分布式算法讲义

(V0.3)

黄 宇

2022 年 8 月 5 日

目录

前言

以前试图写一个重量级的授课讲义，推进非常缓慢。现在尝试写一个轻量级的讲义，以主要文献的推荐和简评为主，推进一段时间，试试效果。

对学习分布式算法比较深入的学生，我经常会问他一个问题，你是把分布式算法，当一个数学对象来看，还是当一个软件对象来看。换一个更“实用”的角度来说就是，你学了分布式算法，是去搞理论计算机科学、数学，还是去搞分布式系统软件。

这是一个基本的定位问题。这本讲义，主要还是将分布式算法当一个软件对象来看，主要面向的是分布式系统的建设者，特别是有一定的实践经验，理论基础比较薄弱的建设者。里面有一些理论性的结果与讨论，也被看作分布式系统构建的further reading。

根据我与系统软件开发人员的接触，以及看他们在网络发布的技术内容，我发现有一部分分布式系统软件开发人员，基本概念不够清晰，相对严谨的建模与论证能力尚有不足。从我的角度而言，目前分布式算法课本是缺位的，一些翻译的国外经典教科书，对于分布式系统开发人员也是不太合适的。希望（未来）本书的内容能对他们有所帮助。

此外，在校的同学，从本科生的算法课一路学下来，学了分布式算法的基本概念，再去实际分布式系统的构建中去深入体会，也是可以的。每年招新学生的时候，分布式算法知识的入门是一个反复出现的任务。一般而言，分布式算法知识和大家的本科课程距离较远，难以自然地衔接。分布式算法的经典课本、论文在本科毕业的时候直接阅读往往比较困难。另外一方面，网上相关技术内容不少，但是质量堪忧，有误导性的“负作用”。为此陆续整理这一本讲义，希望能将每年重复给新学生罗列的学习内容沉淀下来，复用起来。

这本讲义完全是“on demand”式地逐步添加的，而且论述非常简要，更像是向学生讲解所需阅读的入门文献时，所做的注解。未来有一天，如果这本讲义的内容充实到一定程度，或许它能成为一本合格的《分布式算法入门》课本。

分布式算法的系统学习，对本科生《算法设计与分析》课程的学习有一定的要求。具体知识点的多少，是次要的，重要的是学习和思考的范式，两门课程是一以贯之的。进一步内容可以参考我的本科生算法设计与分析课程：《战疫时期的算法课（2020年春季）》[1]。

[1] www.bilibili.com/video/BV11341167sn/

第一部分

分布式系统建模

这一部分讲解计算模型的基本概念。

这一部分还讲解分布式算法的正确性规约，它脱胎于分布式系统设计的需求规约。

第一章　抽象计算模型的概念

1.1　基本模型

我们讨论抽象的分布式算法设计与分析，主要基于消息传递模型（MSG）和共享内存模型（SHM）。这里将主要讲解计算模型的基本组成部分。而模型的具体细节，将放到具体的章节（第三章、第九章）去讲。

我们首先讨论计算模型的两个基本的维度。一个维度是通信的载体，一个维度是时间模型。通信的载体主要包括消息传递（Message-passing，MSG）和共享内存（Shared Memory，SHM）。我们后续的章节也是按照这个维度来组织。时间模型主要包括异步模型（asynchronous model）和同步模型（synchronous model），也包括更深入的半同步模型（partially synchronous model）。

❝ 参考两本教材中系统模型相关章节[Aspnes, 2019, Attiya and Welch, 2004]。[Aspnes, 2019, Sec 2.1.1]，作者对两本书（[Aspnes19]和[Attiya04]）中，对于“消息传递系统”不同建模方式的“异”和“同”进行了比较，并回顾了相关的历史。理解不同建模背后共性的部分，是学习如何对分布式系统进行合理抽象，并进行后续的问题定义、算法设计、算法分析的基础。 ❞

1.2　建模进阶

建模中的一个重要概念是计算模型之间的模拟（simulation）。

❝ Simulation的内容可以参考[Attiya04]的Part II。作者Attiya的代表工作[Attiya et al., 1995]是在MSG模型上，模拟一个SHM的atomic register。因为其研究的原因，Attiya在写教材的时候，也比较偏重模拟技术的讲解与应用。例如对于异步环境中共识不可能性的结论，她就专门使用模拟技术来证明。而直接证明的方式其实更有名，也非常有学习意义，它就是著名的FLP的证明[Fischer et al., 1985]。 ❞

一个与“模型之间的模拟”有些类似，又有重要区别的概念是“问题之间的归约（reduction）”。在深入研究分布式计算的理论问题的时候，这两个技术是以既有结论为跳板，更“省劲”地证明新结论的利器。

其它还有一些高级的系统模型，可以作为主要模型的补充，对照着做一些初步的了解。

❝ 对于一些更高级模型的介绍，参考[Aspnes, 2019, Part III]。 ❞

1.3　编程抽象

将系统建模推到一个更深层次的做法是，以“编程抽象”为核心载体，去解构一个分布式系统。对于分布式系统的使用者而言，分布式系统就是不同层级的各种编程抽象；对于分布式系统的构建者而言，其核心任务就是用设计层的算法、协议和系统层的代码去实现不同的编程抽象。在编程抽象的视角下，分布式系统就是层层编程抽象的组合，一个再复杂的系统，都像是乐高搭成的，都可以解构为更基本的单元。

这一编程抽象的视角，对于系统学习分布式系统理论而言，是非常有帮助的。对于实际开发者而言，它的主要意义在于将核心的概念，层层解构，辨析清楚。

❝ [Cachin et al., 2011]这本书就是基于编程抽象，来组织分布式系统编程相关内容的。❞

第二章　分布式算法的基本性质

本科生算法课中讨论的正确性规约，是必要的基础。基于此，我们可以发现这一规约对于讨论分布式系统做的对不对，是不够的。

分布式系统中的正确性讨论，主要基于safety和liveness这两个核心概念，它们是串行程序中“partial correctness”和“complete correctness”概念的泛化。

❝ 一直以为这两个核心概念是Lamport首创，[Malkhi, 2019]的Introduction章节中也有提及。但是后面又看到这两个概念是在[Alpern and Schneider, 1985]中定义的（这一工作获得了2018年分布式计算理论领域的Dijkstra奖），Lamport应该是对于异步分布式算法，进一步深化了这两个概念。 ❞

上述基本性质之外，还要讨论几个更深入的性质。

公平性（fairness）。

模拟（simulation）。从外在使用者的角度，它是计算模型之间的转换。从模拟算法设计者的角度，它是一种性质或者说规约。

下界与不可能性（impossibility）。这不是单个算法的性质，它是一个问题的所有可能算法的整体性质。

第二部分

消息传递算法

这部分讲解MSG模型上的经典分布式算法。

第三章　MSG模型

第一章主要是介绍基本概念。MSG模型的技术细节，拓展性讨论将在此展开。
节点端in_buf和out_buf。
信道$channel_{ij}$。

第四章　领导者选举问题

领导者选举问题（Leader Election，LE）是一个很重要的理论、技术问题。

对于先入门了一点分布式系统开发，再学习分布式系统基本概念的人而言，有一个有意思的“鸡生蛋、蛋生鸡”的辨析：共识算法需要选举一个leader保证liveness（如[Lamport01]）。同时共识算法（包括与共识等价的atomic broadcast算法）又可以支持LE机制的实现（如ZooKeeper官方的“Recipes and Solutions”）。难么，到底是使用共识实现LE，还是有了LE才能实现共识呢？

在具体算法之前，有两个理论的概念需要辨析：何谓一个LE算法是anonymous的？何谓一个LE算法是uniform的？对于“anonymous”性质的辨析，使得我们从理论上认识到，LE算法中，假设每一个process有一个ID是合理的，且是必须的。

课程的讲解主要参考[Attiya04, Chap 3]中的算法。算法从n^2向nlogn的改进，让我们想起了熟悉的comparison-based sorting。

这些课本上的LE算法，特别是同步模型下的LE算法，它们对process的ID值的应用是让程序员感到“不适”的，感觉更像是在做智力游戏，而不是在真正解决分布式系统中的实际问题。这一感觉是对的，上述算法的讲解，主要目的就是帮助学习者厘清基本概念，掌握基本理论知识，不直接指导系统的构建，但是为系统构建打下理论基础。

下界相关的结果暂时略去，它们属于impossibility results的专题。

第五章　基本共识算法

5.1　问题的定义

首先需要对共识问题的严格定义，仔细地学习与体会。特别是validity的概念，初学的时候容易get不到它在说什么，为什么需要它。

[Attiya04]比较简单直接的引入共识问题。[Lynch96-textbook]的讨论比较系统。[Aspnes19]的讨论，看上去比较“凌乱”地分散在好多章节中，问题、模型都在变化，但其实大概这就是共识问题，及其解法，形成的过程。所以，入门可以看第一个，学院派、学习者先看第二个，有点融会贯通的，看第三个。

5.2　课本上的共识算法

共识算法是自媒体上的热门topic，似乎分布式系统方面的程序员没有不懂共识算法的，说起paxos，raft，zk，拜占庭容错，区块链起来，熟得不行。

相比之下，课本上的共识算法就显得有些落寞。但是要真正理解工业级的共识协议及其系统实现，基本的概念，平淡无奇的“课本共识算法”是必要的铺垫。

[Attiya04, Sec 5.1.3]讲了一个非常简单，非常蛮力的共识算法。但是它对大家理解共识，理解如何利用冗余性来容错，有很重要的帮助。再厉害的共识算法，其实并没有跳出这个蛮力算法的路数。

同样对于拜占庭容错，课堂上我们也只讲蛮力算法[Attiya04, Sec 5.2.4, 5.2.5]。大概的原理是，充分多的好人，充分交换意见。这样的话，真话占明显多数，少量的假话，存在也不影响，它们会因为跟大多数真话不一样，而被识别出来。

拜占庭共识算法总体上相对来说是难的，但是我们在课堂教学中，能够接受某些维度的不计成本。在这一前提下，它的难度是不大的。因而，这类算法更多是教学讲解的意义，它们并不是瞄着系统中的实际应用去的。它们更像是在帮助大家认识这个问题，而不是帮助大家得到一个实用的solution。

学习共识算法，很有必要了解共识算法相关的一些不可能性结果。这一内容放在不可能性证明部分，单独讨论。

第六章　类Paxos共识算法族

包括Paxos [Lamport01]，Raft [Ongaro14]和Zookeeper [Hunt10, Junqueira11]。

严格地讲，Zookeeper的Zab协议是一个atomic broadcast协议，由于atomic broadcast 和consensus 两个问题可以互相规约，是等价的[Chandra96unreliable]，所以我们也经常不仔细区分，都称之为共识算法。

第七章　VR共识算法族

2PC算法。
VR算法。
PBFT算法。

7.1　高山仰止：Paxos和VR

Lamport和他提出的Paxos不仅厉害，而且传奇。开始Lamport提出了Paxos算法，由于他的表述方式过于geek，导致读懂的人不多。对分布式方面的程序员而言，Paxos的出名是由于Google的推动。转述6.824（Spring 2021）的讲述，共识算法开始只是象牙塔里的一个结果而已，在众多结果中，也未见有特别过人、出奇之处。当多数据中心的平台逐步出现，构建大规模分布式系统的需求出现的时候，它的理论基础基本是就绪的，缺的就是一个团队真的把象牙塔里的协议给实实在在地实现出来。最早出现的这个团队是Google，它们实现了Paxos [Chandra07]。后续大家都知道了，几乎每个相关的公司、程序员、自媒体，都来讲讲Paxos。

不仅Paxos有很多直接的变体，在回顾Lamport一身工作的这本书中[Malkhi19]，把几乎所有的共识算法，都看成是Paxos的某种变体。这种提法，是有一定道理的。理解不同共识算法之间的本质联系、成长演化的历程，是全面深入学习共识算法的必要条件。

有一个跟Paxos同一时期提出的共识算法叫Viewstamped Replication（VR），它的提出者是2008年图灵奖得主Barbara Liskov。在Paxos如此热门的大背景下，很多人不知道这个算法，或者知道但是大大低估了它的价值。

说VR多么多么厉害，可能有点偏颇，准确的说是要把VR放到Liskov一系列的研究中去看，Liskov这一系列的工作是“高山仰止”级的。Liskov在1980年左右，从data abstraction，CLU语言设计的工作，逐步转到（广义的）分布式数据库系统方面的工作。在这一过程中，为了提高系统的可靠性，她受2PC协议的启发，设计了VR算法。

VR本身不是太有名，或者说风头完全被Paxos盖住。但是对于拜占庭容错，经典的PBFT算法就是VR的派生。再考虑到后来Liskov在分布式事务方面的奠基性工作[Adya99]等，Liskov的这一系列工作是不输于Lamport的Paxos系列工作的。只不过在这一系列的工作中，VR是略显普通的一份子。

这一段历史，Liskov在[Charron-Bost et al., 2010, Chp. 7]中自己有简要论述。

第八章　分布式系统执行的观察与理解

8.1　观察：切片集合的格结构

理解、构建一个异步MSG系统，一个重要的观念的跃迁，是从“绝对的、全序的时钟观”变成“相对的、偏序的时钟观”。形成这种异步分布时钟观的核心是消息传递的因果关系蕴含的时序happen-before关系和异步系统中时刻的概念：分布异步系统的consistent global snapshot。

作为准备知识，需要读者对离散数学中学的偏序关系、格结构，有较为深入的了解。我为本科生通识课，做过一次相关的报告，可供参考：偏序集与格理论-及其在分布式系统中的应用[1]。

异步分布式消息传递系统的执行轨迹（称之以为一个distributed computation，也有很多其它名称），天然自带happen-before关系，基于这一关系可以定义系统的consistent global snapshot，而系统的所有snapshot集合，具有格结构。

这一happen-before关系的定义参见Lamport奠基性的论文[Lamport78]。这一happen-before关系可以用各种不同的逻辑时钟来表示。各种逻辑时钟机制，可以看成是表示偏序关系的代价和表示偏序关系的能力之间的权衡。虽然Lamport那篇奠基性的论文名气很大，但是学习逻辑时钟，建议从最标准、表达能力最强、代价最大的vector clock入手[Mattern89]。

了解happen-before关系及其vector clock编码之后，就可以正式理解distributed computation中的snapshot和snapshot之间的格结构，主要参见[Mattern89]和[Babaoglu93]。

8.2　理解：全局谓词的规约和检测

把distributed computation的格结构刻画清楚之后，就可以从中提取观察者所感兴趣的信息。观察者所关注的，distributed computation所具有的性质，可以形式化规约为一个全局谓词（global predicate），每个谓词有相应的检测算法。

基于上面的准备，观察和理解distributed computation的基本过程是：

- 观察系统执行，得到trace。
- 解析用户规约的全局谓词。
- 在trace上检测谓词的成立情况。

[1] https://www.bilibili.com/video/BV1iU4y1M7SU/

谓词规约和检测的一些初步知识参见[Chandy85, Cooper91, Babaoglu95]等。一篇关于global temporal谓词的更深刻的理论分析见[Charron95]。

第三部分

共享内存算法

这部分讲解SHM模型上的经典分布式算法。

第九章　SHM模型

第一章主要是介绍基本概念。

SHM模型的技术细节，拓展性讨论将在此展开。

主要讨论异步SHM模型。同步SHM模型在PRAM中有讨论，主要属于并行算法的范畴，这里不讨论。

主要的载体是共享寄存器（shared register）。对于共享寄存器，首先讨论它的访问方式（access pattern），然后讨论它的一致性模型。

我们这里将以原子寄存器（atomic register）为例，讲解SHM模型的基本概念。对于SHM一致性模型的更全面的讨论将在第十章展开。

第十章　SHM一致性模型

[Attiya04, Chap 4]所讲解的SHM，其实是一个退化版。因为对shared register的更新是原子的，所以它其实讲的是SHM上的atomic register抽象。Atomic register是一个非常重要的抽象，它是深入理解SHM模型的支点。它的原始定义参考[Lamport86a, Lamport86b]。

在MSG模型上，模拟出SHM模型中的atomic register抽象，是分布共享内存DSM中的经典问题。所以在一些DSM的经典工作中，也可以读到atomic register的定义，而且读起来可能比SHM相关文献中的定义还好懂一些，包括Attiya的DSM奠基性工作[Attiya95]，Welch关于multi-writer情况的讨论[Shao11]，atomic register fast access下界证明的经典工作[Dutta10]，也包括我们自己下界证明的工作[Huang20]和ASC(Almost Strong Consistency)的工作[Wei17, Ouyang21]。

有了atomic register抽象的铺垫之后，可以更深入地了解SHM的一致性模型，包括带时间的atomic/regular/safe registers [Lamport86a, Lamport86b, Shao11]。还包括不带时间的各种模型，及其背后的统一框架[Steinke04]。这个统一框架很深刻，把所有一致性模型都打碎，碎成乐高一样的原子块，进而所有模型都可以挑若干原子块搭建起来，并且通过上述解构，比较容易看出所有模型在强弱关键下，具有lattice结构。

第十一章　分布共享内存

11.1　基本概念

分布共享内存(Distributed Shared Memory, DSM)的概念可以从理论和系统两种不同的视角来辨析。

从理论的视角，它就是在MSG模型上，模拟出SHM的假象。我们的讨论主要基于这一视角，并且主要讨论atomic register的MSG实现。其它更复杂的DSM概念都可以基于此来逐步深入学习。

从系统的视角，DSM最早在[Li98]中提出。随着多数据中心平台的出现，随着分布式云储存、NoSQL、NewSQL、云原生数据库等新型计算平台上新型数据密集型基础设施软件的出现，DSM可以看成是上述实际软件系统的理论抽象。虽然上述实际系统比DSM概念所描绘的系统要复杂很多，但是DSM的概念是上述系统构建的基础铺垫与支撑。

11.2　Atomic register的MSG实现

Atomic register可以进一步分为single-writer和multi-writer的情况。Single-writer情况的MSG实现是Attiya的代表性工作[Attiya95]。[Lynch97]中把它推广到了multi-writer的情况。

不过早期的论文都比较难懂，相关领域深入研究的读者顶多可以看看[Attiya95]，[Lynch97]基本不用直接去看了。[Aspnes19, Chap 16]有提炼后的重新表述，从学习的角度而言，看课本就可以了，很多不必要的细节都被合理抛弃了。

第十二章　互斥问题

有了SHM模型的基本概念之后，就可以比较系统地了解互斥问题（Mutual Exclusion，MUTEX）。

12.1　问题的定义

需要从safety和liveness两个侧面来理解MUTEX问题的定义。MUTEX其实不是一个确定的问题，而是一个问题族。一般它的safety部分没什么变化，而liveness部分有种类繁多的变体。

要真正掌握MUTEX问题的定义，你得能够看懂别人精确定义的问题，并且能根据自己实际面对的问题，给出最合适的系统建模和问题定义。

从宏观的视角来看，互斥算法可以看成是一个频谱：SHM提供的存储抽象越强，那MUTEX就越容易解决，反之就越难。

12.2　基于强原语的MUTEX算法

[Attiya04, Chap 4]中先讲了几个用强大原语 - 包括test&set，read-modify-write等 - 的MUTEX算法。

作为一个实际系统中的例子，6.824（Spring 2021）的L4 - Primary-backup Replication里有一个用powerful原语的例子。当primary和backup断连时，它们都可以访问一个storage sverver，它们用test&set原语来更新storage server上的一个标志位来实现容错场景下最核心的协同。

12.3　Bakery算法：从“伪MUTEX算法”到“真MUTEX算法”

MUTEX算法中，比较经典的是Lamport提出的Bakery算法。可以分几个步骤来逐步理解这个算法。

- 协同的载体：首先理解进程p_i拥有（own）一个register的概念，或者说single-writer multi-reader register的妙用。每个proc拥有一个register，即只有我能写它，其他人都可以读。这

样每个proc拥有一个register，大家就可以互相通过这些register比较自如地完成交流、协同。

- 取号机制的SHM实现：基于上述概念，就让每个proc类似银行排队取号一样，等待进入临界区。只不过这里的取号是通过每个进程 p_i 都拥有的register来实现的。每个proc看看别人的号，就知道自己取号要取所有号里最大号的下一个号。

- 承诺我主意不改了：上述取号机制的基本原理是对的，但是有比较直觉的漏洞。根据adversary argument的视角（adversary argument的基本概念可以参考我本科生算法课的L7），别人一读到我own的register，我就恶意地去改它，这是可以让上述机制出错的。为此我们课上直接构造了一个这样的例子。所以需要进一步引入一个类似承诺的概念，就是“买定离手”，我不改了。这个机制是通过让每个proc拥有一个flag register $Choosing_i$ 来实现的。

Bakery算法的基本设计讲完了，但是故事才刚刚开始。上面所有的MUTEX算在Lamport看来，都是伪互斥算法，因为底层SHM某种意义上已经提供互斥了，所以你的互斥都是伪互斥，是“剽窃”别人的。要真的自己实现互斥，你连atomic register都不能用，因为atomicity显然已经提供了某种互斥。而他所提出的Bakery算法是真互斥算法，也就是说Bakery算法是不依赖底层register的能力的，它基于最弱的Safe register就能实现。（注意，课题上为了入门学习的方便，我们是简化了模型，讲了Bakery算法的退化版本，即假设SHM提供了atomic register，而基于atomic reigster我们可以实现Bakery算法。）

大家比较熟悉各种register的语义之后，就可以深入辨析Bakery算法的正确性证明，来体会上面的结论。从这一角度来讲，Bakery算法是最牛的MUTEX算法。

12.4　Bakery算法的后续改进

Bakery算法在一个维度有明显的问题，即它要求register能存任意大的值。利用adversary argument的观点很好理解，大家疯狂取号，那这个号就会一直增下去，没有机会归零，总有一个时刻，register存不下应该表示的号了。

为此[Attiya04, Chap 4]讲了一套改进的办法，它是一系列算法，跳板式逐步改进。这一跳板式的改进思路，比算法的细节更重要：

- 2个人，不对等：只考虑两个proc，而且有一个人优先级高，同等条件下，总能抢占另一个人。基于这一条件，比较容易实现MUTEX。

- 2个人，对等：有了上面的跳板，让优先级高的人不固定，也就是每个时刻都是不对等的，但是两个人轮流做那个优先级高的人。

- n个人，对等：有了上面的跳板，就让大家锦标赛，两两比赛（比谁能顺利进入临界区），比出的冠军，直接进入临界区。

第四部分

分布式算法分析技术

这部分讲解重要的分析技术。

第十三章　不可能性结果

13.1　概述

当你熟悉分布式算法、分布式系统的基本概念，对它们有一点宏观性的思考的时候，一个很重要的概念就是分布式算法、分布式系统中的基本的不可能性结果。

❝ 早在89年的PODC上，Lynch就survey了“A hundred impossibility proofs for distributed computing”[Lynch89]，到了03年，Faith Ellen（与人合作）又写了“Hundreds of impossibility results for distributed computing”[Fich2003]。最终到了14年，Hagit Attiya和Faith Ellen汇集成书“Impossibility Results for Distributed Computing”[Attiya14][1]。❞

13.2　FLP

不可能性结果的系统讨论超出introduction级别的教科书的范畴，有需要直接去看[Attiya14]即可。但是有两组结果有一些特别的原因，专门提一下。

一个是异步容错共识的不可能性结果FLP。由于共识算法的大热，它也老被人提起。它的原始证明见[Fischer85]，直接看课本的表述更容易一些[Attiya14]。

看这类证明，必须对分布式系统的执行有一个“超然世外的上帝视角”。分布式系统的执行就像是电影拍摄的片段、素材，你就像剪辑导演，随便复制、改写、拼接，直到得到满足你要求的执行。

13.3　拜占庭共识：“叛徒”比例不能达到$\frac{1}{3}$

同步模型下解决拜占庭共识问题的一个基础结论是：当$3f > n$时，是不可能解决拜占庭共识问题的。其证明的关键在于，采用adversary argument的视角，可以让拜占庭错误的proc做你希望的任意动作，这样你就可以构造两个不可区分的执行。这两个执行的不可区分性与它们分别达成不同的共识，又是矛盾的。

具体的构造是巧妙的，需要结合[Attiya04, Sec 5.2.3]的图示来理解。证明是先讨论n=3的特殊情况，然后再据此推出任意n情况的证明。

[1]上述理论结果背后一个有趣的现象是，上面所提到的作者都是女性，分布式算法领域两本教科书的作者也是女性，她们的女性学生后续也在分布式理论领域做出了出色的结果。

13.4 Chain Argument

这是不可能性结果证明中的一个简答而有效的技术，主要用于构造分布式系统执行之间的不可区分性（indistinguishability）。因为我们下面的理论证明用到了它，所以专门提一下。

这个技术可以让你很形象地理解什么叫做，以一个剪辑导演的视角，拼接分布式系统的执行片段。我们的PODC2020工作的会议报告中，有一个形象的简单例子 [2]。

具体内容可以看[Attiya14, Chap 2]。我自己由于下面研究的关系，实际看的是一份具体的研究工作[Hadjistasi16]（这是一篇PODC16的短文，具体细节可以看它的arxiv长文版本）。

13.5 Fast Access to Atomic Registers

实现atomic register经常需要要两轮（roundtrip）的通信，由此一轮通信的算法就被称为是fast的。

直觉上一轮通信是不可能实现atomic register抽象的，但是严格证明它却不那么容易。还是分single-writer和multi-writer的情况来逐个击破。

[Dutta10]证明了single-writer情况的fast实现不可能。

Multi-writer情况的不可能性结果，期间有一些受限情况下的证明。我们自己的一份工作最终解决了这一问题[Huang20]。

此外，既然fast是不可能严格实现atomic register的，那我们的另一系列的工作就讨论：假设fast是必须的，atomicity会被牺牲到什么程度。针对这一问题，我们提出了ASC（Almost Strong Consistency）的概念，并做了一些初步的理论分析与实验研究 [Wei17, Ouyang21]。

[2] https://www.bilibili.com/video/BV1K54y1D766/

附录 A　一些未入选的重要内容

有一些内容不太适合写在introduction level的分布式算法教科书中，但是又是我学习之后，收获特别大的，所以在附录里面，简单提及一下。

- 2PC：2 Phase Commit（2PC）背后是分布事务的transactional atomicity维护的问题。2PC还有各种变体，比如3PC等。分布式数据库、云原生数据库时代，2PC和共识等机制等还有各种方式、各种程度的融合。这是我们目前很关注的问题，不过放在分布式算法的入门书里面有点太“专”了。

- Failure Detector：虽然容错是分布式系统构建的重要概念，但是failure detector的抽象还是太理论了，只有比较偏理论研究的读者才适合全面深入地学习 [Chandra96unreliable]。

  Weakest failure detector的这一份工作名气也挺大，比较难懂[Chandra96weakest]。后来我们在自己的工作[Huang20]中的一部分证明，其实就是这份工作证明的一个大幅简化版。我们做完[Huang20]我才真正看懂这份工作[Chandra96weakest]，并且收获很大，知道自己的证明其实可以仿照它，写得更严谨，更易懂。

- 基于TLA+的分布式系统形式化规约与建模：这是Lamport发明的语言及工具链，可贵之处是在工业界获得了真正的应用，而不是像很多形式化理论与技术一样，论文发表之后就被束之高阁。我们有一系列基于TLA+规约与验证的工作。同时我们现在还致力于将模型层的规约与验证，与代码层的测试结合起来，预计后续会有这方面的进展。

附录 B　参考文献说明

所有参考文献ID对应的bib，均可以在这个链接查到：https://github.com/alg-nju/disalg-bib-dict。

注：bib-dict是“相关的”文献都会列，不一定每一篇文献都是经典。比如常见的一种情况是，有一篇文献的某个局部/某个侧面，对学习某个知识点有比较高的参考价值，它就会被列进来。深入学习还是要结合讲义的评述，加上自己的深入阅读。读完、读懂之后，自己才能真正知道哪个是经典，它为什么经典。

未来修订中，会陆续把这些指向网络的BibID换成正式的参考文献。

附录 C　讲义的修订历史

- ✍ 2022/08/05：回到latex讲义环境，写了一小部分素材，发现写讲义和写知乎版介绍的概念还是有质的不同，甚至是冲突。回归到传统的讲义写作上下文，继续推进写作。
- ✍ 2022/05/29：知乎版讲义的初步效果还不错，有效推进了讲义的写作。随着基本框架的成型，后续的正式写作，转回到latex+github的写作、发布模式。这算是这本讲义的V0.2版。
- ✍ 2022/05/03：换一个方式，在知乎专栏上写了一版轻量级的讲义。这是当时的说明：“以前试图写一个重量级的授课讲义，推进非常缓慢。现在尝试写一个轻量级的讲义，以主要文献的推荐和简评为主，推进一段时间，试试效果。”。
- ✍ 2021/11/24：2021年陆续写了一个讲义的草稿，基本就是文献的列举，没有实质性的评述与讨论。这算是这本讲义的V0.1版。

参考文献

[Alpern and Schneider, 1985] Alpern, B. and Schneider, F. B. (1985). Defining liveness. *Information Processing Letters*, 21(4):181–185.

[Aspnes, 2019] Aspnes, J. (2019). *Notes on Theory of Distributed Systems*. Yale University, CPSC 465/565.

[Attiya et al., 1995] Attiya, H., Bar-Noy, A., and Dolev, D. (1995). Sharing memory robustly in message-passing systems. *J. ACM*, 42(1):124–142.

[Attiya and Welch, 2004] Attiya, H. and Welch, J. (2004). *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. John Wiley & Sons.

[Cachin et al., 2011] Cachin, C., Guerraoui, R., and Rodrigues, L. (2011). *Introduction to Reliable and Secure Distributed Programming*. Springer Publishing Company, Incorporated, 2nd edition.

[Charron-Bost et al., 2010] Charron-Bost, B., Pedone, F., and Schiper, A., editors (2010). *Replication: Theory and Practice*. Springer-Verlag, Berlin, Heidelberg.

[Fischer et al., 1985] Fischer, M. J., Lynch, N. A., and Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382.

[Malkhi, 2019] Malkhi, D., editor (2019). *Concurrency: The Works of Leslie Lamport*. Association for Computing Machinery, New York, NY, USA.