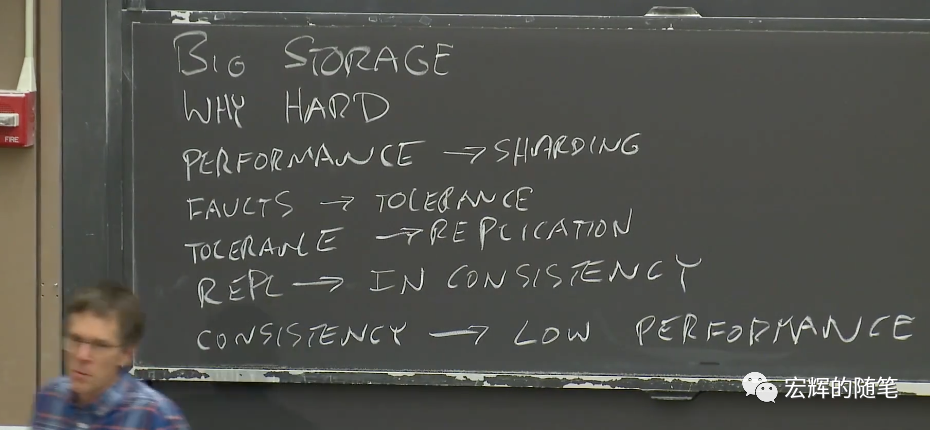
一、整理MapReduce工作流程，分析哪些过程可能需要与GFS存储部件进行数据传输。在这些过程中，哪个环节的数据传输可以优化（削减），通过何种方式？哪些环节数据传输无法优化。

二、结合下图，分析存储系统设计的难点，以及其中的逻辑关系



一、

MapReduce是一种用于分布式数据处理的编程模型。它包括两个主要部分：Map阶段和Reduce阶段。

1. Map阶段：首先，将输入数据分成多个数据块，并将每个数据块分配给一个Map任务。每个Map任务都会对分配给它的数据块进行处理，并将结果以键值对的形式写入缓存。

2. Reduce阶段：在Reduce阶段，Map任务的缓存会被合并，并分配给一个Reduce任务。每个Reduce任务都会处理它所分配的数据，并将结果写入输出文件。

整个MapReduce工作流程中，可能需要与GFS存储部件进行数据传输的环节包括：

1. 将输入数据从GFS存储部件读取到Map任务中。

2. 将Map任务的缓存写入GFS存储部件。

3. 将Map任务的缓存从GFS存储部件读取到Reduce任务中。

4. 将Reduce任务的输出结果写入GFS存储部件。

在这些过程中，可以通过以下方式优化（削减）数据传输：

1. 使用缓存优化：可以使用内存缓存或文件系统缓存来减少对GFS存储部件的访问次数。

2. 使用压缩技术：可以使用压缩技术来减少数据在传输过程中的大小，从而减少网络带宽的使用。

3. 使用数据块的并行处理：可以通过同时处理多个数据块来提高数据处理的效率，从而减少对GFS存储部件的访问次数。

但是，由于MapReduce工作流程的设计和实现，有些环节的数据传输是无法优化的。例如，在Map阶段，Map任务必须读取所有的输入数据，而在Reduce阶段，Reduce任务必须读取所有的Map任务的缓存，这些数据传输是无法优化的。

二、

人们设计大型分布式系统或大型存储系统通常是为了获取巨大的性能加成，进而利用数百台计算机的资源来同时完成大量工作。因此，性能问题就成为了最初的诉求。 之后，很自然的想法就是将数据分割，放到大量的服务器上，这样就可以并行的从多台服务器读取数据。我们将这种方式称之为分片（Sharding）。

如果你有数千台服务器，那么总是会有一台服务器宕机，每天甚至每个小时都可能会发生错误。所以，我们需要自动化的方法而不是人工介入来修复错误。我们需要一个自动的容错系统，这就引出了容错这个话题（fault tolerance）。

实现容错最有用的一种方法是使用复制，只需要维护2-3个数据的副本，当其中一个故障了，你就可以使用另一个。所以，如果想要容错能力，就得有复制（replication）。

如果有复制，那就有了两份数据的副本。可以确定的是，如果你不小心，它们就会不一致。所以，你本来设想的是，有了两个数据副本，你可以任意使用其中一个副本来容错。但是如果你不够小心，两个数据的副本就不是完全一致，严格来说，它们就不再互为副本了。而你获取到的数据内容也将取决于你向哪个副本请求数据。这对于应用程序来说就有些麻烦了。所以，如果我们有了复制，我们就有不一致的问题（inconsistency）。

通过聪明的设计，你可以避免不一致的问题，并且让数据看起来也表现的符合预期。但是为了达到这样的效果，你总是需要额外的工作，需要不同服务器之间通过网络额外的交互，而这样的交互会降低性能。所以如果你想要一致性，你的代价就是低性能。但这明显不是我们最开始所希望的。

现实中，如果你想要好的一致性，你就要付出相应的代价。如果你不想付出代价，那就要忍受一些不确定的行为。我们之后会在很多系统中看到这里介绍的循环。通常，人们很少会乐意为好的一致性付出相应的性能代价。