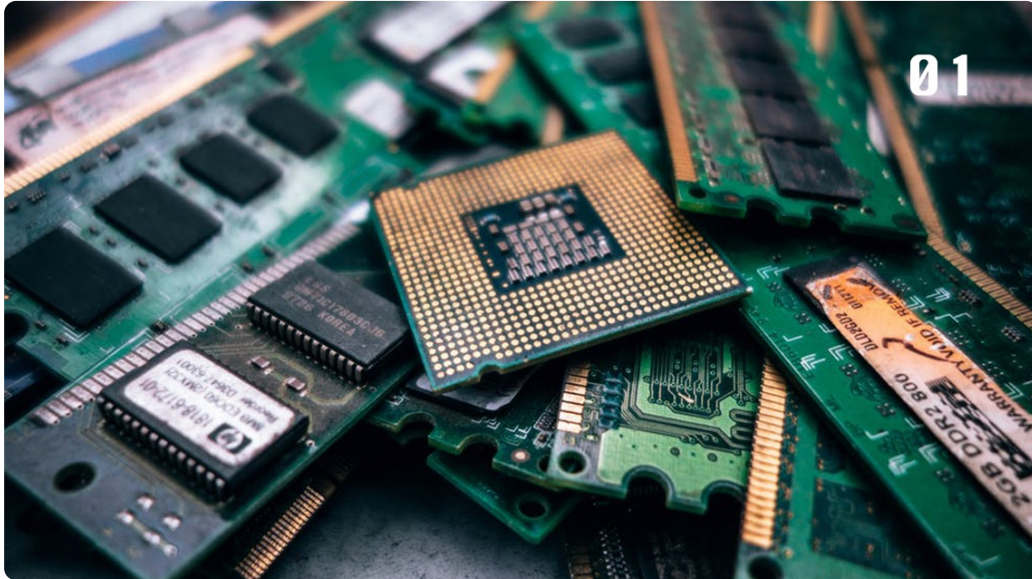


## 01 | 为什么MapReduce会被硅谷一线公司淘汰？

蔡元楠 2019-04-17



00:00

讲述：蔡元楠 大小：9.68M

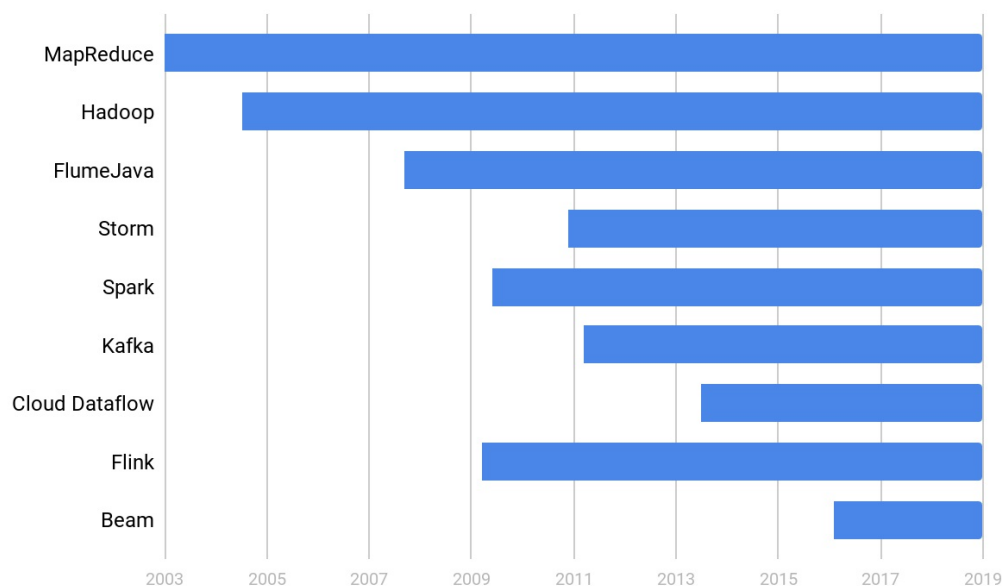
10:33

我有幸几次与来 Google 参观的同行进行交流，当谈起数据处理技术时，他们总是试图打探 MapReduce 方面的经验。

这一点让我颇感惊讶，因为在硅谷，早已没有人去谈论 MapReduce 了。

今天这一讲，我们就来聊聊为什么 MapReduce 会被硅谷一线公司淘汰。

我们先来沿着时间线看一下超大规模数据处理的重要技术以及它们产生的年代。



我认为可以把超大规模数据处理的技术发展分为三个阶段：石器时代，青铜时代，蒸汽机时代。

## 石器时代

我用“石器时代”来比喻 MapReduce 诞生之前的时期。

数据的大规模处理问题早已存在。早在 2003 年的时候，Google 就已经面对大于 600 亿的搜索量。

但是数据的大规模处理技术还处在彷徨阶段。当时每个公司或者个人可能都有自己的一套工具处理数据。却没有提炼抽象出一个系统的方法。

## 青铜时代

2003 年，MapReduce 的诞生标志了超大规模数据处理的第一次革命，而开创这段青铜时代的就是下面这篇论文《MapReduce: Simplified Data Processing on Large Clusters》。

### MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

*Google, Inc.*

#### Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is inspired by the *map* and *reduce* primitives present in Lisp and many other functional languages. We realized that most of our computations involved applying a *map* op-

杰夫 (Jeff Dean) 和桑杰 (Sanjay Ghemawat) 从纷繁复杂的业务逻辑中，为我们抽象出了 Map 和 Reduce 这样足够通用的编程模型。后面的 Hadoop 仅仅是对于 GFS、BigTable、MapReduce 的依葫芦画瓢，我这里不再赘述。

## 蒸汽机时代

到了 2014 年左右，Google 内部已经几乎没人写新的 MapReduce 了。

2016 年开始，Google 在新员工的培训中把 MapReduce 替换成了内部称为 FlumeJava (不要和 Apache Flume 混淆，是两个技术) 的数据处理技术。

这标志着青铜时代的终结，同时也标志着蒸汽机时代的开始。

我跳过“铁器时代”之类的描述，是因为只有工业革命的概念才能解释从 MapReduce 进化到 FlumeJava 的划时代意义。

Google 内部的 FlumeJava 和它后来的开源版本 Apache Beam 所引进的统一的编程模式，将在后面的章节中为你深入解析。

现在你可能有一个疑问：为什么 MapReduce 会被取代？今天我将重点为你解答。

### 高昂的维护成本

使用 MapReduce，你需要严格地遵循分步的 Map 和 Reduce 步骤。当你构造更为复杂的处理架构时，往往需要协调多个 Map 和多个 Reduce 任务。

然而，每一步的 MapReduce 都有可能出错。

为了这些异常处理，很多人开始设计自己的协调系统（orchestration）。例如，做一个状态机（state machine）协调多个 MapReduce，这大大增加了整个系统的复杂度。

如果你搜“MapReduce orchestration”这样的关键词，就会发现有很多书，整整一本都在写怎样协调 MapReduce。

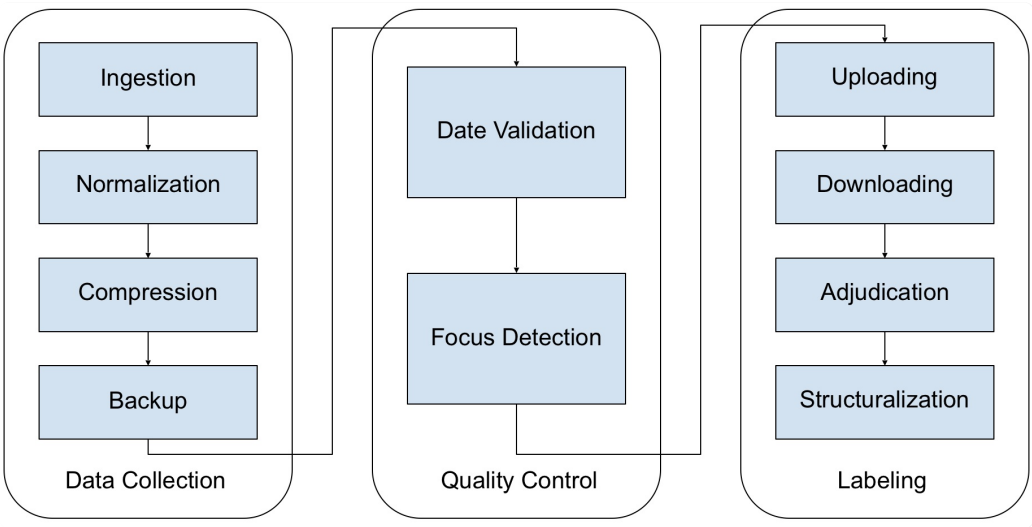
你可能会惊讶于 MapReduce 的复杂度。我也经常会看到一些把 MapReduce 说得过度简单的误导性文章。

例如，“把海量的××数据通过 MapReduce 导入大数据系统学习，就能产生××人工智能”。似乎写文的“专家”动动嘴就能点石成金。

而现实的 MapReduce 系统的复杂度是超过了“伪专家”的认知范围的。下面我来举个例子，告诉你 MapReduce 有多复杂。

想象一下这个情景，你的公司要**预测美团的股价**，其中一个重要特征是活跃在街头的美团外卖电动车数量，而你负责**处理所有美团外卖电动车的图片**。

在真实的商用环境下，为了解决这个问题，你可能至少需要 10 个 MapReduce 任务：



首先，我们需要搜集每日的外卖电动车图片。

数据的搜集往往不全部是公司独自完成，许多公司会选择部分外包或者众包。所以在**数据搜集**（Data collection）部分，你至少需要 4 个 MapReduce 任务：

1. 数据导入（data ingestion）：用来把散落的照片（比如众包公司上传到网盘的照片）下载到你的存储系统。
2. 数据统一化（data normalization）：用来把不同外包公司提供过来的各式各样的照片进行格式统一。
3. 数据压缩（compression）：你需要在质量可接受的范围内保持最小的存储资源消耗。
4. 数据备份（backup）：大规模的数据处理系统我们都需要一定的数据冗余来降低风险。

仅仅是做完数据搜集这一步，离真正的业务应用还差得远。

真实的世界是如此不完美，我们需要一部分数据质量控制（quality control）流程，比如：

1. 数据时间有效性验证（date validation）：检测上传的图片是否是你想要的日期的。
2. 照片对焦检测（focus detection）：你需要筛选掉那些因对焦不准而无法使用的照片。

最后才到你负责的重头戏——找到这些图片里的外卖电动车。而这一步因为人工的介入是最难控制时间的。你需要做 4 步：

1. 数据标注问题上传（question uploading）：上传你的标注工具，让你的标注者开始工作。
2. 标注结果下载（answer downloading）：抓取标注完的数据。
3. 标注异议整合（adjudication）：标注异议经常发生，比如一个标注者认为是美团外卖电动车，另一个标注者认为是京东快递电动车。
4. 标注结果结构化（structuralization）：要让标注结果可用，你需要把可能非结构化的标注结果转化成你的存储系统接受的结构。

这里我不再深入每个 MapReduce 任务的技术细节，因为本章的重点仅仅是理解 MapReduce 的复杂度。

通过这个案例，我想要阐述的观点是，因为真实的商业 MapReduce 场景极端复杂，像上面这样 10 个子任务的 MapReduce 系统在硅谷一线公司司空见惯。

在应用过程中，每一个 MapReduce 任务都有可能出错，都需要重试和异常处理的机制。所以，协调这些子 MapReduce 的任务往往需要和业务逻辑紧密耦合的状态机。

这样过于复杂的维护让系统开发者苦不堪言。

## 时间性能“达不到”用户的期待

除了高昂的维护成本，MapReduce 的时间性能也是个棘手的问题。

MapReduce 是一套如此精巧复杂的系统，如果使用得当，它是青龙偃月刀，如果使用不当，它就是一堆废铁。不幸的是并不是每个人都是关羽。

在实际的工作中，不是每个人都对 MapReduce 细微的配置细节了如指掌。

在现实中，业务往往需求一个刚毕业的新手在 3 个月内上线一套数据处理系统，而他很可能从来没有用过 MapReduce。这种情况下开发的系统是很难发挥好 MapReduce 的性能的。

你一定想问，MapReduce 的性能优化配置究竟复杂在哪里呢？

我想 Google 500 多页的 MapReduce 性能优化手册足够说明它的复杂度了。这里我举例讲讲

MapReduce 的分片 (sharding) 难题, 希望能窥斑见豹, 引发大家的思考。

Google 曾经在 2007 年到 2012 年间做过一个对于 1PB 数据的大规模排序实验, 来测试 MapReduce 的性能。

从 2007 年的排序时间 12 小时, 到 2012 年的排序时间缩短至 0.5 小时。即使是 Google, 也花了 5 年的时间才不断优化了一个 MapReduce 流程的效率。

2011 年, 他们在 Google Research 的博客上公布了初步的成果。



### Sorting Petabytes with MapReduce - The Next Episode

Wednesday, September 7, 2011

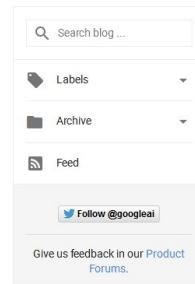
Posted by Grzegorz Czajkowski, Marián Dvorský, Jerry Zhao, and Michael Conley, Systems Infrastructure

Almost three years ago we announced [results of the first ever 'petasort'](#) (sorting a petabyte-worth of 100-byte records, following the [Sort Benchmark](#) rules). It completed in just over six hours on 4000 computers. Recently we repeated the experiment using 8000 computers. The execution time was 33 minutes, an order of magnitude improvement.

Our sorting code is based on [MapReduce](#), which is a key framework for running multiple processes simultaneously at Google. Thousands of applications, supporting most services offered by Google, have been expressed in MapReduce. While not many MapReduce applications operate at a petabyte scale, some do. Their scale is likely to continue growing quickly. The need to help such applications scale motivated us to experiment with data sets larger than one petabyte. In particular, sorting a ten petabyte input set took 6 hours and 27 minutes to complete on 8000 computers. We are not aware of any other sorting experiment successfully completed at this scale.

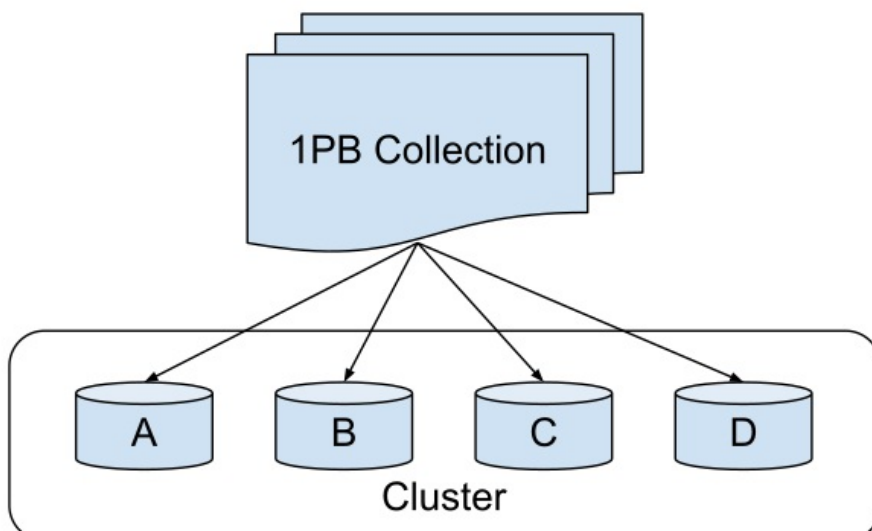
We are excited by these results. While internal improvements to the MapReduce framework contributed significantly, a large part of the credit goes to numerous advances in Google's hardware, cluster management system, and storage stack.

What would it take to scale MapReduce by further orders of magnitude and make processing of such large data sets efficient and easy? One way to find out is to join Google's systems infrastructure team. If you have a passion for distributed computing, are an expert or plan to become one, and feel excited about the challenges of exascale then definitely consider applying for a [software engineering position](#) with Google.



其中有一个重要的发现, 就是他们在 MapReduce 的性能配置上花了非常多的时间。包括了缓冲大小 (buffer size), 分片多少 (number of shards), 预抓取策略 (prefetch), 缓存大小 (cache size) 等等。

所谓的分片, 是指把大规模的的数据分配给不同的机器 / 工人, 流程如下图所示。

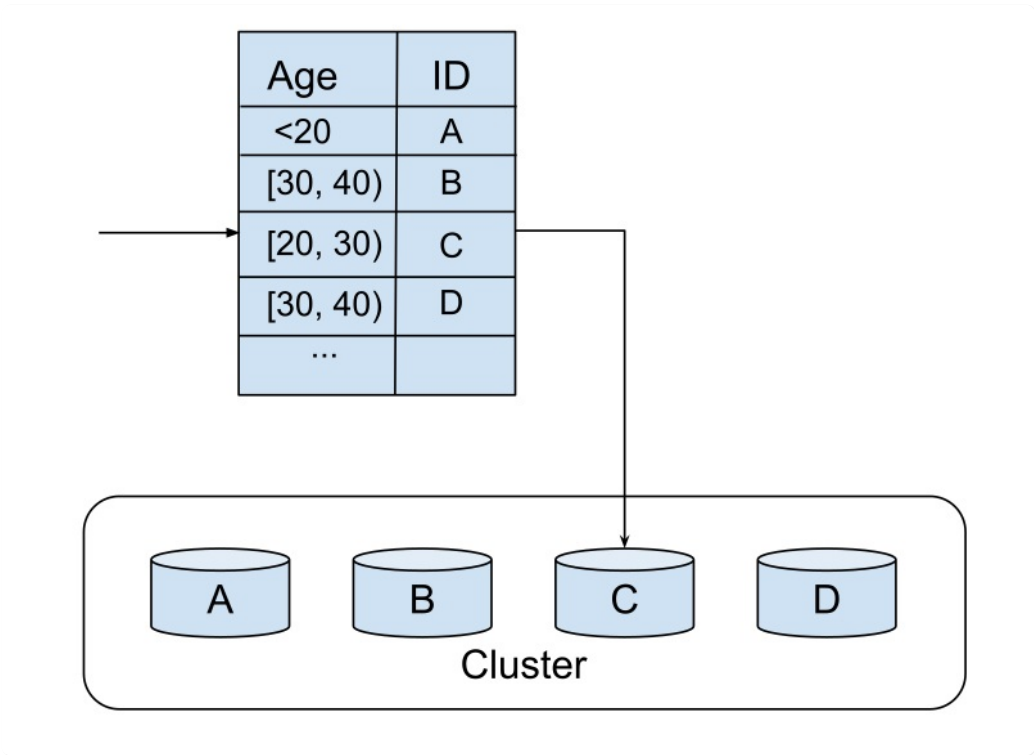




选择一个好的分片函数（sharding function）为何格外重要？让我们来看一个例子。

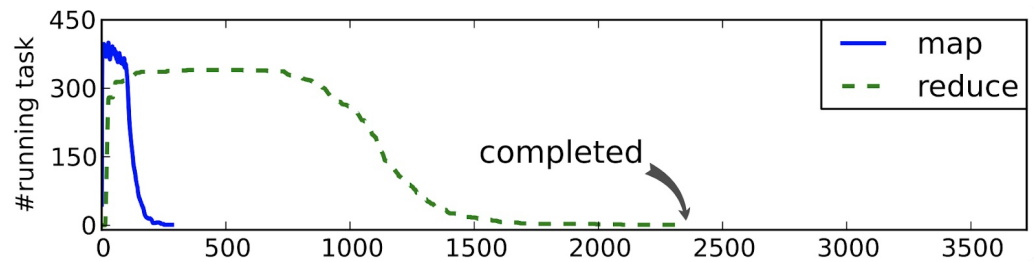
假如你在处理 Facebook 的所有用户数据，你选择了按照用户的年龄作为分片函数（sharding function）。我们来看看这时候会发生什么。

因为用户的年龄分布不均衡（假如在 20~30 这个年龄段的 Facebook 用户最多），导致我们在下图中 worker C 上分配到的任务远大于别的机器上的任务量。



这时候就会发生掉队者问题（stragglers）。别的机器都完成了 Reduce 阶段，只有 worker C 还在工作。

当然它也有改进方法。掉队者问题可以通过 MapReduce 的性能剖析（profiling）发现。如下图所示，箭头处就是掉队的机器。



图片引用：Chen, Qi, Cheng Liu, and Zhen Xiao. “Improving MapReduce performance using smart speculative execution strategy.” IEEE Transactions on Computers 63.4 (2014): 954-967.

回到刚刚的 Google 大规模排序实验。

因为 MapReduce 的分片配置异常复杂，在 2008 年以后，Google 改进了 MapReduce 的分片功能，引进了动态分片技术 (dynamic sharding)，大大简化了使用者对于分片的手工调整。

在这之后，包括动态分片技术在内的各种崭新思想被逐渐引进，奠定了下一代大规模数据处理技术的雏型。

## 小结

这一讲中，我们分析了两个 MapReduce 之所以被硅谷一线公司淘汰的“致命伤”：高昂的维护成本和达不到用户期待的时间性能。

文中也提到了下一代数据处理技术雏型。这就是 2008 年左右在 Google 西雅图研发中心诞生的 FlumeJava，它一举解决了上面 MapReduce 的短板。

另外，它还带来了一些别的优点：更好的可测试性；更好的可监控性；从 1 条数据到 1 亿条数据无缝扩展，不需要修改一行代码，等等。

在后面的章节中，我们将具体展开这几项，通过深入解析 Apache Beam（FlumeJava 的开源版本），揭开 MapReduce 继任者的神秘面纱。

## 思考题

如果你在 Facebook 负责处理例子中的用户数据，你会选择什么分片函数，来保证均匀分布的数据分片？

欢迎你把答案写在留言区，与我和其他同学一起探讨。

如果你觉得有所收获，也欢迎把文章分享给你的朋友。

 极客时间

# 大规模数据处理实战

Google 一线工程师的大数据架构实战经验



蔡元楠  
Google Brain 资深工程师

新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

©



由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。

Ctrl + Enter 发表

0/2000字

提交留言



\_CountingStars 置顶

把年龄倒过来比如 28 岁 变成 82 来分片

👍 52 2019-04-17

作者回复: 谢谢你的答案！这个答案很新颖啊，我觉得光从年龄这个问题上来讲，你的思路是可以把20多岁变成12、22、32、42等等。希望你能在以后遇到问题时也能保持这样创新思维，也希望你能继续留言，我们一起学习进步！



Codelife 置顶

我们最早采用的是哈希算法，后来发现增删节点太麻烦，改为带虚拟节点的一致性哈希环开处理，稍微复杂点，但是性能还好

👍 14 2019-04-17

作者回复: 谢谢你的答案！应该是一个很有经验的高级工程师了吧。使用Consistent hashing是可以很好地解决平均分配和当机器增减后重新hashing的问题。



maye 置顶

个人愚见：虽然MapReduce引擎存在性能和维护成本上的问题，但是由于Hive的封装使其适用性很广泛，学习成本很低，但是实际使用过程中和Spark等相比性能差太多了。不过对于计算引擎模型的理解方面，MapReduce还是一个很经典的入门模型，对于未来迁移到其他计算引擎也是有很大帮助的。还有一个个人问题：不知道蔡老师对于流计算和批处理的关系是怎么看待的？流计算有可能完全取代批处理么？

关于思考题：问题的核心点在于Reducer Key是否倾斜，个人认为可以按照update\_time之类的时间字段进行分片处理。

👍 6 2019-04-17

作者回复: 你好Maye，谢谢你的留言与提问！

第一问我也说说我的愚见吧。关于流处理和批处理的关系我更倾向于批处理可以算是流处理的一个子集吧。我们可以这么抽象地看，流计算所处理的都是无限数据集，而我们从中按照时间窗口抽取一小段出来的话，这一小段有边界的数据集其实也就是批处理所处理的数据集了。所以说批处理算是流处理的一个子集吧。但是现在流计算中两大问题：1）Exactly once delivery 2）message order，还没有非常完美的解决方案，但是我相信可以攻克的。所以未来趋势还是趋于统一。现在Google所推出的Apache Beam项目其实也是想解决这样一个问题，统一批处理和流处理的编程接口。更详细的内容我会在后面的章节展开讲解。

思考题你也看到了问题的本质，就是能找到趋于平均分配的分片处理方式。

欢迎你继续留言提问，一起交流学习进步！



王伟 置顶

你好！我工作中遇到这样的场景：会员在我们平台注册，信息会保存在对应商家的商家库中，现在需要将商家库中的信息实时的同步到另一台服务的会员库中，商家库是按照商家编号分库，而且商家库和会员库没在同一台服务器部署。想请教一下，像这种我们如何做到实时同步？

👍 5 2019-04-17

作者回复: 你好王伟！首先感谢你的提问！

我不确定你所说的实时同步是想表达Eventual Consistency还是Strong Consistency，那我就争对



两个都说说自己的愚见吧。

因为会员信息都会保存在商家库中，所以这里我假设商家库的信息可以作为source of truth。

如果你指的是Eventual Consistency的话，可以在会员更新商家库的同时将会员信息利用Pub/Sub发送给会员库去更新。考虑到Pub/Sub中间有可能会丢包，我们可以再建立一个定时任务每隔一段时间将全部商家库中的信息扫描一遍再更新到会员库中。当然具体的实现可以再作优化，因为商家库是按商家编号分库的，我们可以记录下哪些商家编号的表最近有更新我们就只扫描那些表，而不用扫描全局的表。

如果你指的是Strong Consistency的话，我们可以在中间再创建一个State Machine，记录是否两个库都同时更新了。在读取会员信息的时候，我们需要查询这个State Machine，只有当两个库同时都更新的时候才将会员信息返回。根据第九讲的CAP理论，这样的做法其实会牺牲掉Availability，也就是你的服务可用性。

当然具体的需求你会比我更了解，所以相信你能够从中做出设计上的取舍。也欢迎你继续留言提问，我们可以一起讨论学习进步！



alexgreenbar 置顶

赞一个，几乎每问必答，无论是否小白问题，很务实，具备高手风范！

👍 4 2019-04-17

作者回复: 😊 很多问题确实很有思考价值，我也学到了很多之前没考虑到的。



SpanningWings 置顶

还想到一个问题有关consistent hashing的。map reduce下层的GFS也没有采用consistent hashing来控制分片，这又是为什么？老师有空回答下吗？

👍 3 2019-04-18

作者回复: 再次看到了你的提问，感谢！

以下纯属个人愚见。

无论是MapReduce的partitioning，还是GFS的chunkservers，它们的设计思想都是将文件分割成固定大小的chunks来维护，而每一个chunk都会有一个deterministic的64位唯一标识符。这种设计思想是和consistent hashing不一样的，可以称为是Central Coordinator。

而历史原因也是存在的，GFS和MapReduce是分别在2003年和2004年公开论文的，而Distributed Hash Table这种思想，也就是Consistent hashing，是在2007年Amazon发表了Dynamo: Amazon's Highly Available Key-value Store这篇论文后被大家所广泛认同的。

最后我想说的是，设计一个通用架构给所有开发者使用和根据自身应用场景所设计出来的架构，它们的侧重点会有所不同。如果只是自身业务需要并且不需要太考虑时间复杂度，那当然可以自己来实现consistent hashing，毕竟hashing后取模和consistent hashing时每次都要计算环节点的时间复杂度肯定是不一样的。

希望这对你有所帮助，如果有所收获的话也欢迎你分享给朋友，谢谢！



cricket1981 置顶

如果不需要按某些字段做聚合分析，只是普通数据处理的话，直接用Round Robin分片即可。我想了解什么是“动态分片”技术？即使不用MR，其他大数据处理框架也需要用到“分片”，毕竟大数据的处理是“分而治之”，如何分才能分得好是关键。日常工作中经常遇到数据倾斜问题，也是由于分片不合理导致的。如果对于待处理的数据你了解到好办，知道用哪些字段作分片最合适，但如果遇到不熟悉的数

据你又该如何分片？而不是等到出现数据倾斜问题的时候才发现，进行分片修改再重跑呢？谢谢老师指教！

👍 3 2019-04-17

作者回复: Round robin确实能保证均匀但是有个很大的问题是没有容错。因为在分布式处理的时候数据处理顺序是“随机”的，可能是shard 1/2/3也可能是 shard 1/3/2，如果发现shard 2所有任务挂了（机器坏了）需要重试，如果有确定的sharding function很容易找出shard 2的任务，round robin的话就无法还原shard 2任务了。当然你可以说我再搞个数据库把round robin结果保存，但那样就更复杂了。



明翼 置顶

一般用户信息表都存在一个id，有的是递增的数字id，有的是类似uuid随机字符串，对于递增的直接对机器数量取余，如果是字符串通过比较均衡的hash函数操作后再对机器数量取余即可。

👍 3 2019-04-17

作者回复: 谢谢你的答案！这个答案不错。不过取余运算在机器有增减的时候会遇到麻烦，所有的用户必须重新取余运算一遍。Consistent Hashing可以很好地解决这个问题。欢迎你继续留言，我们一起学习进步！



摇山樵客™

年龄是值域在0-120（假定）之间的数值，难以分片的原因正是因为年龄的十位数权重过大，所以我觉得一切有效降低十位数权重的哈希算法应该都是可行的。

- 1.对于年龄ABC，比如倒置CBA，或 $(C * \text{大质数} + B * \text{较小质数} + C) \% \text{numPartitions}$ ，这类方法应该可以明显改善分布不均，但是对某些单一热点无解，比如25岁用户特别多；
- 2.随机分区，可做到很好均衡，对combine，io等优化不友好
3. 先采样+动态合并和拆分，实现过于复杂，效果可能不稳定

这是我的想法，请老师指正。

👍 4 2019-04-17

作者回复: 谢谢你的答案！你在每个答案里都分别给出这个答案所存在的不足，这一点我是非常赞赏的。在开发设计中没有哪个答案是特别完美的，我们能做的是分析哪一个才是最符合自身应用需求，进而改善。

1. 是的，倒置年龄的digit可以改善均分的问题，但是也存在hot spot的问题。
2. 我在其它的留言也回复过，随机分区的话还有一个缺点是当分区任务失败需要重新分区的时候，分区结果不再是deterministic的。
3. 总结得不错。

欢迎你继续留言，我们一起学习进步！



Destroy、

在评论在看到Consistent hashing，特地去搜索看了下，终于明白了。评论干货很多。。

👍 4 2019-04-17

作者回复: 哈哈有收获就好



渡码

认同您说的MR的局限性，因此建立在MR之上的hive有用武之地。面对不断出现的新框架我们怎么快速掌握它的设计，尤其是即便看文档也会觉得模棱两可，这时候有必要深入到源码中吗，您在这后续课程

中有没有相关的经验分享

👍 3 2019-04-17

作者回复: 第一个问题hive和MR是apple and orange，不太好对比吧。hive更类似于SQL。

第二个问题恰恰是这个专栏的重点，会教大家怎么解析框架的设计思路。



monkeyking

按照user\_id哈希或者给user\_id加一个随机数前缀

👍 3 2019-04-17

作者回复: 是对的思路！随机前缀这个我在另一个回复上也提到了，“真”随机会影响错误重试，因为没法还原当时的随机数，比如分片2的任务全部失败，找不到哪些是分片2了。



JensonYao

MapReduce是从纷繁复杂的业务逻辑中，为我们抽象出了 Map 和 Reduce这样足够通用的编程模型。

缺点：

1、复杂度高

当你构造更为复杂的处理架构时，往往进行任务划分，而且每一步都可能出错。而且往往比认为的复杂的多。

2、时间性能达不到用户要求

Google500 多页的 MapReduce 性能优化手册

1PB的排序从12小时优化到0.5小时花了5年

思考题：如果你在 Facebook 负责处理例子中的用户数据，你会选择什么分片函数，来保证均匀分布的数据分片？

由于没有过相关的经验，从网上查了下资料，常见的数据分片有1、hash 2、consistent hash without virtual node 3、consistent hash with virtual node 4、range based

文章中使用的方法就是range based方法，缺点在于区间大小固定，但是数据量不确定，所以会导致不均匀。

其他三种方法都可以保证均匀分布的数据分片，但是节点增删导致的数据迁移成本不同。

1、hash函数节点增删时，可能需要调整散列函数函数，导致大量的数据迁移

consistent hash是将数据按照特征值映射到一个首尾相接的hash环上，同时也将节点映射到这个环上。对于数据，从数据在环上的位置开始，顺时针找到的第一个节点即为数据的存储节点

2、consistent hash without virtual node 增删的时候只会影响到hash环上响应的节点，不会发生大规模的数据迁移。但是，在增加节点的时候，只能分摊一个已存在节点的压力；同样，在其中一个节点挂掉的时候，该节点的压力也会被全部转移到下一个节点

3、consistent hash with virtual node 在实际工程中，一般会引入虚拟节点（virtual node）的概念。即不是将物理节点映射在hash环上，而是将虚拟节点映射到hash环上。虚拟节点的数目远大于物理节点，因此一个物理节点需要负责多个虚拟节点的真实存储。操作数据的时候，先通过hash环找到对应的虚拟节点，再通过虚拟节点与物理节点的映射关系找到对应的物理节点。引入虚拟节点后的一致性hash需要维护的元数据也会增加：第一，虚拟节点在hash环上的问题，且虚拟节点的数目又比较多；第二，虚拟节点与物理节点的映射关系。但带来的好处是明显的，当一个物理节点失效是，hash环上多个虚拟节点失效，对应的压力也就会发散到多个其余的虚拟节点，事实上也就是多个其余的物理节点。在增加物理节点的时候同样如此。

引用blog: <http://www.cnblogs.com/xybaby/p/7076731.html>

所以这样看具体采用何种方式要结合其他的因素（显示场景，成本？），如何抉择我也不是很清楚。

👍 3 2019-04-17

作者回复: 线下做了研究了很好啊。这三个看起来都可以吧。一般场景我觉得可以选择复杂度低的第一种，后面的对于普通场景可能都有点overkill。



hua168

那现在还在用MapReduce的大数据软件怎么搞了？也会被慢慢淘汰？还需要学习吗？



3 2019-04-17

作者回复: 如果还没开始学，就可以直接开始学我们这里介绍的apache beam吧。如果已经开始学了，肯定也是有收获的，学习永远没有最好的“时机”，因为技术永远在发展更新啊。就像买不到最时髦的衣服一样。还是如第一篇虽说，看一个技术要看到它怎么解决问题，学习他的思路。



Li221

给元楠老师100个赞，每问必答，是所有专栏里最敬业的老师了！



2 2019-04-18

作者回复: 谢谢！大家的问题确实都很有意义。希望后面也能看到你的交流讨论



张德

给作者一百五十个赞👍



2 2019-04-17



买老实

都是数学的均衡排列的 好像都是数学方法的抽象



2 2019-04-17



有铭

今天这篇文章将MapReduce复杂度问题让我豁然开朗，以前我总觉得很多讲MapReduce的文章，讲的轻描淡写的，颇有点“把冰箱门打开，把大象装进去，把冰箱门关上”的味道。我当初就觉得可能MapReduce没那么简单。



2 2019-04-17

作者回复: 谢谢你的认同！也希望后面能继续看到你的留言。



孙稚昊

我们公司现在还在使用hadoop streaming 的MapReduce，默认mapper 结果是按key sort 过得，在 reducer 中借此实现join和group by的复杂操作，经常为了Join 一个table就要多写四个job



2 2019-04-17

作者回复: 是的，我觉得你总结的很好！



孙稚昊

现在写MapReduce job 开几百个worker经常有1，2个卡着不结束，基本都要在下班前赶着启动耗时长的任务。我们分片用户是用的 country+username 的 hash，还是比较均匀的



2 2019-04-17

作者回复: 看来你很有经验！确实是很经常出现的问题



mjl

个人理解，对于已知数据分布情况的数据，我们大多数情况下能找到合适的一个分区策略对数据进行分片。但实际上这对于数据开发者来说，就需要知道整体数据的一个基本情况。而对于数据倾斜，基本分为分区策略不当导致的倾斜以及单热点key的倾斜，对于后者，无论用户设置什么分区策略，都无法对数据进行分割。

对于数据倾斜问题的话，spark 3.0版本计划合入的AE功能给出了一定的方案。对于倾斜的partition，在shuffleWrite阶段就可以统计每个map输出的各个分区信息，然后根据这些信息来调整下一个stage的并发度。进一步的话，对于两表join，一张表有存在热点key的话，可以广播另外一张表的该partition，最终与其他分区的join结果做union。基于这个思路的话，engine其实是能很灵活的处理数据倾斜类问题，而不用用户去花精力研究选择。

👍 2 2019-04-17

作者回复: 这个思路看起来是做了很多课后研究了！希望后面也能继续参与讨论！



leben krieg

MR的劣势刚好对应了Spark的优势

1. 通过DAG RDD进行数据链式处理，最终只有一个job，大大降低了大数量MR的维护成本
2. 优先基于内存计算的Spark相对于基于磁盘计算的MR也大幅度提高了计算性能，缩短计算时间

个人觉得，这两点可以作为MR和Spark的主要区别。

👍 2 2019-04-17

作者回复: 谢谢你的留言！我认同你的观点，MR确实在每一步都需要经过磁盘的数据读取和结果的写入，速度上也就比不上Spark了。希望后面能继续看见你留言，一起学习进步！



楚翔style

mapreduce更适合处理离线数据吧，而且数据量大了比spark要稳定。楠兄怎么看？

👍 2 2019-04-17

作者回复: “离线数据”你指的是批处理？怎么定义“稳定”？



veio007

如果是我，外面弄个自增的计数器，然后每次计数器的当前值对worker个数取模，但是就算每个人分配同等数量的数据，同样会出现有人快有人慢的情况，机器的性能处理能力不一样或者其他干扰项都会有影响

👍 2 2019-04-17

作者回复: 这个可能和round robin那位同学思路差不多，也是合理的方法。重复一下在那边我的看法，这个需要对一个master节点来同步，而且出错了难以还原比如work 2坏了所有任务重试，找不到当时的任务了。也会增加系统复杂度。



呆小木

对用户id取哈希值，然后用哈希值对分区数取模。由于不同的id还是有可能计算得到相同的哈希值，也就是所谓的哈希碰撞，从而产生数据倾斜，可以在Map端给id拼上一个随机字符串，让它计算得到的哈希值分配更均匀，到Reduce端再去掉随机串。请老师指点☺

👍 2 2019-04-17

作者回复: 是正确的思路，似乎不需要先哈希，可以直接对用户id取模。随机字符串那步骤似乎



有点过度复杂化了



西西弗与卡夫卡

按用户在数据库中的唯一id

👍 2 2019-04-17

作者回复: 再加上取模的话的确是mapreduce默认的配置, 但是要注意根据shard数量动态调整模的数量。



孙稚昊

如果我读Beam的文档没有理解错的话, Beam只是spark 和flink 的各种API的一个封装, 本身并没有runner, 该有的调优还得在spark 和flink上面做, 所以除了google以外的公司用Beam的还是蛮少的

👍 1 2019-04-18

作者回复: 再次看到了你的留言, 谢谢!

就像我之前的回答一样, Beam的诞生更多是想抽象出一个统一的编程模型来处理批处理和流处理, 使不同的平台相互兼容, 让开发者有能力在不同的平台中转移。无论是Spark还是Flink, 它们都可以选择根据Dataflow Model来编写自己的底层实现。

关于调优的话Beam有根据不同平台来编写专门的API去编写配置。当然了, 因为需要统一编程接口, 你对底层的控制就没有原生Spark或者Flink那么好。

关于是否采用Beam的问题, 历史因素也占了很大比重。很多时候进行平台的转移可能对开发者来说是一个overkill。就像我之前所说, 现在越来越多的平台开始采用Dataflow Model来编写自己的底层实现。所以理论上, 在未来开发者或者公司就可以不必过于担心转移数据处理平台时的迁移成本了

希望对你有帮助, 如果有收获也请分享给朋友!



Geek\_daeed

请问老师, python作为主力语言的话, 学习升级的路线是怎样的呢, 有没有推荐的路线呢?

👍 1 2019-04-18

作者回复: 似乎并没有提到python为主力语言, 我们示例用python是因为简单方便大部分人看懂。



hallo128

年龄数据一般来说是服从0-100范围内的正态分布, 我们可以按正态分布进行模拟确定近似等数据量的分位点。以后再数据量逐渐被收集后, 再不断的优化分位点的确定。

👍 1 2019-04-17

作者回复: 仔细分析年龄分布倒也可以吧, 但是不太可拓展, 以后分布变了或者别的问题就不一定适应了。



牛冠群

您好, 学习周期有点长, 能不能加快些进度。感谢!

👍 1 2019-04-17

作者回复: 看到“30天速成”字样的资料可以直接扔掉



尚科

思考题, 可考虑使用用户注册日期的月、日来分片

👍 1 2019-04-17

作者回复: 很新颖, 我没想到, 确实对这个特定场景有些有效。我能想到的问题是如果机器数量大于12 (12个月), 不太可拓展。



买老实

好像在图片 抗噪点 和噪音抗噪都有类似的算法问题

👍 1 2019-04-17



hd900

根据用户的注册时长划分, 老用户新用户分开

👍 1 2019-04-17

作者回复: 很有意思的思路! 我不知道, 不知道注册时长是否是均匀分布



孙稚昊

我们组还在用Hadoop Streaming而没有使用spark的原因是spark 内存使用不加节制, 经常新起的job把周期性job的内存吃光, 导致他们有时会挂掉, 不知道这个问题是否有很好的解决方法。我们还是在保守地使用hadoop streaming, 麻烦得很

👍 1 2019-04-17

作者回复: 你的问题都很实际啊。job和job的互相抢占并不是spark独有的问题, 都需要一些优先级系统, 哪些job优先级高。还有异常处理, 错误重试, 任何数据处理系统都需要解决。这也是为什么这篇文章里提到多任务状态机, 很多时候为了异常处理免不了这些系统以至于增加了复杂度。



Mark Lee

把身份证后面的随机数截取到前面, 或者有规律的加3到4位的随机数

👍 1 2019-04-17

作者回复: 哈哈, 身份证太有意思了! 我不知道使用用户的身份id是否合法/道德/隐私。纯技术上似乎可行



M

请问文中所说的分片是指InputFormat读取数据时还是指shuffle时的分发?

👍 1 2019-04-17

作者回复: shuffle时, 但是前一个mr的输出也会成为下一个mr的input。



清水小亭

期待更新

👍 1 2019-04-17


作者回复: 谢谢，也期待你的留言，与你一起学习进步！

 Geek\_e8d453

小白来说思路，因为不知道要分多少份，开始想的是可以按生日来分，这样大数据的情况下只有2.29可能有点问题其他应该是均衡的，但这样只有365份。后来又想到如果有个唯一对应并且自增的用户id，就从末位开始。如果是16位的取3位就能切4000份了。。。。。

👍 1 2019-04-17

作者回复: 谢谢你的答案！问题本质上是想找到一个能很好均分的方法来分片。采取按照出生日期来划分的话还需要考虑到如果某一台机器宕机了，我们要如何重新分配原本属于这台机器上的用户。希望能继续看到你的留言，我们一起学习进步！

 Zach\_

可以使用 `hash(userId)%机器节点数` 来做嘛？

👍 1 2019-04-17

作者回复: 可以，这是最常见的sharding function

 白发少年

您好，我们的业务数据存储到mysql中的，可是在大数据这块，我们需要把mysql的数据导入到hbase,可是mysql的数据有可能实时更改，怎么做到mysql的数据更改后，能实时同步到hbase？

👍 1 2019-04-17

作者回复: 好像和专栏这节内容不想关。可以做个dual-write吧，或者一个pub/sub通知另一个。后面的章节会提到类似的概念。

 sudo

文中提到MapReduce被淘汰了，但有很多场景利用了MapReduce Streaming跑复杂的python逻辑，如跑个TensorFlow inference，没法直接改成Spark，请问下这种场景目前的最佳实践是什么？

👍 1 2019-04-17

作者回复: 专栏后面有介绍类似的应用，欢迎继续关注。改成beam或者spark完全可以。

 Austin

hash partition

👍 1 2019-04-17

作者回复: 谢谢你的答案！欢迎你继续留言，我们一起学习进步！

 无形

首选讨论可行的分区策论，比如年龄段、地区、账号ID等。然后选择某一比如年龄分组，然后统计各个年龄段的人数分布情况，求方差，如果数据量分布量差异比较大就缩小年龄分组范围重复以上操作，如果数据量差异还比较大，换一种分区策略，选择地区重复以上策略，如果任意单一策略都没法使得机器

处理的数据量接近，就同时使用两种策略，比如年龄和地区相结合，重复以上操作，直到试完了所有的策略的全组合，对每个组合进行比较，选择方差最小的作为分片的方法

👍 1 2019-04-17

作者回复: 谢谢你的答案！在分片的时候也需要考虑实施方案的时间复杂度噢。欢迎你继续留言讨论，一起学习进步！

👤 2 JohnT3e

如果处理逻辑和用户年龄相关，那么可以在原始年龄上加随机值的方式，尽可能散列。如果需要的话，后期再对结果进行合并。如果处理逻辑和年龄无关，那么可以选择散列性好的字段，比如用户唯一标识。在sharding时hash生成方面可以选取一些散列性好的算法，比如murmurhash算法。

👍 1 2019-04-17

作者回复: 谢谢你的答案！这个答案不错。相信你看到了问题的本质，希望能尽可能地让分片平均分布。欢迎你继续留言，我们一起学习进步！

👤 Keanu

我会考虑使用用户邮箱名首位的字母或数字进行分片。

👍 1 2019-04-17

作者回复: 谢谢你的答案！如果有些首位字母或数字的使用率很高，而其它字母或数字的使用率低的话，那些使用率高的字母或数字会造成Hot Spot的问题噢。欢迎你继续留言，我们一起学习进步！

👤 Freud

元楠老师，关于思考我认为引用随机标记的思想，第一次map输入时，在key上拼接随机数，经过第一个mrjob的处理后，再将标记去掉，这样可以大大减小数据倾斜。可以基本保证第一次数据分片的随机性。这个思想是我在Hive优化中学习到的，期待您的指导意见

👍 1 2019-04-17

作者回复: 听起来不错，但是得保证随机数是均匀的和是可重复的。可重复的意思是，比如你在shard 2上任务都失败了，需要能还原出错的任务重试。如果是“真”随机，就无法还原了。需要是比较好的伪随机。

👤 Rainbow

我们线上离线分析还是走hive，底层还是mapreduce，按老师说早淘汰了，我们该怎么改进

👍 1 2019-04-17

作者回复: 如果能应对现在的应用也可以，不需要为了赶时髦而换。

👤 笑若海

使用分布式hash

👍 1 2019-04-17

作者回复: 谢谢你的答案！DHT是一个不错的选择！欢迎你继续留言，我们一起学习进步！



Mac Kwan

我会考虑使用用户ID进行取模运算或者哈希计算来尝试把数据尽可能地均匀地分配给不同的worker

👍 1 2019-04-17

作者回复: 谢谢你的答案！这个答案不错。不过取模运算在机器有增减的时候会遇到麻烦，所有的用户必须重新取模运算一遍。Consistent Hashing可以很好地解决这个问题。欢迎你继续留言，我们一起学习进步！



旭东

区分热点数据（占用）大的数据，以及关联性大的数据。应该有个数据抽样分析，再做具体分片策略

👍 1 2019-04-17

作者回复: 这个点不错，可以先分析一下数据分布



刘楠

id hash后，取模？

👍 1 2019-04-17

作者回复: 谢谢你的答案！取模运算在机器有增减的时候会遇到麻烦，所有的用户必须重新取模运算一遍。Consistent Hashing可以很好地解决这个问题。欢迎你继续留言，我们一起学习进步！



李强Belo

如果一定要用用户的年龄来做分片，我觉得可以事先调查全球人类年龄结构发布，结合实际跑下来的性能分析，可以得出fb自己的用户年龄分布，然后在用到真实的业务计算场景中。当然，每个业务也还有自己的用户年龄分布，因为业务的特殊性。这确实比较麻烦，要预先做的工作很多。各业务团队需要自己去挖掘用户年龄分布。

👍 1 2019-04-17

作者回复: 谢谢你的答案！看得出来你是一个思维很缜密的人！欢迎你继续留言，我们一起学习进步！



jacksu

用户ID

👍 1 2019-04-17

作者回复: 谢谢你的答案！用户ID也是有可能造成Hot Spot问题的噢。欢迎你继续留言提问，我们一起学习进步！



鸿

哈希取模的方式

👍 1 2019-04-17

作者回复: 谢谢你的答案！这个答案不错。不过取模运算在机器有增减的时候会遇到麻烦，所有的用户必须重新取模运算一遍。Consistent Hashing可以很好地解决这个问题。欢迎你继续留言，我们一起学习进步！





Lion

按地区和数量两个条件分区。

👍 1 2019-04-17

作者回复: 好像没有完全理解你的意思? 你说是按照用户所在地? 可能地区和年龄一样分布不均吧



莫非

能否选择用户的出生月份作为分片函数呢?

👍 1 2019-04-17

作者回复: 可以考虑一下如果机器数量大于12的话?



henry

按每个用户在数据文件中的行数来分配

👍 1 2019-04-17

作者回复: 也是可以, 但是很多数据库并不会带原生“行数”这个概念, 特别是分布式的数据库。



Daryl

id拼接时间戳, 取模来分片

👍 1 2019-04-17

作者回复: 谢谢你的答案! id拼接时间戳的操作可以让Key更加的随机, 你看到了问题的本质。取模运算在机器有增减的时候会遇到麻烦, 所有的用户必须重新取模运算一遍。Consistent Hashing可以很好地解决这个问题。欢迎你继续留言, 我们一起学习进步!



godtrue

没使用过MR, 听文章的意思这个东西既费成本(难使用难维护)又达不到预期的性能, 那确实应该被淘汰。

大的东西, 比如数据, 一次处理不了只能分批处理, 分治的思想是深入骨髓的, 关键怎么分查起来方便快速是个难题。

思考题: 我目前能想到的思路, 一个就是一致性哈希算法来分, 另一个先将数据区别对待, 比如20以下和30以上的作为一波, 20~30作为另一波, 然后针对这两波数据采用不同的分片处理方式这样数据倾斜应该会少一些。

👍 2019-04-18

作者回复: 看数据的分布是在正确的思路



Alphacoo

想问下大牛几个问题:

- 1.mapreduce在某些问题中会不会还可以用?
- 2.怎么学习Beam? 直接看官方文档?
- 3.Hadoop的另外两个部分hdfs和yarn该学到什么程度?

👍 2019-04-18

作者回复: 1 可以用  
2 这个专栏可以帮你设置一个很好的起跑线。后面还是通过实践精进吧  
3 看你的应用场景？我觉得很少人会需要了解hdfs的底层实现，了解一下设计理念肯定有帮助



Vito

倒排索引，Hbase 设计rowkey也面临同样的问题。

👍 2019-04-18

作者回复: 确实有一些相似之处。很好的比较。

有收获欢迎分享给朋友。



毕洪宇

请问后面会分享liquid shard是怎么做的吗？比如如何解决data consistency, how to compute proper split size；我觉得对于解决一般的straggler 问题非常具有启发性，多谢

👍 2019-04-18

作者回复: 的确是很有价值的问题。会在后面探讨。

感谢提问，觉得有收获欢迎分享给朋友



刘鹤东[机智]

老师好，零基础小白提问，文中的美团案例，如果不用MR，其他方案能够减少处理流程吗，减少的是哪几步，效率上会提高多少呢？

👍 2019-04-18

作者回复: 问题很好。并不能减少业务的步骤。但是我这里的点是想说MR每部分开，多步骤管理比较困难。

如果觉得有收获，欢迎把文章分享给朋友



SpanningWings

谢谢老师。大家提到consistent hashing，这样会打的很均匀。我有个问题就是老师的问题是假设FB的处理哈，我不知道这样的处理是不是还包括未来的查询。如果是查询的话那我们是不是要建立些索引什么的，那这样一致性哈希是不是就不合适了。还是按照一个重要的key来分片，然后动态调整分片的范围比较好？这样查询可以避免full table scan.

👍 2019-04-18

作者回复: 谢谢你的答案！这一讲的问题我只是单纯想让大家思考处理大规模数据如何分片的问题，不涉及到后面的查询操作，不够你能考虑到其它use cases也是非常不错的。希望能继续看到你的留言，一起学习进步！



J Zhang

我想对于分片，hbase 的rowkey设计最佳实践给予了很大的参考价值，比如可以通过加盐这种方式可以使得数据很均衡

👍 2019-04-18

作者回复: 确实有相似之处, 但是我在另几个问题里提到, 需要保证加盐是deterministic, 不然错误重试无法还原分片任务。

感谢提问, 有收获欢迎把专栏分享给朋友。



桂桂

请问, 常用标注工具有吗, 能推荐吗



2019-04-18

作者回复: 恕我直言这方面还是市场空白。不同应用场景需要的标注功能挺不一样, 比如我做的医疗AI领域对于标注者专业知识要求高, 也需要异议整合等功能, 还需要多模态同时标注。我觉得你在评估工具好不好时, 可以从两个方面看, 一个是是否方便拓展, 比如今天标注这些问题, 明天那些问题模态不一样能否很快设置。另一个就是功能是否满足你们场景需要了。

感谢提问, 有收获欢迎分享。



大王叫我来巡山

收益匪浅, 写书的一定要为自己负责, 看过太多垃圾书, 并且是书带你进坑, 其实作者的功力都不够, 在描述某些问题的处理的时候能看出来作者自己都没有处理过, 卖得好的书不一定是好书



2019-04-18

作者回复: 感谢肯定, 有收获欢迎分享