深度探索分布式理论经典论文

 duanple

关注他

1.序

通过对Google发表的论文进行梳理，我们了解到了当前分布式系统领域的一些最新热点和发展趋势。梳理下这些论文，我们会发现它们主要发表在OSDI、SOSP、SIGMOD、VLDB、Macro、Eurosys、SIGCOMM、CIDR、SIGARCH、SIGCOMM等顶级期刊和会议上。反过来通过关注这些会议和期刊，我们就可以持续跟踪该领域的最新进展。但是也会发现这些会议和期刊每个每年都会发表几十上百篇文章，让人应接不暇。同时如在第一篇文章所指出的，这些文章背后的理论基础却很少发生变化，基本上还是几十年前就已提出的。为了更好更快地理解这些层出不穷的新论文，理清其所依赖的理论基础显得尤为重要。正如前文所述，"如果要真正理解这些论文，除了论文本身内容之外，也还需要去了解传统的分布式系统和关系数据库理论"。

要了解这些理论有很多方法，最好的办法还是阅读提出该理论的原始论文。但是这些论文涉及到分布式领域的基本理论问题，其中又有很多伴随着复杂的算法和证明，因此阅读起来并不容易，很多都要比Google发表的那些论文还要难懂。要理解它们，需要不断反复地阅读，并结合一定的实践。虽然过程艰辛，但是通过这个过程可以让我们：1)开阔视野，对整个分布式的历史及发展有个更好的了解；2)阅读本身就像是与作者思维的碰撞与交流，通过了解问题的背景、思考过程和解决方法，能帮助我们像论文作者们一样思考分布式系统；3)通过实践联系理论，有助于进一步将实践总结深化升华；4)如果只看网上他人的笔记或理解，一方面不是原汁原味可能遗漏了某些重要的点，另一方面这些介绍也可能有偏差，而阅读原始论文则让我们可以更近距离地感受大师们的思考，同时可以避免这些问题；5)阅读这些理论论文可以为阅读其他论文打好基础，"会当凌绝顶，一览众山小"，把这些理论都看懂的话再阅读其他论文时会感觉很容易。

笔者从2011年开始阅读这些理论论文，之后基本上每篇都进行了翻译，并把对于其中难点的理解记录了下来，可以作为阅读这些论文的一个参考。关于整个的阅读初衷以及为什么选择翻译原文及对原文理解作注的方式，可以看这篇文章。同时面对如此众多的经典论文，其中又有很多是在几年之前看的，译文中理解偏颇、翻译错误之处在所难免，欢迎指正。所有相关译文可以从这里找到：论文列表，但是单纯看这个列表，难免会让人感觉眼花缭乱而不知该如何下手。因此我们在本文中对这些论文进行了分类整理，并着重讲述它们的背景、核心思想、重要性。

定的分布式系统实践经验，当然如果要真正理解它们还需要结合本文及每篇的译文进一步深入阅读原文。

2.分布式理论脉络

关于分布式理论的研究已经有数十年的历史，与关系数据库相比，我们可以发现该领域的知识相对分散，同时此前国内大学课程(尤其是本科课程)中也很少有专门讲述该领域知识的相关课程(国外课程:CMU-15-440，CMU-15-712，MIT-6.824，Standford-CS244b，UC Berkley-CS294-91)，很多人关于分布式理论的了解都是源于工作之后的一些学习。我们综合采用如下一些方法选择出了分布式理论方面最重要的一些论文：1.研究该领域大师的相关工作，比如Dijkstra、Lamport等人；2.参考分布式计算领域的重要奖项：Edsger W. Dijkstra奖(PODC最具影响力论文奖)；3.查看Google系列论文中的参考文献，选择出其中引用最多最为重要的那些；4.借鉴他人总结的一些论文列表。

2.1 大师

首先我们来了解下分布式领域的大师，看下他们所作的工作、研究动态、发表的重要论文、书籍，可以帮助我们构建起该领域的基础理论知识。对于计算机科学来说，首先要看图灵奖，获奖者基本都是该领域宗师级的人物。目前应该说只有Leslie Lamport是因为在分布式方面的贡献获得图灵奖。除此之后，还有几位图灵奖得主的研究领域也遍及了分布式，但是他们获得图灵奖更多是因为其他领域的贡献。

2.1.1 Lamport

Leslie Lamport，2013年图灵奖获得者，他从20世纪70年代起就对分布式计算领域做出了许多基础性贡献。他在微软的同事，1992年图灵奖得主Butler Lampson如此评价“Lamport对并发系统理论和实践在质量、范围和重要性上的贡献都是难以超越的。它们完全可以和Dijkstra、Hoare、Milner和Pneuli等所有前辈图灵奖得主的成就相提并论。虽然他能像这些前辈一样做好理论研究，但他最大的优点是作为一名应用数学家，十分了解如何利用数学工具来解决具有非凡现实意义的问题”。

他最重要的论文主要是如下几篇：“Time, Clocks, and the Ordering of Events in a Distributed System”,该论文获得了2000年度首届PODC最具影响力论文奖；“Reaching agreement in the presence of faults“，该论文获得2005年度PODC最具影响力论文奖；“The Byzantine Generals Problem”；“Distributed Snapshots: Determining Global States of a Distributed System”；“The Part-Time Parliament”，该论文中描述了解决分布式一致性的Paxos算法。

TLA+。同时他还是著名排版软件LaTex的作者，更详细的介绍参考：微软科学家Leslie Lamport荣获2013年图灵奖。

大家都知道Lamport还是比较特立独行的，很少有人能让他服气，而且很多人都被他无情鄙视过，但是有两位却是Lamport特别推崇的，他们是Djikstra和Lampson。下面我们来继续看下这两位，他们都是图灵奖得主，同时也都在分布式领域做出了重要贡献。

2.1.2 Dijkstra

Edsger Wybe Dijkstra(1930-2002)，荷兰计算机科学家，1972年图灵奖获得者。很多人对Djikstra的认识仅限于他的最短路径算法，实际上他的研究领域非常广(戳这里)，他在计算机科学的多个领域都做出了开创性贡献，像编译器、操作系统、分布式系统、程序设计、编程语言、程序验证、软件工程、图论等。具体如：独立发明了逆波兰式并用栈实现表达式的求值；提出了用于将中缀表达式转换为逆波兰式的shunting-yard算法；银行家算法；哲学家就餐问题、睡眠理发师问题；临界区、死锁、信号量；提出GOTO有害论，开启结构化编程运动，此前对人们来说程序只是写给计算机的最重要的是能run起来，此后人们意识到除了能run还需要别人也能看懂；1972年与C.A.R. Hoare 和Ole-Johan Dahl合著<<结构化编程>>一书；参与了第一个ALGOL60编译器的设计和实现，这是最早支持递归的编译器；设计实现了具有多任务支持的批处理系统THE，并首次提出操作系统的分层结构。从70年代开始，Dijkstra的研究兴趣开始转向形式化验证。这些贡献即使是放到整个计算机科学历史上来看，也足以让Dijkstra成为开门立派式的人物。

他在分布式领域有两篇重要论文，每篇都很短小但意义非凡。其中这篇"Solution of a Problem in Concurrent Programming Control"，虽然只有一页但是却开创了并发/分布式计算领域。另一篇”Self-stabilizing Systems in Spite of Distributed Control”只有两页，但是提出了” Self-stabilizing”这一重要概念，为以后的self-managing计算系统和容错系统提供了重要的理论基础，正是因为这一贡献使他获得了2002年的ACM PODC最具影响力论文奖。另外对比Lamport和Djkstra，会发现他们的科研经历有很多类似之处，都有很强的数学背景，Djkstra后来研究程序验证理论，Lamport后来也把他的主要精力放在了TLA+上。Djkstra早在1972年就获得了图灵奖，同时正是Djkstra关于并发的研究为Lamport打开了分布式相关研究的大门，我想对于Lamport来说Djkstra应该就像一个学术领路人般的存在。

最后简单地看如下几个数字，通过这几个数字也可以看出Dijkstra在计算机科学史上的重要地位：

1)1972年获得图灵奖，得奖时42岁。首届图灵奖是1966年，在他之前获奖的只有AlanJ.Perlis、MauriceV.Wilkes、RichardW.Hamming、MarvinL.Minsky、JamesH.Wilkinson、John McCarthy、

Commands, Nondeterminancy, and Formal Derivation of Programs、Go To Statement Considered Harmful、Programming Considered as a Human Activity。

3)ACM有图灵奖，IEEE有冯诺依曼奖，数据库有SIGMOD Edgar F. Codd Innovations Award，分布式计算有PODC Dijkstra奖。

2.1.3 Lampson

Butler W.Lampson，1992年图灵奖得主，1992年时任DEC公司高级研究员和主任设计师。1943年12月23日生于华盛顿。他曾在哈佛大学就读，学的是文科。1964年获得文学学士学位之后他进入加州大学伯克利分校研究生院，改修理工科，于1967年获得博士学位。留校任教4年以后，兰普森进入产业界，先后在施乐(Xerox)公司的Palo Alto研究中心(即著名的PARC)和DEC公司工作，1995年加盟微软，任软件总工程师至今。他在硬件、软件、程序设计语言、计算机应用、网络……诸多方面都有许多成果。硬件方面：在PARC时有以太网(Ethernet)，Alto计算机系统和Dorado系统。在DEC时，兰普森主持了用世界上最快的计算机芯片Alpha作中央处理器的A1pha工作站体系结构的设计。这是当前世界上最负盛名的64位工作站，是所谓第四代工作站中的典型和佼佼者。软件方面：前述SDS-940和Alto的操作系统。程序设计语言方面：LISP、Mesa、Euclid、SNOBOL等。应用方面：Bravo编辑器，Star办公系统。网络方面：Grapevine电子邮件系统，Dover网络打印机。更详细的介绍参考：计算机科学名人堂：Butler Wright Lampson。他在分布式领域的重要著作如下：

Distributed System--Architecture and Implementation：an Advanced Course，Springer，1981

How to Build a Highly Availability System using Consensus 1996

同时他和Jim Gray独立提出了用于解决分布式事务提交问题的两阶段提交。

2.1.4 Liskov

Barbara Liskov，2008年图灵奖获得者，美国第一个获得计算机科学博士学位的女性。主要贡献在程序设计语言和系统设计领域，在容错和分布式计算领域也做出了重要贡献。领导参与了很多影响深远的项目，像Venus分时操作系统，设计实现了CLU编程语言，在CLU中引入的很多特性被广泛应用在今天的面向对象编程中，著名的"Liskov替换原则"也是她提出的。

她1988年发表的论文“Viewstamped Replication: A New Primary Copy Method to Support

2.1.5 小结

如果要找出计算机科学史上最伟大的两位科学家，应该非A.M.Turing和John von Neumann莫属。如果要找出计算机科学史上最重要的两篇论文，那应该非A.M.Turing的“On Computable Numbers with an Application to the Entscheidungsproblem”(1936)和John von Neumann的"The First Draft Report on the EDVAC"(1945)莫属。

那么在分布式领域最重要的两位科学家是谁呢？毋庸置疑Lamport肯定是其中之一，而另外一个人我想应该是Dijkstra。从某种程度上来说，Dijkstra/Lamport之于分布式计算，就像图灵/冯诺依曼之于计算机科学。可以说正是Dijkstra开创了分布式计算这一领域，而Lamport则追随他的脚步在这一领域不断深耕。

2.2 Edsger W. Dijkstra Prize

了解该领域的重要奖项，可以帮助我们进一步了解该领域的大师们，因为这些奖项的获奖者基本都是在该领域某一方向上做出了关键贡献。

分布式领域有一个重要奖项：Edsger W. Dijkstra Prize，原来叫PODC最具影响力论文奖，始于2000年，2002年Dijkstra去世后，为了纪念他改名为Edsger W. Dijkstra奖(从这点也能看出Dijkstra对于分布式领域的影响)，截止2018年已经有20多篇论文获奖。该奖项每年选择一两篇对分布式计算理论和实践具有重要意义的经典论文，这些论文通常都经过了至少十年的检验。简单列下前三篇：“Time, Clocks, and the Ordering of Events in a Distributed System”，“Impossibility of Distributed Consensus with One Faulty Process”，“Self-stabilizing systems in spite of distributed control”。这个奖项为我们了解分布式理论提供了绝佳的参考，这些获奖论文本身就是分布式理论方面最重要的那些，同时可以按照获奖时间来排个序，基本上越早获奖说明越重要。

完整获奖列表见：podc.org/dijkstra/。

通过这个列表我们可以找到分布式领域其他一些重要的科学家，比如像Nacy Lynch("Distributed Algorithms"作者，证明了FLP结论、CAP定理，提出了部分同步模型)、Fred B. Schneider("Defining liveness"、"Implementing fault-tolerant services using the state machine approach: A tutorial"、"Fail-stop processors: An approach to designing fault-tolerant computing systems")。

在阅读Google论文时，容易被忽略的一个地方就是每篇论文的参考文献，如果我们把这些论文的参考文献整理一下，按照引用量排个序，也基本上可以找到一个分布式领域的重要论文列表。

以Chubby为例，参考文献中涉及的分布式理论就有：” Impossibility of distributed consensus with one faulty process “、” Leases: An ef- ficient fault-tolerant mechanism for distributed file cache consistency “、” The part-time parliament “、” Paxos made simple “、” How to build a highly available system using consensus “、” Viewstamped replication: A general primary copy method to support highly-available distributed systems “等。看过这些理论论文后，那么Chubby看起来就会更容易。

2.4 Reading List

下面是网上的一些分布式系统经典论文列表，可以发现国外很多大学的分布式课程都有对应的Reading List，同时很多课堂内容就是讨论这些论文，这点值得国内学校学习。

A Distributed Systems Reading List，中文。

A Distributed Systems Seminar Reading List

Rutgers CS 417 Reading List 、WISC CS739-fa14 Reading List、MIT Distributed Systems Reading Group Papers、Cornel CS6410

Readings in Distributed Systems(国外课程集合)

Awesome CS Courses

3.分布式理论介绍(经典论文导读)

纸上得来终觉浅，绝知此事要躬行

--陆游

根据第2节中描述的方法，我们整理出了分布式理论最重要的一系列论文，基本上涵盖了Lamport和Dijkstra的重要工作，以及阅读Google论文所必需的那些理论。下面我们就对这些经典理论论文进行一个分类及概要性的介绍。

3.1 起源(互斥执行(面包店算法)

于它开创了并发/分布式计算领域，首次提出并解决了互斥执行问题，提出了critical-section、活锁等概念，这些概念成为操作系统和分布式系统的重要基础。激发了很多人在该领域的研究，后来Knuth Lamport等人都针对该问题提出了自己的解决方法，尤其是对Lamport产生了重要影响，关于此文Lamport有如下评价：

“ That paper is probably why PODC exists; it certainly inspired most of my work. It remains to this day the most influential paper in the field. That it did not win a PODC Influential Paper Award reflects an artificial separation between concurrent and distributed algorithms–a separation that has never existed in Dijkstra’ s work. “

关于并发与分布式，Lamport是这样看待的"What I call concurrency has gone by many names, including parallel computing, concurrent programming, and multiprogramming. I regard distributed computing to be part of the more general topic of concurrency"。

2.A New Solution of Dijkstra’ s Concurrent Programming Problem(Lamport 1974)，译文。本文描述了用于解决互斥执行问题的“面包店算法” 。Lamport发明了很多并发算法，但是在他看来“面包店算法”就像是上帝的发明，他只是发现了它。与其他基于共享内存的同步算法一样，面包店算法也需要一个进程能够从内存中读取一个word，而这个word可能正被另一个进程进行写入(每个内存位置只有一个进程进行写入，因此不存在并发写入的情况)。与之前的算法甚至是在它之后的那些算法相比， “面包店算法”的特别之处在于在并发读写的情况下无论读出来什么样的value值它都可以正确工作。比如在写操作要把value值从0改成1时，并发的读可能获取了一个value值为7456，该算法依然可以工作。在最初设计该算法的时候，Lamport并没有想着要设计一个满足这一属性的算法。他也是在写下该算法的正确性证明后，才意识到该证明并不依赖于并发读写情况下读操作会读到什么样的值。

这篇论文对于Lamport自己来说也非常重要，他后来关于并发/分布式方面的很多贡献和研究都是起源于此。也正是从“面包店算法”开始，他提出了进程内部变量的概念—这些变量可以由多个进程读取，但是只能由一个进程写。此时他意识到这种算法可以很方便地进行分布式实现—某些变量属于某个进程，其他进程通过向owner发送消息进行读取即可。 “面包店算法”作为Lamport在并发/分布式领域发表的第一篇论文，可以算得上是他并发/分布式研究的起点。

3.2容错、故障处理

3.Self-stabilizing Systems in Spite of Distributed Control(Dijkstra 1974)，译文。本文发表于1974年的CACM，并获得2002年第三届PODC最具影响力论文奖。全文只有两页，却是分布式领域非常重要的一篇论文。该论文发表之后，一开始并没有引起关注，除了Lamport读了该文后

文是Dijkstra最非凡的工作之一，是容错领域里程碑式的工作，虽然该论文中从未直接提到“容错”和“可靠性”这样的名词。Lamport预测Self-stabilizing将会成为容错领域的重要概念，以及新的重要研究领域。实际证明Lamport是预测是对的，而Lamport关于该论文的评价也被认为是他最重要的贡献之一，正是因为他人们才认识到该论文提出的self-stabilization的重要性。

评注：本文虽然只有两页，但是并不容易读懂，当时看过第一遍后感觉虽然里面的单词你都认识，但是就是不知道它在说什么。

4.Why Do Computers Stop and What Can be Done About It?(Jim Gary 1985)，译文。本文获得了2011年的ACM SIGOPS Hall of Fame Award。早在互联网出现以前，Tandem Computers 就已经构建了具有高度容错性和可用性的系统。Tandem Computers 是最早从事容错服务器制造的厂商，它制造的机器广泛应用在银行证券等在线处理交易领域。本文即是Jim Gray在Tandem Computers工作期间所撰写的，文中揭示了Tandem Computers 的“NonStop”神话所依赖的那些重要技术：isolation、failing fast、transactional updates、process pairs、supervision。同时提出了容错领域的很多重要概念，诸如：Availability 、Reliability 、MTBF、MTTR。虽然这篇文章是写在1985年，距今已30多年，但是影响深远，其中的很多内容即使在今天看来依然非常有意义。而17年后，当Internet逐渐成为世界主流的时候，Berkley(作者之一是David A. Patterson)又与时俱进地写了一篇why do Internet services fail, and what can be done about it?，光从名字上来看就是对Jim Gray这篇经典论文的一种致敬。

3.3 Time Clock、HappenBefore、分布式快照

5.Time Clocks and the Ordering of Events in a Distributed System(Lamport 1978)，译文。该论文于1978年7月发表在”Communication of ACM”上，并于2000年获得了首届PODC最具影响力论文奖，于2007年获得了ACM SIGOPS Hall of Fame Award。本文包含了两个重要的想法，每个都成为了主导分布式计算领域研究十多年甚至更长时间的重要课题。

1)关于分布式系统中事件发生的先后关系(又称为clock condition)的精确定义和用来对分布式系统中的事件时序进行定义和确定的框架。用于实现clock condition的最简单方式，就是由Lamport在本文中提出的”logical clocks”，这一概念在该领域产生了深远的影响，这也是该论文被引用地如此之多的原因。同时它也开启了人们关于vector 和 matrix clock 、consistent cuts概念(解决了如何定义分布式系统中的状态这一问题)，stable and nonstable predicate detection，认识逻辑(比如用于描述分布式协议的一些知识，常识和定理)的语义基础等方面的研究。最后，最重要的是它非常早地指出了分布式系统与其他系统的本质不同，同时它也是第一篇给出了可以用来描述这些不同的数学理论基础(“happen before”relation)。

限是按照请求的先后顺序获取的。更重要的是，该论文还解释了如何将该协议用来作为管理replication的通用方法。从该方法还引出了如下问题：a)Byzantine agreement，那些用来保证所有的状态机即使在出错情况下也能够得到相同输入的协议。很多工作都是源于这个问题，包括fast protocols, impossibility results, failure model hierarchies等等。b)Byzantine clock synchronization 和ordered multicast protocols。这些协议是用来对并发请求进行排序并保证得到相同的排序结果，通过与agreement协议结合可以保证所有状态机都具有相同的状态。

评注：Lamport在多个场合指出，本文是他最具影响力的一篇文章(Paxos是工业应用最广泛的)。它影响了人们思考分布式系统的方法，直到今天对于人们理解分布式系统仍具有重要指导意义。如果要在本文中再选择一篇最经典的论文的话，我想应该非这篇莫属了，**其他都可以不看但是本文一定不要错过**。看过之后，你会发现本文中的**Physical Clocks**就是Google Spanner中的True Time的理论基础，而且是在Spanner论文发表30多年前就提出的。另外happen before为后来的线性一致性，Spanner中的外部一致性，并发程序的正确性验证等提供了重要基础。

6.Distributed Snapshots-Determining Global States of a Distributed System(Chandy & Lamport 1985)，译文。本文获得了2014年的PODC最具影响力论文奖，及2013年的ACM SIGOPS Hall of Fame Award。本文中的分布式snapshot算法，又称为Chandy-Lamport算法。分布式snapshot的难点在于在没有全局时钟及全局观察者的情况下，各个节点各自异步记录的状态很可能是不一致的。本文描述了一种用于记录异步分布式系统的一致性全局状态的算法。分布式snapshot对于解决分布式系统中的一些基本问题具有重要意义，比如它可以用来进行死锁检测、系统终止检测或者是用来验证某些全局性的stable property是否成立。同时Chandy和Lamport提出的算法也非常优雅和简单，这也是它之所以这么成功的原因。

Chandy-Lamport算法流程实际上很简单，而本文真正的难点在于如何证明通过这个算法记录的全局状态是有效的，因为通过该算法记录的这个状态可能在系统实际的执行过程中根本就没有出现。本文第一次清晰定义了在分布式系统中何谓一致性的全局状态，一致性的全局状态是我们观察异步分布式系统的重要基石。后续提出的很多重要概念比如vector clock、执行过程的同构性、全局性的断言检测、concurrent common knowledge等都是基于一致性的全局状态。

评注：本文读起来也很有难度，原文中并没有提到它与vector clock(因为当时还没有vector clock)的关系。但是此后在1988年独立提出vector clock的两篇论文中Virtual Time and Global States of Distributed Systems、Timestamps in Message-Passing Systems That Preserve the Partial Ordering，都提到了consistent cuts的概念，而Chandy-Lamport算法实际上得到的就是一个consistent cut。而本文与HappenBefore实际上也有联系，这也是我们把这两篇文章放到一块的原因。同时对于如何定义分布式的全局状态，其中采用的思考方法非常巧妙。

Consensus指分布式系统中的节点如何就某些东西达成一致，可能是一个值，一系列动作，也可能是一个决定。通常有如下应用场景：1. 决定是否将一个事务提交到数据库 2.对当前时间的界定达成一致以实现同步化时钟 3.执行分布式算法的下一步 4.选择一个leader节点。Consensus一直是分布式系统理论研究的核心问题：1)它可以用来刻化不同强度的系统模型之间的差异，根据FLP结论，对于同步系统来说，它是可解的，但是对于异步系统，即使只有一个单元出错，它也是不可解的；2)实际分布式系统构建依赖于它，比如Group membership systems、fault-tolerant replicated state machines、data stores –这些典型的分布式系统都在某种程度上依赖于它。

关于Consensus的论文和研究也非常多，整个关于Cosensus、2PC和事务的历史，可以参考这篇文章的介绍：A brief history of Consensus, 2PC and Transaction(译)。

3.4.1 拜占庭将军问题

7.The Byzantine General Problem(Lamport etc 1982)，译文。该论文的内容实际上已发表在1980年的"Reaching Agreement in the Presence of Faults"(获得2005年PODC最具影响力论文奖)中，本文只是引入了拜占庭将军这一说法，同时增加了一些新的内容，之所以引入拜占庭将军这一叫法，是受Dijkstra的哲学家就餐问题影响。在拜占庭将军问题中，进程可能会说谎而且它们还会尽力地去欺骗其他进程。这个问题看起来比FLP更难，但是它确实存在一个解，只是这个解的开销非常大，大到很难实用。后来Liskov在论文“Practical Byzantine Fault Tolerance”中对其进行了改进，以使其可以实用。

评注：普通的Consensus中，假设进程都是可信的，不会说谎不会篡改信息。而在拜占庭将军问题则假设存在不可信的进程，这些进程可能会说谎扰乱其他进程，针对这种情况提出解决方案。对于一个公司内部的分布式系统来说，通常都会假设进程是可信的、非恶意的。但是如果整个系统处于一个非可信的环境中，比如近来大火的区块链，用户是散布于互联网的各个角落，很多是恶意的，这一场景刚好是拜占庭将军问题所要解决的。因此我们看到在论文发表的30多年后，又重新焕发了生命力，也再一次让我们看到Lamport思想和研究内容的超前性。

3.4.2 FLP

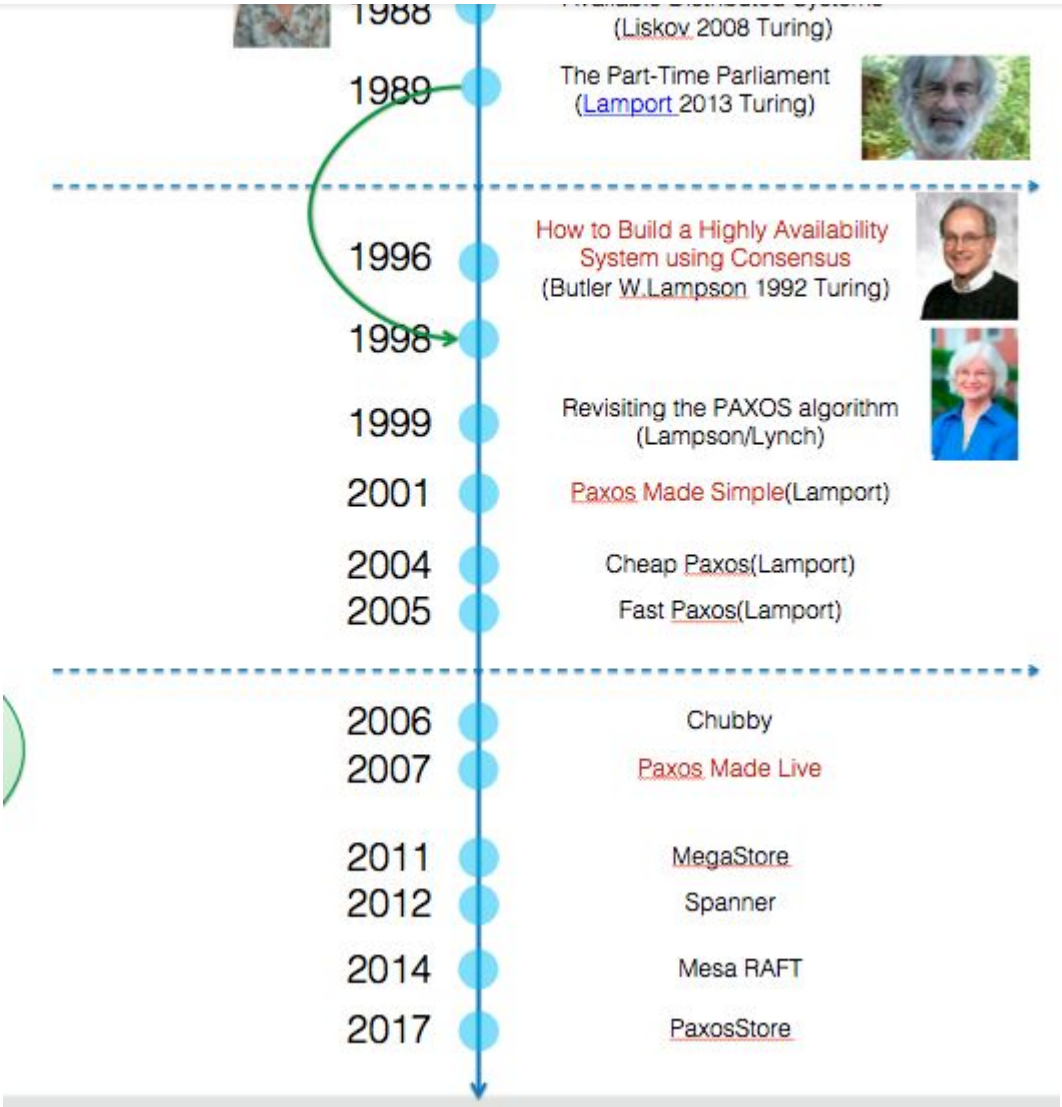
8.Impossibility of Distributed Consensus with One Faulty Process(Michael J. Fischer , Nancy A. Lynch , Michael S. Paterson 1983)，译文。这篇论文虽然只有短短的6页不到，但却包含了一个分布式系统领域最重要的结论。同时因为该结论的重要性和影响力，该论文获得了2001年第二届PODC最具影响力论文奖。这个著名的结论被称为FLP结论或者FLP不可能性，”FLP”即该论文的三位作者Fischer Lynch Paterson的首字母。本文证明的结论是：在多个进程组

该结论对后来的分布式理论研究产生了深远影响，终止了人们对于异步系统下完美一致性算法的探寻。这促使后来的研究者们尝试通过修改这些假设或者是修改问题描述来打破FLP结论，比如Consensus in the presence of partial synchrony(2007 Dijkstra Prize)表明在一个部分同步系统中就是可解的，再比如Unreliable Failure Detectors for Reliable Distributed Systems(2010 Dijkstra Prize)表明在具有一定错误检测能力的情况下该问题也是可解的，而 The Weakest Failure Detector for Solving Consensus(2010 Dijkstra Prize)甚至给出了要解决Consensus所需要的错误检测能力的下界。需要注意FLP结论成立的前提1)假设处理过程是完全异步的，也就是说对于进程的处理速率以及消息传输延时没有任何假设2)假设进程无法访问同步时钟，因此就不能使用那些依赖于超时的算法。此外FLP结论证明方法被应用到其他不可能性证明中，需要提一下的是Lynch最初接触异步Consensus问题是在她1982年访问Xerox PARC时，Lampson向她提起了这个问题，促使她开始这一问题的研究。

评注：本文也不好理解，不过这个结论还是很重要的，可以简单记下结论但是需要注意其成立的前提。

3.4.3 Paxos

在众多分布式理论中，Paxos应该是最著名的一个了，而Paxos本身实际上也有着非常有趣的历史，其中的趣闻详见笔者于2012年写的一篇总结关于Paxos的历史，具体如下图所示我们可以把Paxos的历史分为三个阶段(理论创立/成为理论研究热点/在工业界广泛应用)，而这个过程中Lamport、Liskov、Lampson、Lynch又一次成为了其中的主角。



在上述论文中，Lamport的The Part Time Parliament(1989)和Liskov的Viewstamped Replication: A New Primary Copy Method to Support Highly-Available Distributed Systems(1988)这两篇最初提出Paxos的论文共同获得了2012年的ACM SIGOPS Hall of Fame Award。同时在这些论文里面，图中标红的3篇比较适合也更值得阅读，具体如下：

9.How to Build a Highly Availability System using Consensus(Lampson 1996)，译文。
关于这篇文章，它最重要的意义在于使得Paxos算法为理论研究领域的科学家们所重视，并最终使得Leslie Lamport那篇“The Part-Time Parliament”从故纸堆里重见天日。通过这篇文章可以看到图灵奖得主Lampson是如何理解Paxos的。

这篇文章写在1996年，在此之前基本没人看懂Leslie Lamport那篇“The Part-Time

一开始被拒，没有人重视)，Butler W.Lampson是一个例外，他立刻意识到这个算法的重要性，并在他的演讲和一篇论文(即本文)中对该算法进行了描述，这引起了Nancy Lynch(分布式理论研究大牛，Distributed Algorithms一书作者)的关注”。此后的1998年，Lynch和Lampson还合写了一篇文章“Revisiting the Paxos algorithm”，发表在1999年的Theoretical Computer Science上，从那个时候开始Paxos才逐渐引起理论科学家们的关注，而真正为大众熟知应该是在Google发表Chubby之后了。

10.Paxos Made Simple(Lamport 2001)，译文。PODC2001会议上，Lamport听到人们在抱怨paxos算法是那么的难以理解。人们总是被那些古希腊的名称弄得晕头转向，而使得他们觉得论文难以理解，实际上算法本身是很简单的。于是在会议期间他找了几个人聚在一起，试着直接向他们口头解释该算法。回家之后将这些内容整理了下来，后来又基于Fred Schneider 和 Butler Lampson的建议做了修改。就形成了本文这个版本，虽然已经有13页长了，但是其中仍未包含任何一个比n1>n2更复杂的公式。

评注：与最初的那篇The Part Time Parliament相比，这篇Paxos Made Simple的确是比较Simple了，但是本文只是从数学上解释证明了Paxos，仍未解决怎么将Paxos应用于实践的问题，从理论到实现的细节并未介绍很多。为了解决实现的细节问题，Diego Ongaro和John Ousterhout后来提出了RAFT。

11.Paxos Made Live - An Engineering Perspective(Google 2007)，译文。Google在它的分布式系统中，大量使用了Paxos，比如Chubby、MegaStore、Spanner、Mesa等系统中。这篇文章详细讲述了Google在最初实现Paxos碰到的一系列问题及解决方案，是一篇全面讲解分布式系统工程实践的文章。其中提到的很多真实发生的场景，相信做过分布式系统的人都会感同身受，每个人或多或少都会遇到过一些类似的问题。与理论界的文章相比，这篇文章显得非常实际，所有内容来自Google一线工程师的实战，非常值得一看。重要的不是如何实现Paxos(实际上大多数人都不需要实现Paxos)，而是应理解如何将理论应用到实践，如何弥补理论与实践的差异。

3.4.4 分布式事务提交

并发控制与恢复机制虽然源于数据库，今天实际上已被广泛应用各类其他系统中。介绍并发控制与恢复的经典书籍“Concurrency Control and Recovery in Database Systems”，里面包含了关于可串行化、2PL、MVCC、Logging、Checkpointing、2PC、3PC的经典论述。虽然这是一本书，但是在学术界的引用量却异常的高，达到了5000+。本书第一作者Philip Bernstein，是第三届SIGMOD Edgar F. Codd Innovations Award得主，在他之前的两位是Michael Stonebraker、Jim Gray。

情况，某些节点正常工作但是其他一些节点或者网络出错了。如前所述，分布式事务的Commit或Abort操作必须要保证在事务的数据访问涉及到的所有节点上执行，在允许局部失败的情况下，这个问题变得非常复杂。我们将可以保证这种一致性的算法称为原子性提交协议(ACP-atomic commitment protocol)。目前已经证明：1)如果可能发生通信故障或者完全故障，所有的ACPs都可能会导致进程被阻塞 2)没有一种ACP可以保证故障进程的独立可恢复性。

12. Two Phase Commit(Lampson 1976/Jim Gary 1979)，译文。两阶段提交协议是最简单最流行的ACP，由Jim Gray和Lampson分别独立提出。Lampson在Crash Recovery in a Distributed System(获得2010 SIGOPS Hall of Fame Award)、Jim Gray在Notes on Database Operating Systems (1978)中分别描述了两阶段提交(2PC)。其正常流程非常简单，网上可见的大部分文章主要都在讲正常流程，但是关于异常情况及处理却很少见到。本书针对各种可能的异常情况及处理方式进行了详细描述，可以帮助我们深刻理解分布式系统中的异常及处理。

可惜的是两阶段提交是Blocking的。在进程投了Yes之后到它获取足够的信息来确定最终决定是啥之前，这中间的这个时间段称为进程的**不确定区间(uncertainty period)**，当进程处在这个时间段时，我们就说进程是不确定的(uncertain)。协调者没有不确定区间，只要协调者活着并且其他人都可以与它通信2PC就不会被阻塞，但是如果协调者挂了，整个过程就可能被阻塞。处于**不确定区间**的进程既不知道最终决定是要Commit还是Abort，也不能单方面地决定Abort。如果进程在**不确定区间**超时，同时可以进行通信的那些进程本身也是不确定的，那么进程就会被阻塞。举例来说，比如现在所有进程都投了Yes，然后协调者在向参与者发送事务提交结果之前挂了，同时其中一个参与者也挂了，这些剩余的参与者就处于**不确定区间**，它们是不知道事务到底是commit还是abort的，而且它们也不能重新选个协调者决定，因为如果要commit，挂掉的那个参与者可能投了no，如果要abort，挂掉的那个参与者可能已经接收到了协调者的commit并进行了本地处理。它们必须要block住，直到协调者恢复。

13. “NonBlocking Commit Protocols” (Dale Skeen 1981)。文中指出对于一个分布式系统，需要3阶段的提交算法来避免2PC中的阻塞问题。根据“Concurrency Control and Recovery in Database Systems”中关于3PC的描述，其核心思想是维护这样一个**不变性**：只要活着的进程中有人是uncertainty, 任意进程(包括失败的进程)都不会commit。实现方法：增加一个Precommit阶段，由两阶段变为三阶段，确认所有人走出uncentainty阶段，然后再commit。

书中提到了两种3PC协议，第一种基本3PC协议只保证了节点fail-stop情况下(同时假设没有网络问题)的正确性，在该协议中如果某个进程停止运行了，那么活着的进程通信一下：如果某个进程是aborted(还没有投票或者投了no或者收到了abort)，那么就决定abort；如果某个进程是committed(收到了commit)，那么就决定commit；如果所有活着进程都汇报自己的状态是

uncentain的节点走出这个状态。可以想象一下，对于该协议来说如果uncentainty节点与commitable节点出现网络分区，比如uncentainty节点相互之间可以通讯，commitable节点之间也可以通讯，但是两票节点之间无法通讯，就会出现脑裂，那些commitable节点会认为要提交，那些uncentainty节点则认为要abort。

为了解决脑裂问题，提出了增强版3PC协议，核心思想是引入majority机制，只有当协调者可以与超过半数节点通讯时才能决定事务提交结果。这就保证了两个协调者之间必定至少有一个共同的参与者。如果可以通讯的节点数不超过半数，3PC也会Block住，与2PC相比，它只是降低了Block的概率，但是无法完全消除。虽然3PC已经提出很久了，但是由于其复杂性在实际系统中都还是采用2PC来实现事务提交。同时说到这里，我们也可以发现3PC与Paxos的关联，增强版3PC大概是在1982年提出的，推测Lamport在提出Paxos时应该也参考了3PC中的这些做法。

14. Consensus on Transaction Commit(Lamport & Jim Gray 2004)，译文。提到Paxos，人们会禁不住想到Lamport，提到事务，那当仁不让就是Jim Gray了，而本文就是由这两位所写的关于Paxos和事务提交的文章。分布式事务提交问题需要在事务是提交还是撤销上达成一致，在协调者不工作的时候，经典的两阶段提交协议会阻塞。容错的Consensus算法也是需要达成一致，但是只要有半数以上的进程正常工作就不会被阻塞。因此一个简单的解决方法就是提高协调者的可用性，比如我们让协调者采用多个副本，相互之间通过Paxos进行日志同步实现高可用。

而本文提出的Paxos Commit算法则在此基础上更近一步，Paxos提交算法会在每个参与者关于是提交还是撤销的决定上运行一个Paxos一致性算法，这样就得到了一个具有2F+1个协调者同时只要至少F+1个正常工作就不会被阻塞的事务提交协议。该算法将Paxos与两阶段提交紧密结合，使得它与两阶段提交相比具有相同的稳定性存储写延迟，同时在没有错误发生的情况下可以具有相同的消息延迟，但是需要使用更多的消息。经典的两阶段提交算法可以看做是Paxos提交在F=0时的一个特例。

3.4.5 总结

虽然我们把拜占庭将军、Paxos、分布式事务提交都放到了Consensus这个大类下。这三者有其相似的地方，但也有很多不同，采用的算法也是各不相同。很长一段时间内，研究拜占庭将军、Paxos的人们是一拨，而研究事务提交的则是另一拨。在1986年，关注一致性和事务的人们聚在了一起。关于这场会议JimGray写了一篇文章“A Comparison of the Byzantine Agreement Problem and the Transaction Commit Problem.” (1987)，文中指出“在这次会议之前，人们普遍认为分布式系统中的事务提交问题是拜占庭将军问题的一个退化版本。或许这次会议的最大意义在于指出二者很少有共同点”。

在一个value上达成一致，即使是那些出错的进程。一个事务当且仅当所有的RM都准备好提交时才会被提交。而普通的consensus问题只关注那些没有出错的进程可以达到一致。因此uniform consensus比普通的一致性问题要难。而Paxos Commit并没有使用Paxos算法来直接解决事务提交问题，它并不是用来解决uniform consensus，而是用来让系统容错。此后论文Revisiting the relationship between non-blocking atomic commitment and consensus(2005)对具有错误检测能力的异步系统中的consensus和非block提交问题进行了对比。

再具体地来比较下事务提交和Paxos，对于事务来说，参与者是个独立有主见的个体，每个参与者实际负责的是不同的数据集合(partition)，同时受事务完整性约束，在一次提交中有的参与者检查后可能发现自己必须要提no，如果事务提交了会违背完整性约束。因为必须要尊重每个参与者的投票结果，所以即使是那些已经挂掉的进程也不能忽略。但是对于Paxos来说，通常参与者组成的是一个多副本系统，挂掉的那个实际上是没有主见的，它只要恢复后能从其他节点同步日志即可，这就是两者的核心区别。同时在一个事务系统中，两个事务之间通常是没有关联的，它们的参与者集合也没有关系，而在一个RSM(Replicated State Machine)中，两个paxos实例之间成员通常是相同的，这也是事务比RSM复杂之处。

3.5 Consistency

Consistency是用来描述并发程序执行正确性的模型，是系统为用户提供的一种保证，用户通过这种保证去理解自己编写的并发程序的运行时行为。系统提供的保证越强，用户编写程序就越简单，但是系统的开销就越大，从这一点上来说与数据库中的隔离级别是类似的。

3.5.1 顺序一致性

15. How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs(Lamport 1979)，译文。此文提供了关于Sequential Consistency一个简单精确的定义。不过它主要讲述的内容是关于：处理器和内存模块在什么条件下可以保证多处理器并行执行情况下的顺序一致性。这篇文章是Lamport在1979年发表的，距今已经40年，虽然只有短短两页，但却是cache-coherence领域最常被引用的一篇早期论文。

3.5.2 线性一致性

16. Linearizability: A Correctness Condition for Concurrent Objects(Maurice Herlihy • Jeannette M Wing 1990)，线性一致性理论。此文首次提出了Linearizability的概念。主要内容包含线性一致性模型的形式化定义，该模型的两个关键属性(locality和nonblocking)及其证明，还包含与其他一些模型比如Sequential Consistency/Serializability(可串行化)的对比。

数据库（ACID，Transaction）做了较好的补充。

3.6 LSM-Tree

18. The Log-Structured Merge-Tree(Patrick O’Neil &Edward Cheng etc. 1996)，译文。随着NoSql系统尤其是类Bigtable系统的流行，LSM-Tree这个名词也开始变得不再陌生。相信大多数了解NoSql系统的人，基本上都会听到过LSM-Tree这个名词，但是读过其原始论文的人估计就不是很多了。LSM-Tree之于Bigtable的重要性就像一致性hash之于Dynamo，B-树之于关系数据库，虽然并不直接涉及分布式，但是鉴于其重要性还是把它单独列在了这里。LSM-Tree核心思想是对变更进行延迟及批量处理，将随机写转换成批量的顺序写，并通过一种类似于归并排序的方式联合使用一个基于内存的组件和一个或多个磁盘组件。

评注：这篇论文原文共30页，涉及了不少公式，而其中关于IO开销的数学分析方法特别值得学习。

3.7 Leases

19. Leases: An efficient fault-tolerant mechanism for distributed file cache consistency(Cary G.Gray and David R. Cheriton 1989)，译文。本文作为最早提出租约机制的论文，获得了2009年的ACM SIGOPS Hall of Fame Award。租约是一个很形象的叫法，虽然最初租约是被用于解决分布式文件缓存的一致性，随着时间的推移，该机制逐步被应用于更多的场景下，比如在GFS、Chubby、Megastore、Spanner中都可以看到它的应用，为深入理解这些系统，租约是一个重要基础。

3.8 经验法则(IO、网络、延迟、部署运维、系统设计、CAP)

3.8.1 五分钟法则

20. The 5 Minute Rule for Trading Memory for Disc Accesses and the 5 Byte Rule for Trading Memory for CPU Time(Jim Gray,Franco Putzolu. 1986)，译文。关于5分钟法则，想必很多人都听说过，而5字节法则则没有那么耳熟能详了，这两个法则是Jim Gray和Franco Putzolu 在1986年提出的。根据文章名称也可以看出，5分钟法则是用来衡量内存与磁盘的，而5字节法则则是在内存和CPU之间的权衡。在该论文发表10年后的1997年，Jim Gray和Goetz Graefe 又在The Five-Minute Rule Ten Years Later and Other Computer Storage Rules of Thumb中对该法则进行了重新的审视。

3.8.2 网络

21. Single-Message Communication(DAG BELSNES 1976)，译文。当通信系统需要传输大量短消息时，减少进程间连接(connection)的创建和销毁以及消息可靠性方面的控制开销是非常重要的。本文描述了几种不同的端到端控制流程，同时研究了它们是否会导致消息丢失及收到重复消息。结果表明(基于对通信网络的一定假设)所有的端到端协议要么会产生消息丢失，要么会导致重复。

此外网络通讯领域还有一个著名的不可解问题：Two Generals Problem，该问题最早是在SomeConstraints and Trade-offs In The Design of Network Communications中证明了其不可解。

3.8.3 延迟处理

22. The Tail at Scale(Jeff Dean, Luiz André Barroso 2013)，译文。Jeff Dean和Luiz André Barroso关于大规模在线服务中延迟处理的经验之谈。大规模在线服务需要在一堆不可预测的组件的基础上创建一个响应可预测的整体。本文列举了一些大规模在线系统中发生高延迟的常见起因，同时描述了一些降低它们的严重程度或者是减轻对系统整体性能影响的技术。

3.8.4 部署运维

23. On Designing and Deploying Internet-Scale Services(James Hamilton 2007)，译文。本文总结了用于设计和开发运维友好的服务的一系列最佳实践，每条都值得细细品味。本文作者James Hamilton目前是亚马逊AWS的VP和杰出工程师，专注于基础设施的效率、可靠性和可伸缩性。在AWS内部地位崇高，是亚马逊员工中为数不多地获准在博客上发表自己重大想法的人之一。

3.8.5 系统设计

End-To-End Arguments in System Design(J.H. Saltzer, D.P. Reed and D.D. Clark 1981)

Hints for Computer System Design(Lampson 1983)

The Design Philosophy of the DARPA Internet Protocols(David D. Clark 1988)

3.8.6.1 CAP

目前网上关于CAP的资料非常多，具体可以看这篇合集：关于CAP的文章。CAP猜想由Eric Brewer在98年提出，99年发表(Harvest, Yield and Scalable Tolerant Systems)，2000年登上PODC主题演讲(CAP keynote)，2002年Nancy Lynch和Seth Gilbert在Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services中对该问题进行了形式化定义和证明，把CAP猜想变成了CAP定理。此后伴随着NoSQL的火热开始被人们所重视，与此同时也伴随了很多争论，鉴于此到了2012年Eric Brewer又对其进行了重新回顾和更新：CAP理论十二年回顾："规则"变了。今天CAP仍然有着被过度使用的倾向，实际上无论从理论还是实践上来说，它产生的影响都没有看起来的那么大。从理论上来说，分布式领域有上百种不可能性结论(A Hundred Impossibility Proofs for Distributed Computing-Nancy Lynch 1989)，与这些结论相比CAP并不出色，与FLP对理论界的深远影响相比，它也没有催生新的研究。从实践上来说，分布式系统的设计实现本身就是多种因素的权衡，而CAP引发了很多误解，从某种程度上来说反而产生了一些副作用。

3.8.6.2 BASE

BASE: An Acid Alternative(Dan Pritchett@Ebay 2008)，译文zz。BASE，代表Basically Available, Soft state, Eventually consistent。提出BASE，就是为了对标ACID。在英语中，acid字面上的意思是”酸，酸性“的意思，而base则有”碱，碱性“的意思，从这个角度看base本身就是为了对标acid而搞出的一个缩写。从对理论和实践的影响上看，与ACID相比差距甚远。同时伴随着软硬件技术的发展，在ACID被各种分布式系统进一步提上日程的今天，它的影响也将逐步减弱。

4.结语

如果看到了这里，说明你很有耐心并且对分布式理论也很感兴趣，但是要理解本文背后的这些论文还需要更大的耐心，即使是有参考译文的情况下可能仍然需要花半年、一年甚至更多时间，对于笔者来说，此前经历的过程和花费的时间更加漫长。这些论文组成了分布式理论的枝干，但是每个枝干上又有很多分枝，笔者也在不断学习中，路漫漫其修远兮。最后借用王国维的读书三境界作为结语。

“古今之成大事业、大学问者，必经过三种之境界： ‘昨夜西风凋碧树，独上高楼，望尽天涯路’ 。此第一境也。 ‘衣带渐宽终不悔，为伊消得人憔悴。’ 此第二境也。 ‘众里寻他千百度，蓦然回首，那人却在灯火阑珊处’ 。此第三境也。”

1.Lamport语录

Which is it? Is it distributed or is it not distributed? Well, we've come to our first nonproblem: What is a distributed systeme. Distribution is in the eye of the beholder. To the user sitting at the keyboard, his IBM personal computer is a nondistributed system. To a flea crawling around on the circuit board, or to the engineer who designed it, it's very much a distributed system.

-- Problems, Unsolved Problems and Non- Problems in Concurrency

2.Lamport Turing Lecture: The Computer Science of Concurrency – The Early Years

3.Dijkstra互斥算法

```
int j;

Li0: b[i]= false;

Li1: if (k != i)

{

c[i] = true;

Li3: if (b[k])

{

k = i;

}

goto Li1;

}

Li4: else
```

```
for (j = 1; j <=n; ++j)
{
if (j != i && !c[j])
{
goto Li1;
}
}
}
critical section;
c[i] = true;
b[i] = true;
remainder of the cycle in which stopping is allowed;
goto Li0
```

参考文献

List of important publications in computer science

Great Papers in Computer Science

Leslie_Lamport

Edsger W. Dijkstra

List of important publications in concurrent, parallel, and distributed computing

GREAT PAPERS IN COMPUTER SCIENCE: A RETROSPECTIVE

book.mixu.net/distsys/a...

A Distributed Systems Reading List

courses.csail.mit.edu/6...

2002 PODC Influential Paper Award

SIGOPS The Hall of Fame Award

[/read]

发布于 2020-12-19

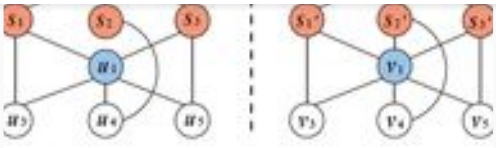
论文 学术 分布式系统

文章被以下专栏收录

分布式领域经典论文 关注专栏

我有故事 你有酒吗
分布式系统 设计、实现和思考 关注专栏

推荐阅读



王涵之 发表于Algor...

论文阅读 | 图匹配算法PPRGM

5 条评论

切换为时间排序

写下你的评论...

lambda 2020-12-23

这份总结真是太棒了

赞

王烨 2020-12-27

赞👍太牛了

赞

Jerry Z 2020-12-28

善

赞

问题不大 2020-12-29

这么好的回答，答主牛逼

赞

笛子不响 20 小时前

感谢作者