文章编号: 1000-5641(2019)05-0168-10

基于 Paxos 的分布式一致性算法的实现与优化

祝朝凡, 郭进伟, 蔡 鹏

(华东师范大学 数据科学与工程学院, 上海 200062)

摘要: 互联网的不断发展, 企业的信息化程度不断加强, 不计其数的数据需要得到及时处理. 但是网络环境不稳定, 容易发生数据丢失、节点宕机, 从而造成严重后果. 因此, 构建可以容错的分布式存储系统变得越来越受欢迎. 为了保证系统的高可用性和一致性, 需要引入分布式一致性算法. 为了提高系统在不稳定网络下的性能, 传统基于 Paxos 的分布式系统允许日志中存在空洞. 然而, 当节点进入恢复状态时, 这些系统通常需要大量网络交互来补全日志空洞, 这极大地增加了节点恢复的时间, 从而影响了系统的可用性. 针对节点恢复过程中补全日志空洞代价过大的问题, 本文重新设计了日志项结构, 优化了数据恢复流程, 通过实验模拟, 验证改进的基于 Paxos 的一致性算法的有效性.

关键词: 分布式存储系统; 一致性; 日志复制; 节点恢复

中图分类号: TP392 **文献标志码**: A **DOI:** 10.3969/j.issn.1000-5641.2019.05.014

Implementation and optimization of a distributed consistency algorithm based on Paxos

ZHU Chao-fan, GUO Jin-wei, CAI Peng

(*School of Data Science and Engineering*, *East China Normal University*, *Shanghai* 200062, *China*)

Abstract: With the ongoing development of the Internet, the degree of informationization in enterprises is continuously increasing, and more and more data needs to be processed in a timely manner. In this context, the instability of network environments may lead to data loss and node downtime, which can have potentially serious consequences. Therefore, building distributed fault-tolerant storage systems is becoming increasingly popular. In order to ensure high availability and consistency across the system, a distributed consistency algorithm needs to be introduced. To improve the performance of unstable networks, traditional distributed systems based on Paxos allow for the existence of holes in the log. However, when a node enters a recovery state, these systems typically require a large amount of network interaction to complete the holes in the log; this greatly increases the time for node recovery and thereby affects system availability. To address the complexity of the node recovery process after completing a hole log, this

收稿日期: 2019-07-28

基金项目: 国家重点研发计划(2018YFB1003303); 国家自然科学基金 (61432006)

第一作者: 祝朝凡, 男, 硕士研究生, 研究方向为分布式数据库. E-mail: chaofanzhu2@163.com.

通信作者: 蔡 鹏, 男, 副教授, 研究方向为高性能事务处理. E-mail: pcai@dase.ecnu.edu.cn.

paper proposes a redesigned log entry structure and optimized data recovery process. The effectiveness of the improved Paxos-based consistency algorithm is verified with experimental simulation.

Keywords: distributed storage systems; consistency; log replication; node recovery

0 引　　言

在互联网技术和大数据技术不断发展的情况下, 传统的集中式数据管理模式已经不再适应企业的需求, 分布式的集群管理模式应运而生.

数据量的急剧增加要求存储系统进行相应升级, 使其存储容量、处理性能都得到巨大的提升, 但是这种 scale up 的方式比较昂贵, 在经过几次更新后就会到达瓶颈[1]. 考虑使用 scale out 的方式来进行系统扩展, 利用物理上分开的多个节点来逻辑上形成一个数据中心, 从而使得数据中心的容量、I/O 带宽和处理性能得到质的飞跃[2]. 单个数据中心为了提供服务, 需要的物理机数量也变得越来越多, 然而物理机的故障是不可避免的, 容易使得该数据中心停止服务, 从而使系统的可用性变差[3].

针对单个数据中心容错能力差, 容易形成性能瓶颈. 采用副本状态机的机制, 利用副本冗余功能, 将单个节点上的数据通过网络上的消息传递方式同步到多个副本上. 在分布式系统中, 这些副本虽然物理上是分开的, 但是逻辑上可以统一对外提供服务, 这样有效地提高了系统的可用性, 同时解决了负载均衡的问题.

分布式系统虽然可以将读写服务转发到多个副本上, 但是却引入了一致性的问题. 根据 Eric Brewer 教授在 2000 年提出的 CAP 理论[4], 在分布式系统中, 数据一致性, 分区容忍性, 系统可用性是不可能同时达到的, 只能保证其中两项可以达到. 而由于在互联网交互应用中, 网络不稳定的情况总是存在, 网络分区始终是不可避免的, 从而在设计分布式系统时, 总是考虑如何在数据一致性和系统可用性上进行取舍. Basic Paxos算法[5]的提出为分布式系统中的一致性研究指出了明路, 本文主要在 Paxos 算法基础上进行研究, 通过改进多个 Paxos 协议的实例对一系列决议达成的过程, 优化并实现了改进的基于 Paxos 的一致性算法来更好地保证副本的一致性要求.

本文的主要贡献包括下面几点.

(1) 为了追踪冗余日志信息, 重新设计日志记录, 在 LogEntry 结构中增加一个数组. 在主节点中缓存最近连续多条日志项. 对于当前日志项, 检查前面多条日志项中的日志内容, 如果日志内容和当前日志项中的内容都是对相同数据项的处理, 那么将对应的日志项索引记录在这个数组中.

(2) 主节点宕机以后, 新主节点在重新提供服务之前, 需要进行数据恢复过程, 将新主节点的状态恢复到最新一致状态. 重新设计数据的恢复算法, 由于日志之间存在空洞, 利用优化后的日志项, 可以省略一些空洞日志的恢复, 降低不必要的网络交互.

(3) 实现基于 Paxos 的分布式一致性算法原型, 并对比了使用新日志项的恢复算法和未使用新日志项的恢复算法在节点恢复到最新一致状态时所用的时间, 证明了新的恢复算法的有效性.

1 相关工作

在分布式系统领域, 一致性问题一直是一个非常重要的研究方向, 涌现了非常多的一致

性算法. 两阶段提交协议是一个众所周知的一致性算法[6], 其中有两种重要的角色, 一种为协调者, 一种为参与者. 在数据一致性模型中, 可以将参与者理解为一个个数据副本. 为了达成数据的一致性, 在两阶段的请求阶段, 协调者请求所有数据副本就某一项值达成一致性. 在提交阶段, 协调者只有收到所有的数据副本同意请求的响应, 才会进行提交操作, 从而达成数据副本的一致性. 然而两阶段提交协议容易产生阻塞问题, 为了系统的强一致性, 任何请求都需要得到明确的响应, 否则系统的资源会被锁定, 造成资源浪费. 并且协调者在系统中占有非常重要的位置, 协调者的宕机会造成整个系统的阻塞. 在两阶段的提交阶段, 并不能保证所有的参与者都顺利进行提交, 因为这个过程可能产生网络异常, 从而造成数据副本的不一致性.

在两阶段提交协议的基础上, 一些研究者做了相应的改进, 提出了三阶段提交协议[7]. 三阶段提交协议采用了超时机制, 同时引入了一个 CanCommit 阶段. 三阶段提交协议可以在某种程度上降低参与者的同步阻塞问题, 但是仍然可能造成数据副本的不一致性问题.

Gifford 提出了 Quorom机制[8], 其主要思想来自数学中的"鸽槽原理". 在 N 个副本的系统中, W 表示更新成功所需要的副本数量, R 表示读成功需要的副本数量. 则需要满足 $W+R>N$, $W>N/2$ 的限定. 因而 Quorom 能够保证一个数据副本不会被同时读或写, 且能够满足数据的串行化更新.

Leslie Lamport 于 1990 年提出了基于消息传递的 Basic Paxos 算法, 该算法可以在保证可用性的前提下很好地提供一定的副本一致性. Basic Paxos 包含两个阶段, 阶段一为 prepare 阶段, 阶段二为 accept 阶段. Paxos 算法利用多数派原理, Proposer 每次在发送给所有的 acceptor 提议后, 只要在超过半数的 acceptor 上更新成功, 就表示本次提案被确认下来了, 最终所有的 acceptor 能够知道这个被确认的提案. 在分布式领域, 进行多次的 Paxos 过程就可以得到一系列的一致的日志序列, 将一致的日志序列应用于副本状态机上, 从而保证副本的一致性. 但是基本的 Paxos 工程化比较难, 随着研究不断深入, 基于 Paxos 的研究越来越成熟, google 公司的分布式锁结构 chubby[9]、megastore[10]和分布式数据库 spanner[11]都是很好的 Paxos 的工程实现. 腾讯公司近些年来利用 Paxos 也实现了一个高可用, 强一致的存储系统 PaxosStore[12], 在提供多主多写的服务化过程中, 需要对基本的 Paxos 协议采取一些约束.

科学家们不断努力, 对 Paxos 的各方面进行了不同程度的改进, 形成了一系列类 Paxos 算法. Multi-Paxos[13]通过利用强领导者, 在领导者正常运行的时间里, 所有提议都由该领导者提出, 同时可以缩减提议达成一致性过程中所需要的步骤, 保证了在实际情况下对一系列提案快速达成一致性. Viewstamped replication 大多数应用在分布式文件系统中来替换两阶段提交协议[14-15]. Cheap-Paxos[16] 使用辅助的节点来保证分布式系统的正常运行, 在 $2N+1$ 个节点组成的系统中, 当出现超过 N 个节点出错的情况下, 使用辅助节点来保证系统的正常运行, 当出错的节点重新加入以后, 替换掉原来的辅助节点. Mencius 算法[17]提出, 为了达到负载均衡的目的, 各个副本可以分别轮流担任主副本. 为了降低分布式系统中的时延, Generalized Paxos 提出[18], 在各个提议之间不冲突的前提下, 这些提议可以被同时表决, 提高了并发的程度, 可以有效降低时延, 提高系统性能. 针对 Paxos 算法难以工程化、难以理解的问题, Diego Ongaro 设计了 Raft 算法[19], 将基本的 Paxos 算法分为一系列的小步骤, 以求明确地描述一致性算法步骤. Raft 算法需要保证日志的连续性, 不像 Multi-Paxos 算法一样允许日志中存在空洞, 同时消息传递只能从主节点流向备节点, 在重新选主的过程中只有

保存有最全日志、最新日志的节点才有可能成为新主节点. Epaxos 算法为了负载均衡, 允许副本从客户端接受请求, 是一个无主的协议, 可以让副本只通过一次通信便可提交提议[20]. 本文在 Paxos 基础上, 尝试对分布式一致性相关问题进行更进一步的研究.

2　基于 Paxos 的分布式系统的实现与优化

2.1　主节点选举

为了减少多个节点同时提出提议, 造成最终确认一个提议的延迟太大,通过选举出一个主节点负责相关业务[21]. 每当客户端有请求时, 直接由选举出来的主节点负责处理请求, 接着将提案通过消息传递给其他副本. 通过租约机制来保证主节点身份.

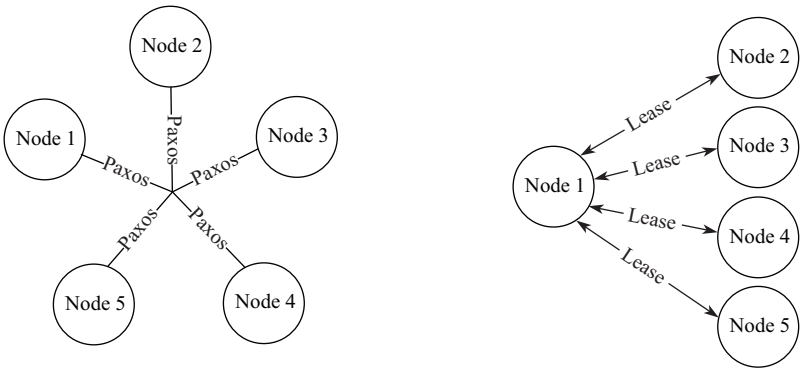

图 1　主节点选举

Fig. 1　Master node election

这里的选主流程是一个基本的 Paxos 过程, 提议值换做“选自己为主节点”. 以图 1 由 5 个节点组成的系统为例, 说明如何在系统中正确地选出一个主节点. 这里的 5 个节点都可以发起提案. 下面以 Node 1 发起提案为例子说明流程, 初始提案值为选 Node 1 为主节点.

步骤一:

Proposer 选择一个初始提议号 proposerID, 然后向其他备节点发送自己的请求. 接着等待其他备节点的回应.

由于所有的其他备节点保存有已经接受过的提案号 acceptedProposalID, 已经接受过的提案值 acceptedValue, 已经响应过的提案号 minProposalID. 所以当 minProposalID > ProposerID 时, 则不响应 Proposer, 当 minProposalID ⩽ proposerID, 则向 Proposer 返回自己的 acceptedProposalID 和 acceptedValue.

步骤二:

如果 Proposer 在一定时间没有收到超过半数节点的回应, 那么 Proposer 提高自己的 proposerID, 重新开始步骤一. 如果 Proposer 在一定时间里收到超过半数节点的回应, 则可以通过步骤一发现可能的提案值. 如果步骤一返回给 Proposer 的 acceptedValue 不为 null, 则将自己的提案更新为响应中最大的提案号对应的那个提案值; 否则使用 Proposer 初始化的时候自己的提案值. 接着 Proposer 向系统其他节点发起 Accept 请求 (ProposerID, Value).

当 Acceptor 接受到 Proposer 的 Accept 请求时, minProposerID > proposerID, 则不接受提案值; minProposerID ⩽ proposerID, 则接受请求. 在一定的时间里, proposer 接受到超过半数的节点的回应, 则表示提案值已经被确定, 即选出了谁为主节点, 否则重新发起步骤一.

当然使用基本的 Paxos 算法进行选主, 有可能发生活锁, 在实现过程中, 通过在再次发起 PrepareRequest 请求前加入随机等待时间来减少活锁发生的可能性. 同时设置各个节点的优先级, 如果提议者超过一个设定的时间未选出主节点, 然后可以使用节点的优先级来进行辅助选主. 这就保证了可以最终选出主节点. 当选出了主节点以后, 主节点通过维护各个备节点的租约信息来保证自己的合法身份, 如图 1 所示, 当 Node 5 上关于 Node 1 主节点的租约信息过期后, 那么 Node 5 会发起“主节点下线的提案”, 如果超过半数节点关于 Node 1 主节点的租约信息已经过期, 则该提案最终会通过, 否则失败. 只有该提案通过了以后, Node 5 才会重新发起选主进程, 这样可以有效地避免由于 Node 5 自身原因而误认为 Node 1 节点已经下线的情况.

2.2 日志复制

2.2.1 日志结构的优化

其中一个日志项包含的内容如下面的结构体所示, 其中 logIndex 表示该日志项在日志文件中的索引位置, acceptedProposalID 表示该日志项已接受提议的提议号, acceptedValue 表示该日志项的内容, includeIndex 数组存放的是该日志项前面 9 个日志项中满足相关条件的日志索引值, 其中 includeIndex 数组的存在是为了后面节点宕机恢复和新主节点恢复服务的. 对于当前日志项, 检查前面 9 条日志项中的日志内容, 如果日志内容和当前日志项中的内容都是对相同数据项的处理, 那么将对应的日志索引记录在 IncludeIndex[9] 这个数组中. 这样在重放日志的时候可以跳过满足条件的日志项, 避免多余网络交互, 从而加快新主节点达到最新的一致性状态.

```
struct LogEntry
{
int logIndex;
int acceptedProposalID;
string acceptedValue;
int includeIndex[9];
};
```

日志的文件结构如图 2 所示, 这里暂时不考虑日志检查点. 日志的文件中包括日志项, 除去日志空洞外最小的未提交的日志项的索引值 (firstUnchonseIndex), 该节点已接受的日志项的最大索引位置 (lastLogIndex). 为了描述方便, 在图中没有展示出每一个日志项的全部内容. 在日志文件中, 当一个日志项已经被确认提交了, 需要在该日志项上做上标记, 这里采用将原来日志项的 acceptedProposalID 记录成无穷大的方法. 因为 acceptedProposalID 记录成无穷大表示该日志项不会再被更改了. 在原来的方法中, 如果需要检索某一个日志项, 需要进行全局的 Paxos 检索流程, 以便得到的日志项是被大多数节点接受的日志, 但是进行全局的 Paxos 需要进行网络中的消息传递, 代价太大. 通过这个方法, 以后检索相关的日志便可以直接在本地的日志文件中进行检索. 同时, 需要在日志文件中记录 firstUnchosenIndex, 表示本地中尚未提交的最小的日志项. 因为, 基于 Paxos 的分布式一致性算法允许日志的空洞存在, 当检索的日志项在本地日志文件中为空时, 还是需要通过消息传递从其他节点中

获得.

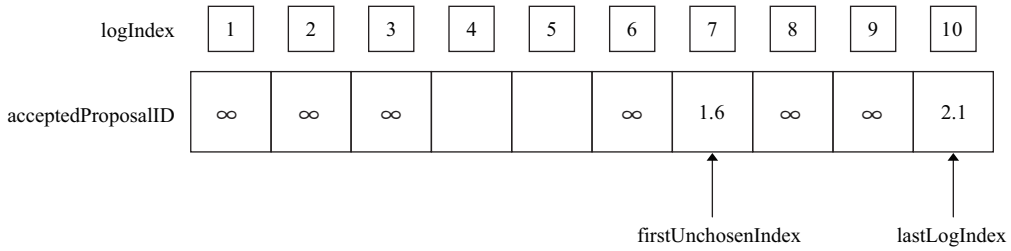

图 2　日志文件结构

Fig. 2　Structure of log file

2.2.2　日志同步

当选主进程结束, 系统中存在唯一的主节点, 主节点负责处理客户端的读写请求, 然后将相关事务日志同步给备节点. 图 3 描述了客户端一次写请求的过程,首先由于系统中存在 Leader 节点, 所以当 Leader 在正常使用 Paxos 协议进行某一次日志同步过程成功了以后, 则代表不存在投票冲突的问题. 对于接下来的日志项的同步可以省略掉 Paxos 协议的 prepare 过程, 而只进行 accept 过程.

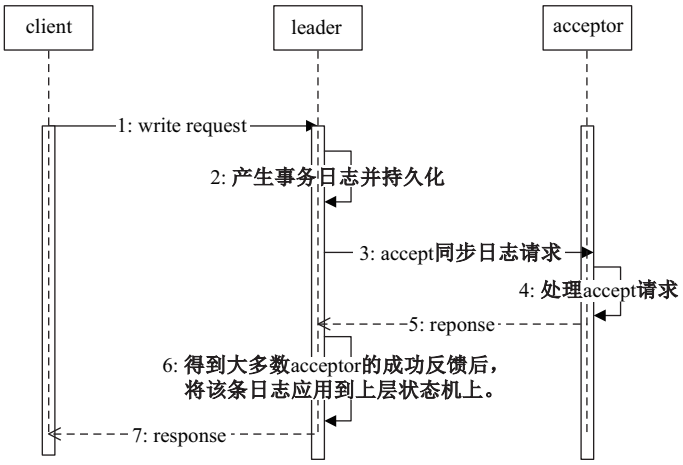

图 3　写请求过程

Fig. 3　Process of write requests

由于系统中只存在一个 Leader 节点, 不存在投票过程中互相阻塞的问题, 所以同步日志在大多数情况下只需要进行如下几个步骤.

(1) 主节点广播 accept(logEntry,firstUnchosenIndex) 请求到所有的 acceptors 上.

(2) acceptor 收到 Accept 请求以后, 对比 logEntry.logIndex 位置处的日志项上的 minProposalID(acceptor 已经响应过的 prepare 请求的最大提议编号)和 request.logEntry.acceptedProposalID 的大小.

如果 minProposalID ⩽ request.logEntry.acceptedProposalID, 则将该 acceptor.minProposalID = request.logEntry.acceptedProposalID, 同时在 acceptor 上持久化传过来的日志项, 如果 acceptor 对应的 index 位置处已经接受过日志项, 则用 request.logEntry 中的内容更改对应日志项.

在 acceptor 上满足 index < request.firstUnchosenIndex 处的日志项, 且日志项中的 acceptedProposalID 等于 request. acceptedProposalID, 则将它们日志项中的 acceptedProposalID 重新设置成无穷大. 然后返回 acceptor 的 minProposalID 和 firstUnchoseIndex 给 Leader.

(3) 主节点接收到 accept 的回应 (reply.n,reply.firstUnchosenIndex), 如果 reply.n > 本节点的提议号, 则表示系统中出现了新主节点, 停止本节点的进程. 如果 reply.firstUnchosenIndex ⩽ lastLogIndex, 并且对应的索引位置处的日志项已经提交, 则将主节点对应的日志项同步到 acceptor 对应的索引位置处.

(4) 主节点接受到大多数 acceptor 对 accept 请求的回应, 则将本地对应的日志项的提议号设置成无穷大, 表示同步日志步骤成功.

2.3 节点恢复的优化

因为基于 Paxos 的分布式一致性算法允许空洞存在的性质, 除了主节点, 任何一个副本节点拥有的日志文件中的日志可能都是不完整的, 所以主节点宕机后, 新主节点重新工作之前, 需要进行数据恢复, 使得新主节点达到最新的状态. 而由于在写日志的时候, 日志项中多了 inclueIndex[9] 数组, 其中记录的是该日志项前面 9 个日志项中与当前日志项处理的是同一个数据元素的日志索引值. 在数据恢复的时候, 若已提交日志的日志尚未全部回放. 则可以 10 个日志项为一组进行数据恢复, 先剔除掉该 10 个日志项中可以不需要恢复的日志项, 如果剔除掉的日志项恰好是日志空洞, 则可避免补全日志空洞的多余操作. 对于尚未提交的日志, 需要通过全局的 Paxos 获得多数派节点认可的最大的 max_logIndex. 然后在 firstUnchosenIndex 到 max_logIndex 之间每一项都进行全局的 paxos 过程, 确保恢复的日志是达成多数派认可的日志, 接着将这些日志回放, 保证节点的状态机恢复到最新的状态, 保证系统的一致性. 详细的过程如下算法 1.

算法 1 节点恢复算法

```
if 已提交的日志没有全部回放
    从日志文件中获取 firstUnApplyIndex= 还未回放的日志起点
    从日志文件中获取 firstUnchosenIndex= 最小的未提交的日志项的索引值
    for(i = firstUnApplyIndex to firstUnchosenIndex-1)do
        以 ⩽10 个日志项为一组, 利用 includeIndex[9] 排除不需要回放的日志项
        剔除后的这组日志如果还有空洞, 利用全局 paxos 获取日志
        将这一组日志回放
    end for
end if
 通过全局 paxos 获取多数派认可的最大日志号 =max_logIndex
 for(i = firstUnchosenIndex to max_logIndex)do
    针对每一项日志进行全局的 paxos 过程
    if(获得大多数节点的确认)
        提交日志号为 i 的日志
        回放日志号为 i 的日志
    end if
 end for
```

3　实验结果

3.1　实验环境

本次实验的机器配置如下: Intel(R) Core(TM)i7-7700 CPU, 内存为 8 G, 100 GB 的硬盘, 64 位 CentOS-6.5 操作系统, 千兆以太网. 实现的改进的基于 Paxos 的一致性算法原型采用的是 java 语言, java 的运行环境为 jdk1.8.0_212.

3.2　选主时间的测试

系统的主节点发生故障以后, 会发生重新选主的进程. 选主的算法采用基本的 Paxos 算法,但是有可能产生活锁, 为了降低活锁, 可以在不同的 Proposer 重新开始 prepare 过程时设置随机的 sleep 时间, 使得 Paxos 进程能够快速选出主节点. 为了测试改进的选主方法能够快速选出主节点, 本次实验分别测试了不同节点数量时的单纯选主时间, 在这里暂时不考虑数据的恢复时间, 因为选主是单独的进程, 所以没有进行数据恢复不会对实验造成影响. 由实验结果所示, 系统能够快速选出主节点, 但是可能随着节点数量的增加, 选主时间会有所增加. 实验结果如图 4 所示.

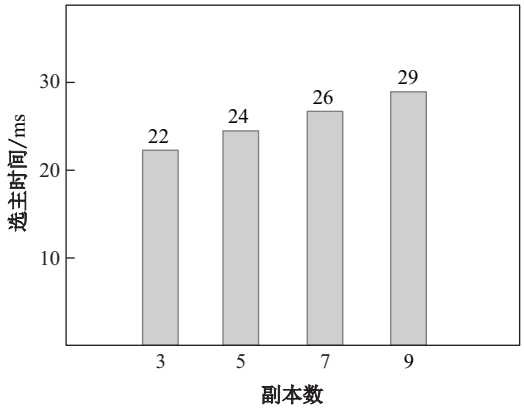

图 4　选主时间

Fig. 4　Time of election

3.3　节点恢复的测试

由于基于 Paxos 的分布式一致性算法的固有性质, 为了确定一系列的日志, 其中日志文件中必然有空洞日志. 而在将日志文件应用到上层状态机的过程中, 需要进行补全空洞的操作, 补全空洞的操作涉及到和其他节点的交互. 本次实验总共设置了 5 个节点, 其中一个为主节点. 为了测试本文所描述的恢复策略的性能, 副本节点暂时不把日志应用到上层状态机上, 当主节点宕机后, 某个副本节点通过选主流程成功当选为新主节点后, 才会将日志应用到状态机上, 以便达到旧主节点的最新状态, 继续对外提供服务. 主节点总共写入了 1 G 的日志文件, 然后宕机, 通过一定方法控制日志文件中的空洞日志比率分别在 0.1‰, 0.2‰, 0.3‰, 0.4‰, 0.5‰ 情况下, 测试新主节点在两种恢复方法下恢复到最新状态所用的时间. 实验结果如图 5 所示. 由实验结果可知, 随着空洞日志比率的增加, 两种方法的恢复时间都会增加. 而在使用了改进的恢复策略后, 由于在日志同步的过程中, 每个日志项中记录了它前 9 个日志项中可以省略恢复的日志号, 如果空洞日志恰好是可以忽略恢复的日志项, 则避免了补全空洞, 减少了恢复的代价. 由此可知, 在日志空洞比较多, 且日志项之间的内容多是对同一类数据的写记录的情况下, 使用改进的恢复策

略可以有较好的性能.

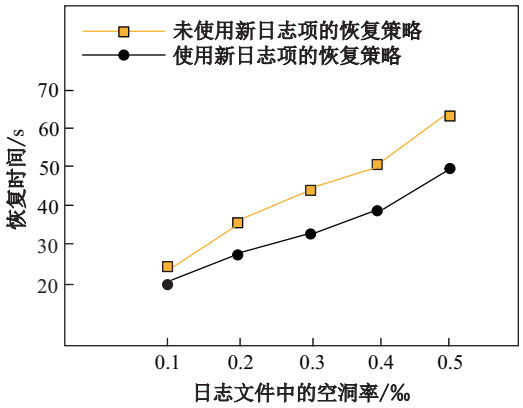

图 5 恢复策略对比

Fig. 5 Comparison of recovery strategy

4 结 论

随着大数据时代的到来, 分布式系统越来越受欢迎, 但是也带来了一致性的问题. 本文实现并优化了基于 Paxos 的分布式一致性算法. 通过选举出一个主节点来对客户端进行同步读写请求, 减少多个节点同时提出提案造成的网络信息交互. 针对节点数据恢复过程中补全空洞日志代价过大的问题, 通过重新设计日志项结构, 在日志项中新增了一个数组结构, 然后设计新的恢复算法来避免对一些空洞日志的恢复. 当主节点宕机后, 使得新主节点能够快速进行数据恢复, 继续对外提供服务. 最后, 通过模拟实验, 证明了改进的基于 Paxos 的分布式一致性算法的可用性.

当然, 分布式一致性是一个具有挑战的领域, 仍然有许多值得研究的问题. 例如如何实现负载均衡, 防止主节点负载过大, 在保证一致性的前提下, 提高系统的吞吐率. 近年来, 随着 RDMA 技术的革新, 它可以将数据直接从一台机器的内存传输到另一台机器中, 不需要经过被访问节点的 CPU, 这也给协议的优化带来了机遇[22].

[参 考 文 献]

[1] ADDISIE A, BERTACCO V. Collaborative accelerators for in-memory MapReduce on scale-up machines[C]//Proceedings of the 24th Asia and South Pacific Design Automation Conference. New York: ACM, 2019: 747-753.

[2] APPUSWAMY R, GKANTSIDIS C, NARAYANAN D, et al. Scale-up vs scale-out for hadoop: Time to rethink?[C]//Proceedings of the 4th annual Symposium on Cloud Computing. New York: ACM, 2013: 20.

[3] KRASKA T, PANG G, FRANKLIN M J, et al. MDCC: Multi-data center consistency[C]//Proceedings of the 8th ACM European Conference on Computer Systems. New York: ACM, 2013: 113-126.

[4] MUÑOZ-ESCOÍ F D, DE JUAN-MARÍN R, GARCÍA-ESCRIVÁ J R, et al. CAP theorem: Revision of its related consistency models[J]. The Computer Journal, 2019, 62(6): 943-960.

[5] LAMPORT L. Paxos made simple[J]. ACM Sigact News, 2001, 32(4): 18-25.

[6] LEE J, MUEHLE M. Distributed transaction management using two-phase commit optimization: U.S. Patent 8,442,962[P]. 2013-5-14.

[7] ATIF M. Analysis and verification of two-phase commit & three-phase commit protocols[C]//2009 International Conference on Emerging Technologies. New York: IEEE, 2009: 326-331.

[8] HERLIHY M. A quorum-consensus replication method for abstract data types[J]. ACM Transactions on Computer Systems (TOCS), 1986, 4(1): 32-53.

[9] BURROWS M. The Chubby lock service for loosely-coupled distributed systems[C]//Proceedings of the 7th Symposium on Operating systems design and implementation. USENIX Association, 2006: 335-350.

[10] BAKER J, BOND C, JAMES C, et al. Megastore: Providing scalable, highly available storage for interactive services [C] //Proceedings of CIDR′11, 2011: 9-12.

[11] CORBETT J C, DEAN J, EPSTEIN M, et al. Spanner: Google's globally distributed database[J]. ACM Transactions on Computer Systems (TOCS), 2013, 31(3): 8.

[12] ZHENG J, LIN Q, XU J, et al. PaxosStore: High-availability storage made practical in WeChat[J]. Proceedings of the VLDB Endowment, 2017, 10(12): 1730-1741.

[13] RAO J, SHEKITA E J, TATA S. Using paxos to build a scalable, consistent, and highly available datastore[J]. Proceedings of the VLDB Endowment, 2011, 4(4): 243-254.

[14] OKI B M, LISKOV B H. Viewstamped replication: A new primary copy method to support highly-available distributed systems[C]//Proceedings of the seventh annual ACM Symposium on Principles of distributed computing. New York: ACM, 1988: 8-17.

[15] OKI B M. Viewstamped replication for highly available distributed systems[R].Massachusetts Inst of Tech Cambridge Lab for Computer Science, 1988.

[16] LAMPORT L, MASSA M. Cheap paxos[C]//International Conference on Dependable Systems and Networks, 2004. New York: IEEE, 2004: 307-314.

[17] MAO Y, JUNQUEIRA F P, MARZULLO K. Mencius: Building efficient replicated state machines for WANs[C]//Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation, OSDI′08, Berkeley, 2008: 369-384.

[18] LAMPORT L B. Generalized paxos: U.S. Patent 7,698,465[P]. 2010-4-13.

[19] ONGARO D, OUSTERHOUT J. In search of an understandable consensus algorithm[C]//2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14). 2014: 305-319.

[20] MORARU I, ANDERSEN D G, KAMINSKY M. Egalitarian paxos[C]//ACM Symposium on Operating Systems Principles, 2012.

[21] LIN W, JIANG H, ZHAO N, et al. An optimized multi-Paxos protocol with centralized failover mechanism for cloud storage applications[C]//International Conference on Collaborative Computing: Networking, Applications and Worksharing. New York: Springer, 2018: 610-625.

[22] POKE M, HOEFLER T. Dare: High-performance state machine replication on rdma networks[C]//Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing. New York: ACM, 2015: 107-118.

(责任编辑: 张　晶)