华北计算技术研究所

研究生毕业论文开题报告

**综合业务开发平台代码一致性智能决策研究**

**Research on Code Consistency Intelligent Decision-making of Integrated Business Development Platform**

学生姓名： 肖韬睿

指导教师： 仇建伟

部 门： 系统一部

专 业： 计算机应用技术

2018年10月

目 录

[目 录 I](#_Toc527938075)

[1 选题背景和意义 1](#_Toc527938076)

[1.1 选题背景 1](#_Toc527938077)

[1.2 选题意义 1](#_Toc527938078)

[2 国内外发展动态 2](#_Toc527938079)

[3 研究目标 3](#_Toc527938080)

[4 研究内容 3](#_Toc527938081)

[4.1 分布式平台一致性问题研究 3](#_Toc527938082)

[4.2 一致性算法的选取规则研究 4](#_Toc527938083)

[4.3 代码一致性的智能决策研究 4](#_Toc527938084)

[5 关键技术 5](#_Toc527938085)

[5.1 分布式一致性理论 5](#_Toc527938086)

[5.1.1 基本原则与理论 5](#_Toc527938087)

[5.1.2 两阶段提交（2PC） 5](#_Toc527938088)

[5.1.3 三阶段提交（3PC） 6](#_Toc527938089)

[5.2 Paxos算法 7](#_Toc527938090)

[5.3 Raft算法 8](#_Toc527938091)

[5.4 Gossip算法 8](#_Toc527938092)

[6 论文实施计划 9](#_Toc527938093)

[7 参考文献 10](#_Toc527938094)

# 选题背景和意义

## 选题背景

当前各类综合业务信息系统需求瞬息万变，用户对软件的研制周期要求也越来越短，传统的开发模式已经很难快速响应用户使用需求。为了解决上述问题，部门项目组以综合业务信息系统快速构建为需求，为需求分析及软件设计开发等人员提供一套快速设计、高效构建的可视化开发平台——综合业务信息系统可视化开发平台。

目前，综合业务开发平台是在单机环境下运行的，为了满足今后的业务需求，现计划由单机环境扩展到多机分布式环境，这时就需要解决开发平台上的代码一致性问题。

## 选题意义

为了实现从单机环境扩展到多机环境的组织结构，必须进行开发平台的分布式环境建设，那么怎样做到平台中程序代码的同步呢？很明显，这就是一致性问题了，针对不同的应用场景，给出的解决办法是否也需要做出相应的改变呢，怎样做到同时兼顾不同的应用场景呢？从20世纪80年代起，对于一致性算法的研究就没有停止过，将一致性算法应用在不同的场景中，一直是分布式系统中的重要问题。

解决一致性问题需要一致性算法的支持。在过去十几年里，Paxos基本成为了分布式领域内一致性协议的代名词，Google的粗粒度锁服务Chubby的设计开发者Burrows曾经说过：“所有一致性协议本质上要么是Paxos要么是其变体”。基于Paxos协议的一致性算法，就是Paxos算法。受到Paxos算法的启发并且为了应对不同的应用场景，还滋生出了许多不同于Paxos的一致性算法。而且Paxos协议只是单纯解决多节点写入问题，且写入后不可更改，它的强一致性会导致系统的可用性降低，所以为了能将Paxos算法应用到分布式平台中也必须做出改进。

最后，为了维持不同应用场景下分布式开发平台的代码一致性，本文将使用多Agent技术实现在不同应用场景下一致性算法的实时选取切换，进行基于规则的代码一致性智能决策研究。

# 国内外发展动态

Paxos算法是莱斯利·兰伯特（Leslie Lamport）于1990年提出的一种基于消息传递的一致性算法[1-2]。Paxos算法解决的问题是一个分布式系统如何就某个值（决议）达成一致。一个典型的场景是，在一个分布式数据库系统中，如果各节点的初始状态一致，每个节点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。一个通用的一致性算法可以应用在许多场景中，是分布式计算中的重要问题。因此从20世纪80年代起对于一致性算法的研究就没有停止过。节点通信存在两种模型：共享内存（Shared memory）和消息传递（Messages passing）。Paxos 算法就是一种基于消息传递模型的一致性算法。

不仅仅是分布式系统中，凡是多个过程需要达成某种一致的场合都可以使用Paxos算法。一致性算法可以通过共享内存（需要锁）或者消息传递实现，Paxos 算法采用的是后者。Paxos算法适用的几种情况：一台机器中多个进程/线程达成数据一致；分布式文件系统或者分布式数据库中多客户端并发读写数据；分布式存储中多个副本响应读写请求的一致性。

虽然Paxos的理论提出已经20多年了，从第一个Paxos的工业实现到现在也已经12年了，但是直到近几年，无论是顶级会议，还是业内会议，与Paxos相关的文章和项目还是层出不穷。转向业内，真正工业级的、独立的Paxos基础库还是相当的少见：

（1）Google并没有开源任何Paxos基础库(连包含Paxos的项目都没有开源过)。

（2）Facebook也没有公布过包含Paxos的产品。

（3）Apache有Zookeeper，但是其协议并不能支持一个高吞吐的状态机复制，且并没有提供独立的第三方库，可供快速接入。

（4）在Github上，能找到的Paxos的独立库，star最高的是腾讯开源的phxpaxos。

因此到目前为止，业内还鲜有成熟可靠的，可供快速使用的独立第三方的Paxos库，开源的Paxos生态也尚未成熟。2006年Google发布了三篇论文，其中在Chubby锁服务[3]使用Paxos作为Chubby Cell中的一致性算法，Paxos的人气从此一路狂飙。Mysql版本5.7.17推出XCOM（MySQL Group Replication），借鉴了Galera的思想，在MySQL上构建了分布式强一致集群的插件。2016年，腾讯推出的基于Paxos协议的独立库phxpaxos，它和MySQL结合后推出了phxsql项目，也是基于MySQL实现的分布式强一致MySQL集群。2017年，阿里巴巴数据库团队面向高性能、全球部署以及阿里业务特征等需求，实现了一个高性能分布式强一致的Paxos独立基础库X-Paxos，X-Paxos为AliSQL X-Cluster解决了分布式一致性问题，同样可以快速赋予其他系统分布式一致性能力。而在现今最为火爆的IT领域——区块链中，传统的一致性算法PBFT（拜占庭容错）、Paxos、Raft也成为了首选。

# 研究目标

基于所内创新基金项目——综合业务信息系统可视化开发平台，利用多台服务器搭建分布式集群，完成开发平台的分布式设计，保证分布式平台的资源共享、用户透明性、高性价比、高可靠性、高灵活性的特点，实现分布式平台的搭建。为了解决诸如单台服务器宕机、负载均衡问题等一系列隐患，以一致性算法Paxos算法为基础，并针对不同应用场景，进行Paxos算法的改进。同时，也会使用较为成熟的开源一致性算法（如Raft算法）作为不同需求下的解决方案，与改进的Paxos算法进行功能上的互补，做到应对不同场景的一致性算法实时切换，即代码一致性的智能决策。最后，做到综合业务开发平台代码一致性能有可靠的保障。

# 研究内容

## 分布式平台一致性问题研究

什么是一致性[4]？一致性是一个抽象的、具有多重含义的计算机术语，在不同应用场景下，有不同的定义和含义。在传统的IT时代，一致性通常指强一致性，强一致性也被称为原子一致性或线性一致性。他有两个要求：任何一次读都能读到某个数据的最近一次写的数据；系统中的所有进程，看到的操作顺序，都和全局时钟下的顺序一致。简言之，在任意时刻，所有节点中的数据是一样的，例如，对于关系型数据库，要求更新过的数据能被后续的访问都可见，这是强一致性。数据更新后，如果能容忍后续的访问只能访问到部分，则是弱一致性。现在主流的一致性协议一般都选择的是弱一致性的特殊版本——最终一致性。最终一致性是指系统中所有的数据副本，在经过一段时间同步后，最终能够达到一个一致的状态，不需要实时保证数据强一致。

一致性算法是用来解决一致性问题的，那么什么是一致性问题呢[5]？在分布式系统中，一致性问题是指对于一组服务器，给定一组操作，需要一个协议使得最后它们的结果达成一致。更详细的解释就是，当其中某个服务器收到客户端的一组指令时，它必须与其它服务器交流以保证所有的服务器都是以同样的顺序收到同样的指令，这样的话所有的服务器会产生一致的结果，看起来就像是一台机器一样。

## 一致性算法的选取规则研究

在分布式平台搭建成功的前提下，下面给出几点应用场景的描述并针对他们制定一致性算法的选取规则：

1. 将改进的Paxos算法作为综合业务开发平台的基本一致性算法，在通常情况下基于Paxos算法实现分布式平台的一致性。
2. 若半数以上服务器宕机，使得Paxos算法的少数服从多数的模式失效时，将改为选用Raft算法，切换为主从模式实现一致性。
3. 若发现服务器有发生动态负载均衡问题的隐患时，将改为选用Gossip算法，切换为流言传递模式实现一致性。
4. 在上述问题消除后，将切换回Paxos算法，以减少资源的浪费。

## 代码一致性的智能决策研究

在开发平台中，程序代码通常与数据息息相关。代码经过编译产生程序，它在开发平台中往往是经常变动的；而数据通常存储在数据库中，不会经常变动。本文讨论的代码一致性实际上是动态的一致性问题，不同于分布式数据库，动态的代码难以保持强一致性，为了保证系统的可用可靠，只能选择达到最终一致性。文中对综合业务信息系统可视化开发平台与改进的Paxos算法以及现有开源的一致性算法进行结合，并能在不同场景下实时切换，并通过多Agent技术实现[6-9]。

Agent是一个具有自适应性和智能性的软件实体，能代表用户或其它程序，以主动服务的方式完成一项工作。多Agent系统是多个Agent组成的集合，它的目标是将大而复杂的系统建设成小的、彼此互相通信和协调的，易于管理的系统。采用多Agent技术将多个决策方法有机地协调起来，可建立基于多Agent协调的环境决策支持系统。Agent采用基于规则的描述方法，可实现环境管理的分布式智能决策。在各异的一致性算法中进行符合规则的实时自主切换，这就是代码一致性的智能决策。

# 关键技术

## 分布式一致性理论

### 基本原则与理论

1. CAP（Consistency一致性，Availability可用性，Partition tolerance分区容错性）理论是当前分布式系统公认的理论，亦即一个分布式系统不可能同时满足这三个特性，只能三求其二。对于分布式系统，P是基本要求，如果没有P就不是分布式系统了，所以一般都是在满足P的情况下，在C和A之间寻求平衡。
2. ACID（Atomicity原子性，Consistency一致性，Isolation隔离性，Durability持久性）是事务的特点，具有强一致性，一般用于单机事务，分布式事务若采用这个原则会丧失一定的可用性，属于CP系统。
3. BASE（Basically Availabe基本可用，Soft state软状态，Eventually consistency最终一致性）理论是对大规模的互联网分布式系统实践的总结，用弱一致性来换取可用性，不同于ACID，属于AP系统。

### 两阶段提交（2PC）

2 Phase Commit，两阶段提交[10]，系统有两个角色协调者和参与者，事务提交过程分为两阶段：

（1）提交事务请求（投票阶段）

* 协调者向参与者发送事务内容，询问是否可以执行事务提交操作，等待响应。
* 参与者执行事务操作，并将undo和redo日志记录。
* 参与者回复协调者，执行成功则回Yes否则No。

（2）执行事务提交（执行阶段）

* 如果都是参与者都回复Yes，则协调者向参与者发送提交请求，否则发送回滚请求。
* 参与者根据协调者的请求执行事务提交或回滚，并向协调者发送Ack消息。
* 协调者收到所有的Ack消息过后判断事务的完成或者中断。

2PC协议用于保证属于多个数据分片上的操作的原子性。这些数据分片可能分布在不同的服务器上，2PC协议保证多台服务器上的操作要么全部成功，要么全部失败。

优点：原理简单、实现方便。

缺点：同步阻塞、单点问题、数据不一致、太过保守。

### 三阶段提交（3PC）

3 Phase Commit，三阶段提交[11]，是二阶段提交的改进，系统也有两个角色协调者和参与者，事务提交过程分为三阶段：

（1）事务询问（canCommit）

* 协调者向参与者发送一个包含事务内容的询问请求，询问是否可以执行事务并等待。
* 参与者根据自己状态判断并回复yes、no。

（2）执行事务预提交（preCommit）

* 若协调者收到全是yes，就发送preCommit请求否则发布abort请求。
* 参与者若收到preCommit则执行事务操作并记录undo和redo然后发送Ack，若收到abort或者超时则中断事务。

（3）执行事务提交（doCommit）

* 协调者收到所有的Ack则发送doCommit请求，若收到了No或者超时则发送abort请求。
* 参与者收到doCommit就执行提交并发送Ack，否则执行回滚并发送Ack。
* 协调者收到Ack判断是完成事务还是中断事务。

三阶段相对于两阶段的改善就是把准备阶段一分为二，亦即多了一个canCommit阶段，按我理解这样就类似于TCP的三步握手，多了一次确认，增大了事务执行成功的概率。而且3PC的协调者即使出了故障，参与者也能继续执行事务所以解决了2PC的阻塞问题，但是也可能因此导致集群数据不一致。

优点：降低参与者阻塞范围，并能够在出现单点故障后继续达成一致

缺点：引入preCommit阶段，在这个阶段如果出现网络分区，协调者无法与参与者正常通信，参与者依然会进行事务提交，造成数据不一致。

## Paxos算法

上面两个协议的协调者都需要人为设置而无法自动生成，是不完整的分布式协议，而Paxos就是一个真正的完整的分布式算法[12-14]。系统一共有几个角色：Proposer（提出提案）、Acceptor（参与决策）、Learner（不参与提案，只负责接收已确定的提案，一般用于提高集群对外提供读服务的能力），实践中一个节点可以同时充当多个角色。提案选定过程也大概分为2阶段：

（1）Prepare阶段

* Proposer选择一个提案编号M，向Acceptor某个超过半数的子集成员发送该编号的Prepare请求。
* Acceptor收到M编号的请求时，若M大于该Acceptor已经响应的所有Prepare请求的编号中的最大编号N，那么他就将N反馈给Proposer，同时承诺不会再批准任何编号小于M的提案。

（2）Accept阶段

* 如果Proposer收到超过半数的Acceptor对于M的prepare请求的响应，就发送一个针对[M,V]提案的Accept请求给Acceptor，其中V是收到的响应编号中编号的最大的提案值，如果响应中不包括任何提案值，那么他就是任意值。
* Acceptor收到这个针对[M,V]的Accept请求只要改Acceptor尚未对大于M编号的提案做出过响应，他就通过这个提案。

（3）Learn阶段（本阶段不属于选定提案的过程）

* Proposer将通过的提案同步到所有的Learner。

特点：Paxos协议的容错性很好，只要有超过半数的节点可用，整个集群就可以自己进行Leader选举，也可以对外服务，通常用来保证一份数据的多个副本之间的一致性，适用于构建一个分布式的一致性状态机。

Paxos算法在实际实现时还需要考虑很多问题，比如如何保证不同进程所采纳的倡议编号全局唯一且递增增长、异常处理（比如倡议者或者接受者崩溃的处理策略）、状态持久化等，所以要使用该算法就必须进行改进[15]。

## Raft算法

Raft协议相比Paxos，容错性和性能都是一致的，但是Raft比Paxos更易理解和实施[16-18]。系统分为几种角色： Leader（发出提案）、Follower（参与决策）、Candidate（Leader选举中的临时角色）。

刚开始所有节点都是Follower状态，然后进行Leader选举。成功后Leader接受所有客户端的请求，然后把日志entry发送给所有Follower，当收到过半的节点的回复（而不是全部节点）时就给客户端返回成功并把commitIndex设置为该entry的index，所以是满足最终一致性的。

Leader同时还会周期性地发送心跳给所有的Follower（会通过心跳同步提交的序号commitIndex），Follower收到后就保持Follower状态（并应用commitIndex及其之前对应的日志entry），如果Follower等待心跳超时了，则开始新的Leader选举：首先把当前term计数加1，自己成为Candidate，然后给自己投票并向其它结点发投票请求。直到以下三种情况：

* 它赢得选举
* 另一个节点成为Leader
* 一段时间没有节点成为Leader

在选举期间，Candidate可能收到来自其它自称为Leader的写请求，如果该Leader的term不小于Candidate的当前term，那么Candidate承认它是一个合法的Leader并回到Follower状态，否则拒绝请求。

如果出现两个Candidate得票一样多，则它们都无法获取超过半数投票，这种情况会持续到超时，然后进行新一轮的选举，这时同时的概率就很低了，那么首先发出投票请求的的Candidate就会得到大多数同意，成为Leader。

特点：Raft协议对Leader的选举做了详细的设计，从而保证不会有多个Leader同时存在。另外，Raft协议把Leader选举、日志复制、安全性等功能分离并模块化，使其更易理解和工程实现。

## Gossip算法

Gossip协议与上述所有协议最大的区别就是它是去中心化的，上面所有的协议都有一个类似于Leader的角色来统筹安排事务的响应、提交与中断，但是Gossip协议中就没有Leader，每个节点都是平等的[19-21]。

每个节点存放了一个key,value,version构成的列表，每隔一定的时间，节点都会主动挑选一个在线节点进行下面的过程（不在线的也会挑一个尝试）：

* 节点1发送key/version。
* 节点2比较所有的key对应的version，并返回version更高的value以及版本更低的key。
* 节点1更新本地的数据，发送上面过程中接收到的key、version以及value。

两个节点各自修改自己较为落后的数据，最终数据达成一致并且都较新。节点加入或退出都很容易。

特点：去中心化，节点之间完全对等，不需要任何的中心节点。Gossip算法一般用于各种最终一致性的领域：失败检测、路由同步、Pub/Sub、动态负载均衡。但Gossip算法的缺点也很明显，冗余通信会对网路带宽、CPU资源造成很大的负载，而这些负载又受限于通信频率，该频率又影响着算法收敛的速度。

# 论文实施计划

毕业设计任务时间划分为五个阶段：

|  |  |
| --- | --- |
| 第一阶段  （2018.09.01-  2018.10.31） | 收集整理资料，对毕设题目进行确定，对整个课题研究进行初步的了解和掌握。对收集的资料进行分类、阅读。归纳相关文献。 |
| 第二阶段  （2018.11.01-  2019.05.31） | 对所收集的材料进行系统性的学习，充分认识相关研究课题的关键难点问题。完成分布式平台的搭建，并改进Paxos算法，发表小论文。 |
| 第三阶段  （2019.06.01-  2019.10.31） | 完成其他一致性算法的调研以及编译应用。 |
| 第四阶段  （2019.11.01-  2020.01.31） | 将分布式开发平台与各种一致性算法进行结合，完成分布式平台代码一致性的智能决策。 |
| 第五阶段  （2020.02.01-  2020.03.31） | 完成毕业设计和毕业论文。 |

# 参考文献

[1] Lamport, L., “Paxos Made Simple”, http://research.microsoft.com/users, 2001, 14 pages, XP-002358841.

[2] Leslie Lamport. The part-time parliament[J]. ACM Transactions on Computer Systems (TOCS), 1998, 16(2):133-169.

[3] Burrows M. The Chubby lock service for loosely-coupled distributed systems[C]// Usenix Symposium on Operating Systems Design and Implementation. 2006:335-350.

[4] Wei R, Beard R W. Distributed Consensus in Multi-vehicle Cooperative Control[M]. Springer London, 2008.

[5] Ongaro D, Ousterhout J. In search of an understandable consensus algorithm[C]// Usenix Conference on Usenix Technical Conference. USENIX Association, 2014:305-320.

[6] Wagenpfeil J, Trachte A, Hatanaka T, et al. Distributed decision making for task switching via a consensus-like algorithm[C]// Conference on American Control Conference. IEEE Press, 2009:5761-5766.

[7] Straì§Kowski T, Rybiński H. A Distributed Decision Rules Calculation Using Apriori Algorithm.[M]// Transactions on Rough Sets XI. Springer Berlin Heidelberg, 2010:161-176.

[8] 张雷. 多层次分布式智能决策支持系统及应用研究[D]. 西北工业大学, 2006.

[9] 胡文斌, 孟波. 基于多Agent的分布式智能群决策支持系统研究[J]. 计算机工程, 2006, 32(6):192-194.

[10] Al-Houmaily Y J, Samaras G. Two-Phase Commit[M]. Springer US, 2009.

[11] Al-Houmaily Y J, Samaras G. Three-Phase Commit[M]. Springer US, 2009.

[12] Chuang H U, Wen-Tao M A, Wang W J, et al. CC-Paxos:Integrating consistency and reliability in wide-area storage systems[J]. Computer Engineering & Design, 2017.

[13] Junqueira F, Mao Y, Marzullo K. Classic paxos vs. fast paxos: caveat emptor[C]// The Workshop on on Hot Topics in System Dependability. 2007.

[14] Álvaro García-Pérez, Gotsman A, Meshman Y, et al. Paxos Consensus, Deconstructed and Abstracted[C]// European Symposium on Programming. Springer, Cham, 2018:912-939.

[15] 张俊林. 大数据日知录:架构与算法[M]. 电子工业出版社, 2014.

[16] D. Ongaro , J. Ousterhout. In Search of an Understandable Consensus Algorithm[C].

[17] Paris J F, Long D D E. Pirogue, a lighter dynamic version of the Raft distributed consensus algorithm[C]// IEEE, International PERFORMANCE Computing and Communications Conference. IEEE Computer Society, 2015:1-8.

[18] Pacholczyk M. The Raft Consensus Algorithm[J].

[19] Renesse R V, Dan D, Gough V, et al. Efficient reconciliation and flow control for anti-entropy protocols[C]// The Workshop on Large-Scale Distributed Systems & MIDDLEWARE. ACM, 2008:1-7.

[20] 刘德辉, 尹刚, 王怀民,等. 分布环境下的Gossip算法综述[J]. 计算机科学, 2010, 37(11):24-28.

[21] Lu, Jie, Tang, et al. A gossip algorithm for convex consensus optimization over networks[J]. IEEE Transactions on Automatic Control, 2012, 56(12):2917-2923.