GFS 论文笔记

摘要

我们设计并实现了 Google File System，这是为大规模分布式数据密集型应用程序打造的一个可扩展 (scalable) 的分布式文件系统。虽然 GFS 运行在廉价的日用硬件上，但它仍然提供了容错功能，并且为大量客户端提供了总体上的高性能。

虽然 GFS 与之前的分布式文件系统有着很多相同的目标，但是，我们的设计已受到了当前和可预计的将来的应用工作负载和技术环境的驱使，这也反映出 GFS 和早期分布式文件系统的明显差距。这也驱使我们重新审视传统分布式文件系统的技术选择，探索极为不同的设计要点。

GFS 成功地满足了我们的存储需求。GFS 作为存储平台，被广泛部署在 Google 内部，用于数据的生成与处理，这些数据用于我们的各类服务中，以及研究和开发工作中所需的大规模数据集。迄今为止，最大的集群通过数千台机器上的数千磁盘提供了数百 TB 的存储服务，为数以百计的客户端所访问。

本文中，我们将展示用以支持分布式应用的文件系统接口扩展，并讨论设计的方方面面，最后给出小规模基准测试和真实使用环境下的测试结果。

1 INTRODUCTION

（第一段内容基本和摘要一致）

第一，组件故障是常态 (norm) 而非异常。文件系统由成百上千的存储机器组成，而这些机器是使用廉价的日用部件组装而成的；其次，有相当数量的客户端访问会访问文件系统。由于组件的大数量和低质量，几乎任何时刻都会有一些组件失效，并且无法从当前故障中恢复。我们遇到过各种 BUG，有的来自应用程序，有的来自操作系统，有的来自人为错误，以及磁盘、内存、网络甚至是电源供给的故障。因此，持续的监控，错误检测，容错，以及自动恢复功能必须集成到系统中。

第二，以传统标准衡量，GFS 的文件非常巨大，数 GB 的很常见。每个文件通常包含许多应用对象，比如 web 文档。当我们经常处理快速增长的数 TB 的数据集——这些数据集由数百万对象组成——的时候，将难以管理数百万的 KB 级的小文件，即使文件系统是支持的。这样一来，就必须重新考虑设计思路和参数设置，比如 I/O 操作和块数据大小。

第三，对大多数文件的修改是通过追加数据而不是覆盖已有数据，也不存在对文件的随机写。文件一经写入就成为只读的，而且通常只允许顺序读。各种类型的数据都遵循这一特性，这些数据中，有的构成了数据分析程序扫描的巨大数据仓库，有的是运行中的应用程序持续生成的数据流，有的是归档数据，有的是在一台机器上产生并在另一台机器上处理过程中生成的中间数据。有了这种针对大文件的访问模式，追加操作只需关注性能优化和原子性保证，而并不关注数据块的缓存。

第四，通过提高整个系统的灵活性，我们可以应用和文件系统 API 的协同设计从中受益。例如，我们降低了对 GFS 一致性模型的要求，在不给应用施加繁重负担的条件下极大地简化了文件系统设计。我们还引入了一种原子追加操作，使得多个客户端能够并发地向一个文件追加数据，而无需额外的同步。这些内容将在下文中详细讨论。

当前出于不同目的，我们部署了多个 GFS 集群。最大的一个拥有 1000 多个存储节点，300 多 TB 的磁盘存储，并且承受着来自不同机器数以百计的持续访问。

2 DESIGN OVERVIEW（设计概要）

2.1 Assumptions（假设/构思）