データクラウドを支える技術と研究動向

奈良先端科学技術大学院大学
情報科学研究科
宮崎　純

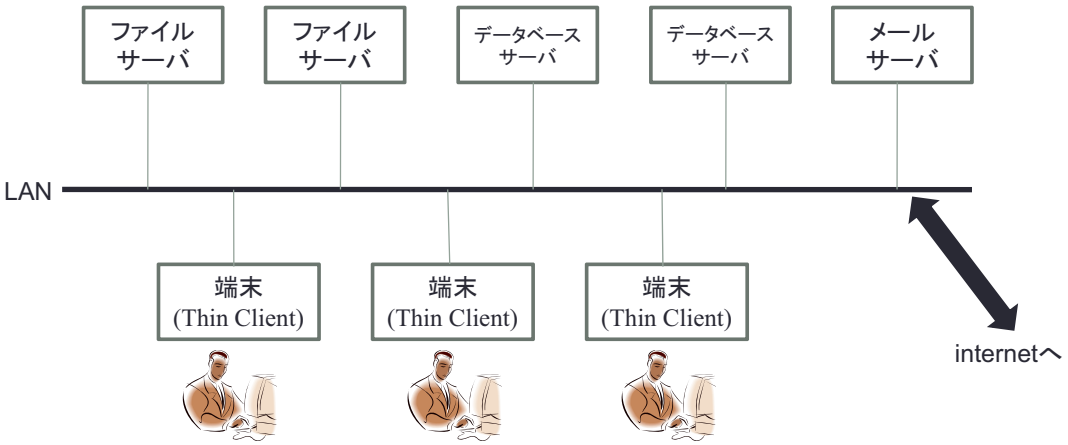
概要　～NOSQL～

概要

- NoSQL 概要
- NoSQLを支える技術
- 事例

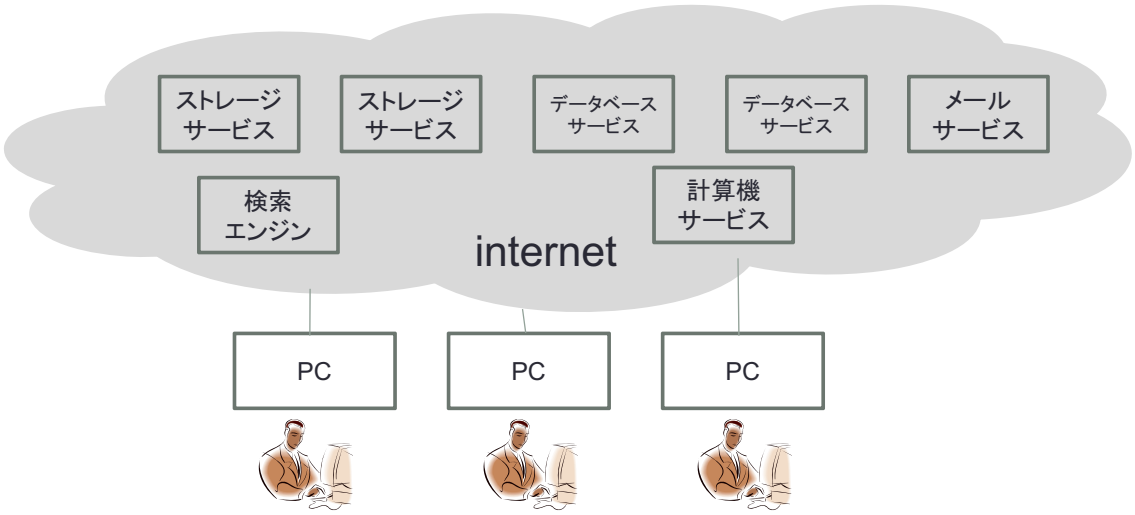
コンピュータ環境の変遷 (1)

- 分散コンピューティング　(1980年代～)
  - サービスが行われる場所が明確
  - サービス対象は機関内



コンピュータ環境の変遷(2)

- クラウドコンピューティング　(2000年代～)
  - サービスが行われる場所がインターネット上のどこか
  - サービス対象は全世界



なぜクラウド？

- 広域（全世界対象）のサービスの提供
  - インターネットを介して、誰でもどこからでも利用可能
    - 検索エンジン
      - Google, Yahoo!, bing, …
    - メール
      - GMAIL, Yahoo! mail, Hotmail,…
    - SNS、ビデオ・写真共有
      - Facebook、mixi、Youtube, Flickr
    - ネットショップ、ネットオークション
      - Amazon、楽天、e-Bay
- データ規模が莫大

スケールアウトするデータ管理が必須

クラウドのデータ管理

- スケールアウトするデータ管理の現実的解決
  - 多数の計算機を並べて（分散システム）でスケールアウト
    - 分散RDBMSは、ACIDトランザクションがネックでスケールしない
      - 特に、大規模な分散環境では2PCがボトルネック

NoSQL

(Not only SQL)

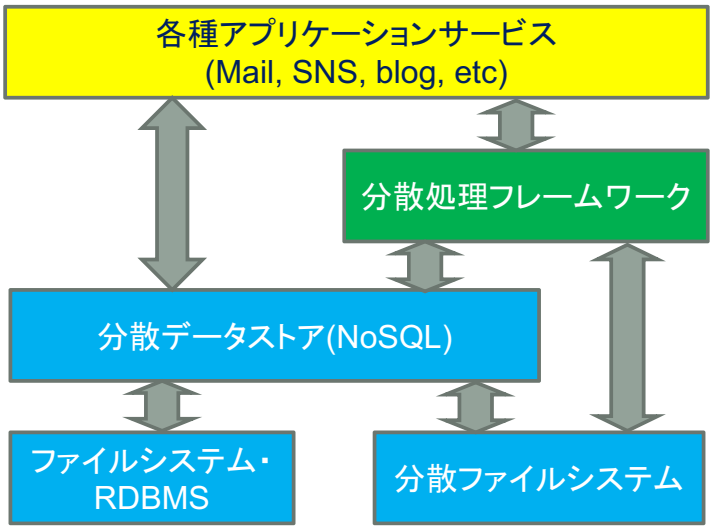
NoSQL

- NoSQLの目指すところ
  - Create, Retrieve, Update, Delete （CRUD）の一貫・高可用・高スループット処理

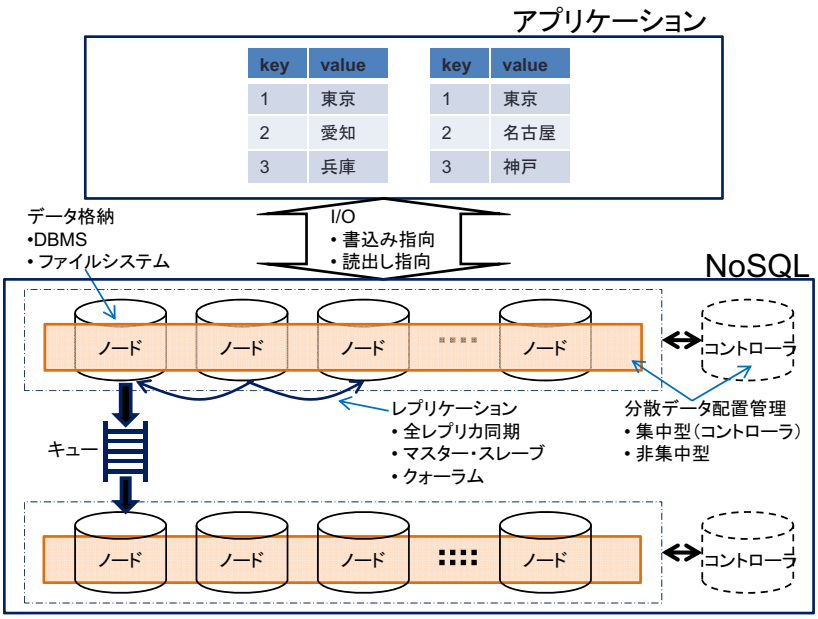
  - RDBMS(SQL)ではサポートできない機能（スケーラビリティ）の提供
  - RDBMS(SQL)ではオーバスペックな機能を簡略化（or 省略）
    （例）
    - ACIDトランザクション → BASEトランザクション
    - 索引 → 限定的なサポート
    - 関係演算 → 省略
    - ローストア → カラムストア

- NoSQL = データベース技術＋分散アルゴリズム
  ＋ディペンダブルコンピューティング

クラウドにおけるNoSQL



NoSQLの構成



NoSQLのデータ（1）

- ロー指向
  - 複数の属性（バリュー）からなるタプル (value1, value2, value3,…..)
- カラム指向
  - キーとバリューのタプル (key, value)
    - 同一キーのカラムデータを組合せてRDBと同等の表を構成可能

Key	Value1
東京	東京
神奈川	横浜
愛知	名古屋
大阪	大阪
兵庫	神戸

+

Key	Value2
東京	関東
神奈川	関東
愛知	中部
大阪	関西
兵庫	関西

Key	Value1	Value2
東京	東京	関東
神奈川	横浜	関東
愛知	名古屋	中部
大阪	大阪	関西
兵庫	神戸	関西

NoSQLのデータ（2）

- より柔軟なスキーマ設計
  - （例）ネステッドリレーション

Key	Value
東京	Key: 首長 / 石原 花 / ソメイヨシノ
神奈川	Key: 人口 / 9M 花 / ヤマユリ
大阪	Key: 人口 / 8.9M 首長 / 橋下 花 / ウメ

NoSQLのデータ（3）

- 索引
  - キー・バリューストアでは、キーからのみ絞り込み検索が可能
    - バリューから検索するには？
      - アプリケーション側で（転置）索引を作成する必要がある

Key	Value1
東京	東京
神奈川	横浜
愛知	名古屋
大阪	大阪
兵庫	神戸

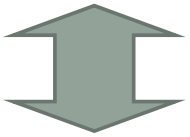
オリジナルデータ

Key	Value2
東京	東京
横浜	神奈川
名古屋	愛知
大阪	大阪
神戸	兵庫

転置索引
（ユーザが作成・維持管理）

BASEトランザクション

- スケールアウト前提の、NoSQLのデーター貫性のモデル
  - BAsically Available（基本的に高可用）
  - Soft-state（緩い状態の許容）
  - Eventually consistent（結果整合性）
    - 一時的にデータが一貫していないかも知れないが、時間が経過すれば一貫した状態に
      - 検索エンジン、SNS等ではそれほど問題とならない



- RDBMSのデーター貫性のモデル
  - ACIDトランザクション：　Atomicity（原子性）、Consistency（一貫性）、Isolation（分離性）、Durability（持続性）
    - 高可用、厳格な状態、データは常に一貫

分散システムの限界

- CAP定理
  - Consistency（一貫性）
  - Availability（可用性）
  - Partition tolerance（ネットワークの分断耐性）

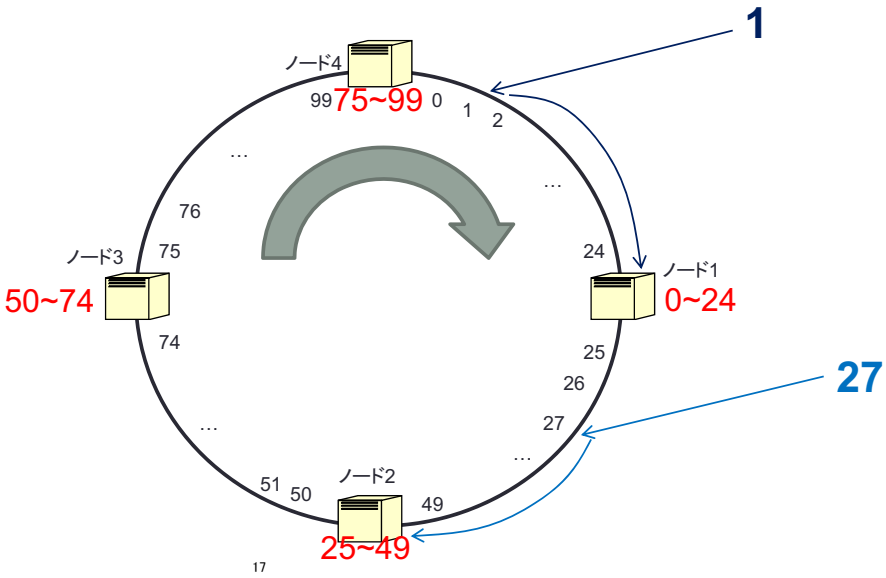
  の三つの要求を同時には満たすことはできない

- NoSQLについて解釈すれば、
  - もしネットワークの分断が発生すれば、一貫性もしくは可用性を選択
    - 選択肢は基本的アプリケーション依存
    - BASEトランザクションを使う場合は、可用性を重視したアプリケーションに有効

NOSQLを支える技術

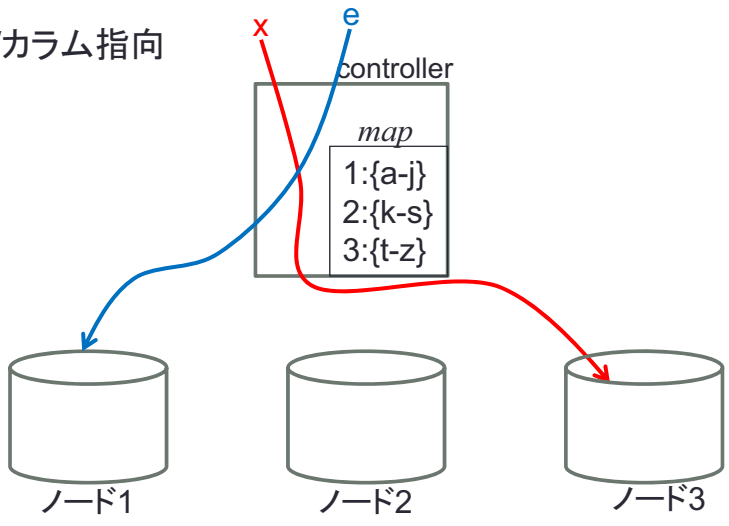
データ分散（1）

- P2P技術の利用
  - Consistent hashing (Amazon Dynamo, Apache Cassandra)
  - 基本的に、キー・バリューストア（カラム指向データ）
  - 非集中管理



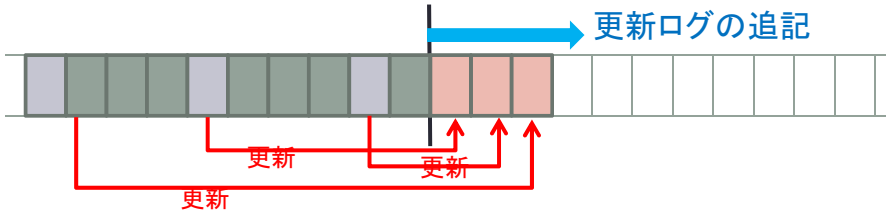
データ分散（2）

- Sharding
  - 基本的に、従来のデータパーティショニング(Google Bigtable, Yahoo! PNUTS)
  - 集中管理
  - ロー指向/カラム指向

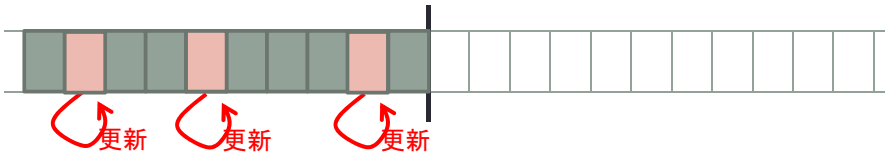


I/O

- 書込み指向
  - Logベース更新 (log-structuredファイルシステムに類似)
    - データ更新時：更新情報をディスクにシーケンシャルに追記
    - データ読出し時：ランダムアクセスが発生



- 読出し指向
  - In-place更新（従来のファイルシステム）
    - データ更新時：ランダムアクセスが発生
    - データ読出し時：シーケンシャルアクセス



NoSQLにおける一貫性と可用性

- (大規模)分散システムでは、故障は必ずどこかで起きる
  - 可用性を上げるには、異なるノードにデータの複製を書込み
    - 複数のノードにデータを書き込むのはオーバヘッド大

  - 一貫性を上げるには、データの複製間での制御が必要
    - 複製の制御を行うのはオーバヘッド大

- 可用性と一貫性のどちらを取るか？
  - アプリケーションの性質に応じて選択

複製 (replication)

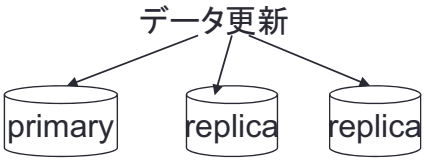
- データの複製（replica）を複数のサイトに配置
- ユーザは複製の存在を意識しなくてよい（複製透過性）
  - 利点
    - あるサイトの障害でも複製を持つサイトで問合せが可能 （耐障害性）
    - 特定のデータに問合せが集中する場合に、複製を持つサイトにその問合せを分散可能 （負荷分散）
  - 欠点
    - 同時に異なるサイトで同じ複製を書き換える場合、複製間の一貫性維持が容易ではない

one-copy serializability (1SR)

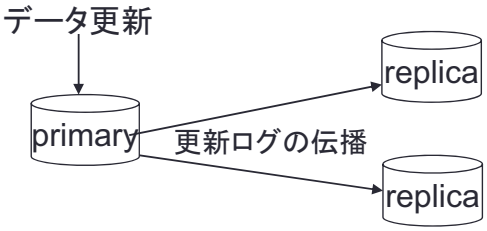
- one-copy serializability
  - あるデータに対して複数の複製が存在しても、あたかも唯一のデータしか存在しないかのように動作すること
    - one-copy serializabilityを達成するための制御方法が複製制御(replica control)

データの複製

- 全てのノードでデータ更新を同期書込み
  - 個々のデータは一貫
  - 同期オーバヘッドがクラウドに向かない



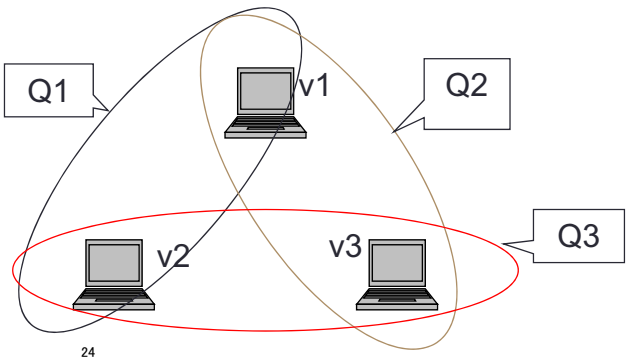
- Primary replicaでのみでデータ更新を書込
  - オーバヘッドが小さく、クラウド向き
    - 更新ログを他のreplicaサーバに非同期転送
  - 時間が経てばデータは一貫
    → eventually consistent



クォーラム（1）

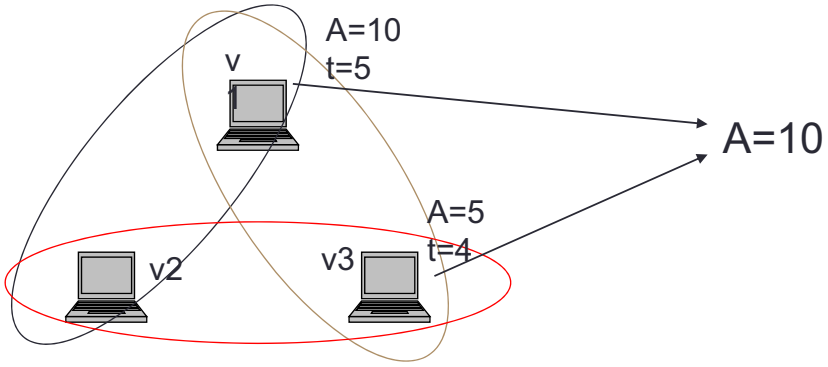
- quorum system
  - ノードの部分集合（ノード集合と呼ぶ）の集合Cにおいて、任意の二つのノード集合が一つ以上の共通ノードを持つシステム
    - $Qi \cap Qj \neq \phi$ for $\forall Qi,Qj \in C$ $(i \neq j)$
  - いかなるノード集合も他のノード集合の真部分集合ではない
    - $Qi \subseteq Qj$ for $\forall Qi,Qj \in C$ $(i \neq j)$

C={{v1,v2},{v2,v3},{v3,v1}}



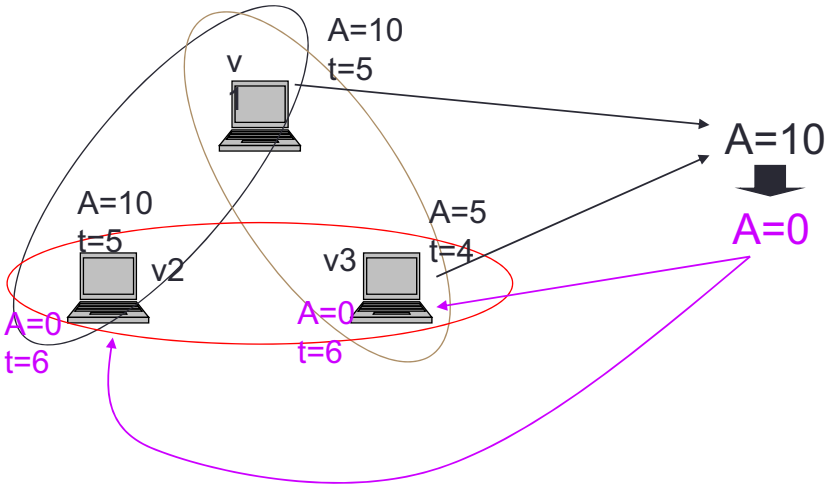
クォーラム(2)

- 読出し任意の正常な一つのクォーラム中で最新のデータを読み出す



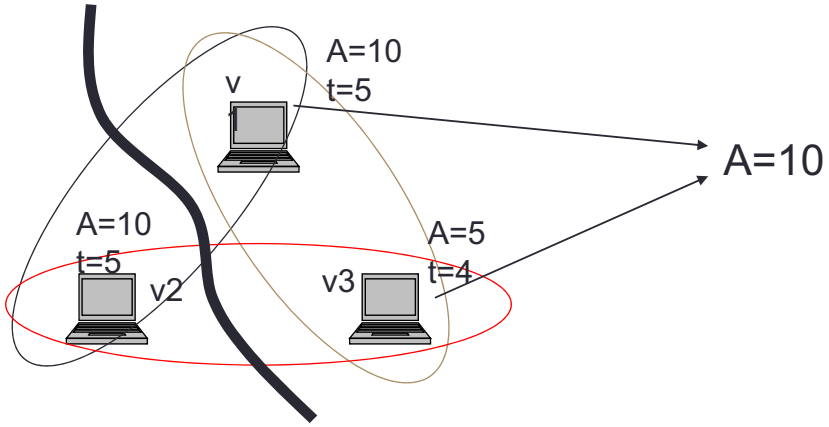
クォーラム(3)

- 書込み
  - 読出しの後、任意の正常な一つのクォーラムに、より新しいタイムスタンプを付けて書込む
  - 過半数以上のノードに書込むことに相当



クォーラム（4）

- ネットワーク分断耐性
  - ネットワークの分断時でも、正常なクォーラムからの読出し、正常なクォーラムへの書込みで一貫したデータが読み書きできる



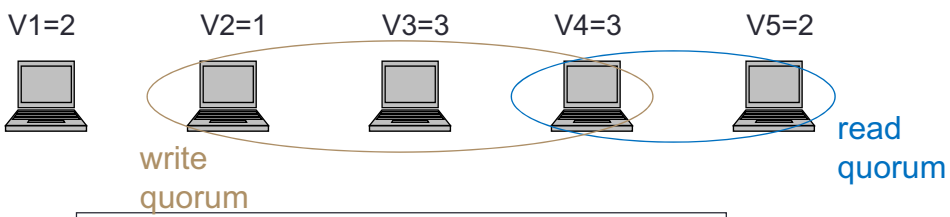
Read/Writeクォーラム（1）

- 各ノードに票(vote)を割当てる
  - 票の総和 V = ΣVi に対して、

    Vr + Vw > V

    Vw > V/2

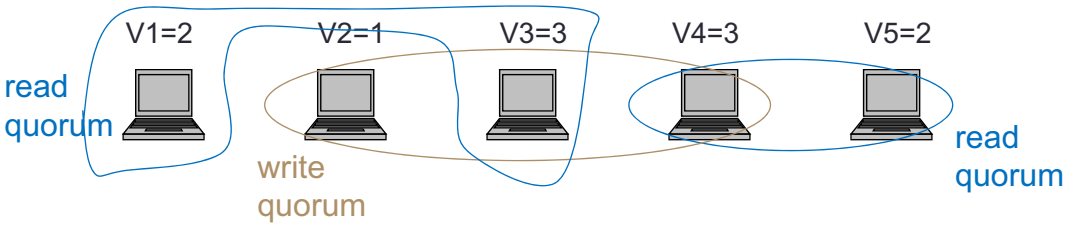
    となるようRead QuorumとWrite Quorumを構成



V=11に対して、Vr=5, Vw=7を設定

Read/Writeクォーラム（2）

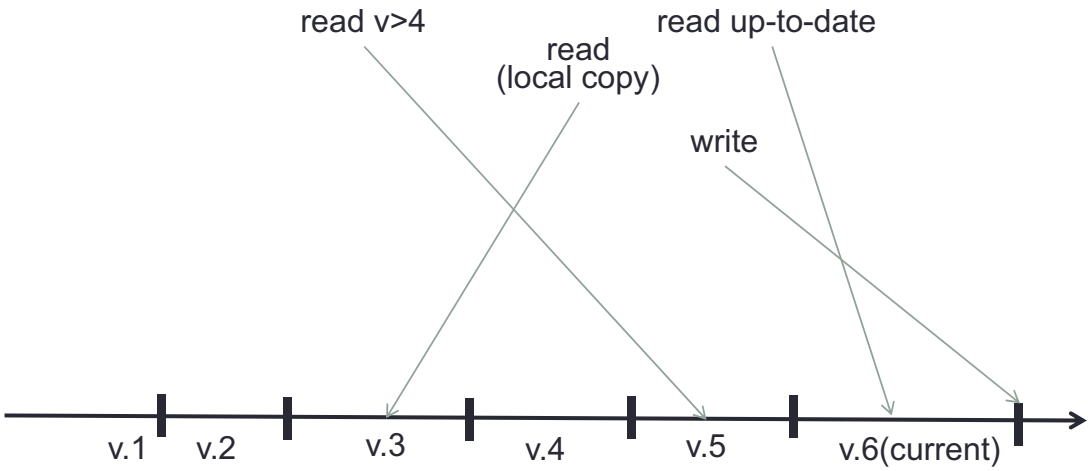
- Read Quorum(Ri)とWrite Quorum(Wj)に関して
  - Ri∩Wj≠ϕ
    - 読み出すデータには必ず最新のものが存在
  - Wi∩Wj≠ϕ (i≠j)
    - タイムスタンプが同一の異なるデータは書き込まれない



票の割当て方により
読み出し/書込みの負荷を調整

Timeline一貫性モデル

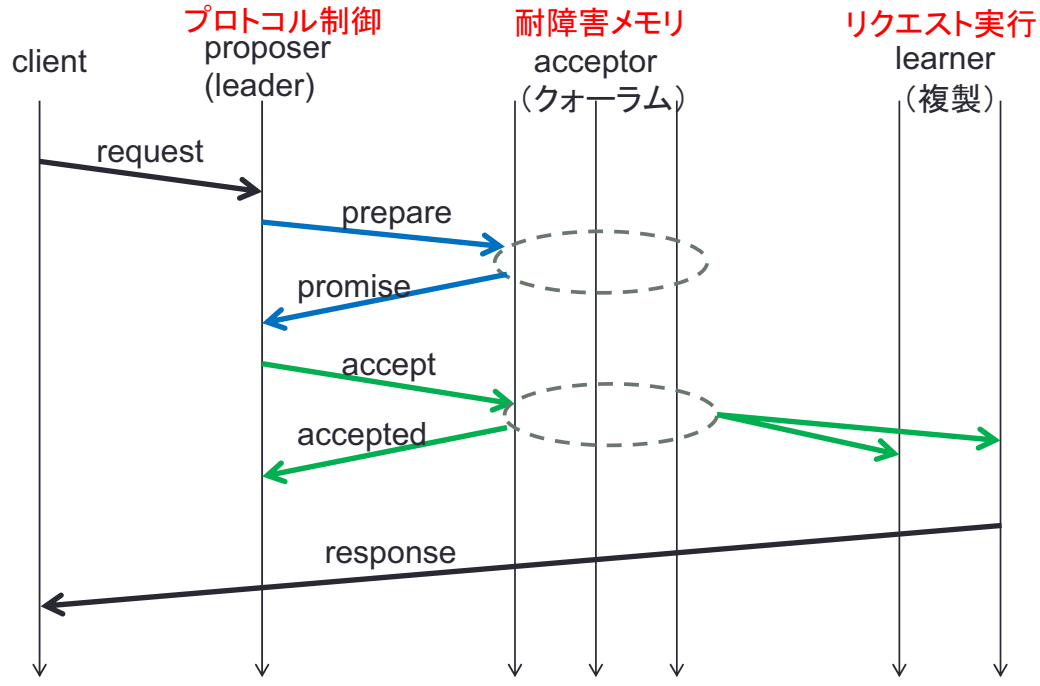
- Yahoo! PNUTSで使用される一貫性モデル
  - アプリケーションに応じて、異なる新鮮さのデータを読出せる



分散合意形成問題

- 故障する可能性のある複数のノード間で、ある値を一意に決定する問題
  - Paxosアルゴリズム
    - 2N+1個のノードのうち、最大N個故障しても正常に動作
    - 正常時は2フェーズで実行

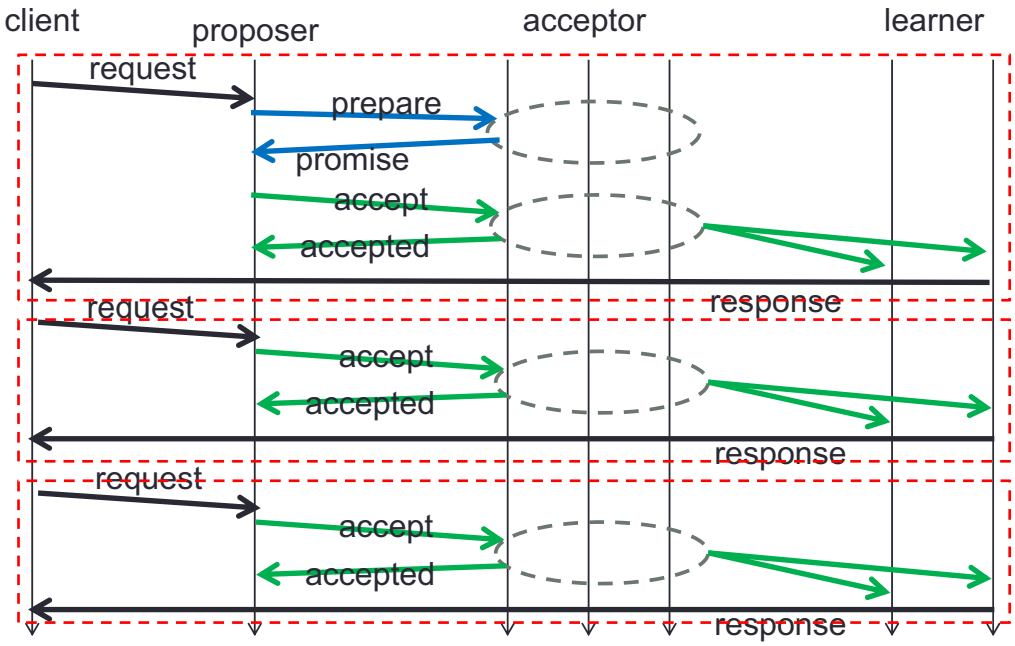
Paxos（1）



※ leader故障時は、投票で新leaderを決定

Paxos（2）

- Multi-Paxos
  - 同一インスタンスに対する2回目以降のリクエストのメッセージ数の削減



事例

主なNoSQLの比較

- Google Bigtable, Apache HBase
  - OLAP等の大規模なバッチ処理
    - カラム指向、書込み指向I/Oが有効
- Amazon Dynamo, Apache Cassandra
  - 電子商取引、SNS
    - 可用性が重要
- Yahoo! PNUTS
  - SNSなどのログイン・プロファイル読み出し
    - ロー指向、読み出し指向I/Oが有効

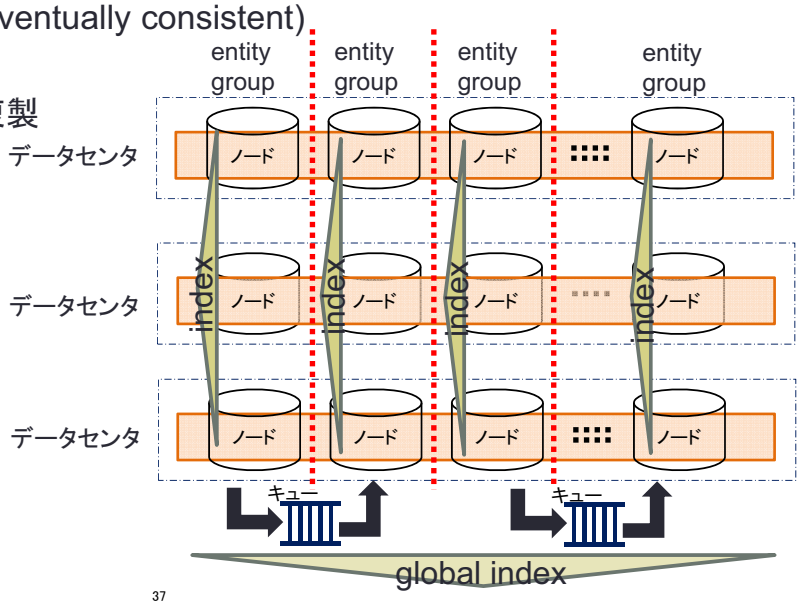
主なシステムの構成

構成要素	データ配置管理	テーブル構成	データ格納	I/O	高可用化
選択肢	集中型 (1, 4)	カラム指向 (1, 2, 3)	ファイルシステム/ tablet (1, 3, 4)	書込み指向 (1, 3)	複製 (1,4)
	非集中型 (2, 3)	ロー指向 (4)	DBMS (2, 4)	読出し指向 (2, 4)	クォーラム (2,3)

1: Google Bigtable, Apache HBase
2: Amazon Dynamo
3: Apache Cassandra
4: Yahoo! PNUTS

Google Megastore

- ACIDトランザクションのサポート
  - 有効範囲はentity group内のみ (entity group: 密な関係のデータ集合)
- 二次索引のサポート
  - local (ACID) / global (eventually consistent)
- 改良型Paxosの利用
  - トランザクションログの複製



NoSQLの研究の方向性

- 性能、可用性のさらなる向上
- 電力 vs. 性能
- Service Level Agreement
- CRUD以外の機能？
- etc

まとめ

- NoSQL
  - NoSQL概要
  - NoSQLで利用される技術
    - データベース技術
    - 分散アルゴリズム
    - ディペンダブルコンピューティング
  - 事例
  - 研究の方向性