**工程实践结题报告**

|  |  |
| --- | --- |
| **题 目** | **分布式系统一致性算法课题研究** |
| **学 院** | **软件学院** |
| **学生姓名** | 杨新泰 郜辉 鲁建惠 |
| **指导教师** | 孟宁 |
| **完成日期** | 2021 年 7 月 |

目 录

[1 引言 1](#_Toc128930097)

[1.1 选题背景及意义 1](#_Toc128930104)

[1.2 研究现状 1](#_Toc128930104)

[1.3 项目的主要工作 2](#_Toc128930104)

[1.4 本文内容章节安排 2](#_Toc128930104)

[2 关键技术介绍 3](#_Toc128930103)

[2.1 Gin Web 3](#_Toc128930104)

[2.2 Paxos算法 4](#_Toc128930105)

[2.3 LevelDB 4](#_Toc128930106)

[3 需求分析 6](#_Toc128930109)

[3.1 路由模块 6](#_Toc128930110)

[3.2 Paxos算法模块 6](#_Toc128930111)

[3.3 存储模块 6](#_Toc128930111)

[3.4 状态机模块 7](#_Toc128930112)

[4 具体实现 8](#_Toc128930115)

[4.1 RPC消息设计 8](#_Toc128930116)

[4.2 路由模块 9](#_Toc128930116)

[4.3 Paxos算法模块 10](#_Toc128930117)

[4.4 存储模块 12](#_Toc128930118)

[4.5 状态机模块 12](#_Toc128930118)

[4.6 系统总体架构 16](#_Toc128930116)

[5 测试 18](#_Toc128930119)

[5.1 性能测试 18](#_Toc128930120)

[5.2 正确性测试 19](#_Toc128930121)

6 [结论 24](#_Toc128930122)

# 1 引言

* 1. 课题背景及意义

分布式系统通过大规模的服务器集群协作提供大规模的计算服务，以其巨大的成本优势、计算弹性、整体高可靠性等优势，在工业应用领域已经全面替代了中央大型机的模式。

然而，在数据存储和强状态的服务的场景中，分布式系统中的计算节点故障切换过程中，会导致数据错误或者丢失的问题。当一个缓存的最新数据的进程故障时，倘若切换到另一个替代进程，虽然可以快速恢复服务， 但是故障的进程上所缓存的其它节点并不知道的数据会彻底丢失。更困难的是， 在分布式系统中，没有一个可靠的方法判断一个进程是故障了还是因为网络分区 导致该进程暂时不可通信。因此，在大多数情况下，都不得不根据业务场景进行有选择的牺牲。牺牲系统可用性或者牺牲数据一致性。

分布式一致性算法（Paxos，Raft 等）的出现提出了一个新的问题解决的方向，通过多个实例进行相同指令序列的计算，少数节点故障或者不可通信时，同时保障可用性和数据一致性。Google 的 MegaStore, Spanner, Chubby 等系统，均基于 Paxos 算法实现。国内近些年，也逐步在工业领域里进行应用。比如，腾讯的PaxosStore。已有的工业应用实践，都还有改良改善的空间。有的系统仍旧需要十几秒的时间才能恢复服务，有的系统无法支持并发写入。性能和成本也是一个巨大的挑战。

* 1. 研究现状

Leslie Lamport在1990年提出了Paxos算法，用来解决分布式系统中的共识性问题。但是在论文中，作者用希腊城邦来比喻分布式系统，所以大多数人很难理解Paxos算法。所以，直到2006年，Google在《Paxos Made Simple》中才讨论了他们是如何在工程中实现Paxos算法的。

Google在过去的十年中撰写了一些论文来介绍他们在分布式数据库上的研究和成果，其中包括MegaStore，Spanner和F1，这些数据都实现了基于Paxos实现了分布式系统中的共识，但由于商业原因，他们并未公布源代码，只在论文中简单地介绍了其原理和工作的过程。

但是，由于Raft算法的提出，以及越来越多人的研究，Paxos算法已经有了一些实现。

* 1. 项目的主要工作

本项目实现了Paxos算法，并且基于Paxos算法实现了数据的同步。同时，对外提供RESTful接口，接受网络的KV请求并将其请求转化为Log同步到各节点，通过解析Log并应用于状态机，实现了各节点数据的一致性。简单地说，本项目基于Paxos算法实现了分布式的KV数据库。

* 1. 本文内容章节安排

本文共分为六章：

第一章介绍了本文的研究背景，研究意义以及主要内容

第二章介绍了相关技术，包括：Gin Web，Paxos，LevelDB等。

第三章对实际情况中分布式系统可能出现的问题进行分析和整理，明确了模块的功能划分，并且分别对各个模块讨论分析了系统开发中可能产生的各种需求。

第四章先讨论了本项目的RPC消息设计，然后详细阐述了在项目中如何使用相关技术设计并实现相关的模块和功能，最后给出了整个系统的总体架构设计示意图。

第五章对完成后的项目进行了性能测试和正确性测试。

第六章是全文的工作的总结和展望

# 2 关键技术介绍

2.1 Gin Web

本项目使用Go语言实现，Go语言是由Google开发的静态语言，具有高性能，高并发等特点，并且支持垃圾回收。目前的 Go 语言在国内外的社区都非常热门，很多著名的开源框架，例如：Kubernetes、etcd 和 Prometheus 等都使用 Go 语言开发。Gin是一个用Go编写的 web 框架，可以直接在我们的项目中使用。

RESTful风格是Roy Thomas Fielding博士于2000年在他的博士论文中提出来的一种Web请求的命名风格，目的是便于不同程序在网络中互相传递信息。RESTful是基于HTTP之上而确定的一组约束和属性。符合这种风格的网络服务，允许客户端发出以统一资源标识符访问，然后根据请求的类型来确定应该执行什么样的操作。

我们使用Gin框架提供的Router来处理HTTP请求。在这里我们可以设置可以访问的URL和对应的处理函数，同时使用RESTful风格的URL。比如，我们将/store/set/的post请求关联到发起Proposal的函数，将/store/get/关联到读取数据的函数。这样，我们就可以通过HTTP协议来调用Paxos算法。

## 2.2 Paxos算法

Lamport在《Paxos Made Simple》中提出了Paxos算法的准确的描述。Paxos算法中有Proposer和Acceptor两个角色，其中Proposer负责提出Proposal，Acceptor负责审批Proposal。Paxos算法允许在一个系统中存在多个Proposer和Acceptor，同时也允许一个节点兼任两个角色。

Paxos算法包含两个阶段：一个是Prepare阶段，二是Acceptor阶段。

在Prepare阶段，每个Proposer都有可能提出Proposal，每个Proposal都会有一个id。Acceptor在收到Proposal之后，会将他的id和本地记录的之前通过的最大id进行比较。如果这个Proposal的id要大那么就会审批通过这个Proposal，并且用这个id更新本地的最大id。这里，我们可以认为Prepare阶段的作用是选出“最新”的提案，并且在Acceptor阶段尝试提交这个提案。

当提案的Proposer收到大多数Acceptor确认之后，Paxos算法就会进入Accept阶段。

在Accept阶段，Proposal会再次向Acceptor发送通过Prepare阶段的Proposal。Acceptor会根据本地记录的最大id来判断是否接受该提案，这是为了拒绝不是“最新”的提案。当Proposer在Accept阶段收到大多数Acceptor的通过确认之后，那么这个分布式系统中的共识就已经达成了。

Paxos算法通过Prepare阶段确认最新的Proposal，通过Accept阶段将这个最新的Proposal通知给其他的节点，成功地解决了分布式系统中的共识问题。

## 2.3 LevelDB

在2006年的OSDI上，Google发表了《Bigtable: A Distributed Storage System for Structured Data》来介绍其分布式数据库Bigtable。这篇论文的主要点有两个，一是使用排序大表实现分布式数据库，二是使用LSM-Tree作为存储引擎的结构。LevelDB正是Google开源的以LSM-Tree作为存储引擎结构的KV数据库，由于LSM-Tree的特性，能够拥有非常高的写入速度，同时由于使用了布隆过滤器以及数据有序的特性，读性能也不会成为性能的瓶颈。经过近二十年的发展，LSM-Tree类的存储引擎几乎成为分布式数据库的首选。

本项目使用LevelDB作为存储数据的数据库，由于LevelDB原生支持Key-Value对的读写，所以我们直接使用其提供的接口，在Prepare阶段和Accept阶段把数据以Index为Key，对应Log为Value的形式写入磁盘，实现数据的持久化。这样我们就能在宕机以后，通过重播磁盘中的Log恢复对应的数据。

# 3 需求分析

我们的项目是实现一个分布式的KV数据库，具体来说，有接受HTTP请求，基于Paxos算法实现共识，数据持久化，状态机同步等功能。这些功能又可根据主题划分为路由模块，Paxos算法模块，存储模块，状态机模块等四大模块。

## 3.1 路由模块

3.1.1 功能需求

路由模块负责接受HTTP请求和绑定请求到对应处理逻辑函数两个功能。

我们实现的是分布式KV数据库，其对应的需要处理的请求只有Get和Set两个操作。其中，Get的逻辑处理函数需要能够从数据库中得到Key对应的Value。 而Set的逻辑处理函数需要将HTTP请求携带的Key和Value发送给Paxos算法实例，开启一轮新的同步，从而实现数据的同步。

另外，由于测试的需要，我们需要提供打印Log的接口PrintLog和模拟节点宕机的接口Crash，以及空消息NOP。其中，PrintLog接口需要打印出节点中所有的Log情况。Crash接口则需要将收到请求的节点杀死来模拟宕机。

3.1.1 非功能需求

路由接口需要遵循RESTful风格，良好的代码和命名风格能够帮助用户快速上手使用，同时充分利用HTTP的不同请求类型，减少URL的数量。考虑到未来可能的变化，路由模块还需要易于拓展，不能有太高的代码耦合。

## 3.2 Paxos算法模块

3.2.1 功能需求

Paxos算法模块的功能需求包括两方面，安全性（Safety）和活跃性（Liveness）。

安全性又表现为两方面：第一，对于每一个日志，最多只有一个值会被选中；第二，如果节点认为一个值是被选中的那么这个值一定是被大多数节点选中的。

活跃性也表现为两方面：第一，在大多数节点正常工作的情况下，总会有一个值被选中；第二，如果一个值被选中了，那么每一个服务器最都能知道这个值。

这两个功能描述起来十分拗口，但是只要严格地按照在2.2节中描述的流程实现算法，我们就能实现上述的目标。

3.2.1 非功能需求

基本的Paxos算法每次确定一个值需要两个阶段，也就是两轮网络通信。如果希望加速Paxos算法那么我们就需要额外的优化。这里，我们采用批处理的方案，将多个HTTP请求封装成一个Paxos的值，这样可以将2\*n轮网络通信优化为2次，同时也将n此数据持久化造成的落盘优化成1次，能够极大地提升性能。

## 3.3 存储模块

3.3.1 功能需求

存储模块主要负责数据的持久化。我们对其的需求包括两个功能，一是写入Key-Value，二是根据Key读取Value。

我们首先要确定的是选择合适的数据作为Value，并选择合适的属性作为Key。根据Paxos算法的描述，我们并不需要将状态机中的具体的Kay-Value数据持久化，而是应该将Paxos算法的Proposal持久化。所以，在存储模块中，我们实际的Value是Paxos的Proposal，考虑到Paxos算法中的id是唯一的，所以可以将其作为Key。

我们在存储模块使用LevelDB作为我们的存储引擎。LevelDB原生支持Key和Value的读写，可以很好地满足我们的需求，我们只需要将对应的数据传入其提供的接口就可以了。

3.3.1 非功能需求

存储模块的肺功能需求主要是性能需求。传统的存储引擎如MySQL采用in-place更新，每次更新都需要一次查找，找到数据位置后在这个位置写入。LevelDB实现的是LSM-Tree结构，采用的是Out-of-Place更新，直接在内存中写入，然后写到磁盘，写数据前不需要查找，大大提高了写入的速度。此外，由于使用了多级结构和排序，读取数据的性能也相当不错。

## 3.4 状态机模块

3.4.1 功能需求

状态机模块主要负责解析Proposal并执行相应的动作。

解析Proposal需要分三种情况，Proposal的类型是Set和Proposal的类型是Delete，以及NOP。其中，Set类型需要继续解析后续的Key和Value，而Delete类型则只需要解析后续的Key。NOP类型的Proposal不需要解析内容。

解析了Proposal以后，要根据其类型执行相应的动作。如果是Set类型，需要将Proposal中的Value放到Key的位置，如果是Delete类型则需要将其Key对应的内容从状态机中删除。最后的NOP的类型则不需要进行处理。

3.4.1 非功能需求

目前，我们的工作希望实现的是KV数据库，所以状态机应当支持KV形式。但是，后续工作可能需要支持不同的模型，如SQL或者是图模型，所以在设计时需要考虑如何实现多模型之间的兼容。此外，由于每个节点对应的状态机都应该只有一个，以此避免线程不安全的情况，所有我们还需要以单例模式实现状态机。

# 4 具体实现

## 4.1 RPC消息设计

本项目中，各节点之间的通信通过RPC实现，所以RPC的消息结构体是我们项目重要的信息交换载体。在本项目中，我们的RPC主要用在Paxos算法的两阶段中，各个节点之间的通信，具体的设计如下：

**type PrepareRequest struct {**

**Index       int**

**ProposalNum int**

**}**

**type PrepareResponse struct {**

**AcceptedProposal int**

**AcceptedValue    string**

**}**

**type AcceptRequest struct {**

**Index         int**

**ProposalNum   int**

**ProposalValue string**

**}**

**type AcceptResponse struct {**

**MinProposal int**

**}**

以上RPC的内容需要严格按照Paxos算法的规定，否则可能丢失信息而造成算法的错误。

## 4.2 路由模块

根据3.1节的分析，我们可以知道，路由模块有两大部分组成，接受HTTP请求和每个URL对应的处理函数。其中，接受HTTP请求可以使用Gin框架提供的Router对象实现，具体来说使用Router.GET/POST/DELETE/…(URL,FUNC)的方式，就可以绑定FUNC来处理用特定请求方式访问的URL。

详细地说，我们分别用下列函数处理对应的URL：

（1）：/store/get/:key ->GetValue()：GetValue函数通过解析HTTP请求中的Key值，并使用Key值在状态机中查询来得到对应的结果。如果有对应的值，那么我们返回值，如果没有值，我们就返回一个错误。

（2）：/store/set/->SetValue()：在这个函数中，我们通过Gin框架的Context得到请求的Key和Value，然后封装成SetReq对象传入到Paxos的Batch中，等待统一处理，如果最后不发生错误就返回。

（3）：/log/print/\*path->PrintLog()：这个函数先调用LevelDB中的PrintLog接口，遍历所有记录，将持久化的所有Log写入到path指定的位置。完成后，将文件位置作为消息体返回给客户端。

（4）：/crash->GetCrash()：这里我们直接调用os.Exit()来杀死当前进程。

（5）：/store/nop->Nop()：这里我们生成一个无操作的请求，并送入到Paxos的Batch中，然后等待回复。

## 4.3 Paxos算法模块

在正式运行Paxos算法之前，会将请求收集在Batch中。Batch有两个限制参数，Limit和Duration。其中Limit用于限制Batch可以缓存的最多请求数，Duration用于限制一个Batch单次可以缓存的最长时间。到达任何一个限制条件，就会将Batch的内容作为值发起Paxos算法。

在Batch中，有两个Channel类型，一个是In Channel，负责接受请求；一个是Out Channel，负责发送请求，还有一个BatchRequest负责缓存请求。所有路由模块接受的请求都会经过In Channel到达Batch。Batch会将其追加到Reqs中，这是一个Request数组。每次追加请求，它都会检查Reqs的大小，如果到达Limit的限制，就会通过Out Channel发送出去。同样地，如果等待时间到达Duration，那么如果Batch中有请求，即使数量没有到达Limit的限制，也会发送出去。

当这个Batch中所有请求都完成以后，就会清空Reqs，继续接受新的请求。

Paxos算法分成两个角色Proposer和Acceptor，也分成两个两个阶段Prepare阶段和Accept阶段，所以我们需要实现两个角色类，角色类中分别实现函数去处理不同阶段的请求。

首先，我们讨论Proposer的实现。Proposer会接收Batch发送的请求包，当他从Channel中收到请求后，会调用函数StartNewInstance()。在这个函数里，我们会找到一个未被用过的Index作为Proposal的Log的Index发起Prepare阶段。在这个函数里，我们使用DoPrepare()来做Prepare阶段的工作。在这个函数中，我们要先生成一个Proposal的id，再把Log Index和Proposal Id封装在PrepareRequest中，使用SendPrepareRequestAndWaitForReply()发送给其他的节点。在收到所有的回复以后，我们检查其中审批通过的数量，如果小于集群中节点数目的一半，那么说明有更新的数据，这次请求作废。如果大于集群中节点数目的一半，那么就说明这个Proposal得到了认可。这时，我们就可以将HTTP请求中的Key和Value作为Proposal的Value，发起Accept阶段。Accept阶段和Prepare阶段大同小异，将之前收集的信息封装在AcceptRequest消息体中，发送给集群内的其他的节点并且等待回复。在收到大多数赞同的回复以后，这个Proposal已经被大多数节点承认，我们要将这个消息发送给其他节点。这时我们发送一个LearnRequest，不影响Paxos算法的运行。如果不通过，那么就需要重新进行Prepare阶段。

然后，我们再考虑Acceptor的实现。Acceptor并不处理HTTP请求，他主要负责审批Proposal。OnPrepare()函数负责处理Prepare阶段，在这个阶段，Acceptor需要检查Proposal发送的Proposal的id是否大于本地记录的id，如果小于本地记录的id，说明这个Proposal的数据不是最新的，不需要处理，如果大于本地记录的id，那么那么就把这个id记录在本地，并且审批通过。OnAccept()函数负责处理Accept阶段，在Accept阶段，也是需要和本地记录的id进行比较，如果小于本地的id，说明这个Proposal在Prepare阶段的通过审批过期了，在Accept阶段就需要拒绝掉。如果大于等于本地记录的id，那么需要更新本地记录的最大id和其对应Value。此外，由于Prepare阶段和Accept阶段只是达成了Log层面一的共识，所以还需要一个Learn阶段实际执行相关的Command。当Proposer在Accept阶段发现得到大多数同意以后，说明这个Proposal被确定了，他会发送一个Learn请求，提示Acceptor已经可以真正地执行这个Proposal相关的Command。OnLearn()函数负责处理这个阶段，Accept根据请求的Index找到相应的Log，将其解析成Command，然后调用状态机的Execute()执行命令。

以上就是Paxos算法的主要的流程。

## 4.4 存储模块

根据3.3节的分析，我们知道至少需要实现数据的读写。这里我们直接调用了LevelDB的接口。由于LevelDB只能存储字符类型的数据，我们还需要实现一些转换函数。另外，由于状态机并未持久化，所以我们需要支持从持久化的数据还原状态机状态。具体函数解析如下，

（1）：WriteLog()：我们分别将Key和Log转换成字符串类型，然后调用LevelDB的Put()接口将数据写入LevelDB。

（2）：ReadLog()：我们先将Index转换成字符类型，然后将其作为Key，调用LevelDB的Get()接口尝试得到对应的Value。如果有对应Value，就返回结果，如果没有，返回错误。

（3）：ParseLog()：由于我们从LevelDB从得到的数据是字符串类型的，所以需要将其解析为对应的Log。首先，我们使用分号来分割字符串，得到对应的属性。其中，第一个属性对应Log的MinProposal，第二个属性对应Log的AcceptedProposal，第三个属性对应Log的AcceptedValue，第四个属性对应Log的IsCommited属性。将这些属性封装到Log中，然后返回这个Log。

（4）：Restore()：这个函数用于恢复状态机的状态。我们还没有实现快照功能，所以我们直接从头扫描Log并执行其中的Log。我们先拿到LevelDB的迭代器，然后从头开始扫描，读取每个KV数据，并使用ParseLog()解析Value，然后将其封装为Command，调用状态机的Execute()函数执行这个Command。

（5）：PrintLog()：这是一个辅助函数，用于打印目前所有的Log。我们使用LevelDB的迭代器遍历所有KV对，并且读取KV数据，然后写入到指定的位置。

## 4.5 状态机模块

根据3.4节的分析，我们知道除了基础的解析命令和执行命令之外，还需要能够支持多种模型和单例。因此，我们使用面向对象的思想，创建一个状态机模型，其拥有Execute()和Retrieve()两个接口。如果需要实现任何的模型，那么只需要继承这个接口，然后实现对应的方法就可以了。然后在对应的实现方法中不提供构造函数，而是先生成一个唯一的状态机，只能通过GetKVStatMachineInstance()访问就能够实现单例模式。

我们在这个项目中实现的是KV模型的状态机，即KVSM。我们在这个模型中直接使用Map存储KV数据。在模型中，有如下接口，

（1）：Retrieve()：用于通过制定的Key来得到对应的Value。直接用Map[key]的方式得到Key。

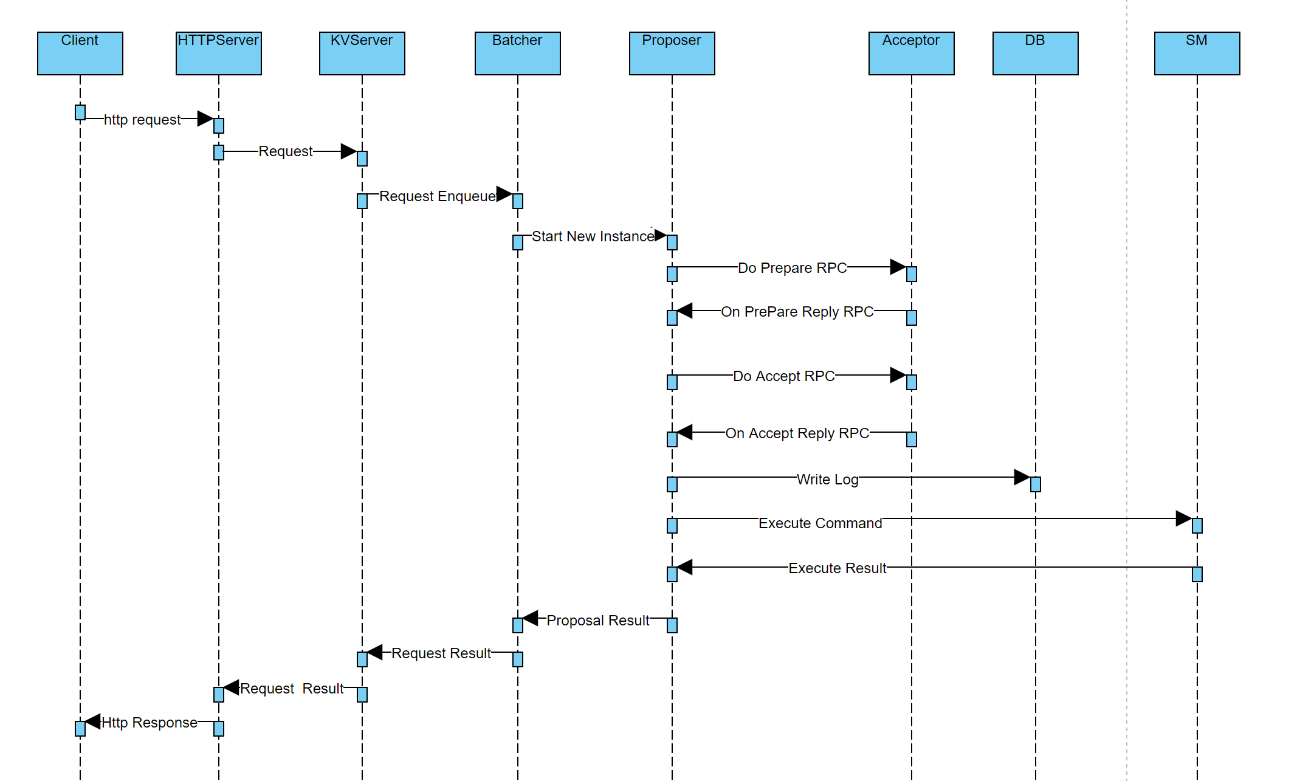
（2）：Execute()：这里，我们要执行的是一个Command，为了实现顺序执行，我们使用一个Map类型的cmds来存储CommandIndex和Command的映射。在执行这个函数时，我们先将传入的Command存储到cmds，然后在一个循环中执行所有未执行的Command，每执行一个Command就将其从cmds中删去。这样就能实现顺序执行Command。

（3）：DoExecute()：这个函数是具体执行命令的函数。我们通过分割Command来得到其内容，这里我们使用空格作为分隔符。Command的第一个属性是类型，我们根据其类型要做不同的动作。如果是SET类型，那么这个Command就有后续两个属性，分别是Key和Value，使用KVSM.map[Key]=Value的方式更新状态机。如果是Delete类型，那么Command有后续一个类型Key，我们要删除map中的对应KV数据。如果是NOP类型，那么我们不需要做任何事情。

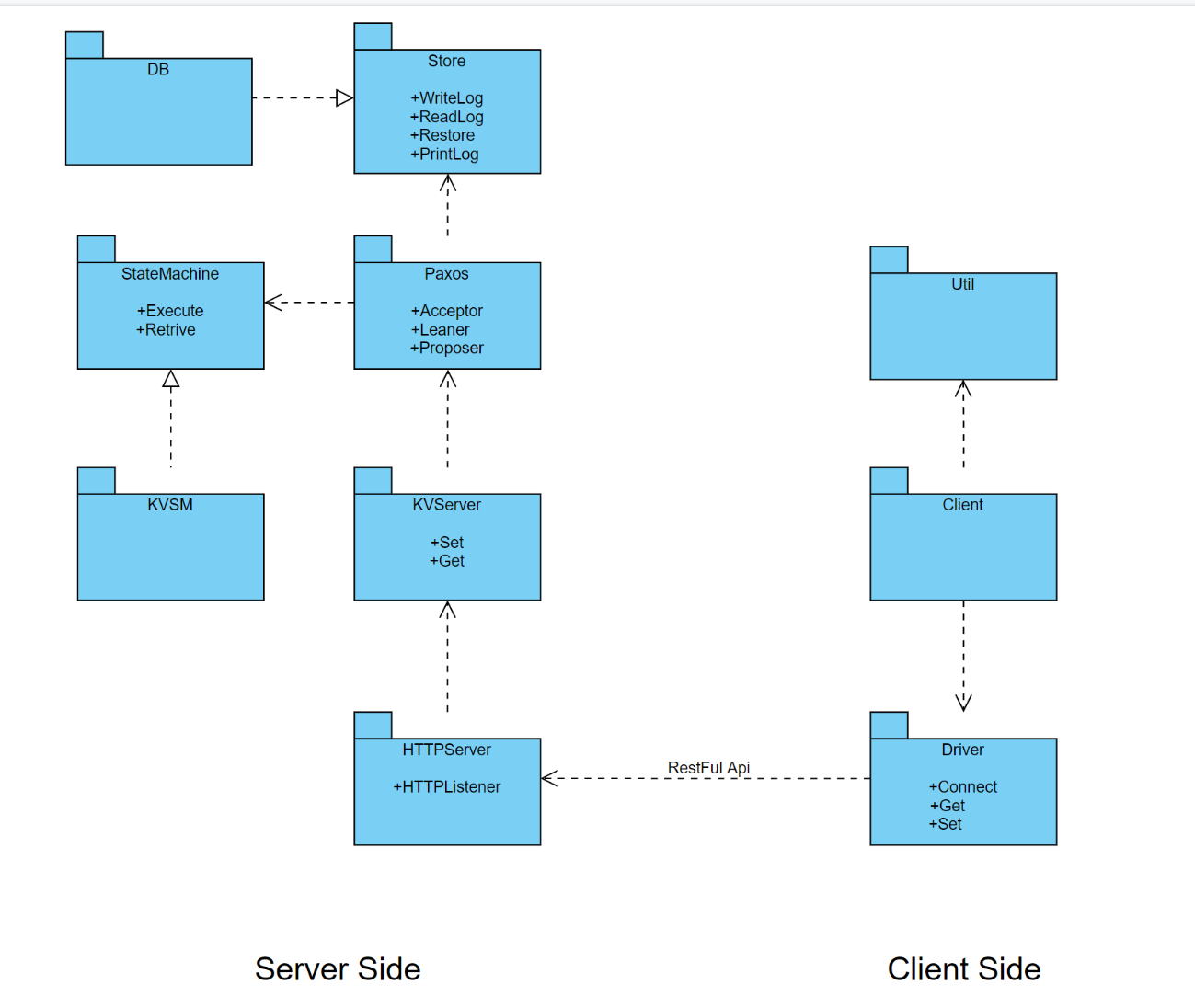
## 4.6 系统总体架构

通过上述的介绍，我们可以描述一次请求的整体流程。客户端发送请求到路由模块，路由模块根据请求URL和请求类型运行相应的处理函数，如果是写入请求，那么请求将会被缓存在Batch中。当Batch到达上限以后，会运行Paxos算法。经过两轮通信，最终确定的Log被写入LevelDB。最后，状态机会把成功写入的Log解析为命令并执行。

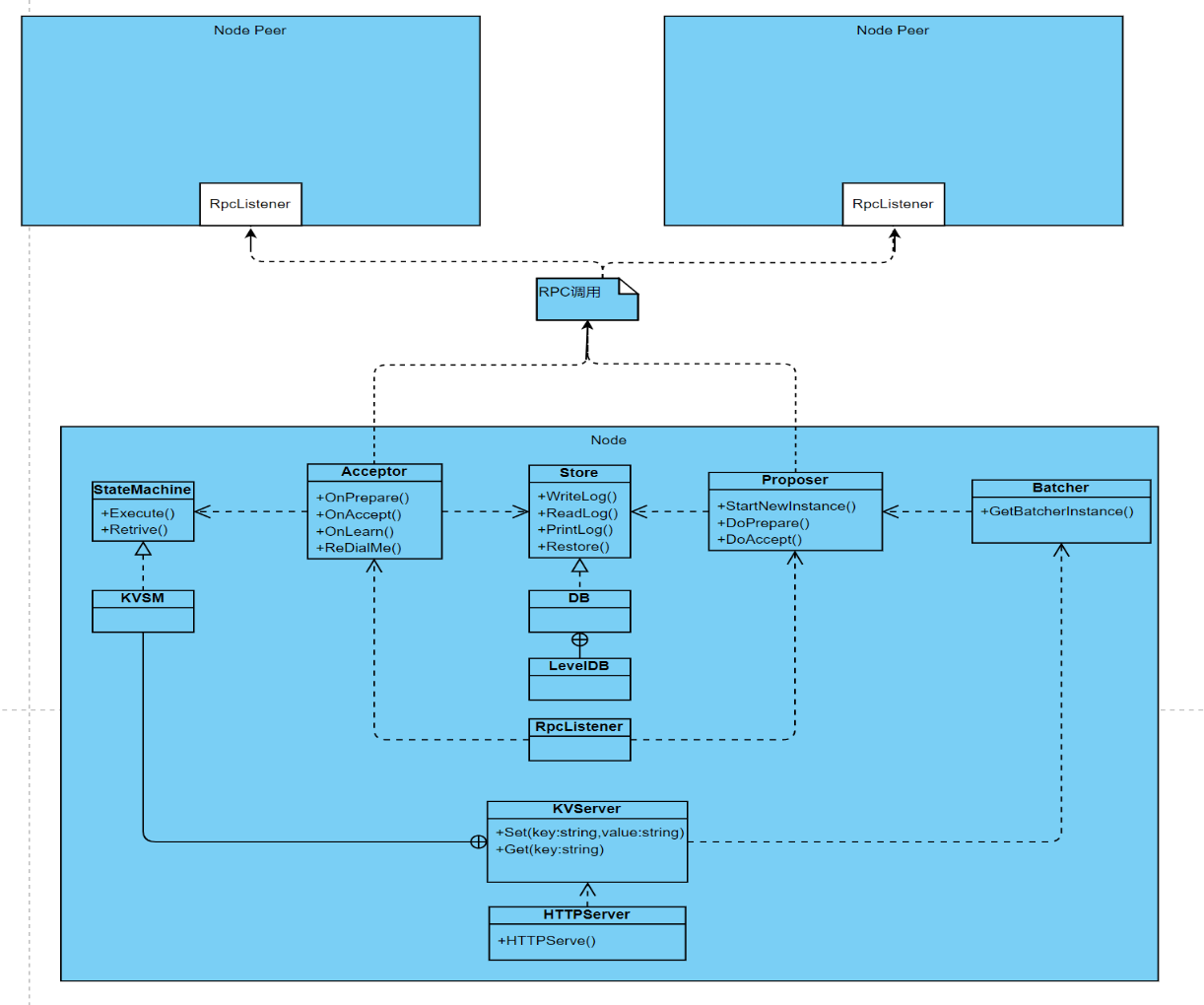
整体的流程图，具体如下：



项目的整体架构使用的是C\S架构，即客户端-服务端模型，其中涉及到的类和其关系如下图所示：



本项目是分布式的项目，涉及多个节点，每个节点都会维护自身的Paxos算法模块，存储模块和状态机模块，然后利用RPC和其他节点进行通信，其示意图如下：



# 5 测试

## 5.1 测试环境

操作系统 : CentOS 7, Linux VM-12-4-centos 3.10.0-1127.19.1.el7.x86\_64

处理器 ：腾讯云 Intel(R) Xeon(R) Gold 6133 CPU @ 2.50GHz，2核

内存 ：4 GB

磁盘 ：80 GB SSD

集群配置 ：部署在docker容器中，标准三节点

## 5.1 功能性测试

5.2.1 数据存取测试

测试方式：

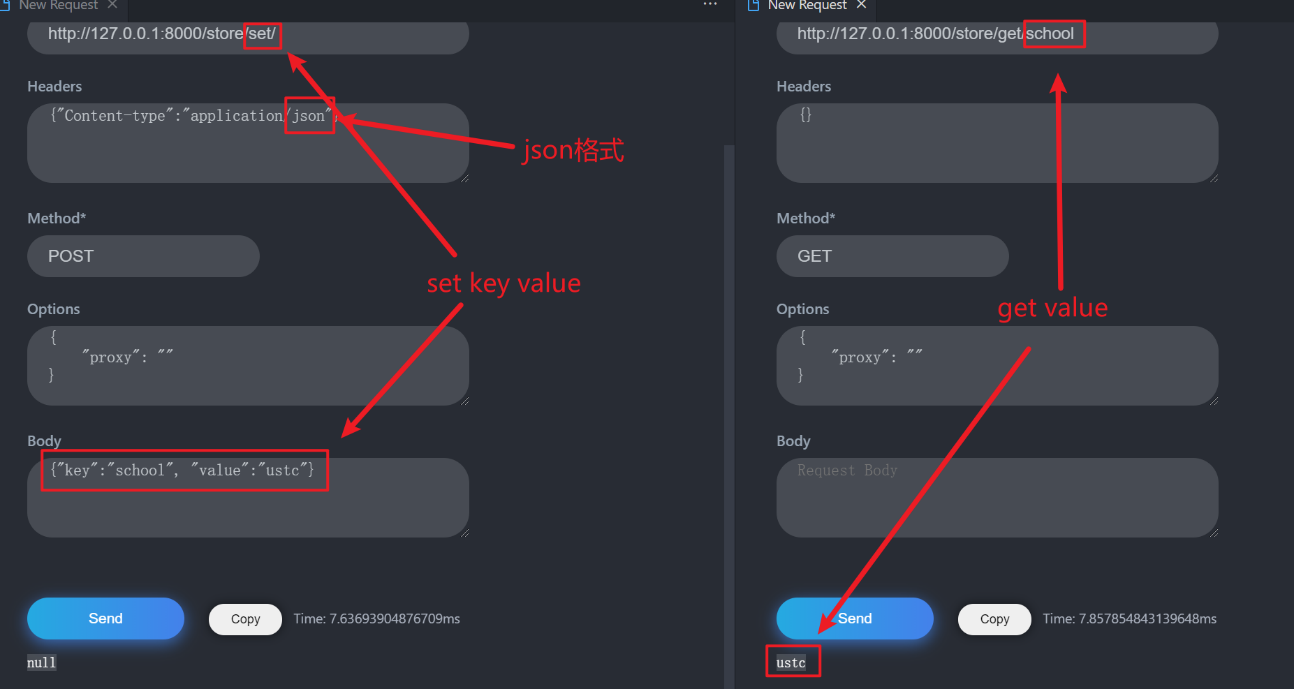
使用 curl 工具调用系统的 http rest api，测试数据的写入与读取

先使用 接口 http://node\_ip:port/store/set/ 接口，构造 post 请求，在请求体中使用 json 字符串携带 key 与 value 。

再通过 http://node\_ip:port/store/get 接口，构造 get 请求，使用 url 传递请求参数 key 。验证数据存取的正确性。

测试结果：

**测试结果验证了 rest api的正确性，在获取空value时返回空值，在set value后能立即获取更新后的值**



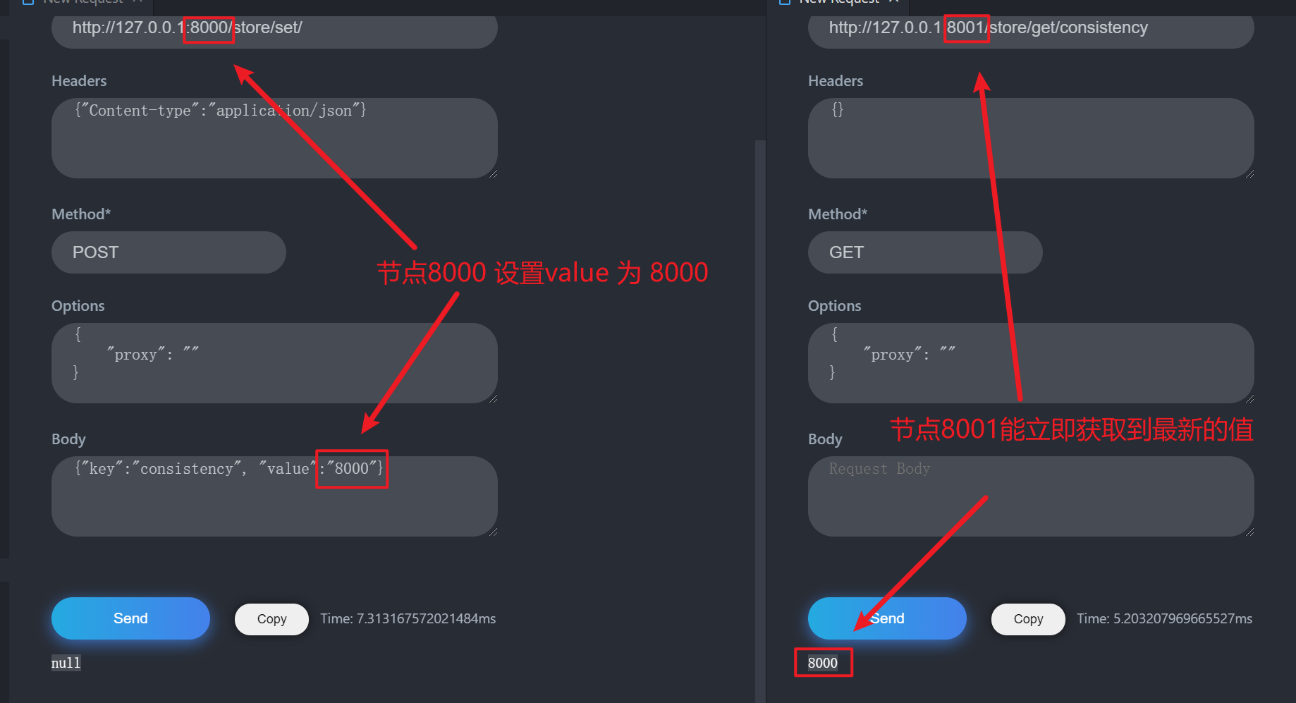
5.2.2 数据一致性测试

测试方式：

向集群中任意一个节点发送set key value请求，向集群所有节点发送get 请求获取刚刚设置的值，测试是否集群所有节点都能立即获取到最新的值

测试结果：

测试通过了检验，所有节点之间保持强一致性，对于外界来说彷佛只有一个副本，对节点任意节点的读写都是最新的。**任何一个读取返回新值后，所有后续读取（在相同或其他客户端上）也必须返回新值**



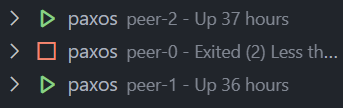
5.2.3 容错能力测试

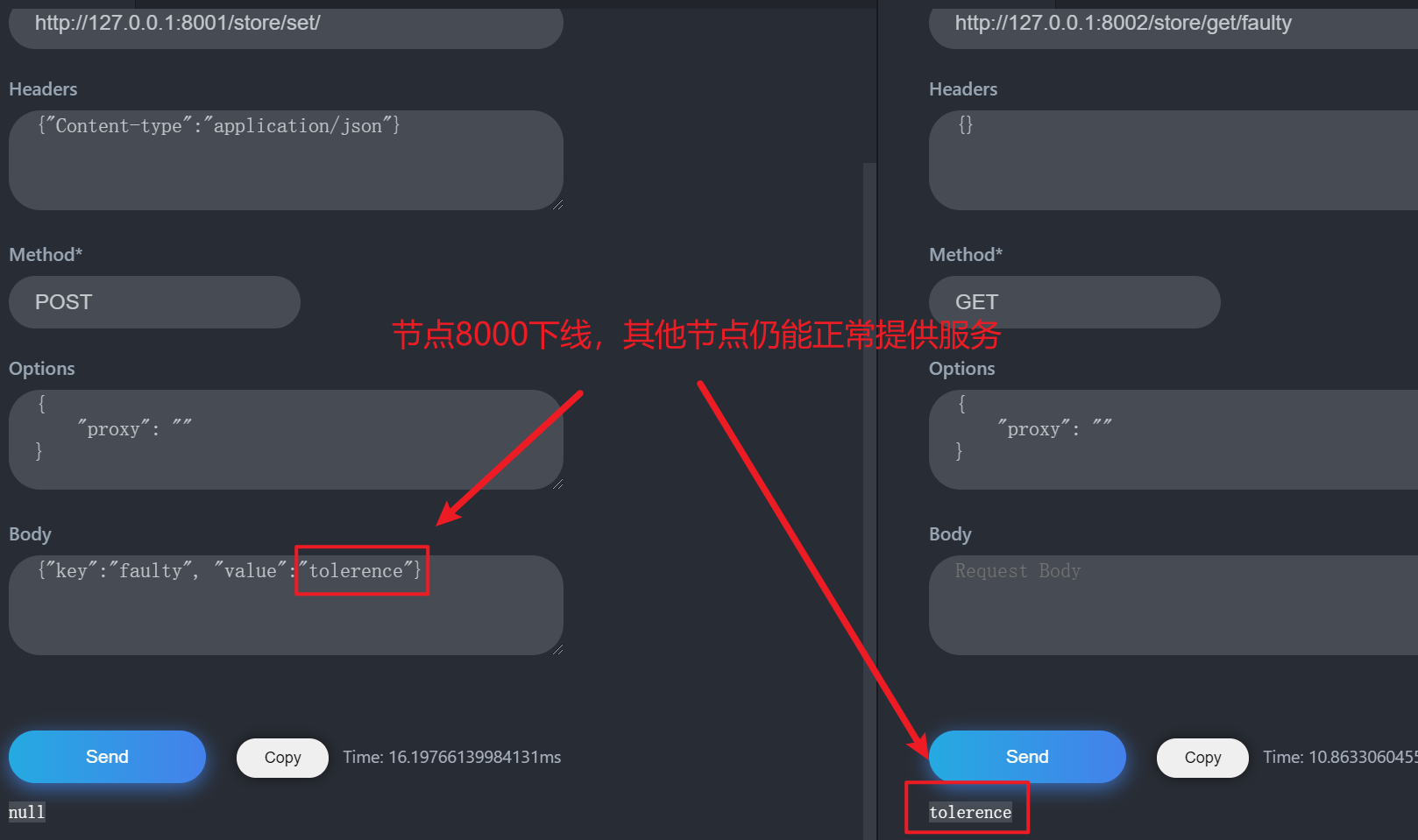
测试方式：

含n个节点的该集群理论上可以容忍最多同时（n-1)/2 个节点宕机，对于典型的3节点配置，可以容忍1个节点故障，整个集群不受影响，继续向外界提供正常服务。在测试中，我们人为下线1个节点，之后继续测试其他节点是否能正常提供服务。

测试结果：

下线节点 8000，保持节点 8001 与 8002 正常运行





测试结果验证了集群的容错性，对于3节点集群能够容忍任意1个节点的宕机，集群继续正常提供服务

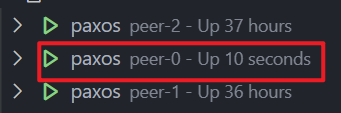
5.2.4 崩溃重启测试

测试方式：

我们下线1个节点，并将其重启。测试其是否能在重启后自动重新加入集群，继续充当节点的服务节点，并恢复到重启前的状态，与集群其他节点一起继续对外提供服务。

测试结果：

节点 8000 被重启





测试通过了验证，崩溃的机器重启后能自动恢复之前的状态，并自动加入集群，提供正常服务。

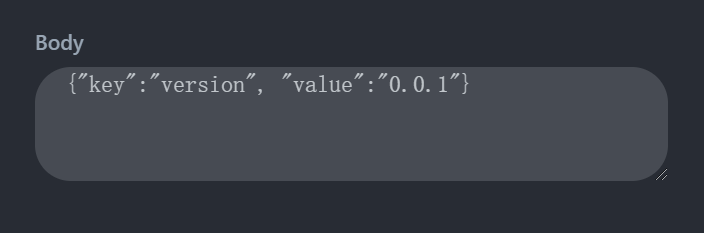
5.2.5 日志追赶能力测试

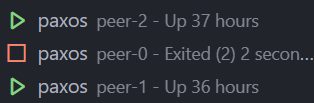
测试方式：

下线1个节点，其他节点继续提供服务，领先下线的节点一部分数据。之后重启下线节点，加入集群。测试下线恢复的节点的数据是否已经更新到最新。

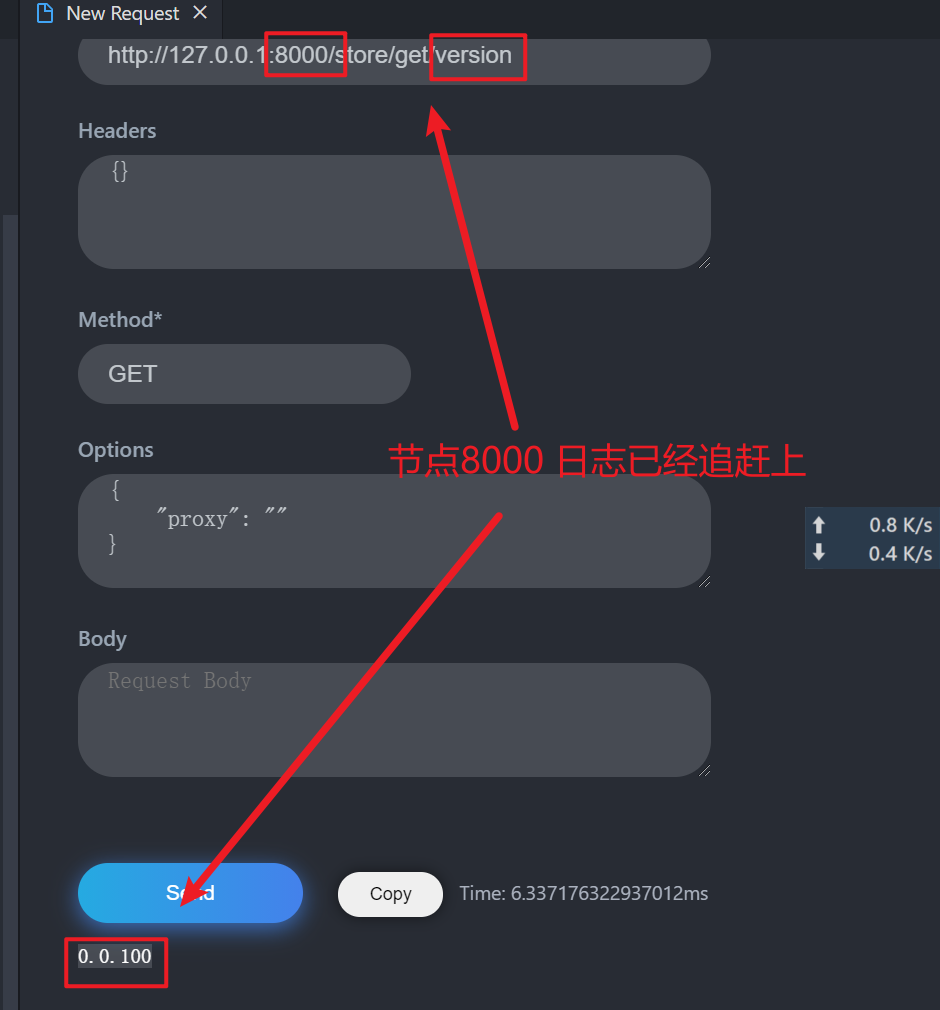
测试结果：

在 version=0.0.1时下线节点8000





并继续更新 version 100 次，直到 version 变为 0.0.100。此时上线节点 8000，并查询节点8000 version的值



测试验证了集群落后节点的日志追赶能力，整个集群即使在某个节点宕机重启后，也能迅速追赶日志，保持集群数据的强一致性。

## 5.2 性能性测试

测试方式：

使用go test -bench测试，分别测试 长字符串与短字符串、单个节点写入与集群写入、同步与异步写入 几种不同组合方式下，集群的性能数据。

测试结果：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试方法 | 测试次数 | 单次操作时间ns | 单次操作占用内存B | 单次操作内存分配次数 |
| Cluster  LargeValue | 219 | 11033069 | 15887 | 93 |
| Cluster  LargeValue  Async | 302 | 5657425 | 16350 | 94 |
| LargeValue | 302 | 4068764 | 15836 | 93 |
| LargeValue  Async | 304 | 3760725 | 16216 | 94 |
| Cluster  SmallValue | 358 | 2895628 | 15749 | 91 |
| Cluster  SmallValue  Async | 412 | 4764931 | 16252 | 93 |
| SmallValue | 417 | 3806715 | 15756 | 92 |
| SmallValue  Async | 510 | 2332716 | 16319 | 93 |

测试结论，集群最快写入可达每秒 500 次左右，在集群负载较大时，长字符串的性能会下降到短字符串的一半左右。在集群压力较小时异步访问方式能有效提升集群吞吐量，当集群负载升高时，这一吞吐量增幅逐渐降低。

# 6 结论

面对近来分布式系统的快速兴起，本项目实现了Paxos算法，并以此为基础实现了分布式的KV数据库，提供RESTful风格的HTTP接口，利用Paxos算法实现了数据的同步，基于LevelDB实现了数据的持久化。总得来说，本项目有以下亮点：

（1）本项目使用Gin框架实现了HTTP请求的接受和处理。将复杂的后端隐藏起来，对外提供了简单易用的网络接口。

（2）本项目使用Go语言实现了Paxos算法，并使用批处理技术进行优化，极大地提升了性能。我们实现的Paxos算法具有通用性，能用于各种系统中。

（3）本项目在Paxos算法的基础上实现了状态机，在Paxos算法同步Proposal的基础上，只要按序执行Proposal的内容，就能实现状态机的一致。我们实现了KV状态机，从而实现了分布式的KV数据库。