CryptLog: A cloud-driven approach to helping clients synchronize sensitive data

Iulia Tamas
Supervisor: Mahesh Balakrishnan

Abstract

A recent advancement in cryptography is the demonstrated viability of homomorphic encryption for protecting data stored in databases, while maintaining the ability to perform useful operations such as range-queries and keyword matching on encrypted data. We propose CryptLog, a way of extending the capabilities of a traditional shared-log approach to synchronization, to include protection of sensitive data from an honest but curious cloud service provider.

I. INTRODUCTION

Cloud-driven architectures and cloud service providers are growing increasingly popular and widespread. Their usefulness is varied, from allowing individual clients to outsource their work, to helping distributed systems servers communicate and synchronize.

Tango proposes using a cloud-stored shared log for backing replicated in-memory data structures. The benefits for the clients of Tango are many: flexibility in choosing the data structures to represent their data or metadata, multi-object transactions, replication and coordination, log replay, object-based streaming. What Tango doesn’t offer, however, is protection against an honest but curious cloud provider. As such, the benefits of Tango come at the cost of having to fully trust the cloud provider.

We propose CryptLog, a Tango inspired system which allows clients to synchronize the state of in-memory data-structures via an encrypted shared log, without losing the ability to perform streaming, transaction validation and snapshotting in the cloud. CryptLog would be the first SMR system that offers both confidentiality and good performance.

II. CONTRIBUTION

I will be working on CryptLog under the supervision of Professor Balakrishnan and in direct collaboration with another undergraduate.

I will be implementing: two state machine replicated data structures (a set and a register) which can be used by client applications, a client application (a scheduler), the state machine replication runtime, and an optimized mechanism for VM streaming over homomorphically encrypted data.

I. SMR Set

A structure that contains:
• a constructor that takes a pointer to the SMR Runtime and an object id as arguments; the constructor registers a callback with the runtime, to allow the runtime communicate to the data structure updates coming from the shared-log; the object id allows the runtime to keep track which log entries are relevant for this data structure

• an in memory set (a local representation of data persisted and shared through the shared-log)

• read methods which operate through the SMR runtime (the runtime ensures that client read operations receive an answer based on up to date data retrieved from the shared-log)

• append method which operates through the SMR runtime (the runtime ensures write operations are persisted by appending to the shared log, log-entries describing the write)

• callback method passed to the runtime through a register_callback function call; the runtime uses this method to pass to the data structure log entries pertaining to the object it represents

The structure described above is very general. There will be very few set-specific instructions, most likely in the read method when the in memory set can be queried, and in the callback for parsing the log entry. This is intentional, as we want it to be very easy for a client to describe their own data structures.

II. SMR Runtime

Data-structure agnostic Runtime which allows the data structure operations to be shared log cognizant.

Sync

sync(object_id)
Keep an index per object_id describing the last shared-log index synchronized. Stream entries from the shared-log, starting with this index. Deserialize the received data in correct format. Pass log entries to the relevant data structures through the callbacks they registered.

Write

write(object_id, data)
Pack data, and some metadata deemed relevant in a log entry. Serialize log entry and send it to the shared log to be appended.

Multi-object transactions

Performed by appending 2 entries to the log: a BeginTx entry when the transaction is initiated, and an EndTx entry when the transaction is completed.

- BeginTx entry: contains a transaction id
- EndTx entry: contains the object id, read set (objects whose current state the transaction validity relies on), the write set (objects whose state is changed by the transaction), and a set of operations.

An operation contains the index of the object it is performed on, an operator, and its arguments.

runtime::BeginTx() // lock is acquired, read set and write set are emptied
// BeginTx entry is added to shared-log
k = A.get(1) // A is added to the read set, k is read potentially after a sync
The code, written in rust, will be available here:

https://github.com/iuliatamas/CryptLog