|  |
| --- |
| **云存储中低成本的版本控制策略** |

|  |  |
| --- | --- |
| 院（系）名 称 ： | 计算机学院 |
| 专 业 名 称 ： | 信息安全 |
| 学 生 姓 名 ： | 李旭贤 |
| 学号： | 2017202110080 |

摘要

云存储系统具有可靠性强及易于管理的优势。虽然传统的云存储系统运用了基于块的重复数据删除等技术，但是在存储数据和通信的过程中依然耗费了大量的成本，这一问题在提供版本控制的云存储系统中更加明显。

在本文中，作者用记录应用程序操作的日志文件代替原始文件数据，并对日志文件进行差值压缩（Delta Compression），减少了云存储系统的存储和通信成本。该方法的基本思想是，可以通过记录对文件的操作和重放恢复原始文件数据，而不用存储大量不同的版本。因此，作者设计了一个新型的文件系统，称作Knockoff。Knockoff压缩了数据和log文件，利用基于块的重复数据删除技术减少文件数据的存储量，利用差值压缩的方法对log文件进行压缩，并且提供了多种不同粒度的版本控制策略。在无版本控制的文件系统实验中，Knockoff在文件传输到云端的过程中分别减少了21%和24%的成本；在提供版本控制的文件系统中，随着版本粒度的增加，相对优势也更加明显。

关键词：云存储系统；重放；差值压缩；版本控制

**目录**

[一、绪论 1](#_Toc500614805)

[1.1 论文来源 1](#_Toc500614806)

[1.2 研究背景 1](#_Toc500614807)

[1.3 研究现状及相关工作 2](#_Toc500614808)

[二、示例 4](#_Toc500614809)

[三、Knockoff系统实现 5](#_Toc500614810)

[3.1 总体架构 5](#_Toc500614811)

[3.1.1 客户端 5](#_Toc500614812)

[3.1.2 服务器 6](#_Toc500614813)

[3.2 版本控制策略 6](#_Toc500614814)

[3.3 文件写入 7](#_Toc500614815)

[3.4 文件存储 8](#_Toc500614816)

[3.5 日志压缩 8](#_Toc500614817)

[四、实验评估 10](#_Toc500614818)

[4.1 实验设置 10](#_Toc500614819)

[4.2 实验结果展示 10](#_Toc500614820)

[五、总结 13](#_Toc500614821)

[参考文献 14](#_Toc500614822)

一、绪论

1.1 论文来源

本文来自于密歇根大学的Xianzheng Dou等人的“Knockoff: Cheap versions in the cloud”，主要研究了在云存储系统中如何提供多种粒度的版本控制策略，同时减少网络通信和文件存储的成本。

1.2 研究背景

云存储系统主要为用户提供文件存储和下载服务，当前有两个主要因素导致的云存储系统成本的增加。第一，由于云存储系统具有易于管理、用户成本低、可靠性强等特点，消费者存储越来越多的数据到云存储系统中，然而，从远端读写数据大大增加了网络通信的成本。第二，消费者希望存储同一文件的不同版本，这也导致了云存储系统存储成本的增加。很多云存储平台满足了用户的这一需求，比如Google Drive、Microsoft OneDrive和DropBox都允许用户存储和读取文件的旧版本。

提供文件的多版本控制能带来诸多好处，例如恢复丢失的或被覆盖的数据、再现数据产生的过程、审查和法律用途等。版本保存的越多，则带来的好处越多。比如，如果每次文件关闭时都保存一个版本，则用户在每次保存操作和应用程序关闭时都能得到文件的一个快照。然而，很多应用只能在程序关闭时保存文件，这导致文件修改过程中的即时数据无法进行恢复与分析，因此很多中间状态的文件只是暂存在内存中而未写入文件系统。在极端情况下，应允许用户对过去时间的任一文件状态进行重现，即超细粒度的版本控制。

版本控制的成本取决于保存版本的频率。比如，每次修改文件都保存一个版本比只在文件关闭时保存一个版本的成本更大，而且客户端需要消耗更多的带宽以传输多个版本的文件到服务器。因此，版本控制策略需要平衡多版本带来的益处和成本。考虑到成本，目前大多数现有的系统都选择了不频繁的版本控制策略。

为了解决上述问题，论文作者提出了一种非传统的方式来进行数据存储和通信。在该方法中，并非全部存储文件原始数据，而是有选择性的用一个日志文件来代替原始数据，该日志文件记录了文件的一系列操作信息（如系统调用结果、线程调度、外部数据读取等），通过这些操作可以根据日志重新计算出原始文件数据。此外，由于日志文件通常很大，作者采用了差值压缩的方式减小日志文件的数据量，减少传输和存储成本。

在分布式云存储系统中用日志文件代替原始数据具有很多优势：第一，用重放计算代替数据通信和存储，可以减少总体成本，因为在云存储系统中计算的成本小于存储和通信的成本；第二，日志文件比原始文件小时，传输日志文件可以减少带宽的消耗，同样，也可以减少服务端的存储量；第三，用日志代替原始文件内容，可以提供频率更高的版本控制策略。

本文的主要贡献如下：

1. 通过利用确定性的记录和重现，为分布式文件系统中的操作传输（Operation Shipping）提供了一种通用的解决方案；
2. 展示了如何将压缩应用于计算操作当中，并采用差值压缩减少了日志文件的大小；
3. 通过实际的文件系统的使用情况，量化分布式文件系统中通用的操作传输的成本和收益。

作者通过对多用户的软件开发场景和为期20天的单用户纵向研究来评估Knockoff。 在不提供版本控制时，Knockoff将文件发送到云端的平均成本分别降低了24％和21％。 随着保存版本的频率越来越高，使用Knockoff的收益也越高。 这种方法的耗费在于实现文件重现的性能开销（在我们的评估中为7-8％）以及在获取旧版本时更大的延迟（实验中设置为最多60s）。

1.3 研究现状及相关工作

Knockoff的基本原理是，可以通过原始数据或计算所需的输入日志来表示数据，即数据和计算之间是等价的。这种方法已经在许多环境中使用，例如容错系统、状态机复制、数据中心存储管理和状态同步。Knockoff中使用等价原则，旨在减少分布式文件系统中客户端和服务器之间的通信开销，并且减少服务端的存储成本。 Lee等人[1]首先在Coda文件系统中应用这个原则，并创造了术语“操作传输”（Operation Shipping）。客户端登录并将用户操作（例如shell命令）发送到服务器代理，以重复操作来生成数据。Chang等人[2]利用这种方法来记录和发送用户活动，如键盘和鼠标输入。虽然操作传输的基本思想是可行的，但是先前的系统记录和传输非确定性操作的类型非常有限，因此不能保证通过日志接收到的状态与原始状态相匹配。shell命令的日志和用户活动的日志都不足以重现通用程序的计算。研究人员认识到了这个缺点，并通过补充重放计算与前向纠错和补偿行动，使用哈希值检测剩余的差异并还原成数据传输。除此之外，多处理器和多线程程序使得在现有系统中很多程序是非确定性的，在之前的文件系统是无法处理的。而且，实现用户操作的重放，在记录和重放端需要相同的运行环境，这在许多客户端和服务器配置中是不现实的。

Adams等[3]认为重计算是一种减少存储的方法，但是没有任何该类文件系统进行实现和评估。其他系统在受限的环境中使用重新计算来减少存储，但必须保证计算是确定性的。Nectar[4]将这个想法应用于DryadLINQ应用程序中，并保证日志是确定性的和功能性的。BADFS[5]使用重新计算来代替复制数据，用户必须指定明确的依赖关系，并且计算必须是确定性的才能生成相同的数据。除了复制数据之外，文件系统中还使用日志记录来跟踪文件的来源，并指导何时保存新版本[6]。更一般地说，重做日志[7]在出现故障时提供事务属性。许多先前的系统对文件数据进行重复数据删除以减少通信和存储。LBFS[8]使用基于块的重复数据删除技术，使用 Rabin指纹将文件分割成块，为每个块计算哈希值，客户机和服务器使用哈希值来避免传输重复的数据块。在按值传输数据时，Knockoff使用与LBFS一样的重复数据删除技术。

版本控制文件系统以特定粒度保留过去的文件版本，例如文件关闭或者修改文件时保存文件。云存储提供商如DropBox和Google Drive目前允许用户保留以前版本的文件数据。 Knockoff通过减少存储和计算成本使版本控制文件系统更高效。它还支持比以前的系统更精细的版本控制。

二、示例

下面的示例说明了为什么存储日志文件可能只需要比存储原始数据少得多的空间。考虑一个简单的应用程序，它读入一个数据文件，计算这个数据的统计转换，并将一个时间戳写入输出文件。 输出数据的大小可能是很多兆字节。然而，程序本身可以通过一个确定性的小日志重现，如图1.1所示为一个简化的日志文件。

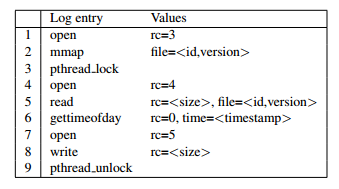


图1.1 非确定性日志的示例

日志记录系统调用（例如open）和同步操作（例如pthread\_lock）的结果。图1.1中的第一项记录了原始执行期间操作系统选择的文件描述符。第二项记录可执行文件的映射；重放此条目将导致在录制过程中使用的确切版本被映射到重放进程地址空间中的相同位置。第4行和第5行从输入文件读取数据，第6行记录原始时间戳，第7行和第8行将统计转换写入输出文件。要注意的是，从文件系统读取的数据不在日志中，因为Knockoff可以根据需要复制所需的文件。另外，写入到输出文件的数据不需要被记录，因为重放执行的结果会被完全重现。

使用差值压缩，此示例应用程序的日志只能有几百个字节的大小，与原始文件的兆字节数据相比，大大减少了数据传输量。输出数据通过从相同的初始状态开始，根据日志重新执行计算，可以恢复出完整的文件数据。

三、Knockoff系统实现

3.1 总体架构

图3.1展示了Knockoff的存储架构，包括客户端和服务端。

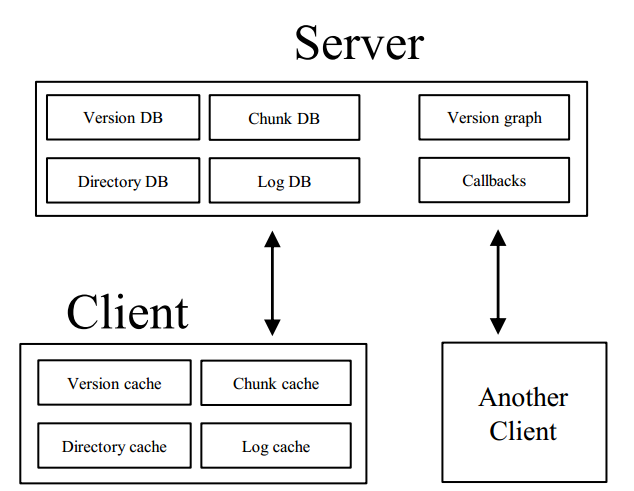


图3.1 Knockoff存储架构

**3.1.1 客户端**

Knockoff客户端将文件系统数据存储在四个持久的缓存中来减少网络延迟。整个文件版本存储在版本缓存（Version Cache）中；这个缓存可以同时保存同一个文件的多个版本。 Knockoff中的每个文件都有一个唯一的整数fileID，因此可以通过指定fileID和版本向量（Version Vector）来从缓存中检索特定版本的文件。版本缓存跟踪存储文件的当前版本。如果服务器产生了一个回调（因为另一个客户端已经更新了文件），该版本被保留在缓存中，但是作为当前版本的指针被删除。

块缓存（Chunk Cache）存储版本缓存中的每个文件产生的块，因此版本缓存仅包含指向块缓存中的块指针。Knockoff使用LBFS基于块的重复数据删除技术将数据库中的每个文件分成多个块，并计算每个块的SHA-512哈希值，块数据库由这些哈希值进行索引。

目录数据存储在目录缓存（Directory Cache）中。Knockoff客户端在数据上实现一个内存索引来加速路径查找。日志缓存（Log Cache）存储由记录程序产生的非确定性日志。

所有的客户端缓存都通过LRU管理，优先考虑保留当前版本。如果版本缓存中的任何版本不再使用，则会从块缓存中删除。只有修改的值被保存到服务器中时，才会保存到缓存中。

**3.1.2 服务器**

服务器维护与客户端类似的数据。服务器的版本数据库存储每个文件的当前版本，并根据版本策略确定过去的文件版本。块数据库存储版本数据库中所有文件的块数据，由块SHA-512哈希值进行索引。目录数据库存储Knockoff的所有目录信息，日志数据库存储客户端记录的非确定性日志。

如果版本控制策略是超细粒度的（Eidetic），则日志数据库将存储由Knockoff客户端记录的每个应用程序的日志；如果版本控制策略是写入时存储（Version on Write），则将每个过去的文件版本都存储在版本数据库中，或者将日志存储在日志数据库中；如果版本控制策略是关闭时存储（Version on Close），则只在文件关闭时进行存储；如果版本控制策略是无多版本控制（No Versioning），则仅存储文件和目录数据的当前版本。

此外，服务器还维护客户端为每个文件设置的回调，如果客户端更新文件，则服务器使用这些回调来确定向哪些客户端通知更新信息。

3.2 版本控制策略

Knockoff提供了四种版本控制策略：

1. No versioning

即无版本控制，Knockoff仅存储文件和日志的当前版本。客户端将关闭后的修改文件发送到服务器。在关闭之后，Knockoff最多延迟10秒钟将新版文件发送到服务器（这个延迟允许合并多个客户端的更新）。 在收到文件修改内容后，服务器将覆盖该文件的以前版本，并产生对其他客户端的回调。只有在产生冲突的情况下，服务器才会保留多个版本。

1. Version on Close

Knockoff在文件关闭时保留一个版本。对于过去版本，Knockoff保存原始文件数据或者日志文件以重放出原始数据。接收到一个文件修改后，服务端保留过去版本而不是覆盖它。客户端可以通过唯一的版本向量获取过去版本的文件。

1. Version on write

在每一个写入操作时保存一个版本。每一次修改文件的系统调用都会产生一个新的文件版本，Knockoff可以重放这些版本。

1. Eidetic

Knockoff以指令粒度保留所有以前的版本。 它可以重现任何计算或文件数据，并通过Arnold确定数据的来源。服务器存储所有应用程序日志，客户端可以通过指定版本向量和指定计数来指定文件的特定版本，该计数指定何时停止重播（以便为相应文件恢复特定的某一状态）。

3.3 文件写入

无版本控制时，Knockoff按如下方式处理写入操作，其他的控制策略与此类似。Knockoff将应用程序执行记录在重放组（Replay Group）中，该重放组记录相关进程的不确定性操作。当应用程序关闭文件时，Knockoff客户端会检查文件的版本向量以确定它是否被修改。如果被修改，Knockoff启动一个10秒的计时器，允许应用程序进行进一步的修改。当计时器到期时，Knockoff启动一个事务，在该事务中，该应用程序对文件的所有修改都被发送到服务器。如果应用程序终止，Knockoff立即启动事务。

在传输文件修改之前，Knockoff先计算传输log文件和重放文件的成本，成本计算方式如公式3-1所示。

(3-1)

其中，值log文件的大小，值传输单位大小文件的通信成本，表示重放文件所需的时间，则表示单位时间的计算所需的成本。

Knockoff按照公式3-2计算传输原始数据所需的成本，其中表示文件块的大小。

(3-2)

若，则将log文件传输到服务端，服务器复制修改后的文件并为其分配新的版本向量。然后，它产生通过log重放应用程序的进程。重放过程执行修改目标文件的系统调用时，会直接更新缓存中的版本。一旦重放达到文件关闭操作，新的缓存版本就完成并被标记为可读取。服务器删除旧版本的文件。重放过程在此时暂停，而不是终止，因为客户端可能会将更多日志数据发送到服务器以重新生成由同一个应用程序所做的其他修改。当客户端通知它正在重放的应用程序已经终止时，服务器会终止一个重放进程。

在极少数情况下，多个应用程序可能同时写入文件。 Knockoff服务器同时重放这些日志，并确保写入按照版本向量正确排序。

如果，则Knockoff可以通过将数据块发送到服务器来降低当前事务的成本。但是，对于长时间运行的应用程序,发送和重放收集的日志可能有助于降低将来文件修改的成本（因为从某一点开始重放的成本低于从程序的开始重放的成本）。Knockoff通过比较应用程序的比率来预测这一点。如果过去版本发送log是有利的，并且当前的应用程序行为与过去的执行相似（两次比率差小于40％），则发送日志。否则，它发送原始数据块。

3.4 文件存储

在存储文件时，若log文件的大小小于原始文件的大小，则可以存储log文件来减少存储成本，但是由于读文件时需要通过计算来重放原始文件，会增大时延。因此，在选择存储原始文件还是log文件需要进行折中的考虑。

在当前的版本中，Knockoff的存储策略如下。对于当前最新的文件版本，保存原始文件数据，以保证文件的读取性能不低于传统的文件系统。对于过去版本，Knockoff设置了一个最大重放时延（Maximum Materialization Delay），若重放所需的时间大于该时延，则存储原始文件，否则存储log文件，默认的最大时延是60s。

Knockoff使用服务器的版本图来判定如何存储文件。 版本图显示了文件版本和不确定性日志之间的关系。文件版本和日志形成图的顶点，图中的版本节点包含该版本中由不同系统调用写入的所有文件字节范围的列表。对于每个范围，节点将日志和特定系统调用存储在写入数据的日志中。重放所有这些日志足以重新计算该特定的版本。但是，版本数据库中已经存在的字节范围不需要重新计算。如果节点所表示的版本本身在版本数据库中，则所有字节范围都存在。否则，特定的字节范围可能仍然在版本数据库中，因为同一文件的另一个版本是按值存储的，并且范围在两个版本之间没有被覆盖。

如果日志包含写入不在版本数据库中的字节范围的系统调用，则Knockoff将版本节点的边缘插入日志节点。每个边的权重等于重新计算所有这些字节范围的时间。可以通过版本图来计算重放某一版本文件所需的最长时间。

3.5 日志压缩

在存储原始文件数据时，采用基于块的重复数据删除技术有很好的效果，大大减小了数据的存储量，这是由于文件中很多的数据块是相同的，只需存储一份即可。而对于log文件来说，只存在极少的相同块，因为log文件中通常带有时间戳，而每个时间戳是不一样的值。然而，log文件的不同块中依然存在极高的相似性，如图3.2所示。根据这一特点，可以采用差值压缩的方法，减少log文件的数据量。

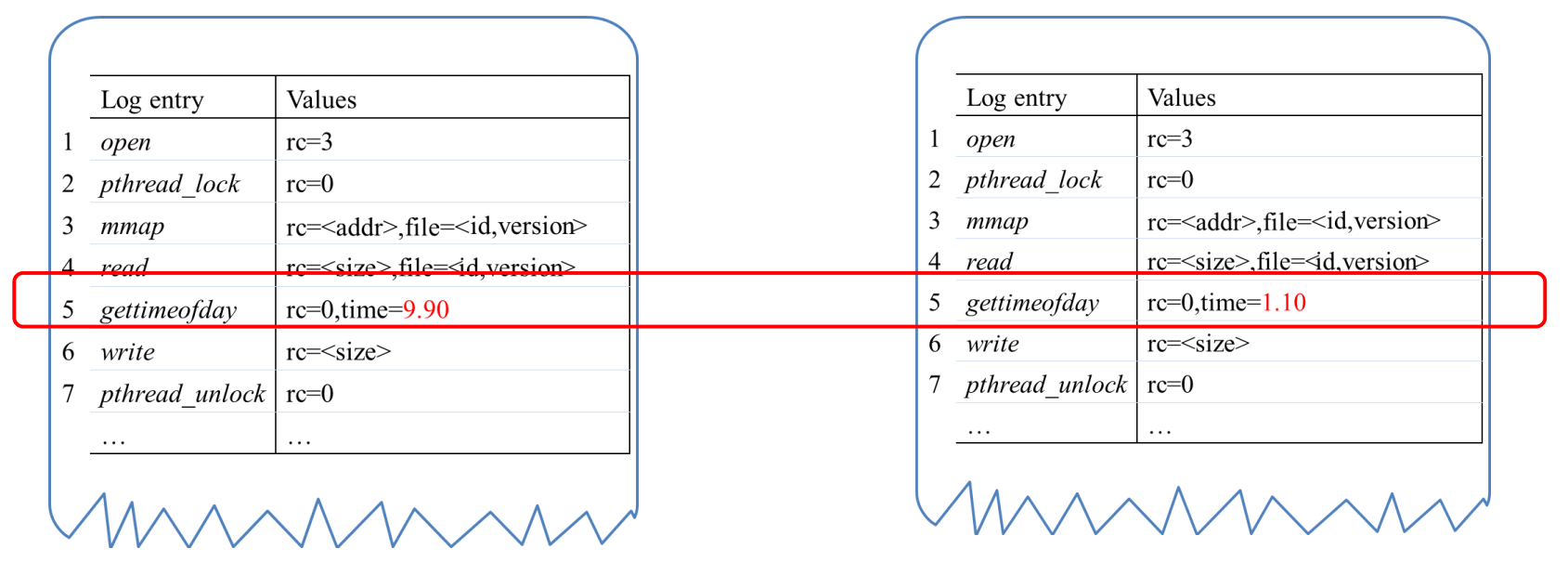


图3.2 log文件示例

四、实验评估

作者的实验主要研究四个问题：

1. Knockoff在云存储系统中减少了多少使用的带宽；
2. Knockoff在云存储系统减少了多少通信和存储成本；
3. Knockoff的性能开销如何；
4. 日志压缩的效果如何。

4.1 实验设置

（1）多用户研究

8位参与者，在一小时内运行几个简单任务。

（2）单用户研究

单一用户，为期20天的研究，并对多种Linux应用程序、文本编辑器、编程语言进行对比实验。

4.2 实验结果展示

（1）带宽使用

作者将Knockoff的带宽使用情况与基于重复块删除的文件系统进行对比，结果如图4.1所示，可知Knockoff减少了24%-47%的带宽使用。

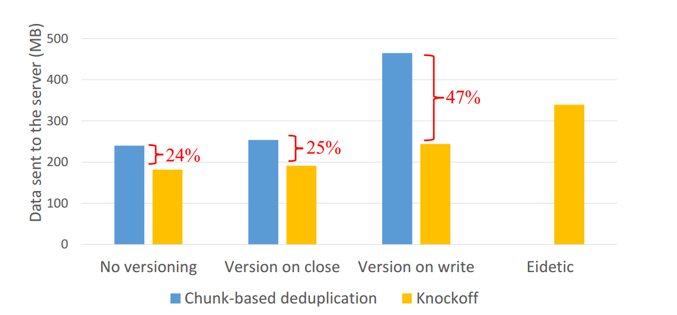


图4.1 带宽使用对比

（2）应用程序差异

作者对不同应用程序的带宽使用情况进行了对比，其结果如图4.2所示，可知在Knockoff文件系统中，不同的应用程序对带宽的节省程度是不一样的。

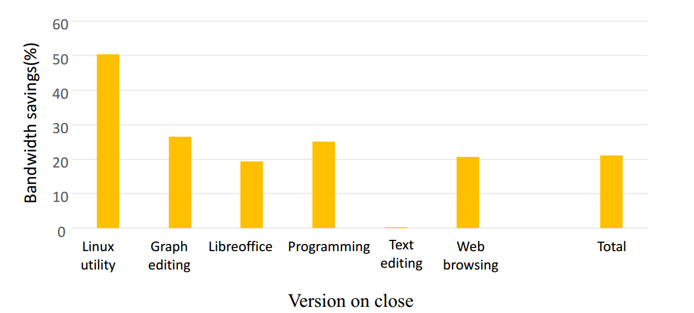


图4.2 不同应用程序的带宽节省程度对比

（3）存储成本

作者将Knockoff文件系统与基于重复块删除技术在存储成本上进行了对比，如图4.3所示。在version on close和version on write的粒度下，分别减少了19%和23%的存储成本，效果很出色。

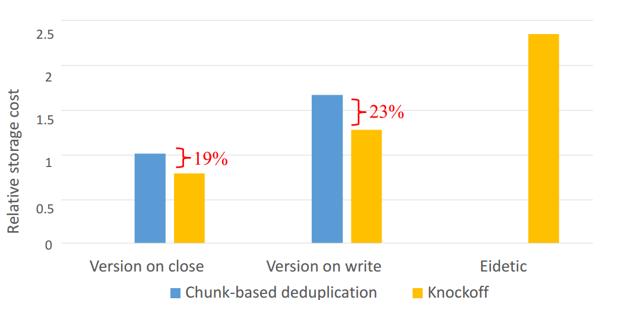


图4.3 相对存储成本对比

（4）性能开销

最后，作者对Knockoff文件系统的性能开销进行了测试，如图4.4所示。相对于传统的云存储文件系统，Knockoff的性能开销增加了7%左右，因为进行文件重计算增加了服务端的计算量，性能开销增加，但是在可接受的范围内。

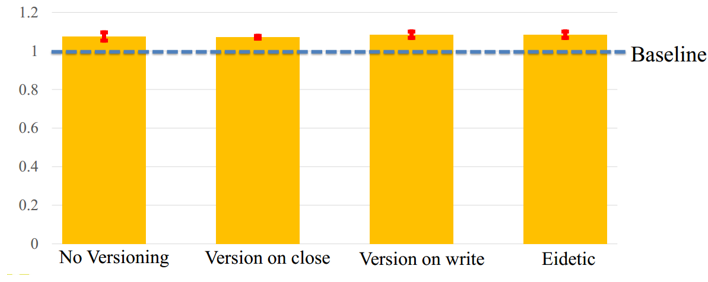


图4.4 Knockoff的性能开销

五、总结

长期以来，操作传输（Operation Shipping）一直被认为是一种可行的降低分布式存储系统成本的技术。 然而，在实际运用中会受到计算平台的应用确定性或标准化的限制，这些限制使得操作传输不适合通用文件系统。Knockoff利用最近在确定性记录和重放方面的进展来解除这些限制。它用日志文件的方式来存储和重放原始文件， 这节省了网络通信并降低了存储空间，从而节省了成本。今后，有望将Knockoff的思想扩展到其他用途，比如减少跨数据中心的通信成本等。

参考文献

[1] Chen L, Lehoczky J, Rajkumar R, et al. On Quality of Service Optimization with Discrete QoS Options[C]// Real-Time Technology and Applications Symposium, 1999. Proceedings of the Fifth IEEE. IEEE, 2002:276.

[2] Chang T Y, Velayutham A, Sivakumar R. Mimic: raw activity shipping for file synchronization in mobile file systems[C]// International Conference on Mobile Systems, Applications, and Services. ACM, 2004:165-176.

[3] Adams I F, Long D D E, Storer M W, et al. Maximizing efficiency by trading storage for computation[C]// Conference on Hot Topics in Cloud Computing. USENIX Association, 2009:17.

[4] Gunda P K, Ravindranath L, Thekkath C A, et al. Nectar: automatic management of data and computation in datacenters[C]// Usenix Conference on Operating Systems Design and Implementation. USENIX Association, 2010:1-8.

[5] John Bent, Douglas Thain, Andrea C. Arpaci-Dusseau, et al. Explicit Control in a Batch-Aware Distributed File System[J]. Dusseau, 2004, 1.

[6] Muniswamyreddy K K, Holland D A. Causality-based versioning[J]. Acm Transactions on Storage, 2009, 5(4):1-28. [7] ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging

[8] Muthitacharoen A, Chen B, Mazières D. A low-bandwidth network file system[C]// Eighteenth ACM Symposium on Operating Systems Principles. ACM, 2001:174-187.