数据放置对大型对象存储系统快速恢复能力的影响

刘仁辉 2017282119392

Abstract—分布式对象存储体系结构已经成为大数据，云计算和高性能计算中高性能存储的事实标准。使用商品硬件来降低成本的对象存储部署通常采用对象复制作为实现数据快速恢复的一种方法。对于有数千个乃至十亿个对象的存储系统来说，在故障发生后修复对象副本是一项令人望而生畏的任务，然而，在真实世界系统上大规模评估这汇总情况越来越困难。如果对象没有及时修复，数据恢复率和可用性都会受到影响。

在这项工作中，本文利用高保真离散事件仿真模型来调查大型对象存储系统上的副本重建，这些副本具有数千个服务器，数十亿个对象和数 PB 的数据。我们评估的行为 CRUSH(一种众所周知的对象放置算法)，并确定聚合重建性能受对象放置策略限制的配置方案。在确定了这种瓶颈的根本原因之后，我们提出了对 CRUSH 及其上的使用策略的增强，以启用可伸缩副本重建。我们使用这些方法在 1,024 节点的商品存储系统上展示 410GiB/s 的模拟聚合重建率（在理想线性缩放比例的 5% 范围内）. 根据系统存储的数据特点，我们还发现了一个意外的重建性能现象。

Index Terms—Storage System, Resilience.

1 引言

分布式对象存储体系结构在大规模存储系统中被广泛应用于各种问题领域。使用商品硬件降低成本的对象存储部署通常采用对象复制作为实现数据恢复的方法。因此，这些系统必须使用分布式重建算法，通过更换丢失的副本来恢复失败事件。分布式重建对于有数千个服务器和数十亿个对象的系统来说是一项艰巨的任务，因为数据的数量以及所涉及的服务器之间的协调程度。这导致了大规模的一个基本问题存储系统：系统如何从存储故障中恢复？该程序的执行（即重建总速率）对数据平均丢失时间（MTTDL）TDL) 有直接影响，特别对于大于两个替换丢失的副本所用的时间越长，后续故障将导致数据丢失的可能性越高。当系统处于降级状态时，感知的应用程序响应时间也可能受到影响。

对象放置算法（即用于将对象副本映射到可用服务器的机制）是影响性能，局部性和负载平衡的存储系统设计的关键要素。对象放置算法还决定了复制的分组以及因此在重建期间可以实现的总并行量。最初在磁盘阵列的背景下提出了解聚集作为在不同子集的磁盘上分布 RAID 条带单元的方式，以便在重建过程中实现更好的负载分配。在复制对象存储系统的情况下，参与复制是存活服务器均匀参与的重建，而完全集群算法将重建负载本地化为尽可能少的受影响服务器。在实践中，大多数布局算法落在这两个极端之间。我们将分析的重点放在算法放置上，通过将确定性函数应用于对象 ID 和候选服务器 ID 来计算存储位置。算法放置是分布式系统设计中流行的选择，因为它消除了为每个对象存储显示布局元数据的需要，并且允许任何对象客户端或服务器独立（）但一致）的计算位置。

解聚集，对象放置和聚合重建性能。显然是存储系统设计的关键要素。然而，分布式对象存储系统的成本和复杂性使得难以对这些组件进行大规模的实验评估。因此，大规模故障恢复的细微差别还不是很清楚，例如，性能上最薄弱的环节是什么，以及如何重建会受到存储在系统中数据性质的影响？最糟糕的是，分布式存储协议可能包含微妙的性能或正确性缺陷，很难直观地发现。我们通过开发高保真分布式对象存储模拟器来解决这个问题，以调查具有数千个服务器，数十亿个对象和 PB 级数据存储容量的存储系统的行为。

我们的分析揭示了大规模的对象放置方法的微妙局限性，并指出了未来存储系统可以采取的方法来解决这个问题。我们发现一个有效的对象放置算法可以实现聚合重建性能的近线性缩放；我们的仿真模型使用了 1,024 个商品存储服务器实现了超过 400GiB/s 的聚合吞吐量。这项工作的贡献包括了一下几点：

1) 使用现有众所周知的物体放置方法评估物体存储系统重建效率的案例研究；
2) 识别和评估对象布局优化，可以提高重建效率；
3) 一个高保真的对象存储系统模拟器，可以为数千个服务器，数十亿个对象和 PB 数据存储容量；
4) 对大规模数据的分析，对其特征进行建模的方法，以及对实验结果的影响。

2 背景

本文的目标用例是一个水平可扩展的数据中心存储系统，由数百或数千台服务器组成，这些服务器由商品组件构建而成，能够为高性能分布式和并行应用程序托管许多 PB 的复制数据。服务器是同类的，配置由系统管理员控制。我们期望服务器故障频繁但不连续。因此，团体成员也相对静止。

在生产过程中可能会出现各种故障模式，但为了清楚起见，我们将重点放在单个服务器完全失败的情况，以使其数据不再可访问。这是了解基准重建行为的一个直接起点。在以后的工作中可以研究更复杂的故障模式。

A. 对象放置

对象放置算法是用于将给定对象及其副本映射到分布式存储系统中的服务器子集的机制。许多大型存储系统选择使用确定性数学函数来简化设计，避免为每个对象存储显示的布局元数据，最流行的确定性算法是一致哈希。基于数字距离度量，将对象标识符一致哈希值映射到“K 最近”服务器标识符。计算是稳定的，如果一个服务器失败，他会仅影响该服务器所拥有的对象副本的映射。所有幸存的复制品都保留在以前的位置。这最大限度地减少了失败后必须传输的数据量。

- 论文题目：*Impact of Data Placement on Resilience in Large-Scale Object Storage Systems.*
- 论文来源：*Mass Storage SystemTechnologies,2017.*

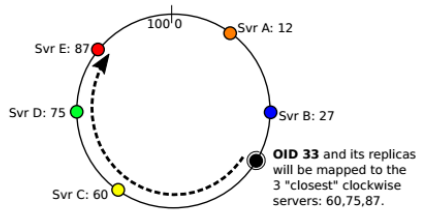

Fig. 1. Object placement example: a one-dimensional consistent hashing ring.

图 1 显示了一个一维环一致哈希方法的例子。为了简单起见，我们将对象 ID 和服务器 ID 空间限制为 0 到 100 之间的值。服务器被分配数字标识符，这些数字标识符可以被视为环上的位置。在这种情况下，服务器 ID 是随机的：12,27,60,75 和 87，尽管一些系统可以明确地分配 ID 以便更均匀地将他们分布在 ID 空间中。每一个对象都映射到环上对象 ID 位置顺时针方向 K 个最近的服务器 ID。请注意，这个环上的“距离”是一个与物理位置无关的虚拟构造。在该示例中，对象 33 跨服务器 60,75 和 87 被三方复制。如果服务器 75 失败，则两个幸存的复制品将保持在原位，并且服务器 12（顺时针方向的下一个最接近的）将负责生成一个新的副本取代它的地位。这个算法可以通过多种方式进行扩展，特别是通过添加虚拟服务器，是给定的服务器出现在环中的多个位置，以更好地平衡平均负载。

更复杂的布局算法包括由 Weil 等开发的 CRUSH。并构成了 Ceph 文件系统的关键组件。CRUSH 将存储目标组织到分层集群映射中（例如，按行，机柜，架子和设备）。放置规则控制副本在该层次结构中的分布方式。例如，放置规则可以强制给定对象的副本跨越故障域以提高恢复率，或者放置规则可以强制将副本放置在同一个存储桶中以改善局部性。

集群映射层次结构的每一个级别都称为一个存储桶，每个存储桶使用可插入算法将对象映射到目标。从某种意义上说，CRUSH 可以被认为是一组放置算法，按照灵活的放置规则组织成层次结构。最受欢迎的桶类型是稻草桶。稻草桶算法是通过将桶中的每个候选目标的对象 ID，副本号码和目标 ID 一起进行散列来为对象选择位置。选择“最长吸管”或最高散列值的目标来存储对象。也可以为每个目标分配一个标量权重值，以影响目标在整个目标上的分布。然而，Ceph 文件系统（使用 CRUSH）实际上并不是为了存储系统中的每个对象执行 CRUSH 算法。而是将对象转换为更小的固定数值的放置组，并且每个组计算放置位置，而不是按照目的。

在以前的工作中，我们开发了一个成为 libch-placement 的模块化一致哈希库来评估一致性哈希算法的权衡。在这项工作中，我们在相同的抽象 API 下添加对 CRUSH 算法的支持。在这项工作中使用的重建模拟器利用 libch-placement 来互换地使用各种 CURSH 或一致的哈希算法来放置数据。

B. 数据群体

大规模存储系统的行为不仅取决于算法决策，还取决于存储数据的性质。诸如对象大小的数据总体特征可以影响各种运行时行为，诸如控制与数据消息的比率，磁盘性能（寻道时间与流式传输吞吐量），流水线传送深度以及服务器对的持续时间由个别对象转移占用。

图 2 比较了三个不同示例数据群体中文件大小的分布情况.1 前两个（基因测序数据的 1000 基因组目录和网络爬虫数据目录的常见爬网语料库）作为亚马逊公共数据集。它们的目的是通过使用 Hadoop 框架进行处理。第三个例子显示了在由阿贡领导计算机构管理的 IBM Blue Gene/Q 系统 Mira 上使用的 GPFS 并行文件系统的内容。Mira 包含来自各种科学领域的数据，以支持 DOEINCITE 程序，数据通过传统的文件系统接口或高级并行 I/O 库来访问。这些直方图显示每

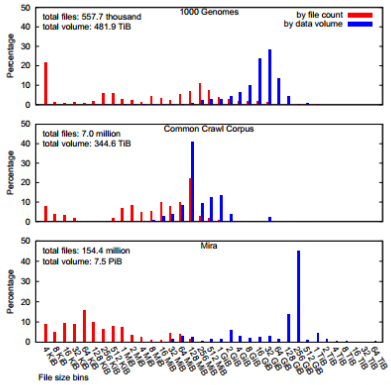

Fig. 2. Distribution of file sizes across example data sets.

个文件大小的文件的数量（红色）和数据量（蓝色）。这是一个重要的区别; 例如，1000 Genomes 数据集包含大量 4 KiB 或更小的元数据文件，但它们构成整个数据量的一小部分。

在这项研究中，我们关注 1000 个基因组用例。为了生成具有代表性的对象群体，我们首先通过 1000 个基因组直方图的加权随机采样来选择文件大小。然后，我们根据 Hadoop 文件系统分块策略将文件划分为组成对象：将每个文件拆分为不同的 64 个 MiB 对象的集合。这种直方图抽样方法可以生成各种规模的替代对象种群，同时仍保留现实世界数据集的总体特征。图 3 显示了这个合成数据总体生成策略的应用示例。我们生成了一个 20 PiB 数据集基于 1000 个基因组的人口，并将其分布绘制在图的顶部，以确认它与图 2 顶部图中观察到的趋势相匹配。图 3 中的第二个图显示了相同合成数据集的对象大小的直方图假设每个文件被分成 64 个 MiB 对象。尽管存在较小的物体，但人口由包含的 64 个 MiB 物体支配人口中最大的文件数据块。

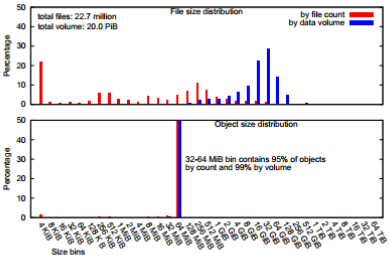

Fig. 3. Distribution of file sizes in a synthetic data set generated by weighted histogram sampling of the 1000 Genomes histogram.

C. 重建协议

一旦在分布式对象存储系统中检测到并确认了一个故障，则存活的服务器必须生成新的副本以恢复原始的恢复水平。我们使用术语重建协议来指代这个过程中的步骤。重建协议可以根据系统中的哪个实体（或多个实体）协调数据传输，数据是被推还是拉，以及数据如何本身被传送（即并发和流水线）。

尽管一些存储系统选择从集中式子节点驱动重建协议，但我们的仿真器遵循类似于 Ceph 的模型，其中每个存活服务器独立负责生成其自己的副本副本。因此，每台服务器都以自己的速度恢复高度的并发性。这种方法自然地被“拉”传输模型所补充，其中目标服务器明确地向源服务器请求数据。这种传输模式通过允许每个服务器控制自己的重建速度避免了任何一台服务器的压倒性。

为了提高网络和磁盘资源的利用率，传输本身在我们的模

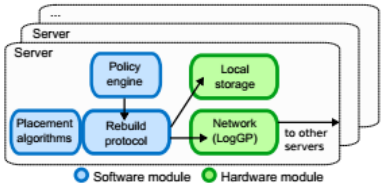

Fig. 4. Discrete event simulation components.

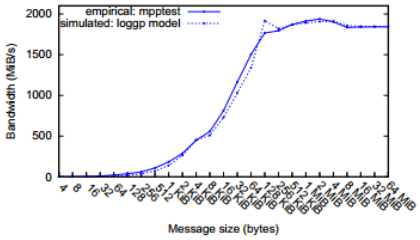

Fig. 5. Point-to-point bandwidth comparison between empirical and simulated performance on QDR InfiniBand network with MPI.

拟器中流水线化，就像在现代实现中一样。流水线的方法和参数在下文中有更详细的描述。除了将每个服务器上的内存消耗限制在 1 GiB 以缓冲输入数据，而不是限制流水线深度外，还有 1 GiB 用于缓冲输出数据。

我们的重建仿真包括分布式协调，基于拉的传输和数据流水线。它不模拟故障检测的辅助故障响应步骤或计算要修复的对象。仿真是在零时刻开始的，假设故障及其影响已经被评估过了，我们将注意力集中在故障恢复的数据传输部分。我们不会忽视计算开销，但是; 在第五节中分别进行评估。

3 模拟方法

离散事件模拟是评估大规模存储系统的有力工具。它可以观察分析模型无法捕捉到的微妙，时变和依赖于工作负载的行为。它还能够探索在现实系统中昂贵或不切实际的故障情景。事实上，如果模拟运行良好，可以用来执行整体揭示大量随机样本统计趋势的实验。

我们使用 CODES 工具包开发了一个分布式对象存储系统 2 的离散事件模拟，该工具包又建立在 ROSS 高性能并行离散事件模拟器上。存储系统模型被分解转化为关键硬件组件和软件组件的子模型，如图 4 所示。策略引擎负责管理整体服务器状态。重建组件在磁盘和网络资源上执行分布式数据传输协议。它使用对象放置算法组件来确定系统上副本的位置。模拟以网络消息和磁盘缓冲区粒度执行，并跟踪每个对象的状态在系统中。尽管我们使用 Ceph 的 CRUSH 算法来进行对象放置，但模拟器并没有模拟实际的 Ceph 文件系统。它是一个分布式对象重建协议的通用模型。

A. 网络模拟验证

我们在阿贡国家实验室选择了一个生产 Linux 集群，作为数据中心平台和互连的代表性例子。每个节点包含两个 2 GHz AMD Opteron 6128 处理器，64 GiB 主内存和一个单端口 Mellanox ConnectX 2 QDR InfiniBand NIC。节点通过与其他计算资源共享的四台 Mellanox IS-5600 交换机相互连接。

我们使用 LogGP 模型为 InfiniBand 网络建模通信成本 [27]。我们假设每个节点都有一个全双工网卡，在交换机中有独立的发送和接收队列和无限的缓存。我们的 LogGP 模型的参数是通过使用 netgauge 实用程序获得的。我们的仿真有两种方式偏离了传统的 LogGP 模型。首先，netgauge 假设开销参数（o）（表示在传输期间消耗的 CPU 时间）与现代网络上的网络结构传输成本重叠，因此我们不将 o 参数应用于通信时间计算。第二，我们利用 netgauge 通过在我们的模型的查找表中使用这些参数来独立计算一系列消息大小的 LogGP 参数。这种方法允许模型更准确地反映协议交叉点和其他结构特定的特征。

图 5 比较了使用我们的仿真框架对 Linux 群集上经验测量的点对点带宽（通过使用 mpptest 测量）与点对点性能的仿真。我们看到仿真性能与示例系统的性能趋势非常接近，包括 4 KiB 和 8 KiB 之间的明显协议交叉点，以及 4 MiB 和 8 MiB 之间的性能意外降幅。该模型的整体均方根误差为 58.128 MiB / s。

B. 磁盘模拟验证

我们假设每个存储服务器都连接到一个商品存储阵列（JBOD）。我们使用 DataON DNS-9470 产品中的 LSI SAS 9207-8i 控制器和 LSI SAS2x36 扩展器作为此类存储设备的一个示例。我们的示例 JBOD 配置了 10 个 Seagate Constellation ES3 3 GB SATA 驱动器，每个驱动器的峰值吞吐量为 175 MB / s，平均延迟时间为 4.16 ms，缓存为 128 MiB。使用具有 512 KiB 块大小的 RAID0 配置中的软件 RAID（mdraid）将这些磁盘合并为一个卷

我们的磁盘模型的设计是由两个要求驱动的。第一个要求是 ROSS 仿真框架的兼容性。ROSS 通过 Time Warp 以乐观模式运行时达到了最高性能，协议：每个模型实体独立和推测地处理自己的本地事件人口。如果它接收到一个时间戳相对于当前状态失序的事件，则它使用反向计算来“回滚”到一个连贯的时间点。流行的 DiskSim 模型与此方法不兼容，因为它不提供反转状态的机制。我们将在未来的工作中研究将此功能添加到 DiskSim 的可能性。我们的磁盘模型的第二个要求是基于真实的设备特征，例如寻道时间和带宽。这种方法通过改变模型参数来反映假设的硬件配置，从而实现“假设”探索，但是却排除了黑盒子模型的使用。我们选择了基于 Ruemmler 和 Wilkes 描述的分析技术来构建一个磁盘模型。

本研究中使用的输入参数是通过在 10 个磁盘上的 DataOn 测试系统上执行 fio 基准来收集的。我们测量了每个访问大小的 I / O 性能 60 秒，因为对于顺序读取，顺序写入，随机读取和随机写入访问模式中的每一个访问大小从 4 KiB 变化到 8 MiB。libaio 引擎在所有情况下都使用了 4 个 Iodepth 和直接 I / O。我们将模型的速率参数设置为顺序读取和顺序写入测量所实现的最大速率。开销参数设置为中值完成延迟减去 4K 请求大小的传输速率。查找参数被设置为顺序和随机延迟值之间的差值除以每个请求大小的 iodepth（4）。因此，读取和写入操作在我们的模型中使用不同的速率和开销值，而随机操作的查找时间通过基于访问大小的查找表来确定。我们将随机 I / O 操作定义为在上一操作的 512 个字节内没有开始的任何操作。Ruemmler 和 Wilkes 模型要求一个固定的寻找参数，但是我们无法使用这种方法实现一个好的模型拟合。我们相信原因是缓存行为，这并不反映在 Reummler 和 Wilkes 模型中。

图 6 将经验性测量的磁盘带宽（使用 fio 测量）与配置为连续发出四个并发 I / O 操作时仿真环境的结果进行比较。表 1 显示了每个模拟模式的总体均方根误差（也称为该模型的缺点数字 [34]）。我们在每种模式下的累计误差都小于 10%。

C. 流水线协议流水线技术是优化大量服务器数据传输的常用技术 [36]，[24]，[22]，因此在重建副本时对性能至关重要。我们通过将对象分割成更小的缓冲区，然后使用异步网络操作和多线程磁盘 I / O 传输这些缓冲区来在我们的模型中实现流水线。管道缓冲区的大小应该足够大，以便分摊启动成本，但是足够小以最大化并发性。图 7 显示了使用第

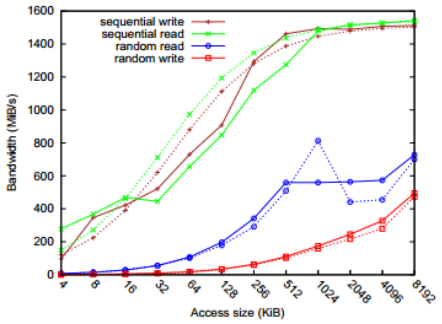

Fig. 6. Disk bandwidth comparison between `fio` benchmark (solid lines) and simulation results (dashed lines) as the access size is varied.

TABLE I
ROOT MEAN SQUARE ERROR OF DISK SIMULATION.

Mode	RMSE (MB/s)	RMSE (%)
random read	10.75	8.69
random write	11.95	3.84
sequential read	90.75	9.45
sequential write	13.32	1.36

III-A 和 III-B 节中获得的网络和磁盘参数在两台服务器之间模拟流水线传输单个 1 个 TiB 对象的结果。每个服务器都被配置为在接收/写入数据的同时，最多承担 1 吉比特的 RAM，以及 1 吉比特的 RAM 读取/发送数据。我们使用单独的内存缓冲池进行发送和接收，以确保服务器始终能够在重新构建自己的本地副本时取得进展，即使在服务于对等方的请求时也是如此。流水线传输使用 16 MiB 缓冲区大小达到 1.5 GiB / s 的近峰值速率; 两台服务器都受到存储吞吐量的瓶颈。我们还绘制了禁用流水线的传输带宽，并观察服务器无法在此配置中使其硬件资源饱和。基于这些结果，我们配置我们的存储系统模型以启用 16 MiB 缓冲区的流水线。请注意，对于 n = 2 个服务器对，峰值汇总传输带宽大致近似于点对点峰值带宽。此假设配置不会利用全双工网络功能，但会在每个存储设备上产生顺序流式访问模式。对于 1.5 GiB /s 的 512 个服务器对（总共 1024 个服务器），这可能导致总计 768 GiB / s 的速率。这个速率完全在大型 InfiniBand 交换机组合的能力范围之内。

4 CRUSH 案例研究

我们通过评估具有以下配置的假设对象存储系统的重建行为来开始我们的实验。通过使用 Ceph 0.94.3 中实现的 CRUSH 算法将对象映射到服务器。对象群体是从第 II-B 节所述的 1000 个基因组文件大小的直方图生成的。网络，磁盘和流水线参数被设置为反映硬件参数在第三部分描述和验证。

每个对象都是三维复制的。总体对象群体大小被缩放以产生至少 20 个 TiB 数据（当考虑到 3 路复制时为 60 个 TiB），并且每个服务器至少有 100 万个对象。CRUSH 配置了一个扁平的 CRUSH 映射（即，所有服务器属于一个具有相同权重的单个桶），使用稻草桶类型。此外，我们遵循在中推荐的安置组参数 Ceph 文档 [37]，使得存储系统中的放置组数量随着服务器数量而不是对象数量的增加而增加。表 II 显示了每个配置中使用的放置组数和对象数（不包括副本）。

仿真从一个随机选择的服务器的失败开始。然后，我们测量所有受影响的副本在存活的服务器上重建的时间。我们不模拟任何辅助程序，如故障检测或同步; 这个模型专注于读取，传输和写入复制数据，这是我们问题领域中最耗时的故障恢复方面。我们将传输的数据总量除以经过的时间产生一个总的重建率。图 8 显示了从 4 到 1024 的一系列系统大小的聚合重建率。两个轴都使用对数标度。我们通过选择随机

TABLE II
TOTAL NUMBER OF PLACEMENT GROUPS AND OBJECTS IN EACH CONFIGURATION

No. of Servers	Placement Groups	Objects
4	128	1,373,187
8	512	2,745,833
16	4096	5,494,324
32	4096	10,990,294
64	4096	21,984,426
128	8192	43,971,596
256	16384	87,949,846
512	32768	175,897,623
1024	65536	351,765,871

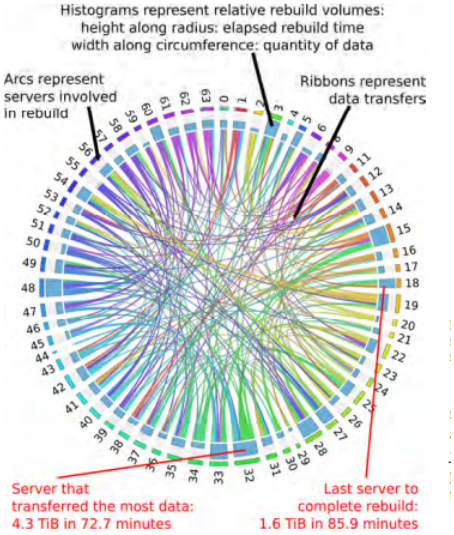

Fig. 9. Circos diagram of data transfers between servers in the slowest 64-server configuration from Figure 8.

服务器来为每个配置收集 15 个随机样本以失败。唯一的例外是 4 个和 8 个服务器数据点，其中我们分别收集 4 个和 8 个样本，因为没有 15 个不同的服务器来选择故障。盒须图显示了每组样本的最小值，第一四分位数，中位数，第四四分位数和最大值。

我们还通过从 4 服务器中值推断理想的线性缩放来对图进行注释; 这显示为绿色。对于多达 16 台服务器，我们看到模拟的重建率合理地跟踪理想的重建率。随着系统规模的增加，它逐渐减少，但是，最终达到了重建率为 49.4GiB/s 的 1,024 台服务器。这个总的重建速度是一个数量级比预计的理想率。

我们选择了最慢的 64 个服务器配置进行进一步调查：这个例子清楚地显示了次优的性能，但是为了清晰的可视化，它足够小。我们通过模拟来记录确切的数据量在服务器对之间转移，并使用 Circos 软件包绘制这些值 [38]。结果如图 9 所示。服务器由外围的颜色编码矩形表示，每个服务器都标有服务器索引编号从 0 到 63. 服务器 10 被省略，因为它是在这个例子中失败的服务器。一个小的直方图也与每个服务器相关联：它的宽度表示该服务器传输（包括发送和接收）数据的相对量，而其高度则表示服务器完成重建过程的时间量。连接服务器对的色带表示这些服务器之间的数据流。

这个数字说明了一个关键的现象：在存活的服务器之间重建的负载是不平衡的。底部的注释突出显示了一个额外的非直觉性的发现：传输最多全部数据的服务器不是花费最长时间完成其重建过程的服务器。服务器 32 与比服务器 18 更多的对等体交换数据，允许它尽可能快地完成其流量并完成其重建过程，尽管传送更多的数据。这个结果表明热点或数据依赖性的存在阻碍了一些服务器比其他更多。虽然超出本文范围的高级调度策略可以帮助减轻这种障碍，但是在这种规模下使用时，底层的不平衡是布局算法的一个不可避免的缺

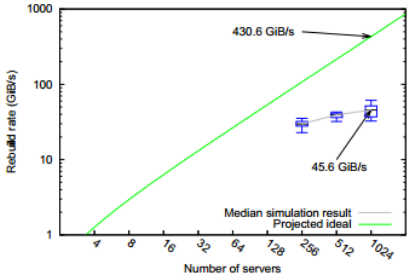

Fig. 11. Simulated aggregate rebuild rate following a single server failure using a CRUSH configuration with a hierarchical cluster map.

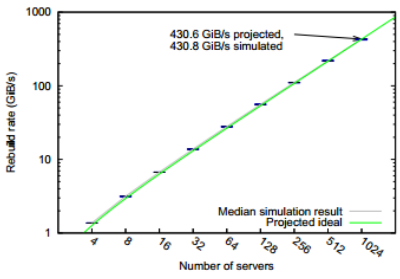

Fig. 12. Simulated aggregate rebuild rate following a single server failure by using an example CRUSH configuration with no placement groups.

点。

图 10 通过显示按照计数排序的 63 个存活服务器的每一个上生成的新副本的数量的直方图来调查不平衡的根本原因。新生成的副本不是均匀分布的; 其实这个例子显示了一个阶梯式模式，最活跃的服务器接收超过 35,000 个对象，最不活跃的服务器根本不接收任何对象。我们使用右侧的轴来覆盖每个服务器的新放置组的数量图加入了由于故障的结果。每服务器放置组计数的范围从 0 到 7 不等，但与每个服务器对象计数的趋势明显匹配。

从表 2 中可以看出，这个 64 服务器系统正在使用 4,096 个放置组。通过 3 路复制，每台服务器因此平均参与 192 个组。这个具体例子（服务器 10）中的失败的服务器参与 190 个放置组。当服务器 10 出现故障时，这 190 个放置组中的每一个必须精确地提升一个新的服务器以补偿服务器 10 的损失。因为 CRUSH 算法是伪随机的，但是这些 190 个副本目标不能保证在 63 个存活服务器上均匀分布。实际上，一个服务器（服务器 28）被添加到 7 个放置组，而另外四个服务器（服务器 7,9 和 55）没有被添加到任何新的放置组。

从这个分析中可以明显看出，微妙的高层次的安置政策，比如在 Ceph 中使用安置组，可以通过限制复制品来降低恢复时间以意想不到的方式进行分解。在这种情况下，集体重建的表现受到限制，在一定程度上不明显。放置策略过于粗糙，以利用更大的可用聚合带宽系统。

A. 分层 CRUSH 地图

CRUSH 的一个显着特征是它能够根据其物理布局来构建组织存储目标的分层地图。一个拥有数百或数千台服务器的生产存储系统可能会利用这种功能，而不是将所有的服务器放在一个单一的扁平桶中。我们使用层次聚类地图重新审视了图 8 中的三个最大的配置调查这种配置如何影响放置和重建行为。我们假设服务器的组织结构如下：每个机架 16 个服务器，每行 8 个机架，以及 2,4 或 8 行，具体取决于是否有 256,512 或 1,024 系统中的所有服务器。然后，我们构建了 CRUSH 布局规则，强制每个对象的副本必须驻留在不同的机架上。

图 11 显示了在这种配置下重复仿真的结果。1024 个服务器的中间总重建率与平桶配置相比略有下降（下降到 45.6 GiB / s，而不是 49.4 GiB / s）。机架感知放置规则通过限制有效副本集合排列的数量来略微减少相对于扁平桶形拓扑的解密，以便防止预期的相关故障模式。为了清楚起见，我们将分析重点放在平面拓扑上。

5 提高重建效率

在本节中，我们将重新审视第四部分的模拟，以评估提高总体重建率的策略。尽管我们使用 CRUSH 布局算法作为我们的出发点，但是我们的目标并不是规定 Ceph 特定的调整参数，而是探索如何构建一般的对象布局算法来改善重建时间和 MTTDL。回想第三节，我们的模拟不能模拟 Ceph 文件系统，因此不用一定嫩捕捉到推荐的 Ceph 参数变化的后果。

A. 战略：取消安置小组

改进 CRUSH 算法中的分组的一个简单可能的策略是简单地消除放置组，并且使用 CRUSH 算法独立地计算每个对象的放置。图 12 显示了使用这种方法可以实现的总重建率。如图 8 所示，我们绘制了与模拟比率一起的理想比例。在这种情况下，我们发现在所有测试的规模上，吞吐量与预期的理想值紧密匹配，因为在幸存的服务器之间重建流量的分配非常均匀。我们观察到分布式重建协议具有接近理想的可扩展性当使用对象粒度副本放置策略时。这个粒度水平也开辟了更细粒度的调度或优先重构的可能性，以进一步改善 MTTDL 。请注意，假设去除安置组可能会对整个 Ceph 存储系统产生重大影响。放置组构造用于故障检测，写指针日志和对等中的各种用途。然而，分析这些 Ceph 特有的设计问题已经超出了本文的范围。我们将分析限制在对象存储系统中更普遍的数据布局问题上。

但是，对象粒度化的布局有一个缺点。我们的模拟不会模拟对象布局计算的 CPU 成本，只是假设新的复制位置已知的数据移动成本。稻草桶算法本身在计算上是昂贵的，它需要每个对象的 O（n）计算，其中 n 是系统中服务器的数量。回顾第 II-A 节，如果没有首先将该 ID 与系统中的每个服务器的 ID 进行散列，则无法选择给定对象 ID 的目标服务器。这种情况如图 13 所示，其中显示了随着服务器数量的增加，CRUSH 吸管桶中的布置计算率。这个测量是在装有两个 2.4 GHz Intel Haswell E5-2620 v3 处理器和一个随机生成的 Linux 节点上进行的对象 ID 值的集合。CRUSH 计算可以以每秒将近 100 万个对象的速度执行，而桶中有 4 个服务器。速度下降到每秒 7 千个对象，有 1024 个服务器。这对于具有数十亿个对象的系统中的对象粒度放置而言是非常昂贵的，特别是当调用需要大容量放置计算的过程时，例如重建，文件系统一致性检查和数据清理。

在某些情况下，这种计算可能会与其他活动重叠，但有效的批量计算可以为这些操作的执行提供更多的灵活性。B. 策略：使用一致的散列作为桶算法 CRUSH 算法还提供统一的，列表和树桶类型作为稻草桶类型的替代，但是这些算法需要不受影响的多余的重新洗牌对象添加或删除服务器时，这些属性使其不适用于大型存储系统。一个理想的 CRUSH 桶算法将避免多余的复制品移动，同时在复制品之间达到平衡分解能力和计算效率。在以前的工作中，我们证明了多个一致的哈希算法可以实现这个目标 [18]，其中最直接的是一个具有欧几里得距离的一维环度量，与图 1 中所示的类似，但每个物理节点的虚拟节点比例很高。如果虚拟节点 ID 值是伪随机生成的，则它们可以显着增加环上副本排序的排列次数。

这种基于环的一致性散列算法可以适用于如下的 CRUSH 桶算法。当 CRUSH 桶被初始化时，N×V 伪随机虚拟 ID 通过用从 0 到 V 的值序列对每个物理服务器号（或 CRUSH 术语中的项）进行散列来生成。生成的虚拟 ID（以及映射回其

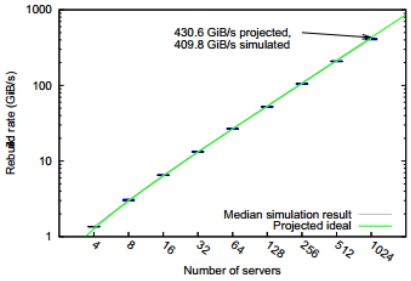

Fig. 14. Simulated aggregate rebuild rate following a single server failure by using the CRUSH vring bucket type and no placement groups.

原始的底层存储桶项目）是放置在一个数组中，并按虚拟 ID 值排序。如果存储桶已扩展，则可以生成新的虚拟 ID 并将其添加到阵列。每个虚拟 ID 元素在我们当前的实现中消耗额外的 32 个字节的内存（虚拟 ID，底层物理 ID，数组索引和总阵列大小各一个整数），尽管这可以针对更紧凑的内存使用进行优化。

CRUSH 查找函数通过排序后的虚拟 ID 数组执行 O（log n）二进制搜索，以找到给定对象 ID 的数字上最接近的虚拟 ID。CRUSH 权重可以通过缩放为给定项目生成的虚拟 ID 的数量来改变其虚拟节点在环上出现的频率，由此改变在查找例程中将被选择的概率

我们使用新的 CRUSH 桶类型（称为 vring bucket 类型）3 来实现此算法，并将 V 的默认值设置为 1024（即每个项目 1024 个虚拟 ID）。图 14 显示了使用这种新的桶算法重复图 12 的聚合重建速率测量的结果。在 1,024 个服务器中，它是理想的缩放目标的 5％以内。这比使用稻草桶的速度要慢一些按照每个对象放置（即，没有放置组），但是在图 15 中，我们看到它以较低的计算开销实现了这个重建率。在 1024 个服务器上，它可以计算每秒接近 597000 个对象的对象放置，比稻草桶算法提高了 82 倍。

这种计算效率的提高将减少整体延迟以及重新计算故障后对象的位置所需的时间。因此，我们发现我们在一个新的 CRUSH 桶类型（称为 vring bucket 类型）3 中实现了这个算法，并将 V 的默认值设置为 1,024（即每个项目有 1024 个虚拟 ID）。图 14 显示了使用这种新的桶算法重复图 12 的聚合重建速率测量的结果。在 1,024 个服务器中，它是理想的缩放目标的 5％以内。这比使用稻草桶的速度要慢一些按照每个对象放置（即，没有放置组），但是在图 15 中，我们看到它以较低的计算开销实现了这个重建率。在 1024 个服务器上，它可以计算每秒接近 597000 个对象的对象放置，比稻草桶算法提高了 82 倍。

这种计算效率的提高将减少整体延迟以及重新计算故障后对象的位置所需的时间。因此，我们发现一致的散列算法可以在 CRUSH 中使用，从而显着降低 CPU 成本，而不会影响分组。

C. 数据人口的影响

我们在 II-B 节中观察到，不同的数据集可以表现出实质上不同的文件和对象大小的分布。我们用不同的对象群体重复了图 14 所示的实验，以说明这会如何影响实验评估。新的对象总体是由 Mira 文件系统直方图的加权统计抽样生成的。然后将每个文件分解成多达 N = 3 个对象的分条对象集（而不是分块对象集），每个对象集又分别使用 3 路复制。对象的大小是根据循环条带策略设置的，具有 4 个 MiB 条带单元，可以在 PVFS [39] 或 Lustre [2] 中找到。并行文件系统。

图 2 已经根据文件大小的分布比较了 1000 个 Genomes 和 Mira 数据集。当我们比较底层对象大小的分布时，差异更加明显，如图 16 所示。米拉数据集包括大量的小文件，这些小文件依次映射到合成对象群体中的小对象。米拉人口中最大的对象也比 1000 基因组人口中最大的对象大得多。在这个例子中，最大的单个对象包含 186GiB 的数据。原因是 Hadoop 风格的组块规定的对象大小限制为 64MiB，而循环条带化对象布局对底层对象大小没有这样的限制。随着文件被追加，对象继续无限增长。

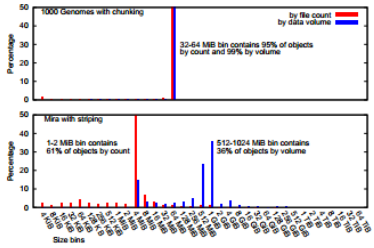

Fig. 16. Comparison of object sizes in synthetic data sets generated by weighted histogram sampling.

图 17 显示了通过比较 1000 个基因组的重建率，对象种群中的这种差异如何转化为聚集重建性能数据集和 Mira 数据集。缩放趋势是相似的，但基于带条纹 HPC 文件系统的对象群体实现的中间总重建率仅为 189.8 GiB / s，还不到同样实验 1000 个基因组对象群体的同一实验的总重建率的一半。有两个原因。首先是小物体的数量越多

D. 评估 PDES 方法

本研究中的所有仿真都是通过在 CODES 仿真工具包中使用乐观并行离散仿真来进行的。其中最复杂的是使用图 17 中的条带化 Mira 群体示例的 1,024 个服务器模拟。该组中的第一个样本跟踪了大约 39 亿个副本（60 个 PiB）的状态，其中 380 万个（61 个 TiB）被重建在仿真和消息级粒度建模。这项工作需要处理超过 2 亿个离散事件。我们使用 256 个 MPI 进程执行仿真，这些进程分布在配备 2.4 GHz Intel Haswell E5-2620 v3 处理器的 Linux 群集的 22 个节点上和一个 InfiniBand 网络。它在 30.2 秒内完成，产生了每秒 670 万事件的有效模拟速率。这样的表现不仅在实验问题上实现了快速的周转时间，而且使我们能够执行用随机故障排列的集合模拟，以便将总体趋势与异常值区分开来。我们发现，高保真并行离散事件模拟能够在规模上对故障情景进行难以处理的评估。

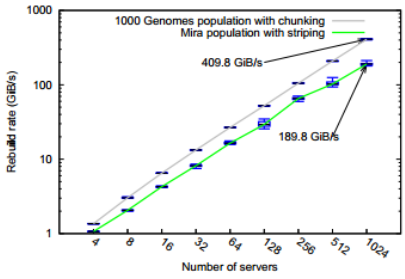

Fig. 17. Simulated aggregate rebuild rate following a single server failure using the CRUSH vring bucket type with two different data population examples.

在这项工作中评估的最大场景不仅对于现实世界的实验是不切实际的，而且在某些情况下也超过了顺序模拟的能力。如果上面突出显示的示例是串行执行的，则会耗尽模拟平台上任何单独节点上可用的 RAM 的 384 GiB，从而无法完成。

6 结论和未来工作

我们使用大规模复制对象存储重建协议的并行离散事件模拟来评估数据放置策略对弹性的影响。我们发现包括以下内

容：

1) 使用对象粒度副本放置时，分布式重建协议具有接近理想的可伸缩性；
2) 微妙的高级别放置策略（例如在 Ceph 中使用放置组）可以通过意外的方式限制副本分组来降低恢复时间；
3) 一致性散列算法可以在 CRUSH 中使用，以显著降低 CPU 成本，同时不会影响分组；
4) 数据总体特征可能会对分布式重建算法的效率产生重大影响；
5) 高保真并行离散时间模拟能够在规模上对故障情况进行难以处理的评估。

这些研究结果突出了未来存储系统可以采用的算法和评估课程，以提高弹性和性能。在未来的工作中，我们希望探索更复杂的故障模式。除了复制的数据之外，我们还想评估擦出编码数据的重构。本工作中描述协议模型也可以与工作负载模型，故障检测协议模型和其他组建大规模存储系统行为的整体观点。我们还希望利用 CODES 中提供的各种子模型来评估不同系统建构策略。