LECTURE 14 FARM

1. 前置知识

- DIMM： 双列直插式内存。区别于SIMM，DIMM支持64位，SIMM只支持32位。
- NVDIMM：非易失性双列直插式内存。价格非常高，在普通DIMM内存的几倍。FaRM使用的为NVRAM（即非易失性DRAM，见下文，使用分布式UPS保证非易失性）。
- RDMA（Remote District Memory Access）：远程直接内存访问技术
- RDMA Queue Pairs = Send Queue + Receive Queue，传送门
- Infiniband 用于大型计算机集群的一种网络技术，传送门

2. 概述

- FaRM获得极佳性能表现的原因：
  - 数据分片，若数据被分成了n片，则在极端理想情况下，事务处理速度能提升n倍。
  - FaRM将数据存储在内存之中
  - 为了防止掉电使用NVRAM
  - RDMA技术：可以直接读内存，而不用CPU
  - kernal bypass 内核旁路技术：网卡相关操作不需要内核的参与，应用程序在自己的内存区域内维护输入输出缓冲区。同时，应用程序也需要自行处理重复的包等TCP协议解决的问题。
- 只使用单向RDMA无法实现事务，因此FaRM也有RPC的参与。
- FaRM采用的是OCC（optimistic concurrency control），读操作不上锁，但是需要在commit phase 的Validation阶段进行验证。

3. 硬件发展

3.1. 非易失性 DRAM

- 实现：分布式UPS（Distributed Uninterruptible Power Supply）

    - 使用锂离子电池，比铅酸电池便宜
    - 使用冗余，电池故障只会影响机架中的一小部分机器
    - 断电时，可以使用UPS的电池将数据写回SSD
- 避免了持久化操作！延长SSD寿命，避免了持久化带来的开销
- SSD 越多，将每GB数据持久化所需要的能量反而越小！（因为保存时间变小了。也许是并发保存数据的）

3.2. RDMA Networking

- RDMA：Remote District Memory Access
- FaRM使用单向RDMA，不需要被访问计算机的CPU资源
- 作者们对比了单向RDMA和传统双向RDMA+RPC后发现NIC是瓶颈，消除了NIC瓶颈后发现CPU成为瓶颈。当前实现中则不需要使用被访问方的CPU资源，因此加大了吞吐量

3.3. 硬件改造的额外成本

- UPS的电池
- SSD需要预留空间用于保存SDRAM数据

4. API

下面使用一个例子来介绍各个API

- txCreate( ) 创建一个事务
- o = txRead(OID)，根据数据对象的ID（也就是OID），读取数据，此时o为副本
- o.somefieldofO += 1，o为内存中的副本，修改o不会修改OID的数据
- txWrite(OID, o)，将o写入OID
- ok = txCommot( )， 这一函数被调用之后，将会执行FaRM commit protocol 中的Commit Phase，在此之前的操作都可以被认为是Execute Phase

事实上，这里的OID是一个抽象的概念，由两部分组成：

- Region ID，Coordinator需要根据Region ID找到primary和backups

- 数据在Region内的偏移

5. 总体架构

- 整个集群的机器形成一个大地址空间
- 在一个事务中，对于数据对象的访问，对用户是透明的。（调用API的程序猿不知道这些对象是在本地还是集群的别的机器上）
- 一个应用线程可以在任意时间开始一个事务，其自动成为Coordinator
- FaRM保证所有**成功提交**的事务的**严格序列化**
- 事务将对数据对象的修改暂存，在成功提交之前对别的事务不可见
- FaRM保证：
  - 读单一数据对象是原子的
  - 后继读取读到同样的数据（可重复读）
  - 读到最新写入的数据
- FaRM提供：
  - 只读事务（单一数据对象）
  - Locality Hint*
- **一个包含四台计算机的FaRM实例**架构图：

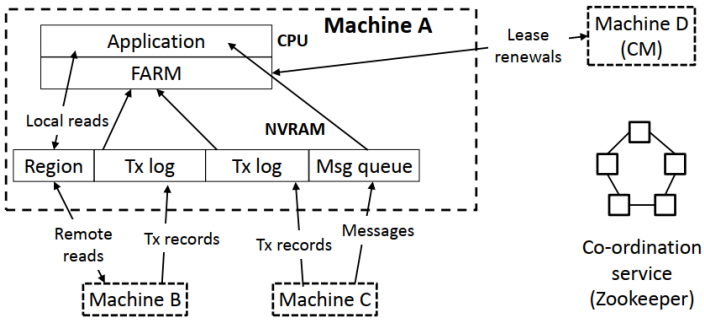


Figure 3. FaRM architecture

5.1. FaRM配置

- FaRM为一个四元组：$< i, S, F, CM >$

- i：64位 单调递增的**配置**标志符
- S：当前配置中，所有计算机的集合
- F：映射$Failed\ Machines \longrightarrow Failure\ Domains$，其中$Failure\ Domains$是那些会独立故障的组成部分
- CM：Configuration Manager

- FaRM的配置通过ZooKeeper在FaRM集群的各台机器之间达成一致，当FaRM配置发生变化时，通过调用ZooKeeper的API来完成修改
- CM维护的信息：
  - 映射：$regionID \longrightarrow all\ replicas$

5.2. Region

- FaRM的地址空间由Region组成，Region大小为$2GB$
- **CM（Configuration Manager）** 维护了一个映射：$regionID \longrightarrow all\ replicas$，这个哈希表本身也是多份复制的。集群中的其他机器通过命令来访问这个映射，并且在此之后会被缓存
- 读取Region需要使用单向RDMA，单向RDMA所需要的 **RDMA 引用**（？） 也会被缓存
- 只会读取Primary Replica
- Region分配过程：
  1. 某机器与CM通信要求分配Region
  2. CM给新的Region分配单调递增的RegionID，并且选择Replica
  3. 通过2PC协议来完成Replica的选择（请你准备作为我的region的replica、请你开始作为我的replica）

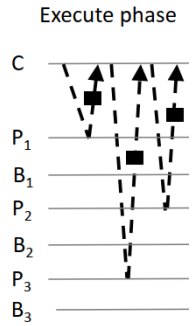
6. 分布式事务执行过程

为了提高性能，FaRM的设计提高了Replication和Xaction的耦合度。

在Spanner中，容忍f台机器的故障需要$2f+1$台机器的集群，而在FaRM中，容忍f台机器的故障只需要$f+1$台机器

6.1. Excute Phase

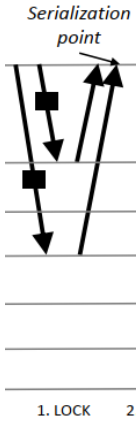
Coordinator通过单向RDMA来获取数据、数据的**版本**和数据的地址。Coordinator可以是Replica！



读取数据之后，Coordinator将处理后的数据保存在本地。

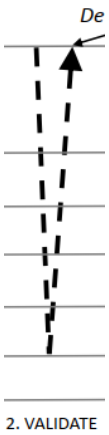
6.2. Commit Phase

1. Lock
    - Coordinator通过RDMA给所有的**修改了数据的Primary**的日志中写入Lock Record，。而后Primary尝试获得**数据在指定版本**的锁。而后Primary返回一条消息报告是否成功。当如下情况发生时，Primary获得锁失败：
        - 数据的版本与一开始在Execute Phase读到的不一致
        - 指定版本的数据的锁 正在被 其他事务持有
    - 在LOCK消息中，Coordinator会携带如下信息：
        - Object ID
        - 数据对象的版本号
        - 数据对象的新数据
    - 在这一阶段，Coordinator必须等待所有Primary处理完毕。因为需要验证数据对象的版本。Primary处理Lock Entry结束后，使用RDMA向Coordinator发送消息。
    - 在此处，Primary会使用TestAndSwap原子指令来测试版本号和锁标志，并且设定锁标志。
    - **终止事务：** 获得锁失败后，Primary向Coordinator发送锁获取失败的消息。Coordinator终止事务，通过RDMA向所有的**Primary**日志中写入Abort Record，并且向应用程序报错



2. Validate

   - 只读取 **只读** 对象的 **版本而不是数据！**
   - 只读取Primary
   - 将版本与执行阶段读到的版本进行对比，如果版本发生变化则终止事务（终止事务同上）
   - *存在一个阈值t_r，若某个Primary中包含超过t_r个数据对象，这些数据对象在当前事务中出现，则使用RPC读取，反之使用RDMA



3. commit backup

   Coordinator通过RDMA向**backup Replica**的log中写入Commit-backup record，并且等待NIC对于RMDA写操作的自动回复。这一过程不需要被写入方的CPU的参与

4. commit primary

   Coordinator通过RDMA向**Primary Replica** 的log中写入Commit-primary record，并且等待回复。当Coordinator收到了其中一个Primary NIC的回复（？？？？？）或者Coordinator和某Primary是同一台机器时，Coordinator即可向应用返回成功。

- Primary如何处理Commit-primary record：Primary更新数据，增加数据的版本，释放数据的锁。而后数据的更新就对其他事务可见了

5. truncate

   Coordinator 会在收到了**所有Primary**在4中的回复之后才会截断日志。Primary 和Replica 的日志内容在被截断之前会一直保存。Backup仅在日志截断时，才会将日志中的内容应用到实际数据上。

6.3. 正确性

- 串行化点（Serialization Point）：已提交事务中，数据在串行化点的版本 和 在执行阶段的版本应该一致。
  - 只读事务：Validation
  - 读写事务：Lock
- 在Commit-backup阶段，必须要求Coordinator等待所有的Backups返回：以避免丢失修改
- 在Commit-primary阶段，必须要求Coordinator等待第一个Primary返回才能向应用返回成功：若提前返回成功，可能不能保证Validation阶段完全成功，因此可能出现在返回成功后事务却需要终止的情况
- 由于协议使用RDMA而不是RPC实现，因此不会使用被操作方的CPU。因此Coordinator必须要维护所有的Primary 和backup的内存（SDRAM）空间。当内存空间不足时，需要Coordinator显式地写回可截断的log record来释放内存空间。

7. 容错

7.1. 错误检测

- FaRM使用租约来完成错误检测，任何一台机器的租约到期就会触发错误检测
- FaRM通过短租约来保证高可用性（$5ms$）

7.1.1. Lease机制

- 每一台机器都有lease，lease很短，通常为$5ms$
- 除了CM之外的所有机器在CM处有租约，CM在除自己之外的所有机器处有租约
- 租约授权过程：

1. 非CM的一台机器m向CM发送一个租约请求
2. CM向该机器回复，该回复中包含了对m的租约的授权，以及CM向m申请的租约请求
3. m再向CM回复，授予CM的租约

- 底层细节：FaRM使用Infiniband send 和 Receive verb（类似于接口）。如果使用RDMA的queue pair，则要求通信双方都维护一个queue pair。为了保证租约相关的消息不被延迟，因此非CM机器和CM都必须使用额外的queue pair，并且CM中需要维护与其他所有机器的queue pair，内存开销很大，并且cache 缺失率很高。使用Infiniband Send 和Receive verb + 不可靠的数据报（datagram）传输，则可以解决这一问题。
- 为了避免消息丢失，每过$\frac{1}{5}$的租约时间，就需要尝试一次租约更新。FaRM有一个专用的租约管理器线程。这个线程以最高的优先级运行，但是并没有绑定某一个硬件线程。该线程采用中断而不是轮询的方式运行，以避免重要的系统任务进入饥饿状态。租约管理器会在初始化时被预分配所有的内存，并且进行调页，避免PF带来的开销。

7.2. Reconfiguration

使用租约来保证一致性的通常方法：服务器必须要在响应对某个数据对象的请求之前，检查自己是否具有这个租约。对于FaRM而言，存在天然的障碍：由于使用RDMA，被访问服务器的CPU不参与，因此无法判断该服务器是否具有租约。（当前的网卡不支持租约）

具体流程：

1. Suspect
    - CM发现非CM服务器租约过期，则CM suspect 该非CM服务器，并且阻塞所有的外部请求
    - 非CM服务器发现CM租约过期，则非CM服务器（按照一致性哈希算法）寻找**备份CM**，若**经过一段时间**后，非CM服务器发现Configuration仍然没有改变，则该非CM服务器成为CM服务器并且开始**Reconfiguration。**
2. Probe

    新CM向所有的服务器发送RMDA读取请求，但不包括在1中被suspect的服务器。同时，若新CM没有收到某些服务器NIC的响应，则也会suspect这些服务器。
    - 为了防止在网络分区情况下出现脑裂，新CM必须在收到**大多数**服务器NIC的响应之后们才能继续进行。
3. Update configuration

新CM更新Configuration，将其改为$< c+1, S, F, CM_{id} >$，其中，S为相应新CM的服务器集合，CM_{id}为当前新CM的id。值得注意的是，为了保证只有一台server可以将Configuration从c更改为$c+1$，FaRM在ZooKeeper之上构建了一个原子的Compare-and-Swap操作（使用Sequence Number）。仅当当前的Configuration为c时，才可以将其改为$c+1$，避免同一时间多台server成为CM，同时更改Configuration。

4. Remap regions

   新CM需要重新设置映射：$regionID \longrightarrow all\ replicas$，保证每个region能进行$f+1$的容错。若一个primary崩溃了，新CM会指定一个backup成为primary。若一个region所有的replicas都崩溃了，或者空间不足，导致无法保证每个region能进行$f+1$的容错，则FaRM会给错误提示。

5. Send new configuration

   新CM向所有的服务器发送NEW-CONFIG，其中包含如下信息：

   - Configuration ID
   - CM ID
   - Configuration中所有其他服务器的ID
   - 新的映射：$regionID \longrightarrow all\ replicas$

   若CM改变，则NEW-CONFIG还作为新CM向其他非CM server发送的lease请求

6. Apply new configuration

   某server在收到NEW-CONFIG之后，若其中的Configuration ID更大，则server更新Configuration ID，缓存映射，为分配给自己的region分配空间。

   - 从这里开始，该server不再对不属于自己所在Configuration的server发送请求，会拒绝来自这些server的读写请求，阻塞来自外部client的请求。

   随后，server向新CM发送NEW-CONFIG-ACK消息。这条消息作为：

   - 该server对新CM租约的授权
   - 该server向新CM申请租约

7. Commit new confi guration

   - 在CM收到了所有的NEW-CONFIG-ACK消息之后，新CM需要等待一段时间，这段时间必须保证那些不再属于自己Configuration中的机器的租约过期

- CM向所有机器发送NEW-CONFIG-COMMIT
- 非CMserver收到了NEW-CONFIG-COMMIT之后，不再阻塞外部请求

---

8. 相关论文

- Paxos变种：Vertical Paxos ： 25
- 四段提交协议：16
- 精准成员：10
- 严格序列化：35
- 使用租约来达成cache一致性：18