

评:Streaming System(简直炸裂,强势安利)

 **阿莱克西斯** 
程序员 话题的优秀回答者

已关注

laike9m 等 1,538 人赞了该文章

(最近会在这个专栏: 用谁都能看懂的方法解释分布式系统, 更新一些关于流式系统的论文, 比如 Flink的Global Snapshot, 敬请期待)

(**预警: 本文中夹杂英文**: 作者想到一个概念想要表达的时候, 想到的是英文就用英文想到的是中文就用中文, **非常任性**, 对中英混杂有所不适者请谨慎阅读)



但是这直接造成列了巨多的点之后,我发现写起来就收不住了,且很多东西很难——讲清楚,于是删减了不少,很多东西也只能一带而过;而到最后,完美的把这些点用一条线和这本书紧密的串联起来也非常难,所以如果本文读起来显得杂乱,请见谅。)

这可能是我2018年遇到的最好的书(毕竟2018年还没过完),如果对流式系统你只想看一本书,那必须是这本。

早在好几年前,在分布式Streaming处理还是比较新的概念的时候,我就亲手用Storm搭建过一个产品级的分布式流处理系统(还记得由于Storm的文档渣的不行,很多东西是看了源码才明白怎么回事儿。。。)

而后来由于对分布式系统的强烈兴趣,我对Spark Streaming, Akka Streaming, 和Flink也有所了解, Spark和Flink相关的比较重要的论文也基本看了个遍。即使把玩过这么多流处理框架,我也还是深深的感觉到流系统是一个比batch的大数据 processing难上几个档次的问题,一方面,时间这个因素给我们需要考虑的问题空间整整增加了一维,即任何的数据处理逻辑都需要开始考虑数据的到来,处理和commit的时间因素,另一方面,我们需要应对的是永无止境的源源不断的数据,这对系统的健壮性和scalability都有远超batch processing的要求。

然而遇到这本书,让我发现我对streaming的理解还不够精髓(主要理解太偏底层实现了,思想被淹没在了不同框架的实现细节里,而忽视了对问题本身的本质理解),这本书改变了我思考问题的模式,让我跳出了流式系统框架的实现细节,站在了更高的维度思考"什么是分布式数据处理"的本质,当我利用书中的思维体系来分析我所遇到的分布式系统,我第一次感觉到流式系统,甚至分布式业务系统,其实是如此的简单,作者用非常流畅的思路解释了流处理中的各种概念,当问题和解法放在一起,你会发现理解window, watermark, trigger,这些东西非常的自然。就好比当你生活在三维问题空间不断锻炼,当你的思维已经非常熟悉三维的问题的各种理论和解法,这时你再去看二维的问题,会有一种一眼看透的感觉。升级你的思维,是这本书将带给你的珍宝。

这本书绝不仅仅是在讲流式处理,它的真名是"总结过去10年大数据处理发展的历史, Google的经验;提出新的理论, **升维我们的思维模式, 降维打击我们的复杂问题**;展现目前的行业趋势, 预测未来的行业走势....", 这本书几乎影响了我对所有分布式系统的理解。

待我慢慢的——道来。

一点历史

分布式流式系统一般指能够实时接收"海量"(非必须)信息并作出增量计算的机群系统,计算节点通过数据流连接,不断根据input流生成output流,或者把计算结果流apply到数据库或者文件系统中,供其他service使用。流式系统不用像沉重的MapReduce那样,必须积累足够的数据,然后才能进行一次大批量数据计算,来产生结果。

讲流式系统就逃不开要讲一下Nathan Marz创造的流处理框架:Storm, 他的Lambda Architecture, 和他的Big Data: Principles and best practices of scalable realtime data systems(May 10, 2015)这本书。

在MapReduce已经盛行,但是大家开始不再满足于积累足够的数据才能batch处理一次数据,而是越来越需要快速反应的增量计算的情况下, Nathan Marz的Storm提供给了世界第一个流行的(注意是第一个流行的而不是第一个)开源的,高可用的分布式流处理框架。

但是Storm对系统设计提出了新的问题;由于分布式环境的不稳定性, Storm当时无法保证生成deterministic的结果,无法保证计算的consistency,从而无法对application层提供足够的正确性支持。

为了应对这个大数据处理的共有问题,他的how-to-beat-the-cap-theorem这篇文章,和Big Data这本书代表着著名的Lambda architecture横空出世和开始广为人知和流行。Lambda

然而Lambda architecture也有其自身的问题,一般需要维护batch和stream两套对同一业务系统的实现,他们的系统模型无法得到统一,处理stream的增量计算和batch计算结果的merge也是很烫手的问题。更重要的,由于streaming系统无法保证准确的计算,系统使用者开始不信任streaming提供的信息,转而更宁愿去等待比较慢的batch计算的结果。这样streaming提供的价值全无。。。

我们需要一个更好的解。。。。(这条线暂且按下不表)

Microservice之后的系统设计思路发展

书接上回(简评)Kubernetes In Action(大势所趋),我们说到"逐利的资本","开源免费的软件","DevOps/Agile"和"云计算运营商"的出现,使得软硬件的cost大大下降,使得**商业需求开始在系统设计中变成特等公民**。微服务开始代替SOA登上历史舞台,抽象了硬件资源的Container和Container Orchestrators必定将大放光彩。

然而! 分布式系统的搭建还是太难了(需要大量的知识积累和设计实现精力)。。。我们需要新的抽象,基础服务来隔离这些复杂度,培养大量可以基于"这些新基础"(而不必理解这些新基础的实现细节)就可以实现复杂分布式业务系统的SDE,让SDE resource使用在真正能够提供业务价值的application上(比如科学模型的建立),而不是一遍遍去用相同的手段去处理这些繁琐的细节。我们需要更高一层的Programming model,基础问题需要从关注使用多少内存,使用多少硬盘,怎么做网络通信这些更底层的细节。升级到考虑什么时候使用sync,什么时候必须要注意failure tolerance, liveness和safety怎么做trade-off,什么时候使用state,什么时候使用CRDT,什么时候需要幂等,怎么做partition如何设计partitionkey,思考云环境和分布式环境的"新基础思考因素"。

另一方面,当需求开始在系统设计中变成特等公民,意味着PM们可以更加轻松的去释放自己的想象力,爆炸的无限灵活的业务需求开始涌现。进一步冲击着老的基础设施。其中,以关系数据库为核心的集中式的储存成为必定要谢幕的构架思路。**"数据不应被它的储存形式所束缚"**成为不断开拓人类科技边界的工程师们的共识。灵活到爆炸的业务需求导致了对灵活的数据表现形式的强烈需求(关系,rang K/V, hash K/V,全文检索, elastic search, data warehousing, inmem cache,数据流,文件),新的问题当然也随之而来:我们如何保证所有的数据在不同的储存层之间保持consistent呢?如何解决这个问题成为了分布式系统设计的关键因素之一,而我们貌似只有2种选择:

1. 基于XA(eXtended Architecture), 2PC的, 不能failure tolerance的强consistency
2. 或者no consistency at all, 容忍多种的数据可能不一样。

我们需要一个更好的解。。。。(这条线也暂且按下不表)

试验! 还是试验

一方面,大数据的爆发,软件的广泛应用使得全社会自动化决策程度持续增高,而数据相关Bug的造成的cost也随之增长,一个bug毁一个公司的成为越来越逼真的故事,数据的corruption为企业无法容忍的错误,这使得human fault tolerance(即软件bug造成数据的永久损伤)成为系统设计需要考虑的重中之重。形式化验证的高cost使得这项技术迟迟不能在工业界得到大量应用,最终导致了Replayable构架的流行,即:用最简单和形式化验证过核心算法的framework/通用基础云设施等来收集原始数据,让只是凭借简陋的testing验证过的业务逻辑代码可以任意重复利用原始数据rerun,这样任何时候发现bug都可以通过重跑系统来重写派生的业务数据。

Replayable构架的能量不仅仅是human fault tolerance,如果我们拓展"什么是错误"的定义,那么当第二版科学模型的效果比第一版好,那么第一版就可以看作是"bug"。所以很"自然"的,一个可以fix&rerun的replayable构架系统也必然可以同时跑多版本来做多版本实验。商业需求的爆炸造就了商业决策的爆炸,自动化的决策执行使得决策的影响越来越大,各个公司决策的竞争,不再只是依赖某大佬的点和经验,而是需要试验并拿出数据支持自己的想法。Data Driven, Metrics Driven Decision Making,是现代高科技公司决策层高管的生命线;一个能用更小的cost来同时试验多个idea,得到试验数据来验证想法的公司,打败竞争对手的几率会大大增加。这

而老一代的message queue: JMS, ActiveMq, 甚至云上的比如AWS SQS, 都无法做到replay, 消息一旦被消费, 就会自动dequeue消失的无影无踪....

我们需要一个更好的解。。。。

新的时代和它给的解

大数据处理这个领域本身, 正在发生着一些革命性的变化。。。

首先, 当数据库开始变的像message queue, 很多数据库比如DyanamoDB, 开始提供Change Data Capture把对表的更改变成Data Stream供别的系统使用; 另一方面, message queue开始变的像一个可以持久化数据的数据库, 当我们也有了可以从任意点replay的可以长期存储的Stream(kafka), Database和Message queue的概念开始变的模糊。Kafka的出现, 直接解决了replayable的数据框架的问题。但它的意义不仅如此。

多数据存储之间的consistency的问题, 本质上与分布式数据库的replication问题一致, 而分布式数据库的replica可以通过total order broadcast, 来广播主节点的WriteAheadLog或者changelog到replica节点, 由于total order broadcast保证了所有replica节点收到的message是绝对相同的顺序, 那么只要replica节点按照受到message的顺序作出对数据相同的更改, 我们就实现了分布式数据库对数据的replication, 而WriteAheadLog或changelog本质就是data stream; 如果我们把数据库的total order broadcast的这项技术应用到分布式业务系统的设计中去, 那么我们就可以解决分布式系统中最难点一个问题: 消息时序, 而kafka可以说为分布式数据(change)提供了一个logic clock和total order, 任何数据生产者只需要把数据输入Kafka, 任何Kafka的consumer都保证会以相同的顺序接收同一个partition上的message, 同时利用Kafka的replayable特性, 数据可以写一次, 而重新以任意形式apply在其他存储形式上(cache, elastic search, datawarehouse, keyvalue DB等)。这样, 我们就解决了多存储的一致性问题。

更进一步, 如果我们在replay message的时候, 不是简单的原原本本的照抄replica, 而是加入业务逻辑处理, 把任何业务逻辑的执行, 都看成是从raw data得到的derived data问题, 那么把业务逻辑加入到流中, 我们就得到了一个经过变化的流, 如果我们把这个流写入到一个表里, 而这个表又可以产生流(通过类似Change Data Capture这种技术, 比如AWS DynamoDB Stream), 而我们又有多个业务processor接收这个新流, 经过处理, 写入Kafka这种流系统, 或者写入可以生成流的table.... 如此循环, 我们就得到了一个以Kafka的message index为logical clock的全局eventually consistent的分布式流处理系统。

顺着这个思路, 任意一个公司的数据处理系统的本质, 都可以看成是一个超大的流处理网络。这个思路把一个分布式数据库维护consistency的各种手段, write ahead log, change log shipment, state replication, materialized view分解开来应用到分布式业务系统的设计里, 这就是名为**Turn Database Inside Out, Database Unbundled**的设计和构架大型分布式企业级应用体系的新思路。

从数据库中来, 到数据库中去, 从数据流来到数据库中去, 从对数据库的更改来, 到流或数据库中去。当source, computation和sink全都在云上, 我们的业务逻辑本质上就是数据库产生任何变化之后trigger的逻辑, 而这就是**Streaming processor as a database**的思路。

当我们有了replayable的source, 自动failover的计算节点(由K8S这种保活), 当我们写入的sink也已和我们的整个流框架整合, 而使用者只需要提供数据处理逻辑的时候, framework就可以通过插入logical id作为fencing token来最终实现sink的commit的幂等(特别是当所有资源都serverless化由云服务商提供), 这就最终使得点到点的exactly once处理成为可能, 克服了end2end argument对于中间件的限制(具体限制看这个答案, [阿莱克西斯: 为什么Akka\(Actor模型\)在中国不温不火?](#)), 因为现在framework不再是中间件了, 而是从end2end都由framework本身全权控制, 用户的程序只是插入到数据流的计算节点中的一步而已; 这个发展, 解决了分布式框架usability的本质问题, 而这也从本质上消灭了Lambda Architecture要解决的问题(即用不准确的Streaming提供更早的estimation, 而利用稳定consistent的batch处理来fix streaming的不准确问题), 因为我们现在也可以用streaming来得到consistent的计算结果了!

(其实没有解决, 只是我们发现batch和stream本质是一样的, 而stream可以做到跟batch至少一样好, 这样我们就没必要维护两套系统实现了。Latency和Consistency的trade-off, 本质上是

Streaming System这本书

新的时代,新的infrastructure革命,使我们创造新的抽象,来更本质的理解分布式数据处理,甚至来屏蔽分布式系统的细节复杂度成为可能。

而Streaming System这本书,则可以说是对这一发展的总结和详述。本书的最大贡献之一,即书的前半部分,是给出了一套构思,建造和分析分布式数据处理系统的思维体系(注意不仅仅是流式处理),在一边把Lambda Architecture的思路按在地上不断摩擦的同时,作者论证了Streaming处理系统其实是batch处理系统的超集,即batch只是固定process/event time window下,利用atWaterMark trigger的数据处理模型而已。所以基本上你可以用这套理论体系来分析目前的几乎“任何分布式数据处理系统”。去思考他们的能力和在各个关键节点所做出的trade-off。

这套思维体系源自Google内部10几年的大数据,高可用,高scalability的分布式应用系统搭建,总结为DataFlow Model这篇论文(你如果不想看这本书,那么请务必读一下这篇论文),并且实现和开源成了一套“programming model, API and portability layer”,即Apache Beam;且Flink作为目前数据处理的行业标杆,它的成功之一也和吸取了这套思想体系并提供了“恰到好处”的表达力有关(关于什么是表达力,我要安利的下一本书会着重讲解,敬请期待喵喵)。(注意:这并不是一本讲Beam的书,用此书提供的思想体系来解分布式数据处理问题,类似于relation algebra来解关系模型处理问题,而Beam Programming model是这套思维体系的具现化工具,就好比SQL是relation algebra的一种具现化语言。注意去通过event time, watermark, trigger, accumulation model等概念去理解核心的思维model,而不要纠结于Beam的API)

这本书的最大贡献之二,即书的后半部分,详细论述了**Stream and Table relativity**,即流/表相对论(名字起的好呀),通过分析MapReduce这种基于整块数据的系统(其实你可以把整块数据想象成不带索引的表),和数据库内部的materialized view的实现,作者论证了table和stream只是一枚硬币的两面,数据库的基于SQL生成的materialized view, MapReduce把大块的数据应该计算生成新的derived数据,在外部看来是table到table,是大数据块到大数据块,然而其内部实现却是要把数据块变成数据流,对流进行流到流操作(filter/map)等,对流进行流到表操作(groupby/aggregate)生成中间的隐式表,然后再用一个full window trigger把隐式表转化为数据流,如此循环最终变成最终表的操作而已。这和第一部分的Beam model完整match,只是在Beam中,是从流开始到流或者表,Stream作为显示元素,table作为隐式元素,而mapreduce和materialized view是从table到table, table作为显示元素,隐藏了中间的数据流和隐式表。

基于Stream and Table relativity,我们进一步清除了Streaming SQL,即用declarative的SQL来表达流式处理系统意图的障碍,作者从而提出把时间这个概念作为first class citizen加入到relation algebra里,把relation algebra发展成为time-varying relation,并提出如何使得SQL可以被拓展成为可以跑在流式数据里的Streaming SQL,并用Bean Model描述了可以如何把Stream SQL翻译回Beam的programming model,这样就打通了从SQL到Beam的通道,那么在beam又可以被底层runner(flink or Spark)执行的情况下,一套标准的Stream SQL的出现成为可能。(但是现在这一切还都是构想,没有任何framework可以完整支持书中构思的Streaming SQL, Apache Calcite目前只支持一部分,且注意Calcite已于flink集成,所以使用flink的玩家可以在flink上跑Calcite支持的SQL)

虽然还没有实现,但是利用作者提供的整体的Beam思想体系,你会发现理解分布式流式系统变的异常简单。

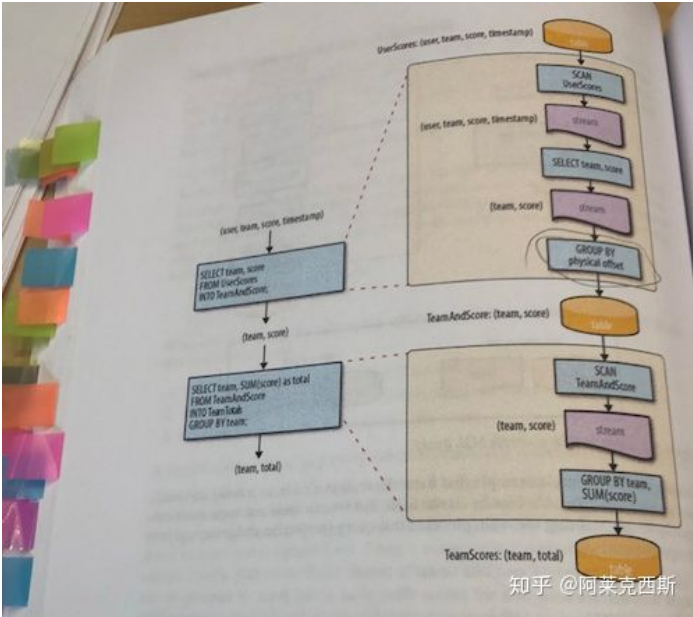
At the end of the day, there is no stream, no table, no difference on batch or streaming, there is only simply data and the logic to process the data, that's all.

本书的其他内容还包括对各种分布式框架特性,优点缺点的简述。对exactly once message processing这个分布式系统设计的大难题的解法的详述。和对乱序的event处理的各种例子(比如session window)。高潮迭起,精彩不断呀!

注意: 书里的图很不清楚。。。虽然作者试图用动图的形式把事情讲清楚,但是同时把2个时间event time和process time放在一起让读者去理解,还是非常的难懂。请一定要把图看懂,不然越到后边越难理解书里的内容。书里的前边很多东西,由于不是此章的重点,都没有讲明白,然

再回头去看前边的东西,相信理解会更深入。(是的。。我就读了两遍,第一遍只是觉得好像学了很多东西,但是细想又感觉思维非常混乱,然后第二遍读完感觉一切都明镜一般了。。。)

内容是彩色的哟~



没注意就写了这么多,好累。。。感觉这是我写的最累的一篇了吧,感觉1年都不想再写文章了。。。。

能看到这里的都是真爱。



(请点赞以示鼓励) (手动可怜)

编辑于 2018-09-02

程序员 分布式系统 Stream Processing

▲ 赞同 1.5K ▼ ● 78 条评论 ➤ 分享 ★ 收藏 ...

文章被以下专栏收录

 **一个书魔程序员的读书简评**
我是阿莱克西斯, 10+年关于高并发, 大数据处理的技术经验, 常年与内部科学家合...

关注专栏



Building Reactive Microservices in Java

Asynchronous and Event-Based

OpenShift简介——Building Reactive Microservices in...

wjvifkd

MIT 6.824 分布式课程 Raft日志分发实现

好歹我也是在写了两年的程序员了,结果没想到做这个Lab这么困难,本来以为代码写完后最多简单改动下就能通过测试,但是实际几乎每个测试用例都能发现新问题. 首先总结下,遇到了以下几个问...

buckethead



阿里P8架构师谈（分布式系列）：分布式事务的特征、原...


优知学院

78 条评论

切换为时间排序

写下你的评论...

精选评论 (1)

 阿莱克西斯 (作者) 回复 彭飞

5 个月前

DDIA就非常不错呀, 还可以顺着每章的引用去深入的学习! , hadoop the definition guide, hbase the definition guide, zookeeper, 高性能spark那本, 这些讲工具的书也很容易理解, 不过这些书一般很长, 内容不够精炼. 想深入研究, 不是浅尝辄止的话, 读论文打好基础对之后学什么都会有加速效果. 这样的话可以看paxos, google的大数据三篇, spanner, flink和spark的论文. 然后不管看什么论文, 有不熟悉的地方, 就根据引用往回看可以把一条线都学了. 分布式算法理论的好书不多, 感觉Distributed Systems: An Algorithmic Approach, Second Edition, 这本还不错.

29 查看回复


评论 (78)

 彭飞

5 个月前

您好, 能推荐几本分布式系统入门的书吗?

2

 阿莱克西斯 (作者) 回复 彭飞

5 个月前

DDIA就非常不错呀, 还可以顺着每章的引用去深入的学习! , hadoop the definition guide, hbase the definition guide, zookeeper, 高性能spark那本, 这些讲工具的书也很容易理解, 不过这些书一般很长, 内容不够精炼. 想深入研究, 不是浅尝辄止的话, 读论文打好基础对之后学什么都会有加速效果. 这样的话可以看paxos, google的大数据三篇, spanner, flink和spark的论文. 然后不管看什么论文, 有不熟悉的地方, 就根据引用往回看可以把一条线都学了. 分布式算法理论的好书不多, 感觉Distributed Systems: An Algorithmic Approach, Second Edition, 这本还不错.

29

 彭飞 回复 阿莱克西斯 (作者)

5 个月前

非常感谢!

赞


展开其他 2 条回复

 xxf09th

5 个月前

看到了封面上的咸鱼

9

 吴志宇

5 个月前

前几天阿里的 Blink 团队在一次 meetup 中, 大沙提出的观点是可以 SQL 统一流和批处理的计算并且阿里已经在着手做相关的工作了.

期待开源... (^ω^)

👍 赞



天空龙 回复 吴志宇

5 个月前

我在谷歌也是作这个的，但是因为业务重点的原因，阿里进展比我们快。

👍 赞

查看全部 7 条回复



任弘迪

5 个月前

ddia就展望过用数据流实现业务系统（比如订单，支付这类）的各种好处，但这方面实践似乎很少看到。

👍 赞



罗宸

5 个月前

感觉跟FRP殊途同归？ 只是前端没有一致性的问题所以抽象起来简单得多。

👍 赞



阿莱克西斯 (作者) 回复 罗宸

5 个月前

FRP还没怎么去了解过，不知道呀 ...

👍 赞



崔扬 回复 罗宸

4 个月前

FRP是什么？缩写现在满天飞啊

👍 赞

展开其他 1 条回复



just

5 个月前

之前玩confluent公司的kstream，ksql的时候就很有这种感觉。看完有两个问题：一个是total order，kafka其实只支持单partition的全序吧，在上下游使用的时候会不会有efficiency或者scale的问题？

还有一个是逻辑时钟，"framework就可以通过插入logical id作为fencing token来最终实现sink的commit的幂等"这句话不是很理解，是说sink的外围系统也是一个消费者？

👍 2



阿莱克西斯 (作者) 回复 just

5 个月前

没错，单partition才是全序，可能忘了提。待我修缮一下；

幂等sink，我的意思是：其实sink只有能够用一个类似transactionId来实现幂等的系统才可以，一般由user控制sink端的话，table设计和integration的时候，需要用户自己在比如数据库的表多一个txID，再commit到sink的时候用保证用相同的txID去commit相同的数据来做到去重(比如storm的trident)。现在framework可以直接帮你做这个，就很便利。

👍 1



just 回复 阿莱克西斯 (作者)

5 个月前

嗯，幂等这块我理解，但是framework也只能帮我们做到基于framework层面的sink吧，比如我一个业务系统既是consumer也是producer，走一个consume-transform-produce，如果接入到外部的系统(比如一个数据库)的时候是不是还是需要自己来保证，这样理解对么？

👍 赞

展开其他 2 条回复



qzeng2490

5 个月前

楼主辛苦了！

👍 赞



- 👍 赞
- 阿莱克西斯 (作者) 回复 黄赞

5 个月前

那本可以网上直接下载吧? 那本也不错的

👍 1
- 东东

5 个月前

kafka 和 flink 就是往这书里面讲的正确方向进化的。

👍 2
- Charles Wei

5 个月前

有电子书么, 我看一本要300RMB

👍 1
- cancy 回复 Charles Wei

1 个月前

哪里有卖的啊?

👍 赞
- Charles Wei 回复 cancy

1 个月前

亚马逊

👍 赞
- buyuan

5 个月前

rx思路

👍 赞
- andwxh

5 个月前

太感动了, 正需要! 我是几年前开始写mr和storm程序, 最近几年心心念念找个好方法合并一下, 怎么合倒是想清楚了, 但是感觉还是欠缺火候, 今年剩下的时间本来计划开搞了, 正好让我来补一下成体系的知识

👍 1
- 阿莱克西斯 (作者) 回复 andwxh

5 个月前

(^ω^) 能对您有帮助, 我也很开心呀

👍 赞
- andwxh 回复 阿莱克西斯 (作者)

5 个月前

客气了哈, 已下单目测下周三四能到手, 等读完了回来补书评💎*(●▽●)*,💎

👍 赞
- Steven Zed

5 个月前

好像国内没有

👍 赞
- 天空龙

5 个月前

赞赞赞。哈哈Tyler 和 Reuven天天在我面前晃过

👍 1
- 阿莱克西斯 (作者) 回复 天空龙

5 个月前

羡慕


👍 赞
- 向上的蜗牛

5 个月前

很吃惊, 我前几天写一个pipeline开源工具也有 replayable 功能。虽然是自己想出来的, 看来这种需求是很普遍。[kkyon/databot](#)

非常赞同Kafka解决多存储一致性的说法。我最近见到的工业界的实例，是使用HRegion在单机存储数据，使用Kafka存写数据，单机pull写到单机并利用HRegion持久化，从而实现最终一致性，在此基础上定制了data model为一些特殊use case服务。

👍 1

南慕伦

5 个月前

这就是数学系毕业的高级程序员转产品经理的视角吗，非常先进和广阔，叹为观止，难以望其项背.....先下单为敬！


👍 2

阿莱克西斯 (作者) 回复 南慕伦

5 个月前

噗... (转PM)戳中痛点...


👍 赞

Ryan Chang

5 个月前

大佬看书速度好快

👍 赞

wn wn


5 个月前

写得好乱，希望这本书不要跟你这个简评一样，

“但是Storm对系统设计提出了新的问题; 由于分布式环境的不稳定性， Storm当时无法保证生成determinitic的结果，无法保证计算的consistency，从而无法对application层提供足够的正确性支持。”

既然都知道要保证计算的consistency，你就不知道怎么保证阅读的consistency么，这么几个简单的词真想不起中文怎么写么...


👍 赞

阿莱克西斯 (作者) 回复 wn wn

5 个月前

因为这些词本就不想用中文，对于很多读者来说，这些专用词汇用英文才是consistency (^ω^)

👍 6

wn wn 回复 阿莱克西斯 (作者)

3 个月前

哪些是专用词汇？ application 吗？ 哪些读者会认为这些个单词是专用词汇，你说“windows”是专用词汇我觉得正常，你说你是外国人写的时候脑子转不过来，我可以理解，我不是对中英文杂写不适，毕竟很多最新技术资料都只能看英文，我就是对你这种用英语四级词汇来强行增加阅读困难的行为表示不解

👍 赞

展开其他 3 条回复