

defined to be the largest vote v in $Votes(\mathcal{B})$ cast by p with $v_{bal} < b$, or to be $null_p$ if there was no such vote. Since $null_p$ is smaller than any real vote cast by p , this means that $MaxVote(b, p, \mathcal{B})$ is the largest vote in the set

$$\{v \in Votes(\mathcal{B}) : (v_{pst} = p) \wedge (v_{bal} < b)\} \cup \{null_p\}$$

For any nonempty set Q of priests, $MaxVote(b, Q, \mathcal{B})$ was defined to equal the maximum of all votes $MaxVote(b, p, \mathcal{B})$ with p in Q .
Conditions $B1(\mathcal{B})$ – $B3(\mathcal{B})$ are stated formally as follows.⁸

$$\begin{aligned} B1(\mathcal{B}) &\triangleq \forall B, B' \in \mathcal{B} : (B \neq B') \Rightarrow (B_{bal} \neq B'_{bal}) \\ B2(\mathcal{B}) &\triangleq \forall B, B' \in \mathcal{B} : B_{qrm} \cap B'_{qrm} \neq \emptyset \\ B3(\mathcal{B}) &\triangleq \forall B \in \mathcal{B} : (MaxVote(B_{bal}, B_{qrm}, \mathcal{B})_{bal} \neq -\infty) \Rightarrow \\ &\quad (B_{dec} = MaxVote(B_{bal}, B_{qrm}, \mathcal{B})_{dec}) \end{aligned}$$

Although the definition of $MaxVote$ depends upon the ordering of votes, $B1(\mathcal{B})$ implies that $MaxVote(b, Q, \mathcal{B})_{dec}$ is independent of how votes with equal ballot numbers were ordered.
To show that these conditions imply consistency, the Paxos first showed that $B1(\mathcal{B})$ – $B3(\mathcal{B})$ imply that, if a ballot number b is larger than any later ballot in \mathcal{B} is for the same decree as B .
Lemma If $B1(\mathcal{B})$, $B2(\mathcal{B})$, and $B3(\mathcal{B})$ hold, then
 $((B_{qrm} \subseteq B_{qrm}') \wedge (B'_{bal} > B_{bal})) \Rightarrow (B'_{dec} = B_{dec})$
for any B, B' in \mathcal{B} .

Paxos理论介绍(1): 朴素Paxos算法理论推导与证明

LynnCui
软件·音乐

已关注

255 人赞同了该文章

发布于 2016-06-28 12:38，编辑于 2016-06-28 13:28

微信重磅开源生产级paxos类库PhxPaxos，本文向大家阐述PhxPaxos的基石“朴素Paxos算法”。

开源地址：

[github.com/tencent-wech...](https://github.com/tencent-wechat/phxpaxos)

这篇文章摘取部分我在微信内部关于Paxos的分享PPT，通过注解的方式尝试与大家说明白朴素Paxos的理论证明。

为何要重点说朴素的Paxos？个人认为这个才是Paxos的精髓所在，也是所有Paxos相关算法的基石所在。另外本文将着重讲解Paxos的算法推导过程，而不是运行过程。因为以我学习算法的经验来看，推导过程对于掌握一门算法至关重要，只有掌握了理论推导过程，才能明白这个算法每一个步骤的含义。

这些PPT内容大部分都引自Lamport的论文 "The Part-Time Parliament" 。

defined to be the largest vote v in $Votes(\mathcal{B})$ cast by p with $v_{bal} < b$, or to be $null_p$ if there was no such vote. Since $null_p$ is smaller than any real vote cast by p , this means that $MaxVote(b, p, \mathcal{B})$ is the largest vote in the set

$$\{v \in Votes(\mathcal{B}) : (v_{pst} = p) \wedge (v_{bal} < b)\} \cup \{null_p\}$$

For any nonempty set Q of priests, $MaxVote(b, Q, \mathcal{B})$ was defined to equal the maximum of all votes $MaxVote(b, p, \mathcal{B})$ with p in Q .
Conditions $B1(\mathcal{B})$ – $B3(\mathcal{B})$ are stated formally as follows.⁸

$$B1(\mathcal{B}) \triangleq \forall B, B' \in \mathcal{B} : (B \neq B') \Rightarrow (B_{bal} \neq B'_{bal})$$
$$B2(\mathcal{B}) \triangleq \forall B, B' \in \mathcal{B} : B_{qrm} \cap B'_{qrm} \neq \emptyset$$
$$B3(\mathcal{B}) \triangleq \forall B \in \mathcal{B} : (MaxVote(B_{bal}, B_{qrm}, \mathcal{B})_{bal} \neq -\infty) \Rightarrow$$
$$(B_{dec} = MaxVote(B_{bal}, B_{qrm}, \mathcal{B})_{dec})$$

Although the definition of $MaxVote$ depends upon the ordering of votes, $B1(\mathcal{B})$ implies that $MaxVote(b, Q, \mathcal{B})_{dec}$ is independent of how votes with equal ballot numbers were ordered.

To show that these conditions imply consistency, the Paxos first showed that $B1(\mathcal{B})$ – $B3(\mathcal{B})$ imply that, if a ballot number b appears in any later ballot in \mathcal{B} is for the same decree as B .

Lemma If $B1(\mathcal{B})$, $B2(\mathcal{B})$, and $B3(\mathcal{B})$ hold, then

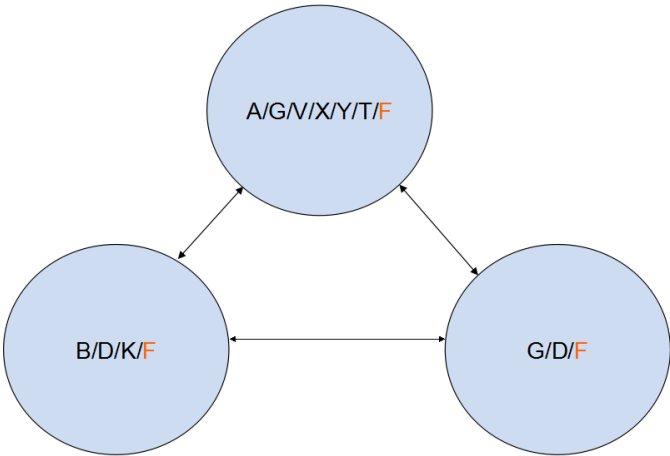
$$((B_{qrm} \subseteq B'_{qrm}) \wedge (B'_{bal} > B_{bal})) \Rightarrow (B'_{dec} = B_{dec})$$

for any B, B' in \mathcal{B} .

页1注解

这是PPT的题图，摆在中间的正是Paxos最为重要的三条约束，掌握这三条约束，即可掌握朴素Paxos。

确定一个值





页2注解

在正式开始讲解之前，希望大家抛开所有对Paxos的展开联想，而回到最朴素的Paxos。最朴素的Paxos解决什么问题？这里举个例子：三个人分别只允许呆在不同的三个城市，他们手上有一张纸和一支笔，他们可以在纸上写下任何内容，但是，当他们停下他们的笔之后，我们希望三个人最后写下的内容都是一样的。

这个就是最朴素的Paxos尝试解决的问题，确定一个值。暂时千万别去想更多的东西，聚焦在确定一个值这么一个看似非常简单的事情身上。

数学定义(*B*一轮投票的集合)

B_{dec} A decree (the one being voted on). 
 B_{qrm} A nonempty set of priests (the ballot's quorum). (A,B,C) or (A,D,E) or (A,B,C,D,E) ...
 B_{vot} A set of priests (the ones who cast votes for the decree).⁷ (A,C,D,E)
 B_{bal} A ballot number. 2
A ballot B was said to be *successful* iff $B_{qrm} \subseteq B_{vot}$, so a successful ballot was one in which every quorum member voted.

编号(bal)/ 参与者(vot)	提议(dec)	A	B	C	D	E
2		Y		Y	Y	Y

B: 一轮投票参与元素的集合。
Bdec: 这轮投票的目标提议。（可以为任何可描述的事务，比如一串二进制，一个图像等）
Bqrm: 一轮成功的投票所需要的投票者集合。（可理解为多数派）
Bvot: 这轮投票实际投票者集合。（实际真正投票成功的投票者）
Bbal: 这轮投票所使用的编号。



页3注解

直入主题，提出一轮投票的定义。通过投票来决定一个提议，是一个非常原始的方法，也是非常显然的公理，这里不展开说。这里提议对应刚刚说到的这个值。这一页每个定义都要弄明白，因为下面会常常用到这些定义。比如你要记住，一轮投票会有一个编号标识他们，称之为Bbal。你还要理解集合的意思，一轮投票集合B概括了这一轮投票的所有参与人，投票编号，提议，以及投票情况等。

比较难理解的Bqrm这里展开解释一下：一轮投票获得通过，必须有Bqrm的人进行了投票，这个Bqrm每次可能都是不同的集合，

但是它的特征是肯定超过总体投票成员的半数，也就是我们常说的多数派。

数学定义(*β*多轮投票的集合)

编号(bal)/ 参与者(vot)	提议(dec)	A	B	C	D	E
2		Y		Y	Y	Y
5			Y	Y		Y

面临问题：
(C,E)投了两个成功的提议，到底哪个才是最终的提议？
(A,D)和B看到的成功提议不一样。

页4注解

很显然，一轮投票是解决不了一致性问题的，因为任意一个人都有可能去发起投票，而不能靠上帝去指定只有某个人去发起，所以必然会面临多轮投票带来的问题。这里提出多轮投票的定义。注意这个多轮投票集合的定义是希腊字母Beta。一轮投票集合是大写字母B。是不一样的。我们希望

数学定义(*MaxVote*比我小的最大编号的投票)

编号 (bal)/ 参与者 (vote)	提议 (dec)	A	B	C	D	E	Maxvote
2		Y		Y	Y	Y	<i>B</i> bal = 2, <i>B</i> qrm = (A,C,D) or ..., <i>MaxVote</i> (2, (A,C,D), <i>β</i>)dec = null
5			Y	Y		Y	<i>MaxVote</i> (5, (C,D,E), <i>β</i>)dec = 
3		Y	Y				<i>MaxVote</i> (3, (A,B,C), <i>β</i>)dec = 
9		Y	Y	Y	Y	Y	<i>MaxVote</i> (9, (A,B,C), <i>β</i>)dec = 

MaxVote(*B*bal, *B*qrm, *β*):
*B*qrm的投票者集合的所有轮投票里面，比*B*bal编号小的投票里面最大编号的投票

页5注解

最重要的定义MaxVote的提出。要尝试解决多轮投票带来的冲突问题，必然要去建立多轮投票之间的联系，MaxVote是一个联系。

MaxVote通过给出一个编号，以及成员，可以在多轮投票里面找到这些成员小于这个编号的所有投票当中，最大编号的那个投票。然后我们希望用到这次投票对应的提议。仔细阅读样例表格里面的每个MaxVote，从而去理解这个定义。

投票约束

$$B1(\mathcal{B}) \triangleq \forall B, B' \in \mathcal{B} : (B \neq B') \Rightarrow (B_{bal} \neq B'_{bal})$$
$$B2(\mathcal{B}) \triangleq \forall B, B' \in \mathcal{B} : B_{qrm} \cap B'_{qrm} \neq \emptyset$$
$$B3(\mathcal{B}) \triangleq \forall B \in \mathcal{B} : (MaxVote(B_{bal}, B_{qrm}, \mathcal{B})_{bal} \neq -\infty) \Rightarrow$$
$$(B_{dec} = MaxVote(B_{bal}, B_{qrm}, \mathcal{B})_{dec})$$

- B1*(*β*): 每轮投票的编号必须是唯一的。
- B2*(*β*): 只有多数派进行了投票，本轮投票才算成功。
- B3*(*β*): 对于一轮投票编号为*B*bal的投票，假如多数派中存在任意一名成员曾经投过比*B*bal编号小的票*B'*，那么*B*dec = *B'*dec.

页6注解

在提出了所有数学定义后，就可以去理解这最优美的三个约束条件了。正是通过这三个约束，使得多轮投票的冲突问题得到解决。

第一点很好理解，要求每轮投票的编号唯一。第二点要求任意两轮投票的Bqrm交集不为空，其实意思很明确，就是要求Bqrm超过半数的意思。第三点是解决冲突的关键所在，它强行约束了每轮投票的提议，使得这轮投票的提议不与之前的产生冲突。通俗一点讲就是，一旦我发现在我之前已经有人投过某个提议的票，那我就要用这个提议。并且是我之前最大编号的投票对应的提议，

投票约束下的多轮投票

编号 (bal)/ 参与者(vot)	提议 (dec)	A	B	C	D	E	Maxvote
2						Y	<i>MaxVote</i> (2, (A,C,E), β) <i>dec</i> = null
3				Y			<i>MaxVote</i> (3, (A,C,D), β) <i>dec</i> = null
5			Y				<i>MaxVote</i> (5, (A,B,D), β) <i>dec</i> = null
9		Y		Y		Y	<i>MaxVote</i> (9, (A,C,E), β) <i>dec</i> =
10		Y					<i>MaxVote</i> (10, (A,B,C), β) <i>dec</i> =
12			Y			Y	<i>MaxVote</i> (12, (A,B,E), β) <i>dec</i> =

页7注解

这是在三个约束条件之下的多轮投票过程。反复阅读这两页，从而理解约束条件3。注意在约束条件下，提议内容的变化。

如何证明约束投票可以引出一致性

证明要求：

当出现一轮投票**B**获得多数派通过，那么不可能再出现一轮投票**B'**,使得**B'bal > Bbal**，而**B'dec ≠ Bdec**。

反证法：

假设真的出现了一轮投票**B'**,使得**B'bal > Bbal**，而**B'dec ≠ Bdec**，那么通过**投票约束**使得这个假设引出**矛盾**，即可反证成功。

页8注解

看似这个投票过程可以引出最终一致的提议内容，但严格的算法推导必然需要严格的证明。这里提出反证法。

证明过程(反证)

编号 (bal)/ 参与者 (vot)	提议 (dec)	A	B	C	Maxvote
B 2		Y		Y	$MaxVote(2, (A, C), \beta)_{dec} = null$
C 3			Y	Y	$MaxVote(3, (B, C), \beta)_{dec} = ?$

1. 挑选C为编号大于Bbal的投票中最小编号的一轮。

2. $Cdec \neq Bdec$.

3. 由于B投票获得通过，所以Cqrm必然包含Bvot至少一个成员。(由B2(β)得出)

4. $MaxVote(Cbal, Cqrm, \beta)_{bal} \geq Bbal$. (由2,3得出)

5. $MaxVote(Cbal, Cqrm, \beta)_{dec} = Cdec$. (由B3(β)得出)

6. $MaxVote(Cbal, Cqrm, \beta)_{dec} \neq Bdec$. (由2,5得出)

7. $MaxVote(Cbal, Cqrm, \beta)_{bal} > Bbal$. (由4,6, B1(β)得出)

8. $MaxVote(Cbal, Cqrm, \beta)_{bal} < Cbal$. (由B3(β)得出)

9. 由1,7,8得出矛盾，证毕。

Proof of Lemma
For any ballot B in \mathcal{B} , let $\Psi(B, \mathcal{B})$ be the set of ballots in \mathcal{B} later than B for a decree different from B 's:
$$\Psi(B, \mathcal{B}) = \{B' \in \mathcal{B} : (B'_{bal} > B_{bal}) \wedge (B'_{dec} \neq B_{dec})\}$$

If $\Psi(B, \mathcal{B})$ is not empty, then B is not a majority. If $\Psi(B, \mathcal{B})$ is empty, then B is a majority. The Paxos algorithm gives a proof by contradiction. They assumed the existence of a B with $B_{qrm} \subseteq B_{vot}$ and $\Psi(B, \mathcal{B}) \neq \emptyset$, and obtained a contradiction as follows.²
1. Choose $C \in \Psi(B, \mathcal{B})$ such that $C_{bal} = \min\{B'_{bal} : B' \in \Psi(B, \mathcal{B})\}$.
PROOF: By 1 and the definition of $\Psi(B, \mathcal{B})$.
2. $B_{dec} \neq C_{dec}$.
PROOF: By B2(B) and the hypothesis that $B_{qrm} \subseteq B_{vot}$.
3. $MaxVote(C_{bal}, C_{qrm}, \mathcal{B})_{bal} \geq B_{bal}$.
PROOF: By 2, 3 and the definition of $MaxVote(C_{bal}, C_{qrm}, \mathcal{B})$.
4. $MaxVote(C_{bal}, C_{qrm}, \mathcal{B})_{dec} = C_{dec}$.
PROOF: By 5 and B3(B).
5. $MaxVote(C_{bal}, C_{qrm}, \mathcal{B})_{dec} \neq B_{dec}$.
PROOF: By 6, 1, and the definition of $\Psi(B, \mathcal{B})$.
6. $MaxVote(C_{bal}, C_{qrm}, \mathcal{B})_{bal} > B_{bal}$.
PROOF: By 4, since 7 and B1(B) imply that $MaxVote(C_{bal}, C_{qrm}, \mathcal{B})_{bal} \neq B_{bal}$.
7. $MaxVote(C_{bal}, C_{qrm}, \mathcal{B})_{bal} < C_{bal}$.
PROOF: By 7, 8, and the definition of $\Psi(B, \mathcal{B})$.
8. Contradiction.
PROOF: By 9, 10, and 1.
End Proof of Lemma

页9注解

这一页已将证明过程进行简化，相信经过你们认真的推敲，肯定是可以搞明白的。注意表格里面的样例，当编号为2的这轮投票已经通过后，又出现了一轮编号为3的投票（2和3中间不可能存在一轮投票），提议跟之前的冲突。我们通过推导得出这个情况是不存在的。

如何获得MaxVote

参与者	A	B	C	D	E
bal	2,3	3,8	5	5,8	5,8
dec					

1. 提议者向多数派的投票者Bqrm发送这轮投票的Bbal，并要求投票者返回编号小于Bbal的最大编号投票B'。

2. 当收到多数派的回应之后，选择回应里面所有投票编号最大的投票C，则Cdec = MaxVote(Bbal, Bqrm, β)dec。

4. MaxVote成功获得。

$MaxVote(2, (A, B, C), \beta)_{dec} = null$ $MaxVote(3, (A, B, E), \beta)_{dec} = \text{star}$

$MaxVote(5, (C, D, E), \beta)_{dec} = null$ $MaxVote(8, (B, D, E), \beta)_{dec} = \text{X}$

页10注解

前面只是提出MaxVote的定义，这里解释计算这个MaxVote的实际操作过程。其实就是我们惯用的轮询法，逐个问呗。只要每次发起投票前，都想多数派的成员逐一询问它们比我当前这轮投票编号小的最大编号投票，即可获得整个集合的MaxVote，从而确定当前这轮投票的提议。

可能获得错的Maxvote

编号 (bal)/ 参与者 (vote)	提议 (dec)	A	B	C	D	E	Maxvote
2		Y				Y	MaxVote(2, (A,C,D), β)dec = null
5			Y	Y	Y		MaxVote(5, (B,C,D), β)dec = null
3		Y	Y	Y			MaxVote(3, (A,B,C), β)dec =

由于编号5的投票在编号3的投票之前，导致刚才的方法无法获得正确的MaxVote(5, (B,C,D), β)dec，从而导致无法满足投票约束，算法失败！

页11注解

聪明的读者可能早已经发现问题了，我们上文所说的多轮投票，似乎编号都是严格递增的，但是现实情况完全不是这样，现实的多轮投票往往都是乱序的，这个大家应该毫无疑问。那么在这种情况下，MaxVote的值可能是会错的。想象一下，在算出一个MaxVote(5,...)之后，才出现一个编号比5小的投票，那么这个投票很有可能会影响到这个MaxVote的值。也就是一个先后来到的乱序问题。而如果MaxVote是错的，我们的证明就失效了。

改进后的算法

编号 (bal)/ 参与者 (vote)	提议 (dec)	A	B	C	D	E	Maxvote
2		Y				Y	MaxVote(2, (A,C,D), β)dec = null
5			Y	Y	Y		MaxVote(5, (B,C,D), β)dec = null
3		Y	Reject	Reject	Reject	Y	MaxVote(3, (A,B,C,D,E), β)dec = ?

算法增加约束：
当投票者_p返回了MaxVote(Bbal, p, β)，为了使得这个结果是正确的，不能再进行任何投票B'使得B'bal < Bbal。
如上表格，当MaxVote(5, (B,C,D), β)dec成功获得后，B,C,D三个投票者都不再接受编号3的投票，使得编号3的投票无法获得多数派的MaxVote从而放弃投票。从而满足投票约束！

页12注解

为了满足这个约束，我们需要对MaxVote的计算过程进行约束。
看过Paxos算法过程的，且是聪明的读者，看完这页可能会想起，哦原来Prepare的Promise要求是这么来的。

正式算法的提出(数学定义)

参与角色 <i>Proposer</i> : 对应提议者 <i>Acceptor</i> : 对应投票者	变量 <i>b</i> : 对应投票编号 <i>bal</i> <i>v</i> : 对应投票提议 <i>dec</i>
角色状态 <i>pb</i> : 承诺不再接受编号小于 <i>pb</i> 的投票 <i>ab</i> : 最大编号投票对应的 <i>bal</i> <i>av</i> : 最大编号投票对应的 <i>dec</i>	
算法阶段 <i>Prepare(b)</i> : 对应获取 <i>MaxVote(b, qrm, β)</i> 的过程 <i>Accept(b,v)</i> : 对应 <i>Bbal=b, Bdec=v</i> 的投票过程	

页13注解

到这里算法已经是非常完善了，剩下就是怎么将这个算法引申到计算机上，在计算机层面上提出算法的过程。大家可以看到实际的算法过程，很多角色都是与我们刚刚描述的东西相对应的。

正式算法的过程

Proposer	Acceptor
<i>last_prepare_b</i> = 0, <i>maxb</i> = 0	<i>pb</i> = 0, <i>ab</i> = 0, <i>av</i> = null
<i>b</i> = <i>last_prepare_b</i> + 1 <i>maxb</i> = 0, <i>v</i> = 😊 Prepare(<i>b</i>)	OnPrepare(<i>b</i>): if <i>b</i> >= <i>pb</i> : <i>pb</i> = <i>b</i> , Prepare_Response (<i>ok</i> , <i>ab</i> , <i>av</i>) else Prepare_Response (<i>reject</i> , 0, null)
OnPrepare_Response(<i>ok</i>, <i>ab</i>, <i>av</i>): if <i>ab</i> > <i>maxb</i> and <i>av</i> != null: <i>maxb</i> = <i>ab</i> , <i>v</i> = <i>av</i> if count(<i>ok</i>) >= count(<i>qrm</i>): Accept(<i>b</i>, <i>v</i>)	OnAccept(<i>b</i>, <i>v</i>): if <i>b</i> == <i>pb</i> : <i>ab</i> = <i>b</i> , <i>av</i> = <i>v</i> , Accept_Response (<i>ok</i>) else Accept_Response (<i>reject</i>)
OnAccept_Response(<i>ok</i>): if count(<i>ok</i>) >= count(<i>qrm</i>): done()	

页14注解

正式算法过程，也就是Lamport的论文 "Paxos Made Simple" 提出的。

我希望大家再回头看这个算法过程时候，知道每一步的含义，以及背后的本质。

算法实例演示

		Acceptor_A			Acceptor_B			Acceptor_C		
		<i>pb</i>	<i>ab</i>	<i>av</i>	<i>pb</i>	<i>ab</i>	<i>av</i>	<i>pb</i>	<i>ab</i>	<i>av</i>
Acceptor	init	0	0	null	0	0	null	0	0	null
Proposer_A	prepare(5)	response(ok, 0, null)			response(ok, 0, null)			response(ok, 0, null)		
Acceptor	before_res	5	0	null	5	0	null	5	0	null
Proposer_A	accept(5, "a")	response(ok)						response(ok)		
Acceptor	before_res	5	5	"a"	5	0	null	5	5	"a"
Proposer_B	prepare(3)	response(reject)						response(reject)		
Proposer_C	prepare(8)				response(ok, 0, null)			response(ok, 5, "a")		
Acceptor	before_res	5	5	"a"	8	0	null	8	5	"a"
Proposer_C	accept(8, "a")				response(ok)			response(ok)		
Acceptor	before_res	5	5	"a"	8	8	"a"	8	8	"a"

页15注解

一个过程演示，这个就不多解释了。

结语

朴素Paxos算法不容易，需要反复的推敲，建议有耐心的读者多看几遍，只要有耐心，肯定是可以弄懂的。关于这个算法有什么用？如何扩展到确定多个值？如何实现以及工程化，我们后续会继续分享。如需了解更多，可关注我们的公众号：微信后台团队。

发布于 2016-06-28 12:38，编辑于 2016-06-28 13:28

「真诚赞赏，手留余香」

赞赏

1 人已赞赏



微信 分布式 一致性



欢迎参与讨论

54 条评论

默认 最新



lzhjie

证明要求：
当出现一轮投票B获得多数派通过，那么不可能出现一轮投票B'，使得B'bal > Bbal，而B'dec != Bdec.

反证法用MaxVote的定义得出矛盾，总觉得太绕，这样证明是否可行
证明过程：
1) 挑选C为编号大于B的投票通过中最小编号的一轮。
2) 由于B和C投票通过，所以Carm必然包含Bvot一个成员。（B2得出）
3)

- 另MaxVote (Cbal, Cqrm,) = B。 (步骤1和MaxVote定义得出)

2017-02-09

回复 2
- Elon Wen

先看了原版论文，再看到这篇文章，把一些之前不太清晰的地方都用通俗的语言讲清楚了，写的很好。赞！不过证明部分还是原文比较严谨（偷笑～）

2017-02-08

回复 1
- 酸性沼泽软泥怪

问一个弱智问题：“三个人分别只允许呆在不同的三个城市”他们之间是通过什么途径通信呢？

2016-06-28

回复 1
- LynnCui 作者

可以通过一切途径，比如微信，电话，飞鸽传书等。但这跟面对面交流的区别是，微信服务器可能突然被炸掉，电话公司倒闭，鸽子中途被杀来烧烤！)

2016-06-28

回复 8
- 无名氏 ▶ LynnCui

哈哈

2016-12-02

回复 1
- WalkerOnSnitches

没搞懂为什么要多轮投票，一旦出现一次成功的投票，直接整个系统停止并拒绝之后一切协议包不就完了吗，为什么要让后续投票保持和前面投票结果一致呢

2021-10-25

回复 喜欢
- 草薨绘名

因为有可能会有一些成员failover（原本的共识可能会重新变成非majority）这个时候需要重新建立一次共识

2023-04-20

回复 喜欢
- Park

卧槽和我刚学的时候的疑惑一样，paxos虽然是多数派投票但最终的目的肯定还是让所有accepter日志一样，个人理解发送accept请求时发的是小于当前编号n的最大accepted值就是在同步（补齐）日志。还有就是，“投票成功就拒绝后面一切”那这个集群岂不是只能接受一次请求🐼

2021-12-22

回复 喜欢
- oronge

博主好，我是刚学 paxos，想请教个问题，在第二阶段，当 acceptor 接到 accept 请求并接受其中的值后是不是应该向 proposer 回送一个确认，不然如果该 accept 请求丢失，这个 acceptor 就会少一个值，感谢！

2020-06-06

回复 喜欢
- fanatic

逻辑没问题，但是maxvote这个点出来得有点突兀了。有点先出结论再论证的感觉。不过原文也是这种感觉，说到底还是大神的思维不一样。。。

2019-07-17

回复 喜欢
- prehonor

如果编号5那轮投票,看到编号3而没看到编号3的投票的提议,但看到编号2的投票的提议,那他是自己提一个新的提议还是使用编号2的提议呢

2019-01-19

回复 喜欢
- 陈Qiu凯

一个下午反复看了前辈的文章，以及评论里朱一聪前辈在别的问题下的回答，建议两个结合着看，对整个推导过程的理解十分有帮助，算是拨云见雾了。

2018-12-09

回复 喜欢
- chosenOne

在正式算法过程中，Acceptor的OnPrepare函数，如果b==pb就promise的话会有问题吧？比如，有两个Proposer提议的编号相同，先后在accept之前发送给Acceptor，都会得到promise，因为此前没有accept，那么这两Proposer可以继续accept阶段，并可以随便提议之前想要的值，最后会导致不一致

2018-11-01

回复 喜欢

这是在工程实现上需要解决的问题 不在paxos算法本身的内容 只要能满足proposal的编号的两个要求 全局唯一性并且可以比较 编号可以是复合数据类型 有很多不同解决方案 timestamp+server id sequence + server id等


2019-12-21 回复 喜欢

@ 卜丁 ▸ Jeson ...

不同机器是独立的，应该怎样保证两个编号相同的proposal就是同一个proposal?

2019-12-21 回复 喜欢

展开其他 3 条回复 >

 我擦laughing ...

您好，请教一个问题，在这paxos算法中，如果有proposer (1,2, 3) ， acceptor (1,2, 3) 。一阶段提交的时候，p2生成 (id2, v2) ， p1生成 (id1, v1) ， 其中id1>id2。如果p2先prepare了a2, a3。之后p1prepare了a1, a2。此时p1和p2都认为自己通过了 (n/2 +1) 的通知。二阶段提交的时候，p2先请求了a3(得到ack), a2 (得到error) 。p1请求了a1 (得到ack) ， a2 (得到ack) 。此时a1, a2,a3的决策结果是不一样的，那么是不是要再通过其它手段来确定最终决策结果吗？这里有点疑惑，希望您的解答

2018-07-11 回复 喜欢

[点击查看全部评论 >](#)



欢迎参与讨论



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

Abstract

Paxos algorithm, when presented in plain English, is very simple.

Paxos讨论

陈钦霖

微信PaxosStore：深入浅出 Paxos算法协议

“与其预测未来，不如限制未来”，这应该是Paxos协议的核心思想。Paxos协议本身是比较简单的，如何将Paxos协议工程化，才是真正的难题。这是来自微信工程师的经验，以供参考。引言早在1990...

OpenIM



Paxos Made Easy: Paxos算法的几何意义与证明

hnes 发表于Turin...



打个比方说 Paxos 的过种 确性

欢歌