Building a Database on S3

[Ref: http://www.systems.ethz.ch/education/past-courses/fs09/HotDMS/pdf/dbons3.pdf]

Flow of the presentation

- Background
- Brief intro to S3, SQS, EC2.
- Discuss details of implementation.
- Discuss Costs, Performance and Results

Background

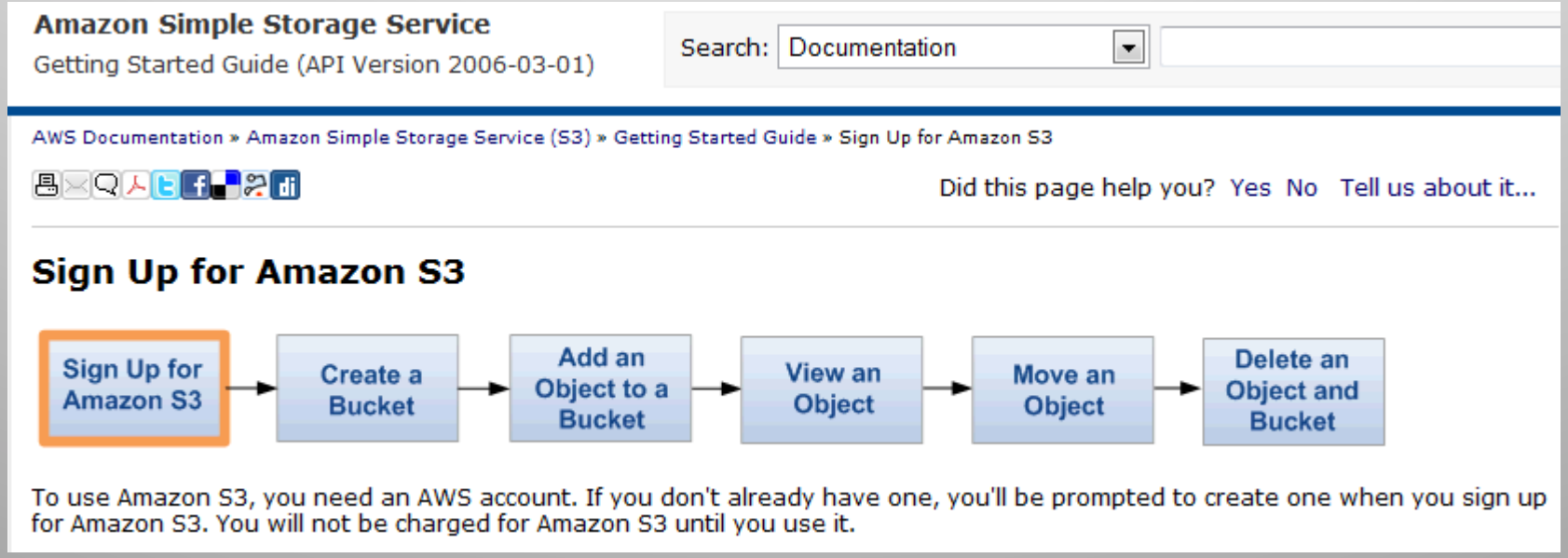
- Running a service becomes particularly challenging and expensive if the service is ‘successful’.
- Need to address cost to operate a service on the Web, ideally with 24-7 availability and acceptable latency.
- Required: Hosted server and a database which both need to be administrated; this paper focuses on the means to implement database component.
- Utility Computing provides cost effective answer for Storage, CPU, Network Bandwidth [User unaware of details] ; Infinitely Scalable, available. Consistent Response Time.

Components used for DB implementation.

AWS – Amazon Web Service

- S3 [Simple Storage System]
- SQS [Simple Queuing System]
- EC2 [Elastic Computing Cloud]
- SimpleDB

Accessing these Resources: Easy Setup in few steps at Amazon.com



Analogy

S3 [Simple Storage System]

SQS [Simple Queuing System]

What is S3

- Infinite store for objects of variable size ranging 1 Byte to 5 GB.
- Access via URI using SOAP/REST based interface.
- Methods e.g. get-if-modified-since enable caching based on a TTL protocol
- User defined metadata up to 4KB can be associated to an object and can be read and updated independently of the rest of the object.
- Object are associated to a bucket. Selective querying possible.
- Users can grant read and write authorization to other users for entire buckets.
- Alternatively, access privileges can be given on individual objects.

What is SQS

- SQS allows users to manage a infinite number of queues with infinite capacity.
- Each queue is referenced by a URI and supports sending and receiving messages via a HTTP or REST-based interface.
- The max. message size 256 KB for REST based interface
- And 8 KB for the HTTP interface.
- Any bytestream can be put into a message.
- Supported Methods: createQueue, send, receive, delete, addGrant.

COSTS

- **S3**: $0.15 to store 1 GB of data for one month.
- Seagate HDD 160 GB = $70 [In 2012 1.5TB costs $100]
- r/w $0.01 per 10,000 get & $0.01 per 1,000 put requests
- $.10-0.18/GB network bandwidth consumption - depending upon the total monthly volume therefore smugmug uses S3 as a persistent store.
- **SQS**: $0.01 to send 1,000 messages.
- network bandwidth $0.10 /GB of data Xferred.
- $0.10 per GB is the minimum for heavy users

Performance

- S3

Page Size [KB]	*Resp. Time [secs]*	*Bandwidth [KB/secs]*
10	0.14	71.4
100	0.45	222.2
1,000	3.87	258.4

Table 1: Resp. Time, Bandwidth of S3, Vary Page Size

- SQS

Operation	*Time [secs]*
send	0.31
receive	0.16
delete	0.16

Table 2: Response Times of SQS

Putting the Pieces together: Implementing the Database

Using S3 as a Disk

- Client Server Architecture
- Record Manager
- Page Manager
- B-tree Indexes
- Logging
- Security

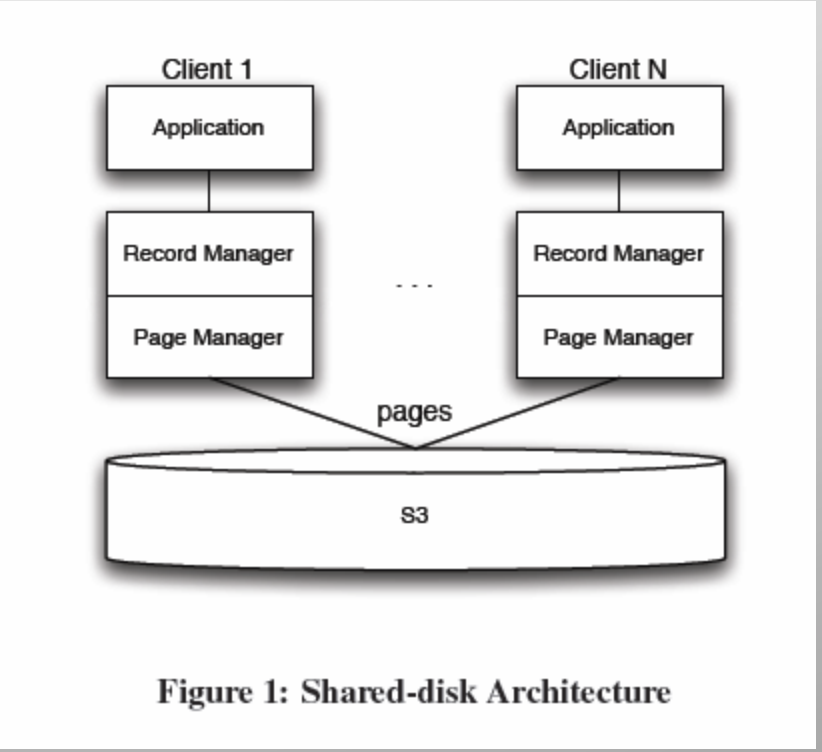


Figure 1: Shared-disk Architecture

Basic Commit Protocol

- Overview
- PU Queues
- Checkpoint Protocol for Data Pages
- Checkpoint Protocol for B-trees
- Checkpoint Strategies

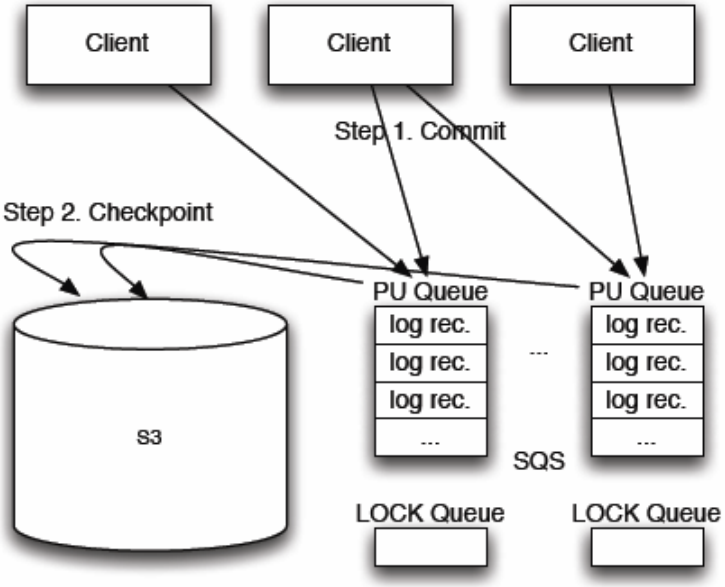


Figure 2: Basic Commit Protocol

TRANSACTIONAL PROPERTIES

- Atomicity
- Consistency Levels
- Isolation: The Limits

EXPERIMENTS

- Software and Hardware Used
- TPCW Benchmark
- Experiment 1: Running Time [secs]

	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]

EXPERIMENTS (cont…)

- Experiment 2: 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 [$]

EXPERIMENTS (cont…)

- Experiment 3: Vary Checkpoint Interval

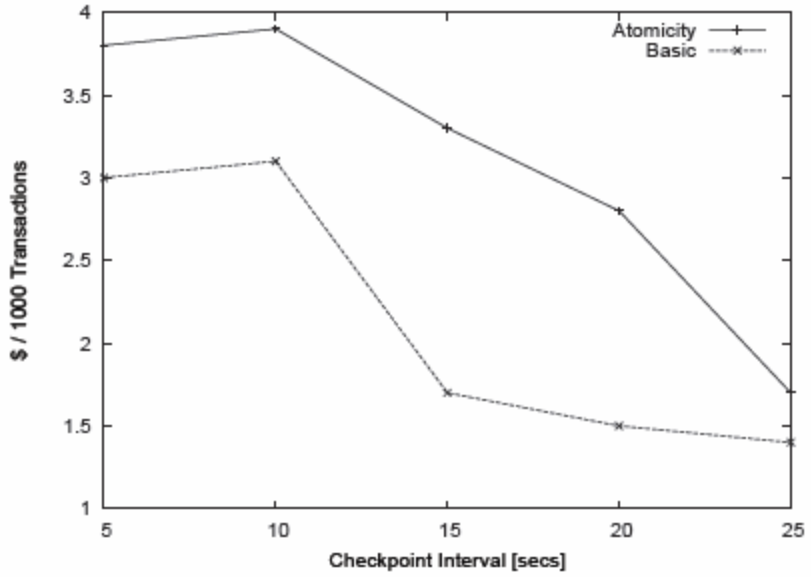


Figure 3: Cost per 1000 Transacts., Vary Checkpoint Interval

CONCLUSION

- Web-based applications need high scalability and availability at low and predictable cost.
- No client must ever be blocked by other clients accessing the same data or due to hardware failures at the service provider.
- Instead, clients expect constant and predictable response times when interacting with a Web-based service.
- Utility computing has the potential to meet all these requirements.
- Utility computing was initially designed for specific workloads.
- This paper showed the opportunities and limitations to apply utility computing to general-purpose workloads, using AWS and in particular S3 for storage as an example.
- As of today, utility computing is not attractive for high-performance transaction processing; such application scenarios are best supported by conventional database systems.
- Utility computing, however, is a viable candidate for many Web 2.0 and interactive applications