海量存储结课论文

WAFL写入复制文件系统中

高性能的元数据

完整性保护方法

院（系）名 称：计算机学院

专 业 名 称 ：软件工程

学 生 姓 名 ：吴璠

指 导 教 师 ：

二○一七年十二月

摘 要

我们引入了低成本的增量校验和技术，保护元数据块免受内存中的涂鸦，以及一个轻量级的能强制执行文件系统一致性的基于摘要的事务审计机制。与以前的工作相比，我们的技术将性能开销降低了一个数量级，同时还帮助区分杂文和逻辑错误。我们还提供了一个机制来查明生产系统上的涂鸦原因。我们的技术已经在一些文件系统中产品化，性能开销微乎其微，在数百万运行时间的基础上，大大减少了过去五年中损坏相关的事件。

**ABSTRACT**

We introduce a low-cost incremental checksum technique that protects metadata blocks against in-memory scribbles, and a lightweight digest-based transaction auditing mechanism that enforces file system consistency invariants. Compared with previous work, our techniques reduce performance overhead by an order of magnitude. They also help distinguish scribbles from logic bugs. We also present a mechanism to pinpoint the cause of scribbles on production systems. Our techniques have been productized in the NetApp® WAFL® (Write Anywhere File Layout) file system with negligible performance overhead, greatly reducing corruption-related incidents over the past five years, based on millions of runtime hours.

目 录

[1 绪论 5](#_Toc500100379)

[2 动机 6](#_Toc500100380)

[2.1元数据编写时损坏 6](#_Toc500100381)

[2.2涂鸦与逻辑错误 6](#_Toc500100382)

[2.3 造成数据损坏的原因 7](#_Toc500100383)

[2.4COW文件系统的Recon机制 7](#_Toc500100384)

[3 方法 8](#_Toc500100385)

[3.1 解决方案概览 8](#_Toc500100386)

[4 防乱写 8](#_Toc500100387)

[4.1 end-to-end 校验和 8](#_Toc500100388)

[4.2校验和初始化 9](#_Toc500100389)

[4.3 校验和递增的更新 9](#_Toc500100390)

[4.4 无锁的多处理器更新 10](#_Toc500100391)

[4.5 校验和证明 10](#_Toc500100392)

[4.6使用页面保护实行检测 11](#_Toc500100393)

[5 事务审计 12](#_Toc500100394)

[5.1基于摘要的审计 12](#_Toc500100395)

[5.2审计的性能 12](#_Toc500100396)

[6 实现 13](#_Toc500100397)

[7 评估 14](#_Toc500100398)

[7.1增量校验和性能 14](#_Toc500100399)

[7.2充分保护性能分析 14](#_Toc500100400)

[7.3乱写检测 15](#_Toc500100401)

[8 结论 15](#_Toc500100402)

# 1 绪论

存储系统包括不可靠的硬件组件，如磁盘，磁盘架，存储互连结构，RAM，CPU和数据传输总线。这个硬件是由一个软件堆栈或一个专门的存储操作系统驱动的，这个操作系统是建立在ext3，ext4，ZFS，btrfs或WAFL。该软件是由异构组件构建的，由于可能影响软件生态系统其他部分的固有缺陷，这些组件可能也不可靠。软件中的硬件故障和错误都会破坏数据。文件系统必须提供检测，避免和从这种损坏中恢复的机制。一般而言，数据在写入永久介质之前可能会被损坏，而数据则驻留在介质上或读取路径中。静止数据通过使用诸如校验和和清理[53]等检测技术以及通过使用诸如冗余的恢复技术来防止介质故障。文件系统崩溃一致性由诸如日志等技术提供，影子分页或软更新。但是，通过软件错误或硬件故障或文件中的逻辑错误系统代码造成的内存涂鸦仍然可能引入元数据不一致在写入路径中。交易审计机制Recon[1]是一个很有希望能提高写入完整性的方法。然而，使用Recon的原始版本的元数据块必须被缓存，并且随后要与修改后的副本进行比较进行审计。在写时复制文件系统中，如WAFL，这个要求可能会导致严重的性能回归，特别是在元数据繁重的工作负载下。使用高度优化的WAFL实现侦察式审计导致不可接受的30％吞吐量回归关键工作负载。此外，侦察式审计无法区分损坏这是由内存涂鸦造成的还是由逻辑错误引起的。对于工业规模的部署，能够区分这两个原因对于快速解决损坏问题至关重要。在本文中，我们介绍两种新颖的技术组合可以更多地检测任意的不一致可以比Recon更有效率。他们也提供了至关重要的区分杂文和逻辑错误的能力。首先，我们通过介绍一个单一的滚动校验和每个元数据块的生命周期（是否在内存中或在持久媒体）以保护它免受随机涂鸦。接下来，我们介绍一个基于摘要的交易审计系统来防止逻辑错误。不像Recon，基于摘要的审计不需要缓存原始版本的元数据块，因此可以实现性能开销忽略不计。通过为涂鸦和逻辑错误提供单独的解决方案，我们可以根据需要优先执行审计不变量投资回报，这对于一个很大的投资是无价的WAFL等复杂的文件系统实现。另外，文件系统通过两种技术被双重保护免受涂鸦的影响。最后，我们还介绍了一个低成本的页面保护技术能探查出由软件造成的涂鸦的根本原因，从而迅速解决损坏的错误。

本文的主要贡献是：（1）端到端的元数据校验和，（2）基于轻量级的摘要交易审计，（3）针对涂鸦代码路径的细粒度页面保护。

# 2 动机

NetApp是一家存储和数据管理公司。它提供软件，系统和服务进行管理存储数据，包括其旗舰ONTAP操作系统。 ONTAP实施专有文件系统称为WAFL（随处写入文件布局）。世界各地部署了许多ONTAP系统，以及过去二十年来所引发的各种硬件和软件缺陷。发现不一致时，文件系统被标记为不一致，并启动文件系统检查。WAFL提供在线[35]和离线文件系统一致性检查。脱机检查涉及显著的停机时间，它和数据集的大小成正比。在线检查减少了停机时间但会影响系统性能，直到完成。在很多情况下，检查最终会发现数据丢失，文件损坏，甚至会建议通过备份来恢复。主要目标是为了客户和NetApp的支持成本巩固WAFL抵御不一致以减少中断。

## 2.1元数据编写时损坏

两种类型的问题可能导致文件系统恢复运行：（1）由于写入路径中的元数据损坏内存损坏导致的不一致，以及（2）由于媒体故障导致元数据丢失这超出了底层RAID机制的冗余门槛出现的不一致。 本文讨论了问题1.请注意，持久块校验和无法保护从内存中的元数据损坏，因为校验和计算发生在块被潦草之后[64]。写路径中的元数据损坏可能是由于三个主要原因：（1）元数据计算和更新中的逻辑错误，（2）元数据块的涂写被用作新的元数据计算的输入，并且（3）在元数据块被写入之前对其进行涂抹持久存储。

## 2.2涂鸦与逻辑错误

让我们首先区分两种损坏类型错误：涂鸦和逻辑错误。一个涂鸦（Heisenbug）[29]覆盖任意数据元件。它通常是随机发生的，可能性与损坏任何数据或元数据相同，产生不可预知的结果。由于其不可预知的性质，一个涂鸦是系统测试难以复制的。由于观察到的症状很远，因此很难诊断从原来的位置和时间删除它的发生。误弄一个逻辑错误的涂鸦可能会导致一个无用的错误追逐，以及工程资源的浪费。通常，涂鸦是已知但未固定的错误的原因这对产品质量产生不利影响。大多数硬件涂鸦不被发现。ECC内存可以检测故障，但有一些限制，而且可能并不总是使用，因为它的成本。软件引起的涂鸦可以使用处理器的页面保护机制来检测，但是也是一样的成本和限制很大。相反，一个逻辑错误（Bohrbug）是固有的元数据计算和更新逻辑。一个计算与逻辑错误产生不正确的结果，但商店它到了自己的内存位置。所以，一个逻辑错误是更可预测，影响有限。观察到的症状可预见地局限于特定的行为方面，这使诊断更容易。

## 2.3 造成数据损坏的原因

软件的一些问题例如缓冲区溢出、不正确的指针计算和在共享的地址空间中的指针悬挂会导致涂鸦的产生，在WAFI中，文件系统缓冲区高速缓存存储器可以被分配到其他的用途然后被回收，这使得包括元数据缓冲区的文件缓冲区会成为软件涂鸦的受害者。涂鸦也可以是由于硬件故障导致的，比如内存错误，DMA错误或者是CPU寄存器未被发现的位翻转。涂写错误并不是很少见，由于硬件故障导致的内存涂鸦预计将来会更常见[14,55]。

## 2.4COW文件系统的Recon机制

Recon是一个事务审计机制，它能在事务提交之前保证所有文件系统一致性不变量。Recon检查文件系统下面的物理块，并在读写元数据块是推断它们的类型。Recon机制允许解析和解释块，类似于语义智能磁盘。但是，由于每次交易增加的元数据开销，Recon的设计中的元数据缓存和比较使对于诸如WAFL的COW文件系统是变得不可持续。

在WAFL中，每个写入用户数据块都会让该文件系统读取相应的父间接块，将旧的块指针释放到这个用户数据块，然后分配一个新的块指针。这个过程递归构成文件系统映像的块的树，全部路径向上到超级块。因此，写入一个用户数据块可能需要更多的读写一个元数据块。即使WAFL通过在一个单独的批处理众多的操作开销交易摊销这一点，类似Recon的审计的开销仍然特别高。

# 3 方法

## 3.1 解决方案概览

内存中元数据校验和：为了区分的内存中乱写与逻辑错误找出脏乱块，我们使用内存中的一般校验和保护元数据块。在块被写入持久性存储之前，校验和能验证块是否被乱写了。

元数据页面保护：即使校验和验证可以防止涂鸦持续存在，但它不能找出最初错误的代码。我们使用处理器辅助页面细粒度的保护来捕捉错误源头的代码。由于这种方法有更高的执行代价，我们建议在当元数据块校验和失败是可重现时在客户系统上启用它。

事务审计：通过比较事务中发生的变化的摘要确认一致性不变量。创建和比较摘要是非常有效的。当元数据块被修改时它的更改将被添加到摘要中。因此，我们不需要缓存原始块的副本。

通过为涂写检测提供单独的解决方案，我们可以灵活地优先执行审计不变量，从而获得更高的投资回报。我们可以从易出现逻辑错误的代码路径开始，如复杂度较高的代码路径或代码路径对应于较新的未经测试的特征。

# 4 防乱写

## 4.1 end-to-end 校验和

我们使用单个滚动或增量校验和保护每个元数据块的整个生命周期，无论是在内存中还是在持久存储中。当一个块从永久性存储中读入时，存储的块的校验和也被读取，然后由文件系统验证。而在内存中，这证实了校验和用于保护块，并通过块的每个合法更新逐步更新。当一个块作为事务的一个部分被写入静态存储中，文件系统会计算出一个新的校验和。这个新计算的校验和用于验证增量更新的内存校验和，如果验证失败，我们知道该块被乱写。这种情况下块仍然受到保护，无论是在内存还是持久存储中。

## 4.2校验和初始化

一个事务中写出来的元数据块不是新创建的就是之前从永久存储中读取出来但是在整个事务的过程中被修改的已经存在的块。当一个块存在于内存上时我们立刻初始化内存中的校验和，如果是新创建的，则根据相应元数据的初始状态将校验和初始化为可导出的常量值。如果从存储器读入，则校验和被初始化为存储和验证的校验和。

## 4.3 校验和递增的更新

内存中元数据块的合法修改通常是通过调用在文件系统代码中定义好的API来完成。我们在每个API中添加一个钩子使相应的块校验和在变动上与时间保持一致。 在块被修改之前，通过使用旧的校验和增量计算新的校验和，然后覆盖原始数据，原始数据在块中的位置放置新数据。在接下来的几段中，我们精确地展示当一个块中的单个字节被修改时如何递增地重新计算Adler[23]校验和。

在下面的等式中，“div”和“mod”表示运算符分别返回一个商和一个余数。 假设一个数据块D由n个字节D1，D2，...，Dn组成; Adler32方法计算D的校验和方法如下：

*A* = (1+*D*1 +*D*2 + …+*Dn*) mod 65521  
*B* = (*n×D*1 + (*n-*1)*×D*2 + …+*Dn* +*n*) mod 65521  
*Adler32*(*D*) = *A*+*B×*65536

增量校验和更新是最佳的，因为它需要额外的内存访问才能读取原始内容，读取旧的校验和，并更新校验和。在大多数情况下，文件系统代码在覆盖之前读取元数据；因此，原来的内容可能在处理器中缓存。计算新校验和的CPU成本与修改的数据量成比例。在内存开销方面，这个方法需要一个每个数据块附加4个字节以保持滚动校验和。

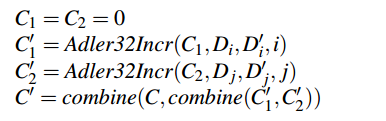
可以使用更强大的64位校验和。因为它们适合于这种增量计算的形式，Adler，Fletcher或者任何其他的与位置有关的校验和都可以使用。对于WAFL，我们使用了一个修改版本和之前展示的高度优化的实现了简单的增量校验和计算。

## 4.4 无锁的多处理器更新

WAFL被设计为在多个处理器上同时运行。因此，元数据块被多个处理器并行修改是相当普遍的。然而，为了尽量减少高速缓存线抖动和昂贵的锁争用，WAFL多处理器编程模型避免了使用自旋锁可能。为了避免锁争用，一个增量校验和块可以跨处理器分割，并以一个无锁的方式更新。每个处理器计算并累积增量校验和，以便自己在每个处理器校验和字段中更新。每个处理器被结合起来导出最后的增量校验和。两个独立的校验和片段C1和C2可以联系起来如下：

*A* = (*C*1 mod 65536+*C*2 mod 65536) mod 65521  
*B* = (*C*1 div 65536+*C*2 div 65536) mod 65521  
*combine*(*C*1*;C*2) = *A*+ *B×*65536

两个同时修改字节Di和Dj的处理器可以维护它们各自的每个处理器的校验和片段C1和C2。每个片段都使用前一节中的Adler32Incr独立更新。在验证之前，碎片与块的原始校验和C结合起来一起计算最终校验和C’：



但是，每个处理器校验和片段需要每个块额外的内存（4个字节）。这额外的内存是相当值得的权衡的，因为它可以节省我们获得和争夺自旋锁的成本。

## 4.5 校验和证明

当一个块被写出来作为永久性存储的一部分，我们重新计算整个块的校验和并将其与增量校验和进行比较。如果两个校验和不匹配，我们检测到该块上存在涂鸦。该验证可以在之前或之后执行该块的写入I / O被发送到存储器。在里面前一种情况，可以优化校验和计算通过将其与RAID奇偶校验计算相结合。但是，这种方法在校验和计算之后打开了一个不可检测的窗口，但在之前涂写写入I / O。写入I / O后校验和的验证将关闭该窗口。在任何情况下，数据传输所注入的损坏同时它服务的写入I / O无法被检测到直到块被再次读取。在检测到涂鸦后，我们中止正在进行的事务提交，防止数据损坏持续。为了保护ONTAP节点免受来自同一错误的任何其他潜在损坏，我们重新启动节点而不是放弃一个单独的事务。因为ONTAP配置为高可用性对，相邻节点获取重新引导节点的文件系统的所有权，而这些文件系统仍然是一致的，因为它们是由最近完成的事务。合作节点然后重放记录在NVRAM日志中的用户操作，以及之前的操作并从存储中读取一致的元数据，进行修改，然后作为一个全新的事务的一部分。

## 4.6使用页面保护实行检测

校验和验证失败表明该块已被乱写，但不能找出出错代码路径，因为涂鸦可能已经发生很长时间了。如果出错代码路径共享相同的地址空间那么定期地址空间保护不能起作用；ONTAP中的许多内核代码都是如此。如果一个涂鸦是可复写的，我们提供一个选项使元数据页面保护能直接找出代码路径。为了在重要和频繁的元数据修改时可以提供良好的表现，我们结合了页面级保护和x86-64处理器中的WriteProtect Enable（WP）位。为了保护元数据块免受乱写，我们保持个人的元数据页面默认为只读。一个可以安全更新的方法是在一个合法的更新操作者在修改块之前标记通信页面为只读模式。当请求的修改完成时，页面可以被标记为只读的。因此，一个非法的修改找到页面只读，生成一个页面错误，并产生指向涂鸦者的堆栈轨迹。然而，如果元数据经常被修改，这个方案就不能很好地执行。频繁切换页面权限（只读以读取/写入到只读）不仅造成TLB冲击，而且造成了一个处理器间TLB失效中断风暴。在我们的实验中使用这个方案来保护所有的间接块和位图阻止大约70％的性能下降，从而使该方案即使在调试模式下无法使用。为了减少性能影响，我们保留页面只读的所有的时间。要启用合法写入，我们禁用全局保护（通过翻转WP位），修改后重新启用。有了这个方法，性能下降归结为约20％，这对于诊断模式是可接受的。当CPU核心禁止写保护时，它可以写信给任何地址。

# 5 事务审计

## 5.1基于摘要的审计

基于摘要的审计有助于验证分布式一致性不变的价格。关键的想法是创建摘要并交叉验证摘要而不是个别更改。这个战略大大减少了在验证和仍然需要的工作量提供强有力的保证，在实践中有用。我们拦截间接和位图块的修改通过在修改API中使用上述钩子来执行操作，并且创建这些修改的摘要。一个间接块包含64位块数字B1; B2;… Bn。假设块号Bi，Bj，和Bk在交易中由新分配的块号Ni，Nj和Nk替换。 API中的钩子修改了间接块更新校验和原始块号码在一个自由摘要和新的块分配的摘要中的数字;每个摘要都保持不变每个事务的所有更新的元数据文件系统：

S*IndFree* = *Bi* +*Bj* +*Bk* + …S*IndAlloc* = *Ni* +*Nj* +*Nk* + …

同样，在位图块修改API中挂钩维护所有块号的每个事务摘要其相应位从0翻转到1（新分配），反之亦然（释放）:

S*BitmapFree* = *Bi* +*B j* +*Bk* + …S*BitmapAlloc* = *Ni* +*Nj* +*Nk* +…

在交易结束时，审计验证了两套摘要相互吻合

S*IndFree* == S*BitmapFree*S*IndAlloc* == S*BitmapAlloc*

## 5.2审计的性能

基于摘要的审计性能比Reconlike审计好得多，因为摘要的计算成本低廉而元数据块不需要被缓存和比较。具体来说：（1）我们添加到摘要中的数据在处理器的缓存中很容易获得，因为它刚刚被访问; （2）摘要更新只涉及到每个添加操作和一个内存访问元数据更新; （3）由于缓冲区缓存未命中而没有昂贵的I / O操作; （4）摘要是随着每个元数据更新操作逐步创建，这消除了我们独立识别和验证所有更改的独占阶段的需要。请注意，摘要更新与块的增量校验和更新使用相同的钩。因此，我们有效地使用一个内存访问来做三件事：（1）修改间接块，（2）更新审计摘要，（3）更新增量校验和。最后的验证是一个便宜几个字节的比较。基于摘要的位图和间接块的审计变化将交易审计的整体成本从30％的吞吐量回归大大降低到了较低的水平我们的数据库/ OLTP基准测试结果。

# 6 实现

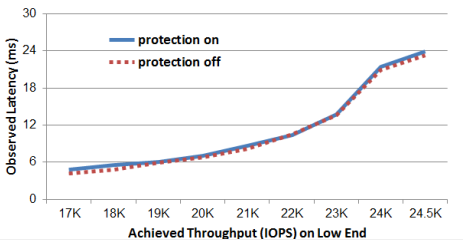
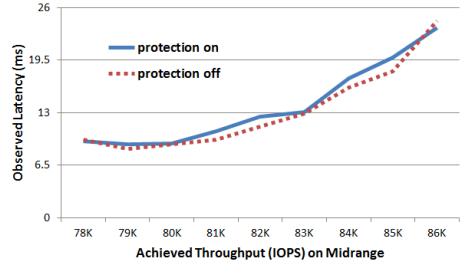
拦截修改：所有三个功能，增量校验和；页面细化保护和事务审计；必须拦截对受保护的元数据块的所有修改。我们插入一个统一的钩子用于更新元数据的每个合法API。API提供所有必要的参数到钩子更新增量校验和，更新相应的摘要，并切换内存保护。元数据块的所有修改都经过了众所周知的WAFL APIs，所以很容易插入我们的挂钩在这些API中。但是，给了两个以上数十年的代码增长和流失，有一个代码中隐藏着一些隐蔽的地方直接更新元数据块;我们使用页面保护功能来查找它们。所有的元数据页面默认是只读的，并且需要挂钩来切换模式。当功能打开时，任何更新来自不明确的代码路径的元数据立即生成了页面错误，并且启用了有用的堆栈跟踪。

我们模块化代码路径并插入钩子。特殊的优化：两个常见的修改一个文件系统的元数据是：（1）块的更新指针在间接块中，（2）翻转位图来表示分配或释放的状态块。我们为这些情况编写了自定义的，优化的校验和计算例程，即修改了a固定长度块指针（在WAFL的情况下是64位）并修改一个位图位。对于他人更新，我们创建了一个广义的增量校验和计算例程是基于偏移量（在块）和更新的长度，以便它可以处理可变长度的修改。一个例子的使用这种一般化的计算是文件删除处理，这需要批量更新元数据块。复杂性：拦截每个元数据块的修改有点侵入性，但它使代码更加复杂模块化并促进了更好的开发实践。上一方面，我们的实现比Recon更复杂，因为我们截取了每个元数据块的修改。但另一方面更简单，因为我们不必实现缓存因此避免了许多与之相关的问题缓存。

# 7 评估

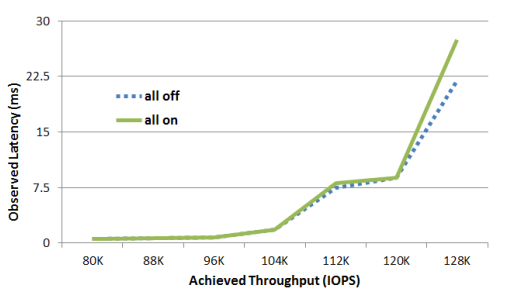
## 7.1增量校验和性能

下面左右两张图分别显示了这个成本，左边具有12个英特尔®Westmere的中端系统内核，96GB DRAM和4GB NVRAM；和右边具有4个英特尔Wolfdale内核，20GB DRAM的低端系统，和2GB的NVRAM。足够数量的SAS很难磁盘被连接到两个系统，以消除任何存储瓶颈。实验运行使用我们的数据库/带有增量校验和的OLTP基准测试打开并关闭，观察到的延迟是绘制了与实现的IOPS吞吐量增加输入IOPS负载。我们只绘制了负载延迟小于或等于30ms的点，这是SPC-1基准允许的最大值。我们看到在-0.9ms到10ms范围内的延迟增加中端和低端为1.7ms和零到0.8ms系统；这些范围转换为-3.5％-10.5％和零到17％。在低端系统上，吞吐量回归从5％（5ms）到零（24ms）。处理待处理的文件删除[37]，WAFL必须异步走几次间接块，清除块指针，并大量更新相关元数据。



## 7.2充分保护性能分析

我们测量性能开销审计和增量校验和一起工作。如前面第5.2节中所解释的那样，CPU成本保持审计摘要是微不足道的，成本通过将其与增量校验和更新组合来进一步最小化。内存开销很小每个审计不变只需要几个字节的摘要。相比中端来说图1系统采用的系统如图2所示是一个高端系统，具有CPU周期，下一代英特尔芯片组，更大的处理器缓存，更多DRAM和固态硬盘。

图2

## 7.3乱写检测

默认情况下，页面保护在发货时处于关闭状态产品;它可以在需要诊断时打开。如果一个系统反复出现损坏错误，另外两个保护机制阻止它被写入永久存储器。然而，重复文件系统重新启动可能会对客户造成干扰。因为流氓代码路径或线程已经运行在检测到涂鸦时完成，通常需要大约一个人月的高级开发者时间才能找到平均记忆涂抹的根本原因错误。但是，如果客户愿意承担20％，通过打开页面保护来降低性能根本原因出现在由此产生的核心转储中生成页面错误。

# 8 结论

文中引入了两种技术，即增量校验和以及基于摘要的审计，可以防止内存中的涂鸦和逻辑错误破坏持久的文件系统元数据。文中反驳了普遍持有的信念强大的数据完整性需要高性能罚款;我们用可忽略不计的绩效税来实现诚信。我们将杂文与逻辑错误区分开来，并提供诊断功能以查明问题软件涂鸦的罪魁祸首。这些技术大大提高了数据的完整性WAFL，导致恢复运行空前减少。通过在开发周期的早期阶段发现腐败，这些技术已经使我们的工程师快速创新而不冒数据完整性的风险。我们相信端到端的增量校验可以应用于用户数据块，从而提供低成本的往返应用级保护。这个技术在保护托管在第三方基础架构上的应用程序时尤其有用硬件的可靠性无法建立或保证。而且，持续的校验保护可以加强对诱导腐败的申请攻击共享云基础架构。数据库和文件系统托管在面向附件或云上存储是潜在的好处的例子端到端的保护。

# 参考文献

[1] [Harendra Kumar](http://dblp.org/pers/hd/k/Kumar:Harendra), [Yuvraj Patel](http://dblp.org/pers/hd/p/Patel:Yuvraj), [Ram Kesavan](http://dblp.org/pers/hd/k/Kesavan:Ram), [Sumith Makam](http://dblp.org/pers/hd/m/Makam:Sumith): High Performance Metadata Integrity Protection in the WAFL Copy-on-Write File System FAST 2107 : 197-212

[2] Daniel Fryer, Kuei Sun, Rahat Mahmood, TingHao Cheng, Shaun Benjamin, Ashvin Goel, and Angela Demke Brown. Recon: Verifying file system consistency at runtime. In *FAST ’12*, San Jose, CA,February 2012.

[3] James Hamilton. Successfully Challenging the Server Tax. <http://perspectives.mvdirona.com/2009/09/03/> SuccessfullyChallengingTheServerTax.aspx

[4] Lakshmi N. Bairavasundaram, Garth R. Goodson, Bianca Schroeder, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. An Analysis of Data Corruption in the Storage Stack. In *FAST*, 2008.

[5] Emery D. Berger and Benjamin G. Zorn. Diehard:Probabilistic memory safety for unsafe languages. In *Proceedings of the 27th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI ’06, pages 158–168, New York, NY, USA, 2006. ACM.