分　数:	
评卷人:	

華中科技大學

研究生（数据中心技术）课程论文（报告）

题 目：分布式一致性协议综述

学　　号 M202073518

姓　　名 陈延良

专　　业 电子信息

课程指导教师 施展　童薇

院（系、所） 计算机科学与技术学院

2020 年 12 月 21 日

分布式一致性协议综述

陈延良[1)]

[1)](华中科技大学计算机科学与技术学院 湖北 武汉 430074)

摘 要 随着互联网的飞速发展，越来越多的信息被数字化。为了面对海量的存储数据，分布式存储系统应运而生。但是如何提供一个高可靠性和高可用性的分布式服务成为一个重要问题。对此，通常的解决办法是采用复制技术在系统中存储多个副本，但是也因此带来了多个副本之间的数据一致性问题。在这个背景下，分布式一致性协议变得不可或缺。本报告综述了多个一致性协议。介绍了传统的分布式一致性协议，如 Paxos、Raft 等。同时介绍几种改进的一致性协议，引入纠删吗技术以减少系统冗余和网络带宽消耗的 RS-Paxos、CRaft 协议；以及通过分离式架构，引入 RDMA 技术，提供一种更细粒度资源分配的 Sift 协议。

关键词 Paxos 协议；Raft 协议；复制状态机；纠删码；RS-Paxos 协议；CRaft 协议；Sift 协议；

Overview of Distributed Consensus Protocol

Chen Yanliang [1)]

[1)](School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, Hubei 430074)

Abstract With the rapid development of the Internet, more and more information is digitized. In order to face the massive storage data, the distributed storage system arises at the historic moment. However, how to provide a distributed service with high reliability and availability becomes an important issue. The usual solution to this problem is to use replication to store multiple copies in the system, but this also brings the problem of data consistency between multiple copies. In this context, distributed conformance protocols become indispensable. This report reviews several conformance protocols. Traditional distributed consistency protocols, such as Paxos, Raft, etc., are introduced. At the same time, several improved consistency protocols are introduced. Rs-paxos and CRaft protocol, which introduce erasure correction technology to reduce system redundancy and network bandwidth consumption, are introduced. And Sift protocol to provide a more granular resource allocation through the introduction of RDMA technology through a split architecture.

Key words Paxos; Raft; State machine replication; RS-Paxos; CRaft; Sift;

1 引言

在分布式系统中为了提高可靠性及可用性会采取冗余技术，即将数据复制到多个副本中，这样系统中有部分节点宕机时还能继续提供服务，但是由于一份数据在多个节点保存，所以在副本之间会存在一致性问题；比如当系统出现分区时，如果客户端继续对系统进行数据写入，可能会导致不同分区的内容不一致，所以需要一致性协议来让节点之间达成共识。

2 背景

常见的一致协议有：Paxos[1]和 Raft[2]等。它们可以容忍分布式服务中的临时故障，通过在每个服务器的日志中保持一致的顺序，使服务器集合成为一个一致的组。使用这些一致协议，能够保证它们总是返回正确的结果，即使可能发生部分机器故障，只要保证一定数量的机器处于正常状态，就能让系统继续保持可用。所以这些协议能够保证安全性和一定的活性。Raft 和 Paxos 已经应用于真实的大型系统如 etcd、TiKV 和 FSS 中。

在共识问题中，要容忍任何 F 故障，至少需要 N = (2F + 1)服务器。否则，网络划分可能会导致分裂的小组对不同的内容达成一致，这是违背共识概念的。因此，使用一致协议来容忍故障可能会导致很高的网络和存储成本，甚至达到原始数据量的 N 倍。由于这些协议现在被应用在大规模系统中，而且数据量越来越大，这些成本成为了真正的挑战，它们会使得系统难以实现低延迟和高吞吐量。

与全拷贝复制相比，纠删码是一种降低存储和网络成本的有效技术。它将数据分割成片段，并对原始数据片段进行编码以生成奇偶校验片段。原始数据可以从任何足够大的片段子集中恢复，因此纠删码可以容忍错误。如果每个服务器只需要存储一个片段(可以是原始数据片段或奇偶校验片段)，而不是数据的完整副本，存储和网络成本可以大大降低。基于以上特性，纠删编码可以很好地解决一致协议中存储长度和网络开销的问题。

3 相关协议介绍

3.1 Paxos协议

Paxos 一致性算法出自于 Lamport, 是分布式系统中的经典算法，算法的引入设计了一个虚拟的希腊城邦 Paxos，这个岛按照议会民主制的政治模式制订法律，但是没有人愿意将自己的全部时间和精力放在这种事情上。所以无论是议员，议长或者传递纸条的服务员 都不能承诺别人需要时一定会出现，也无法承诺批准决议或者传递消息的时间。下面简单介绍下 Paxos 协议；首先在 Paxos 中角色定义如下：

1）Proposer：提议(proposal)发起者，处理客户端请求，将客户端的请求发送到集群中，以便决定这个值是否可以被批准；冲突时进行调剂，可以理解为议员。

2）Acceptor：提议批准者，负责处理接收到的提议，会存储一些状态来决定是否接收一个值。可以理解为国会。

3）Listener（learner）：接受者，不参与决策，对集群一致性没影响，可以理解为记录员。

每一个 process 可能扮演多个角色，以上角色只是一种协议的抽象；对于流程，系统在接收到发来的提议请求(议案)时，可大致分为两个阶段：

1）Prepare 阶段：提议者获取一个议案编号 n，向所有节点广播 prepare(n)请求；接收者接受到的 prepare 请求的编号 n 大于它已经回应的任何 prepare 请求，它就回应已经批准(accept)的编号最高的议案, 并承诺不再回应任何编号小于 n 的议案。

2) Accept 阶段：proposer 若收到大多数 acceptor 的回应，它就广播 accept (n, value)到所有节点；acceptor 接收到请求，它就批准该议案，除非它已经回应了一个编号大于 n 的议案。

Proposer 可以提出多个议案，也可以在任何时候放弃一个议案，只要遵循上面的算法，就能保证其正确性。更复杂的情况这里不做过介绍。Paxos 协议也有不同的版本如 Multi-Paxos、Fast-Paxos 等。

3.2 Raft协议

原始的分布式协议 Paxos 比较晦涩难懂，也没有提供一个足够好的用于构建现实系统的基础，这着重介绍另一个算法 Raft 协议。

一致性算法是基于状态机的（Replicated state machines)，复制状态机通常都是基于复制日志实现

的。状态机从日志中处理相同顺序的相同指令，所以产生的结果也是相同的。所以保证复制日志相同就是一致性算法的工作了。

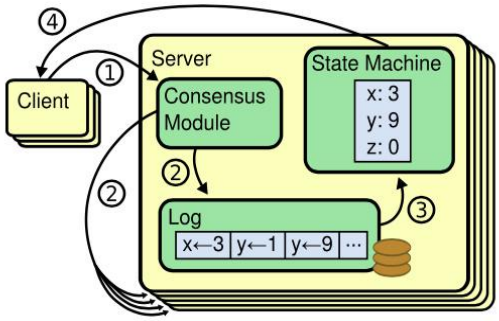

图 3-1 基于状态机和日志的实现

Raft 提供了和 Paxos 算法相同的功能和性能，但算法结构和 Paxos 不同， Raft 算法更加容易理解并且更容易构建实际的系统。Raft 是一个基于 leader 的协议，通过选取一个 leader 然后赋予它全部的管理复制日志的权利，下面介绍三个角色 Leader、Candidate、Followe 及其定义：

1）Leader：领导者，每个阶段只有一个 Leader，数据只从 Leader 流向其他节点，即给予 Leader 全部的管理复制日志责任来实现一致性。

2）Candidate：候选者，一定条件下由 Follower 转变而来，向其他 Follower 节点索要选票，试图成为 Leader。

3） Follower： 跟随者， 响应来自 Leader 和 Candidate 的请求， 在必要的时候可以成为 Candidate。

每个机器只能扮演其中一个角色，Raft 把时间分割成任意长度的 term，term 代表每一任 leader 的任期，如图 3-2，下面分以下几板块介绍 Raft 协议：

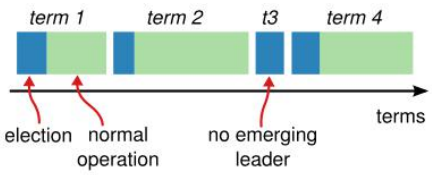

图 3-2 term 时间片

1） Leader election：领导人的选举，当一个节点没有听到 leader 发来的心跳信息时，节点会启动一个定时器，定时器超时则会变为 candidate 即 leader 候选者，然后向其他节点发生请求投票的 RPCs，当收到大多数节点同意时就当选 leader，且从此之后定时发送心跳来反馈给其他 follower 节点，follower 节点通过心跳信息来确认 leader 是否存活。

2） Log Replication：日志的复制，client 与向 leader 发送增加 entry 请求，leader 便会添加一个 entry，但是这个 entry 不会立马提交，他会首先将其复制到其他 follower 节点，如果有大多数节点返回确认信息，则 leader 提交(submit)，同时 leader 再告诉其他 follower 节点，这个 entry 是可以提交的；（这里要注意可以提交和已经提交是两码事，因为这些消息可能因为超时没有正确送达 follower 节点）

3） Newly-election：当然 leader 节点也有可能出现宕机，挂掉之后就会导致节点启动定时器，超时后就会有节点进行新 leader 的申请，这时候考虑一种瓜分选票的情况，若存在多个节点同时竞选 leader，这时可能会导致所有节点都不能达到大多数选票的标准，从而使所有节点均无法竞选成功，所以在 Raft 中采用的是一个随机时延的方式解决，即当无法竞选成功时随机隔一段时间，如果等待的这段时间里依然没有收到 leader 心跳（没有其他节点先于自己当选），则再进行申请。由于引入随机性，当有某个节点重启比较快，在其他节点定时器超时前就将选票送达大多数节点，则可以成功当选；并发送心跳信息来终止其他节点的票选请求。

4） Safty：Raft 使用多种策略保证安全性，日志只能够增加，不能删除或覆盖；Follower 只会同意比自己具有更新的日志条目的节点发送来的请求信息；Leader 节点会让其他节点保持与自己相同；Term 值更高的具有发言权；考虑下面这种情形，对于突然出现的分区网络，首先最多只可能有一个 partition 可以得到大多数选票从而产生一个合法 leader，如果这个 leader 还是之前的 leader 当然不会存在问题，只是待网络恢复后将其信息恢复到其他节点即可；但是如果是一个新节点成为 leader，则说明他与旧 leader 在不同分区，新选出的 leader 的 term 会+1，在分区期间 client 可能会继续向 leader 发送请求，可以预见的是旧 leader 因为缺少足够的选民，所以数据更新并不能成功，而新 leader 会正常更新，所以在分区恢复后会存在不一致问题，Raft 处理策略是旧 leader 的分区发现 term 值小于新 leader 的值，则放弃 leader，新 leader 会将自己的数据同步到其他节点上；所以 Raft 是安全的；同时可以观测到对于 2F+1 节点的集群， Raft 的活性

liveness=F。

3.3 RS码

纠删码 RS 介绍：里德-所罗门码，纠删码是一种通用技术，能够节省存储空间同时还具备一定容错；其原理是一条 k-1 次曲线可以通过 k 个系数或曲线上的点来确定；在编码过程中，将源数据转化为 k 个数据段，将其理解为 k-1 高次曲线的系数，然后通过一定的技巧计算得到 m 个点，这样 k+m 个点中只要保证得到 k 个，就便可恢复曲线并得到系数，也就是恢复得到源数据。

点的构造通常使用单位矩阵和范德蒙矩阵或柯西矩阵构成，如下图；

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ \hline 1 & 1 & 1 & \dots & 1 \\ 1 & 2 & 2^2 & \dots & 2^{k-1} \\ 1 & 3 & 3^2 & \dots & 3^{k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & m & m^2 & \dots & m^{k-1} \end{bmatrix} \times \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \dots \\ d_k \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \dots \\ d_k \\ y_1 \\ y_2 \\ y_3 \\ \dots \\ y_m \end{bmatrix}$$

图 3-3 纠删码原理

对于生成的 d1, d2 ... dk 和 y1, y2 ... ym，一共 k+m 个数据段，只要保证有 k 个数据段（不管是源数据段 d 还是生成的校验段 y），就可以通过逆向算法得到其他所有片段，也就是说可以处理 m 个突发错误。

3.4 RS-Paxos协议

如果只是简单地将 RS 与 Paxos 结合起来是无法工作的，这主要是由于原生 Paxos 是 complete dumplication，所以在原生 Paxos 中只需要读的法定人数与写的法定人数至少有一个交集即可，而这是很容满足的，因为原生 Paxos 中本身就是需要大多数的健康服务器才能正常工作，两个大多数之间必至少有一个交集；但是在 RS-Paxos 中，至少需要 k 个 fragment 才可以恢复，所以必须保证读法定人数和写法定人数交集至少为 k；

如何解决以上问题，很直观的理解就是 Qw 和 Qr 增大就可以，然后再通过集合的数学知识简单推导就得到 Qw+Qr-N>=k，举例如下图：

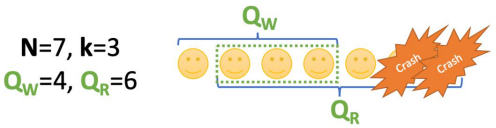

图 3-4 RS-Paxos 举例

但是这样会出现一个问题，那就是系统的活性降低了，原生的 Paxos 是可以容忍 F 个 Crash，但是 RS-Paxos 显然要低，至于低多少，取决与 max{Qw，Qr}：

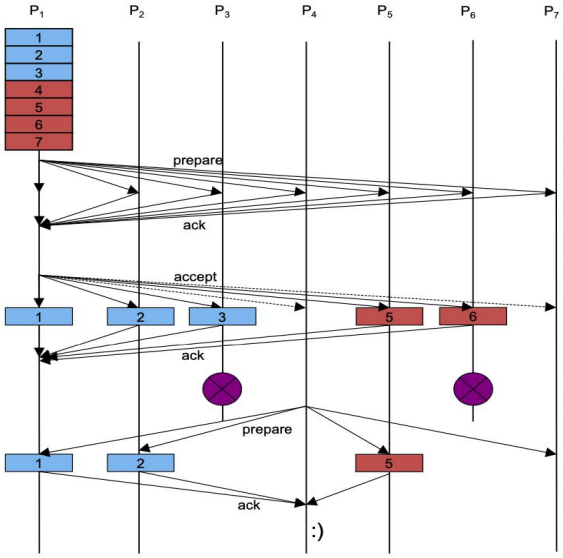

图 3-5 RS-Paxos 应对错误

3.5 CRaft协议

CRaft 协议是基于 Raft 协议的，所以也存在相同的三个角色定义 Leader、Candidate、Follower，这里不再赘述；不同的是 CRaft 引入了两种复制模式，以及一个额外的判断机制 Prediction，并基于 Prediction 的结果选择其中一种复制模式，另外，不同于普通的 Raft，当出现新一轮选举时，即将就任的新 Leader 还需要额外执行一次 LeaderPre operation 阶段，从而保证系统的安全性和可用性，下面将从以下几个板块介绍 CRaft：

1）Coded-fragment Replication：领导人的选举，当一个节点没有听到 leader 发来的心跳信息时，节点会启动一个定时器，定时器超时则会变为 candidate 即 leader 候选者，然后向其他节点发生请求投票的 RPCs，当收到大多数节点同意时就当选 leader，且从此之后定时发送心跳来反馈给其他 follower 节点，follower 节点通过心跳来确认 leader

是 否 存 活 。

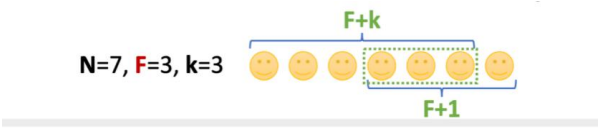

图 3-6 CRaft 举例

2）Complete-entry Replication：前面介绍了正常情况下的传输方式，但是 CRaft 协议是为了实现活性 liveness=F，所以在最糟糕的情况，可能只有 F+1 个服务器节点存活，那当正 常服务器数量介于 F+1 至 F+k 之间时，应当如何处理呢？一个很自然的方法就是，就行全副本数据复制，即 Complete-entry Replication，这样同样考虑读法定人数和写法定人数的交集，这里我们发现写法 定人数和读法定人数均为 F+1，则其交集至少为 1，而这个交集由于是全副 本，故我们完全可以通过这个交集来恢复数据!(实际上，这就是普通的 Raft 协议)，也就是说当正常服务器节点数量少于 F+k 时，CRaft 协议将退化为 Raft 协议。那么现在的新问题是，如何确定哪种复制模式呢，CRaft 引入了一个 新概念 Prediction；

3）Prediction：前面分析得知，当服务器数量大于 F+k 时，进行 fragments 的方式复制数 据，但是若数量小于 F+k 时，用这种方式就可能会出错，所以准确判断正确服 务器数量是非常关键的。所以论文设计了一个 prediction，即通过最新的心跳答 案来预测正常服务器的数量，一般来说这种预测是相当准确的，所以大多数情 况下都能进行正确的复制模式，保证了效率；流程如下：

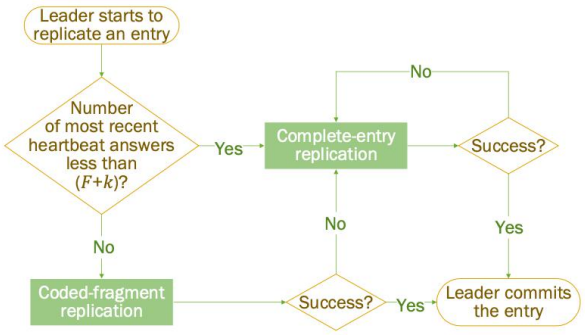

图 2-7 流程图

4）Newly-elected leader：当 leader 宕机掉时，正常情况下，若另一个节点按照多数原则，成为新 leader，为了保证 leader 具有全副本，则需要通过收集 RS 码的至少 k 个数据 段来恢复每一个 entry 的全副本，可以预见的是，所有正常提交的 entry(由于提交的前提是读法定人数和写法定人数满足一定的条件)都可以通过 follower 节点中的 fragments 来恢复数据，但是对于尚未提交的 fragment 需要如何处理呢？在 CRaft 中则引入了一个 LeaderPre operation 来进行一系列的操作保证新 leader 的一些特性。

5）LeaderPre operation：newly-elected leader 检查 log 中未提交的 coded-fragments，向其 follower 查询，至少得得到(F+1)个应答;若 F+1 个应答中包含至少 k 个 fragment 或 者一个完整 copy 则标记为允许提交;否则删除其所有 follower 的 fragment;并且在 LeaderPre 期间保持心跳信息发送避免新一轮的选举。

3.6 小结

本章从最原始的 Paxos 协议讲起，然后介绍了基于 Paxos 协议引入 leader 的易于实际部署的 Raft 协议，协议简单易懂，但是存在大量的相同副本，浪费了带宽和存储空间，接着引入纠删码，但是简单的将 RS 码与 Paxos 或 Raft 结合，是不能正常保持容错的，第一个将 RS 码与一致性协议结合起来的是 RS-Paxos，但是引入了读和写法定人数，存在较大的延迟，且无法保证 2F+1 节点中容忍 F 的错误，然后引入 CRaft 协议，能够使用纠删码降低存储和网络带宽消耗的同时，保持 F 的错误容忍度。但是以上这些协议都是基于 leader 的，所有的节点都可能成为 leader，所以拥有相同的计算资源和存储资源，但在正常运行情况下，这些资源都大大的浪费掉了，下面介绍一种分离式架构的一致性协议，将 CPU 和内存解耦，使用 RDMA 协议，能够提高更细粒度的资源分配，从而到达在大规模系统中大大降低部署成本的目的。

4 Sift 协议

4.1 Sift协议设计

Sift 协议是一种基于 CPU 和内存解耦的分布式一致性协议，通过 RDMA 进行通信。下图是 Sift 协议中节点间的整体架构。

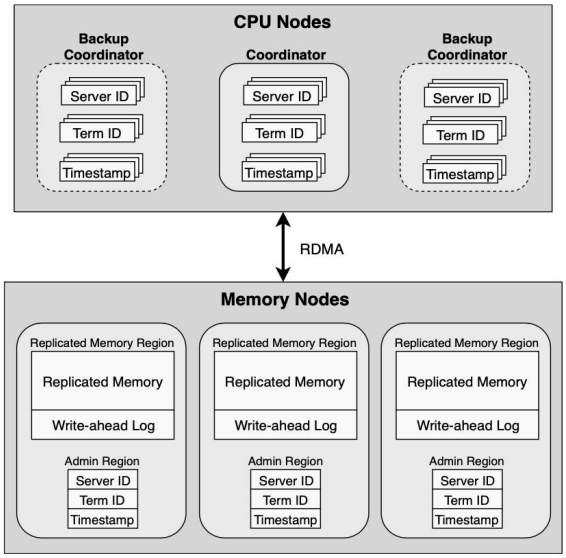

Figure 1: Sift architecture.

1) CPU 节点：

（1）Coordinator：serve client request、replicate the log and state machine

server ID:16bit 每个 CPU 节点都有一个固定的 server ID

term id:16bit 每次 heartbeat 将它的 term_id 加 1

Timestamp:32bit 时间戳

（2）Backup coordinator：heartbeats 信息超时后尝试成为 coodinator

2) Memory nodes：

（1）Replicated memory region：

Replicated memory：复制的内存是一个连续的内存块，客户端通过逻辑地址与之交互

Write-ahead log：write-ahead 日志允许使用一个 RDMA 操作并行提交多个写操作，同时在后台 apply updates 到 replicated memory 中。write-ahead 日志也用于 coordinator recovery.

(2) Admin region：管理区域是一个内存块 memory block，包含 termid、nodeid 和 timestamp 数据，由 CPU 节点用于交换 heartbeat 心跳读/写。

4.2 Replicated memory 层

4.2.1 Coordinator election

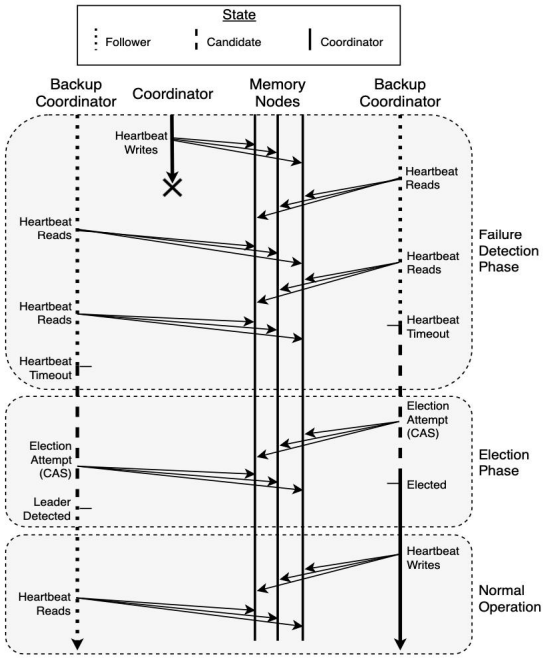

Figure 2: Example of a coordinator election scenario. Two backup CPU nodes compete to become the new coordinator once a coordinator failure is detected.

coordinator CPU 节点通过 RDMA 的 CAS 操作周期性的发送 heartbeat 到每一个 memory 节点的 admin region 区域；backup CPU 节点依次从这些 memory 节点中读取 heartbearts，然后 follower 将 timestamp 和上一次读取的 timestamp 作对比，如果没更新则成为 candidate 并触发一次新的选举 (引入一个 election timeout 参数来决定 heartbeat 读取频率) 同时也可以配置为允许错过多次 heartbeats，并在每次写入后更新；

backup CUP 节点过渡到 candidate 后, 它会将它的本地 termid+1，然后尝试一个 RDMA 的 CAS 操作向所有的 memory 节点发送消息 (包括 nodeid 和 term_id)

若多个 candidate 节点竞争，但是最多只有一个能成功；第一个在大多数 memory 节点 CAS 操作成功的 CPU 节点将成为 coordinator, 而其他的节点看到其他节点成功竞选（从 CAS 操作的返回值判断）后就会回落到 followe 状态；若果没有 candidate 竞选成功，每一个 candidate 将执行一次随机回退周期，然后再重新启动 CAS 操作，并且使用上一次失败 CAS 操作的返回值。同样每一次重新尝试 term_id 要+1；

4.2.2 Normal operation

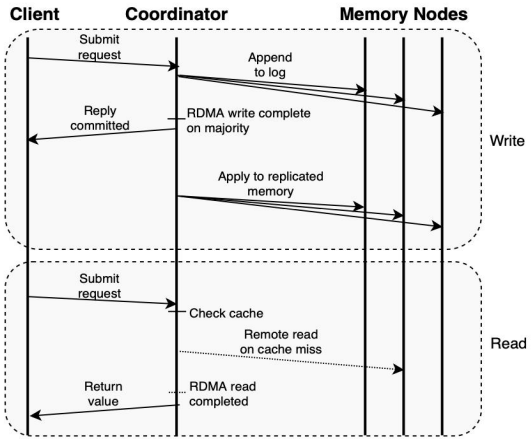

Figure 3: Messages exchanged for reads and writes.

1) Read request：读请求

Clients 使用特定的地址和大小来向 coodinator 发送读请求。coordinator 为相应块获取一个本地读锁，再使用单边 RDMA 从内存节点读取值；而且没有必要达到 quorum，因为所有的请求都是被 coordinator 处理的，所以它可以有效地维护整个复制内存的读租约。

2) Write request：写请求

coordinator 为相应块获取一个本地写锁（为了防止读取到脏数据，一旦提交就释放该锁），并使用单边 RDMA 将更新附加到 memory 节点上的 write-ahead log 中；coordinator 可以通过 RDMA 协议的可靠变体，接收到大多数 memory 节点附加成功的 ack，于是就能 reply committed 给客户端；coordinator 接着告知 memory 节点在后台更新 replicated memory；还提供了一个接口，可以一次接受多个写请求（不能与其他写请求冲突）；还允许直接写入复制的内存区域，而不进行日志记录。

4.2.3 Fault recovery

1) Coordinator failure：容忍 Fc 个 CPU 故障只需要 Fc+1 个 CPU 即可

① backup CPU 节点可以通过读取 heartbeat 机制来检测是否需要进入 coordinator election

② 新 coordinator 读取所有 memory 节点的循环日志来构造一致的、最新的 log 版本来进行 log recovery

2) Memory node failures：容忍 Fm 个 Memory 故障需要 2Fm+1 个 memory 节点

① coordinator 一直记录存活的 memory 节点，如果某个 memory 节点失败，就将它从存活的 memory 节点列表中删除。并且后台又一个恢复线程定期轮询所有失败的内存节点

② 当 memory 节点重新恢复时，coordinator 须将 replicated memory region 中的所有数据复制到该 memory 节点。这是通过逐渐的持有相应区域的读锁实现全部数据的复制（确保在此期间不发生更新操作，但可以读）

4.3 Key-Value 层

4.3.1 Architecture

Sift 协议程序状态机的常用表达形式（可以理解为实例化）在 Replicated memory layer 的上部，它与 Replicated memory layer 交互，将其作为它的本地内存(通过逻辑地址访问)；管理 key-value store 的进程与 Sift coordinator 进程共存。

KV 存储使用链式哈希表, 同时有四个主要的数据结构：(所有这些结构都存在于 replicated memory 中的预定义位置)

① an array of data blocks：一组数据块，每个数据块都是预定义的大小

② an index table：index 表，保存 data blocks 的指针，同时也 cash 在 coordinator 处

③ a bitmap：保存可用的 data blocks，同时也 cache 在 coordinator 处

④ a circular write-ahead log：与被 replicated memory 系统使用的 log 是分离的

4.3.2 Put request

Put 请求处理过程, 首先将更新附加到 key-value store 的 write-ahead log 中，这个 write-ahead log 位于 replicated memory 上（是支持直接写的，不用 log），这样就能在一次 RDMA roundtrip 中就让 put requests 被提交。一旦 log 写完成，就也意味着更新已经被提交到 replicated memory，我们就可以 reply client，然后在后台 apply 被记录的 put request。由于 write-ahead log 是 circular（循环的而不是 append），所以未完成的（已 log 但未 apply）的更新数量会受到日志大小限制

若要 apply 更新，先使用键的散列在 index table 中执行查找。如果 index table 中没有 entry，则使用 bitmap 分配一个新的 block，并将其指针插入 index table 中。否则，就遍历 index table entry 的指针所指的链（使用每个已分配的 block 中的 next

pointer)。如果找到具有相同 key 的 block，则更新该 value 并将其写回 replicated memory。如果链中不包含该 key，将分配一个新的 block 并添加到链中，并将其指针写入上一个块的 next pointer

4.3.3 Get request

get 请求的处理过程，首先检查缓存。如果该键不在缓存中，则从 replicated memory 中检索该值。为了确保一致性, 我们的 cache 记录 entry 是否已经 applied，并且也不会驱逐那些有未决更新(pending updates) 的 entry 条目。

4.3.4 Failure handing

键值层只需要从管理 key-value store 的进程的失败中恢复。新的键-值进程首先从 replicated memory 中加载 index table 和 bitmap。使用 memory 中的这些结构，它 read and replay write-ahead log 的内容即可。replay 之后进程就可以开始处理 client 请求；其他故障归结为 replicated memory layer 中 memory 节点和 coordinator 节点的故障（见上）

4.4 SiftEC 协议

Sift 的分离式架构以及集中复制逻辑, 引入纠删码是非常容易的，修改 replicated memory 结构，将每个块分割成 Fm+1 块, 并使用 Reed-Solomon 代码生成；通过以 non-encoded 的形式存储 write-ahead log 来处理 coordinator 在 Fm+1 个 memory 节点收到它们的对应 block 之前发生故障的情况；值得一提的，引入纠删码后 SiftEC 仍然能够在 2F+1 的节点中保持 F 的活性。

5 总结

在本论文综述中，从最经典的 Lamport 大师的 Paxos 论文出发

1）先查阅 Raft 论文，Raft 协议基于 leader 易于实现，已用于实际应用；

2）然后介绍了第一个引入纠删码的 RS-Paxos 协议；

3）鉴于 RS-Paxos 协议的较低活性，提出改良版本的 CRaft 协议；

4）然后从资源节约角度出发，Sift 提出 CPU 资源与内存资源解耦的分离式架构，同时也能够应用纠删码并保证和 Raft 相同活性的 SiftEC；

对以上这几个协议的了解中，对分布式一致性协议的原理以及各个协议的优缺点又有了进一步认知。一个好的一致性协议不仅需要较高的可靠性和可用性，也需要精简的设计使得易于部署，并能够维持较高吞吐量的同时尽可能的降低成本。

参 考 文 献

[1] Diego Ongaro and John K. Ousterhout. In search of an understandable consensus algorithm. In Proceedings of the 2014 USENIX Annual Technical Conference, (USENIX ATC '14), pages 305–319, 2014.

[2] Leslie Lamport. The part-time parliament. ACM Trans- actions on Computer Systems (TOCS), 16(2):133–169, 1998.

[3] LAMPORT, L. Paxos made simple. ACM SIGACT News32, 4 (Dec.2001), 18–25.

[4] ShuaiMu, KangChen, YongweiWu, and WeiminZheng. When Paxos meets erasure code: Reduce network and storage cost in state machine replication. In Proceedings of the 23rd International ACM Symposium on High-Performance Parallel and Distributed Computing (HPDC'14), pages 61–72, 2014.

[5] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. Journal of The Society for Industrial and Applied Mathematics, 8(2):300–304, 1960.

[6] SCHNEIDER, F. B. Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Computing Surveys 22,4 (Dec.1990), 299–319. In Proceeding of the 2020 USENIX Symposium on Networked System Design and Implementation (NSDI'20), pages 157-180, 2020.

[7] Raft consensus algorithm website. http://raft.github.io.

[8] The Secret Lives of Data website. http://thesecretlivesofdata.com/raft.