181870207 作业3 😄

1. 简述MapReduce的主要功能和设计思想。

主要功能

1. 任务调度：**分配调度节点、监控节点、同步控制、计算性能优化**

> 提交的一个计算作业将被划分为很多个计算任务，任务调度功能主要负责为这些划分后的计算任务分配和调度计算节点；同时负责监控这些节点的运行情况，并负责map节点执行的同步控制（barrier）；同时也负责一些计算性能优化处理，如对最慢的计算任务采用多备份执行、选最快完成者作为结果

2. 数据、代码互定位：**代码向数据迁移**

> 为了减少数据通信，一个基本原则是本地化数据处理，即一个计算节点尽可能处理其本地磁盘上所分布存储的数据，这实现了代码向数据的迁移；当无法进行这种本地化数据迁移时，在寻找其他可用节点并将数据从网络上传送到该节点，但尽可能从数据本地机架上寻找可用节点以减少通信延迟

3. 出错处理：**检测错误、重新分配**

> 以低端商用服务器构成的大规模MapReduce计算集群中，节点硬件出错和软件bug是常态，因此MapReduce需要能检测并隔离出错节点，并调度分配新的节点接管出错节点的计算任务

4. 分布式数据存储与文件管理

> 海量数据处理需要一个良好的分布数据存储和文件管理系统支撑,该文件系统能够把海量数据分布存储在各个节点的本地磁盘上，但保持整个数据在逻辑上成为一个完整的数据文件；为了提供数据存储容错机制,该文件系统还要提供数据块的多备份存储管理能力

5. Combiner和Partitioner：**数据处理**

> 为了减少数据通信开销,中间结果数据进入reduce节点前需要进行合并(combine)处理,把具有同样主键的数据合并到一起避免重复传送；一个reduce节点所处理的数据可能会来自多个map节点，因此, map节点输出的中间结果需使用一定的策略进行适当的划分(partitioner)处理，保证相关数据发送到同一个reduce节点

设计思想

1. 向"外"横向扩展，而非向"上"纵向扩展口失效被认为是常态

> MapReduce集群的构筑选用价格便宜、易于扩展的大量低端商用服务器,而非价格昂贵、不易扩展的高端服务器(SMP)
> 低端服务器市场与高容量Desktop PC有重叠的市场,因此,由于相互间价格的竞争、可互换的部件和规模经济效应，使得低端服务器保持较低的价格
> 基于TPC-C在2007年底的性能评估结果，一个低端服务器平台与高端的共享存储器结构的服务器平台相比，其性价比大约要高4倍;如果把外存价格除外,低端服务器性价比大约提高12倍
> 对于大规模数据处理，由于有大量数据存储需要,显而易见,基于低端服务器的集群远比基于高端服务器的集群优越，这就是为什么MapReduce并行计算集群会基于低端服务器实现

2. 把处理向数据迁移

> 传统高性能计算系统通常有很多处理器节点与一些外存储器节点相连，如用区域存储网络(SAN, Storage Area Network)连接的磁盘阵列，因此，大规模数据处理时外存文件数据1/0访问会成为一个制约系统性能的瓶颈。
> 为了减少大规模数据并行计算系统中的数据通信开销，应当考虑将处

> 理向数据靠拢和迁移,取代把数据传送到处理节点(数据向处理器或代码迁移)的传统方式。
>
> MapReduce采用了**数据/代码互定位**的技术方法,计算节点将首先尽量负责计算其本地存储的数据,以发挥数据本地化特点locality, 仅当节点无法处理本地数据时,再采用就近原则寻找其它可用计算节点，并把数据传送到该可用计算节点。

3. 失效被认为时常态

> MapReduce集群中使用大量的低端服务器，因此，节点硬件失效和软件出错是常态,因而:
> 一个良好设计、具有容错性的并行计算系统不能因为节点失效而影响计算服务的质量,任何节点失效都不应当导致结果的不一致或不确定性;任何一个节点失效时,其它节点要能够无缝接管失效节点的计算任务;当失效节点恢复后应能自动无缝加入集群,而不需要管理员人工进行系统配置
> MapReduce并行计算软件框架使用了多种有效的机制,如节点自动重启技术,使集群和计算框架具有对付节点失效的健壮性,能有效处理失效节点的检测和恢复。

4. 顺序处理数据、避免随机访问数据

> 大规模数据处理的特点决定了大量的数据记录不可能存放在内存、而只可能放在外存中进行处理。
> 磁盘的顺序访问和随机访问在性能上有巨大的差异，顺序访问速度更快
> MapReduce设计为面向大数据集批处理的并行计算系统，所有计算都被**组织成很长的流式操作**，以便能利用分布在集群中大量节点上磁盘集合的高传输带宽。

5. 为应用开发者隐藏系统层细节

> 软件工程实践指南中,专业程序员认为之所以写程序困难,是因为程序员需要记住大多的编程细节(从变量名到复杂算法的边界情况处理),这对大脑记忆是一个巨大的认知负担,需要高度集中注意力
> 而并行程序编写有更多困难,如需要考虑多线程中诸如同步等复杂柴琰的细节,由于并发执行中的不可预测性,程序的调试查错也十分闲难;大规模数据处理时程序员需要考虑诸如数据分布存储管理、数据分发、数

> 据通信和同步、计算结果收集等诸多细节问题
> MapReduce提供了一种抽象机制将程序员与系统层细节隔离开来,程序员仅需描述需要计算什么 (**what to compute**),而具体怎么去做(**how to compute**)就交由系统的执行框架处理,这样程序员可从系统层细节中解放出来,而致力于其应用本身计算问题的算法设计

6. 平滑无缝的可扩展性

> 主要包括两层意义上的扩展性:**数据扩展和系统规模扩展**
> 理想的软件算法应当能随着数据规模的扩大而表现出持续的有效性,性能上的下降程度应与数据规模扩大的倍数相当
> 在集群规模上,要求算法的计算性能应能随着节点数的增加保持接近线性程度的增长
> 绝大多数现有的单机算法都达不到以上理想的要求;把中间结果数据维护在内存中的单机算法在大规模数据处理时很快失效;从单机到基于大规模集群的并行计算从根本上需要完全不同的算法设计
> 奇妙的是,MapReduce几乎能实现以上理想的扩展性特征。多项研究发现基于 MapReduce的计算性能可随节点数目增长保持近似于线性的增长

---

2. 简述GFS的基本设计原则和数据访问过程。

基本设计原则

1. Google GFS是一个基于分布式集群的大型分布式文件系统，为MapReduce计算框架提供数据存储和数据可靠性支撑;
2. GFS是一个构建在分布节点本地文伴系统之上的一个逻辑上文件系统，它将数据存储在物理上分布的每个节点上，但通过GFS将整个数据形成一个逻辑上整体的文件。
3. 廉价本地磁盘分布存储
   各节点本地分布式存储数据，优点是不需要采用价格较贵的集中式磁盘阵列，容量可随节点数增加自动增加

4. 多数据自动备份解决可靠性
   采用廉价的普通磁盘,把磁盘数据出错视为常态，用自动多数据备份存储解决数据存储可靠性问题
5. 为上层的 MapReduce计算框架提供支撑
   GFS作为向上层MapReduce执行框架的底层数据存储支撑，负责处理所有的数据自动存储和容错处理，因而上层框架不需要考感底层的数据存储和数据容错问题

数据访问工作过程

1. 程序运行前，数据已经存储在GFS文件系统中；程序运行时应用程序会告诉GFS Server所要访问的文件名或者数据块索引是什么·
2. GFS Server根据文件名和数据块索引在其文件目录空间中查找和得该文件或数据块，找出数据块具体在哪些chunk server上；将这些位置信息回送给应用程序
3. 应用程序根据GFS Server返回的具体chunk数据块的位置信息，直接访问相应的chunk server
4. 应用程序根据返回的具体的chunk数据块位置信息直接读取指定位置的数据进行计算处理

GFS访问具体数据时**不需要经过GFS Master**，因此避免了master成为访问瓶颈

因为一个大数据会存储在不同的chunk server中，**应用程序可以实现并发访问**

---

3. 简述BigTable的数据模型设计。

1. A Table in BigTable is a：

- Sparse：稀疏的
- Distributed：分布式的
- Persistent：持续的
- Multidimensional：多维的
- Sorted map：排序图

2. BigTable不是一个完全的关系型数据库，它支持数据设计的动态控制：可以在一个列族（column family）里增加或删除修饰符（qualifier）。它允许客户机获取数据的位置属性。数据通过列名和行名索引

3. 所有数据被当作字符串处理，数据的位置可以通过'careful choice of the schema'控制

4. 行：

   所有的行按照row key字典序排序

   Tablets ： rows with consecutive keys

5. 列

   有列族的概念，列采用 Family ： qualifier的形式区分

   column family is the unit of access control

   可以在某个column family里面增加column

6. cell

   指的是 the storage referenced by **a particular row key，column key，and timestamp**

   different cell in a table can contain multiple versions indexed by timestamp