分散システム

第10回 フォールトトレラント性（1）

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

2025年11月

はじめに

- フォールトトレラント性（Fault tolerance，耐故障性）とは，障害に耐えてシステムが動作を継続すること
- 非分散システムでは，障害が発生するとシステム全体のダウンにつながるが，分散システムでは，部分的に障害が発生しても他のコンポーネントに影響を与えないことがあり得る
- 障害が発生したとしても，程度の差はあれど，分散システムは動作を継続することが求められる

分散システムにおける要求事項

- 可用性（Availability）
  - システムがある瞬間において機能している確率
- 信頼性（Reliability）
  - ある期間に対してシステムが故障せずに機能し続ける確率
- 安全性（Safety）
  - システムが一時的に正しく機能しなくなった時に，重大な問題が発生しないという性質
- 保守性（Maintainability）
  - システムがどれほど容易に障害から回復できるか

これらをまとめて**ディペンダビリティ（Dependability）**という

基本概念

- システムが予定していた動作をできなくなった場合，**故障（fail）**したという
- **エラー（error）**は，障害を引き起こすかも知れないシステムのある状態のこと
- エラーの要因は，**障害（fault）**と呼ばれる
- **フォールトトレラント性（fault tolerance）**とは，システムに障害が起きてもサービスを提供し続けることをいう（障害によりエラーが起きても故障はさせない）

障害の分類

- 過渡障害（transient fault）
  - 一度だけ起こり，消滅する障害
  - ネットワーク上でのビットの消失など
- 間欠障害（intermittent fault）
  - 一時的に現れ，しばらくすると消える障害
  - コネクタの接触不良など
- 永久障害（permanent fault）
  - 修復されるまで存在し続ける障害
  - ソフトウェアのバグ，ディスクのクラッシュなど

故障モデル

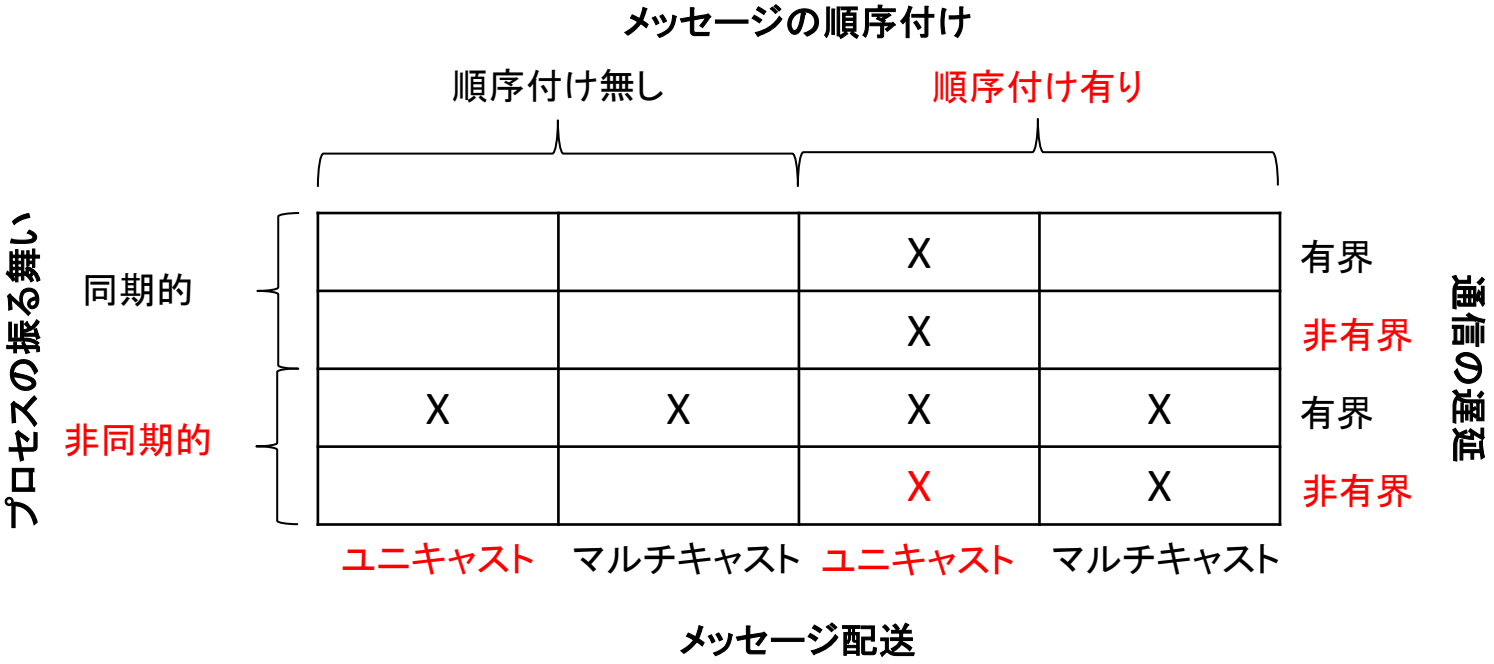
故障の種類	内容
クラッシュ障害 （Crash failure）	サーバが停止するが，停止するまでは正常に動作
欠落障害 （Omission failure） • 受信欠落 （Receive omission） • 送信欠落 （Send omission）	サーバが到着する要求への応答に失敗 • サーバが到着するメッセージの受信に失敗 • サーバがメッセージの送信に失敗
タイミング障害 （Timing failure）	サーバが指定された時間内に応答できていない
ビザンチン障害 （Byzantine failure）	サーバが任意の時間に任意の応答を返す

プロセスの多重化

- 能動的多重化
  - クライアントからのメッセージはすべてのレプリカに送付
  - すべてのレプリカの状態が一致する必要がある
    - プロセスの状態変更が決定的
    - 命令の実行が全順序性と原子性を満たす
- 受動的多重化
  - クライアントからのメッセージ処理は主レプリカで実行
  - 他のレプリカは主レプリカの状態に従う
  - プロセスの動作は決定的でなくても良い

障害を前提とした合意

- あるプロセスの集合が，通信やプロセスに障害が発生する可能性があったとしても，何らかの合意（例えば，コーディネータの選任）に至ることが要求される
- 分散システムにおける合意が実現できる状況は下図のようになる (Turek and Shasha, 1992)



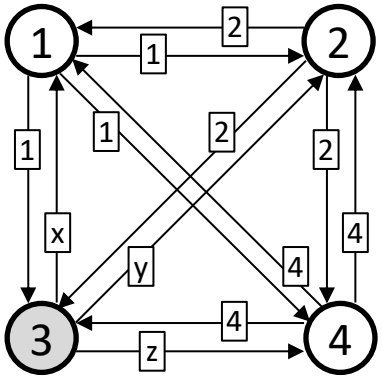
ビザンチン合意問題

(Lamport他, 1982)

- 裏切り者（故障したプロセス）がいる状況においても，複数の兵士（プロセス）の間で，戦力（各プロセスが保持している値）に対する合意を形成したい
- 仮定
  - プロセスは同期的に動作
  - メッセージは順序を保って1対1で配信
  - 通信遅延には上限がある
- N 個のプロセスがあり，それぞれのプロセス i が値 v_i を他へ提供する
- プロセス i が故障していなければ，$V[i] = v_i$ ，故障していれば $V[i]$ が不定となるような長さ N のベクトル V を構成することが目標
- 最大 k 個のプロセスが故障していることを仮定

ビザンチン合意問題のアルゴリズム

3個のプロセスが正常，1個のプロセスが故障している場合



ステップ1: 各正常プロセス i は他のプロセスへ値 v_i を送信。ここで，$v_i = i$ とする。プロセス3は故障しているので，不定値を送信

---

1 は (1, 2, x, 4) を得る
2 は (1, 2, y, 4) を得る
3 は (1, 2, 3, 4) を得る
4 は (1, 2, z, 4) を得る

ステップ2: ステップ1で送信された値を各プロセスがベクトルの形式で収集

1が得るもの	2が得るもの	4が得るもの
(**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)

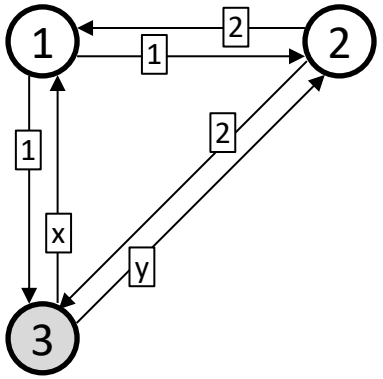
ステップ3: 各プロセスはステップ2で収集したベクトルを他のプロセスへ送信。プロセス3はここでも不定値を送信

Step 4: 各プロセスはステップ3で収集したベクトルの i 番目の要素を調べ，過半数を超えたものを採用

合意のための条件

2個のプロセスが正常，1個のプロセスが故障している場合

ステップ1



ステップ2

1 は (1, 2, x) を得る
2 は (1, 2, y) を得る
3 は (1, 2, 3) を得る

ステップ3

1が得るもの	2が得るもの
(1, 2, y)	(1, 2, x)
(a, b, c)	(d, e, f)

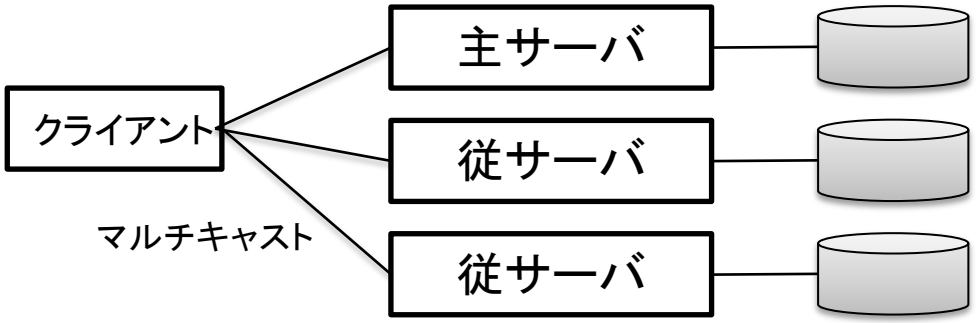
ステップ4

どの要素も過半数を満たさない

- k 個の故障プロセスが存在するとき，合意には $2k+1$ 個の正常プロセスが必要なことが Lamport らによって証明された。
- メッセージの到達に必要な上限時間が保証されない場合，一つのプロセスが故障しただけでも合意に至ることができないことが Fischer らによって証明された。

高信頼グループ間通信

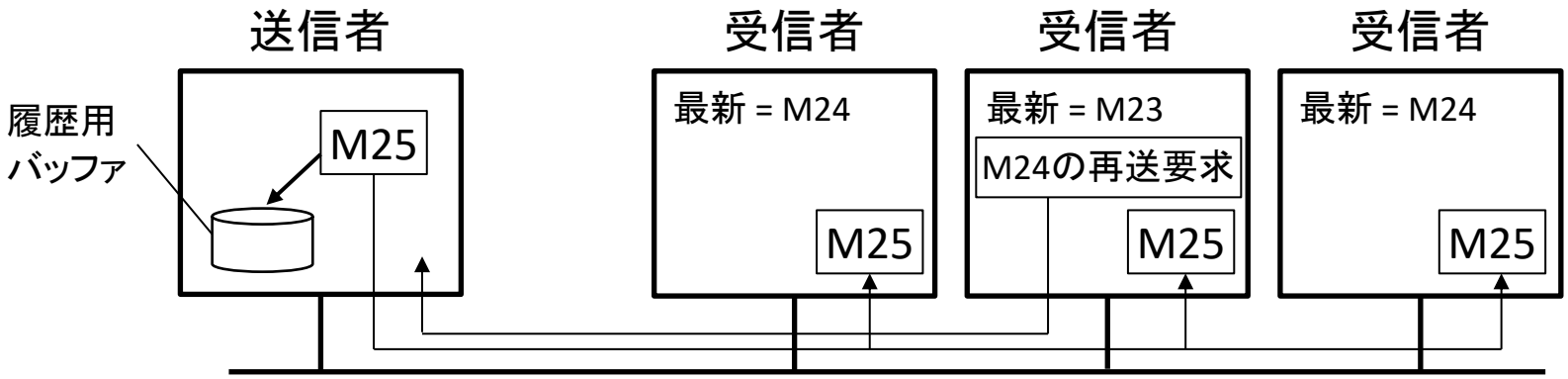
- 1対多の通信において，複数のプロセスへメッセージを配信する場合に満たすべき要件
  - すべてのメッセージがすべてのプロセスに届く
  - すべてのメッセージがすべてのプロセスに同じ順序で届く
  - プロセスが故障した場合，残りのプロセスすべてに配送されるか，全く配送されないかのどちらかである
- 複製されたデータベースの更新などに必要



すべてのデータベースを
同じように更新しないと
一貫性が保持されない

高信頼マルチキャストの基礎

- プロセスに故障がない場合を想定
- 実現方法
  - 送信者はメッセージに通し番号を付与
  - 送信者は送信メッセージをバッファに記録
  - 受信者は受信したメッセージの最新の通し番号を記録
  - 受信者はメッセージ受信時に抜けているメッセージがないかを通し番号により確認し，もしあれば，送信者に再送を要求

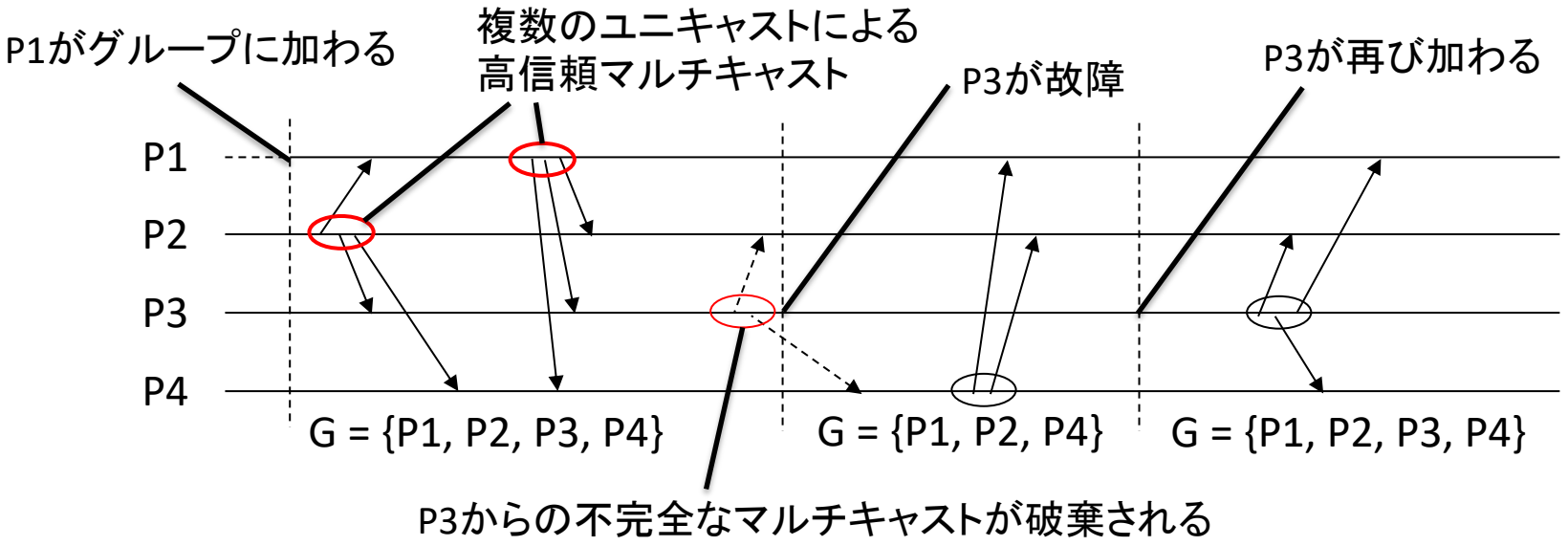


メッセージの順序保証

- 送信者が送った順序でメッセージが届くことを保証
  - 前ページの実現方法において，受信者はメッセージ受信時に送信者に送達確認通知（ACK）を送信
  - 送信者は，すべてのプロセスからACKを受信した後，次のメッセージを送信可
  - 送信者は，一定時間内にACKを受信しない場合，メッセージを再送信
- グループ内のプロセス数が多くなると，ACKの数が膨大になり（ACK爆発），送信者やネットワークの負荷になる
  - フィードバック抑制によるACKメッセージの削減
  - 階層的ネットワークによる通信相手のグループ化

原子マルチキャスト（Atomic Multicast）

- プロセスが故障する場合を想定
- 仮想同期（Virtually Synchrony）
  - 各プロセスがメッセージを受信すべきプロセスのリスト（グループビュー）を保持
  - グループビューGに含まれる障害の起こっていないすべてのプロセスにメッセージが配送されることを保証



仮想同期におけるメッセージの順序性

(1) 無順序
- メッセージの受信順序について何の保証もなし

(2) FIFO順序
- 同じプロセスからのメッセージは送信された順に配送される
- 異なるプロセスから送信されたメッセージの受信順序に関する制約はなし

(3) 因果順序
- メッセージ間の潜在的な因果関係（メッセージ m1 はメッセージ m2 より先に到着すべし等）が保持されるように配送

(4) 全順序
- グループ内のすべてのプロセスに同じ順序でメッセージを配送

故障検出

- 故障を適切に隠蔽するために必要
- 基本的な2つの手法
  - 能動的手法
    - プロセスがお互いに「生きているか」というメッセージを交換し合う
    - **ping**処理
  - 受動的手法
    - 他のプロセスからメッセージが届くのを待つ
- タイムアウト（時間切れ）により検出
  - ネットワークに信頼性がない場合，本当にプロセスが故障しているかを判断不能
  - より高度な故障検出サブシステムが必要