论文学习体会

每个chunkreplica都作为一个简单的Linux文件存储在chunkerver上，并且只根据需要进行扩展。 懒惰的空间分配避免了由于内部分裂而浪费空间，也许是对如此大的块大小的最大反对。 大块大小提供了几个重要的优势。 首先，它减少了客户端与主机交互的需要，因为读取和写入相同的块只需要向主机请求一个初始的块位置信息。 这种减少对于我们的工作负载特别重要，因为应用程序主要是按顺序读写大型文件。 即使是小的随机读取，客户端也可以舒适地缓存多TB工作集的所有块位置信息。 其次，由于在大块上，客户端更有可能对给定的块执行许多操作，因此它可以通过保持persis来减少网络开销帐篷TCP连接到块服务器在很长一段时间内。 第三，它减少了存储在主机上的元数据的大小。 另一方面，一个大的块大小，即使是懒惰的空间分配，也有其缺点。 一个小文件由少量的块组成，也许只有一个。 如果许多客户端访问同一文件，存储这些块的块服务器可能成为热点。

元数据主存储元数据的三种主要类型：文件和块名空间、从文件到块的映射以及每个块副本的位置。 所有元数据都保存在主存储器中。 前两种类型（命名空间和文件到链接映射）也通过将突变记录到存储在主机本地磁盘上并在远程机器上复制的操作日志来保持持久性。 使用日志允许我们简单、可靠地更新主状态，并且在发生主崩溃时不会产生不一致的风险。

操作日志操作日志包含关键元数据更改的历史记录。 它是GFS的核心。 它不仅是元数据的唯一持久记录，而且是定义并发操作顺序的逻辑时间线。 文件和块以及它们的版本都是唯一的，并且永远由创建它们的逻辑时间标识。 由于操作日志是至关重要的，我们必须可靠地存储它，并且在元数据更改持久化之前不能使更改对客户端可见。 否则，即使块本身存活下来，我们也会有效地失去整个文件系统或最近的客户端操作。 因此，我们在多个远程机器上复制它，并且只有在将相应的日志记录冲洗到本地和远程磁盘之后才能响应客户端操作。主在冲洗之前将几个日志记录一起批处理，从而减少了冲洗和复制对整个系统吞吐量的影响。 主机通过重放操作日志恢复其文件系统状态。 为了尽量减少启动时间，我们必须保持日志的小。 每当日志增长时，主检查其状态之后日志记录数量有限。 检查点是一个紧凑的B树状形式，可以直接映射到内存中，用于命名空间查找，而无需额外的解析。 这进一步加快了恢复速度并提高了可用性。 由于构建检查点可能需要一段时间，所以主的内部状态的结构可以在不延迟传入突变的情况下创建新的检查点。 主切换到一个新的日志文件，并在一个单独的线程中创建新的检查点。 新检查点包括切换前的所有突变。 它可以在一分钟左右为一个有几百万个文件的集群创建。 完成后，它将被写入本地和远程磁盘。 恢复只需要最新的完整检查点和后续日志文件。 旧的检查点和日志文件可以自由删除，尽管我们保留了一些以防灾难。 检查点期间的失败不会影响正确性，因为恢复代码检测和跳过不完整的检查点。

最初，GFS被认为是我们生产系统的后端文件系统。 随着时间的推移，这种用法演变为包括研究和开发任务。 它开始时几乎不支持权限和配额之类的东西，但现在包含了这些基本形式。 虽然生产系统有良好的纪律和控制，但用户有时没有。 需要更多的基础设施来防止用户相互干扰。 我们最大的问题是与磁盘和Linux相关的问题。 我们的许多磁盘向Linux驱动程序声称，它们支持一系列IDE协议版本，但实际上只对最近版本可靠地响应。 由于协议版本非常相似，这些驱动器大多工作，但偶尔不匹配会导致驱动器和内核对驱动器的状态不一致。 由于内核中的问题，这会悄悄地损坏数据。 这个问题促使我们使用校验和来检测数据损坏，同时我们修改内核来处理这些协议不匹配。

突变是一种操作，它改变块的内容或元数据，如写或附加操作。 每个突变都在所有块的副本中执行。 我们使用租约来保持跨副本的一致突变顺序。 主人将一个chunklease授予其中一个副本，我们称之为primary。 主选择对块的所有突变的串行顺序。 所有复制体在应用突变时都遵循这个顺序。 因此，全局突变顺序首先由主服务器选择的租约授予顺序定义，在租约中由主服务器分配的序列号定义。

租赁机制的设计是为了尽量减少主机的管理开销。 租约的初始超时时间为60秒。 但是，只要chunkis发生了突变，主服务器就可以无限期地请求并接收来自主服务器的扩展。 这些扩展请求和赠款都是基于主服务器和所有块服务器之间定期交换的心跳消息。 主程序有时可能试图在租约到期前撤销租约（例如，当主程序希望禁用正在重命名的文件上的突变时）。 即使主服务器失去了与主服务器的通信，它也可以在旧租约到期后安全地向另一个副本授予新租约。

在图2中，我们通过这些编号步骤遵循写的控制流程来说明这个过程。 1. 客户端询问主服务器，哪个chunkerver持有chunk的当前租约，以及其他副本的位置。 如果没有人有租约，主将一个授予它选择的副本（未显示）。 2. 主复用主副本的身份和其他（次要）副本的位置。 客户缓存这些数据以用于未来的突变。 它需要再次联系主人，只有当主变得无法到达或回复它不再持有租约。3. 客户端将数据推送到所有副本。 客户可以按任何顺序这样做。 每个chunkerver将数据存储在内部LRU缓冲缓存中，直到数据被使用或老化。 通过将数据流与控制流解耦，我们可以通过基于网络拓扑来调度昂贵的数据流来提高性能，而不管哪个块服务器是主要的。

1. 一旦所有副本都确认接收了数据，客户端就会向主服务器发送写请求。 请求标识先前推送到所有副本的数据。 主为它接收到的所有突变分配连续序列号，可能来自多个客户端，这提供了必要的序列化。 它以序列号顺序将突变应用于自己的局部状态。 5. 主程序将写请求转发给所有次要副本。 每个次级副本应用由主副本分配的相同序列号顺序的突变。
2. 第二代人都回复到初级，表示他们已经完成了操作。 7. 主要回复客户端。 在任何副本中遇到的任何错误都将报告给客户端。 在出现错误的情况下，写可能在次要副本的主副本和任意子集上成功。 （如果在初选时失败，就不会给它分配序列号并转发。） 客户端请求被认为失败，修改后的区域处于不一致状态。 我们的客户端代码通过重试失败的突变来处理这些错误。 它将在步骤（3)到步骤(7）中做一些尝试，然后从编写开始返回到重试。

GFS已经成功地满足了我们的存储需求，并且在Google内部被广泛用作研发和生产数据处理的存储平台。它是一个重要的工具，它使我们能够在整个网络的规模上继续创新和解决问题。