論文紹介
Building a Database on S3

計算工学専攻　呉 怡 (09M38447)

論文情報

- “Building a Database on S3”
  - 著者
    - M Brantner, D Florescu, D Graf, D Kossmann, T Kraska
  - 出典：SIGMOD2008
  - 引用数：25(Google Scholar)
  - URL
    - http://portal.acm.org/citation.cfm?id=1376645&dl=ACM

発表の流れ

1. 背景
2. AWS(Amazon Web Services)
3. USING S3 AS A DISK
4. BASIC COMMIT PROTOCOLS
5. TRANSACTIONAL PROPERTIES
6. 評価実験
7. まとめと今後の課題

背景(1/2)

- YouTubeやGoogleのようにWebサービスを提供するのは難しい
  - コストの問題
  - スケラービリティと可用性の問題

- クラウドコンピューティング
  - スケーラビリティと完全な可用性
  - クライアント側の故障の影響を受けない
  - 応答時間が定数時間内であること
  - Pay by use
  - 個人でも簡単にWebサービスを構築できる

背景(2/2)

- Amazon S3の現状
  - サイズの大きいオブジェクトや更新しないデータによく使われる
    - 例：マルチメディアデータ、バックアップ
  - ローカルディスクに比べると遅いが、高いスケラービリティと可用性を持つ
  - 結果整合性しか保証しない
    - 結果整合性(Eventual Consistency)
    - 一貫性を必要とするアプリケーションには対応できない

目的

- クラウド型ストレージの利用範囲を一般的なウェブサービスに広げる
- Web-basedデータベースをS3で用いるためのプロトコルを提案
  - スケラービリティと有用性を保ちながら、データベースシステムにおける異なるレベルの一貫性を実現する
    - サイズの小さいオブジェクトの更新
    - B-tree
    - ACID 性
      - A: 原子性 (Atomicity)
      - C: 一貫性 (Consistency)
      - I: 分離性 (Isolation)
      - D: 持続性 (Durability)

AWS
S3 - Simple Storage Service (1/2)

- 概要
  - サイズ1B~5GBのオブジェクトを蓄積可能
    - オブジェクトはキーによって識別される
    - get(uri), put(uri, bytestream), get-if-modified-since(uri, timestamp)
    - オブジェクトに独自のメタデータを付与することができる
  - 容量無制限 + スケーラブル + 有用性 + スループット

- バケット
  - オブジェクトは必ずあるbucketに属している
  - bucket単位で閲覧権限を設定できる
  - URIで指定する

AWS
S3 - Simple Storage Service (2/2)

- 料金
  - $(70/160GB)/24 = 毎月$0.02/GB ⇔ 毎月$0.15/GB
  - $0.01 per 10,000 get, $0.01 per 1,000 put + データ転送料金
- レイテンシ
  - ローカルディスクの約2～3倍
  - Readは100msec以上で、Writeの場合はReadの約3倍
- バンド幅
  - 100KB以上のサイズでないと十分な帯域幅が得られない
    - データ転送はページ単位で行われるため

AWS
SQS - Simple Queue Service (1/2)

- 概要
  - サイズなどに関する制限のないメッセージ・キューを提供
    - 主にAWS間のメッセージ通信に用いられている
    - 4日間を経過したキュー内のメッセージは自動的に削除される
  - メッセージに含められるテキストデータのサイズは最大8 KB
  - creatQueue(uri), send(uri, msg),
    receive(uri, number-of-msg, timeout)

- 料金
  - $0.01 per 10,000 requests + データ転送料金

AWS
SQS - Simple Queue Service (2/2)

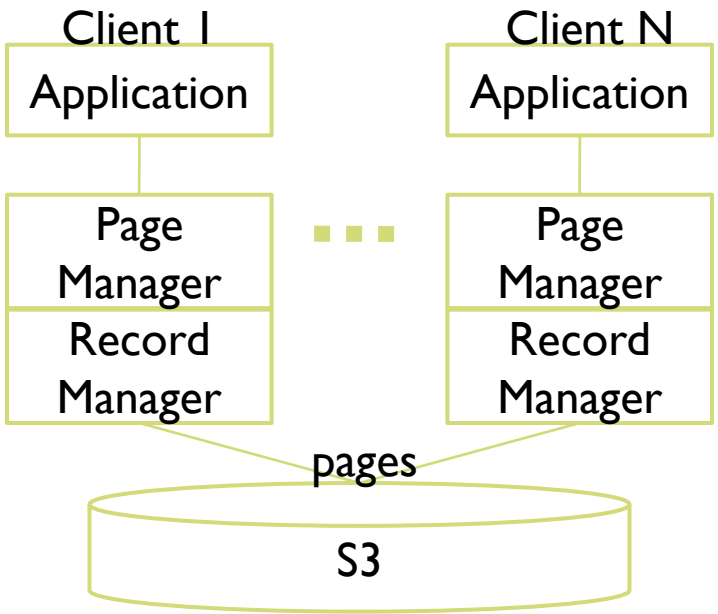
- スケーラビリティー
  - SQSは全てのリクエストに対して、定数時間内にackまたはメッセージを返す

Operation	Round Trip Time [secs]
send	0.31
receive	0.16
delect	0.16

  - SQSはできるだけFIFOでメッセージを返すが、正しい順番であることを保証しない
  - リクエストされる全てのメッセージを返すとは限らない
    - 200メッセージを含むキューに対して、トップ100のメッセージを要求すると、約20のメッセージが返される

USING S3 AS A DISK

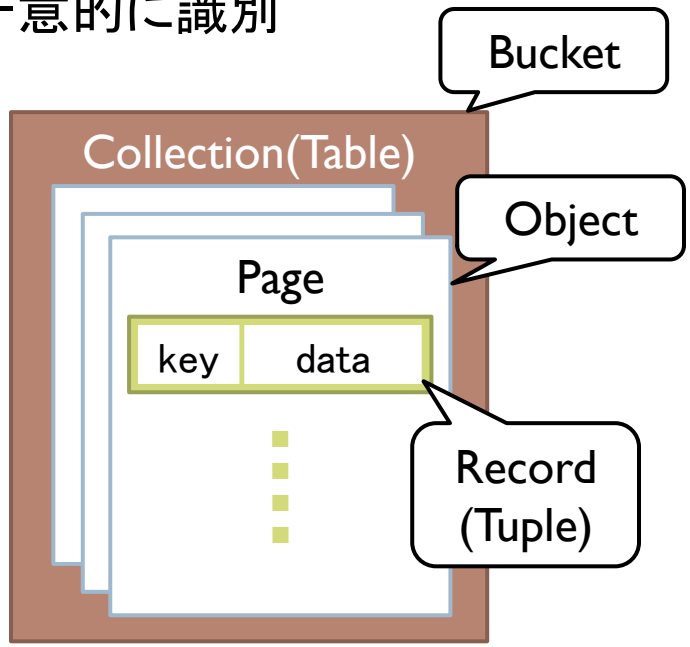
- クライアント・サーバアーキテクチャ
  - URIからページを取得し、ローカルで読み書きを行なって、ライトバックする
    - ページにはレコードや索引などが含まれる
  - レコードは可変長のバイトストリームで表される
    - 関係タプル、XML文書など
  - レコードマネージャー
    - レコードをページにしまう
    - 新規レコードのためのスペースの確保
  - ページマネージャー
    - S3にR/Wリクエストを出す
    - 送信されてきたページをローカルに一時保存する



goi@de.cs.titech.ac.jp

レコードマネージャー

- レコード
  - キーとデータからなる
    - レコードサイズはページサイズに制限される
  - コレクションと関連付けられている
    - キーでコレクションの中からレコードを一意的に識別
    - コレクションはURIによって指定できる
- レコードマネージャー
  - レコードのcreate, read, update
  - コレクションのスキャン



ページマネージャー

- S3とデータをやり取りするためのバッファプール
  - S3からページをread, update, createする
    - ライトセット(write set)はクライアントのMM、または二次記憶装置に収まる必要がある
- COMMITとABORTメソッド
  - COMMIT
    - S3に変更を転送し、関連ページにunmodifiedをつける
  - ABORT
    - 関連ページを破棄する
- TTLプロトコルの利用
  - TTLの切れたページに対し、S3から最新のバージョンを取得する

B-tree

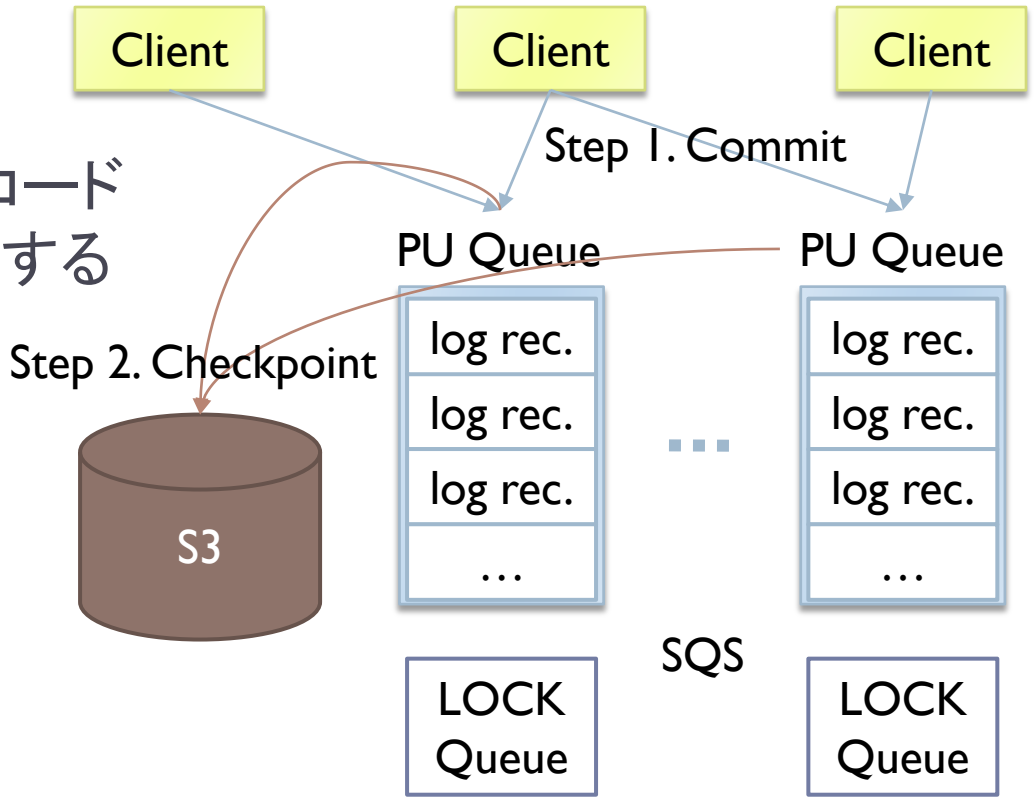
- ページマネージャーでB-treeを実装
- ルートと中間ノードはページとして格納される
  - Keyと次のレベルを指すURIを持つ
- 葉ノート
  - プライマリインデックスの葉ノードには(key, data)の形となるエントリーが含まれる
  - セカンダリインデックスの葉ノードには(search key, record key)の形のエントリーが含まれる
- 各レベルの隣接するノードはポインタでつないでいる
- ルートページのURIでB-treeを識別する
  - ルートのURIはいつでも同じ

ロキング

- 全てのログレコードが冪等(idempotent)であるとする
  - 同じログを複数回適用しても結果は同じ
- この論文では以下のredo log recordを使用
  - (insert, key, payload)
  - (delete, key)
  - (update, key, afterimage)

BASIC COMMIT PROTOCOLS

- 概要
  - クライアントがS3に対して更新をコミットするためのプロトコル
- 手順
  1. クライアントがログレコードを生成し、SQSに送信する
  2. 送信されたログをS3にあるページに適用



PUキュー

- Pending Update Queue

- １つのB-treeが１つのPUキューに関連付けられている
  - B-tree に対応するPU Queueは木と同時に作られる
  - PUキューのURIはルートノードのURIから求められる
  - B-treeに対する全てのinsertとdeleteログがPUキューに送信される

- プライマリB-treeの各葉ノードにも対応するPUキューを持っている

Checkpoint Protocol for Data Pages

- updateログのcheckpoint処理に関するプロトコル
  - 入力：PUキュー
  - 同時に複数のクライアントがcheckpointを行うことはできない
    - 一部のupdateは上書きされるかもしれない
  - LOCKキュー
    - PUキューと同様に、ページがcreateしたときに作られる
    - あるPUキューと関連付けられている
  - 処理
    - まずLOCKキューからトークンメッセージを取得し、そのメッセージをロックする
    - timeout時間内に、checkpointを終了させる必要がある

Checkpoint Protocol for B-trees

- 処理手順
  1. LOCKキューからトークンを取得
  2. PUキューからログを取得し、ソートを行なう
  3. 最初のログが対応する葉ノードを検索し、get-if-modifiedメソッドで最新バージョンを読み込む
  4. その葉ノードに関連する全てのログを適用する
  5. タイムアウト時間内に処理が終われば、新しいノードをS3に格納する
  6. 時間がなければ、適用しようとしたログをPUキューから削除

Checkpoint Strategies(1/2)

- クライアントがいつでもcheckpointを行なうようにするプロトコル
- あるページ(または索引)Xに対するcheckpointは以下の権限で行なうとする
  - Reader、Writer、Watchdog、Owner

- Writer
  - 各データページのヘッダーには最後にcheckpointを行なったタイムスタンプを保持している
  - タイムスタンプと自分の時計の差がしきい値(checkpoint interval)以上ならコミット

Checkpoint Strategies(2/2)

- Reader
  - checkpoint がしばらく起こっていないオブジェクトに対して、1/x の確率でcheckpointを行なう
    - x > checkpoint interval
  - トランザクションとチェックポイントは非同期で行われる

TRANSACTIONAL PROPERTIES (1/3)

- Durability
  - 成功して終了したトランザクションの結果は、他のトランザクションによる明示的な変更がない限り、永久的に残る
  - SQSで実現
- Atomicity
  - トランザクションによるupdateはall or noneとなる
  - クライアントごとにATOMICキューを用意
    - PUキューに直接ログを送る代わりに、まずATOMICキューに送る
  - トランザクションidを使って、トランザクションに関する全てのログが送信できたら、commit(id)レコードをATOMICキューに送る

TRANSACTIONAL PROPERTIES (2/3)

- Consistency Levels
  - Strict Consistency（厳密一貫性）
  - Client-side Consistency
    - Monotonic Reads（単調読み込み整合性）
      - If a client process reads the value of a data item x, any successive read operation on x by that client will always return the same value or a more recent value
      - ページのタイムスタンプを利用
    - Monotonic Writes（単調書き込み整合性）
      - A write operation by a client on data item x is completed before any successive write operation on x by the same client
      - ページ毎にカウンターを用意し、updateが来た場合カウンターを回す
      - (client, id, counter value)の情報を保持することで実現

TRANSACTIONAL PROPERTIES (3/3)

- Client-side Consistency（続き）
  - Read your writes
    - The effect of a write operation by a client on data item x will always be seen by a successive read operation on x by the same client
    - 単調読み込み整合性が保証されれば成り立つ
  - Write follows read
    - “A write operation by a client on data item x following a previous read operation on x by the same client, is guaranteed to take place on the same or a more recent value of x that was read
    - クライアントは直接データアイテムをwriteすることはないので、成り立つ

EXPERIMENTS & RESULTS

- 以下3つの異なるレベルの一貫性を前提に実験を行なう
  - Basic
    - 結果整合性のみ保証
  - Monotonicity
    - クライアント側一貫性を保証
  - Atomicity
    - トランザクションの原始性を保証
  - BaseLine
    - 一貫性を保証しない、直接S3のページに書き込む

TPC-Wベンチマーク

- TPC-Wベンチマーク
  - TPC (Transaction Processing Performance Council：「トランザクション処理性能評議会」)によって制定されたWeb コマース向けベンチマーク
  - オンライン書店において、商品に対する問い合わせ及び注文を行なう

- 実験方法
  1. DBにある顧客レコードを検索
  2. 6件の商品を検索
  3. 6商品のうち3件を注文

実験結果：実行時間

- 一貫性のレベルが高いほど、実行が短い
  - BaseLine：トランザクションがある度に、S3でコミットが起こる
  - SQSにログを送るだけなので、S3より早い
    - ログレコードのサイズがページサイズより小さい
    - SQSの高いスケーラビリティと有用性
    - Atomicity：SQSに送信するログレコードの量が少なくなる

	Avg.	*Max.*
Naïve	11.3	12.1
Basic	4.0	5.9
Monotonicity	4.0	6.8
Atomicity	2.8	4.6

Table 3: Running Time per Transaction [secs]

実験結果：コスト（$）

- 一貫性のレベルが上がるにつれ、コストが増加する
  - Atomicityプロトコルにおけるcheckpoint処理が最もコストが高い

	Total	*Chckp. + Atomic Q.*	Transaction
Naïve	0.15	0	0.15
Basic	1.8	1.1	0.7
Monotonicity	2.1	1.4	0.7
Atomicity	2.9	2.6	0.3

Table 4: Cost per 1000 Transactions [$]

まとめと今後の課題

- まとめ
  - クラウド型ストレージは高速なトランザクション処理に対応できていない
  - この論文では、S3をデータベースシステムに用いるためのプロトコルを提案
    - 完全な一貫性やDBトランザクションの特徴などをあきらめた

- 今後の課題
  - スケーラビリティと有用性よりも、ACID性を求める場合
  - 新しいJOINや最適化の手法が必要
  - セキュリティーの問題