我的看论文计划1
1 Bitcoin: A Peer-to Peer Electronic Cash System
2 Hidden Technical Debt in Machine Learning Systems
3 Software 2.0
4 BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
5 The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks
6 Releasing the death-grip of null hypothesis statistical testing (p < .05)

谈起大数据，Hadoop MapReduce HBase Spark等等组件实际已经成为工业标准。中国乃至全世界的大数据项目，绝大多数其实是构建在Apache Hadoop之上的。Hadoop继承了分布式文件系统HDFS和分布式计算引擎MapReduce。但是这些设计其实要thanks to Google。Hadoop的设计思想可以说完全来自于Google当年发表的三篇论文。Google的大数据老三篇可以说是每个搞大数据的同学要必读的论文。

是哪老三篇呢?

Ghemawat, S., Gobioff, H., & Leung, S. T. (2003). The Google file system.

https://s3.amazonaws.com/academia.edu.documents/28578273/the.google.file.system.pdf?response-content-disposition=inline%3B%20filename%3DThe_Google_file_system.pdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWOWYYGZ2Y53UL3A%2F20200113%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20200113T122314Z&X-Amz-Expires=3600&X-Amz-SignedHeaders=host&X-Amz-Signature=3b7935eb6fddc44e70d5a4c0bf69e09c74e5462db3cd1326c837ecc5d2876f9e

Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), 107-113.

https://www.cs.amherst.edu/~ccmcgeoch/cs34/papers/p107-dean.pdf

Fay Chang, J. D., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., ... & Gruber, R. E. (2006, November). Bigtable: A distributed storage system for structured data. In Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (Vol. 7, pp. 15-15).

https://static.usenix.org/event/osdi06/tech/chang/chang.pdf

希望大家还是能够读一下原文，可以看到很多系统设计从0到1的思想在里面。同时也可以学到如何给一个分布式系统做benchmark。

https://zhuanlan.zhihu.com/p/37670430

https://posts.careerengine.us/p/5e532205e0087b4d0ca085fc

自己读过的论文中，最惊艳的可以归为几类：

1. 构思巧妙型：GAN，这类可能打死都想不到。
2. 打破范式型：Transformer，这类敢于抛弃已有范式，需要胆识。
3. 大道至简型：ResNet、BatchNorm、AdderNet，这类抓住问题本质进行解决。
4. 大力出奇迹型：AlphaGo、GPT-3，这类需要算力及数据，很考验工程能力。

https://zhuanlan.zhihu.com/p/161054119
前言
不知不觉，2020年已经过去一半了，最近突然反应过来自己也看了不少文献资料了，就想着把看过的文献和觉得比较好的书籍做

一个总结，基本都是大数据分布式领域的，回顾自己学识的同时，也给想从事或这个领域的小伙伴一些参考 。最后顺便把接下来要看的东西列个列表，也会将自己学习的心得和经验分享出来，有需要的童鞋可以参考参考。

另外有些文献看完我会进行整理和输出，这部分链接我一并附在文献的介绍后面，后面看的书或是文献也会保持这种习惯，如果觉得有兴趣欢迎各位大佬交流，顺便也可以点波关注~~

论文总结
MapReduce 《MapReduce Simplified Data Processing on Large Clusters》
从现在的眼光来看，Mapreduce可以说可圈可点。但在那个年代，这个思想可以说是相当先进的。不得不说Google一直引领技术潮流，包括近几年流行的k8s也是Google主导。

这篇文章主要介绍了Mapreduce的流程还有一些细节方面的介绍，如果已经有使用过Mapreduce编程的小伙伴应该看一遍就能懂。另外，看完如果想加以巩固的话，推荐做MIT6.824的Lab1，用go实现一个Mapreduce。至于什么是Mit6.824，百度一下就知道喔。我以前也有写过一篇介绍MR，有兴趣的童鞋不妨看看：从分治算法到 Hadoop MapReduce。

地址：MapReduce: Simplified Data Processing on Large Cluster

GFS 《The Google File System》
GFS和Mapreduce这两篇论文直接催生了Hadoop的诞生。不同于Mapreduce，Hadoop的hdfs到今天依旧是工业界主流是海量数据存储方案，这证明了这一存储方案的优越性。

这篇文章介绍了Google内部存储方案GFS的实现，namenode存储哪些元数据信息，datanode如何保存数（问题可见这篇博客），带着问题阅读这篇论文。

不过熟悉Hdfs的童鞋读过后应该会发现，GFS和Hdfs其实是有些不一样的。比如上传的流程，namenode存储元数据的方式，至于为什么，等待各位童鞋挖掘答案啦。

另外在Hadoop之前用于存储"大数据"的是RAID，对这块有兴趣的童鞋可以看看这篇：从 RAID 到 Hadoop Hdfs 『大数据存储的进化史』。

论文地址：The Google File System

Bigtabble 《Bigtable A Distributed Storage System for Structured Data》
Bigtable，目前业内闻名的Nodel组件Hbase就是它的开源实现。这篇文章主要介绍了Google内部基于GFS的分布式结构化数据存储系统。

GFS本身是适合追加数据而不适合随机写，文章介绍Bigdata为了适配这种特点而使用的LSM-tree存储结构，而后又阐述一些优化的方案，诸如布隆过滤器。关于LSM-tree有兴趣的小伙伴可以看看这篇：数据的存储结构浅析LSM-Tree和B-tree。

论文地址：Bigtable: A Distributed Storage System for Structured Data

Spark RDD 《Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing》
Spark RDD的论文，RDD的全名叫弹性分布式数据集。当初MapReduce模型兴起的时候，大家都以为已经迎来了曙光，但一段时间后才发现这东西其实也不是万能，尤其是在机器学习等需要迭代计算的地方。而究其原因，其实是MapReduce在计算过程中，中间数据需要多次落盘，导致增加许多磁盘IO。

相比之下，RDD使用的DAG计算模型则更加优越。一方面是它将多个计算逻辑梳理为一个DAG有向无环图，可以一定程度减少不必要的shuffle等耗时操作。另一方面，更加侧重于使用内存进行计算，减少磁盘开销。

读这篇论文会收获到有关RDD的设计细节。

论文地址：Resilient Distributed Datasets: A Fault-Tolerant Abstraction for
In-Memory Cluster Computing

Spark SQL 《Spark SQL: Relational Data Processing in Spark》

在Spark SQL模块中，提出了DataFrame API，方便用户进行关系型操作（join，group by）等，而其底层使用的还是RDD。

另外一条SQL语句的执行逻辑，包括解析，验证，优化，生成物理执行计划，执行过程中的优化逻辑等等，这里内容都可以在这篇文章找到。

对SQL解析感兴趣的小伙伴，这篇不要错过，还有下面会介绍到的Calcite的论文，都是跟SQL解析相关的，不过Calcite侧重于适配多个数据源和内部组件的可插拔，上手难度会更高些。

我以前有结合这篇文章，写了Spark SQL的源码解析系列，有兴趣的童鞋可以看看Spark SQL源码剖析（一）SQL解析框架Catalyst流程概述。

论文地址：Discretized Streams: Fault-Tolerant Streaming Computation at Scale

Spark Streaming《Discretized Streams: Fault-Tolerant Streaming Computation at Scale》
流式处理被誉为大数据技术的未来，Spark Streaming在现在看来有些落后了（跟Flink相比）。

在流处理领域中，由于数据是源源不断的，但系统通常无法保证一直是健康状态，数据也有可能出现落后的情况，所以容错是很重要的点。Spark Streaming主要通过备份和上游重放结合的方式来保存数据和状态信息实现容错，而一切的核心是微批的处理思想，这里就不展开太多了。

另一个点是延迟，Spark streaming由于使用了微批，延迟只能做到亚秒级，可以说成也微批，败也微批。现在Spark的流处理模块改用Flink一样的算法重写，不过好像还没完全实现完成。

通过这篇文章可以了解到Spark streaming的设计思想，对错误处理的实现机制，还有落后节点的处理。

论文地址：Discretized Streams: Fault-Tolerant Streaming Computation at Scale

Raft共识《In Search of an Understandable Consensus Algorithm》
共识，可以说是分布式时代的基石，很多系统的基础功能都是在共识的基础上实现的。按我的理解，共识是了解分布式系统理论原理的一把钥匙。

最早的时候，分布式系统一致性共识一直是Paxos算法的天下。就是说其分布式一致性就会想到Paxos，但Paxos算法太过复杂难以理解和工程化。所以就有了Raft算法。

这篇文章主要讲述Raft算法的具体流程，包括领导者选举，日志复制等内容，看完你会发现，原来分布式共识算法就跟个小玩具一样。

有兴趣深入的童鞋可以再接着做MIT6.824的Lab2，算是一个很有挑战是实验了。

对了，看的时候可以搭配我以前的这篇博客喔分布式系统一致性问题与Raft算法（上）

论文地址：In Search of an Understandable Consensus Algorithm

Calcite《Apache Calcite: A Foundational Framework for Optimized Query Processing Over Heterogeneous Data Sources》
Calcite也提供了通过SQL管理数据的功能，但是它本身并不负责管理数据源和元数据信息。

它设计出来的目标，是因为在后来在各个领域，流处理，批处理，文本检索等等都有各自专长的工具，这些工具通常都需要用到SQL解析模块。如果每个工具，比如Flink，ElasticSearch等自己开发一套SQL解析工具那无疑是在重复造轮子。

Calcite就是为了专门解决这个问题，所以它的主要考虑目标是通用性和可插拔。它里面用到的parser，validate，optimizer模块都可以单独拿出来使用。比如Hive就是自己直线parser和validate，使用了Calcite的optimizer来对SQL优化。

相对而言，Calcite的门槛会更高一些，但通用性更好，如果对SQL解析这块业务有需求的人可以考虑了解看看。

论文地址：Apache Calcite: A Foundational Framework for Optimized Query Processing Over Heterogeneous Data Sources

AnalyticDB《AnalyticDB: Real-time OLAP Database System at Alibaba Cloud》
AnalyticDB是阿里巴巴刚发表不久的一篇系统论文，它的一个可以实时分析的OLAP数据库。

目前业界开源的支持流式的OLAP数据库，包括预计算的Kylin streaming，偏向时间数据的Apache Druid，还有Clickhouse等。

但很难有系统可以做到尽善尽美，即很难同时兼顾海量数据，灵活性，性能都较为优秀。

而AnalyticDB可以说是较为成功的一个系统，它确实在很多方面都做的比较好，在设计上也有不少创新的点。对OLAP这块内容有研究的小伙伴可以看看文章。当然这个目前还不是开源的，仅有论文可以参考。

我之前写过一篇博文，AnalyticDB实现和特点浅析，里面根据论文介绍了AnalyticDB的实现，一些特点还与当前业界开源系统做了对比，有兴趣可以看看。

论文地址：AnalyticDB: Real-time OLAP Database System at AlibabaCloud

S4（Storm）《S4: Distributed Stream Computing Platform》
S4是比较早期的流处理方面的论文，在那个时代的创新点在于，可以让用户自定义计算逻辑而非仅使用算子进行计算。

当然它的缺陷也比较明显，比如对落后数据直接忽视，对数据exactly once语义支持的不完善等等。

论文地址：S4: Distributed Stream Computing Platform

ZooKeeper《ZooKeeper: Wait-free coordination for Internet-scale systems》
Zookeeper是一个比较知名的开源分布式共识组件。论文中有说到它底层使用的是ZAB协议（但具体的细节也没说明），但其实自己观察就会发现，ZAB协议跟Raft算法是很像的，只是对一些细节部分做了一定的修改。

论文更偏向其对这样一个共识系统的功能和系统设计实现，对底层的算法介绍偏少。推荐先看Raft算法那篇，然后再看这篇Zookeeper的会好很多。

论文地址：ZooKeeper: Wait-free coordination for Internet-scale systems

Yarn《Apache Hadoop YARN: Yet Another Resource Negotiator》
yarn是一个调度管理系统。最早的时候，Hadoop的资源管理功能是由JobTracker负责的。但它同时还负责了很多功能，这样就容易出错并且有单点故障问题，而后yarn就独立出来。后面发现yarn越来越受到欢迎，就逐渐开放，然后发展到一个可以让大家都接入的资源调度系统。

这篇论文主要讲述yarn的设计结构，里面的各个模块，工作原理等等。我以前也有写过yarn的博文，可以结合看看Hadoop Yarn框架原理解析。

论文地址：Apache Hadoop YARN: Yet Another Resource Negotiator

DDIA
这其实是一本书来着，中文全程是《据密集型应用系统设计》。

可以说是讲述分布式系统中"道"那一部分的书籍，它并非纯理论的书籍，而是很好得和工业界的一些实战结合起来。真心觉得每一个从事分布式系统相关工作的开发人员都应该读一读这本书。

其实一直有打算尝试写一篇文章串起这本书的内容，不过工程有些浩大，导致一拖再拖，汗 = =! 。

后续待读列表
顺便贴下我后面打算看的一些文献，把简介也附上，给各位童鞋一个参考:)。

容器技术《Large-scale cluster management at Google with Borg》
容器和编排技术应该算这几年比较热门的一个板块，这篇讲述的是Google内部的容器Borg。

地址：Large-scale cluster management at Google with Borg

Lambda 架构《Lambda Architecture for Cost-effective Batch and Speed Big Data processing》
地址：Lambda Architecture for Cost-effective Batch and Speed Big Data processing
数据模型已经从最开始的离线T+1处理模式，转变Lambda架构，现在还有新的纯实时的Kappa架构。

这篇文章主要就是介绍Lambda架构的。

分布式快照算法《Distributed Snapshots: Determining Global States of Distributed Systems》
文中介绍的Chandy-Lamport，基本是当前主流分布式计算系统的标配，包括Spark，Flink等等。

主要介绍分布式系统中如何保证快照一致性。

地址：Distributed Snapshots: Determining Global States of Distributed Systems

SQL优化器模型Volcano The Volcano Optimizer Generator: Extensibility and Efficient Search
Volcano 模型的经典论文，因为最近在看SQL解析优化相关内容，这部分可能会优先级比较高。

The Volcano Optimizer Generator: Extensibility and Efficient Search

SQL优化器Cascades The Cascades Framework for Query Optimization
和上面一篇Cascades模型是一脉相承之作。

The Cascades Framework for Query Optimization

Dataflow 《The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing (VLDB)》
来自 Google 的将 stream processing 模型和 batch processing 模型统一的尝试。在 Dataflow model 下，底层依赖 FlumeJava 支持 batch processing，依赖 MillWheel 支持 stream processing。Dataflow model 的开源实现是 Apache Beam 项目。

地址：The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing (VLDB)

Flink 《Apache Flink: Stream and Batch Processing in a Single Engine》
Apache Flink 是一个处理 streaming data 和 batch data 的开源系统。Flink 的设计哲学是，包括实时分析 (real-time analytics)、持续数据处理 (continuous data pipelines)、历史数据处理 (historic data processing / batch)、迭代式算法 (iterative algorithms - machine learning, graph analysis) 等的很多类数据处理应用，都能用 pipelined fault-tolerant 的 dataflows 执行模型来表达。

地址：Apache Flink: Stream and Batch Processing in a Single Engine

MillWheel 《MillWheel: Fault-Tolerant Stream Processing at Internet Scale》
MillWheel 是 Google 内部研发的实时流数据处理系统，具有分布式、低延迟、高可用、支持 exactly-once 语义的特点。不出意外，MillWheel 是 Google 强大 infra structure 和强大 engeering 能力的综合体现 —— 利用 Bigtable/Spanner 作为后备状态存储、保证 exactly-once 特性等等。另外，MillWheel 将 watermark 机制发扬光大，对 event time 有着非常好的支持。推荐对 streaming system 感兴趣的朋友一定多读几遍此篇论文 —— 虽然此篇已经发表了几年，但工业界开源的系统尚未完全达到 MillWheel 的水平。

地址：MillWheel: Fault-Tolerant Stream Processing at Internet Scale

END-TO-END ARGUMENTS IN SYSTEM DESIGN
这篇讲述的是分布式理论方面的只是，论证了这样一个观点：端到端的可靠通信，只能通过通信两端的application层来保证，而中间件(比如SQS, Kinesis, ActiveMQ, 到更低层Netty乃至TCP)只能提高效率，而无法保证通信的可靠性。

这篇论文发表的时间是在1984年，算是比较老的文献，不过其中的观点到如今依旧不算过时。想看这篇文章是受到知乎一个大神

的安利。

不过这种关于设计原则的论文一般都会写得比较抽象，比较难啃。
地址：END-TO-END ARGUMENTS IN SYSTEM DESIGN

Rethinking the Design of the Internet- The end to end arguments vs. the brave new world
《Streaming System》
Streaming System是一本介绍流计算相关概念的书，该书没有介绍很多实际的用例以及流计算的实现的具体方法，但是从理念上介绍了流计算相关的思想以及实现的特点，有助于提高对流计算的理解。

怎么读论文
每个人都有自己的学习方法，一些方法没有好坏之分，只有适合不适合自己。所以这里我也只说明我自己阅读文献的一些方法，希望能给各位小伙伴一点参考。

工具
工欲善其事必先利其器，好的pdf阅读工具是必不可少的。我目前用过比较合适的是mac下的Adobe Acrobat DC for mac，免费的。而windows下的Adobe家的pdf没用过不做评价。windows下用的是Gaaiho Reader。

我个人觉得读文件比较需要用到的两个功能，一个是添加附注，一个是文字高亮。

上述两个工具，都可以直接选择文字标识高亮，还有右键添加附注，相对而言比较轻巧且均免费。

添加附注是可以让你随时对自己看的内容记录下来，后面再看的时候按照自己附注的线索阅读就行，否则过一阵子再看论文会有一种陌生感。

高亮则可以将重点部分高亮起来，起到突出重点的作用。

阅读方法
我一直信奉输出倒逼输入，看我上面的论文介绍应该也发现了，很多东西我看完都会输出。所以我学习东西的核心思想就是输入倒逼输出。

好处什么的就不介绍了，见仁见智。只说一些点，首先，论文通常看一遍是不够的，基本上都是两三遍起步（一些发现没价值的除外），一些关键点的论述更是应该多阅读几遍。

第一遍的时候可以先通篇泛读，把握文献的整体结构，这一遍我一般会先侧重与论文出现的背景，它要解决的问题是什么，与当前一些方案相比有什么优势（劣势一般论文中不会说= =）。再看看解决方案的大概内容，有没有比较感兴趣或可能用的到的点。必要的地方做一做笔记，主要是为了后面回顾的时候快速明白看过的内容。

第二遍重点了解论文中解决方案的整体实现流程。其中肯定有些不懂的地方，还有精彩的，以后可能用的到的地方，这些内容都先记录下来。一般第二遍后起码会对论文的整体内容有比较清晰的了解。

第三遍主要是针对一些技术点的深入，可以与当前业界的一些方案相互比较，或者是查阅一下其他资料深入了解一些点的原理。甚至可以找到论文对应实现的系统，查阅对应的源码了解具体的实现过程。

如果还是觉得有不明白的地方，可以重复上述流程。

最后如果觉得论文有价值或者对论文方向感兴趣，可以找一个点与论文结合起来输出一篇文章。当然单纯论文解读也是可以，但那样有点重复造轮子的感觉。

更好的做法，应该是寻找对应领域的文章，相互比对分析然后再产出。比如说看了Spark Streaming，可以结合Flink等系统的资料，输出流处理方面的文章，不过这个最大的问题就是太耗时间了（哭笑），仅适用于想深入钻研的领域且有足够的时间。

以上~

PS：由于本人水平有限，部分阐述可能存在失误，如果有发现问题欢迎在评论区指正。