしたとき、送信するメッセージはどれか。
  - 2相コミットにおいて、コーディネータがVOTE_REQUESTを送信後、一部の参加者からVOTE_ABORTが送られてきたとき、送信するメッセージはどれか。

    選択肢：VOTE_COMMIT, VOTE_ABORT, GLOBAL_COMMIT, GLOBAL_ABORT

講義内容

- フォールトトレラント性
  - 概説
    - 基本概念
    - 障害モデル
  - 冗長性による障害の隠蔽
    - プロセス多重化
    - ビザンチン将軍問題
  - 高信頼クライアントサーバ間通信
  - 分散コミット
  - エラーからの回復
    - 安定ストレージ
    - チェックポイント作り

エラーからの回復(Recovery)

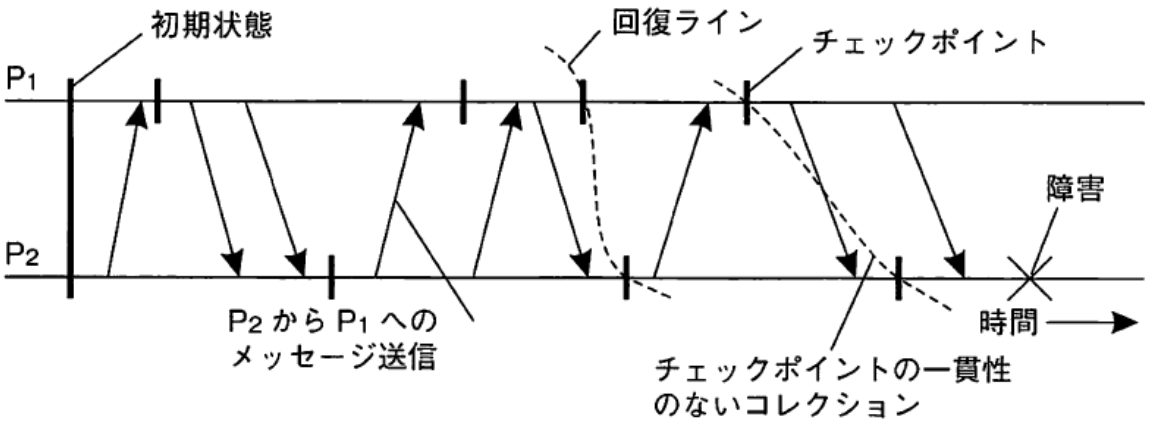
- 後ろ向き回復(backward recovery)
  - 以前の正しい状態に戻す
  - 時々正しい状態を記録し、チェックポイントを作成
    →後からチェックポイントに復帰(ロールバック)
- 前向き回復(forward recovery)
  - 実行を継続できるような正しい状態に
    遷移(ロールフォワード)
  - 発生する可能性のあるエラーが分かっていなければ
    できない
- 分散システムでは後ろ向き回復が一般的

安定ストレージ

- 自然災害以外の故障に耐えることを狙った記憶媒体
  - 正確なチェックポイント確保のため、様々なストレージ媒体の故障に耐えることが望ましい
  - RAID
    - RAID1: 二重化(ミラーリング)
    - RAID5: ディスク故障から復帰できるよう、冗長データ(パリティ)を生成

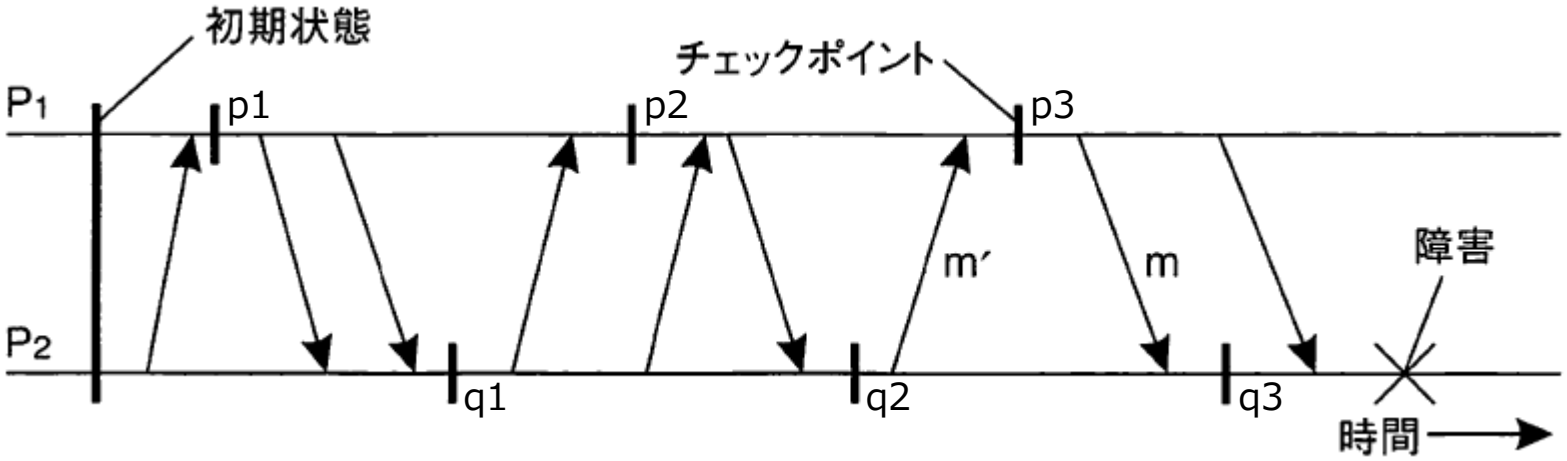
チェックポイント作り

- 分散システムにおけるチェックポイント
  - 一貫したグローバルな状態(分散スナップショット)が必要
  - 各プロセスはローカルの安定ストレージにすべての状態を保存
    → ローカルな状態からグローバルな状態を復元
  - 回復ライン(最新の分散スナップショット)に戻すことが望ましい



独立チェックポイント作り

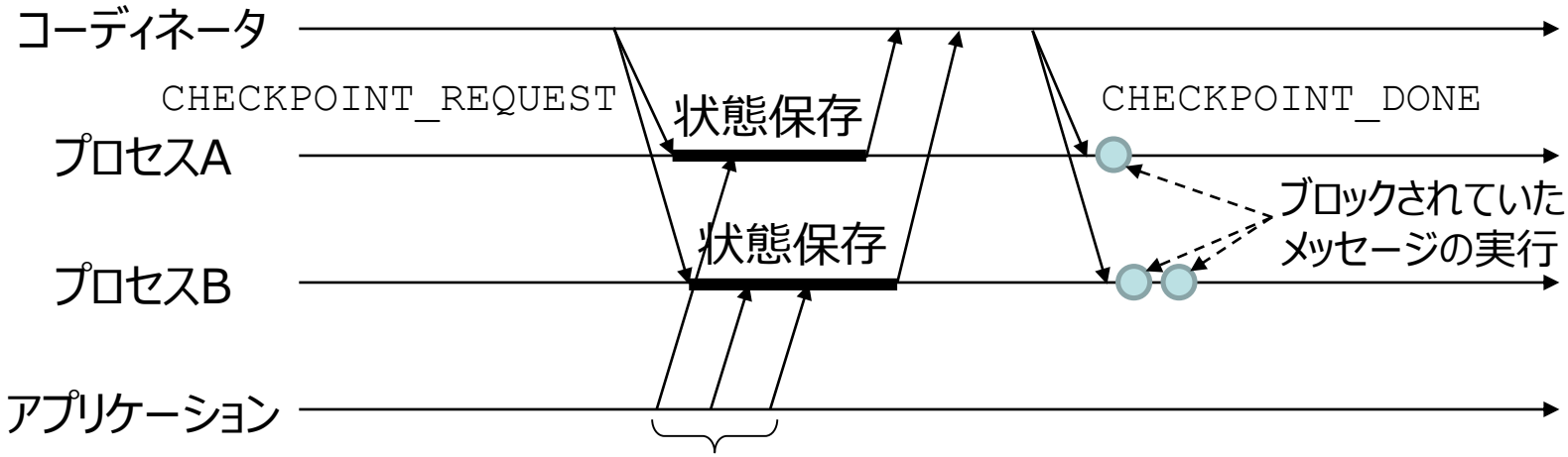
- 各プロセスが独立してチェックポイントを作成
  - 理想的な回復ラインが得られないことがある
  - 連鎖的なロールバック(ドミノ効果)が発生



(p3,q3)→mの送信前と受信後なので矛盾
(p3,q2)→m'の送信前と受信後なので矛盾
(p2,q2)→(以下略)
初期状態まで戻らざるを得ない

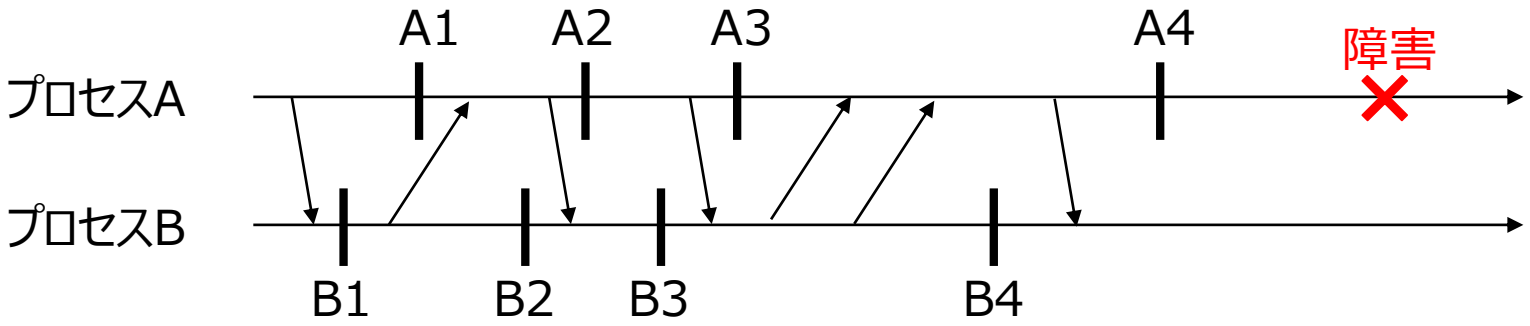
協調チェックポイント作り

- 独立チェックポイント作りの問題点を解決
  - すべてのプロセスが協調して分散スナップショットを作成
  - コーディネータによる2相ブロッキング
- プロセス間の連携が必要なため、性能低下
  - 無駄なチェックポイントを作らない分、ストレージへの負荷は減少



確認問題

- プロセスA, Bが以下のようにチェックポイントを作っていた場合、適切な回復ラインはどこか。A□-B□の形式で答えよ。



参考文献

- 「分散システム」
  水野 忠則 監修、共立出版、2015
- 「分散システム 原理とパラダイム 第2版」
  アンドリュー・S・タネンバウム 他 著、
  ピアソン・エデュケーション、2009
- 「JIS-X0014:1999」
  http://kikakurui.com/x0/X0014-1999-01.html