谷歌三驾马车

金于淮19301061

MapReduce

MapReduce是一个用于大规模数据（大于1TB）处理的分布式计算模型、编程模型，它最初是由Google设计并实现的，在Google提出时，给它的定义是：Map/Reduce是一个编程模型，是一个用于处理和生成大规模数据集的相关的实现。

MapReduce的主要思想“Map（映射）”和“Reduce（规约）”都来自于函数式编程语言。MapReduce极大地方便了编程人员在不会分布式并行编程的情况下，将自己的程序运行在分布式系统之上。用户只需要定义一个map函数来处理一个key/value对以生成一批中间的key/value对，再定义一个reduce函数将所有这些中间的有着相同key的values合并起来。很多现实世界中的任务都可用这个模型来表达，具有较强的实用价值。

MapReduce是一个基于集群的高性能并行计算平台。通过MapReduce可以将市场上普通的商用服务器构成一个包含数十、数百甚至数千个节点的分布和并行计算集群。

MapReduce是一个并行计算与运行软件框架。它提供了一个庞大但设计精良的并行计算软件框架，能自动完成计算任务的并行化处理，自动划分计算数据和计算任务，在集群节点上自动分配和执行任务以及收集计算结果，将数据分布存储、数据通信、容错处理等并行计算涉及到的很多系统底层的复杂细节交由系统负责处理，大大减少了软件开发人员的负担。MapReduce是一个并行程序设计模型与方法。它借助于函数式程序设计语言Lisp的设计思想，提供了一种简便的并行程序设计方法，用Map和Reduce两个函数编程实现基本的并行计算任务，提供了抽象的操作和并行编程接口，以简单方便地完成大规模数据的编程和计算处理。

Google MapReduce 所执行的分布式计算会以一组键值对作为输入，输出另一组键值对，用户则通过编写 Map 函数和 Reduce 函数来指定所要进行的计算。由用户编写的Map函数将被应用在每一个输入键值对上，并输出若干键值对作为中间结果。之后，MapReduce 框架则会将与同一个键 II 相关联的值都传递到同一次 Reduce 函数调用中。同样由用户编写的 Reduce 函数以键 II 以及与该键相关联的值的集合作为参数，对传入的值进行合并并输出合并后的值的集合。MapReduce的数据处理模型非常简单：map函数和reduce函数的输入和输出都遵循<key,value>键值对的格式，简单的用符号表示就是：

Map：（K1，V1）——> list(K2，V2)

Reduce：（K2，list< V2>）——> list<K3，V3>

Map-Reduce框架的运作完全基于<key,value>对，即数据的输入是一批<key,value>对，生成的结果也是一批<key,value>对，只是有时候它们的类型不一样而已。Key和value的类由于需要支持被序列化（serialize）操作，所以它们必须要实现Writable接口，而且key的类还必须实现WritableComparable接口，使得可以让框架对数据集的执行排序操作。

GFS

GFS是一个分布式文件系统，用来存储大量的较大文件，它可以在廉价的硬件上实现存储文件，并做到容错性，并且针对多个客户同时访问提供比较有竞争力的性能。

GFS可以：

- 把一个较大的文件切分成不同的单元块。
- 把每一个单元块存储在ChunkServer上，并且每一块都会复制在多个ChunkServer服务器上。
- 每一个文件包含多少块和哪些块这些元数据存储在GFS Master服务器上。
- 这是一个低成本的分布式存储系统，用来数据量非常大的存储场景，通常为mapreduce的大数据处理模型提供输入和输出的存储系统。

GFS与以往的文件系统的不同的观点如下：

1. 部件错误不再被当作异常，而是将其作为常见的情况加以处理。因为文件系统由成百上千个用于存储的机器构成，而这些机器是由廉价的普通部件组成并被大量的客户机访问。部件的数量和质量使得一些机器随时都有可能无法工作并且有一部分还可能无法恢复。所以实时地监控、错误检测、容错、自动恢复对系统来说必不可 少。

2.按照传统的标准，文件都非常大。长度达几个GB的文件是很平常的。每个文件通常包含很多应用对象。当经常要处理快速增长的、包含数以 万计的对象、长度达TB的数据集时，我们很难管理成千上万的KB规模的文件块，即使底层文件系统提供支持。因此，设计中操作的参数、块的大小必须要重新考 虑。对大型的文件的管理一定要能做到高效，对小型的文件也必须支持，但不必优化。

3.大部分文件的更新是通过添加新数据完成的，而不是改变已 存在的数据。在一个文件中随机的操作在实践中几乎不存在。一旦写完，文件就只可读，很多数据都有这些特性。一些数据可能组成一个大仓库以供数据分析程序扫 描。有些是运行中的程序连续产生的数据流。有些是档案性质的数据，有些是在某个机器上产生、在另外一个机器上处理的中间数据。由于这些对大型文件的访问方 式，添加操作成为性能优化和原子性保证的焦点。而在客户机中缓存数据块则失去了吸引力。

4.工作量主要由两种读操作构成：对大量数据的流方式 的读操作和对少量数据的随机方式的读操作。在前一种读操作中，可能要读几百KB，通常达 1MB和更多。来自同一个客户的连续操作通常会读文件的一个连续的区域。随机的读操作通常在一个随机的偏移处读几个KB。性能敏感的应用程序通常将对少量 数据的读操作进行分类并进行批处理以使得读操作稳定地向前推进，而不要让它来反反复复地读。

5.工作量还包含许多对大量数据进行的、连续的、向文件添加数据的写操作。所写的数据的规模和读相似。一旦写完，文件很少改动。在随机位置对少量数据的写操作也支持，但不必非常高效。

6.系统必须高效地实现定义完好的大量客户同时向同一个文件的添加操作的语义。

7.高可持续带宽比低延迟更重要

BigTable

介绍：Bigtable是一个可以管理结构化数据的分布式存储系统，它本身支持水平的横向扩展，通过使用成千上万的连接服务器，来支持PB量级的数据处理。同时Bigtable 也是一种压缩的、高性能的、高可扩展性的，基于 Google 文件系统（Google File System，GFS）的数据存储系统，用于存储大规模结构化数据，适用于云端计算。

Bigtable 的设计是为了能可靠地处理 PB 级的海量数据，使其能够部署在千台机器上。Bigtable 借鉴了 **parallel databases**（并行数据库）和 **main-memory databases(**内存数据库**)**的一些特性，但是提供了一个完全不同接口：

- 不支持完整的关系数据模型，只为用户提供简单的数据模型；
- 模型支持动态控制数据的分布和格式，在 Bigtable 中，数据并没有固定的格式，用户可以自定义数据的 schema；
- 允许客户机自己推断（reason）底层数据存储的 locality propertie；
- 支持使用列名或者行名作为索引，名字可以是任意字符串；
- 将数据视为未解释的字符串，尽管客户端未必真的就是用字符串格式进行存储，比如客户端经常将各种 structured and semi-structured（结构化和半结构化，它们不是纯文本）的序列化到 BitTable 的字符串中；
- 允许客户端动态控制 BitTable 的数据来源：内存 or 磁盘；

Bigtable 会把数据存储在若干个 Table（表）中，Table 中的每个 Cell（数据单元）的形式如下：

$$(row : \text{string}, column : \text{string}, time : \text{int64}) \rightarrow \text{string}$$

Cell 内的数据由字节串（`string`）构成，使用行、列和时间戳三个维度进行定位。

Bigtable 在存储数据时会按照 Cell 的 Row Key 对 Table 进行字典排序，并提供行级事务的支持（类似于 MongoDB，不支持跨行事务）。作为分布式的存储引擎，Bigtable 会把一个 Table 按 Row 切分成若干个相邻的 Tablet，并将 Tablet 分配到不同的 Tablet Server 上存储。如此一来，客户端查询较为接近的 Row Key 时 Cell 落在同一个 Tablet 上的概念也会更大，查询的效率也会更高。

除外，Bigtable 会按照由若干个 Column 组成的 Column Family（列族）对 Table 的访问权限控制。Column Key 由 `family:qualifier` 的形式组成，用户在使用前必须首先声明 Table 中有哪些 Column Family，声明后即可在该 Column Family 中创建任意 Column。由于同一个 Column Family 中存储的数据通常属于同一类型，Bigtable 还会对属于同一 Column Family 的数据进行合并压缩。由于 Bigtable 允许用户以 Column Family 为单位为其他用户设定数据访问权限，数据统计作业有时也会从一个 Column Family 中读出数据后，将统计结果写入到另一个 Column Family 中。

Table 中的不同 Cell 可以保存同一份数据的多个版本，以时间戳进行区分。时间戳本质上为 64 位整数，可由 Bigtable 自动设定为数据写入的当前时间（微秒），也可由应用自行设定，但应用需要自行确保 Cell 间不会出现冲突。对于拥有相同 Row Key 和 Column Key 的 Cell，Bigtable 会按照时间戳降序进行排序，如此一来最新的数据便会被首先读取。在此基础上，用户还可以设定让 Bigtable 只保存最近若干个版本的数据或是时间戳在指定时间范围内的数据。

Bigtable 实现了如下四个目标：

- Wide Applicability - 广泛的适用性；
- Scalability - 可伸缩性（特指横向）；
- High Performance - 高性能；
- High Availability - 高可用性；