云计算项目设计实验报告

——分布式算法 Paxos 算法 c++实现

刘亚辉 16340157

一、 参考资料

https://blog.csdn.net/sparkliang/article/details/5740882 （原文中文）
http://www.jdon.com/artichect/paxos.html （分布式算法介绍）
http://iunknown.iteye.com/blog/2246484?from=message&isappinstalled=0 （两军问题）
http://www.cnblogs.com/endsock/p/3480093.html （现实问题描述算法）
https://en.wikipedia.org/wiki/Paxos_(computer_science) （Wiki）

二、 Paxos 算法原理

Paxos 算法解决的问题是一个分布式系统如何就某个值达成一致。在一个分布式系统中，如果各节点的初始状态一致，每个节点都执行相同的操作序列，那么他们最后能得到一个一致的状态，。为保证每个节点执行相同的命令序列，需要在每一条指令上执行一个“一致性算法”以保证每个节点看到的指令一致。

Paxos 算法中的四种角色：

Proposer：提议者

提议者用户提出议题，带着 Client 产生的议题，向 Acceptor 提出。

Acceptor：决策者

处理 Proposer 提出的议题。

Client：产生议题者

产生议题交由 Proposer 提出。

Learner：最终决策学习者

Learner 用于找到一个被多数 acceptor 批准的议案，并通知其他 learner，如此就可以得知是否选择了一个决议。

在这四种角色中，Proposer 和 Acceptor 起主要作用，Proposer 可以合并 Client 和 learner：Proposer 作为 Client 的使者，向 Acceptor 提出 Client 产生的议题，让 Acceptor 来决策。其中，Paxos 算法中以上两种角色行为如下：

1. Proposer 提出议题；
2. Acceptor 初步接受（承诺）或者初步不接受（拒绝）；
3. 如果第 2 步 Acceptor 初步接受（承诺），则 Proposer 再次向 Acceptor 确认是否最终接受；如果第 2 步半数以上 Acceptor 初步不接受，则 Proposer 可以学习被接受的提案，并回到第 1 步；
4. Acceptor 最终接受或者最终不接受。

分为两阶段提交的理解：如果 Proposer 仅一次提交议题，那么该议题可能不被 Acceptor 承诺，或者承诺之后又被其他 Proposer 提出的议题所覆盖，最终导致 Proposer 无法与其他 Proposer 达成一致，一致性验证出错。分为两段提交，可以规避这个问题，在第一阶段只是承诺，可以通过第二步进一步验证；如果在第一步不被 Acceptor 承诺，则可以进行学习，再次提出议题。

Paxos 约束条件：

1. Acceptor 必须承诺它接收到的第一个提案。
2. 一个提案被选中需要过半数的 Acceptor 接受。
3. 当编号为 proposalId，值为 proposalValue 的提案（{proposalId，proposalValue}）被过半的 Acceptor 接受后，之后只有值等于 proposalValue 且编号大于 proposalId 的提案才可以被接受。
4. 只有 Acceptor 没有承诺过的提案才能采用自己的值，否则 Proposer 需要学习其他提案（被接受的提案中编号最大的）。

下面以多军作战问题为原型介绍 paxos 的正确性及其执行过程：

计算机通讯理论中的两军问题讲述通信的双方通过 ACK 来达成共识。这与 Paxos 算法有类似之后。通信双方需要达成共识，并且在通信不稳定的情况下消息也不会被篡改。

假设现在有四支军队，红蓝双方，红方处于劣势只有一支军队，蓝方有三支军队。蓝方相距较远，之间需要传令兵来传递命令。并且，蓝方任意一支军队无法战胜红方军队，只有两支及以上蓝方军队同时进攻红方军队蓝方才可以获胜。所以，蓝方各军队需要进行通信达成共识，选择在同一时间发起进攻。

现在进行模拟蓝方军队之间的通讯：每只军队都有一个参谋负责提出进攻时间，每只军队也有一个将军来批准或否决参谋提出的进攻时间。（参谋数目也不一定要和将军数目相等）不显式定义 learner 和 client 的角色，参谋可以完成这两部分工作。

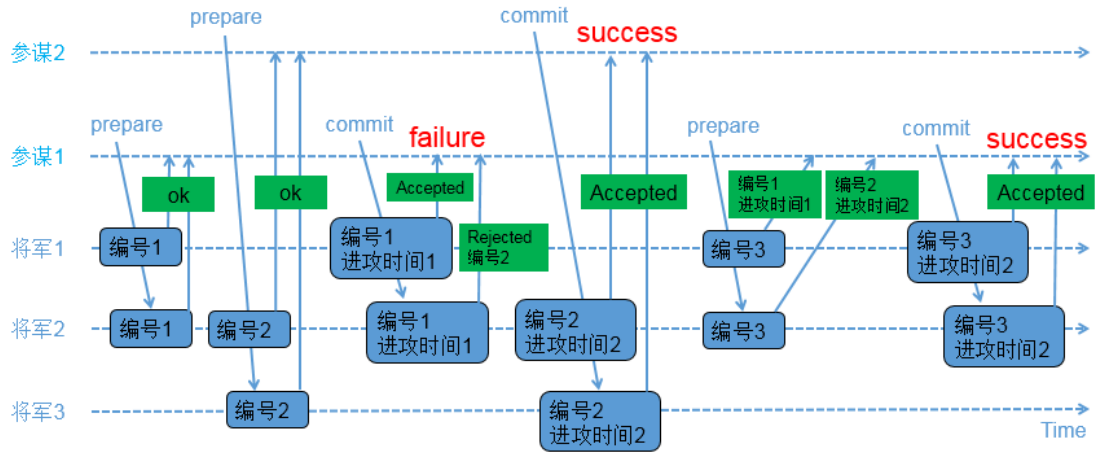
那么，参谋和将军的行为如下：

1. 参谋提出作战时间，附上编号
2. 将军初步接受（承诺）或者初步不接受（拒绝），根据编号的大小来进行选择
3. 超过半数将军初步接受（承诺）作战时间，参谋再次让各位将军最终接受该作战时间；否则参谋学习其他被将军接受的作战时间（编号最大的），并回到第一步
4. 将军最终接受一个作战时间，且作战时间的编号可以修改，但是值不能修改。

之所以在此过程中使用两段提交，是因为参谋的两段提交作战时间可以规避自己第一阶段提交议题被其他参谋议题覆盖的问题，并利用学习再次提交的方式达成一致。下面以该多军作战问题为原型尝试实现 Paxos 算法。

http://iunknown.iteye.com/blog/2246484?from=message&isappinstalled=0

解释如下：



上图中，参谋 1，参谋 2 的执行过程如下：

1） 参谋 1 发起提议，派通信兵带信给 3 个将军，内容为（编号 1）；

2） 3 个将军的情况如下

a) 将军 1 和将军 2 收到参谋 1 的提议，将军 1 和将军 2 把（编号 1）记录下来，如果有其他参谋提出更小的编号，将被拒绝；同时让通信兵带信回去，内容为（ok）；

b) 负责通知将军 3 的通信兵被抓，因此将军 3 没收到参谋 1 的提议；

3） 参谋 2 在同一时间也发起了提议，派通信兵带信给 3 个将军，内容为（编号 2）；

4） 3 个将军的情况如下

a) 将军 2 和将军 3 收到参谋 2 的提议，将军 2 和将军 3 把（编号 2）记录下来，如果有其他参谋提出更小的编号，将被拒绝；同时让通信兵带信回去，内容为（ok）；

b) 负责通知将军 1 的通信兵被抓，因此将军 1 没收到参谋 2 的提议；

5） 参谋 1 收到至少 2 个将军的回复，再次派通信兵带信给有答复的 2 个将军，内容为（编号 1，进攻时间 1）；

6） 2 个将军的情况如下

a) 将军 1 收到了（编号 1，进攻时间 1），和自己保存的编号相同，因此把（编号 1，进攻时间 1）保存下来；同时让通信兵带信回去，内容为（Accepted）；

b) 将军 2 收到了（编号 1，进攻时间 1），由于（编号 1）小于已经保存的（编号 2），因此让通信兵带信回去，内容为（Rejected，编号 2）；

7） 参谋 2 收到至少 2 个将军的回复，再次派通信兵带信给有答复的 2 个将军，内容为（编号 2，进攻时间 2）；

8） 将军 2 和将军 3 收到了（编号 2，进攻时间 2），和自己保存的编号相同，因此把（编号 2，进攻时间 2）保存下来，同时让通信兵带信回去，内容为（Accepted）；

9） 参谋 2 收到至少 2 个将军的（Accepted）内容，确认进攻时间已经被多数派接受；

10） 参谋 1 只收到了 1 个将军的（Accepted）内容，同时收到一个（Rejected，编号 2）；参谋 1 重新发起提议，派通信兵带信给 3 个将军，内容为（编号 3）；

11） 3 个将军的情况如下

a) 将军 1 收到参谋 1 的提议，由于（编号 3）大于之前保存的（编号 1），因此把（编号 3）保存下来；由于将军 1 已经接受参谋 1 前一次的提议，因此让通信兵带信回去，内容为（编号 1，进攻时间 1）；

b) 将军 2 收到参谋 1 的提议，由于（编号 3）大于之前保存的（编号 2），因此把（编号 3）保存下来；由于将军 2 已经接受参谋 2 的提议，因此让通信兵带信回去，内容为（编号 2，进攻时间 2）；

c) 负责通知将军 3 的通信兵被抓，因此将军 3 没收到参谋 1 的提议；

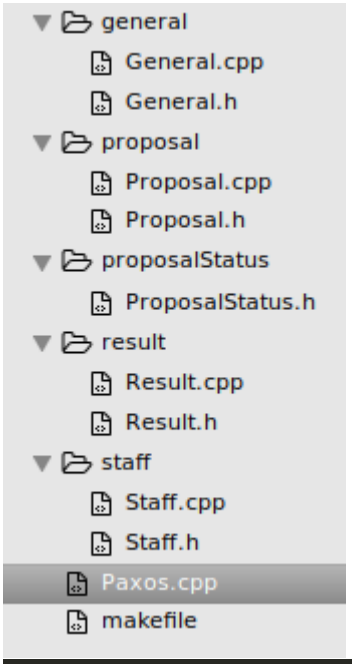
12） 参谋 1 收到了至少 2 个将军的回复，比较两个回复的编号大小，选择大编号对应的进攻时间作为最新的提议；参谋 1 再次派通信兵带信给有答复的 2 个将军，内容为（编号 3，进攻时间 2）；

13） 将军 1 和将军 2 收到了（编号 3，进攻时间 2），和自己保存的编号相同，因此保存（编号 3，进攻时间 2），同时让通信兵带信回去，内容为（Accepted）；

14） 参谋 1 收到了至少 2 个将军的（accepted）内容，确认进攻时间已经被多数派接受；

三、 实验模块设计

以多军作战为原型：实现军队之间的通讯，保证作战时间的一致性。



主要代码有左侧几个模块：general 将军模块、staff 参谋模块，proposal 作战时间提议模块、result 将军返回信息模块、proposalStatus 提议接受状态模块以及主函数文件 Paxos.cpp

下面一一介绍每个模块的实现及功能：

general 模块：

此模块主要实现的功能是处理参谋发出的提议的请求，接受还是拒绝，拒绝时需要返回给参谋已经接受的提议供参谋学习。如下图所示，有两个函数 handlePrepareTocommit（承诺模块）和 handleCommit（最终接受模块）分别处理两个阶段参谋的提交。

```
//this method will return the info whether the propos
Result *handlePrepareToCommit(Proposal *proposal);
//this method will return the info whether the propos
Result *handleCommit(Proposal *proposal);
```

staff 模块：

此模块是参谋两阶段提出作战时间的功能模块，如下图所示，prepareToCommit 为第一阶段，请求半数以上的将军初步接受（承诺）自己的作战时间；当第一阶段成功之后 commit 提交作战时间请求将军最终接受。

```
    //in this method, the staff prepare to propose the attack time.
    bool prepareToCommit();
    //in this method, the staff commit the attack time.
    void commit();
};
```

proposal 模块：

此模块主要存储的实现作战时间的内容，包括编号及具体作战时间。

```
    //define some methods to set and get information.
    int getId();
    void setId(int _id);
    string getAttackTime();
    void setAttackTime(string _attackTime);
```

result 模块：

此模块是将军返回给参谋的信息模块，包括是否承诺，是否接受等信息，并附上已经承诺、接受的作战时间信息。

```
//accepted represents whether the current proposal has been promised or not.
void setPromise(bool _promise);
bool getPromise();

//accepted represents whether the current proposal has been accepted or not.
void setAccepted(bool _accepted);
bool getAccepted();

//result contains the current promised proposal or accepted proposal
void setProposal(Proposal* _proposal);
Proposal* getProposal();

//status denotes the general's status
void setStatus(Status _status);
int getStatus();
```

proposalStatus 模块：

此模块只是提供将军承诺、接受状态值。

```
//define three status
//NONE denotes that there is not a proposal has been proposed.
//PROMISED denotes that there has been a promised proposal.
//ACCEPTED denotes that there has been a accepted proposal.
enum Status{NONE, PROMISED, ACCEPTED};
```

主函数模块：

此模块用与创建将军和参谋实例，并模拟参谋提出作战时间，将军进行请求以验证结果的正确性。下面代码为加锁、释放锁来进行线程同步。

```
pthread_mutex_lock(&prepareMutex);
cout<<"The staff "<<staff->getId()<<" prepares to commit."<<endl;
staff->prepareToCommit();
cout<<"The staff "<<staff->getId()<<" has finished preparing to commit."<<endl;
//unlock
pthread_mutex_unlock(&prepareMutex);

pthread_mutex_lock(&commitMutex);
cout<<"The staff "<<staff->getId()<<" commits."<<endl;
staff->commit();
cout<<"The staff "<<staff->getId()<<" has finished committing."<<endl;
pthread_mutex_unlock(&commitMutex);
```

四、 关键代码分析

staff 模块 prepareToCommit 函数分析：

在下方实现代码中，while 循环只有当该参谋的作战时间被半数以上的将军所承诺或者已经有一个提议被接受的时候才退出循环。

for 循环中给所有将军通讯，获得他们的返回信息，并记录承诺的将军数和已经接受其他作战时间的各自的将军数目。循环结束后，判断是否有半数以上的将军作出承诺；判断是否已经有被接受的作战时间。若当前参谋的提议未被承诺，则学习编号最大的提议的作战时间值，并再一次向所有将军发送请求。如此循环知道满足退出循环的条件。

```
bool Staff::prepareToCommit(){
    int promisedCount=0;
    //this loop will be break if the staff's proposal has been promised by more than
    //half of generals or there is an accepted proposal.
    while(true){
        //use map to count how many generals has accepted the proposals
        map<Proposal*, int> acceptedProposals;
        promisedCount=0;
        for(auto general:generals){
            Result* result = general->handlePrepareToCommit(proposal);

            //Analog the cost of time when deliver the information
            sleep(rand()%2);

            //if the information was lost, the continue
            if(nullptr == result)
                continue;

            //if the staff's proposal has been promised, the count increases by 1.
            if(result->getPromise())
                promisedCount++;
            else{
                //judge if there is a proposal has been accepted.
                if (result->getStatus() == ACCEPTED){
                    acceptedProposals[result->getProposal()]++;
                }
            }
            delete result;
        }

        //if more than half generals has promised the proposal, the loop can break.
        if(promisedCount >= halfGeneralsCount){
            for(auto iter=acceptedProposals.begin(); iter!=acceptedProposals.end(); iter++)
                delete iter->first;
            break;
        }

        //traverse the entire map, try to find a proposal that has been accpted by more than half gengerals.
        for(auto iter=acceptedProposals.begin(); iter!=acceptedProposals.end(); iter++)
            if(iter->second >= halfGeneralsCount){
                std::cout<<"Staff"<<staffId<<": attack time has been determined! ["<<proposal->getAttackTime()<<"]"<<std::endl;
                for(auto iter=acceptedProposals.begin(); iter!=acceptedProposals.end(); iter++)
                    delete iter->first;
                return true;
            }
```

```
        //update the propsoal's id, then the staff can prepare to commit again.
        proposal->setId(generateId(staffId, cycleCount, staffsCount));
        //find the proposal that in acceptedProposals which has the max id.
        //then the staff use its attackTime, because with our method, the proposal with max id more likely to be accepted.
        if(acceptedProposals.size()>0){
            auto iter=acceptedProposals.begin();
            int maxxId=(iter->first)->getId();
            proposal->setAttackTime((iter->first)->getAttackTime());

            for(; iter!=acceptedProposals.end(); iter++){
                if((iter->first)->getId()>maxxId)
                    proposal->setAttackTime((iter->first)->getAttackTime());
            }
        }
        cycleCount++;

        for(auto iter=acceptedProposals.begin(); iter!=acceptedProposals.end(); iter++)
            delete iter->first;
    }
    return false;
}
```

staff 模块 commit 代码分析：

此模块主体与上方的代码类似，退出循环的条件为参谋的提议被半数以上的将军所接受。For 循环中向所有的将军发送请求，使其接受参谋的提出的作战时间，并记录接受的将军数和已经接受其他作战时间的各自的将军数目。退出循环之后，判断作战时间是

否被半数以上将军所接受，若条件为假，则需要学习编号较高的其他作战时间的值。然后再次提交。

```
void Staff::commit(){
    int acceptedCount=0;
    //this loop will be break if there is an accepted proposal.
    while(true){
        //similarly, use map to count how many generals has accepted the proposals
        map<Proposal*, int> acceptedProposals;
        acceptedCount=0;

        for(auto general:generals){
            Result* result = general->handleCommit(proposal);

            //Analog the cost of time when deliver the information
            sleep(rand()%2);
            //if the information was lost, the continue
            if(nullptr == result)
                continue;

            //if the staff's proposal has been accepted, the count increases by 1.
            if(result->getAccepted())
                acceptedCount++;
            else{
                //record the accepted proposal and increase the count.
                acceptedProposals[result->getProposal()]++;
            }
            delete result;
        }
        //if more than half generals has accepted the proposal, the loop can break.
        if(acceptedCount>=halfGeneralsCount){
            std::cout<<"Staff"<<staffId<<": attack time has been determined! ["<<proposal->getAttackTime()<<"]"<<std::endl;
            for(auto iter=acceptedProposals.begin(); iter!=acceptedProposals.end(); iter++)
                delete iter->first;
            return;
        }else{
            //update the propsoal's id, then the staff can commit again.
            proposal->setId(generateId(staffId, cycleCount, staffsCount));
            //find the proposal that in acceptedProposals which has the max id.
            //then the staff use its attackTime, because with our method, the proposal with max id more likely to be accepted.
            if(acceptedProposals.size()>0){
                auto iter=acceptedProposals.begin();
                int maxxId=(iter->first)->getId();
                proposal->setAttackTime((iter->first)->getAttackTime());

                for(; iter!=acceptedProposals.end(); iter++){
                    if((iter->first)->getId()>maxxId)
                        proposal->setAttackTime((iter->first)->getAttackTime());
                }
            }
            cycleCount++;
            for(auto iter=acceptedProposals.begin(); iter!=acceptedProposals.end(); iter++)
                delete iter->first;
            //if the attack time has not been made, the staff can prepare and commit again.
            if(prepareToCommit())
                return ;
        }
    }
}
```

general 模块 handlePrepareToCommit 函数分析：
acceptorStatus 存储的是当前将军的接受状态：无，已承诺，已接受。当该将军没有接受或者承诺的作战时间时，第一个提议的作战时间必须被接受；当该将军已经承诺了一个作战时间时，假如有更高编号的作战时间，则更新承诺；当该将军已经接受了一个作战时间时，所能更新的承诺只可以是编号增大但时间不变的作战时间信息。在每一个 if 条件句中，都返回一个结果值，保存着当前承诺、接受的作战时间信息。这样做可以和第二阶段的请求返回值相类似，减少复杂度。

```
if(acceptorStatus == NONE){
    //store the info to result.
    result->setStatus(acceptorStatus);
    result->setPromise(true);
    result->setProposal(nullptr);
    //set the acceptorStatus to PROMISED
    acceptorStatus=PROMISED;
    //record the promised proposal
    promisedProposal->setId(proposal->getId());
    promisedProposal->setAttackTime(proposal->getAttackTime());
    return result;
}
if(acceptorStatus == PROMISED){
    //generals only update the promised proposal when the new proposal's id is bigger.
    if(promisedProposal->getId() > proposal->getId()){
        result->setStatus(acceptorStatus);
        result->setPromise(false);
        result->setProposal(promisedProposal);
    }else{
        promisedProposal->setId(proposal->getId());
        promisedProposal->setAttackTime(proposal->getAttackTime());

        result->setStatus(acceptorStatus);
        result->setPromise(true);
        result->setProposal(promisedProposal);
    }
    return result;
}

if(acceptorStatus==ACCEPTED){
    //when there is final decision, general update the proposal if only the id has been update.
    if(promisedProposal->getId() < proposal->getId() && promisedProposal->getAttackTime() == proposal->getAttackTime()){
        promisedProposal->setId(proposal->getId());

        result->setStatus(acceptorStatus);
        result->setPromise(true);
        result->setProposal(promisedProposal);
    }
    else{
        result->setStatus(acceptorStatus);
        result->setPromise(false);
        result->setProposal(acceptedProposal);
    }
    return result;
}

return nullptr;
```

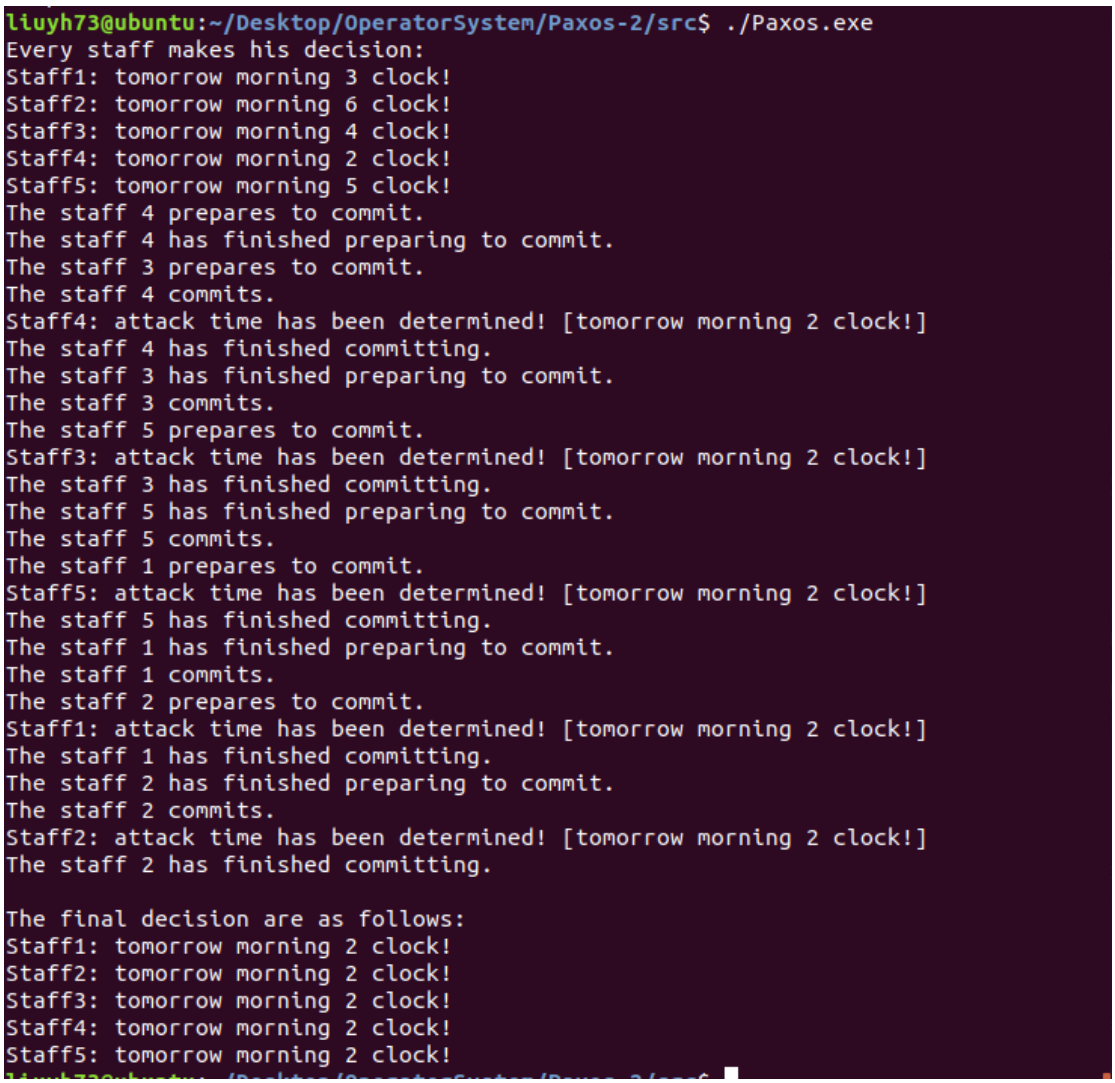
general 模块 handleCommit 函数分析：

在该模块中，将军只有两个状态，PROMISED 或者 ACCEPTED。当将军处于承诺状态中时，表示现在还没有被接受的作战时间，所以此时只能接受比已经承诺的作战信息编号大的作战时间，并更新为接受状态；当将军处于接受状态时，只能更新作战时间不变编号增大的信息。返回提示信息值，保存着当前接受的作战时间信息。

```
Result* General::handleCommit(Proposal *proposal){
    Result *result = new Result();

    //Analog information is intercepted.
    if(rand()%4==0){
        delete result;
        return nullptr;
    }

    //now the acceptorStatus can only be equal to PROMISED or ACCEPTED
    if(acceptorStatus==PROMISED){
        //when there is not the accepted proposal, the new committing proposal
        //can be update only its id isn't smaller than promised proposal
        if(proposal->getId() >= promisedProposal->getId()){
            promisedProposal->setId(proposal->getId());
            promisedProposal->setAttackTime(proposal->getAttackTime());

            //set the accepted proposal.
            acceptedProposal->setId(proposal->getId());
            acceptedProposal->setAttackTime(proposal->getAttackTime());
            //set the acceptorStatus to ACCEPTED
            //set the result
            acceptorStatus=ACCEPTED;
            result->setAccepted(true);
            result->setStatus(acceptorStatus);
            result->setProposal(proposal);
        }else{
            result->setAccepted(false);
            result->setStatus(acceptorStatus);
            result->setProposal(proposal);
        }
        return result;
    }

    if(acceptorStatus==ACCEPTED){
        //if the decision has been made, general update the proposal if only the id has been update.
        if(proposal->getId() > acceptedProposal->getId() && proposal->getAttackTime() == acceptedProposal->getAttackTime()){
            acceptedProposal->setId(proposal->getId());
            result->setAccepted(true);
            result->setStatus(acceptorStatus);
            result->setProposal(acceptedProposal);
        }else{
            result->setAccepted(false);
            result->setStatus(acceptorStatus);
            result->setProposal(acceptedProposal);
        }
        return result;
    }
    return nullptr;
}
```

五、 运行结果

从以下结果可以看出，最终所有的参谋达成了一致，约定明早 2 点发起进攻，实现了一致性。



六、 遇到的问题

在实现的过程中发现，当参谋第二阶段不被接受时，需要学习其他被接受的作战时间，之后返回第一阶段再次提交。此时无法利用加锁来保证只有一个参谋可以进行 prepareToCommit 操作，通过查阅资料，发现 java 实现的算法中使用 synchronized 关键字修饰函数即可解决该问题，但 c++却无法解决。

经过测试，该 c++代码实现仍然可以保证所有参谋的作战时间的一致性。实现过程中可能还有其他 bug 没有发现，仍然需要完善。

七、 思考与总结

现在，大数据和云对我们的生活产生了显著的影响。云计算的发展使得分布式系统一致性的问题越来越重要。在学习的过程中，我对线程间通信理解更加深刻，多个线程达成数据一致性即可保障各种安全问题。hadoop 分布式系统的实现中，也应该是在该分布式一致性原理的基础上进行设计维护数据的一致性。在实现的过程中，我们需要对算法的原理更加深入了解，不可以浅尝辄止，尝试书写一下代码便会发现许多细节问题，并可以进一步提高我们的编程能力。