

dict the future, the proposer controls it by extracting a promise that there won't be any such acceptances. In other words, the proposer requests that the acceptors not accept any more proposals numbered less than n . This leads to the following algorithm for issuing proposals.

1. A proposer chooses a new proposal number n and sends a request to each member of some set of acceptors, asking it to respond with:

(a) A promise never again to accept a proposal numbered less than n , and

(b) The proposal with the highest number less than n that it has accepted, if any.

PhxPaxos

I will call such a request a *prepare* request with number n .

(Multi-Paxos)

2. If the proposer receives the requested responses from a majority of the acceptors, then it chooses a proposal number n and value v , where v is the value of the highest-numbered proposal among the responses, or is any value selected by the proposer if the responders reported no proposals.



赞同 86



分享

Paxos理论介绍(2): Multi-Paxos与Leader



LynnCui
软件·音乐

已关注

86 人赞同了该文章

发布于 2016-07-01 14:25

前文：[Paxos理论介绍\(1\): 朴素Paxos算法理论推导与证明](#)

理解朴素Paxos是阅读本文的前提。

Multi-Paxos

朴素Paxos算法通过多轮的Prepare/Accept过程来确定一个值，我们称这个过程为一个Instance。Multi-Paxos是通过Paxos算法来确定很多个值，而且这些值的顺序在各个节点完全一致。概括来讲就是确定一个全局顺序。

多个Instance怎么运作？首先我们先构建最简易的模式，各个Instance独立运作。

Instance	1	2	3	4	5	6
Paxos	Prepare Accept	Prepare Accept	Prepare Accept	Prepare Accept	Prepare Accept	Prepare Accept

每个Instance独立运作一个朴素Paxos算法，我们保证仅当Instance i 的值被确定后，方可进行 $i+1$ 的Paxos算法，这样我们就保证了Instance的有序性。

但这样效率是比较差的，众所周知朴素Paxos算法的Latency很高，Multi-Paxos算法希望找到多个Instance的Paxos算法之间的联系，从而尝试在某些情况去掉Prepare步骤。

下面我尝试描述一个Sample的演进情况来阐述这个算法，因为这个算法的要点其实非常简单，而且无需更多证明。

首先我们定义Multi-Paxos的参与要素：

- 3个参与节点 A/B/C.
- Prepare(b) NodeA节点发起Prepare携带的编号。
- Promise

赞同 86



74 条评论

分享

喜欢

收藏

申请转载



1(A)的意思是A节点产生的编号1，2(B)代表编号2由B节点产生。绿色表示Accept通过，红色表示拒绝。

下图描述了A/B/C三个节点并行提交的演进过程：

NodeA	Instance	1	2	3	4	5	6
	Prepare(b)	1(A)	1(A)	3(A)	2(A)	4(A)	2(A)
	Promise(b)	1(A)	2(B)	3(A)	2(A)	10(C)	2(A)
	Accept(b)	1(A)	1(A)	3(A)	2(A)	4(A)	2(A)

这种情况下NodeA节点几乎每个Instance都收到其他节点发来的Prepare，导致Promise编号过大，迫使自己不断提升编号来Prepare。这种情况并未能找到任何的优化突破口。

下图描述了只有A节点提交的演进过程：

NodeA	Instance	1	2	3	4	5	6
	Prepare(b)	1(A)	1(A)	1(A)	1(A)	1(A)	1(A)
	Promise(b)	1(A)	1(A)	1(A)	1(A)	1(A)	1(A)
	Accept(b)	1(A)	1(A)	1(A)	1(A)	1(A)	1(A)

这种情况我们会立刻发现，在没有其他节点提交的干扰下，每次Prepare的编号都是一样的。于是乎我们想，为何不把Promised(b)变成全局的？来看下图：

NodeA	Instance	1	2	3	4	5	6
	Prepare(b)	1(A)					
	Promised(b)	1(A)	1(A)	1(A)	1(A)	1(A)	1(A)
	Accept(b)	1(A)					

假设我们在Instance i进行Prepare(b)，我们要求对这个b进行Promise的生效范围是Instance[i, ∞)，那么在i之后我们就无需在做任何Prepare了。可想而知，假设上图Instance 1之后都没有任何除NodeA之外其他节点的提交，我们就可以预期接下来Node A的Accept都是可以通过的。那么这个去Prepare状态什么时候打破？我们来看有其他节点进行提交的情况：

NodeA	Instance	1	2	3	4	5	6
	Prepare(b)	1(A)					3(A)
	Promised(b)	1(A)	1(A)	1(A)	2(B)	2(B)	3(A)
	Accept(b)	1(A)	1(A)	1(A)	1(A)		3(A)

Instance 4出现了B的提交，使得Promised(b)变成了2(B)，从而导致Node A的Accept被拒绝。而NodeA如何继续提交？必须得提高自己的Prepare编号从而抢占Promised(b)。这里出现了很明显的去Prepare的窗口期Instance[1,3]，而这种期间很明显的标志就是只有一个节点在提交。

重点：不Prepare直接Accept为啥是安全的？因为Accept的b已经被Promise过。

总结

Multi-Paxos通过改变Promised(b)的生效范围至全局的Instance，从而使得一些单一节点的连续提交获得去

解有误。大家看到这里也应该明白这里的因果关系，Multi-Paxos是适应某种请求特征情况下的优化，而不是要求请求满足这种特征。所以Multi-Paxos接受并行提交。

Leader

为何还要说Leader，虽然Multi-Paxos允许并行提交，但这种情况下效率是要退化到朴素Paxos的，所以我们并不希望长时间处于这种情况，Leader的作用是希望大部分时间都只有一个节点在提交，这样才能最大发挥Mulit-Paxos的优化效果。

怎么得到一个Leader，真的非常之简单，Lamport的论文甚至的不屑一提。我们观察Multi-Paxos算法，首先能做Accept(b)必然是b已经被Promised了，而连续的Accept(b)被打断，必然是由于Promised(b)被提升了，也就是出现了其他节点的提交(提交会先Prepare从而提升b)。那么重点来了，如何避免其他节点进行提交，我们只需要做一件事即可完成。

收到来自其他节点的Accept，则进行一段时间的拒绝提交请求。

这个解读起来就是各个节点都想着不要去打破这种连续的Accept状态，而当有一个节点在连续的Accept，那么其他节点必然持续不断的拒绝请求。这个Leader就这样无形的被产生出来了，我们压根没有刻意去“选举”，它就是来自于Multi-Paxos算法。

题外话：为何网上出现很多非常复杂的选举Leader算法，有的甚至利用Paxos算法去选举Leader，我觉的他们很有可能是没有完全理解Multi-Paxos，走入了必须有Leader这个误区。

用Paxos算法来进行选举是有意义的，但不应该用在Leader上面。Paxos的应用除了写之外，还有很重要的一环就是读，很多时候我们希望要读到Latest，通常的做法就是选举出一个Master。Master含义是在任一时刻只能有一个节点认为自己是Master，在这种约束下，读写我都在Master上进行，就可以获得Latest的效果。Master与Leader有本质上的区别，要达到Master这种强一致的唯一性，必须得通过强一致性算法才能选举出来。而当我们实现了Paxos算法后，选举Master也就变得非常简单了，会涉及到一些租约的东西，后面再分享。

说的再多不如阅读源码，猛击进入我们的开源Paxos类库实现：[github.com/tencent-wech...](https://github.com/tencent-wechat/paxos)

发布于 2016-07-01 14:25

「真诚赞赏，手留余香」

赞赏

还没有人赞赏，快来当第一个赞赏的人吧！

微信 分布式系统 算法



欢迎参与讨论

74 条评论

默认 最新



sainty

3个参与节点 A/B/C。
Prepare(b) NodeA节点发起Prepare携带的编号。
Promise(b) NodeA节点承诺的编号。
Accept(b) NodeA节点发起Accept携带的编号。

各位看官，为什么NodeA要用Prepare(b) Promise(b) Accept(b) 来标记呢？分分钟跳戏啊有木有？

2020-08-15

回复 5

的节点请求过来的值覆盖吗？朴素Paxos貌似是不会被别的节点覆盖，会把该值V告诉给提交过来的proposer。Multi-Paxos Accept之后如何重新开始一轮新的paxos,这点一直想不通

2023-03-17

回复 喜欢

 你好凯蒂  ...

看不懂

2022-01-08

回复 喜欢

 sparkelf ...

leader断联怎么办呢？

2021-11-11



回复 喜欢

 哎呦喂 ...

所以master和leader的区别是啥 啊

2021-07-12


回复 喜欢

 天涯孤客  ...

Prepare(b)这个b是什么意思？6个instance提交的都是A，那promise阶段这个2(B)和10(C)是哪来的。麻烦能讲清楚点吗。

2021-04-22


回复 喜欢

 ATLAS ...

大佬请教一下 multi paxos中新的leader在重确认(recovery)之前的日志的时候 可以响应client的请求吗 还是说recovery阶段client需要等待？

2020-03-07


回复 喜欢

 Akriaaa ...

需要等待, 但应该比raft重新选个主的时间短

2020-12-19


回复 喜欢

 卜丁 ...

问个不屑一提的问题，选leader的时候，怎样拒绝其他节点的请求？在分布式理论层面，各节点的请求应该是无法区分的吧？虽然实际请求中可能带ip等信息。

2019-12-21



回复 喜欢

 Akriaaa ...

paxos instance都有全量的节点信息.

2020-12-19



回复 喜欢

 vikingfans  ...

有个疑问，paxos的作用是多个线程可以确定一个值，那么如果只能由leader提出议案的话，还要paxos干嘛。一个只能一个leader提出议案肯定不会有不一致的问题啊。

2018-05-24

回复 喜欢

 炼金术士  sainty ...

出现两个leader的时候就靠paxos的约束来保证一致了。

2020-11-16

回复 喜欢

 sainty ...

同问

2020-08-15



回复 喜欢

 西行寺·幽幽子  ...

我想问下，那个省略prepare的图中，也省略了accept的过程，，，，这个能省略吗？

2018-05-09

回复 喜欢

 Sylarxx  西行寺·幽幽子 ...

hhhhhhhhh，我认为原文的意思是：instance还是得一个接一个的处理的，如果instance并行，还得用滑动窗口（本文不讨论）。因为instance还是得一个一个处理，所以他那个只画了instance1处理完的（Accept成功）。之后的instance已经promise过了，只需要等待Accept就好了吧。

2018-12-05

回复 喜欢

知乎

首发于
分布式一致性与高可用实践

2018-12-05

点击查看全部评论 >

回复 喜欢



欢迎参与讨论



文章被以下专栏收录



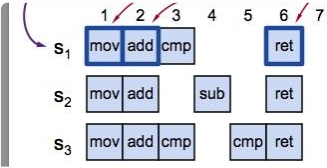
分布式一致性与高可用实践
微信后台在分布式一致性与高可用上的实践经验介绍

推荐阅读



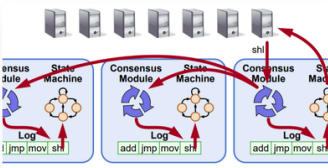
Paxos的应用场景

阿Q



比较raft，basic paxos以及multi-paxos

uncle creepy



共识算法 之 Multi-Paxos (一)

暗淡了乌云

Paxos前传-The Part-Ti Parliament (一)

前言之前研究了诸如Raft、Z算法，总觉得意犹未尽，因为其然而不知其何以然是一件非常不舒服的事情。本文是对出分布式一致性问题（虽然论文中描述的好像是一个考记
丁凯