我首先读的论文是《**MapReduce: Simplifified Data Processing on Large Clusters**》，一些大型集群计算的输入数据通常很大，计算必须分布在数百台或数千台机器上，以便在合理的时间内完成。如何并行化计算、分发数据和处理故障的问题会用大量复杂代码掩盖原始的简单计算来处理这些问题，本文要解决的问题就是简化大型集群上的数据处理。该论文的主要贡献是一个简单而强大的接口MapReduce，使自动并行化和分发大规模计算，结合该接口的实现，在大型商品个人电脑集群上实现高性能。用户只需指定一个映射函数，它处理一个键/值对来生成一组中间键/值对，以及一个合并与同一中间键关联的所有中间值的减少函数。

其后本文展示了很多有趣的程序简单实例，来展示MapReduce是如何进行计算的。

1. 分布式 Grep：如果映射函数与提供的模式匹配，则会发出一行。减少函数是一个标识函数，它只是将提供 的中间数据复制到输出中。
2. URL 访问频率的计数：地图功能处理网页请求和输出的日(URL，1>。减少函数将同一 URL 的所有值加在一起，并发出一个(URL，总计数)对。
3. 反向 Web 链接图：映射函数为在名为源的页面中找到 的目标 URL 的每个链接输出（目标、源）对。rowe 函数连接与给定目标 URL 关联的所有源 URL 的列表， 并发出对：（目标、列表（源））
4. 每个主机的术语向量：术语向量将文档或一组文档中 出现的最重要的单词汇总为（单词、频率）对的列表。 映射函数为每个输入文档发出对（主机名从文档的 URL 中提取）。减少函数传递给定主机的所有每个文 档的项向量。它将这些项向量加在一起，丢弃不常见 的项，然后发出一个最终的（主机名，项向量）对。
5. 倒置索引：映射函数解析每个文档，并发出（word、 文档 ID>对。减少函数接受给定单词的所有对，对相应的文档 ID 进行排序，并发出一个（word、列表（文 档 ID）>对。所有输出对的集合形成了一个简单的反 向索引。很容易增加这种计算来跟踪单词的位置。
6. 分布式排序：映射函数从每个记录中提取键，并发出 一个（键、记录）对。减少函数使所有对保持不变。

用这种函数式风格编写的程序MapReduce可以自动并行化，并在大型的商品机器集群上执行。运行时系统负责对输入数据进行分区、跨一组机器调度程序执行、处理机器故障和管理所需的机器间通信等细节。

MapReduce有以下优点：

- 适合PB级以上海量数据的离线处理
- 隐藏了并行化、容错、数据分发以及负载均衡等细节
- 允许没有分布式或并行系统经验的程序员轻松开发分布式任务程序
- 伸缩性好，使用更多的服务器可以获得更多的吞吐量

但同时也有一些限制：

- 不擅长实时计算
- 无法进行流式计算，因为 *MapReduce* 的输入数据是静态的
- 无多阶段管道，对于先后依赖的任务，*MapReduce* 必须把数据写入硬盘，再由下一个 *MapReduce* 任务调用这些数据，造成了多余的磁盘 I/O

同时我对问题进行总结：

- MapReduce 如何节约网络带宽?

1. 集群中所有服务器既执行 *GFS* ，也执行 *MapReduce* 的 *worker*
2. *master* 调度时会优先使 *map* 任务执行在存储有相关输入数据的服务器上
3. *reduce worker* 直接通过 *RPC* 从 *map worker* 获取中间数据，而不是通过 *GFS* ，因此中间数据只需要进行一次网络传输

4. *R* 远小于中间 *key* 的数量，因此中间键值对会被划分到一个拥有很多 *key* 的文件中，传输更大的文件（相对于一个文件拥有更少的 *key*）效率更高

- MapReduce 如何获得好的负载均衡?

1. 通过备用任务缓解 *straggler* 问题
2. 使 *task* 数远多于 *worker* 数，*master* 将空闲任务分给已经完成任务的 *worker*

同时作者也在文章中提到，他们从这项工作中学到了一些东西。首先，限制编程模型使计算易于并行化和分发，并使这些计算容 错。第二，网络带宽是一种稀缺的资源。因此，我们 系统中的许多优化的目标是减少跨网络发送的数据 量：局部优化允许我们从本地磁盘读取数据，并且将中间数据的单一副本写入本地磁盘可以节省网络带宽。第三，冗余执行可以用于减少慢速机器的影响， 并处理机器故障和数据丢失。我想这也是我们应该学习的。

关于本文，我最大的感受就是，看到了作者从提出问题到分析问题，再到提出解决方案，测试性能，最后评估的整个研究过程，研究不易，需要艰苦钻研的精神，这也是最需要我学习的。