VENUS

Verification for Untrusted Cloud Storage

Masoud Valafar
Cloud Overview

- **Advantages:**
  - Scalability/availability
  - Pay-on-demand
  - Minimizing in-house infrastructure

- **Challenge:**
  - Security
    - Privacy
    - Integrity
    - Consistency
Where does this work fit?

- Addressing integrity and consistency of data – approaches:
  - From within the clouds
  - From Outside the clouds
- Venus assures
  - **Integrity**: a data object read by any client is previously written by another client (protection against malicious provider)
  - **Consistency**: allowing multiple clients to access data concurrently in a consistent fashion
- Goals:
  - Clients dynamics
  - Clients do their operation independently
  - No additional trusted component needed
  - Works on normal cloud commodity
Contributions and assumptions

Contributions

- Providing cryptographic integrity and consistency without introducing trusted component
- Does not involve client-to-client coordination
- Simple semantics for clients – allowing optimistic operation execution and providing consistency later

Implementation

Assumptions:

- No malicious client (providing security against cloud)
- A number of clients (in the core set) don’t crash
VENUS Structure

- Commodity storage service
- Verifier – hosted in the cloud, may become faulty
- Clients
  - Read
  - Write
- Core set – publicly known, help detecting consistency and failures, manage client membership
VENUS Design

- Optimism!
  - Advantage: clients can work concurrently
  - clients do their job (integrity ensured), check for consistency problem
  - Causal consistency is ensured to
  - Question: What happens after an erroneous operation?
- Asynchronous call back interface
  - Issues periodic consistency and failure notification
- VENUS ensures existence of a global sequence of operations, in which legal operations appears according to their order of execution
  - Legal: every read returns the value written by the last write (or failure if no object exist)
  - Sequence includes green operations of all clients
- VENUS guarantees complete operations either become legal or will be detected if fail
Write Operation

- Client $C_i$ Writing $x$ in $p_x$
- Waits for Ack (to make sure it doesn’t crash)
- Send submit, $p_x$ and $h_x$ to verifier
- Verifier creates an ordered list of submit messages, a global sequence of operations $H$
Read Operation

- Client asks for object \( p_x \)
- Verifier responds with \((p_x, h_x)\)
- Client gets the object and checks the hash
- It will return fail if \( p_x \) doesn’t exist or hash doesn’t match
Consistency/Integrity

- How can we make sure that $p_x$ and $h_x$ correspond to the latest write operation?
- In order to verify operations we need $\text{version}(o)$ which is consisted of pair $(vc(o),vh(o))$
- Verifier supplies the context of operations
  - context of operation $o$ is the prefix of $H$ up to operation $o$ as determined by executer of the operation
- before $C_i$ completes $o$, it determines its vector-clock, $vc(o)$ that contains the timestamp of the latest operation of $C_j$ in the j-th entry
- $vh(o)$ contains the cryptographic hash of the prefix of $H$ up to that operation
Consistency/Integrity (2)

- Order on versions: \( \text{version}(o) \leq \text{version}(o') \)
  - \( \text{vc}(o) \leq \text{vc}(o') \) (for every \( k \), \( \text{vc}(o)[k] \leq \text{vc}(o')[k] \))
  - For every \( k \) that \( \text{vc}(o)[k] = \text{vc}(o')[k] \),
    \( \text{vh}(o)[k] = \text{vh}(o')[k] \)
Consistency/Integrity (3)

- Information that verifier keeps
  - $Ver$, an array that contains the last version received from each client
  - $c$, the index of the client from which the maximal version was received
  - $Pending$, list of pending operations
  - $proofs$, contains an array of proofs from SUBMIT messages
  - $Paths$, containing tuples received if the last operations of a client is a write

- Verifier sends $c$, $version(o_c)$, $Pending$, $proofs$
  - If the operation is read also, $(p_x, h_x, t_x)$ from the last pending write operation, other wide from $Paths$
Consistency/Integrity (4)

- Information that clients keep
  - Information of the last operation $o_{prev}$
  - $P_{prev}$ and $h_{prev}$ (if the last operation is read)

- Client actions
  - Checking hashes and signatures
  - Checking $\text{version}(o_c) \geq \text{version}(o_{prev})$
  - if operation is read, then checking $t_x$ with corresponding item in Pending or $ts(o_c)$
  - Client computes $\text{version}(o)$ and checks with returned version
  - There is at most one operation for each client in Pending
  - Checking timestamps of each operation in pending so that it is the last operation of that client
  - Validity of proof for each client that has an operation in pending – proof contain the valid signature of clients on their last operations
Consistency/Integrity (5)

- Clients also keep $CVer$ which contains the last version of each client.
- To get the latest versions:
  - Refresh it via verifier: actual clients activities or dummy-reads.
  - Client-to-client communication (whenever a client executes an operation).
- Clients updates their $CVer$ entries and check whether their operations go green by majority (core-set) checking.
- Failures will be broadcasted to core sets, and they broadcast it to other clients.
Optimization

- Do all the checks when receiving dummy-submit reply
- If both replies are the same, then move on
- Need garbage collection due to creating dummy objects
Dynamic Join

- Clients have global identifiers
- Clients in core sets need to know the public key of new clients
- Informing a client about other clients (for checking signatures)
Implementation

- implementation
  - HTTP based communication to S3
  - Clients communicate through email
  - GnuPG for signature
Evaluation

- Average operation latency for a single client executing operations (using just one client)
- The overhead of using VENUS depends on the location of verifier

Figure 10: Average latency of a read and write operations, with 95% confidence intervals. The overhead is negligible when the verifier is the same LAN as the client. The overhead for WAN is constant.
Evaluation (2)

- 10 clients, 3 in the core, half do read and half do write, each 50 operations
- Fig. 11 shows average time for an operation to complete
- The average latency depends on the inter-invoke latency because versions advance in a slower rate

Figure 11: Average latency for operations with multiple clients to become red and green respectively.
Evaluation (3)

- Amazon S3 does not support pipelining HTTP requests, therefore, throughput is number of client threads over average operation latency.
- Throughput of VENUS is identical to raw S3.

Figure 12: Average throughput with multiple clients.
Goals revisited

- Goals:
  - Dynamic client participation
  - No additional trusted component needed (what about users in core-set?)
  - Works on normal cloud commodity
  - Clients do their operation independently

- Other issues:
  - How to recover from an error
  - Malicious users
  - Overhead in data communication
Questions?