PaxoSim

Summary of Distributed System Project

梁凉 5100379044，许悦韬 5100379039

# Implementation

## Architecture

本项目完全通过模拟来实现，即完全不使用socket等库进行网络包的收发，转而虚拟出一整套网络环境，并根据网络环境提高一套收发消息的接口，同时由一个统一的调度器负责虚拟环境中网络节点的运行。这种方式，在模拟网络故障时，拥有较大的灵活性，同时方便演示的进行。

我们根据现实的情况，认为应虚拟出如下结构：framework, node, connection.

### Node

Node对应着现实中的网络节点。Node可以发送或接收数据包，并拥有一定概率会崩溃一段时间，重新启动后可以根据之间保存的数据进行一定程度的恢复。在Node中运行着Paxos算法。Node的Storage为Paxos算法提供持久化支持。

Node有2个参数可以配置，分别是故障率*failRate(nfr)*和每次故障平均时间*averageFailTime(nft)*。为保证*nfr*为故障时间占总运行时间比例的期望值，单位时间内，每个node发生故障的概率是：

每次node发生故障的时间为*[0, 2nft]*的随机数。

### Connection

Connection用于模拟节点之间的网络连接。Connection的传输速度可以变化，并且拥有一定的丢包率和重发包率，同时存在着一定的连接中断几率。基本上可以模拟现实的网络连接和Paxos运行环境。

Connection有4个参数可以配置，分别是lossRate(clr)，dupRate(cdr)，minTime(cmint), maxTime(cmaxt)。在发送消息按以下方式处理故障：

|  |
| --- |
| Function SendWithFailure(clr, cdr)  If rand() < cdr: SendWithFailure // 重复发送消息  If rand() < clr: Return // 遗失消息  SendMessage() |

消息传递的时间在*[cmint, cmaxt]*范围内随机。很明显，该区间范围越大，消息顺序就越可能被打乱。

### Framework

Framework用于初始化Node以及Connection的属性，调度Node的执行。并以给定时间为单位逐次递进整个系统内的状态。

上述结构的关系如图

**Framework**

**Node**

**1**

**Node N**

**……**

**Connection**

**Node**

**2**

**Interface**

**Send Message**

**Redirect to Connection**

**Receive Message**

## Paxos

PAXOS按照Lamport论文中的描述实现。

每个Paxos可以设置Timeout时间，当其发送prepare后经过timeout时间仍然未收到足够多的promise，则其会更新N并重发prepare。

为了保证每个进程的N不同，不同Node上的Paxos算法在启动时会附带一个参数i。如有P个Node，则i从0到P-1不重复地初始化各个Node上的Paxos。每次某个Paxos进程需要增加N时，直接在当前N值上加P。因此Node-0会产生[0, P, 2P, 3P …]，Node-1会产生[1, P + 1, 2P + 1, 3P + 1 …]，Node-P-1会产生[P-1, P + P - 1, 2P + P – 1, 3P + P – 1 …]。从而保证各个Node产生的N均不相同。

为了能在Crash后恢复，每次Paxos在发送消息前需要把当前状态保存在Node的持久化存储下。

# Evaluation

注：每次测试均进行100次，取平均值以减少噪音。

## Node Failure

### 目的

测试各个节点故障概率下Paxos算法达成一致所需要的轮数。

### 设置

|  |
| --- |
| Timeout: 100 ms  Message deliver time: 5 – 20 ms  Node fail time: 0 – 100 ms  Message loss rate: 0  Message duplicate rate: 0  Proposer number: 1  Node fail rate: X |

### 测试结果

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Average Consensus Rounds | P = 3 | 10 | 30 | 100 |
| X = 0 | 1.00 | 1.00 | 1.00 | 1.00 |
| 0.1 | 1.09 | 1.01 | 1.06 | 1.06 |
| 0.2 | 1.33 | 1.21 | 1.09 | 1.18 |
| 0.3 | 1.87 | 1.47 | 1.24 | 1.31 |
| 0.4 | 2.9 | 2.13 | 1.48 | 1.42 |
| 0.5 | 5.01 | 4.08 | 3.21 | 3.65 |

*直方图，纵坐标为轮数，横坐标为X，P=3, 10, 30, 100组成一簇。*

### 结论

可见在节点故障率较小的情况下对Paxos算法的影响较小。即使每个节点有0.2的故障率（在现实生活中已经属于极高），平均需要达成一致所需的轮数也只增加约20%。

## Connection Failure

### 目的

测试各种网络故障下Paxos算法达成一致所需要的轮数。

### 设置

|  |
| --- |
| Message deliver time: 5 – 20 ms  Node failure rate: 0  Message loss rate: X  Message duplicate rate: X  Proposer number: 1 |

### 测试结果

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Average Consensus Rounds | N = 3 | 10 | 30 | 100 |
| X = 0 | 1.00 | 1.00 | 1.00 | 1.00 |
| 0.1 | 1.44 | 1.06 | 1.00 | 1.00 |
| 0.2 | 2.03 | 1.28 | 1.01 | 1.00 |
| 0.3 | 2.36 | 2.16 | 1.16 | 1.01 |
| 0.4 | 3.11 | 2.52 | 1.82 | 1.17 |
| 0.5 | 3.62 | 3.70 | 2.96 | 2.35 |

*直方图，纵坐标为轮数，横坐标为X，P=3, 10, 30, 100组成一簇。*

### 结论

可见在网络故障率较小的情况下对Paxos算法的影响较小。节点越多，对网络故障就越鲁棒。若只有3个节点，则0.3的丢失率和重复率会导致平均所需要的投票轮数增加136%；若有100个节点，则相同环境下，投票轮数只增加1%。

## Node and Connection Failure

### 目的

测试节点故障和网络故障同时存在的环境下Paxos算法达成一致所需要的轮数。

### 设置

|  |
| --- |
| Message deliver time: 5 – 20 ms  Node fail time: 0 – 100 ms  Node failure rate: X  Message loss rate: X  Message duplicate rate: X  Proposer number: 1 |

### 测试结果

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Average Consensus Rounds | N = 3 | 10 | 30 | 100 |
| X = 0 | 1.00 | 1.00 | 1.00 | 1.00 |
| 0.1 | 1.63 | 1.16 | 1.06 | 1.07 |
| 0.2 | 3.17 | 2.43 | 1.23 | 1.16 |
| 0.3 | 4.74 | 7.18 | 6.34 | 6.69 |
| 0.4 | 12.42 | 25.96 | 87.05 | TLE |
| 0.5 | 27.83 | 171.82 | TLE | TLE |

TLE：Time Limit Exceeded. 达成一致所需的时间太多，无法在较短时间内完成模拟。

*直方图，纵坐标为轮数，横坐标为X，P=3, 10, 30, 100组成一簇。*

### 结论

在故障率较低（0-0.2）的情况下，即在现实中可能出现的情况下，故障对Paxos算法的影响较小；且节点越多，所需要的投票轮数就越少。

在故障率较高（0.3+）的情况下，投票轮数呈指数级上升趋势，且与前面测试结果相反的是：节点越多，所需要的投票轮数就越多。不过现实中一般不会有如此高的故障率。

## Multiple Proposers

### 目的

测试下多个Proposer对Paxos算法达成一致所需要的轮数造成的影响。

### 设置

|  |
| --- |
| Message deliver time: 5 – 20 ms  Node fail time: 0 – 100 ms  Node failure rate: 0.3  Message loss rate: 0.3  Message duplicate rate: 0.3  Proposer number: X |

### 测试结果

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Average Consensus Rounds | N = 3 | 10 | 30 | 100 |
| X = 1 | 5.08 | 6.18 | 5.06 | 5.02 |
| 2 | 5.91 | 6.72 | 5.94 | 7.34 |
| 3 | 6.38 | 6.53 | 5.6 | 6.53 |
| 10 | N/A | 8.39 | 6.3 | 5.88 |
| 30 | N/A | N/A | 5.33 | 6.47 |
| 100 | N/A | N/A | N/A | 4.30 |

N/A：Proposer比节点数多，不可能出现的情况。

*直方图，纵坐标为轮数，横坐标为X，P=3, 10, 30, 100组成一簇。*

### 结论

可见多个Proposer不会对投票轮数产生特别明显的影响，但多个Proposer会明显对消息数目产生影响。

# Conclusion

经过测试我们发现，我们实现的Paxos算法对于节点和网络故障均有良好的鲁棒性，也不会被多个Proposer造成巨大的影响。但我们的实现无法处理拜占庭错误。

# Teamwork

|  |  |
| --- | --- |
| Framework | 梁凉 |
| Paxos | 许悦韬 |
| UI | 梁凉 & 许悦韬 |
| Experiments | 梁凉 & 许悦韬 |