Building a Database on S3

By

Sandeepkrishnan

Introduction

- Next wave on the Web to provide services.
- Make it easy for everyone to provide services, not just Google, Amazon.
- Goal of utility computing
  - Storage, CPU, network bandwidth as a commodity at low unit cost.
  - Scalability not a problem.
  - Full availability at any time – never blocked.
  - Clients can fail at any time.
  - Response times constant (R/W).
  - Pay by use.

- Most prominent utility service is S3
  - Part of Amazon Web Services AWS
    - S3, SQS, EC2, SimpleDB.
- S3 is Amazon's simple storage service
  - Infinite scalability, high availability, low cost
  - Currently for multi-media documents.
  - For data rarely updated
    - Smugmug implemented on top of S3.
    - S3 popular as a backup.
    - Products to backup data from MySQL to S3.
- But will S3 work for other kids of data?

- Disadvantages of S3
  - Slow compared to local disk drive.
  - Sacrifice consistency – undetermined amount of time to update object.
  - Updates not applied in same order as initiated.
  - Eventual consistency is only guarantee.

S3

- S3 – Simple Storage System.
- Infinite store for objects from 1B to 5GB.
  - Object is a byte container identified by URI.
  - Can Read/Update with SOAP or REST-based interface.
    - Get(uri): returns object
    - Put(uri, bytestream): writes new version.
    - Get-if-modified-since(uri,TS) gets new version if object changes since TS.

- In S3, each object is associated to a bucket
  - Users can specify in to which bucket new object is to be placed.
  - Users can retrieve all objects of a bucket or individual objects.
  - Use buckets as unit of security.

  Issues:
  - Latency is a problem, reading takes 100msecs.
  - Data has to be read in larger chunks.
  - S3 is not free
    - $0.15 to store 1GB of data per month.
    - $.01 per 10K get requests, per 1K put requests.

SQS

- SQS – Simple Queuing system
  - Allows users to manage an infinite number of queues with infinite capacity.
  - Each queue referenced by a URI, supports send/ receive messages via HTTP or REST-based interface.
  - Size of message 8KB for HTTP
  - Supports: Create Queue, Send message to Queue, Receive messages, Delete message, Grant another user send/receive messages etc.
  - Cost of SQS - $.01 to sent 1K messages.

EC2

- EC2 – Elastic Computing Cloud
  - Allows renting machines (CPU + Disk) for specified period of time.
  - Client gets virtual machine hosted on Amazon server.
  - $0.10 per hour regardless of how heavily machine used.
  - All requests from EC2 to S3 and SQS are free.

Using S3 as a disk

- Client-Server Architecture
  - Similar to distributed shared-disk DB systems.
  - Client retrieves pages from S3, based on URIs, buffers them locally, updates them, writes them back.
  - Record is bytestream of variable size (constrained by page size).
  - Focus on:
    - page manager – coordinates R/W, buffers pages.
    - Record manager – record oriented interface, organizes records on pages, free-space management.

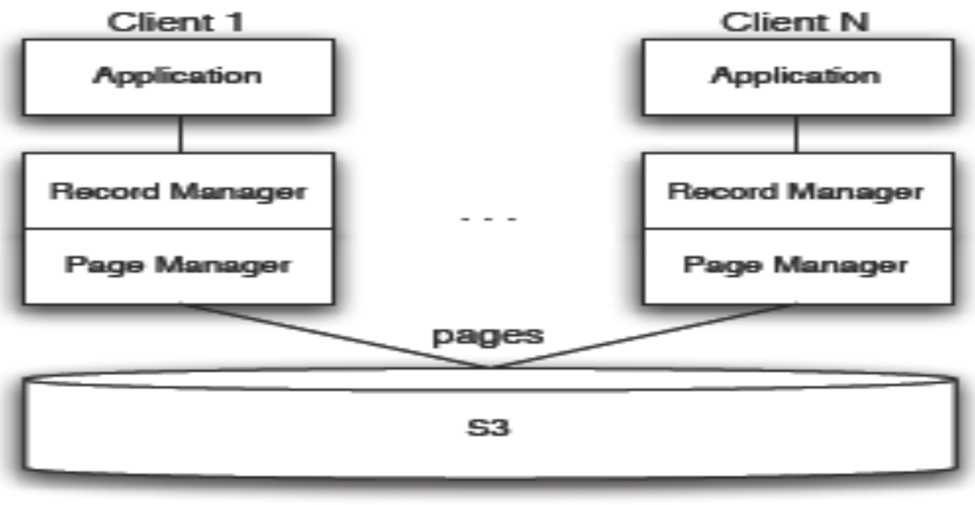


Figure 1: Shared-disk Architecture

Page manager, record manager, etc. could be executed on EC2 or whole client stack installed on laptops or mobile phones to implement Web 2.0 application.

Record Manager

- Record Manager manages Records
  - Each record is associated with a collection.
  - Record is composed of key and data.
  - Record is stored in one page, pages stored as single object.
  - Collection is implemented as a bucket in S3.
  - Collection is identified by URI.
  - Function to Create a new record : Create(key, payload, uri).
  - Read record based on key : Read(key, uri).

- Update based on key : Update(key, payload, uri).
- Delete based on key : Delete(key, uri).
- Scan : Scan(uri).
- Also support Commit & Abort.

Page manager

- Implements buffer pool for S3 pages.
- Supports reading, updating, marking as updated, creating new pages on S3.
- Implements commit and abort.
- Commit must propagate changes to S3.
- If abort, discard from client’s buffer pool.
- No pages evicted from buffer pool as part of commit.
- Pages are refreshed using TTL.

B-tree indexes

- Root, intermediate nodes stored as pages with (key, uri of next level).
- Leaf pages of primary index have (key, payload data).
  - Store records of the collection.
- Leaf pages of secondary index have (search key, record key).
  - Retrieve keys of matching records, go to primary index to retrieve records with payload data.
- Nodes at each level are chained.
- Root page should always be referenced same URI (splitting root node).

Logging

- Use traditional strategies
  - Make extensive use of redo log records.
  - Insert log, delete log record, update log record associated with a data page etc.
  - Redo logging - log records are idempotent – can apply more than once with same result.
  - If operation involves updates to a record and updates to one or several secondary index the separate log records are created .

Security

- Everybody has access to S3.
- S3 gives clients control of the data.
- Client who owns a collection, can give other clients R/W privileges to collection (bucket) or individual pages of that collection.
- Cannot do SQL views – but can be implemented on top of S3.
- If provider not trusted, can encrypt data.
- Can assign a curator for a collection to approve all updates.

Basic Commit Protocols

- Updates by one client can be overwritten by another even if 2 are updating different records on same page.
- Unit of transfer is a page rather than record.
- Last update wins in the case of concurrent updates.

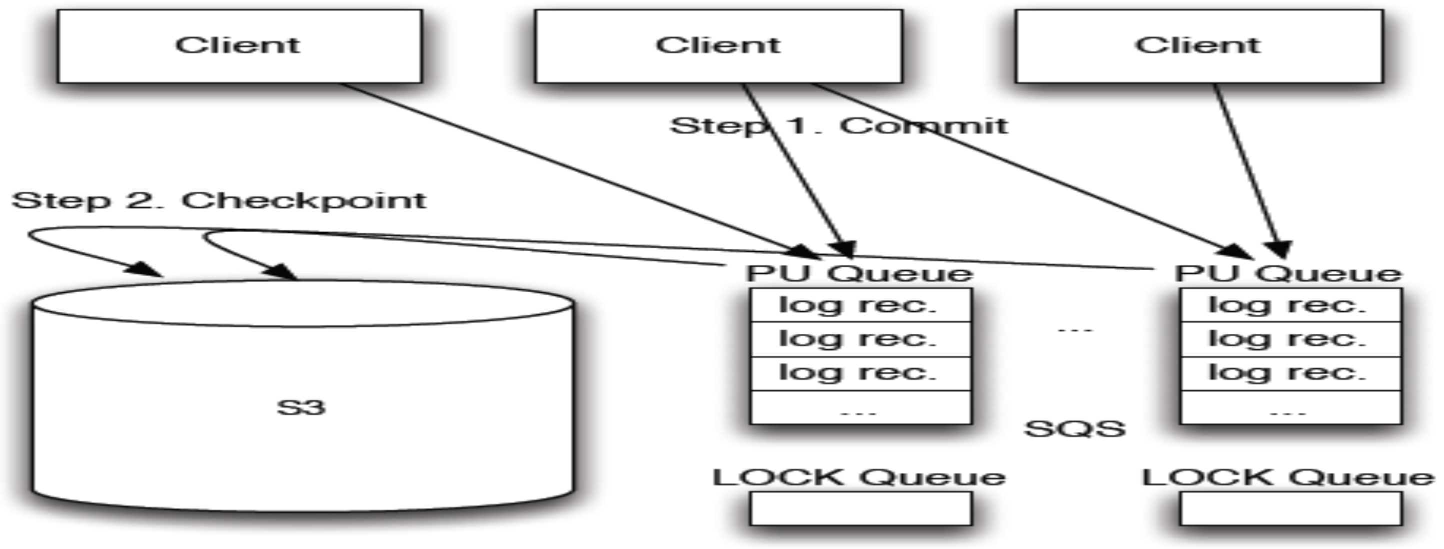


Figure 2: Basic Commit Protocol

- Client generates log records for all updates committed and sends to SQS.
- Log records are applied to pages on S3 – called check pointing.
- First step is carried out in constant time.
- Second step involves synchronization. But can be carried out asynchronously.

PU Queues

- Pending Update queues - Clients propagate log records to PU Qs.

- A PU Q can be check pointed by only a single client at the same time.

- Each B-tree(primary & secondary) has one PU Q – URI is derived from URI of B-tree – insert and delete log records are submitted to PU Qs of B-tree .

- One PU Q associated with each leaf node(data pages) of a Primary B-tree of a collection – only update log records are submitted to PU Qs of data pages.

Checkpoint Protocol for Data Pages

- Checkpoint of update log records is executed.
- Input of a checkpoint is a PU Queue.
- Make sure no other client carrying out checkpoint concurrently.
- Associate a Lock Queue with a PU Queue.
- Receives a token from Lock Queue if then can lock object.
- Set time out, must complete checkpoint by then.
- Update log record are applied as follows:

  receive token from Lock queue.

  Refresh cached copy, if not cached get copy from S3.

  receive log records from PU queue.

  Apply log records to local copy.

  Put new version of page to S3 or terminate.

  Delete all log records from PU queue.

Checkpoint Protocol for B-trees

- Checkpoint of insert & delete log records.
- More complicated than check pointing a data page because several tree pages are involved.
  - Obtain token from Lock Queue.
  - Receive log records form the PU Queue.
  - Sort the log records by Key.
  - Find leaf node which is affected for first log record.
  - Apply all log records that are relevant to that leaf node.
  - Put new version to S3 or terminate.
  - Delete log records.
  - Continue if there are still unprocessed log records.

Transactional Properties

- Atomicity
  - Use Atomic queues associated with each client.
  - Commit logs to Atomic Qs rather than PU Qs.
  - Every log record has id of commit for client.

    Client commit to Atomic Q.

    Then send all log records to PU Q.

    Delete commit record from Atomic Q.
  - During failure

    Client check Atomic Q at restart.

    winners & losers.

    winners are log records which carry same id as commit records found in atomic Q.

    winners are propagated to PU Q & losers deleted immediately.

Consistency Levels

- Weaker Levels of Consistency(client side) is achieved.

  - Read you writes : Automatically satisfied.

  - Monotonic read : If read value of x, any successive read by client reads that or a more recent value.
    - Keep track of record of highest commit TS cached by client

  - Monotonic write – W to x is completed before any successive write to x by same client
    - Counter for each page, increment when update, keep track of counter & client id(in log record & page header), log records are ordered and out of order is detected.

Experiments and Results

- Experiments were conducted by implementing protocols discussed earlier.
- Two main metrics studied were latency and cost.
- Observed that s3 was found to beat conventional technology in all other metrics considered.

- Experiments were carried out with following increasing levels of consistency:
  - Basic: supports only eventual consistency
  - Monotonicity: Top of basic protocol. Supports full client side consistency (R and Ws, R your Ws and W follows R).
  - Atomicity: Top of basic protocol. Highest level of consistency supported.
  - Naive approach – write all dirty pages to S3 rather than 2 phase commit.
    - Not even eventual consistency – updates lost
  - All 4 variants support same interface at record manager. Implementation of R, W, create, index probe, abort same for all in page manager, record manager etc. Variants differ only in implementation of commits and checkpoints.

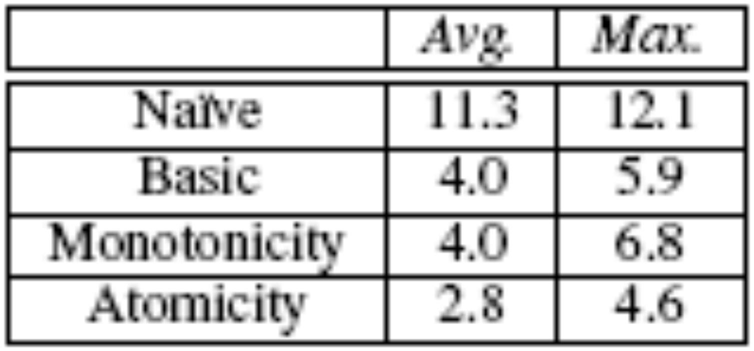
- Single client - Mac with 2.15 MHz Intel processor.
- Page size of data – 100KB.
- B-tree node size – 57 KB.
- TTL of client’s cache - 100 s.
- Cache size - 5 MB.
- 1 GB of network traffic - $0.18.

TCP-W Benchmark

- Models online bookstore with queries asking for availability of products, places orders
  - Retrieve customer record, search for 6 products, place orders for 3 products (random).
  - Study running time and cost of transactions for different consistency levels.

Running time

Average and maximum execution time in seconds per transaction are high

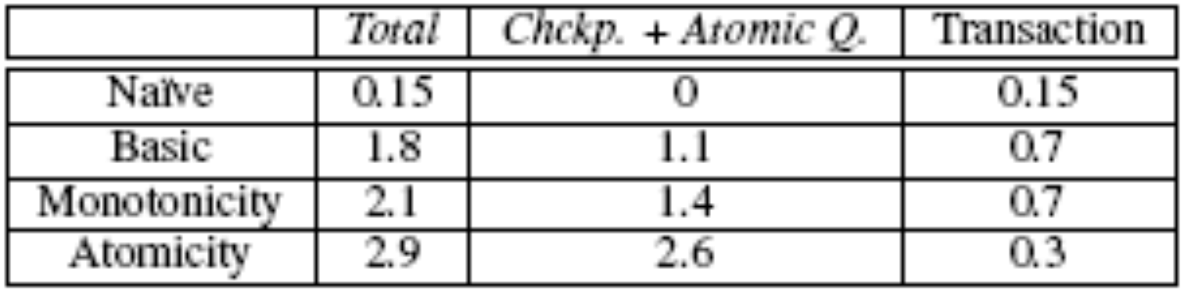


	Avg.	*Max.*
Naïve	11.3	12.1
Basic	4.0	5.9
Monotonicity	4.0	6.8
Atomicity	2.8	4.6

Table 3: Running Time per Transaction [secs]

- High execution times are expected. Believe results acceptable in interactive environment.
- Higher consistency means lower running time because of commit protocols.
- Naive has highest running time.
- Atomicity has fastest commit as it batches log records.

Cost ($)



	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 [$]

- Computed by running large number of transactions, taking cost measurements of AWS, dividing total cost by the number of transactions.

- < in latency, > in level of consistency, the cost increases.

- Highest level of consistency, cost is 20 times higher compared to naïve.

Cost ($)

- Interaction With SQS becomes expensive.

-Great deal of cost is spent to carryout checkpoints and to process atomic queue.

-For a bookstore, a transactional cost of about 0.3 cents is affordable.

- Updates have big influence on the cost.

- Not cheap but in many scenarios affordable.

Varying Checkpoint Interval

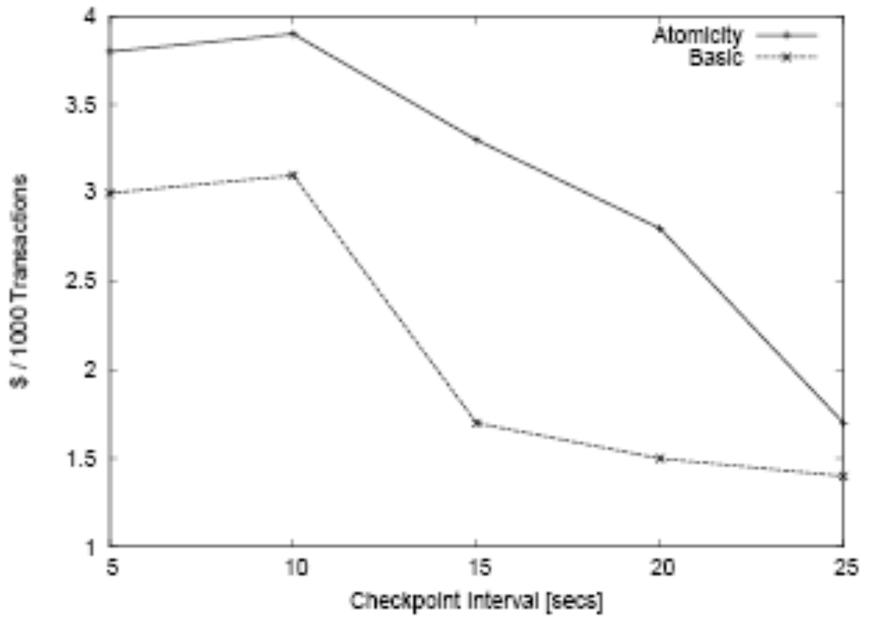


Figure 3: Cost per 1000 Transacts., Vary Checkpoint Interval

- Total cost as a function of checkpoint interval
- Increasing interval decreases cost
- Less than 10 seconds means checkpoint for every update. Does not make sense

Varying Checkpoint Interval

-Cost is reduced for checkpoint above 10 seconds.

- Curve flattens after about 20 seconds.

-At Infinity curve converges to about $7 per 1000 transactions.

- Best settings of checkpoint interval depends on the workload and on the skew in the update pattern.

- Also depends freshness of data the application requires.

Conclusions

- Utility computing is not attractive for high-performance transaction processing.
- Paper is first step for the above
  - Abandoned strict consistency and DB-style transactions.
  - May need ACID properties more than scalability and availability.
  - New algorithms for join, query optimization.
  - Need ways to Scan through several pages (chained I/O).
  - Need right security structure.