Clock Replay Design Document

This is a high level design document briefing the role of classes identified in the diagram replay-class-dig.zargo. To help everyone understand the design, I have considered HTTP protocol as the backbone of this request that shall be considered for replay. The same can be extended to other protocols too, albeit a few modifications. Handling session related information like cookies, etc. is out of the scope of this document since it varies based on the protocol. You may see a few references to Filters etc. Again the detailed description of such classes and entities is out of scope of this document.

After much deliberation, we have decided to provide two kinds of replaying methods Fast and Session. Fast replay emulates the parallel execution of requests, though they are considered in a sequence. While the session method plays all the requests in the fashion they arrive at a web server, sequence and time gap, etc.

DBReader - The main reader class/service that provides information about partitions (capture related files on disk, etc.), and packets of data from the source (stream or DB or whatever).

PacketReader - An interface that defines a contract for the implementing classes to deserialize a packet that has been returned by the DBReader.

HttpPacketReader - A specific implementation of the PacketReader, meant for HTTP protocol.

PacketFilter - A filter implementation that returns necessary packets from the DBReader.

ReplayService - The replay service that plays captured requests. Generally the service is a small cloud with a configurable number of replay processes running on various machines. One of them can be promoted to a master service (hopefully via a client interface), which then broadcasts a message “IamMaster” to other replay processes. The processes then downgrade themselves to slave configuration. When a client sends a request to replay all the captured requests, the master service requests the DBReader for available partitions and other information. It then sends the information to replay processes. Each slave replay process then communicates with the DBReader using the packet reader and gets its set of packets (usually a configurable number). It then replays the read requests by creating Task objects and supplying them to the TaskExecutor.

Use cases for the replay service:

The service can be requested to run a fast replay or session replay of the captured requests.

1. Fast Replay – In this method, requests are replayed in the order they are captured. User session is immaterial and time delay between requests is dropped if necessary. It simply creates a Task for each request in sequence and places the Task in the TaskExecutor's task queue. Any random FastThread from the pool just grabs the request and replays it. While one thread is busy executing the request, it creates many such tasks and places them in the executor's pool, thereby achieving parallelism. The maximum number of threads that can be created by the executor is configurable.

2. In case of session replay, user session is very important. The time delay between requests also matters, i.e. if there is a gap in time between two requests then such a gap will be emulated. The service gathers all the requests related to one user session, creates one Task with all of them, and then places the task in the executor's task queue. The executor creates a SessionThread for each such task. In other words, there will be one SessionThread associated with one user session. The maximum number of threads in the pool will not exceed a configurable number.

TaskExecutor - A pool of executor threads.

FastThread- A thread that executes one task (request). Once the response to the request is received, its job is done. Fast thread sends synchronous HTTP requests.

SessionThread - A thread that executes one task. The task contains all the requests originating from one user session. Session thread sends async HTTP requests.

FastTask - A job submitted to each FastThread in the executor's pool. The task contains on request that needs replay.

SessionTask - A job submitted to each SessionThread in the executor's pool. The task contains all the requests originating from a user's session.

Since there are two replay methods, the knowledge of such replay needs to be present in some component of the product. I have chosen to make the threads and the replay service smarter. One may certainly question the rationale behind this and and argue that the replay service should be dumb.