分散システム基礎とクラウドでの活用

第3回：グループ管理

国立情報学研究所
石川　冬樹
f-ishikawa@nii.ac.jp

今回の内容

- グループからのプロセスの脱落（や追加，削除）を扱う
  - 投票型の障害対応に関する重要な概念として，quorumおよびビザンチン合意を学ぶ
  - グループ管理サービスの構築方法を議論する

目次

- 固定グループにおける定足数アプローチ
- グループ管理サービスの構築

Quorum

- (プロセス数固定の)複製管理の方法において，過半数を用いる方法もよく知られている
  - 書き込みの際には，過半数のプロセスに書き込みが成功することを確認する
  - ➡競合する書き込み2つが両立することはない
  - 読み込みの際には，過半数のプロセスから読み込むようにする
  - ➡少なくとも1個のプロセスは最新の値を知っている
- 実際には「過半数」に限らないこともよく知られている(Quorum：定足数)

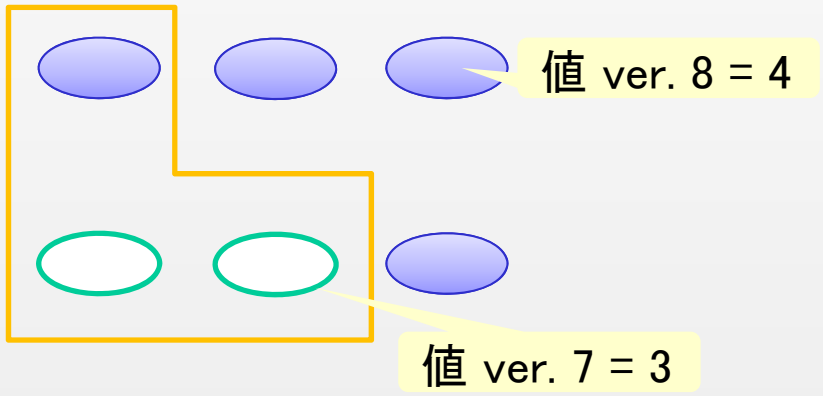
Quorum

- 読み込まなければならない最小のプロセス数QR（Quorum-Read）と，書き込みを行わなければならな最小のプロセス数QW（Quorum-Write）について，
  QW+QW と QR+QWがともに
  プロセス数を超えていればよい

  - JBoss（Javaアプリケーションサーバ）などでも利用されている（事例は次回以降に主に言及）

Quorum

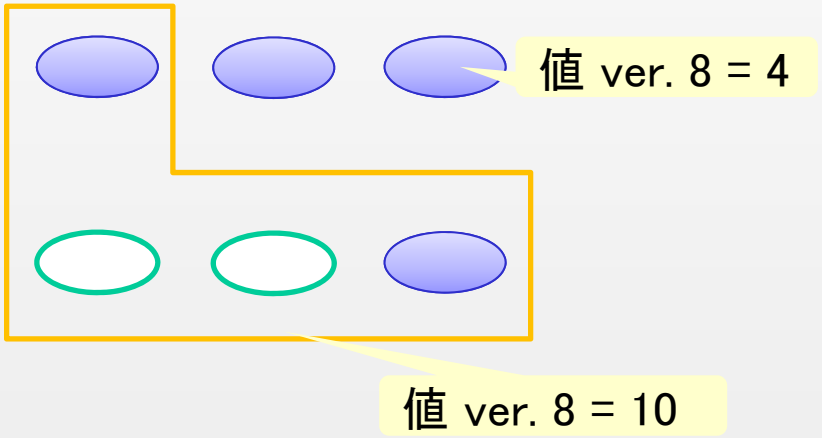
4個に最新の値書き込み



3個読めば最新の値を見つけられる
（4 + 3 > 6）

書き込み3，読み込み3や，
書き込み4，読み込み2だと，
そうとは限らない

4個に最新の値書き込み



4個に競合する書き込みをしようとすると，
競合に気づくことになる
（4 + 4 > 6）

書き込み3個だと，競合する書き込みが
2つ両立しうる

Quorum

- なお，プロセス数がnのときに，
  - QW=1，QR=nとすると，
    「全てに書き込み」「読み出しはどれでも1つから」
  - QW=n，QR=1とすると，
    「1つだけに書き込み」「読み出しはすべてから」

ビザンチン将軍問題

- 通信に障害はなくプロセスに障害がありうる場合を表す比喩
  - シナリオ
    - A軍がn個に分かれている
    - 将軍は電話でそれぞれの部隊の人数を知らせ合い，軍の全人数について同意する
  - 問題
    - n人の将軍のうちm人は裏切り者であり，不正確で矛盾する情報を流し，同意に至るのを防ごうとする
    - nとmの関係により同意に至れない

ビザンチン将軍問題

- よく知られた解法 [Lamport, 1982]
  - 各将軍は，他の全員にそれぞれ自分の部隊の人数を伝える
  - 各将軍は，自身が把握した各部隊の人数を他の全員に送る
  - 各将軍は，受け取った各部隊の人数について過半数が同じ値であればそれを記憶，そうでなければ不明とする

ビザンチン将軍問題

- 例：4人のうち将軍3が裏切り者の場合
  - 各将軍は，他の全員にそれぞれ自分の部隊の人数を伝える

将軍1が得る情報：　部隊1 -> C1，部隊2 -> C2，部隊3 -> a，部隊4 -> C4
将軍2が得る情報：　部隊1 -> C1，部隊2 -> C2，部隊3 -> b，部隊4 -> C4
将軍3が得る情報：　部隊1 -> C1，部隊2 -> C2，部隊3 -> C3，部隊4 -> C4
将軍4が得る情報：　部隊1 -> C1，部隊2 -> C2，部隊3 -> c，部隊4 -> C4

C1, C2, C3, C4: それぞれ正解
a, b, c, …: 将軍3が流す不正確な情報

ビザンチン将軍問題

- 例：4人のうち将軍3が裏切り者の場合
  - 各将軍は，自身が把握した各部隊の人数を他の全員に送る

将軍1が将軍2, 3, 4から受け取る情報：(C1, C2, b, C4), (d, e, f, g), (C1, C2, c, C4)

将軍2が将軍1, 3, 4から受け取る情報：(C1, C2, a, C4), (h, i, j, k), (C1, C2, c, C4)

将軍4が将軍1, 2, 3から受け取る情報：(C1, C2, a, C4), (C1, C2, b, C4), (l, m, n, o)

C1, C2, C3, C4: それぞれ正解
a, b, c, …: 将軍3が流す不正確な情報

ビザンチン将軍問題

- 例：4人のうち将軍3が裏切り者の場合
  - 各将軍は，受け取った各部隊の人数について過半数が同じ値であればそれを記憶，そうでなければ不明とする

将軍1が将軍2, 3, 4から受け取る情報：(C1, C2, b, C4), (d, e, f, g), (C1, C2, c, C4)
-> (C1, C2, ?, C4)

将軍2が将軍1, 3, 4から受け取る情報：(C1, C2, a, C4), (h, i, j, k), (C1, C2, c, C4)
-> (C1, C2, ?, C4)

将軍4が将軍1, 2, 3から受け取る情報：(C1, C2, a, C4), (C1, C2, b, C4), (l, m, n, o)
-> (C1, C2, ?, C4)

忠実な将軍は合意に至ることができる

C1, C2, C3, C4: それぞれ正解
a, b, c, …: 将軍3が流す不正確な情報

ビザンチン将軍問題

- 別の例: 3人のうち将軍3が裏切り者の場合
  - 各将軍は，他の全員にそれぞれ自分の部隊の人数を伝える

将軍1が得る情報: 部隊1 -> C1，部隊2 -> C2，部隊3 -> a
将軍2が得る情報: 部隊1 -> C1，部隊2 -> C2，部隊3 -> b
将軍3が得る情報: 部隊1 -> C1，部隊2 -> C2，部隊3 -> C3

C1, C2, C3, C4: それぞれ正解
a, b, c, …: 将軍3が流す不正確な情報

ビザンチン将軍問題

- 別の例: 3人のうち将軍3が裏切り者の場合
  - 各将軍は，自身が把握した各部隊の人数を他の全員に送る

将軍1が将軍2, 3から受け取る情報: (C1, C2, b), (c, d, e)

将軍2が将軍1, 3から受け取る情報: (C1, C2, a), (f, g, h)

C1, C2, C3, C4: それぞれ正解
a, b, c, …: 将軍3が流す不正確な情報

ビザンチン将軍問題

- 別の例：3人のうち将軍3が裏切り者の場合
  - 各将軍は，受け取った各部隊の人数について過半数が同じ値であればそれを記憶，そうでなければ不明とする

将軍1が将軍2, 3から受け取る情報：(C1, C2, b), (c, d, e)
-> (?, ?, ?)
将軍2が将軍1, 3から受け取る情報：(C1, C2, a), (f, g, h)
-> (?, ?, ?)

忠実な将軍は合意に至ることができない

C1, C2, C3, C4: それぞれ正解
a, b, c, …: 将軍3が流す不正確な情報

ビザンチン将軍問題

- この解法が満たす性質
  - m個障害を持つプロセスがあっても，2m+1個正常なプロセスがあれば（全体の3分の2より多く正常ならば），合意形成が可能
- 概念を説明する比喩，および障害の分類としては著名
  - ビザンチン故障（実行結果のぶれや壊れ，もしくは応答なしなど）
  - それに耐えられるか：ビザンチン・フォールトトレラント性（BFT）

ビザンチン将軍問題

- 「プロトコル」としては想定がなかなか合わない？
  - 通信については楽観的
  - プロセスについては，乗っ取りや誤動作などを悲観的に想定する一方，それらが起きるプロセスの数は一定以下と確信できなければならない
- 使い処はこれまでの複製管理などとは違う
  - 例：あまり信頼できないサーバ群にデータを複製して置き，データの完全性（改ざんなし，など）を実現しようとする

目次

- 固定グループにおける定足数アプローチ
- グループ管理サービスの構築

グループ管理：意義

- これまで，分散コミット，マルチキャスト，複製管理と，プロセスのグループが固定であるような想定で議論してきた
- 実際には
  - プロセスが追加されたり，削除されたりすることもある
  - プロセスが，クラッシュまたはネットワーク切断により，グループから（少なくとも一時的に）脱落する可能性は当然考えるべき

グループ管理：アプローチ

- グループにどのプロセスが属するか，という情報を集中管理するサービスの構築を考える（GMS: Group Management Service）
  - 性能および耐故障性のため，サービス自体も複数のプロセスにより実現するが，そちらは数が少ないという想定で，プロトコルで実現する（GMP: Group Management Protocol）

※ 基礎知識・技術としてよく知られた単語や方法ではなく，教科書「Guide to …」の著者らの方法

グループ管理：GMS

- GMSが提供する機能インターフェース
  （いずれもidempotentで，どのサービスプロセスを呼んでもよい ）
  - 新しいプロセスを登録
    （そのプロセスが自分自身で登録）
  - プロセスを削除
    （他のプロセスが監視結果などから呼び出してもよいし，自分自身で呼び出してもよい）
  - プロセスを監視
    （対象プロセスが削除されたら通知するように，コールバックを登録しておく）

グループ管理：GMP（生存確認）

- GMPにおける生存確認
  - プロセスは互いにpingなどを通し生存確認を行う
  - 応答がない（プロセスかネットワークが落ちたと思われる）プロセスは，落ちたとして「敬遠」する
    - そのプロセスからのメッセージは拒否
    - そのプロセスが落ちたと他のプロセスに周知
  - このように扱われたプロセスが実は生きていた場合，他のプロセスからの拒否で気づき，新たなプロセスとして自信を再度グループに登録する

グループ管理：GMP（調整者）

- GMPにおける調整者
  - 調整者を設け，追加または削除を管理する
  - グループへの参加が古い順に，調整者になっていく（落ちたと思われる際には引き継がれる）
  - 追加，削除は複数のプロセスに対して行えるが，削除の場合は現在のプロセスリストの過半数未満だけ
  - 追加または削除の確認メッセージを，現在のプロセスリスト内の全プロセスに送り，ackを待つ
    （次頁に続く）

グループ管理：GMP（調整者）

- （続）
  - 調整者は，過半数のプロセスからackが来たら，更新内容を確定するように周知する
    - またackが来なかったなどエラーについて報告（削除処理の起動を兼ねるとしてもよい）
  - 調整者が過半数からackが得られなかった場合，過半数いる逆側が自分たちを削除し，新しいプロセスを追加している可能性がある（避けられない）
  - GMSを構成するプロセスは限定されたサーバで運用されることが多く，検出できるだろう

グループ管理：GMP（調整者）

- （続）
  - 調整者が新しい調整者により削除されようとする場合，新しい調整者は古い調整者の削除について通知する
    - 古い調整者が完了していない追加，削除に関する情報をついでに集める
    - 各プロセスは古い調整者を「敬遠」するようになる
  - その後，これまで示した手続きをとる

グループ管理：GMP（調整者）

- ネットワーク分断に対して問題ないのか？
  - もしも「追加・削除」実行が2つ（以上）並行に起動されていてしまっていたら？
  - このプロトコルでは，過半数のackをとる
  - ➡ 少なくとも1個以上のプロセスが両方の「追加・削除」実行に参加しており，そのことに気づける

グループ管理：GMPまとめ

- 以上の方法でGMPが達成できる性質
  - グループに属するプロセスの初期リストから始め，追加・削除が行われたリストが更新されていく
  - 各プロセスは，その更新の履歴列の部分列（そのプロセスが加わってから，抜けるまで）の報告を受けることになる
  - i番目のリストにおける過半数が，i+1番目のリストに同意しなければならない
  - プロセスpがプロセスqが落ちているようだと判断し，GMSが機能しているならば，pまたはqの片方または両方がリストから抜ける

グループ管理：その他

- ネットワークの遅延上限に関する知識
  - それがあれば，「自身が切り離されてしまい，周りはもう自分を「敬遠」するはず」と確実に気づける
- ネットワーク分断時の扱い
  - GMPにおいて，過半数ではない数のプロセスだけがつながっていることに気づいたとき，そのまま実行を続ける拡張もある
  - ➡ ネットワークが繋がった際に，知識をマージし，過半数を作れる「本体」になれるか確認する

グループ管理：GMS

- 以上のように実現されるGMSを，他の大多数のプロセスはクライアントとして利用する
- なお，このGMSの実現は，単に「グループに属するプロセスのリスト」というデータの複製管理をしただけでもある
  - クライアントの引数が一意になるようにし，操作がidempotentに設計したため，呼び出そうとした複製（サービスプロセス）が落ちても大丈夫になっている点が留意事項

今回のまとめ

- 特にプロセスの障害を考えるときは，具体的なプロトコルだけでなく，典型的な考え方を統合して，性能や耐故障性などを総合的に考慮して実現方法を検討する
  - 調整者の設定やその固定順序での交代，操作をidempotentに，定足数に基づいた確定，役割交代するための情報の冗長保存と削除，・・・
- 次回：イベントの順序づけについて議論する