软件说明文档

Software Description Document

“518 三缺一”队

谢天祺　231307040028

王冬雨　231307040024

张捷　231307040030

（计算机科学与技术 学硕 2023 级）

实验课程：分布式计算

Distributed Computing

主讲教师：毛莺池 谢在鹏

复现论文：**Lunule: An Agile and Judicious Metadata Load Balancer for CephFS**

学 分 数：2 Credits

开设学院：计算机与软件学院 College of Computers & Software

开设时间：2023~2024 学年第一学期 1st Semester, 2023~2024

河 海 大 学

目录

1 项目简述

1.1 复现论文简述

本次分布式小组作业选取了 2021 年 SC 会议中的 Lunule: an agile and judicious metadata load balancer for CephFS 进行复现。Cepfs 采用一种动态的子树分区方法，将命名空间分层，并将子树分布到多个元数据服务器上。然而，由于不准确的不平衡预测、忽略工作负载特征以及不必要的/无效的迁移活动，该方法存在严重的负载问题，可能导致性能低下。本文提出了新的元数据负载均衡器 Lunule，它精确地模拟了 MDS 的负载分布和不平衡的紧迫性，并根据不同的工作负载动态地调整迁移计划，包括迁移量的确定和子树的选择。

1.2 复现内容简述

文章介绍的 Lunule 是本次复现的主要内容，这是一种基于动态子树分区的 CephFS 元数据负载均衡器。CephFS 动态子树管理中存在严重的元数据负载不平衡问题，导致元数据性能不佳和资源浪费，究其根本原因存在两个方面的问题，一是负载监控模块无法准确建模各个 MDS 的负载和集群的负载，不能容忍良性的失衡，负载失衡水平停留在安全区间。二是迁移决策模块不恰当地选择子树迁移候选，没有考虑这些子树的访问模式和未来的负载方差。

为了克服这两个关键性的问题，文章提出了 Lunule 负载均衡器，Lunule 是由一个分析模型驱动，该分析模型准确地捕获了整个 MDS 集群的工作负载强度级别，能够在需要时进行重新的平衡，此分析模型不使用平均负载统计，而使用变异系数计算 MDS 集群的实时不平衡因子，最大限度地减少噪声对迁移决策的负面影响。同时 Lunule 引入了一个敏感的参数来量化不平衡情况是否安全，来减少不必要的迁移操作。

Lunule 进行负载均衡操作时，需要确定输入和输出的 MDS，输出的 MDS 元数据负载较大，需要将负载迁移到其他的 MDS 节点，输入的 MDS 需要有足够的容量在容纳传入的负载，并通过考虑 MDS 上未来的负载变化计算两个不同身份的 MDS 之间迁移多少数据。在完成重新的平衡之后，Lunule 会选择每个输出的 MDS 上的某一组子树来实现迁移决策，无效的子树迁移会混乱 MDS 集群的负载平衡，准确地预测不同子树的未来访问频率，能够帮助实现良好的负载平衡，因此需要设计一个迁移指标预估子树过去访问活动的时空局部性对其未来负载的影响。对于时间局部性的影响问题，文章并不选择当前 CephFS 中使用的简单的累积流行计数器，而是考虑最近时间间隔内测量元数据访问的复发性，同时考虑空间局

部性的影响，即目标子树的元数据访问的均匀分布和兄弟子树之间的访问相关性。迁移指标大小与迁移子树候选项选择成正向的关系。

1.3 相关工作

本小组首先基于论文内容复现了基于动态子树分区的CephFS元数据负载均衡器Lunule。其具有以下两个关键任务,一是 Lunule 需要随着时间的推移准确监测和报告整个 MDS 集群的不平衡强度水平，二是依靠实时收集的统计数据，Lunule 应该能够为各种工作负载制定合适的计划，精心选择合理数量的子树分区作为 MDS 集之间迁移的候选分区。本次复现工作在阿里云平台进行，使用 16 个节点服务器，操作系统为 CentOS7.9，其中一个节点安装 CephFS 作为 Manager，其余 5 个服务器作为 MDS、9 个服务器作为 OSD、1 个服务器作为 Client，构建相关依赖环境后进行复现，使用 CNN：ImageNet DataSet ILSVRC2012 数据集成功完成了实验。

本组复现论文主要完成了以下工作：

1. 根据开题报告中的实验环境要求构建基本的实验环境；
2. 构建相关依赖环境后，完成代码的编译和插入；
3. 下载并使用数据集 ILSVRC2012 进行测试，对实验结果进行测量和统计。

2 相关工作

2.1 CephFS 介绍

CephFS 是一种被广泛采用的开源、兼容 posix 的分布式文件系统(CephFS)[1]。它的目标是高性能、大数据存储，并最大限度地兼容各种应用，即从互联网服务到人工智能计算等许多关键领域不断增长的大数据是它最主要的应用方面。

在 CephFS 中，元数据与数据分开管理。元数据也就是命名空间的空间层次结构，这种分开管理使得元数据和数据都可以独立扩展。在这样的架构中，必须在实际的数据访问之前先获得元数据。随着分布式系统的不断应用发展，许多文件系统工作负载是元数据密集型的，即超过 50%的文件系统操作集中在元数据上[2]，同时绝大多数文件都很小[3]，使得 SSD 等新兴存储设备良好的适应了元数据的传输，这样小而繁的元数据的重要性不断提高。

十年来，CephFS 一直遵循 MDS 的传统模式。这中以 MDS 为基础的操作是大多数现代分布式文件系统的首选，MDS 之间平衡负载能够反应分布式文件系统的特点，使得在内存中缓存可以缓存更多多的元数据，同时并行处理元数据请求[4]。与单纯的数据集群相比，扩展 MDS 集群的性能更值得研究，主要是因为元数据包含文件系统结构信息，并且表现出更高程度的相互依存[5]，拥有文件系统的结构信息能够更好的在空间上平衡负载。为了扩展大规模的元数据访问，CephFS 采用动态子树分区的方法，将分层命名空间拆分更小的单位，即将子树分布在多个元数据服务器上，并根据每个 MDS 的工作负载强度水平在 MDS 之间定期迁移这些子树[6]。

2.2 国内外研究现状

近年来，在使用 CephFS 的过程中，传统的 CephFS 的处理效率和处理能力不能满足大数据时代日益增长的数据量、多变的数据特征的需求，因此国内外研究人员采取了不同的思路对 CephFS 的进行了一定的创新和优化。

为了满足元数据的特性，研究者提出了类似于 SkyFS[7]这种采用基于哈希的映射，通过计算对应文件名或路径名的哈希值将文件的元数据与授权的 MDS 进行配对。尽管 MDS 之间的元数据分布均匀，但它破坏了空间局部性，这种方式使得静态分区很难适应动态的变化，扩展 MDS 集群的难度过高，并不能满足现实中大数据应用的要求[8]。

既然基于哈希的映射方式并不能满足现实要求，所以 CephFS 中的采用动态子树分区机制来提供更好的灵活性，并且能够在局部性和负载分配之间取得平衡。尽管从运行结构的角

度来看 CephFS 的动态子树分区机制考虑到了一些解决动态问题的尝试，但 CephFS 动态子树管理中存在着严重的元数据负载不平衡问题，导致元数据性能不佳和资源浪费。元数据的访问通常并不是均匀分布在多个 MDS 之间，在最坏的情况下，负载最大的 MDS 处理的元数据请求量比负载最少的 MDS 处理的元数据请求量甚至会高出 220 倍，即使在后台有频繁和活跃的迁移活动。

提高 CephFS 中动态子树分区元数据管理性能的研究还有待挖掘。Mantle[9]将负载统计收集和迁移决策步骤与 CephFS 元数据管理的其余部分全部解耦，通过可编程 api 允许用户指定函数来确定何时迁移以及迁移多少，api 作为一个有限的载体，并不能全部涵盖动态运行过程中重要的子树选择特性。因此导出准确的负载模型和合理的启发式元数据迁移和负载平衡仍然是该领域研究的重点内容。

2.3 Lunule 负载均衡器的贡献

为了解决前文所述在该领域所面临的问题，复现的实验将动态的选择过程合并到 CephFS 中，并通过扩展 CephFS 的元数据服务构建 Lunule 负载均衡器。文章对 CephFS 动态子树分区机制中的负载不平衡现象进行了全面研究，并确定了其在 MDS 集群中迁移负载效率低下的根本原因。根据 CephFS 动态子树分区机制的特性，设计了一种新的元数据负载均衡器 Lunule，准确地模拟了 MDS 负载分布和失衡的紧迫性，并在迁移量确定和子树选择方面动态适应不同工作负载的迁移计划。文章还对 Lunule 在五个实际工作负载及其混合上进行了深入的评估，结果表明 Lunule 的元数据性能明显优于 CephFS - Vanilla 和 GreedySpill 两个作为基线的标准，而且 Lunule 可以在 16 个 mds 的集群上继续进行线性扩展。

3 问题分析

在文章中作者使用 5 个典型的元数据繁重工作负载，测试了使用 5 节点 MDS 集群部署的 CephFS，这些工作负载来自不同的领域，从 web 服务器访问到机器学习，测试数据集中元数据访问对工作负载性能至关重要,以此可以显示元数据负载平衡的重要性以及实现它所面临的挑战。Lunule 负载均衡器主要解决了以下两个 CephFS 存在的问题，即容易产生负载失衡和存在无效迁移的问题，将在以下两个小节进行具体分析。

3.1 容易产生负载失衡

首先需要分析 CephFS 本身的动态子树分区的核心特性，每个 MDS 上的负载平衡器将名称空间划分为子树和目录片段（dirfrags）。子树是嵌套目录和文件的集合，而目录片段是单个目录(其大小超过特定大小)的分区。这些元数据在分布式系统的工作负载发生变化时,以分片的形式在 MDS 之间进行迁移，最终实现简单的负载平衡。简而言之动态子树分区机制就是根据不同的 MDS 服务器的使用模式和当前集群负载，动态智能地将任意子树重新委派给不同的服务器。整个负载均衡过程是循环的四个运行步骤，在第一阶段，MDS 节点参与元数据负载统计，判断 MDS 集群是否存在不均衡，是否存在不均衡，如果曾在这样的情况,负载均衡就要被触发,触发后进入第二阶段,需要将 MDS 划分为输入 MDS 和输出 MDS，并计算输入 MDS 与输出 MDS 之间的迁移负载数量。在第三阶段，每个输出的 MDS 需要选择其内部的一组字数或者是部分目录片段,将它们发放至输出队列中,等待计划的负载迁移。在第四阶段,输出前检测对于输出负载的格式要求后根据标准两阶段提交协议来完成真正的迁移。

在知晓 CephFS 的动态子树分区后，根据文章中的实验结果可以进行进一步的分析，所有的实验表明了元数据操作不平衡的现象在所有工作负载中都会存在,只是严重程度的区别,因此可以认为对于元数据大量处理的情况中动态子树分区机制一定会浪费不少的资源,究其原因是 CephFS 内置的元数据负载平衡器总是会错误地决定不重新平衡，它总是认为最繁忙的 MDS 负载接近平均值，不会考虑不同 MDS 之间的负载差值。同时内置的平衡器也会因为负载最高的 MDS 与平均值相差较大而在整体集群负载适中的情况下依旧选择了迁移，可以得出结果，CephFS 内置的元数据负载平衡器经常无法准确地理解集群状态，从而导致不能及时且合理的发现不平衡情况进行重新平衡，因此容易产生负载失衡。

3.2 存在子树无效迁移

CephFS 的动态子树分区还存在着无效迁移的情况，内置的平衡器会仅仅交换负载，对于整体性能几乎没有帮助，反正浪费了系统性能，这种效应是因为负载负载沉重的 MDS 会计划将元数据尽可能多地迁移到其他对等节点，在一次重新平衡时间间隔内通常会超过最大迁移容量，在确定迁移的负载后，并不考虑元数据迁移的滞后效应，导致了过度的迁移，迁移完成后，输入 MDS 急剧变热，而输出 MDS 可能会闲置，因此可以得出结论 CephFS 内置的元数据负载平衡器过于激进的迁移决策无法发挥负载平衡的优势，反而引入了新的不平衡情况。

CephFS 的动态子树分区机制在 NLP 等任务上会出现迁移趋势与 MDS 繁忙程度矛盾的情况，这是因为内置的元数据负载平衡器是基于时间位置的平衡策略，单纯选择热点作为迁移候选者，并不考虑是否之后会重新扫描文件，从而导致了内部的负载平衡机制无效了。这反映了动态子树分区机制的内部负载平衡器策略过于单调，并不能考虑工作负载的特性，不能适应不同的负载任务。

3.3 本章小结

在分析玩所有的 CephFS 的动态子树分区所存在的问题后，可以考虑如何去解决上述三种问题。首先，CephFS 内置的元数据负载平衡器经常无法准确地理解集群状态，从而导致不能及时且合理的发现不平衡情况进行重新平衡，这代表负载的迁移会出现可能与前台请求争用资源的后台流量的情况，必须更加准确地监控集群负载并确定何时触发重新平衡过程。其次，CephFS 内置的元数据负载平衡器过于激进的迁移决策无法发挥负载平衡的优势，反而引入了新的不平衡情况，这要求负载平衡器应该就迁移多少负载做出正确的决定，以减少迁移频率并避免长期迁移作业。第三，内部负载平衡器策略过于单调，并不能考虑工作负载的特性，不能适应不同的负载任务，这要求必须适当地找到一组目录作为无效迁移避免的迁移候选者，以满足负载任务的特性。

4 源代码功能分析

4.1 Lunule 的体系结构

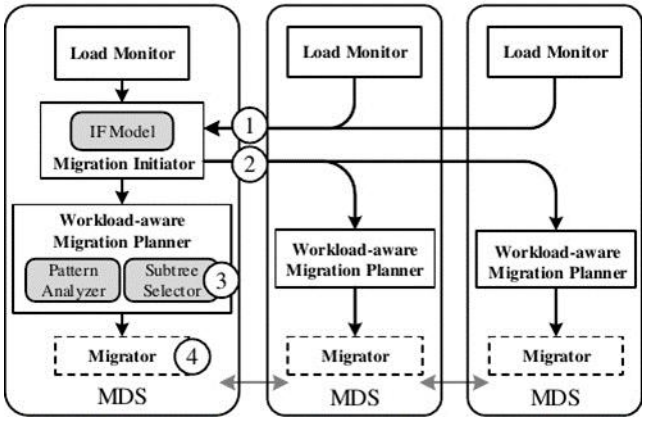

图 1 Lunule 体系结构视图

上图给出了 Lunule 的体系结构视图，它用三个关键组件增加了现有的元数据服务。

首先，Lunule 在每个 MDS 上部署一个负载监视器（Load Monitor）来监视负载压力，并收集相应 MDS 每秒处理的元数据请求的数量，这在 src/mon/MDSMonitor.cc 中实现，对元数据的请求处理主要通过 update_metadata，remove_from_metadata，load_metadata，count_metadata，dump_metadata 五个函数共同完成，分别执行更新，撤销，载入，计算，挂载五个功能，其中 count_metadata 函数存在两个不同的输入用来计算，分别是对应子树分片和目录分片，两者相互转化能够成功收集 MDS 的元数据请求，保证了 Lunule 能够监视负载压力。

另一个关键组件是感知工作负载的负载感知迁移计划器（Migration Planner），它也部署在每个 MDS 上独立运行，其功能主要在 src/mon/MgrMonitor.cc 文件中实现。文件中它主要依靠 load_metadata，count_metadata，dump_metadata 三个函数对元数据进行加载、计算、挂载等基本操作。之后在将其进一步划分成两部分，第一个是模式分析器（Pattern Analyzer），它学习不同工作负载的 I/O 模式，集中在相应 MDS 管理的子树上，并通过理解过去的工作负载的影响和预测未来元数据访问的差异来计算迁移的概率。第二个是，当触发重新平衡时，在输出的 MDS 上，有一个子树选择器（Subtree Selector）选择一组适当的子树进行迁移，该子树满足迁移启动器计算的数量，并适合目标工作负载的未来访问。第三个组件是在集群的 MDS 上有一个迁移启动器（Migration Initiator），用于实时决定何时应该进行迁移，以及应该在元数据服务器之间交换多少元数据等信息，这主要在 src/mon/MgrStatMonitor.cc 文件中实现，其中不仅对其进行了初始化等静态配置，同时对整个环境进行了监控，能够检测到服务器之间的元数据交流。所做的决策依赖于 IF 模型的值，此值汇总了 MDS 之间的负

载分布，从负载监视器其中收集的统计信息根据 IF 分析模型计算得到。虽然迁移启动器是一个集中式组件，但它不会成为性能瓶颈，因为迁移过程发生在后台，每次 epoch 都运行，并且消耗很少的资源，这在 src/tools/rbd_recover_tool/epoch_h 中实现，默认情况下 epoch 是 10 秒钟，其中 Lunule 在一个 Epoch 的固定时间内做出迁移决定。在每个 epoch 中，每个负载监视器将观察到的元数据吞吐量数发送给迁移启动器，当元数据集群的不平衡度，即 IF 值超过预定义的阈值时，迁移启动器触发负载重新平衡过程，并生成迁移计划，将输出（exporter）和输入（importer）的角色分配给 MDS，并将输出和输入的需求和能力进行配对。之后工作流感知迁移计划器将通知每个输出 MDS 上分配的迁移任务。在迁移任务的到达时，子树选择器选择合适的子树分区列表，然后将其输入到现有的迁移器中，从较重负载的 MDS 重新定位到负载较轻的 MDS。

4.2 IF 模型再平衡

Lunule 设计了一个新的一个分析模型，将收集到的元数据负载状态作为输入，并预测整个元数据集群的不平衡因子值，表示为 IF，代表每个 epoch 元数据负载不平衡的强度水平，这在 src/boost/boost/hana/if.hpp 中进行了定义，在结构体 if_impl 中对于 IF 值的计算和比较都提供了封装，IF 分析模型的优势在于解决了所使用的线性模型所实现的预测不准确性，并识别出不需要再平衡的不平衡情况。

文章中并没有使用依赖于与收集的平均负载数量的简单比较的线性元数据负载建模方法，因为即使有大量的迁移活动，根据第三章中的分析可以看出线性模型依旧会导致某些 MDS 上的负载倾斜。文章采用变异系数(Coefficient of Variation ,CoV)作为模型的基本构件。CoV 是数据点在平均值周围分散的统计度量，通常用于比较不同系列数据之间的数据分散程度。文中选择使用 IOPS 作为主要度量来估计目标 MDS 集群的忙碌程度，因为 IOPS 能够自然地反映每个 MDS 的即时负载，这在 src/boost/boost/multiprecision/detail/integer_ops.hpp 中进行了实现，在其中 eval_lcm 和 eval_gcd 函数的帮助下，对于负载不均衡的程度将变异系数进行了归一化处理来对应于最不平衡的情况，不直接使用变异系数的值，反而转化为使用一个预定义的阈值来表示是否需要进行再平衡。在某些情况下，不平衡的情况不一定都需要再平衡，如果负载不同的情况下，所有 MDS 都大大低于其最大吞吐量。为了解决这种情况，模型中额外引入了一个紧急参数 U 来描述当前的不平衡是否是良性的，其中 U 值越高，表示需要迁移的趋势越急，因此将 U 建模为一个 logistic 函数，这主要是在在 src/boost/boost/math/distributions/logistic.hpp 文件中进行了实现，其中通过 cdf 函数和 pdf 函数对 U 值进行了转化，quantile 函数表明了其增长受到最大负载 MDS 负载的限制。U 表示当前最重的负载与最大 MDS 的吞吐量相比的情况，实质就是当负载低于最大吞吐量时降低的 IF 值。

迁移计划器利用上述模型计算在每个 epoch 的 MDS 集群的 IF 值，当负载不平衡程度超过预定义阈值时，则进入角色确定阶段。角色决定器从每个 MDS 节点收集负载的统计信息，计算出一个矩阵 MDSij。矩阵中的每一项表示从 i 到 j 需要发送的负载数，迁移的数量值（amount）对于最大化迁移利益和避免过度迁移的负面影响起着关键作用。为了更好的完善

优化目标，文章进行了三个方面的改进，这在 src/boost/libs/range/test/algorithm.cpp 文件中可以看到简化后的改进流程，考虑到迁移带来的开销，每个输出和输入 MDS 的输出和输入需求设置了上限，并将其设置为一个 epoch 内的最大容量。该容量为一个固定值，可以计算一个 MDS 可以发送或接收的最大索引节点数。第二，对于每个输入，需要进一步考虑其未来负载变化的影响，以避免过度迁移。因此收集历史负载统计应用线性回归模型来预测下一个 epoch 的可能负载。如果未来的负荷增加不能满足缺口，则可以分配输入角色，并且计算它的预期输入量。第三，对于输入输出对，检查输出是否有迁移需求，同时检测输入是否有能力容纳这种迁移，如果能，使迁移量等于迁移需求和输入能力的最小值，然后从输入和输出的收益中减去相应的数值，使得输入和输出在生成迁移计划时满足双向需求，满足上述改进情况在文件中提供了三种情况的处理过程，即 test_random_algorithms 函数可以有三种种输入情况，分别是随机的输入，设定的人数，和默认的输入，对于三种情况仅仅是输入处理上的区分，流程上没有太大区别，依旧是以上述三个改进方面最为顺序执行。

4.3 子树选择

进行迁移决策后，就需要选择一组适当的子树，将其从输出移动到输入，这在文件夹 src/boost/libs/beast/subtree 中的各代码文件提供了子树的各项基本属性与操作。执行子树选择决定了在相应迁移后是否能实现目标平衡，由第三章的分析可知，CephFS 内置的迁移策略无法在各种场景下提供良好的性能，基于局部热点的原解决方案忽略了空间局部性的影响，在人工智能和大数据分析场景中，空间局部性比时间局部性更占主导地位，而且子树的局部热点也无法模拟未来负载的显著方差。

在 Lunule 中，为了将迁移候选选择阶段转变到下一工作负载感知阶段，首先需要将为子树分配一个迁移索引（mIndex）,这个索引对应于目标子树随着时间推移所预测的未来负载。迁移索引越大，对应子树被迁移的概率就越高。迁移具有更高迁移索引的子树可能能够将繁忙 MDS 上多余的工作负载转移到负载较轻的 MDS 上，从而消除无效但是浪费性能的迁移。

迁移索引的计算需要综合考虑工作负载中表现出的时间局部性和空间局部性的影响，这在文件 src/erasure-code/jerasure/jerasure/src/jerasure.c 中进行了实现，其中 jerasure_make_decoding_bitmatrix 对迁移索引的计算进行了详细的过程，其中分别引入 a 和 b 作为时间局部性和空间局部性的影响因子，表明子树上最近的工作负载倾向于这两种访问模式中的一种。在每个 MDS 上维护历史元数据访问跟踪并将其分解为固定大小的短序列来估计这两个影响因子的值。每个 epoch 周期计算最近的短序列的循环访问比率，即循环访问的索引节点数除以总索引节点数，这个比率是在每个子树的基础上计算的，同时跟踪每个子树中未访问的索引节点，并将最近的短序列中未访问的索引节点与总元数据访问数的比值分配给 b。除了估计时间和空间的局部倾向性外，还需要预测未来这两类的访问数量，通过计算最后 N 个短序列中对应子树的元数据访问次数获得对应的 lt 和 ls。考虑 lt 的情况对于一个给定的子树，如果它的一个未访问的索引节点在当前的切割窗口中被访问，就把它的 ls

增加 1。最后，当工作负载呈现空间局部性时，兄弟子树之间存在很强的访问相关性,将以一定的概率选择它的一个兄弟子树，并将所选子树的 ls 加 1。

每个 MDS 根据其迁移索引按降序对其管理的子树集进行排序，这主要在文件 src/mds/MDBalancer.cc 中进行了实现，handle_export_pins 函数对输出子树进行了规定，get_load 对输入子树进行了规定，send_heartbeat 函数对局部热点进行的查找，prep_rebalance 函数在进行负载平衡时进行预先的平衡，simple_determine_rebalance 函数为简单的负载平衡决策，try_rebalance 函数是尝试进行平衡，subtract_export 函数为了获取一组子树，其中对于每个迁移决策三元组<输出，输入，数量>，输出 MDS 首先扫描其排序的子树列表，尝试找到一组子树，让子树的总负载数匹配三元组中的数量。对于每个遇到的子树，如果它的迁移索引等于或低于三元组中的数量，就将其放入候选列表中，然后相应地减少数量。否则就必须拆分扫描的子树，如果元数据访问集中在子树本身，就将它分成两个子树，其中一个迁移索引匹配决策三元组中的数量，如果它的一些后代子树是局部热点，那就根据数量将它们从祖先中移除。

参考文献

[1] Red Hat. Ceph file system official site. https://ceph.io/ceph-storage/file-system/,2021.

[2] Cristina L Abad, Huong Luu, Nathan Roberts, Kihwal Lee, Yi Lu, and Roy HCampbell. Metadata traces and workload models for evaluating big storage systems. In UCC, 2012.

[3] Shuanglong Zhang, Helen Catanese, and Andy An-I Wang. The composite-file file system: Decoupling the one-to-one mapping of files and metadata for better performance. In FAST, 2016.

[4] Siyang Li, Youyou Lu, Jiwu Shu, Yang Hu, and Tao Li. Locofs: A loosely-coupled metadata service for distributed file systems. In SC, 2017.

[5] Sage A Weil, Kristal T Pollack, Scott A Brandt, and Ethan L Miller. Dynamic metadata management for petabyte-scale file systems. In SC, 2004.

[6] Sage A Weil. Ceph: reliable, scalable, and high-performance distributed storage. PhD thesis, University of California, Santa Cruz, 2007.

[7] Jing Xing, Jin Xiong, Ninghui Sun, and Jie Ma. Adaptive and scalable metadata management to support a trillion files. In SC, 2009.

[8] Y. Chen, C. Li, M. Lv, X. Shao, Y. Li, and Y. Xu. Explicit data correlations-directed metadata prefetching method in distributed file systems. IEEE TPDS, 2019.

[9] Michael A. Sevilla, Noah Watkins, Carlos Maltzahn, Ike Nassi, Scott A. Brandt, Sage A. Weil, Greg Farnum, and Sam Fineberg. Mantle: A programmable metadata load balancer for the ceph file system. In SC, 2015.