分散システム

第11回 フォールトトレラント性（2）

双見 京介（FUTAMI Kyosuke）
高田秀志（TAKADA Hideyuki）

2025年11月

高信頼クライアントサーバ間通信

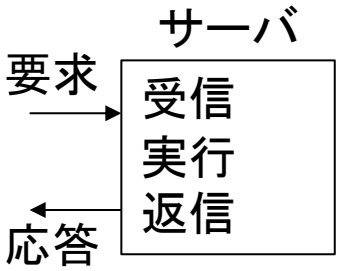
- Point-to-point通信
  - トランスポート層のプロトコルであるTCPにより高信頼な通信が可能
  - 複数経路を確保して信頼性を向上させるためのTCPの拡張として，MTCP（Multipath TCP）がある
- 故障がある場合の遠隔手続き呼び出し
  - クライアントサーバシステムにおけるクライアントやサーバのクラッシュ，メッセージの喪失への対処
  - どこまで透過性を確保できるかが重要

RPCにおける障害の発生

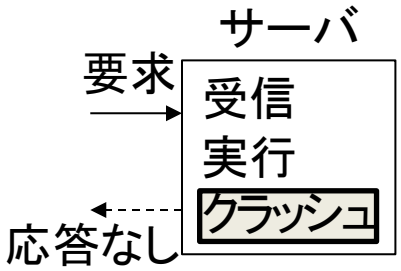
- 障害の発生をどこまで隠蔽できるか
- 起こりうる障害を5つに分類
  1. クライアントがサーバの位置を特定できない
  2. クライアントからサーバへの要求メッセージが消失
  3. サーバが要求を受けた後にクラッシュ
  4. サーバからクライアントへの応答メッセージが消失
  5. クライアントで要求メッセージ送信後に障害が発生
- それぞれの障害によって対処すべき問題が異なり，異なる解決策が求められる

障害発生における問題と解決策(1)

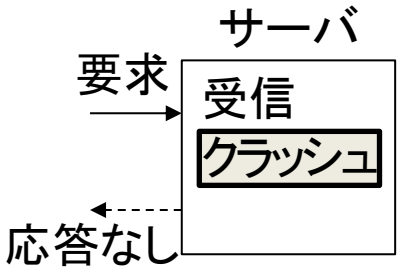
1. クライアントがサーバの位置を特定できない
   - 例えば，すべてのサーバがダウンしているなど
   - 解決策の一つとして，例外を上げてユーザに通知することが考えられるが，RPCの透過性が損なわれる
2. クライアントからサーバへの要求メッセージが消失
   - クライアント側でタイマを設定しておき，制限時間内にサーバから応答がない場合に要求メッセージを再送信する
3. サーバが要求を受けた後にクラッシュ
   - クライアント側では，サービスが実行されたか否かを知る術がないため，特別な対処が必要



(a) 正常ケース



(b) 実行後にクラッシュ



(c) 実行前にクラッシュ

サーバクラッシュに対する原理

- **「最低1回（at least once）」セマンティクス**
  - サービスが最低でも1回は実行されることを保証（1回以上の場合もあり得る）
  - サーバの再起動を待って，同じ処理をもう一度試みる
- **「最大1回（at most once）」セマンティクス**
  - サービスが最大でも1回まで実行されることを保証（0回の場合もあり得る）
  - すぐに要求を諦めて例外を通知
- 何もしない
  - サービスが実行されるかどうかについて何の保証もしない
  - サーバからの応答がなくてもクライアントは何もしない
- **「厳密1回（exactly once）」セマンティクス**
  - サービスが必ず1回だけ実行されることを保証
  - 最も望ましいが，実装する方法が一般には存在しない

障害に対する方策

クライアント側の方策とサーバ側の方策をどのように組み合わせても厳密1回セマンティクスを実現できない

サーバ（印刷サービスを提供）

クライアント

再送信の方策

常に行う
行わない
ACK受信時のみ
ACKを受信しない時のみ

ACK送信後に印刷する方策印刷後にACK送信する方策

M → P

MPC	MC(P)	C(MP)
DUP	OK	OK
OK	ZERO	ZERO
DUP	OK	ZERO
OK	ZERO	OK

P → M

PMC	PC(M)	C(PM)
DUP	DUP	OK
OK	OK	ZERO
DUP	OK	ZERO
OK	DUP	OK

M: ACKを送信　　OK: 1回だけ印刷される
P: 印刷を実行　　DUP: 2回印刷される
C: クラッシュ　　ZERO: 一度も印刷されない

MPC：ACKを送信して印刷後，クラッシュした
MC(P)：ACKを送信後クラッシュし，印刷が行われなった

どのような方策をとっても，サーバクラッシュのタイミングによってはDUPやZEROになってしまう

障害発生における問題と解決策(2)

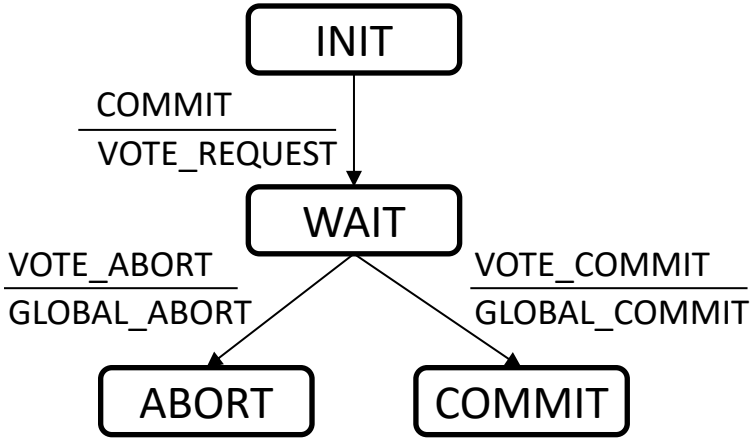
4. サーバからクライアントへの応答メッセージが消失
   - 要求メッセージや応答メッセージが喪失したのか，サーバが単に遅いだけなのか，クライアント側で感知することが不可能
   - 解決策の一つとして，クライアント側で要求メッセージに通し番号を付与し，サーバ側で同じ番号の要求メッセージを無視する
5. クライアントで要求メッセージ送信後に障害が発生
   - サービス自体は実行されているが，応答を受け取るクライアントがいない
   - このような状態になった要求はオーファン（orphan）と呼ばれ，適切に処分される必要がある

分散コミット（Distributed commit）

- ある操作が，グループ内のすべてのプロセスで実行されるか，あるいは，どのプロセスでも実行されないかのどちらかを保証（**原子コミット**）
- 2相コミット（Two-phase commit, 2PC）により実現
  - 投票フェーズ
    1. コーディネータは，VOTE_REQUEST メッセージをすべての参加者に送信
    2. 参加者は，操作を確定する場合は VOTE_COMMIT，操作を取り消す場合は VOTE_ABORT を返信
  - 決定フェーズ
    1. すべての参加者が VOTE_COMMIT を応答した場合，コーディネータは操作を確定することを決定し GLOBAL_COMMIT メッセージを送り，VOTE_ABORT を応答した参加者が一つでもある場合は，操作を取り消すことを決定し GLOBAL_ABORT メッセージを送る
    2. 参加者は，GLOBAL_COMMIT メッセージを受け取った場合はローカルに操作を確定し，GLOBAL_ABORT メッセージを受け取った場合にはローカルに操作を取り消す

2PCにおける状態遷移

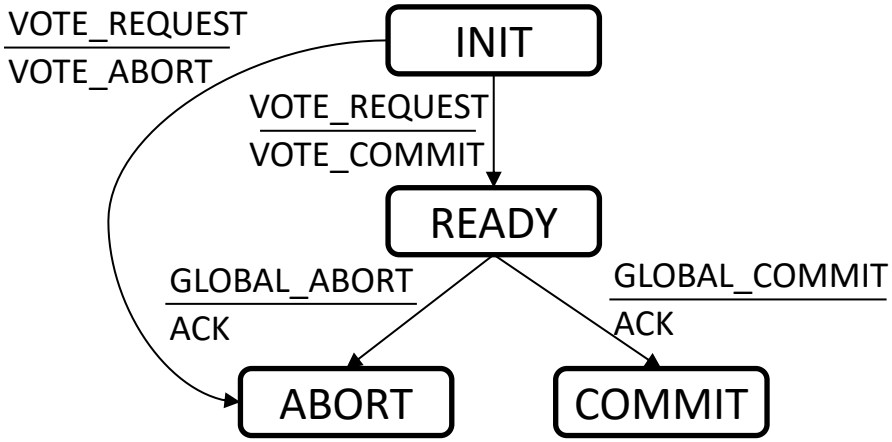
コーディネータ



INIT: 初期状態
アプリケーションから COMMIT メッセージを受け取ると，参加者に VOTE_REQUEST を送信

WAIT: 参加者からの応答待ち状態
参加者の一つから VOTE_ABORT を受け取ると，参加者に GLOBAL_ABORT を送信。参加者全員から VOTE_COMMIT を受け取ると，参加者に GLOBAL_COMMIT を送信

参加者



INIT: 初期状態
コーディネータから VOTE_REQUEST を受け取ると，操作を確定する場合には VOTE_COMMIT，操作を取り消す場合にはVOTE_ABORT を送信

READY: コーディネータからの応答待ち状態
GLOBAL_COMMIT を受け取ると操作を確定。GLOBAL_ABORT を受け取ると操作を取り消し

2PCにおける障害への対処

- タイムアウト（時間切れ）処理
  - 参加者がINIT状態で待っている時
    - 処理を取り消すことを決め，VOTE_ABORT をコーディネータに送信
  - コーディネータがWAIT状態で停止している時
    - すべての参加者に GLOBAL_ABORT を送信
  - 参加者がREADY状態で停止している時
    - コーディネータが復旧するまですべての参加者が待つ
    - 参加者 P が他の参加者 Q に問い合わせ，下表の方針で取るべき行動を決める

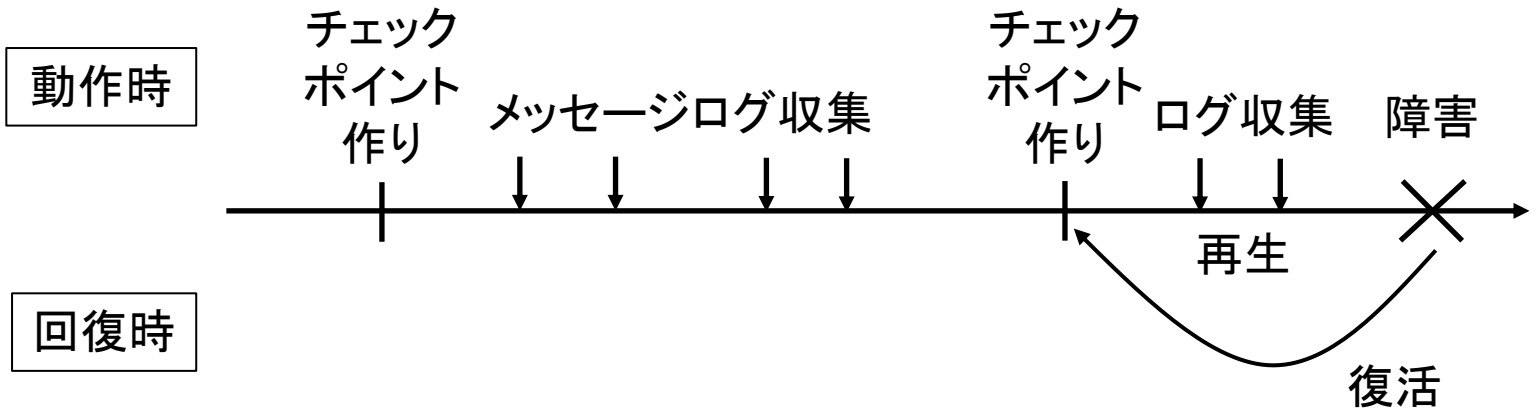
Qの状態	Pの取るべき行動
COMMIT	処理をCOMMIT
ABORT	処理をABORT
INIT	処理をABORT
READY	さらに他の参加者に問い合わせる

回復

- **前方回復（Forward recovery）**では，システムを実行可能な新しい状態へ移行させる
  - どのようなエラーが発生するのかを前もって知っておく必要がある
  - 喪失したパケットを正常に受信されたパケットから構成する**消失訂正（Erasure correction）**という手法が知られている
- **後方回復（Backward recovery）**では，現在のエラー状態から，以前の正しい状態へシステムを戻す
  - システムの状態を逐一記録しておき，問題が発生した時に記録された状態に復元する
  - システムの状態を記録した各時点を**チェックポイント（checkpoint）**という
  - チェックポイントはRAID（Redundant Array of Inexpensive Disks）のような安定ストレージに記憶される

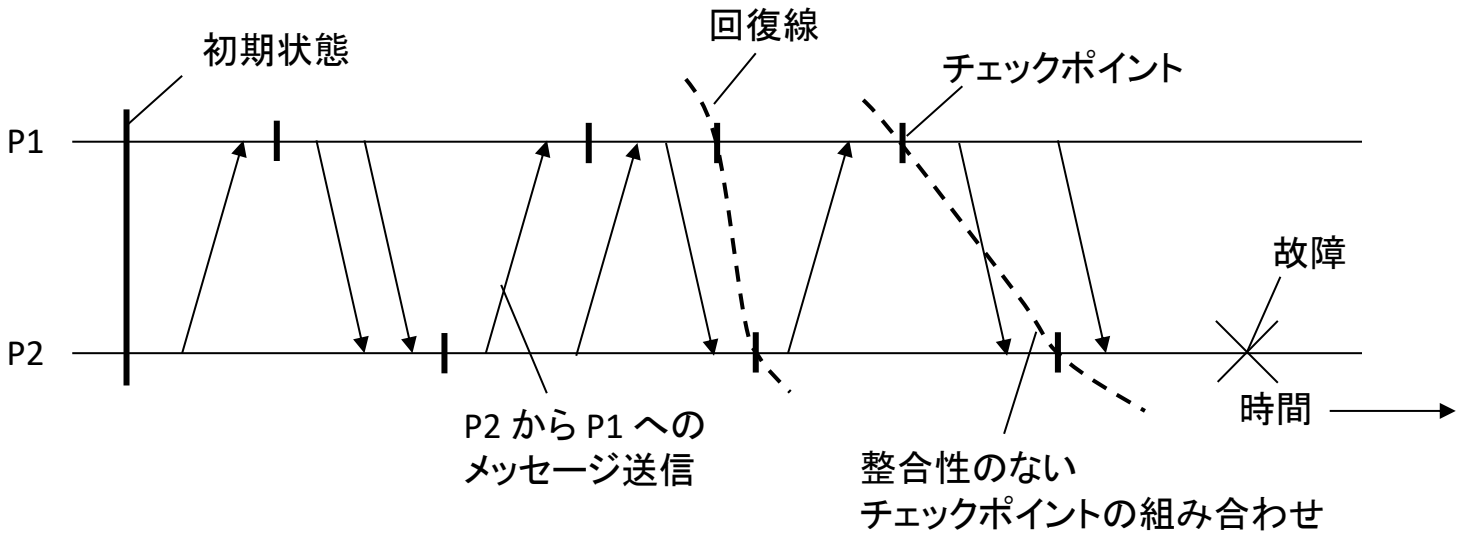
メッセージログ収集

- チェックポイント作りと同時に採用
- 送信者ベースログ収集と受信者ベースログ収集
- 受信プロセスがクラッシュした際，最も新しいチェックポイントを復活させ，送信されたメッセージを再生

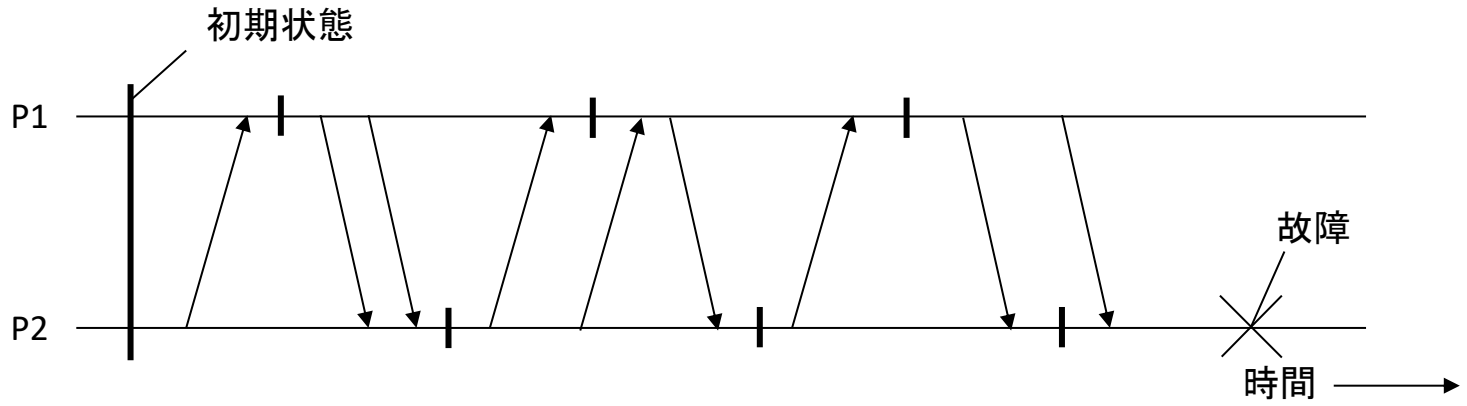


チェックポインティング

- 分散システム全体として整合性のある分散スナップショットを記録する必要がある
  - プロセス P がメッセージの受信を記録していれば，そのメッセージの送信を記録しているプロセス Q が存在しなければならない
  - 最も最近の分散スナップショットが回復線（recovery line）として復元される



独立チェックポイント作り



- プロセスがお互いに独立してチェックポイントを保存
- 回復線を見つけられない場合，連鎖的にチェックポイントを逆上る必要がある（ドミノ効果）
- 上図の場合では，初期状態のみ一貫性が取れている

協調チェックポイント作り

- すべてのプロセスが協調して状態を保存
- ドミノ効果が発生しない