武汉大学海量存储课程论文

分布式系统中冗余机制
于底层节点文件系统错误的反应表现

专 业 名 称 ： 计算机技术

班 级 ： 硕士 5 班

学 生 姓 名 ： 王震

学 生 学 号 ： 2017282110207

二〇一七年十二月

摘　要

数据量的爆炸增长给存储系统带来了更高的要求，不仅要求更快的速度，更大的容量更要求了更稳定的存储，更安全的数据质量。但是，在存储系统运行过程中不定期发生的崩溃重启或者读写不同步，脏数据污染仍然会威胁系统存储的数据安全。利用分布式系统进行备份来进行同步恢复的问题在理论上可以解决这些潜在存在的问题，但是实际运行过程中，分布式系统带来的冗余并不能完全的容忍出现的所有存储错误。本文首先通过多个参考文献对分布式系统中可能会出现的存储错误进行详细介绍，然后分析描述对分布式系统对这些错误的反应，抛开不同系统带来的处理方法的差异，尝试按照自己理解寻找新的处理方法。

关键词：分布式系统 冗余 存储错误 数据污染 存储错误的发现与修正

目　录

1 研究背景

近年来随着互联网技术的不断发展，可以产生数据的设备数量不断提高，换句话说也就是世界上产生的数据量正在爆炸式增长，对于服务供应商来说，用户的数据是宝贵的，那么日益增多的数据量也就给他们带来了更多的存储需求。在保证能够以很快的数据存储足够多的数据同时，数据安全也是他们考虑的关键因素之一。

在进行一些市场调查以及文献翻阅之后，现在的存储现状可以总结为：1.服务器每年会崩溃 2 次（2~4%的失败率）。2.每年有 1~5%的硬盘会损坏掉不能继续使用。3.DRAM error 会以比人们想象中更频繁的频率(每年 2%)发生，并且简单的 ECC 策略并足以修正 DRAM error[1]。

而存储系统发生的突发情况就会带来数据污染，在一份测试报告[2]里的数据污染大致可以分为以下三类：

1. 磁盘错误：他们每两个小时向 3000 多个节点写 1 个特殊的 2GB 文件然后再尝试读取来检查错误，这个行为持续了 5 周，期间一共在 100 个节点上发现了 500 多个错误。

   (1)单比特错误：也就是单个比特数据发生错误，这种错误占磁盘错误 10%.

   (2)扇形错误：通常是 512 个比特大小的错误，占磁盘错误的百分之 10%.

   (3)64kb 区域大小的错误：占磁盘错误的 80%。

2. RAID 错误：他们每周都会在 492 个 RAID 系统里运行验证命令，这种验证行为持续了 4 周，结果是虽然误码率很低，但是在读写 2.4Pb 的数据时就已经有 300 多个错误了。

3. 内存错误：虽然在 3 个月内 1300 个节点只有 3 个双字节错误但是根据系统说明不应该产生任何错误，双字节错误并不能被自动修正。

很多文献[1,3,4]都已经肯定了分布式管理对于处理存储系统中不可预料错误以及数据污染的效果，现实中像谷歌，百度，网易腾讯等的服务器也使用了异地备份等方案来保障数据安全以及数据的干净可用，但是分布式管理真的可以处理所有情况吗？

2 背景知识

本章将会解释存储系统里的故障，错误，数据污染等名词，并对典型的分布式系统做一个简单的介绍。

首先是故障（fault）与错误（error）的区别[5]，以及数据污染的简单分类。

故障（fault）：故障是导致错误的一个潜在因素，就像一个固定点或者是一个高能微粒的罢工，故障可以导致错误（active），也可以是静默发生（dormant），不会引起错误。

错误（error）：错误是由一个活跃故障导致的存储里不正确的一部分，就像内存里的一个错误的数值。错误也许可以被发现然后被更高级的方法比如奇偶校验或者错误纠正方法或分布式系统提供的冗余进行修正，也有可能被发现之后修正不能，最坏的情况就是错误静默的发生了，系统没有检测到并且仍然认为该节点存储的已经被污染的数据仍然可信可用。常见的 error 像读到扇区错误数据，写只读磁盘，写磁盘时磁盘已满或者检测不到磁盘等。

数据污染（Corruption Data）：数据污染统称为存储系统里的某个存储节点里的数据不是应有的数值，详细的说就是在存储系统读取数值时读到 0 或垃圾数据。

典型的分布式系统由分布于多个计算机结点上的若干个数据库系统组成,它提供有效的存取手段来操纵这些结点上的子数据库。分布式数据库在使用上可视为一个完整的数据库,而实际上它是分布在地理分散的各个结点上。分布式系统常用的架构是 master-slave 模式，由 master 节点承载主要的数据写入（增删修改）工作，由 slave 节点为用户提供数据读取服务来实现负载平衡。其采用的复制机制可以提供高可用性和容错性。

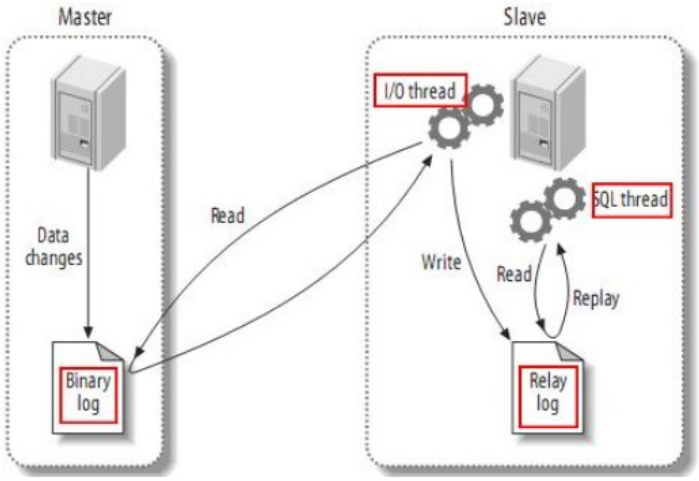

图 2.1. Master-slave 模式

图 2.1 是复制的工作原理，具体有 3 个步骤：

(1)master 将改变记录到二进制日志(binary log)中（这些记录叫做二进制日志事件，binary log events）；

(2)slave 将 master 的 binary log events 拷贝到它的中继日志(relay log)；

(3)slave 重做中继日志中的事件，将改变反映它自己的数据。

基本上大部分的分布式系统提供的冗余都是基于副本-恢复的机制，这在分布式系统子节点的文件系统不发生错误时已足以应对所有的分布式系统错误（由网络带来的文件故障不在本论文考虑范围内），市面上的分布式系统（像 Redis，Zookeeper，Kafka 等）只是在通信协议，情况处理等应用的细节上不同，像如果 Kafka 系统中的一个节点的文件入口被污染了，这个故障系统是可以监测到的但系统会把这个节点最后的一个正确日志入口之后的数据截取丢弃，按照我们的逻辑这个节点的数据已经被污染了不再完整不再干净可用，但是这个节点最终仍有可能成为 master 节点然后数据就这样安静的丢失掉了，这就跟 ZooKeeper，MongoDB 和 LogCabin 不一样，他们系统中截取掉错误数据的节点是不会成为 leader 的，但所有的分布式系统主要思想还是基本的提供冗余，只是实现细节的不同。

3 研究现状

3.1 文件系统错误

分布式存储系统的子节点运行的文件系统会不可预料的发生各种错误，具体可以分为两类：一种是块错误，即这些块不可达也就是不能读写，另一种是块数据污染，也就是说这些块里的没有包含正确数据。

当块可达的时候磁盘有时会返回很明显的错误，也就是说文件系统本身是可以识别出块错误的，比如在磁盘上的纠错码会抱怨这个块有一个位数据损坏。之前已经有一个涵盖了一百万磁盘超越 32 个月运行的报告可以显示出有 8.5%的近线存储和大概 1.9%的企业级存储磁盘也已经出现了一种或多种块错误情况，最近的一些报告也显示出相似的错误也在固态硬盘上出现。

在磁盘纠错码没有发现块错误或因为 bug 导致写错误或者块引导错误时，文件系统可能会收到“脏“数据，也就是并不是它请求的正确数据的数据。块数据污染之所以潜伏是因为数据块以磁盘没有发现的形式被污染。然后文件系统在不知情的情况下就会多次访问这些被污染的数据块，然后把这些污染数据返回给应用。在一个长达 41 个月涵盖 153 万个磁盘的报告数据里，有 400000 多个磁盘文件校验不匹配，也就是数据被污染。已经有很多报告或者文献已经说明了存储错误以及数据污染的普遍性，就存储错误与数据污染发生的频率而言，文件系统并没有一个比较好的策略来解决很多时候，

很多时候，当文件系统遭遇到来自底层的一个文件错误时，它只会简单的把这个错误按照其本来的样子传给其应用。例如，ext4，默认的 Linux 文件系统，当其底层的数据块不可达或者被污染之后，它就会简单的把 error 或者污染数据返回给上层应用。在极少的其他情况下，文件系统也许会把底层错误转化成不同的东西，比如 brfs 和 ZFS 会把底层数据块的污染数据转化成一个 error，那么其应用就会收到一个 error code 而不是毫无防备的收到一个错误的数据，在这种情况下，我们就会把文件系统丢给应用的这些错误成为文件系统错误。

3.2 分布式系统现状

现代的分布式存储系统也像单机的文件系统一样，仍然依赖于节点本地文件系统来安全的管理用户数据，然而，与单机文件系统不同的是，分布式存储系统设

定上会把数据存储在复制的分片中。其实一个认真设计的分布式存储系统可以不依赖于本地文件系统悄无声息地使用冗余特性从错误和数据污染中恢复过来，理想情况下，即使一个数据副本被污染了，分布式系统作为一个整体应该保证这个数据副本的其他副本不会被影响，并且该被污染的数据副本可以通过冗余被修复。相似的，发生在一个节点上的错误也不应该影响到整个分布式系统的正常运转。

在系统设计中，端对端的数据完整性保证以及错误处理的情形已经开始被重视。谷歌的文件系统中底层文件系统里的IDE磁盘经常会污染区服务器里的数据，这就带来了端对端故障检测与修复机制的需求。相似的，谷歌在建造大规模网络服务的经验也强调了更高层的软件应该怎样提供数据可靠性。基于分布式系统可以进行端对端数据完整保证以及错误处理的前提，本论文就现有的分布式系统怎么使用端对端手段从节点本地文件系统错误中恢复及其效果进行了相对的研究。

4 测试方法

4.1 文件系统错误模型

本研究中设计到的文件系统错误如表 4-1 所示。

表 4-1　文件系统错误分类

错误类型		文件操作	产生原因举例
数据污染	0 或者垃圾数据	读	Ext 或者 XFS 里的导航丢失或者写丢失
错误	输入输出	读	ZFS,BRFS 里的块污染或者块损坏
		写	文件系统里的文件为只读或者 brfs 文件系统中的磁盘损坏
	空间错误	写	所有文件系统中都有可能出现的磁盘已满或者配额超过

本研究中使用的错误模型有两个重要的特性：首先，错误模型固定定义每一次错误注射为在单个节点的单个文件系统数据块中只发生一个错误，因为我们想给应用最大的数据恢复余地。然后，错误模型只把错误注射入应用级别的磁盘结构中而不是文件系统的元数据中，因为应用可以收到文件系统返回的 error，所以应用来处理这种错误情形在理论上是完全可行的。

本研究的错误模型注入错误的过程是完全模拟的现实场景，结果具有极大的参考价值。例如，如果一个被标记为已被污染的数据块被写了，后来进行读取动作将会看到其上一个正确的数据，而不是污染的数据。类似的，当一个数据块被标记为读写错误，那么当这个文件被发现并且重建之后，在该数据块后来被读写时，我们不会返回错误。再比如，当一个空间错误被返回之后，后续的所有要求更多的额外空间的操作都会遭遇到相同的空间错误。

4.2 预期结果

对于每个被注入错误的分布式系统，本研究都会观察这个系统会如何反应。具

体来说就是手机系统的日志文件以及用户可见的输出例如服务器状态，返回 code，错误信息以及输出信息。一旦一个系统的反应得到之后，本研究就会把这个的行为反馈和期望的行为作对比，以下是期望的系统反应：

1. 提交的数据不应该丢失。
2. 查询请求不应该沉默的返回错误数据
3. 集群应该既可以读也可以写
4. 查询请求不应当在多次尝试之后仍然失败。

我们认为我们期望的系统反应非常的合理因为理想情况下，在单个节点的一个文件系统数据块中发生的错误不应当导致任何意料之外的情况，当我们发现有系统反应与期望不同时，我们会记录该反应现象并尝试分析相关的应用代码以及文件缺陷。

这里有一个概念是本地的表现与全局影响，在一个分布式存储系统中，多个的存储节点使用它们各自的本地文件系统来存储用户的数据，当错误被注入之后我们需要观察两件事：错误被注入的那个节点的表现以及这个错误产生的全局影响。

大部分情况下，一个本地节点会对注入的错误做出反应，其有可能会崩溃，或者部分崩溃。在某些情况下，节点可以通过重新尝试或者使用内部冗余数据进行恢复。也有可能的是，这个节点可以发现并无视掉这个错误或者只是输出一个错误日志信息。更有甚者根本就发现不了被注入的错误，更不用说对该错误做出反应。

而某个错误的全局影响是外界可见的，全局影响则由分布式系统协议里规定的如何对本地系统文件中的错误代码作出反应来决定。例如，即使某个节点可以在本地无视并且丢失污染的数据，分布式系统协议可以修复这个问题，最后产生正常的观测结果。有时根据分布式协议的不同，像全局的数据污染，数据丢失，查询失败，读写不能都是可能产生的，当一个节点本地崩溃之后，分布式系统在手动干预之前都会以更低冗余度来运行。

5 测试结果

5.1 单独的系统测试表现

原论文里对 8 个系统进行了测试研究，本文中只以 Redis 为例子进行解读。

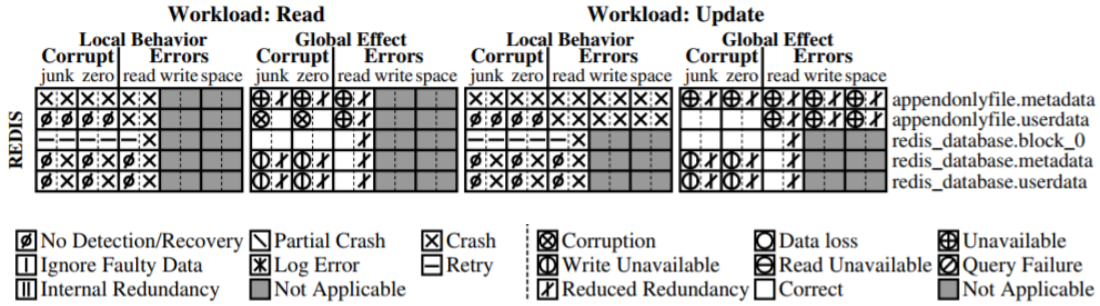

图 5-1 Redis 的系统反应

Redis 是一个流行的数据结构存储系统，用作数据库，缓存和消息代理。 Redis 使用一个简单的只可追加原则记录用户数据（appendonly file，aof）。 Redis 基于只准追加原则产生的定期快照来创建一个 Redis 数据库文件（Redis database file，rdb）。在启动期间，follower 从 leader 那里重新同步数据库文件。 而当目前的 leader 失败之后，Redis 不能主动的选择 leader。Redis 没有把校验和检验用于个人用户数据，因此，它不能检测数据污染。

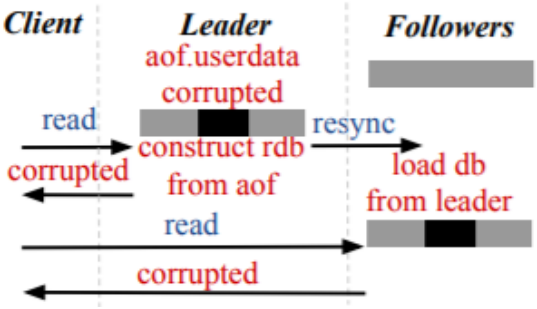

图 5-2

图 5-1 里显示了 Redis 对于读取与更新过程中 corruption 和 errors 的本地反应以及全局影响。在图里可以看到，在本地的错误处理中，几乎全部都会出错，没有任何容错能力，而全局影响中表现良好的时，本地的反应只有两个：没有发现错误或者发现错误之后重试，但这两种行为并不是百分之百奏效，也就是说分布式系统自身存在一定容错性，本地文件系统在发现文件系统错误之后进行重试这一策略也给分布式系统带来了一定的容错能力。同时当 leader 崩溃时，集群变得不可用，当 flowers 崩溃时，整个分布式系统会以较低冗余度运行。如果 leader 里的数据被

污染了，它会引起整体的用户可见的数据污染，而如果 follower 的数据被污染了，没有任何实质性有危害性的全局影响。

图 5-2 显示了重新同步协议如何将只能追加的文件中损坏的用户数据从 leader 传播到 follower 并最终导致全球用户可见的数据污染。如果 follower 被破坏，同样的协议会无意的通过从 leader 的数据中提取数据来修复数据污染问题。在只可追加文件中的元数据结构中产生的数据污染会导致 leader 产生崩溃从而导致集群不可用。由于领导在同步过程中重新发送了 Redis 数据库文件，同样的原因会导致数据损坏从而引起 follower 的崩溃。 这些崩溃最终造成集群的写入不能。

5.2 跨系统的表现总结

本节将会对 8 个系统的对错误的反应表现进行一个整体的跨系统的总结并尝试归纳理解一些分布式系统普遍存在的问题。

1） 系统使用了不同的数据完整性策略

不同的分布式系统会采用不同的数据完整性策略来保证数据的完整性，数据完整性策略不同，对相同错误的具体反应也不尽相同。Zookeeper，MongoDB 等系统尝试使用校验和检验来减少硬盘磁盘中数据污染带来的影响而有些系统比如 Redis 或者 RethinkDB 则会依赖于底层的存储快来处理数据完整性问题。

有的时候完全不相关的设置也会影响数据完整性，在 Cassanda 系统中，如果压缩选项被关闭，那么数据污染就不会被发现然后被修复，这就导致了用户可见的悄悄产生的数据污染。

本研究也发现某些分布式系统正在采用不适当的校验和检验方法，例如 ZooKeeper 使用了只适合在解压之后进行错误检测的算法 Adler32，在研究过程中 ZooKeeper 也不负众望的提供给了应用污染的数据，因此我们在思考分布式系统在选择数据完整性策略时是否应该多考虑一些因素。

2） 本地表现：错误经常不被发现，即使被发现，崩溃是最平常的本地反应

本研究发现错误在本地经常不能被检测到，有时这就会立刻导致一个灾难性的全局影响。例如，在 Redis 里，在 leader 里的只可追加文件中发生的数据污染不能被发现，这就导致了全局的静默数据污染。同时，leader 里的 Redis 数据库文件中的数据污染没有被发现，在其被复制到 follower 之后，会引起 follower 崩溃从而降低整个分布式存储系统的可用性。相似的，在 Cassadra 中，发生在 rablesst_data 中未被发现的数据污染会导致返回给用户被污染的数据并将其传播给其他的复制

副本。同样的，RethinkDB 不能检测发生在 leader 数据里的数据污染，这也会导致全局用户可见的数据丢失。而当一些系统发现了错误之后并尝试对其作出反应时，它们只是把错误当成某些行为的副作用。例如，Redis 与 Kafka 有的时候对于会把错误当成失败的反序列化的副作用。

在整个测试过程中我们观察到崩溃是本地最常见的反应。即使单个节点的崩溃不会直接有效的影响到整个集群的工作，但其他节点随后也可能崩溃会使整个系统的可用性岌岌可危。而对于崩溃的这个反应，有的时候重启并没有什么效果，节点会在重启之后再次崩溃，在人工干预之前反复重启只会带来反复崩溃，比起数据污染，崩溃更常发生在错误应对上。

3） 冗余并没有被充分利用：一个单独的错误可能会产生集群范围的灾难性影响。

在人们的普遍认知中，分布式系统的冗余机制可以帮助分布式系统从单个文件错误中恢复，本研究的观察结果中发现即使是一个单独的文件错误或者数据污染都可以在集群范围产生不利影响就像完全不可用的静默数据污染或者数据丢失或者大量数据的不可访问。在某些情形下，几乎所有的分布式分布式系统都没有把冗余当作数据恢复的来源并且会错过使用其他备份副本进行数据恢复的机会，请注意以上现象发生的前提是在整个分布式系统只在一个文件节点注入了一个文件系统错误，既然数据和功能已经有备份，意料之外的错误反应行为不应该出现，但它们都出现了。

某些分布式系统，像 MongoDB 和 LogCabin 可以自动的从一些但不是全部数据污染重恢复，具体的当遇到一个被污染的数据入口时，这些操作系统会本地无视掉出错的数据然后 leader 选举算法会保证有数据条目被污染的节点不会成为 leader。那么这个被污染的节点最终可以从由正确数据的 leader 节点中恢复，然而大部分情况下即使是这些系统也不能自动的使用冗余来进行来进行数据恢复，他们的大部分反应大部分都是崩溃。

本研究也发现当一小部分数据的出错最终会影响极大范围的数据。有的系统会丢失掉一整个日志文件，像 redis 的 aof 文件出错会导致所有文件的检测不到或者读写不能。综上所述我们发现冗余在现在的分布式操作系统中没有被充分利用，人们期望中的冗余可以提升功能以及数据的可用性的认知并不是事实。

4） 崩溃与错误处理混合交错相容。

在很多分布式系统中的错误发现以及解决的代码中都无心地发现与处理两个

比较基本的问题：崩溃与数据污染。

存储系统会小心的使用校验和检验来检测可能导致崩溃的部分更新的数据或者被污染的数据，当校验和检验没有匹配成功，也就是识别到了数据污染的情况时，所有的分布式处理系统总是调用崩溃处理的代码，即使这个数据污染不是由于崩溃引起，最后就会导致意外之外的后果比如数据丢失。

这种情况一个比较典型的例子就是 RethinkDB。它不用应用级别的校验和检验来处理数据污染情况，然而它使用校验和检验来从崩溃中恢复其元区块数据。当其的某一个园=元区快数据被污染之后，它会直接调用崩溃回复代码，该代码认为系统在最后一个事务进行时发生了奔溃然后它就会回滚事务，最后导致数据丢失。相似的当 Kafka 的日志文件被污染之后，恢复的代码会把数据污染当成崩溃的一个信号，因此它不会修复被污染的入口，而是会截断入口地址，导致之后的所有数据丢失。这些现象的根本原因就是分布式操作系统没有能力分辨镇横的存储栈的数据污染与系统崩溃的区别，这就导致了系统崩溃和数据污染的发现以及恢复的牵连混合。

5） 普遍使用的分布式协议中的细微差别可能会传播数据污染或者数据丢失。

在不同分布式系统中普遍使用的协议，比如 leader 选取以及反序列化中的细微差别可能会传播数据污染或者数据丢失。例如，在 Kafka 中，其 leader 选取协议里的一个节点的数据丢失可能会引起全局的数据丢失。Kafka 有一个同步复制的节点集，里面所有的节点理论山都可能成为 leader。而当一个 Kalfa 节点被污染之后，它会无视掉当前节点的所有后续节点直接把文件截取到最后一个正确的入口。但他仍可能会被选为 leader，这可能就会使数据丢失进行传播。这就跟 ZooKeeper 等系统的 leader 选取策略恰恰相反，它们中已经被检测到污染的节点是不可能成为 leader 的。

而读修复以及重新同步协议在不同分布式系统中的差别也会带来不同的错误处理结果，会把数据污染或者数据丢失进行大范围的传播。

结论与思考

综合本次研究的观察结果来看，就某个分布式系统的某个本地节点的文件系统错误处理而言，理想中应该大放异彩的冗余并没有提供很好的容错性，不同的分布式协议及处理策略也会导致各种意料之外的情形。这不仅引发了我的一些思考：

1. 毫无疑问冗余的提出是很有意义的也应该是很有用的，当前表现不明朗主要是分布式文件系统不能自动的利用冗余来进行数据恢复，那么基本上有两个原因：一个是冗余进行恢复的时机，一个是冗余恢复的过程。现有的文件系统对于错误的处理基本上是当作崩溃来处理，那么对于错误的精确认识方面是不是可以寻找更好的算法。在冗余恢复的过程中同样要避免文件系统带来的影响，这个以后的存储研究工作提出了更高的要求。
2. 不同的分布式系统基于不同的需求采用了多种多样的数据完整性测策略，错误应对方法以及分布式协议等，但仍然出现了相同的现象和错误处理反应，比如崩溃以及崩溃与错误处理共存，这可不可能是分布式系统或者底层文件系统导致的问题，在考虑冗余之外是否要考虑文件系统可以处理错误的可能性。

参考文献

[1] Fiala D. Detection and Correction of Silent Data Corruption for Large-Scale High-Performance Computing[J]. Parallel & Distributed Processing Workshops & Phd Forum IEEE International Sympos, 2012, 7196(5):1-12.

[2] Robert Harris. Data corruption is worse than you know. http://www.zdnet.com/article/data-corruption-is-worse-than-youknow/.

[3] Spreitzer, Mike J, Theimer, Marvin M, Petersen, Karin, et al. Dealing with server corruption in weakly consistent, replicated data systems[J]. Wireless Networks, 1999, 5(5):357-371.

[4] J. R. Sklaroff, “Redundancy management technique for space shuttle computers,” IBM Journal of Research and Development,vol. 20, no. 1, pp. 20–28, 1976.

[5] Vilas Sridharan, Nathan DeBardeleben, Sean Blanchard, et al. Memory Errors in Modern Systems:The Good, The Bad, and The Ugly[J]. ACM SIGPLAN Notices, 2015, 50(4):297-310.

[6] 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.