|  |  |
| --- | --- |
| 学号 | 2017282110236 |
| 密级 |  |

**武汉大学课程结业论文**

|  |
| --- |
| **应对文件系统故障--分布式响应机制** |

|  |  |
| --- | --- |
| 院（系）名 称 ： | 计算机学院 |
| 专 业 名 称 ： | 计算机技术 |
| 学 生 姓 名 ： | 王俊秋 |

二〇一七年十二月

**郑 重 声 明**

在完成海量存储课程的学习后，自主研习授课老师介绍的FAST 2017会议论文，了解现在国际上对于海量存储课题的研究方向和尖端技术。撰写这篇课程结业论文，主要是研读了FAST 2017的一篇会议论文，并且结合与这篇论文相关知识的三四篇论文以及一些网上能查阅到的相关知识完成了这样一篇结课论文。

本人签名： 王俊秋 日期：2017年12月

摘 要

本文主要分析了现代分布式存储系统在文件系统故障如数据损坏和读写错误的情况下的行为。我们描述了八种流行的分布式存储系统，揭示了许多与文件系统容错相关的问题。我们发现，现代分布式系统并不总是使用冗余来从文件系统故障中恢复：单个文件系统故障可能导致灾难性结果，如数据丢失、损坏和不可用。我们还发现上述结果是由于许多系统中常见的文件系统故障处理中的基本问题而引起的。

本文的研究对下一代容错分布式和云存储系统的设计具有重要意义。

关键词：分布式存储；文件系统；故障；冗余；

**ABSTRACT**

This article mainly analyzes the modern distributed storage system in the file system failure such as data corruption and read and write errors in the case of behavior. We describe eight popular distributed storage systems and reveal many of the problems associated with file system fault tolerance. We found that modern distributed systems do not always use redundancy to recover from file system failures: single file system failures can lead to catastrophic results such as data loss, corruption, and unavailable. We also found that the above result is due to the basic problem in file system troubleshooting that is common in many systems.

The research in this paper is of great significance to the design of the next generation of fault-tolerant distributed and cloud storage systems.

**Key Words**: distributed storage; file system; fault; redundancy;

目 录

[1 引言 1](#_Toc500339186)

[1.1 研究背景 1](#_Toc500339187)

[1.2 研究现状 1](#_Toc500339188)

[1.2.1 文件系统故障 1](#_Toc500339189)

[1.2.2 故障原因分析 2](#_Toc500339190)

[1.2.3 错误处理的必要性 2](#_Toc500339191)

[2 相关工作 4](#_Toc500339192)

[2.1 八种文件系统 4](#_Toc500339193)

[2.2 文件系统故障 4](#_Toc500339194)

[2.3 分布式存储系统 5](#_Toc500339195)

[3 分布式存储响应模式 7](#_Toc500339196)

[3.1 构建测试系统 7](#_Toc500339197)

[3.1.1 故障模型 7](#_Toc500339198)

[3.1.2 故障注入 7](#_Toc500339199)

[3.1.3 行为推断 8](#_Toc500339200)

[3.1.4 可靠性测试 9](#_Toc500339201)

[3.1.5 错误研究 9](#_Toc500339202)

[3.2 系统行为分析 9](#_Toc500339203)

[3.3 观察和结果 9](#_Toc500339204)

[4 系统行为分析 10](#_Toc500339205)

[4.1 Redis 10](#_Toc500339206)

[4.2 ZooKeeper 11](#_Toc500339207)

[4.3 Cassandra 11](#_Toc500339208)

[4.4 Kafka 12](#_Toc500339209)

[4.5 RethinkDB 13](#_Toc500339210)

[4.6 MongoDB 14](#_Toc500339211)

[4.7 LogCabin 15](#_Toc500339212)

[4.8 CockroachDB 15](#_Toc500339213)

[5 跨系统分析 17](#_Toc500339214)

[5.1 观察结果 17](#_Toc500339215)

[5.2 结果总结 21](#_Toc500339216)

[5.3 对文件系统的影响 21](#_Toc500339217)

[5.4 汇总 22](#_Toc500339218)

[6 总结和展望 24](#_Toc500339219)

[6.1 论文总结 24](#_Toc500339220)

[6.2 未来工作展望 25](#_Toc500339221)

[参考文献 26](#_Toc500339222)

[致谢 27](#_Toc500339223)

# 引言

## 研究背景

随着信息技术和互联网技术的发展，许多传统的工业产业逐渐走向数字化，网络化和智能化，在这个过程中，衍生出了多种多样的数据，这些数据呈现爆炸性增长的趋势，我们开始进入大数据时代，在数据体量巨大且应用价值高的情况下，急需一种高效的存储和管理方法，从而对数据的分析提供支持。

而传统的存储管理技术采用关系数据库和本地文件系统结合的存储方式，无法满足用户的需求：一方面，传统的存储方式以单机存储为主，无法为大规模数据提供高效存储和快速计算的支持；另一方面，传统存储方式管理的数据类型单一，关联关系简单，无法适应类型多样且关系复杂的大数据环境，为了对大量数据进行有效的管理，在现代社会使用可靠的、高效的分布式存储管理方法，这种方法既满足海量存储的需要，也能适应大数据环境的特征。

在分布式存储系统的广泛使用过程中，发现了诸多文件系统故障例如数据损坏和读写错误。因此，通过研究对八种比较流行的分布式存储系统进行研究，发现了许多和文件系统容错性相关的问题。研究发现，现代分布式系统始终不能使用冗余从文件系统故障中恢复：单个文件系统故障有可能导致灾难性的结果：数据丢失、损坏和不可用。通过对这一问题的研究，有助于研发和改进下一代容错分布式和云存储系统。

## 研究现状

现代分布式存储系统以复制（建立副本）的方式存储数据以提高可靠性。每个副本都在硬件上的本地文件系统上工作，用以存储和管理关键用户数据。

### 文件系统故障

在大多数情况下，现代分布式存储系统通过复制（制作副本）可以屏蔽诸如系统崩溃，电源故障以及磁盘或网络故障等故障。然而，存储设备（如磁盘和闪存驱动器）呈现出更复杂的故障模型，例如：其中某些数据块可能无法访问（读取和写入错误）；或者更糟糕的，数据可能被无声地被破坏。这些复杂的故障被称为部分存储故障。

文件系统如ext3，NTFS和ZFS在处理部分存储故障时，在某些情况下，文件系统只是简单地将故障传播到应用程序;例如，如果底层设备块损坏，ext4将原样返回损坏的数据给应用程序。在其他情况下，文件系统会在错误传递给应用程序之前对错误做出反应并将其转换为另一个错误，例如，btrfs将底层的块损坏转换为读取错误。无论哪种情况，我们都将文件系统引发的故障称为文件系统故障。

### 故障原因分析

通过对故障发生情况的进一步分析，可以得出分布式存储系统常见的文件系统出错的原因在于以下几点。

1. 不同的分布式管理系统采用不同的数据完整性策略来防止文件系统故障。一些系统使用校验和，而另一些则使用堆栈中的较低层来检测和处理损坏。
2. 故障在本地通常不被发现，有时候，这样的局部未被发现的缺陷通过静默传播，会直接导致有害的全球效应。
3. 大多数情况下，即使系统可靠地检测到故障，它们也只是会发生系统崩溃而不是使用冗余类恢复故障。
4. 冗余未充分利用，虽然分布式存储系统跨多个节点复制数据和功能，但单个节点上的单个文件系统故障可能会导致有害的集群范围影响，然而，许多分布式存储系统并不总是使用冗余作为恢复的来源。
5. 数据崩溃和数据损坏处理纠缠在一起。系统通常会混淆从数据崩溃中恢复和从数据损坏中恢复这两种情况，意外地调用错误的恢复子系统来处理错误，将最终导致数据丢失等不良后果。
6. 本地故障处理行为和全局分布式协议（如读取修复，领导者选举和重新同步）有时会以不安全的方式进行交互，从而导致数据损坏蔓延到完整的副本或数据丢失。

### 错误处理的必要性

从上面的研究中，不难发现，在大多数现代分布式存储系统中，单个文件系统故障会导致灾难性后果。尽管在分布式存储中普遍存在校验和，冗余和其他弹性方法，但单个文件系统故障可能会导致数据丢失、损坏、不可用，并且在某些情况下将损坏扩散到其他完整的副本，导致其他副本也不可用，或者在查询过程中返回意外的错误。例如，日志初始化期间的单个写入错误可能会导致Zoo Keeper中写入不可用。同样，Redis和Cassandra中的一个节点中的损坏数据可以传播到其他完整的副本。在Kafka和RethinkDB中，一个节点上的损坏会导致用户可见的数据丢失。由于分布式存储系统固有地存储冗余数据副本，而且我们一次只能注入一个错误，所以这些行为是不可取的。

因此，基于云服务的现代分布式存储系统响应文件系统故障的行为是至关重要的，然而，现代研究技术中对于现代分布式存储系统如何对文件系统故障作出反应这一命题知之甚少。

普遍的和广泛的期望是，更高层（即跨越复制品）的冗余能够从本地文件系统故障中恢复。例如，分布式存储系统的一个节点中的不可访问的数据块，在理想情况下不会导致用户可见的数据丢失，因为相同的数据被冗余地存储在许多节点上。

基于这种期望，我们对现代分布式存储系统出现在本地文件系统故障时的表现进行了研究，并且对现代分布式系统能否使用冗余从单个文件系统故障中恢复进行了测试，这些内容将在后面的章节进行详细的介绍。

# 相关工作

## 八种文件系统

为研究现代分布式存储系统如何应对本地文件系统故障，该文献[1]的作者们构建了一个名为Cords的故障注入框架，其中包括以下关键部分：errfs，系统地注入文件系统故障的用户级FUSE文件系统；errbench，一套系统特定的工作负载，可以驱动系统与本地存储进行交互。对于每个注入的故障，线自动观察结果系统行为。

研究人员研究了八个广泛应用的分布式文件系统：

Redis：内存中键值数据存储；

ZooKeeper：快速、高可用、容错、分布式的协调服务系统；

Cassandra：面向列的分布式数据存储；

Kafka：分布式发布-订阅消息系统；

RethinkDB：存储Jason的文档型数据库；

MongoDB：基于分布式文件存储的介于关系型和非关系型之间的数据库；

LogCabin：基于Raft共识算法构建的小型存储系统；

CockroachDB：可实现跨数据中心同步的可伸缩开源数据库。

## 文件系统故障

文件系统下的存储堆栈中的图层由许多复杂的硬件和软件组件构成。存储堆栈的底部是媒体介质（磁盘或闪存设备），媒体介质上层的固件起到控制媒体的作用。固件的命令由设备驱动程序提交。文件系统可能会遇到由各种底层故障导致的错误，包括媒体介质错误，磁盘中的机械和电器问题，固件的问题和总线控制器中的问题。有时候，由于除了操作系统和设备驱动程序的其他部分的软件错误，甚至文件系统自身的错误也会导致文件系统损坏。

鉴于这些原因，文件系统出现两个问题：

1. 块错误：某些块不可访问（也称为潜在扇区错误）
2. 块损坏：其中某些块不包含预期数据。

磁盘返回一个明确的错误时，文件系统可以检测到被访问的块的某些问题。这些学者在32个月内对超过100万个磁盘驱动器的研究结果表明，8.5％的近线磁盘和1.9％的企业级磁盘产生了一个或多个潜在扇区错误。在他们最近的研究中显示，在基于闪存的SSD中出现类似的错误。同样，最近一项关于闪存可靠性的研究[2]表明，在数百万闪存器件中，高达63％和2.5％的闪存器件分别经历至少一次读写错误。由于驱动器固件中的错误导致写入错误或丢失，文件系统可能会收到损坏的数据，或者磁盘内ECC未检测到位错误。块的破坏是隐秘的，因为块被磁盘本身检测不到的方式损坏。在很多情况下，文件系统会不知不觉地访问这些损坏的块，并将其无声地返回给应用程序。 Bairavasundaram等人在对超过41个月的153万个磁盘驱动器的研究中发现，超过40万个块有校验和不匹配[3]，事实证据表明存储错误和数据损坏很是常见。鉴于存储损坏的频率和错误，文件系统遇到这种错误的概率是不可忽略的。

在许多情况下，当文件系统遇到来自其底层的故障时，它只是将它原样传递到应用程序。例如，Linux默认的文件系统ext4，当底层的块不可访问或损坏时，只是简单地将错误或损坏的数据返回给应用程序。在另一些情况下，文件系统可能会将潜在的故障转换为另一种故障：例如，btrfs和ZFS将底层损坏转换为错误 - 当访问底层损坏的磁盘块时，应用程序将收到错误而不是损坏的数据。无论哪种情况，我们都将文件系统引发的这些故障称为文件系统故障。

## 分布式存储系统

考虑到本地文件系统可能会返回损坏的数据或错误，数据完整性和适当的错误处理的责任落在应用程序上，因为它们关心安全存储和管理关键用户数据。大多数单机应用程序（如独立数据库和非复制键值存储系统）完全依赖本地文件系统来可靠地存储用户数据;他们很少有办法从本地文件系统故障中恢复。例如，在读取时，如果本地文件系统返回错误或损坏的数据，则应用程序无法恢复该段数据。他们最好的行动方式是可靠地检测到这些故障，并向用户发送适当的错误信息。

与单机应用程序类似，现代分布式存储系统也依靠本地文件系统来安全地管理关键用户数据。但是，与单机应用程序不同，分布式存储系统固有地以复制的方式存储数据。精心设计的分布式存储系统可能使用冗余来从错误和损坏中恢复，而不管本地文件系统提供的支持。在理想情况下，即使一个副本损坏，整个分布式存储系统也不应受到影响，因为其他副本上存在相同数据的其他完整副本。类似地，考虑到功能（应用程序代码）也跨多个节点复制，一个节点中的错误不应该影响系统的全局可用性。Ghemawat等人也描述了在Google文件系统中需要这种基于端到端校验和的检测和恢复，因为底层便宜的IDE磁盘通常会破坏块服务器中的数据[4]。同样，Google在构建大型互联网服务方面的经验教训也强调了高层软件应该如何提供可靠性。考虑到分布式系统的端到端数据完整性和错误处理的可能性，研究者研究了现代分布式存储系统能否以及如何使用端到端技术从本地文件系统故障中恢复。

# 分布式存储响应模式

## 构建测试系统

在Arpaci-Dusseau R H等人的研究中[1]，针对文件系统可能会将错误或者损坏的数据返回到它上面运行的应用程序这一情况进行了测试研究，首先他们提出了一个故障模型（定义了文件系统可能遇到的系统故障条件），通过相应的方法来注入我们的模型定义的故障，并观察注入故障的影响。

### 故障模型

构建故障模型的目标是，注入当前和未来文件系统中代表故障条件的故障，并将分布式系统引入很少测试的错误情况。

在Arpaci-Dusseau R H等人的研究中[1]，他们的故障模型有两个重要特征。首先，考虑在单个节点中一次只向单个文件系统块注入单一故障，他们的研究专注于在单个节点注入单个故障的最基本的情况，故障模型旨在为应用程序提供最大的恢复余地。另一方面，相关的错误可能排除这样的余地。例如，如果包含重要应用程序级数据结构的两个或多个文件系统块被损坏（可能存在相关的错误模型），那么应用程序挽救其状态的可能性就会降低。

其次，该故障模型只将错误注入应用程序级别的磁盘结构，而不是文件系统元数据。文件系统可能能够保护自己的（元）数据；但是，如果用户数据损坏或无法访问，则应用程序将收到损坏的块或者可能收到错误（如果文件系统具有用户数据的校验和）。因此，应用程序如何处理这种情况是至关重要的。

|  |  |  |  |
| --- | --- | --- | --- |
| **错误类型** | | **操作** | **原因示例** |
| 块区  损坏 | 被零 | 读 | 在ext和XFS中丢失和错误的写入 |
| 破损 | 读 | 在ext和XFS中丢失和错误的写入 |
| 块区错误 | I/O 错误  (EIO) | 读 | 所有文件系统中的潜在扇区错误，ZFS，btrfs中的磁盘损坏 |
| 写 | 文件系统挂载为只读，磁盘损坏在btrfs中 |
| 空间错误 | 写 | 磁盘已满，在所有文件系统中超出配额 |
| 位损坏 | | 读 | 在ext和XFS中没有被设备内ECC检测到的位错误 |

### 故障注入

errfs可以注入两种类型的块损坏：被零或破损。对于块损坏，errfs会执行读取操作，并在返回应用程序之前更改标记为损坏的块的内容， errfs可以注入三种类型的块错误：EIO读取（读取错误），EIO写入（写入错误）或ENOSPC和EDQUOT写入，需要额外的空间（空间错误）。为了模拟错误，errfs不执行操作，而是简单地返回一个适当的错误代码。对于位损坏，errfs需要特定于应用程序的信息，包括块中的各个字段以及它们的偏移量和长度。要注入一点损坏，在返回数据之前，errfs会在标记为损坏的字段中翻转一下。

### 行为推断

对于注入单个故障的每个工作负载运行，研究学者对于注入单个故障的每个工作负载运行，观察系统的行为。我们系统特定的行为推理脚本收集系统

来自系统日志文件和客户端可见输出（例如服务器状态，返回码，错误（stderr）和输出消息（stdout））的行为。一旦已知注入故障的系统行为，我们将观察到的行为与预期的行为进行比较。以下测试系统的预期行为：

•承诺的数据不应该丢失

•查询不应该默默地返回损坏的数据

•群集应该可用于读取和写入

•重试后查询不应该失败。

学者们相信这些期望是合理的，因为在分布式系统的单个节点中的单个故障理想情况下不会导致任何不良行为。如果观察到的行为与预期不符，那么会将特定运行（工作负载和注入的错误的组合）标记为错误，分析相关的应用程序代码，联系开发人员和文件错误。局部行为和全球效应。在分布式系统中，多个节点使用本地文件系统来存储用户数据。当一个节点注入故障时，我们需要观察两件事：注入故障的节点的局部行为和故障的全局效应。

在大多数情况下，节点在本地对注入的故障作出反应。，由于注入故障，节点可能会崩溃或部分崩溃（只有少数线程的进程被终止）。在某些情况下，节点可以通过重试任何失败的操作或通过使用内部冗余数据（复制副本中的文件间相同数据是冗余的情况）来解决问题。或者，节点可以检测并忽略损坏的数据或只记录错误消息。最后，节点可能甚至不能检测或采取任何措施来防止故障。

故障的全局效应是外部可见的结果。全局效应取决于分布式协议（如领导者选举，共识，恢复，修复）如何响应故障节点的本地行为。例如，即使一个节点可以在本地忽略损坏的数据并丢失，全局恢复协议也可以解决这个问题，从而导致一个正确的外部可观察的行为。有时，由于分布式协议的反应，全局损坏，数据丢失，读取不可用性，写入不可用性，不可用性或查询失败可能是可能的。当一个节点作为局部反应简单地崩溃时，系统以减少的冗余运行，直到手动干预。

给定工作负荷和故障的这些本地行为和全局效应可能会有所不同

在由注入故障的节点扮演的角色（领导者或追随者）。为了简单，我们统一使用术语领导者和追随者，而不是主从。

我们在这里注意到，我们的工作负载套件和模型并不完整。首先，我们的套件只包含在内

简单的读写工作负载，而更复杂的工作负载可能会产生更多的见解。其次，该模型不会注入所有可能的文件系统错误;相反，它只注入了诸如腐败，读取，写入和空间错误等一系列错误。但是，即使我们简单的工作负载和故障模型也会导致系统陷入极端情况，导致有趣的行为。这个框架可以扩展到包含更复杂的错误，并且工作负载套件可以增加更复杂的工作负载；故而把它作为未来工作的一个途径。

### 可靠性测试

### 错误研究

## 系统行为分析

## 观察和结果

# 系统行为分析

在进行了如上基本介绍后，研究学者对八种文件系统进行了详细的行为分析，针对每个系统的描述注入块错误和块错误被注入不同磁盘结构时候的行为，对于每个系统，我们还会显示系统中的磁盘文件和逻辑数据结构的格式。对于一些系统，还另外介绍了注入位错误时的行为。

## Redis

Redis是一个流行的数据结构存储，用作数据库，缓存和消息代理。 Redis使用异步主备份复制;因此，有一个数据丢失的窗口。但是，只有当至少有N个滞后少于M秒的跟随者连接到领导者时，Redis才能被配置为接受写入。当前领导失败时，Redis不会自动选举领导。

Redis的磁盘结构：Redis使用简单的附加日志文件（aof）来存储修改数据库状态的命令或操作序列。附录文件不是校验和。在记录一系列操作之前，记录数据库标识符;此标识符指定稍后重播附件文件时要应用操作的数据库。从aof获取定期快照以创建redis数据库文件（rdb）。启动期间，追随者重新同步领导的rdb文件。整个rdb文件受到单个校验和的保护。

行为分析：将块损坏和块错误引入不同的磁盘结构时，当appendonly文件中的元数据结构发生损坏或访问错误时，该节点只会崩溃。如果领导者崩溃，那么集群变得不可用，如果追随者崩溃，则集群以减少的冗余（两个工作负载的第一种全局效应）运行。 Redis不使用附加文件中的用户数据的校验和；因此，它不检测腐败（这两个工作负载的第二种本地行为）。如果领导者被破坏，那么它会导致全局用户可见的损坏，如果关注者被破坏，那么就没有有害的全局效应（读取工作负载的第二种全局效应）。重新同步协议会将损坏的用户数据从领导者传播到关注者，导致全球用户可见的损坏。相比之下，附加文件用户数据中的错误导致崩溃（两个工作负载的第二种本地行为）;领导者和追随者的崩溃分别导致集群不可用性和冗余度降低（这两个工作负载的第二种全局效应）。

redis\_database第一个块中的问题通过再次从appendonly文件中的数据重试和创建redis\_数据库文件来解决。当跟随者的dis\_database文件被损坏时，它会崩溃，从而导致冗余度降低。由于领导者在重新同步过程中发送了rdb文件，因此同样的损坏会导致追随者崩溃，而这些崩溃最终使群集不可用于写入。

## ZooKeeper

ZooKeeper是一种用于存储配置信息，命名和分布式同步的流行服务。 ZooKeeper提供了分层的名称空间（数据树），并支持在数据树中创建和删除节点等操作。 ZooKeeper实现状态机复制，并使用原子广播协议（ZAB）来维护系统中所有节点的标识状态。只要大多数节点正常工作，系统仍然可用。它通过在日志中持久化操作并持久化数据树的快照来提供持久性。

磁盘结构： ZooKeeper使用日志文件来附加用户数据。该日志包含一个日志头（魔术，版本等），后面跟着一系列事务。一个事务由一个事务头部组成，并受到校验和的保护。事务头包含纪元，会话ID等等。 ZooKeeper维护两个重要的元数据结构：epoch（接受的和当前的epoch）和myid（节点标识符）。通过首先写入epoch\_tmp然后将其重命名为epoch来更新时期。

行为分析：ZooKeeper可以使用校验和检测日志的transaction\_head和trans- action\_body中的损坏，但是只是通过崩溃（作出反应。当epoch和myid被破坏或无法读取时，节点会简单地崩溃（两个工作负载的第一行和第三行）。同样，它在大多数错误情况下崩溃，导致减少冗余。在所有的崩溃情况下，ZooKeeper都可以可靠地选出新的领导者，从而确保可用性。当ZooKeeper的尾部被损坏时，它会在本地忽略一个事务。领导选举协议阻止该节点成为领导者。最终，受损节点通过联系领导者来修复其日志，从而导致正确的行为。

不幸的是，ZooKeeper不能从写错误恢复到事务头和日志尾部。图3（b）描述了这种情况。在日志初始化期间发生写入错误时，错误处理代码尝试正常关闭节点，但仅杀死事务处理线程;仲裁线程保持活动（部分崩溃）。因此，其他节点认为领导者是健康的，不会选出新的领导者。但是，由于领导者已经部分崩溃，所以不能提出任何交易，导致无限期的写入不可用。请注意，此方案不会对读取工作负载造成有害的全局影响，因为读取可由任何ZooKeeper节点本地提供服务，而不需要领导者提出新的事务。

## Cassandra

Cassandra是Dynamo-like NoSQL商店。与我们研究的其他系统不同，Cassandra是一个分散的系统;它没有领导者和追随者。系统将所有的数据均匀地分布在一个节点簇中，形成一个环。 Cassandra将数据复制到由复制因子指定的多个节点。它还支持不同的读写一致性级别。在Cassandra中，行被组织到表中，行根据主键的散列在集群中的节点之间分配。 Cassandra还提供了类似于SQL的查询语言（CQL）。

磁盘结构：在Cassandra中，本地存储引擎是将数据存储在sstable中的日志结构合并（LSM）树的一种变体。为每个键空间维护一个单独的sstable;我们将用户创建的键空间的sstables称为tablesst。sstable中的磁盘文件。每个sstable由Bloom过滤器（tablesst\_filter）组成;该过滤器提供了一种快速的方式来确定给定的密钥是否存在。如果在过滤器中找到键，则访问表摘要（tablesst\_summary）和表索引（tablesst\_index）。 tablesst\_index包含数据文件中数据项的偏移量（tablesst\_data）。 tablesst\_data包含表中的所有行。

行为分析。 Cassandra仅对用户数据进行校验和验证，只是为了启用压缩的副作用。因此，我们在Cassandra中进行了两个实验，一个禁用了用户表的压缩，另一个启用了压缩。

关闭压缩时，在tablesst\_data中的用户数据上检测不到损坏。在读取查询中，协调器节点收集并比较来自R副本的数据的摘要（散列）。如果摘要不匹配，则使用最新的时间戳获取策略解决值中的冲突。如果时间戳之间有联系，那么选择词汇最大值并将其安装在其他副本上。在R = 3时，如果损坏值在词汇上大于原始值，则将损坏的值返回给用户，并将损坏传播到其他完整副本（全局效果的第三行对于R = 3读取工作负载时，垃圾损坏）。另一方面，如果损坏的值在词汇上较小，则修复损坏的节点（当损坏为零时R = 3读取工作负载的第三行全局效应）。读取R = 1的受损节点总是返回损坏的数据。 tablesst\_index中的故障导致查询失败（读取工作负载的第五行全局效应）。在大多数情况下，即使在R = 3的情况下运行，在R = 1配置中观察到的用户可见问题也不会被修复。

当压缩启用时，Cassandra为每个压缩块维护一个校验和。当压缩数据中的值被破坏时，由于存储的校验和与解压缩数据的计算校验和不匹配，解压缩失败。因此，检测到tab-lesst\_data.userdata损坏，并导致表扫描和点查询失败（图5（c）中读取工作负载的第一行）;对不在此损坏块中的数据的点查询不受影响。即使在R = 3运行查询并且导致查询失败的情况下，损坏也不会自动修复。由于我们不改变系统模式sstables的压缩特性，因此我们不对这些结构重复这个实验。

## Kafka

Kafka是一个分布式持久消息队列，客户可以在其中发布和订阅消息。 卡夫卡由一个领导者和一组追随者组成。 系统将消息流存储在称为主题的类别中; 一个话题可以有零个或多个消费者订阅。 主题中的每个消息由一个键，一个值和一个时间戳组成。

磁盘结构： 传入消息附加到日志文件。 每个消息都进行校验和，并与一个消息ID和一个可选密钥相关联。 Kafka维护一个索引文件，将消息索引到日志中的字节偏移量。重要的元数据结构（如节点标识符）保存在称为元的文件中。 replication\_checkpoint和recovery\_checkpoint结构分别指示到目前为止向追随者复制了多少个消息以及到目前为止已将多少消息刷新到磁盘。 首先写入临时文件（repl\_checkpoint\_tmp），然后将其重命名为最终文件，即可更新复制偏移量。

行为分析。在读写错误上，卡夫卡大多数是崩溃的。当领导者的日志条目被破坏时，它会在本地忽略该条目以及日志中的所有后续条目，从而导致数据丢失。领导者然后指示跟随者也这样做。在从领导接收到这个指令后，追随者检查领导的偏移量是否大于他们的检查点偏移量。如果这种情况不成立，那么追随者就会发出一个致命的断言，简直就是崩溃。关注者崩溃后，群集将变得不可用（写入工作负载的全局效应的第一行和第二行）。

我们进行了另一个实验，其中领导者的数据崩坏发生在后续检查点之前，消息偏移到他们的recovery-offset-checkpoint文件。如果追随者没有检查点（已经被领导者截断），则他们按照领导者的指示截断条目，导致无声的永久数据丢失。在这种情况下，追随者继续运行，没有崩溃。

索引中的损坏使用内部冗余是固定的。领导者的replication\_checkpoint中的错误会导致数据丢失，因为领导者无法记录关注者的复制偏移量。当领导者分别不能读取或写入replication\_checkpoint和replication\_checkpoint\_tmp时，Kafka变得不可用。

位腐败。在消息日志的任何字段中有点翻转时，节点会截断损坏的消息和所有后续消息。如果这个腐败发生在追随者身上，那么领导者会把截断的消息提供给追随者。领导者的同一个单位翻转导致无声的数据丢失。复制偏移中的位翻转有时会导致数据丢失。

## RethinkDB

RethinkDB是一个分布式数据库，适用于将查询结果推送到实时Web应用程序。 RethinkDB使用Raft共识协议来维护集群元数据。它依靠底层存储栈来处理数据完整性，而不会维护用户数据的检查。

磁盘结构：RethinkDB使用持久的B树来存储所有数据。事务存储在树的叶节点上;一个事务由三个块组成：db.transaction\_head，db.transaction\_body和db.transaction\_tail。 B树中的元块指向构成数据库的当前版本和先前版本的数据块。在更新时，首先仔细写入新的数据块并刷新到磁盘。然后，具有校验和的元区块被更新以指向新的数据块，从而启用原子更新。 B树数据块和元块是单个数据库文件的一部分。

行为分析。对于数据库头和内部B树节点的任何错误，RethinkDB只是崩溃。如果领导崩溃，则自动选出新领导。 RethinkDB依靠文件系统来确保数据块的完整性;因此，它没有检测到交易主体和尾部中的腐败。当领导者的这些块被损坏时，RethinkDB默默地返回损坏的数据。

当指向事务的事务头或元块在领导者（领导者的全局效应的最后一行）被破坏时，数据是如何丢失的。即使追随者有相同数据的完整拷贝，即使我们用大多数选项执行读取，领导者也不能修复其损坏或丢失的数据。当下面的人被破坏时，他们不是通过联系领导来解决的。虽然这不会导致用户可见的直接的腐败或损失（因为领导者的数据是最终返回的数据），但是当腐败的追随者成为未来的领导者时，这种做法就是如此。

## MongoDB

MongoDB是一个流行的面向文档的数据库，使用类似JSON的文档。 MongoDB使用副本集提供高可用性。它使用主备份复制，每个复制副本由一个主副本和一个副副本组成。所有写入和读取操作都是在默认情况下完成的。主节点失败时，副本集自动选择新的主节点。 MongoDB支持多个存储引擎。对于我们的实验，我们使用WiredTiger作为存储引擎。

磁盘结构：插入或更新项目时，会将其添加到日志中，并更新内存数据库。如果写入操作将选项j指定为true，则WiredTiger将强制日志的fsync。 WiredTiger使用多版本并发控制，并定期将内存数据的一致视图设置为集合文件和索引文件。主WiredTiger文件包含有关最新检查点文件的信息。存储引擎信息存储在storage\_bson文件中。

行为分析。MongoDB在大多数错误上崩溃，导致冗余度降低。如果当前的领导者崩溃，新的领导者会自动选出。 MongoDB使用所有文件的校验和;任何文件的任何块中的损坏都会导致校验和不匹配和最终崩溃，从而导致冗余度降低。

上述情况的一个例外是，除了日志标题之外的其他块被损坏。在这种情况下，MongoDB检测并忽略损坏的块；然后，被破坏的节点会截断其损坏的日志，然后成为追随者，最后通过联系领导者来修复日志。在追加日志时存在空间错误的情况下，查询失败。

## LogCabin

LogCabin提供了复制和一致的数据存储，作为其他分布式系统的地方，用于维护其核心元数据（如配置设置）。 Log-Cabin实现状态机复制并使用Raft一致性协议。在Log-Cabin中，默认情况下，所有的读写都通过领导。

磁盘结构：LogCabin实现一个分段的日志来存储数据;每个段都是文件系统上的一个文件。有两种类型的段文件:开放段和封闭段。打开的段是数据附加到的当前文件。当开放的部分被充分利用时，它被关闭并且打开一个新的部分。两个元数据文件（metadata1和metadata2）维护Raft元数据和有关日志的信息。元数据文件交替更新;当元数据文件部分更新或损坏时，LogCabin将使用包含稍旧元数据的其他元数据文件。

行为分析。LogCabin在所有读取，写入和空间错误上崩溃。类似地，如果打开的段文件头或封闭段中的块损坏，则LogCabin简单地崩溃。 LogCabin使用校验和识别打开段中的任何其他块中的损坏，并通过简单地放弃和忽略损坏的条目以及该段中的所有后续条目来作出反应。如果日志指针文件损坏，则LogCabin忽略该指针文件，并使用另一个指针文件。

在上述两种情况下，领导选举协议确保了受损节点不成为领导；损坏的节点成为追随者，并通过联系新的领导者来修复其日志。这确保在任何故障情况下，LogCabin不会全局损坏或丢失用户数据。

## CockroachDB

CockroachDB是一个分布式SQL数据库，建立在事务性和高度一致的键值存储之上。它可以承受磁盘，机器，机架和数据中心故障。蟑螂数据库使用筏，只要大部分副本仍然可用，系统可以继续取得进展。它支持强一致的ACID事务，并且还提供了一个类似于SQL的查询语言。

磁盘结构： CockroachDB使用RocksDB的调优版本作为本地存储;存储引擎是将传入数据附加到持久日志的LSM树;然后定期压缩内存中的数据以创建sst文件。清单文件列出构成LSM树中特定级别的一组sst文件，并且当前文件指向最新清单。

行为分析。大多数情况下，CockroachDB只是在任何数据结构上的损坏和错误上崩溃，导致冗余度降低。领导者日志文件第一个块中的故障导致整个群集不可用，因为一些追随者在领导者崩溃之后也会崩溃。其他一些日志元数据块中的损坏和错误可能会导致数据丢失，CockroachDB在默默返回零行。 sst文件中的损坏会导致查询失败，并显示错误消息（如表不存在或db不存在）。总的来说，我们发现CockroachDB在故障处理方面存在很多问题。然而，由于蟑螂数据库仍处于积极的发展阶段，未来可靠性可能会提高。

# 跨系统分析

## 观察结果

我们现在提出一组关于所有八个系统的数据完整性和错误处理的观察结果。

（1）系统采用多种数据完整性策略。现代分布式存储系统使用一系列技术来检测并从腐败中恢复。在频谱的一端，有一些系统试图通过使用校验和（例如ZooKeeper，MongoDB，CockroachDB）来防止存储堆栈中的数据损坏，而另一端则包括完全信任和依赖存储栈中的较低层来处理数据完整性问题（例如，RethinkDB和Redis）。尽管采用了大量的数据完整性策略，但是所有系统都表现出不希望的行为

有时，看起来不相关的配置设置会影响数据的完整性。例如，在Cas- sandra中，校验和仅被验证为启用压缩的副作用。由于这种行为，关闭压缩时无法检测或修复损坏，导致用户可见的沉默损坏。

我们也发现一些系统使用不合适的校验和算法。例如，ZooKeeper使用Adler32，它仅适用于解压缩后的错误检测，并且可能会碰撞非常短的字符串[48]。在我们的实验中，我们能够注入造成校验和碰撞的腐败，使ZooKeeper能够处理损坏的数据。我们认为，像ZooKeeper这样的元数据存储可靠地存储配置设置等小实体并不是不合理的。一般来说，我们认为需要更多的关注来了解可能的校验和选择的稳健性。

（2）故障往往未被发现。我们发现故障通常在本地不被发现。有时候，这会导致直接有害的全球效应。例如，在雷迪斯，领导人追加档案中的腐败现象未被发现，导致全球沉默的腐败。另外，领导者的rdb中的腐败也没有被发现，当被发送给追随者时，导致他们崩溃，导致无法使用。同样，在Cassandra中，tablesst\_data的损坏未被检测到，这导致将损坏的数据返回给用户，并有时将其传播到完整的副本。同样，RethinkDB也不会在领导者的交易头部检测到腐败，从而导致全球用户可见的数据丢失。同样，交易机构的腐败现象也未被发现，导致全球性的沉默腐败。追随者也没有发现同样的错误;如果一个腐败的追随者将来成为领导者，那么全球数据丢失或腐败是可能的。

尽管有些系统有目的地检测到故障并作出反应，但有些系统只是作为副作用对故障做出反应。例如，ZooKeeper，MongoDB和LogCabin仔细检测和反应腐败。另一方面，Redis，Kafka和RethinkDB有时只是作为失败的反序列化的副作用而对腐败作出反应。

(3). 崩溃是最常见的反应。我们观察到，碰撞是对断层最常见的本地反应。当系统检测到错误或遇到错误时，它们就会崩溃，从而导致冗余度的降低（从行为分析中图形的局部行为中大量崩溃符号可以看出）。虽然单个节点的崩溃不会立即影响群集的可用性，但总的不可用性即将发生，因为其他节点也可能随后失败。另外，即使一个节点崩溃，需要写入或读取所有副本的工作负载也不会成功。在崩溃之后，如果故障是粘滞的，简单地重新启动并不会起作用;该节点将重复崩溃，直到手动干预修复了潜在的问题。我们也观察到节点比错误更容易崩溃。

我们也观察到，失败的操作很少重新尝试。尽管在使用重试的几种情况下重试有帮助，但我们观察到，有时无限期地重试操作可能导致更多的问题。例如，当ZooKeeper由于空间错误而无法写入新纪元信息（至epoch\_tmp）时，它会删除并创建一个新文件，保持旧文件描述符处于打开状态。由于ZooKeeper盲目地重试这个序列，并且由于空间错误是粘性的，节点很快就会耗尽描述符和崩溃，从而降低可用性。

(4). 冗余未充分利用：单一故障可能造成灾难性的集群范围影响。与广泛的分布式系统冗余可以帮助从单一故障中恢复的预期相反，我们观察到，即使是单个错误或损坏也会导致不利的集群范围的问题，例如总体不可用性，无声的损坏以及过多的数据丢失或不可访问性。在许多情况下，几乎所有的系统都不使用冗余作为恢复的来源;他们错过了使用其他完整副本进行恢复的机会。请注意，我们在研究中发现的所有错误和不良行为都是由于一次只在单个节点中注入单个错误。鉴于数据和功能被复制，理想情况下，不应出现任何不良行为。

一些系统（MongoDB和LogCabin）通过利用其他副本从一些（不是全部）数据库中自动恢复。这种恢复涉及本地和分布式恢复行动之间的协同作用。具体而言，在遇到损坏的条目时，这些系统忽略错误数据（本地恢复策略）。然后，领导者选举算法确保数据项已被损坏并因此被忽略的节点不成为领导者（全局恢复策略）。结果，损坏的节点最终通过从当前领导者获取损坏的数据来恢复损坏的数据。在很多情况下，即使这些系统也不能通过冗余来自动恢复。例如，LogCabin和MongoDB分别在关闭的段或集合被损坏时就会崩溃。

我们还发现，只有一小部分数据出错时，会影响过多的数据。表3显示了当一小部分数据发生故障时受到影响的不同范围。受影响的部分可能会无声无息或无法访问。例如，在Redis中，当仅附件文件中的元数据发生错误时，或者在仅附加文件数据中存在读写错误时，所有用户数据都可能无法访问。同样，在Cassandra中，当小部分数据出错时，整个表格可能无法访问。卡夫卡有时可能会丢失整个日志或从损坏的条目开始直到日志结束的所有条目。当RethinkDB的一小部分损坏或指向该事务的元区块损坏时，RethinkDB将丢失所有作为事务更新的数据。

总之，我们发现冗余没有被有效地用作恢复的来源，并且冗余可以帮助实现功能和数据的一般期望是不现实的。

(5). 崩溃和腐败处理纠缠在一起。我们发现，在许多系统中，检测和恢复代码并不试图区分两个根本不同的问题：崩溃和数据损坏。

存储系统实现崩溃一致的更新协议（即使在更新期间崩溃的情况下，数据应该总是可恢复的，不应该被损坏或丢失）。为此，系统谨慎地命令写入并使用校验和来检测部分更新的数据或由于崩溃而可能发生的损坏。在检测到由于损坏导致的校验和不匹配时，所有系统总是运行崩溃恢复代码（即使损坏实际上不是由于崩溃而是由于存储堆栈中的实际损坏），最终导致不希望的效应，诸如数据丢失。

这个问题的一个典型例子是RethinkDB。 RethinkDB不使用应用程序级校验和来处理损坏。但是，它为其元区块使用校验和来从崩溃中恢复。每当元区块损坏时，RethinkDB检测元区块校验和不匹配并调用其崩溃恢复代码。崩溃恢复代码认为系统在最后一个事务提交时崩溃。因此，它会回滚承诺的和已经确认的交易，从而导致数据丢失。

我们在其他系统中观察到相同的纠缠问题。例如， Kafka中崩溃和腐败处理的纠缠，所有传入的消息都经过校验和后附加到消息日志中，在消息2的追加期间发生崩溃使该消息部分更新的情况，在恢复期间触发校验和不匹配。 Kafka对校验和不匹配的恢复操作是截断校验和不匹配的消息和所有后续消息。在这种情况下，由于部分更新导致失配;因此，截断消息是安全的，因为客户端还没有被确认更新。相反，另一种情况下，第二条消息已被成功提交，客户端已被确认。之后，持有第一条消息的磁盘块被损坏，导致校验和不匹配。但是，恢复代码错误地将这种损坏视为崩溃的信号;因此，它截断并丢失承诺的消息1和2。

同样，ZooKeeper在检测到log.transaction\_tail中的数据损坏后，得出结论：系统在上次事务提交期间崩溃。在检测到这种损坏时，节点忽略事务并下降成为跟随者。最终，恢复协议可以恢复领导者丢失的交易。即使这种情况不会导致任何全球性问题，它也会通知崩溃和腐败处理如何纠缠在一起。

LogCabin尝试使用以下逻辑将崩溃与腐败区分开来：如果封闭段（已满的段）中的块已损坏，则会正确地将该问题标记为损坏，并通过简单的崩溃作出反应。另一方面，如果开放段中的块（仍然用于持久化事务）已损坏，则将其检测为崩溃并调用其通常的崩溃恢复过程。 MongoDB也以类似的方式区分了期刊腐败中的收益腐败。即使是试图辨别腐败事件的系统也并不总是这样做。

纠结检测和恢复崩溃和腐败的重要后果。在损坏（崩溃）恢复期间，某些系统会提取过多的数据来解决问题。例如，当日志条目在LogCabin和MongoDB中损坏时，他们可以通过联系其他副本来修复损坏的日志。不幸的是，他们通过忽略损坏的条目和所有后续条目直到日志结束并随后获取所有被忽略的数据而​​不是仅仅获取损坏的条目来这样做。由于在上次提交事务期间发生了崩溃，因此这些系统会假定损坏的条目是日志中的最后一个条目。同样，卡夫卡的追随者也从领导者获取额外的数据，而不是只有损坏的条目。

（6）本地故障处理和全局协议以不安全的方式进行交互。我们发现局部故障处理行为和常用的分布式协议，如领导选举，读取修复[23]和重新同步以不安全的方式进行交互;这种不安全的交互导致不希望的结果，例如腐败或数据丢失的传播。

例如，在Kafka中，受损节点上的本地故障处理行为与领导选举协议不安全地交互，将节点上的本地数据丢失转化为全局数据丢失。 Kafka维护一段称为“同步复制”（ISR）的元数据。该集合中的任何节点都包含所有提交的数据，并有资格成为领导者。当日志条目在Kafka节点上损坏时，它会忽略日志中的当前和所有后续条目，并截断日志直到最后一个正确的条目。理想情况下，现在这个节点不应该是ISR的一部分，因为它已经丢失了一些提交的日志条目。但是，这个节点不会从ISR中移除，因此仍然可以成为领导者。因此，发给领导的读取请求导致无声的数据丢失。此外，领导还指示追随者截断日志以匹配其日志，从而触发对追随者的断言，导致其崩溃。因此，所有将来的写入都变得不可用。Kafka的本地行为（即截断日志）和全球协议（领导者选举）之间不安全的交互会导致数据丢失和写入不可用。这是行为

与ZooKeeper，MongoDB和LogCabin的领导选举协议相比，截断了日志条目的节点不会成为领导者。

在Dynamo风格的仲裁系统中使用了修复协议来修复任何已经失效的副本数据。在读取请求上，协调器从配置数量的副本中收集正在读取的数据的摘要。如果所有摘要匹配，则简单地返回协调器的本地数据。如果摘要不匹配，则应用内部冲突解决策略，并将解析的值安装在副本上。在Cassandra实施阅读修复，冲突解决办法解决词汇更大的价值;如果注入的破坏字节在词汇上大于原始值，则损坏的值被传播到所有其他完整的副本。

类似地，在Redis中，当数据项在领导者上被破坏时，它不被检测到。随后，再同步协议将损坏的数据从领导者传播给追随者，覆盖追随者上存在的正确版本的数据。

## 结果总结

现在总结以上的行为分析结果。通过分析，冗余在许多系统中不提供容错能力：一个节点上的单个文件系统故障导致灾难性的结果，例如无声的用户可见损坏，不可用性，数据丢失，查询失败或有时甚至是损坏的扩散数据到其他完整的副本。理想情况下，这些灾难性后果不应该出现，因为我们一次只在系统中的单个节点上注入单个文件系统故障。

文件系统故障处理的根本原因，导致了不可取的行为。这些基本问题在所有系统中都很常见。首先，在许多系统中，故障通常在本地不被发现。即使检测到故障，最常见的本地反应是使节点崩溃。所有系统都错过了将冗余用作本地文件系统故障恢复源的机会。我们还发现，崩溃和腐败处理是在许多系统中进行的。最后，局部故障处理行为和全球协议以不安全的方式相互作用，导致灾难性后果。

## 对文件系统的影响

以上所有错误都可能发生在XFS和所有的ext文件系统上，包括缺省的Linux文件系统ext4。鉴于这些文件系统通常用作大型分布式存储部署副本中的本地文件系统，并由开发人员推荐[51,56,64,77]，我们的研究结果对此类实际部署具有重要意义。

文件系统（如btrfs和ZFS）使用用户数据的校验和;在检测到损坏时，它们返回一个错误，而不是让应用程序静静地访问损坏的数据。因此，由于注入块损坏而发生的错误不会在这些文件系统上显示。我们还发现，在这样的文件系统上部署端到端校验和的应用程序出乎意料地导致了不良的交互。具体而言，应用程序由于错误比腐败更频繁地崩溃。在腐败的情况下，一些应用程序（例如LogCabin，ZooKeeper）可以使用校验和和冗余来恢复，从而导致正确的行为;但是，当腐败转化为错误时，这些应用程序崩溃，导致可用性降低。

## 汇总

我们现在考虑为什么分布式存储系统不能容忍单个文件系统故障。在一些系统中（例如RethinkDB和Redis），我们发现主要的原因是他们期望底层的存储栈层可以可靠地存储数据。随着越来越多的部署迁移到可靠的存储硬件，固件和软件可能不成为现实的云，存储系统需要开始采用端到端的完整性策略。

接下来，我们认为分布式系统中的恢复代码没有经过严格测试，导致不良行为。尽管许多系统使用校验和和其他技术，但是执行此类机器的恢复代码却没有经过仔细的测试。我们建议未来的分布式系统需要使用像我们这样的故障注入框架来严格地测试故障恢复代码。

第三，虽然一系列研究工作和企业存储系统提供了解决部分故障的软件指导方针，但这种智慧并没有被过滤到商品化的分布式存储系统。我们的研究结果为分布式系统提供了动力，建立在现有的研究工作基础上，可以容忍除崩溃之外的实际故障。

最后，虽然冗余被有效地用来提供改进的可用性，但是它仍然是作为从文件系统和其他部分故障中恢复的来源而被破坏的。为了有效地使用冗余，首先必须仔细设计磁盘上的数据结构，以便识别损坏或不可用的数据部分。接下来，腐败恢复必须从崩溃恢复中解耦，以仅修复数据的损坏或无法访问的部分。有时，如果完整的副本不可访问，恢复损坏的数据可能是不可能的。在这种情况下，结果应该由设计来定义，而不是作为实现细节。

我们联系了我们发现的行为系统的开发者。 RethinkDB和Redis依靠底层存储层来确保数据的完整性。 RethinkDB开发人员打算将设计更改为将来包含应用程序级别的校验和，并更新文档以反映我们报告的错误，直到解决问题。他们也证实了腐败和事故处理之间的纠缠。

由Cords发现的ZooKeeper中的写入不可用性错误是由真实世界的用户遇到的，并且已经修复。 ZooKeeper开发者提到，在检测腐败方面崩溃并不是一个有意识的设计决定。 LogCabin的开发者也证实了腐败和开放段的崩溃处理之间的纠缠。他们补充说，很难区分部分写作和开放段的腐败。 CockroachDB和Kafka的开发人员也对我们的错误报告做出了回应。

# 总结和展望

## 论文总结

以上研究工作从文件系统故障，故障注入，可靠性测试和错误研究的先前研究的几个机构得到灵感。我们现在描述我们的工作与这些途径之间的关系。

存储堆栈中的腐败和错误。过去的几项研究详细分析了存储错误。具体而言，Bairavasundaram 等和施罗德（Schroeder）等人显示磁盘中潜在扇区错误的发生率。同样，研究也显示了现实世界中数据腐败的普遍性。此外，研究表明，便宜的近线磁盘比企业级设备更容易出错和损坏。由于大规模部署通常倾向于使用廉价的硬件（并且为软件构建可靠性），所以诸如磁盘错误和损坏等问题变得越来越重要。这些先前的研究激励我们研究这些故障在分布式存储系统中的影响。

通用故障注入。我们的工作涉及到向系统注入故障并测试其稳健性的努力。几个努力已经建立了分布式系统的通用故障注入器。这些故障注入框架的大部分旨在注入各种类型的故障，并强调框架向多个平台和系统的可移植性。例如，Han等人（Doctor）[5]，这是一个可以注入处理器，内存和通信故障的综合框架。与通用故障注入器不同的是，它通过特别关注文件系统故障。

文件系统故障注入研究。一些研究表明文件系统如ext3，IBM JFS，ReiserFS和ZFS等是如何对存储和内存故障做出特别反应的。这些研究仔细地注入文件系统下方的磁盘故障，并观察文件系统如何对故障做出反应。在这些研究中使用的注射方法是类型意识的：故障不是随意注射的;相反，它们被注入到文件系统的各种特定的磁盘结构中。与随机故障注入技术相比，类型感知故障注入有助于快速运行多个文件系统代码路径。 Cords中的故障注入方法类似于这种类型感知的注入器。这些研究的结果表明，某些文件系统（如ext3）只是将腐败传播给应用程序，因为它们没有对用户数据使用校验和。结果还显示某些文件系统（如ZFS）使用校验和来处理用户数据，因此将基础损坏转换为读取错误。这些结果意味着希望保持数据完整性的应用程序必须处理这种情况，从而激励我们的研究。

应用故障注入研究。一些研究显示了在文件系统之上运行的应用程序如何对文件系统故障做出反应。例如，Subramanian et al。研究MySql数据库引擎如何对磁盘损坏做出反应。同样，张等人。研究文件同步服务（如Dropbox）如何在存在本地文件系统损坏的情况下作出反应。然而，没有研究检查现代分布式存储系统。我们的工作重点是测试存储故障下分布式系统的行为。我们相信我们的工作是第一次全面检查存储故障在多个分布式存储系统中的影响。

测试分布式系统。几个模型检测器已经成功地发现了分布式系统中的错误。线暴露了模型检查器无法发现的错误。模型检查器通常重新排列网络消息并注入崩溃来查找错误;他们不会注入存储相关的故障。与模型检测器类似，Jepsen等工具对错误网络下的分布式系统进行测试也是对Cords的补充。我们以前的工作研究文件系统崩溃行为如何影响分布式系统。但是，这些故障只发生在崩溃，不像Cords引入的块损坏和错误。我们的目标故障注入框架可以比模型检查器更快地检查大型存储系统，而检查器通常受到状态空间爆炸的影响。

错误研究。最近的一些错误研究已经深入分析了分布式系统中常见的问题。袁等人。在他们的研究中，34％的灾难性失败是由于意料之外的错误状况。我们的结果也表明系统不能很好地处理读写错误;这种糟糕的错误处理在很多情况下会导致有害的全局效应。我们相信，错误研究和错误注入研究是相辅相成的。而错误研究表明通过检查导致在野外遇到错误的事件序列来构建测试用例，像我们这样的错误注入研究集中于注入一种类型的错误，发现新的错误和设计缺陷。

## 未来工作展望

以上研究表明现代分布式存储系统中对文件系统的故障不是根深蒂固无法解决的。这些系统没有配备有效地使用复制副本上的冗余来从本地文件系统故障中恢复; 由于单个本地文件系统故障，用户可见的问题（如数据丢失，损坏和不可用性）可能会出现。 随着分布式存储系统成为存储关键用户数据的主要选择，对所有类型的故障进行仔细设计是非常重要的。 我们的研究是朝着这个方向迈出的一步，我们希望我们的工作将带来更多的工作，建设下一代容错恢复分布式系统。

参考文献

1. Arpaci-Dusseau R H, Arpaci-Dusseau R H, Arpaci-Dusseau R H, et al. Redundancy Does Not Imply Fault Tolerance: Analysis of Distributed Storage Reactions to File-System Faults[J]. Acm Transactions on Storage, 2017, 13(3):20.
2. Apache. Cassandra. Retrieved from <http://cassandra.apache.org/>.
3. Arpacidusseau A C, Arpacidusseau R H, Bairavasundaram L N, et al. Semantically-Smart Disk Systems: Past, Present,and Future. Sigmetrics Performance Evaluation Review (PER), 33(4):29- 35, March 2006. [11] Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau. Fail-Stutter FaultTolerance[J]. 2008.
4. Howard S G, Gobioff H, Leung S. The Google File System[J]. Acm Sigops Operating Systems Review, 2003, 37(5):29-43.
5. Seungjae Han, Kang G. Shin, and Harold A. Rosenberg. 1995. DOCTOR: An integrated software fault injection environment for distributed real-time systems. In Proceedings of the International Computer Performance and De- pendability Symposium (IPDS’95).

致谢

感谢海量存储课程任课老师的辛勤教导，让我学习到大量和存储相关的知识，并在一定程度上提升了我阅读论文的能力和速度，并对国际前沿学者研究的问题有了一定的了解，开阔了眼界，丰富了学识。