クラウドシステム基礎
第6回：クラウドサービスの設計思想（２）

国立情報学研究所
石川　冬樹
f-ishikawa@nii.ac.jp

今回の内容

- スケーラビリティや可用性，伸縮性のための
  クラウドサービスにおける設計思想について，
  引き続き議論する

目次

- 演習：クラウドサービスの活用
- 補足：複製管理に関わる他のサービス例

演習：データストアの機能制限

- スケーラビリティ・可用性を重視したデータストアを利用することを想定し，その制限の受け入れ可否について議論する
  - 例： 前回のDynamo（内部版）
  - 書き込みが大量に来ることがあることを前提とし，スケーラビリティ・可用性を重視
  - データストアのクライアントに対して保証する，一貫性，耐久性，およびAPIが提供する機能に制限がある

演習：データストアの機能制限

- 想定するデータストア
  - キーに対応する値の読み書き（get/put）のAPI
  - 読み（get）はイベンチュアル一貫性を保証：ある複製において「成功」として受け付けられたputの結果を，getが反映しないことがある
    - R+W>Nとなるような RやWへの確認をとることなく，クライアントにget/putの結果を返している
    - putの結果はほとんどの場合1秒以内に全複製に反映されgetされるようになると仮定する
    - 競合バージョンについては内部版Dynamo同様の把握が可能だとする

演習：データストアの機能制限

- カウンター機能（1）：オンライン通販サイトにおける人気度合いを測る
  - 商品ごとに，商品ページの閲覧数，「欲しい」「いいね」ボタンが押された回数などを数えたい
  - これらの値を用い，おおよその人気をすぐに把握したい（例えば，トップページやカテゴリ別ページのオススメ表示を分・時間単位で更新したい）
- 想定するデータストアを用い，カウンター処理（値のインクリメント・取得）を実現することを検討
  - 頻度が高い書き込みであることは間違いない

演習：データストアの機能制限

- カウンター機能（2）：オンライン通販サイトにおける，限定商品の在庫を把握する
  - 購買記録はRDBなどにしっかり記録（ACID）
  - 「まだ買える」ことの確認のための在庫数読み込みは頻発するため，商品ごとの在庫数は上記RDBとは別にも保持し，高速にアクセスしたい
  - 購入時にはこの在庫数情報も併せて更新する
- 想定するデータストアを用い，カウンター処理（値のデクリメント・取得）を実現することを検討

演習：データストアの機能制限

- 議論の焦点：想定データストアでどう実現するか？適切に実現できるか？
  - アプリケーション側がAPIを用いるロジックおよび，データストアに入れるデータの構造
  - 必要なら，データストア拡張も検討とし簡易設計を
  - 必要なら，アプリケーション側の妥協（真に必要な要件とそうでないものの明確化）を考えてもよい
  - アクセスは十分に多いことを想定し，可用性・スケーラビリティを重視（ノードは豊富とする）
  - 耐久性（put結果が失われる確率）は議論しない

演習：データストアの機能制限

- 以上のカウンター機能の実現について，グループで議論する
- 話題は（講義内容に関連する範囲で）発散してもよい
  - そもそもの内部版Dynamo設計に関する議論
  - Amazon（やFacebookなど）の表示内容の確認と設計の予測
  - ・・・
- その後，現在の一般向けAmazon DynamoDBのドキュメント（次頁）を読んでみて，また議論する

演習：データストアの機能制限

- 現在の一般向けAmazon DynamoDBのドキュメント
  - 入口：
    http://docs.aws.amazon.com/ja_jp/amazondynamodb/latest/developerguide/Introduction.html
  - 「DynamoDB での項目の操作」が今回の内容に特に関連：
    http://docs.aws.amazon.com/ja_jp/amazondynamodb/latest/developerguide/WorkingWithItems.html

目次

- 演習： クラウドサービスの活用
- 補足：複製管理に関わる他のサービス例

Memcached

- メモリ上に保持したKey-Valueペアの読み書きのためのAPI(とそのオープンソース実装)
  - 典型的な利用法は,「まずメモリ上から読み出しを試み, もしなければバックエンドから取得し追加」
  - 多数の実装と利用
    - Wikipedia, YouTube, Flickr, Twitter, WordPress.com, mixi
  - 各プロセスのメモリは基本互いに独立
    - 一部の実装では値の更新をプロセス間で共有
- AWSではElastiCacheというサービスにて利用

Google BigTable

- 概要
  - 一貫した表データアクセスを実現
  - (K, K`, V)タプルに対する操作(行，列，値)
  - さらに時刻も保持
    - 古くなったものが消えるように設定できる
    - またはアプリケーションが明示的に削除
  - 元々はWeb検索の情報保持のために開発

	contents:	anchor:cnnsi.com	anchor:my.look.ca
com.cnn.www	<HTML> …	“CNN”	“CNN.com”

Google BigTable

- 高速アクセスの可能性を開発者が指定
  - 列のキーは事前登録されたfamilyに分類される
    - 前頁の anchor:XXX, anchor:YYY
  - 同じ行および同じ列のファミリーに属するデータは，高速にアクセスされるよう物理的に「まとめて」置かれる
  - 複数のアプリケーションにより用いられる想定
    - familyの事前登録時に，名前の衝突に気づく
    - 概念的には一つの大きな表だが，値が入っていないところがほとんど（その分の領域は使わない）

Google BigTable

- トランザクション・耐故障性
  - トランザクションのサポート
    - 1つの行を原子的に更新
    - あるセルをカウンターとして利用可能
      （同じ値が2度読まれることはない）
  - データセンター間分断の場合には，個々が独立して動作し，後でミラーリングにより復旧
    （データセンター内での分断発生は前例なし）

Google BigTable

- 設計思想
  - 行や列，列のfamilyを対象とするよう，APIやトランザクションの種類を絞っている
    - RDBのあらゆる操作が可能になっているわけではない
  - それら（だけ）が十分に速く動作するように，複製された小さなサーバ集合に割り当てる
  - （「あきらめ」を含めた）APIの設計と，実装方式の設計との連動により成功している

Yahoo! Zookeeper

- Yahoo!による，ファイルシステム風操作を提供するKey-Valueストア
  - ファイル名がKeyとなる
  - ファイルのバージョン番号も保持
  - 全順序アトミックブロードキャストを利用
  - ファイル名の階層構造に基づき，処理を分散

その他

- RDB風の操作を提供しているKey-Valueストア
  - Yahoo PNUTS, Cassandraなど
  - 強い一貫性を保証しないなどトレードオフはある
- Chubby: BigTableの裏側のロックサービス
  - 全順序マルチキャストによるロック管理
  - ノード数に対してスケールしない
    （独立したロック集合を扱うChubbyインスタンスを
    たくさん作ることはできる）

今回のまとめ（前回と同じ）

- Web企業が自身の要求から産み出したクラウドでの技術は，これまでと異なる設計思想に基づいている
  - スケーラビリティや可用性，伸縮性のため，一貫性や実現可能な機能が意図的に制限されている
  - 開発者（データストア利用者）側が留意すべき制約が多くなっている
- 次回：これらの設計思想に関連するCAP定理について説明し，本講義の振り返りを行う