1. 简述MapReduce的主要功能和设计思想。

主要功能：

提供一个统一的计算框架，将大数据计算任务划分为多个较小的、重复处理的子任务，对每个数据记录/元素作感兴趣的处理，获取中间结果信息，对其进行收集、整理、合并，生成最终结果并输出。

1. 任务调度：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任务分配和调度计算节点（map和reduce节点），同时负责监控这些节点的执行状态，并负责map节点执行的同步控制（barrier）和进行一些计算性能优化处理。
2. 数据/代码互定位：为了减少数据通信，一个基本原则是本地化数据处理，即节点尽可能处理本地磁盘上存储的数据，实现代码向数据的迁移，无法进行该操作时再使用数据向代码迁移以减少通信延迟。
3. 出错处理：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有bug是常态，因此Map Reduce需要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能力。
5. Combiner和Partitioner：为了减少数据通信开销，中间结果进入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要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一个reduce节点。

设计思想：

1.向“外”横向扩展，而非向“上”纵向扩展。Map Reduce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。低端服务器规模经济价格低、性价比高、且能够满足大规模数据处理的数据存储需要。

2.失效被认为是常态。低端服务器中硬件失效和软件出错都是常态，因此一个良好设计、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性或不确定性。任何一个节点失效，其他节点要能够无缝接管失效节点的计算任务，失效节点恢复后要能够自动无缝加入集群而不需要管理员人工进行系统配置。

3．把处理向数据迁移。传统高性能计算系统常有很多处理器与一些外存储器节点相联，因此大规模数据处理时外存文件数据I/O访问会成为制约系统性能的瓶颈。为减少并行计算系统中的通信开销，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的传统方式。

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，只可能在外存中进行处理。Map Reduce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。并行程序编写中有大量困难，如需要考虑多线程中诸如同步等复杂繁琐的细节，由于并发执行中的不可预测性，程序的调试查错也十分困难，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、计算结果收集等细节问题。Map Reduce提供了一种抽象机制隔绝程序员和系统层细节，使之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可扩展性：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。

2. 简述GFS的基本设计原则和数据访问过程。

基本设计原则：Google GFS是一个基于分布式集群的大型分布式文件系统，为Map Reduce计算框架提供数据存储和数据可靠性支撑；GFS是一个构建在分布节点本地文件系统之上的一个逻辑文件系统，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。

1. 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。
2. 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。
3. 为上层的Map Reduce计算框架作为支撑：GFS作为向上层Map Reduce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虑底层的数据存储和数据容错问题。

数据访问过程：

1. 在程序运行前，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定位该文件或数据块，找出数据块具体在哪些ChunkServer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数据块位置信息，直接访问相应的Chunk Server
4. 应用程序根据GFS Server返回的具体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。

特点：应用程序访问具体数据是不需要经过GFS Master，因此，避免了Master成为访问瓶颈。由于一个大数据会存储在不同的Chunk Server中，应用程序可实现并发访问。

1. 简述BigTable的数据模型设计。

Bigtable是一个稀疏的、分布式的、持久化存储的多维度排序Map。Map的索引是行关键字、列关键字以及时间戳；Map中的每个value都是一个未经解析的字符串。Bigtable不是[关系型数据库](https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%BA%93)，但是却沿用了很多关系型数据库的术语，像table（表）、row（行）、column（列）等。Bigtable用于存储关系较为复杂的半结构化数据。数据大致可以分为以下三类：

1.非结构化数据：包括所有格式的办公文档、文本、图片、图像、音频和视频信息等。

2.结构化数据：一般存储在关系数据库中，可以用二维关系表结构来表示。结构化数据的模式(Schema，包括属性、数据类型以及数据之间的联系)和内容是分开的，数据的模式需要预先定义。

3.半结构化数据：介于非结构化数据和结构化数据之间，HTML文档就属于半结构化数据。它一般是自描述的，与结构化数据最大的区别在于，半结构化数据的模式结构和内容混在一起，没有明显的区分，也不需要预先定义数据的模式结构。

本质上说，Bigtable是一个键值（[key-value](https://baike.baidu.com/item/key-value)）映射，Bigtable是一个稀疏的，[分布式](https://baike.baidu.com/item/%E5%88%86%E5%B8%83%E5%BC%8F)的，持久化的，多维的排序映射。Bigtable的键有三维，分别是行键（row key）、列键（column key）和[时间戳](https://baike.baidu.com/item/%E6%97%B6%E9%97%B4%E6%88%B3)（timestamp），行键和列键都是字节串，时间戳是64位整型；而值是一个字节串。

与二维关系表结构不同的是，Bigtable的键是三维的。二维表就像日常生活中遇到的普通表格或数据库中的表格，根据行列就能找到信息，而且行列唯一确定一条信息。三维结构更像图书馆里的一个书架，行列可以确定书架上的一个空间(图书馆的书架上相邻空间放的都是一类书，由上文可知，在Bigtable里，"类似的"数据也是放的很近的)，而不能唯一的确定某一本书。时间戳给用户介绍了这些书的"顺序"，用户可以根据顺序拿到自己想要的那本书。