Principles of Reactive Systems

- Responsive
- Resilient
- Elastic
- Message-driven
Architecture

Web (Play) → backend (Akka) → Journal (Cassandra) → Fast data processing (Spark)
Problem: Splitting application
Monolithic
Splitting MLP

Frontend nodes

Backend nodes
Akka Distributed Pub/Sub

- Location transparency
- Publish/Subscribe
- Dynamically add/remove
Distributed Pub/Sub Mediator

Frontend nodes

Backend nodes
Code
Frontend

// Starting up the mediator
val mediator = DistributedPubSubExtension(system).mediator

// Sending message through mediator
mediator ? DistributedPubSubMediator.Send("/user/course-aggregate", CreateCourse(course), localAffinity = true)
// Starting up the mediator
val mediator = DistributedPubSubExtension(system).mediator

// Setting up the services
val courseAggregate = 
    system.actorOf(CourseRepository.props, "course-aggregate")
mediator ! DistributedPubSubMediator.Put(courseAggregate)

val levelAggregate = 
    system.actorOf(LevelRepository.props, "level-aggregate")
mediator ! DistributedPubSubMediator.Put(levelAggregate)
Problem: Distributed State
State on single node
State across nodes

Frontend nodes

Backend nodes
Akka Cluster Sharding

- Partition actors across Cluster
- Location transparency
- Rebalancing
How does it work?

Shard Coordinator

Shard Region 1

Shard Region 2

19
Code
Backend: Create ShardRegion

// Backend: Start cluster sharing in non-proxy mode
ClusterSharding(system).start(
  "users",
  Some(User.props),
  UserSharding.idExtractor,
  UserSharding.shardResolver())
// Extract the user shard
// Rule of thumb: 10 times the max numbers of sharding nodes
def shardResolver(shardCount: Int = 20) = {
  case UserService.Envelope(email, _) =>
    email.hashCode % shardCount.toString
}

// Extract the id and payload of the user actor
val idExtractor: ShardRegion.IdExtractor = {
  case UserService.Envelope(email, payload) =>
    (email, payload)
}
// Start sharding in proxy mode on every frontend node
val shardRegion: ActorRef = ClusterSharding(system).start(
  "users",
  None,
  UserSharding.idExtractor,
  UserSharding.shardResolver())

// Send message via shard region
shardRegion ? UserService.Envelope(email, GetUser(email))
Problem: Loosing State
State in memory

Frontend nodes

Backend nodes
Persist state
Akka Persistence

- Stores state in Journal DB
- Event Sourcing
- Restore state
Code
Write user state

class User extends PersistentActor {

  // User id / sharding extractor id
  override def persistenceId: String = self.path.name

  // Persist UserCreated event to Cassandra journal
  override def receiveCommand: Receive = {
    case UserService.CreateUser(userProgress) =>
      persist(UserCreated(userProgress)) { userCreated =>
        // Update user state
        onUserCreated(userCreated)
        sender() ! UserService.UserCreated
      }
  }

  ...

Recover user state

...
Problem: Replicate Cache
Singleton Cache

Frontend nodes

Backend nodes
What is CRDT?

Conflict-free replicated data type is a type of specially designed data structure used to achieve strong eventual consistency.

Great talk on CRDT by Sean Cribbs: [Eventually Consistent Data Structures](#)
CvRDT

source $f(x_1)$

$g(x_2)$

merge

merge
CvRDT - Example
CmRDT
Akka Data replication

• CRDT for Akka cluster
• Replicated in-memory data store
• Supports many types: (GSet, ORSet, PNCounter…)

CRDT Replicator

Frontend nodes

Backend nodes
Code
Frontend

//starting up the replicator
val replicator = DataReplication(system).replicator

//get cache entry
val f = replicator ? Get(key, ReadLocal, None)
Backend

//starting up the replicator
val replicator = DataReplication(system).replicator

//updating the cache
replicator ! Update(Course.cacheKey, GSet.empty, WriteLocal) ( _ + courseCreated.course)
Q & Option[A]

@nraychaudhuri
@markusjura