Challenges in Building Distributed Finite State Machines

Smita Vijayakumar
Exotel
smita@exotel.in

ABSTRACT
Finite State Machines (FSM) are classic solutions for building stateful systems. Building finite state machines for stateful distributed systems however, involves additional complexities. This is because distributed systems introduce an inherent tradeoff between latency and consistency; or, in the presence of partitions, availability and consistency. In a soft real-time scenario, processing of events needs to happen in a time-sensitive manner. Shared resources also need to be accessed and manipulated correctly. What are the design considerations when building such an FSM that can run on a distributed system? We shall discuss some challenges and potential solutions.

AUDIENCE
This technical talk is intended for audience with some technical knowledge of and hands-on exposure to distributed systems. Audience is expected to be familiar with FSM and its properties. It is also assumed that it has basic knowledge of distributed systems, their characteristics and use cases. Knowledge of PACELC[1] and CAP[2] theorems will be helpful.

INTRODUCTION
The recent emergence of cloud computing is making the vision of utility computing realisable, i.e., computing resources and services from cloud can be delivered, utilised and paid for in the same fashion as conventional utilities like water and electricity. Cloud hosting also comes with a host of other advantages like high availability, ease of deployment and flexibility. Due to this, cloud computing has become the de facto hosting solution for applications requiring these properties.

We consider applications that require finite state machines to be defined as a part of the system. Though there are many applications built using state machines, we shall illustrate the challenges and complexity by specifically citing the example of a cloud telephony platform. Cloud telephony systems render telecommunication services by using cloud infrastructure. Since telecommunication services are critical and highly sensitive to downtimes, availability is of paramount importance. Also, being soft real-time systems, latency has to be kept under reasonable limits. For example, for speech applications the mouth-to-ear delay is recommended to be under 300ms for acceptable quality[3]. Lastly, the system is also required to be as correct and consistent as possible. This gives rise to a need to make design decisions keeping inherent trade offs that need to be made in a distributed system[1].

Let us examine some of these challenges in more detail. For sake of clarity, let us assume an FSM based distributed service. Also, assume that the state machine is instantiated per flow (say, call flow), and brought down upon flow completion. Therefore, there are as many instances of the state machine as there are active flows in the system (denoted by f_1 to f_n). The service itself has to run on multiple hardware or virtual machines to ensure high availability of the service (denoted by m_1 to m_i). Hence, an event pertaining to a flow may arrive at any of the machines to be processed. Let us assume an event for a flow f_j is processed by m_p and the next event for f_j is processed by m_q. This requires m_q to view the current state of the FSM instance for flow f_j as the final state arrived at by m_p. Herein lies the first challenge in designing distributed datastore write and reads. Is it desirable to have less latent writes to the datastore risking correctness in data views, or should correctness be paramount even if more latent than the first approach?

The second challenge in such a system is ensuring multiple instances of the state machine do not simultaneously process the same flow. This is because events valid in one state might not be so in the other state. Therefore, events have to be strictly processed in sequence. When parallel processing of multiple events for the same flow occurs, this strict ordering is lost, thereby compromising the correctness of the system. Hence, when ensuring high availability in the system, there is also a need for a distributed mutual exclusion. To overcome this problem, we used distributing locking mechanism. When an event for a particular flow is received, the service running on that machine requests for an exclusive processor lock for that flow. If granted, the state machine is allowed to run the flow. Else, it enqueues the event on the distributed queue and assumes the lock owner will process it at some time in future. Strategies can also be put in place to force-release locks. Redlock[4] implemented with Redis, and Apache

Helix[5] running on top of Zookeeper provide the more well-know open-source solutions for distributed lock management.

Timer management is another problem that calls for attention. One other challenge when implementing distributed telephony FSM is keeping a watch on active flows. Since active flows necessarily consume resources, it is important to ensure all flows reach the end state. To this effect a timer is required to periodically poll the status of the flow, and ensure it is still in an active state. When doing so, timer management has to occur outside of the state machine service since the timed expiry events have to be highly available as well.

There is also a need to ensure strict ordering in the system. For this, the downstream service emitting events towards the distributed FSM service is required to wait for acknowledgement for every event before emitting the next event for the same flow. The acknowledgement is sent only after the event is enqueued in the distributed queue, and therefore, ensures correctness in ordering of events.

OUTCOMES/CONCLUSION

The motivation of this talk is to present practical challenges in building distributed FSMs. The talk will focus on the nature of challenges and how some of these challenges were overcome when designing a cloud telephony platform. The audience is expected to gain an insight into tradeoffs to be made, potential pitfalls in designing such state machines, and possible solutions to overcome the same.

PARTICIPATION STATEMENT

I agree to attend the conference if my talk is accepted.

BIO

Smita graduated with a Masters in Computer Sciences under the guidance of Prof. Gagan Agrawal at The Ohio State University. Their work on resource management on cloud environments is published as a book[6]. Smita worked in the San Francisco bay area for network giants before relocating to Bangalore. Presently at Exotel as an engineer, she has developed keen interest in using distributed systems for solving real-world problems. She is excited about exploring and applying this for efficient solutions to problems on hand at Exotel. She was a speaker at GopherCon India 2016[7].

REFERENCES/BIBLIOGRAPGHY

1. D. Abadi, "Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story," in *Computer*, vol. 45, no. 2, pp. 37-42, Feb. 2012.
2. Seth Gilbert and Nancy Lynch, “Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services”, *ACM SIGACT News*, Volume 33 Issue 2 (2002), pg. 51-59
3. ITU-T Recommendation G.114, One-Way Transmission Time, in Series G: Transmission Systems and Media, Digital Systems and Networks, Telecommunication Standardisation Sector of ITU (2000): https://www.itu.int/rec/T-REC-G.114/en
4. Distributed locks with Redis: http://redis.io/topics/distlock
5. Distributed Lock Manager: http://helix.apache.org/0.6.2-incubating-docs/recipes/lock_manager.html
6. Resource and Accuracy Management in Cloud Environments (Smita Vijayakumar): www.amazon.com/Resource-Accuracy-Management-Cloud-Environments/dp/3659308064
7. Gophercon Talk by Smita Vijayakumar - Go's Context Library: http://www.slideshare.net/exotel/gophercon-talk-by-smita-vijayakumar-gos-context-library