Scala 实践略谈

——从零到一支广告技术团队

凤凰木 <weiwen@weiwen.org>

May 29, 2018

内容

团队

实践

团队

起点

- 时间: 2017 年 3 月
- 业务: Ad Exchange(ADX)

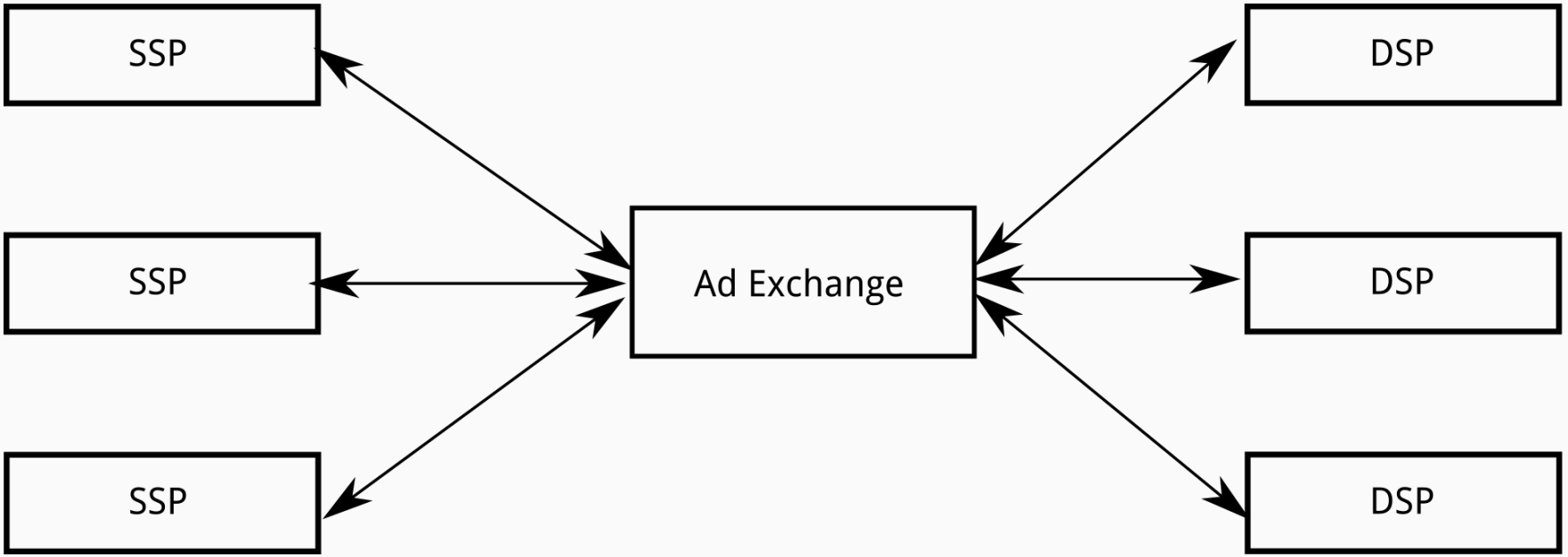
需求和挑战

- 广告系统特有的业务复杂性
- 高并发、高延迟的实时竞价服务
- 多维度、准实时数的数据报表系统

需求和挑战

- 旧系统的技术债务
- 没有 Scala 程序员
- 不破不立

Ad Exchange



入门教材

由浅入深

1. Scala School! Twitter 的 Scala 快速入门, 有中文版*Link*
2. A Tour of Scala, Scala 官方出品的入门指南*Link*
3. Effective Scala, Twitter 的 Scala 最佳实践*Link*
4. (书) 快学 Scala*Link*
5. (书) Scala 函数式编程*Link*
6. Scala 标准库 API*Link*

关于函数式编程理论

1. 虽然理论很重要，但生命和代码更重要
2. 不宜上来就学范畴论和 Lambda 演算理论
3. 不宜沉迷于理论

关于函数式编程理论

当代码已经很脏的时候

1. 函数式抽象
2. 抽象的层次不宜过高

Scala 新手易犯的错误和解决办法

- 问题: 滥用 var, for, while
- 解法:
  1. 禁用 var, for, while
  2. 学习 Scala Collections API(map, flatMap, fold*, etc.)
  3. 记住写的是表达式，而不是语句

Scala 新手易犯的错误和解决办法

- 问题: Java 固有思维，脱离设计模式很茫然
- 解法: 提出需求，写代码示例

Scala 新手易犯的错误和解决办法

- 问题: 不熟悉 Scala 项目开发流程
- 解法:
  1. 预先部署好开发环境 (repositories, sbt build config, etc)
  2. 写好项目模板
  3. 创建简单项目练习

Scala 新手易犯的错误和解决办法

- 问题: 不习惯异步编程
- 解法:
  1. 禁止写出存在阻塞的代码 (禁用 Await)
  2. 学习 Future 相关的一切

现实永远比理论复杂

```
// easy
val users: Future[List[User]] = ???

// WTF!!!!!
val users: Future[List[Future[Option[User]]]] = ???

// easy
val ids: Future[List[Int]] = ???

def findUserById(id: Int): Future[Option[User]] = ???

def updateAndReturnInfoByUser(user: User): Future[Option[Info]] = ???

// WTF??????
def updateAndReturnInfoByIds(ids: List[Int]): Future[List[Info]] = ???
```

结论-1

关于先天条件，

- 天赋不是必须的，起点也不如预想的重要
- 实践优先与理论，以产出为导向
- 范畴论，函数式编程理论，一开始都不是必需的
- 大多数不是 Scala 特有问题，而是程序员成长的普遍问题

结论-2

培养合格的工程师，而不是会写 Scala 的程序员

- 打好 Linux 基础
- 由简入繁，逐步学会设计
  - 函数的设计
  - Service API(高内聚，低耦合)
  - 数据存储结构的设计 (表，各种 NewSQL DB 的存储细节)
  - 由功能模块到子系统的架构设计
  - 始终保持清醒，学会用设计的思维去处理要实现的业务逻辑
- 学习数据中间件 (RMDBS, Kafka, Redis, Aerospike, etc.)
- 学习 JVM 基础知识
- 学习分布式系统理论

结论-3

Scala 技术栈的考虑

- 考虑所处的业务领域
- 有没有能处理大部分问题的人
- 团队的风格和喜好

实践

处理高延迟的并发场景

- Akka Http as HTTP Service
- Finagle Http as HTTP Client
- Future.sequence() in Concurrency
- Finagle Thriftmux as RPC Framework

处理高延迟的并发场景

- 异步 + 消息队列的模式
- 处理竞价的过程中不碰磁盘
- 水平线性拓展

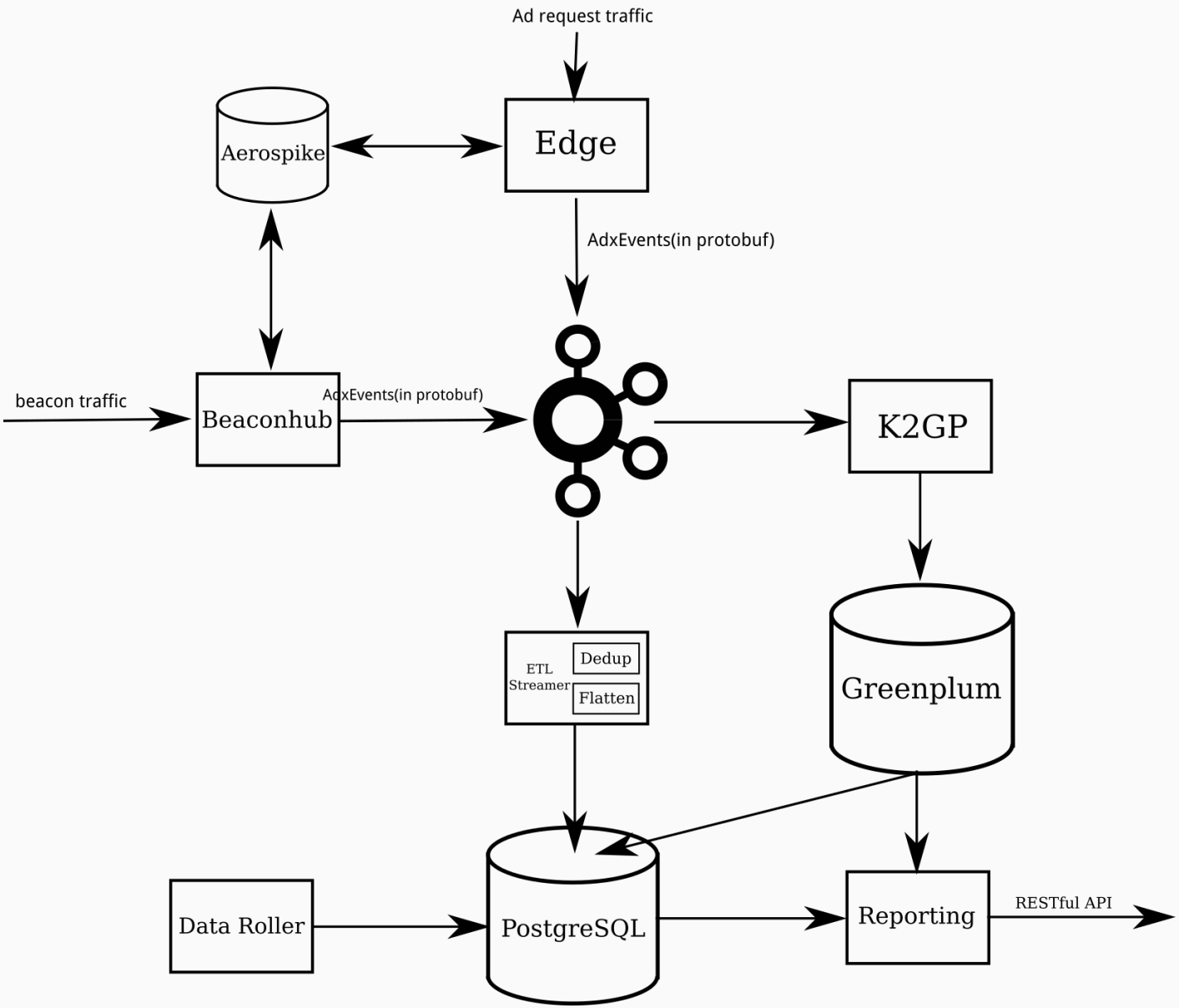
Queue

```scala
def sendToKafka(t: Int): Future[_] = ???

val queue: SourceQueueWithComplete[Int] = Source
  .queue[Int](bufferSize = 1000, OverflowStrategy.backpressure)
  .mapAsync(parallelism = 20)(sendToKafka)
  .to(Sink.ignore)
  .run()

queue.offer(elem = 1)
queue.offer(elem = 2)
```

处理流式的长链数据流

- Lambda 架构，MicroService 化
- 流式处理 + 逐级聚合 + 最终一致性校验
- Akka Stream based Streaming
- Akka Stream based Data Roller
- Slick in Reporting

数据与报表系统简图


Adx Reporting Architecture
Ad request traffic
Aerospike
Edge
AdxEvents(in protobuf)
beacon traffic
Beaconhub
AdxEvents(in protobuf)
K2GP
ETL
Streamer
Dedup
Flatten
Greenplum
Data Roller
PostgreSQL
Reporting
RESTful API

函数式流聚合

```
case class DataRow(
                    dspId: Int,
                    sspId: Int,
                    impressions: Long,
                    clicks: Long
                  ) {
  def +(that: DataRow): DataRow = {
    require(this.dspId == that.dspId && this.sspId == that.sspId)
    DataRow(
      dspId = this.dspId,
      sspId = this.dspId,
      impressions = this.impressions + that.impressions,
      clicks = this.clicks + that.clicks
    )
  }
}
```

函数式流聚合

```scala
val rows: List[DataRow]

val aggrs: List[DataRow] = rows
  .groupBy(r ⇒ (r.dspId, r.sspId))
  .flatMap { case (_, rs) ⇒
    rs.reduceLeftOption(_ + _).toList
  }
  .toList
```

定时任务系统

```
Source
  .tick(Duration.Zero, tickInterval, ())
  .mapAsync( parallelism = 1)(_ ⇒ dspAppConfigDAO.queryAll())
  .withAttributes(ActorAttributes.supervisionStrategy(decider))
  .map { xs ⇒
    val m: Map[(Long, AdType), DspAppConfig] =
      xs.map(x ⇒ (x.dspAppId, x.adType) → x).toMap
    dspAppConfigCache.updateAll(m)
  }
  .to(Sink.ignore)
  .run()
```

定时任务系统

Cluster Singleton

```
system.actorOf(
  ClusterSingletonManager.props(
    singletonProps = ClusterSingletonJobActor.props(name)(cancelable)
    , terminationMessage = ClusterSingletonJobs.TerminateCluster
    , settings = ClusterSingletonManagerSettings(system)
  )
)
```

常用库和组件, Kafka

Kafka

- Kafka 1.0
- Akka Stream Kafka

Kafka Pipeline

```scala
trait ProducerPipeLine[K, V, T] {

  val producerKeySerializer: Serializer[K]
  val producerValueSerializer: Serializer[V]

  val bufferSize: Int = 10000
  val overflowStrategy: OverflowStrategy = akka.stream.OverflowStrategy.backpressure

  def keyTransfer(t: T): K
  def valueTransfer(t: T): V

  def queue(topic: String)(implicit system: ActorSystem, mt: Materializer): SourceQueueWithComplete[T] =
    Source
      .queue[T](bufferSize, overflowStrategy)
      .map(e ⇒ ProducerMessage.Message(new ProducerRecord(topic, keyTransfer(e), valueTransfer(e)), e))
      .via(Producer.flow(ProducerSettings(system, producerKeySerializer, producerValueSerializer)))
      .withAttributes(ActorAttributes.supervisionStrategy(decider))
      .map(_ ⇒ SimpleMetrics.kafkaSentCounter.inc())
      .to(Sink.ignore)
      .run()

  protected val decider: Supervision.Decider = {
    case NonFatal(e) ⇒ Supervision.Resume
  }
}
```

Kafka Pipeline(Simple)

```scala
trait SimpleKafkaSink[T] extends ProducerPipeLine[String, Array[Byte], T] {

  override val producerKeySerializer: Serializer[String] = new StringSerializer

  override val producerValueSerializer: Serializer[Array[Byte]] = new ByteArraySerializer

  override def keyTransfer(t: T): String = java.util.UUID.randomUUID().toString

  def toKafka(x: T):Future[QueueOfferResult]

}
```

常用库和组件, SQL

- DB
  - PostgreSQL
  - MySQL
  - Greenplum
- Libary
  - Slick
  - Shapeless

常用库和组件, Slick

Slick

- 规整的 CRUD:
  - Slick Table Query
  - 类型安全，重构方便
- 复杂 OLAP:
  - 裸 SQL 丢给 Slick/HikariCP
  - 不要过度迷恋
  - 不要作徒劳的挣扎

常用库和组件, Aerospike

Aerospike

- 性能惊人，然而并非高可用
- Aerospike 3.x && 4.0
- Aerospike Java Client

常用库和组件, JSON

JSON

- circe
- json4s

常用库和组件, RPC

RPC

- Finagle Thriftmux
- scrooge-sbt-plugin
- Twitter Server

常用库和组件, Test

Test

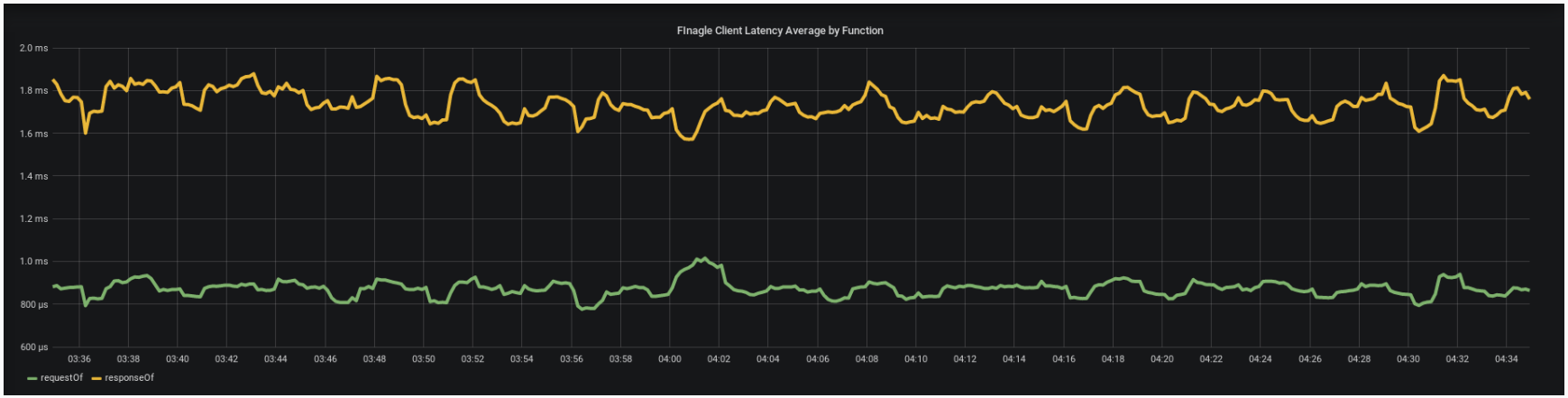
- ScalaTest
- Specs2
- Akka Http Testkit

常用库和组件, Merics

Merics

- Prometheus Java Client
- Grafana

Metrics


FInagle Client Latency Average by Function
2.0 ms
1.8 ms
1.6 ms
1.4 ms
1.2 ms
1.0 ms
800 µs
600 µs
03:36
03:38
03:40
03:42
03:44
03:46
03:48
03:50
03:52
03:54
03:56
03:58
04:00
04:02
04:04
04:06
04:08
04:10
04:12
04:14
04:16
04:18
04:20
04:22
04:24
04:26
04:28
04:30
04:32
04:34
requestOf
responseOf

Metrics


Aerospike Latency Average
1.0 ms
800 µs
600 µs
400 µs
200 µs
0 ns
21:30
21:40
21:50
22:00
22:10
22:20
22:30
22:40
22:50
23:00
23:10
23:20
23:30
23:40
delete
get
put
Aerospike Throughput
15K ops
10K ops
5K ops
0 ops
21:30
21:40
21:50
22:00
22:10
22:20
22:30
22:40
22:50
23:00
23:10
23:20
23:30
23:40
get
put

常用库和组件, CI/CD

CI/CD

- SBT Native Packager
- Sbt Scoverage
- Sbt Coursier
- Gitlab

完。

谢谢大家！