Cloud Computing
Towards Elastic Transactional Cloud Storage with Range Query Support

Saarbrücken, November 30th, 2010
Outline

• Motivation
• Related work
• System architecture of ecStore
  ▫ distributed storage layer (BATON)
  ▫ replication layer (self-tuning range histogram)
  ▫ transaction layer
• Performance study
• Conclusion
Motivation

• Cloud computing should be used as a utility

• Cloud storage has to be adjusted dynamically

• Minimal startup costs

• Pay-per-use model

• Elastically scale on-demand
  ▫ Allow users scale up and down on the fly

• Can only be archived when storage nodes could be easily added into or removed from the system
Outline

- **Motivation**
- **Related work**
- **System architecture of ecStore**
  - distributed storage layer (BATON)
  - replication layer (self-tuning range histogram)
  - transaction layer
- **Performance study**
- **Conclusion**
Related work

• Replication in distributed and peer-to-peer systems
  ▫ Primary copy of data is responsible to handle both read and write request from clients
  ▫ Only support operations on a single data items
  ▫ Data resided on a storage node is replicated on the successor node
  ▫ Pessimistic replication technique
Related work

- **Distributed and parallel databases**
  - B+-tree, optimistic scheme, two-phase commit protocol
  - Online load balancing in range-partitioned systems using data migration and self-tuning approach to re-organize the data in a shared-nothing system
  - Traditional parallel database technologies not fit 100% for scalable storage
Related work

• **Cloud data and transaction service**
  ▫ Data management system on top of the Amazon S3 based on the client-server model
  ▫ System with a not tightly coupled transactional component and a data component
  ▫ Storage nodes are organized on a ring-based distributed hash table (DHT) and each data item is asynchronously replicated on the successor storage nodes
Weaknesses of cloud storage services

- Guarantees on consistency (data updates)
- No range query support
- Data migration to balance the storage load
- No support transactional semantics across multiple keys
Outline

• Motivation
• Related work
• System architecture of ecStore
  ▫ distributed storage layer (BATON)
  ▫ replication layer (self-tuning range histogram)
  ▫ transaction layer
• Performance study
• Conclusion
ecStore (elastic cloud storage system)

- Scalable storage system within the cloud cluster
- The architecture follows a stratum design
- Organizes storage nodes as a balanced tree structured overlay and assigns a data range for each storage node

- Data objects are distributed and replicated in a cluster of commodity computer nodes
Architecture of ecStore

- Automated data partitioning and replication
- Load balancing
- Efficient range query
- Transactional access
Outline

• Motivation
• Related work
• System architecture of ecStore
  ▫ distributed storage layer (BATON)
  ▫ replication layer (self-tuning range histogram)
  ▫ transaction layer
• Performance study
• Conclusion
Distributed storage layer

- Distributed data structure
  - Decluster data objects across storage nodes
  - Facilitates parallelism to improve performance

- DHT-based structure (distributed hash table)
  - BATON (BAlance Tree Overlay Network)
BATON

- Tree-based structure
  - To realize a scalable range-partitioned system

- Support efficient range query processing

- Automatically repartition and redistribute the data when storage nodes are added into or removed from the system
Outline

• **Motivation**
• **Related work**
• **System architecture of ecStore**
  ▫ distributed storage layer (BATON)
  ▫ replication layer (self-tuning range histogram)
  ▫ transaction layer
• **Performance study**
• **Conclusion**
Replication layer

- BATON does not provide replication and transaction support
- Extend BATON to efficiently support load-adaptive replication for large-scale data
  - Two-tier partial replication strategy
    - Data availability
    - Load balancing function
- Tuning the replication process based on data popularity in common
  - Self-tuning range histogram
Replication in BATON

• Usually BATON is range instead of hash based
• “Where to replicate a certain data object?”

• Approaches
  ▫ Straightforward approach
  ▫ Replication based on data range
  ▫ Shift key value scheme (ecStore)
Replication in BATON

- **Straightforward approach**
  - Replicate data on the surrounding nodes
  - Replicas identified by the location of primary copy
  - It is complicated to identify the surrounding links of a failure node
Replication in BATON

• Replication based on data range
  ▫ If the key of a data item belongs to a certain range
    • Hash the range value
    • Use the output to determine the identity of the storage node where we can store the replica
  ▫ Hashing breaks the order of replicated data
Replication in BATON

- **Shift key value scheme (ecStore)**
  - Different replicas will be stored in the same BATON structure of the primary copy but associated with their virtual keys
  - Well distributed across the storage nodes in the cluster
  - Shifting the initial key to multiple virtual keys
  - Preserves the order of replicated data
Two-tier partial replication

• “Which data should be replicated?”

• Approaches
  ▫ Straightforward approach
  ▫ Data migration
  ▫ Two-tier replication mechanism (ecStore)
Two-tier partial replication

• **Straightforward approach**
  ▫ Replicate all data objects with the same replication level $K$
  ▫ If $K$ is large, the system storage and the overhead to keep them consistent can be considerably high

• **Data migration**
  ▫ Migrating hot data from one overloaded node to another node only shuffles the hotspot throughout the system
Two-tier partial replication

• Two-tier replication mechanism (ecStore)
  ▫ Provide both data availability and load balancing
  ▫ Each data object is associated with two kinds of replicas – secondary and slave replicas
Two-tier partial replication

- **First tier**
  - Small level K replication for all data objects

- **Second tier**
  - Popular data objects are associated with additional replicas – called slave replicas
  - Facilitate load balancing for frequently accessed objects
BATON - Two-tier partial replication

Diagram showing a tree structure with nodes labeled from a to t, each associated with a range of values and some notes on replication levels.

- Secondary replica of data 62 with virtual key = 12
- Slave replica
- Primary copy of data key 62
- Secondary replica of data 62 with virtual key = 37
- Secondary replica of data 62 with virtual key = 87
Self-tuning range histogram

- Only a small number of replicas
  - Histogram maintenance cost minimal

- Histogram to approximately estimate the access frequency of a data range
- When load balancing process is triggered, the storage node will replicate most popular data ranges to other lighter-loaded nodes
- Piggy-back the load information on the query
Self-tuning range histogram

• Dynamically restructuring the histogram
  ▫ Splitting/merging the buckets

• Total number of buckets is kept constant
  ▫ Merge consecutive buckets with similar frequency into a bucket with a larger data range
  ▫ Split the bucket with high access frequency into buckets with smaller data range

• Only replicate the data ranges maintained by small buckets
Self-tuning range histogram

- Reduce the cost of maintaining unnecessary replicas
- No benefits for load balancing anymore
  - Discard slave replica of a data range
Replica consistency management

- Cloud storage has provided 24x7 data availability
- Updating all copies synchronously is not suitable

- Pessimistic replication technique
  - Update needs to be reflected on all replicas before coming to effect

- Optimistic replication method (ecStore)
  - Primary copy is always updated immediately
Replica consistency management

- Write-ahead logging scheme
- Guarantees that updates to the primary copy are durable and eventually propagated to the secondary copies

- Adaptive read consistency by using the quorum model for read operations
- Write request will update primary copy first and asynchronously propagate it to the replicas
Replica consistency Management

• Adopts the notion of BASE (BAsically available, Soft state, Eventually consistency)

• Does not need to implement the two-phase commit protocol for refresh transaction
Outline

- Motivation
- Related work
- System architecture of ecStore
  - distributed storage layer (BATON)
  - replication layer (self-tuning range histogram)
  - transaction layer
- Performance study
- Conclusion
Transaction management layer

- **Multi-versioning**
  - Enhances the performance of read-dominant apps
  - Can benefit the read-only transactions

- **Optimistic concurrency control**
  - Advantages of apps where users access mutually exclusive data
  - Protects system from locking overheads

- **Commit protocol and Recovery control**
  - Guarantees the data durability requirement
  - Atomicity and durability
Transaction management layer

• Data in the Cloud
  ▫ Perform operations on recent snapshot of data
  ▫ Independent between concurrent transactions
  ▫ Hybrid scheme of multi-version and optimistic concurrency control
    • Isolation and consistency for large-scale databases
Transaction management layer

- Multi-version Optimistic Concurrency Scheme
  - Startup timestamp, commit timestamp
  - Read-only transactions run against a consistent snapshot of the database
    - Can commit without the validation phase
  - Update transactions use version number
    - To check for write-write/write-read conflicts
  - Update transaction can only commit if the version of the object is the same as in the read phase
  - Snapshot isolation property
    - Not serializable in all executions
      - Not checking read-before-write conflicts
Transaction management layer

• Commit protocol
  ▫ Read-only transactions
    • Consistent snapshot of the database – no commit
  ▫ Update transactions
    • The log and commit records are stored in a local dedicated disk and also replicated over the storage nodes in the system
Transaction management layer

- Recovery control
  - A storage node can safely leave the system
    - No recovery process is needed
  - Unsafe departure
    - Short-term failure (software bugs ...)
      - Check its local log store
    - Long-term failure (hardware crashes ...)
      - Another healthy node take care of the range index that previously is managed by the failure node
      - New responsible node will recover the data
      - New responsible node will check the transaction logs
        - Redo operations by forwarding the log records
Outline

• Motivation
• Related work
• System architecture of ecStore
  ▫ distributed storage layer (BATON)
  ▫ replication layer (self-tuning range histogram)
  ▫ transaction layer
• Performance study
• Conclusion
Performance study

- Pessimistic replication method is outperformed by the optimistic replication
Performance study

- Results show that the proposed load-adaptive replication method can effectively balance the system load distribution under skewed workloads.
Outline

- Motivation
- Related work
- System architecture of ecStore
  - distributed storage layer (BATON)
  - replication layer (self-tuning range histogram)
  - transaction layer
- Performance study
- Conclusion
Conclusion

• ecStore
  ▫ Underlying BATON distributed index
    • Load-adaptive replication
    • Multi-version optimistic concurrency control
Thanks for your attention!

Questions?