科学应用的瞬时突发缓冲文件系统

Teng Wang † Kathryn Mohror 3 Adam Moody 3 Kento Sato 3 Weikuan Yu †
†佛罗里达州立大学{twang, yuw}@cs.fsu.ed
u 3劳伦斯利弗莫尔国家实验室{凯瑟琳, moody20, sato5}@llnl.gov

Abstract-Burst buffer正在成为大型超级计算机上不可或缺的硬件资源，用于缓冲科学应用的突发I/O。然而，在一个批处理提交的作业内的应用程序有效地共享突发缓冲区并在不同的批处理作业之间循环利用，目前缺乏软件支持。此外，突发缓冲区需要应对来自数据密集型科学应用程序的各种具有挑战性的I/O模式。在这项研究中，我们设计了一个临时突发缓冲区文件系统(BurstFS)，它支持可扩展和有效地从突发缓冲区聚合I/O带宽，同时具有与批量提交作业相同的生命周期。BurstFS提供了多种技术，包括可扩展的元数据索引、同址I/O委托、服务器端读集群和流水线。通过广泛的调优和分析，我们已经验证了BurstFS已经实现了我们的设计目标，在并行写和读的聚合I/O带宽方面具有线性可伸缩性。

I. 介绍

随着科学和分析数据集的爆炸式增长，突发缓冲区已被规定为大规模高性能计算(HPC)系统中不可或缺的组成部分[2,3,6,9,10,11,13,24]。部署突发缓冲区主要有两种策略。一种策略是在本地将快速存储附加到每个计算节点，称为节点本地突发缓冲区。另一种是提供一个额外的可被计算节点远程共享的快速存储层，称为远程或共享突发缓冲器。虽然这两种策略都被用于当前和下一代系统[9,10,12]，但我们在这项工作中专注于节点本地突发缓冲策略。

突发缓冲区是一种强大的硬件资源，用于科学应用程序缓冲其突发I/O流量。然而，突发缓冲器的使用还没有得到充分的研究，也没有跨系统的突发缓冲器软件接口标准化。目前，用户可以自由地以一种特殊的方式探索突发缓冲区的使用。然而，领域科学家更愿意专注于他们的科学问题，而不是摆弄如何最好地使用突发缓冲区的复杂性。

一些研究已经探索了使用位置感知的分布式文件系统(例如，HDFS[37])来管理节点本地突发缓冲区[4,45,48]。在这样的文件系统中，每个进程将其主数据存储到本地突发缓冲区。因为计算进程可以与其数据共存，所以实现线性可扩展的聚合带宽是可行的[48,32]。

然而，突发缓冲区只对用户作业临时可用。用户作业在分配期间可以利用本地突发缓冲区，但当分配终止时，作业就失去了对突发缓冲区存储的访问。传统的文件系统，如HDFS[37]或Lustre[17,31]，通常被设计为无限期地保存数据，按HPC系统生命周期的顺序。它们利用长时间运行的守护进程进行I/O服务，这对于临时突发缓冲区的使用来说是不必要的。此外，这些文件系统的I/O服务的构建和清理会导致计算核心、存储和内存方面的资源浪费。因此，为了科学用户有效地使用突发缓冲区，开发用于标准化突发缓冲区使用的软件是至关重要的，这样它们就可以无缝地集成到领先超级计算机上的HPC工具库中。

HPC应用程序通常表现出两种主要的I/O模式:共享文件(N-1)和每个进程文件(N-N)[15](详见第II-A节)。对于节点本地突发缓冲区，使用N-N模式，应用程序可以通过让每个进程在本地写入/读取其文件来实现可扩展的带宽。节点本地突发缓冲区的困难在于N-1 I/O模式，在这种模式下，所有进程都写入共享文件的一部分。特别是，共享文件要求所有数据段的元数据在写入时被正确地构造、索引和收集，然后在任何进程能够定位其目标数据进行读访问之前，用全局布局来制定。虽然这个问题已经在持久化并行文件系统上进行了研究[15,47]，但高效地制定和服务共享文件的全局布局的问题仍然是跨突发缓冲区的临时文件系统的关键问题。

此外，来自科学应用的数据集通常是多维的。此类数据集通常以多维的特定顺序存储，但经常根据科学模拟或分析的性质从不同维度读取。通常，多维数据集中数据元素的写入顺序和读取顺序之间存在不兼容性，这通常会导致一个进程检索其所需的数据元素[39]的许多小型非连续读取操作(更多细节请参见第II-B节)。一个有效的节点本地突发缓冲文件系统还需要提供一种机制，使科学应用程序能够高效地读取多维数据集，而不需要许多昂贵的小型读取操作。

在这项研究中，我们设计了一种短暂的突发

SC16; Salt Lake City, Utah, USA; November 2016 978-1-4673-8815-3/16/$31.00 2016c IEEE

缓冲文件系统(BurstFS)，具有与批处理提交作业相同的临时生命周期。BurstFS将写入本地突发缓冲区的数据的元数据组织到分布式键值存储中。为了应对上述I/O模式带来的挑战，我们在BurstFS中设计了几种技术，包括可伸缩元数据索引、同址I/O委托以及服务器端读取集群和管道。我们使用了许多I/O内核和基准来评估BurstFS的性能，并通过调优和分析验证我们的设计选择。

总而言之，我们的研究做出了以下贡献。

- 我们设计并实现了一个突发缓冲区文件系统，以满足领导力超级计算机对有效利用突发缓冲区的需求。
- 我们在BurstFS中引入了几种机制，包括用于快速定位共享文件数据段的可扩展元数据索引，用于可扩展和可回收的I/O管理的共定位I/O委托，以及用于支持快速访问多维数据集的服务器端集群和管道。
- 我们通过广泛的I/O内核和基准测试来评估BurstFS的性能。我们的结果表明，BurstFS在并行写和读的聚合I/O带宽方面实现了线性可伸缩性。
- 据我们所知，BurstFS是第一个在同一作业中与一个或一批科学应用程序共存且短暂生命周期的文件系统。

II. 突发缓冲区的I/O模式的背景

在本节中，我们概述了突发缓冲区需要支持的典型I/O模式，以便于理解我们对BurstFS的设计目标。

A. 使用共享或每进程文件的检查点/重启I/O

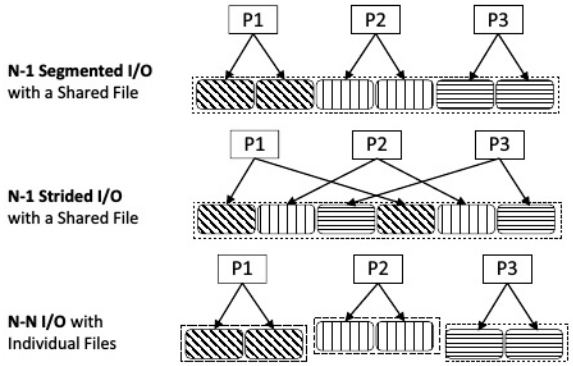


图1:检查点/重启I/O模式(改编自[15])。

检查点/重启是HPC应用程序中常用的容错机制。在运行期间，应用程序进程定期将其在内存中的状态保存在称为检查点的文件中，通常写入并行文件系统(PFS)。然后，在发生故障时，可以读取最近的检查点来重新启动作业。为简单起见，检查点操作通常是跨应用程序中的所有进程并发的，并在没有消息在传输时发生在程序同步点。在当前的HPC系统中，检查点可占总I/O流量的75%-80%[32]。虽然与今天的系统相比，关于检查点操作将如何在百亿亿次系统上发生变化的争论正在进行，但普遍的共识是，由于作业规模更大，每个检查点的数据大小将增加，而由于总体失败率增加，检查点之间的间隔将减少[18,29]。更大的文件大小和更短的检查点间隔将要求更快的存储带宽[25]数量级。

检查点/重启有两种主要的I/O模式，N-1和N-N模式，如图1所示。在N-N I/O中，每个进程向一个唯一的文件写入/读取数据。在N-1 I/O中，所有进程写入或读取单个共享文件。N-1 I/O可以进一步分为两种模式:N-1分段式和N-1跨步式。在N-1分段I/O中，每个进程访问一个不重叠的、连续的文件区域。在N-1步进I/O中，进程将它们的I/O相互交错。

在传统的并行文件系统[31,34,36,43]中，大文件分条分布在一个或多个存储服务器上。虽然条带化允许每个进程与多个存储服务器并行交互，但当多个进程并发访问同一个存储服务器时，它也会导致条带化开销[46]和I/O争用[15,41]。由于减少了争用和突发缓冲设备(例如NVRAM和SSD)的性能优势，使写/读本地化的分布式突发缓冲文件系统对这种检查点/重启工作负载非常有利。

B. 科学应用中的多维I/O访问

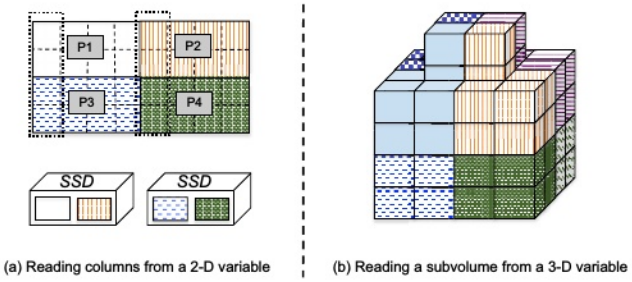


图2:多维变量的I/O访问模式。

HPC系统上另一个常见的I/O模式是科学应用中对多维数据变量的数据访问。虽然多维变量以一种特定的顺序写入，但它们通常以不同于写入顺序的顺序读取进行分析或可视化[27,39]。

图2(a)展示了一个二维变量上的读模式样本。这个变量最初被分解为四个块，作为四个数据块写入到两个ssd中。当这个变量被读回进行分析时，一个进程可能只需要这个变量的一个或多个列。然而，这两列在数据块中是不连续存储的。

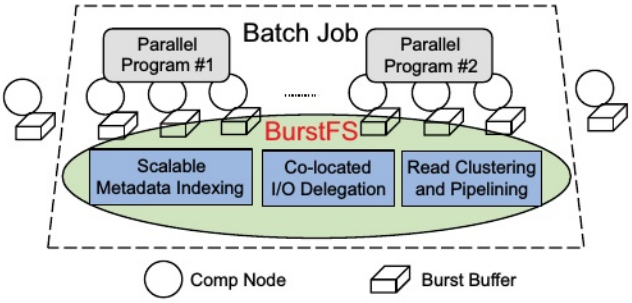


图3:BurstFS系统架构。

因此，这个过程需要向四个不同的数据块发出小的读取请求，以便检索其数据进行分析。图2(b)展示了一个类似但更复杂的场景，其中有一个三维变量。三维变量最初存储为八个不同的块，跨越突发缓冲区。一个过程可能只需要变量中间的一个子体积来进行分析。这个子卷必须从八个不同的块中收集，才能完成它的数据访问，从而导致许多小的读取操作。综上所述，一个用于突发缓冲区的高性能文件系统必须提供一种机制，在不需要许多小型读取操作的情况下，高效地读取多维数据集。

III. 临时突发缓冲文件系统

我们将突发缓冲文件系统(BurstFS)设计为一个短暂的文件系统，具有与HPC作业相同的生命周期。我们对BurstFS的总体目标是支持跨分布式、节点本地存储的可扩展I/O操作聚合，用于数据密集型模拟、分析、可视化和检查点/重启。BurstFS实例在批处理作业开始时启动，为作业中的所有应用程序提供数据服务，并在作业分配结束时终止。BurstFS的系统架构如图3所示。

当在HPC系统上为批处理作业分配一组计算节点时，将使用本地附加的突发缓冲区(可能由内存、SSD或其他快速存储设备组成)在这些节点上动态构建一个BurstFS实例。这些突发缓冲区使非常快速的日志结构本地写入成为可能;也就是说，所有进程都可以将它们的写入存储到本地日志中。接下来，在这些节点的一部分上启动的一个或多个并行程序可以利用BurstFS向突发缓冲区写入数据或从突发缓冲区读取数据。此外，BurstFS实例仅在批处理作业的生命周期内存在。所有已分配的资源和节点将在计划执行结束时被清理以便重用。这避免了对其他作业的事后干扰或对文件和存储系统操作的潜在不可预见的并发症。此外，同一作业分配中的并行程序(例如，在同一批处理脚本中启动的程序)可以在同一个BurstFS实例上共享数据和存储，这可以大大减少后端持久文件系统在这些程序之间共享数据的需求。

BurstFS用一个可配置的前缀挂载，并透明地拦截该前缀下的所有POSIX函数[40]。

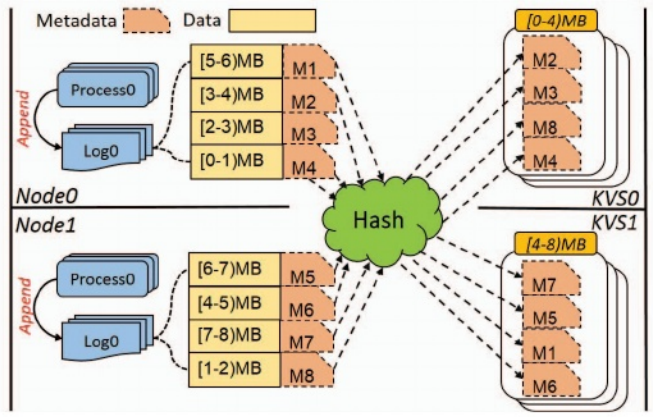


图4:BurstFS分布式键值存储示意图。

不同程序之间的数据共享可以通过使用相同的前缀挂载BurstFS来实现。在从最后一个程序卸载操作之后，所有BurstFS实例依次刷新它们的数据以实现数据持久化(如果请求的话)，清理它们的资源并退出。

为了支持第二节中讨论的具有挑战性的I/O模式，我们在BurstFS中设计了几种技术，包括可扩展的元数据索引、位于同一位置的I/O委托以及服务器端读取集群和管道，如图3所示。BurstFS将本地日志数据的元数据组织到分布式键值存储中。它支持可扩展的元数据索引，这样就可以快速生成数据的全局视图，以方便快速读取操作。它还提供了一个惰性同步方案，以减轻元数据更新的成本和频率。此外，BurstFS支持位于同一位置的I/O委托，用于可扩展和可回收的I/O管理。此外，我们还引入了一种称为服务器端读聚类和流水线的机制来提高读性能。我们将在本节的其余部分详细阐述这些技术。

A. 可扩展的元数据索引

正如第1节所讨论的，N-1 I/O模式的挑战之一是访问分散在所有节点上的段的元数据。当所有进程都在读取它们的数据，并且每个进程都需要从所有节点收集元数据时，这会导致一个巨大的可扩展性问题。

1)元数据的分布式键值存储:BurstFS使用元数据的分布式键值存储以及数据段的日志结构写来解决这个问题。它利用MDHIM[23]构建分布式键值存储，并提供额外的特性来有效地处理突发的读写操作。

图4显示了BurstFS的数据和元数据的组织。每个进程将其数据存储到本地突发缓冲区作为数据日志，这些日志被组织为数据段。新数据总是附加到数据日志中，即通过日志结构写入存储。使用这样的日志结构写入，一个进程的所有段都存储在一起，而不管它们相对于其他进程的数据的全局逻辑位置。

当并行程序中的进程创建一个全局共享文件时，为每个段生成一个键值对(例如M1或M2等)。一个键由文件ID(8字节哈希值)和段在共享文件中的逻辑偏移量组成。该值描述了段的实际位置，包括托管突发缓冲区、包含该段的日志(同一节点上多个进程可以有多个日志)、日志中的物理偏移量和长度。然后，所有段的键值对(KVP)可以为共享文件提供全局布局。所有KVP都一致地散列并分布在键值服务器(例如，KVS0、KVS1等)之间。在这样的组织中，元数据存储和服务分布在多个key-value服务器上。来自并行应用程序的许多进程可以快速检索元数据，并形成共享文件布局的全局视图。

*2)延迟同步:*在BurstFS中，我们还*开发了延迟同步*，以提供对突发写的有效支持。每个进程提供一个小内存池，用于保存写操作中的元数据KVP，并且在可配置间隔结束时，KVP会定期存储到分布式键值存储中。*fsync*操作可以强制显式同步。BurstFS利用MDHIM的批放操作在几次往返中一起传输这些KVP，从而最大限度地减少了单个放操作引起的延迟。在同步间隔期间，BurstFS在内存池中搜索连续的KVP，以便进行潜在的组合。一个组合的KVP可以跨越更大的范围。如图4所示，段[2-3)MB和[3-4)MB是连续的，并且映射到同一个服务器(KVS0)，因此它们的KVP被合并为一个KVP。当每个进程发出的许多数据段在逻辑上是连续的(例如图1中的N-1分段和N-N写入)时，延迟同步可以显著减少所需的KVP。

*3)并行范围查询:*在开始读操作之前，BurstFS必须首先查找分布式数据段的元数据。因此，它搜索偏移量在请求范围内的所有KVP，例如，[offset, offset+count]是pread中请求的范围。对于批处理读请求，BurstFS需要搜索批处理读请求所针对的所有KVP。为了为不同的读操作检索所请求的元数据条目，我们需要支持对键值存储的各种范围查询。但是，MDHIM不直接支持范围查询;客户端可以通过使用重复的游标类型操作迭代范围内的连续KVP来间接执行范围查询。客户端必须顺序地为一个范围服务器调用一个或多个游标操作，并且必须搜索多个范围服务器，直到找到所有KVP为止。所有游标操作到多个范围服务器的额外往返延迟会严重延迟读取操作。

为了减轻这种情况，我们为MDHIM客户机和服务器引入了并行扩展。在客户端，我们将传入的范围请求进行转换，并将其分解为多个小范围查询，以基于一致性哈希发送到每个服务器。与顺序游标操作相比，这个

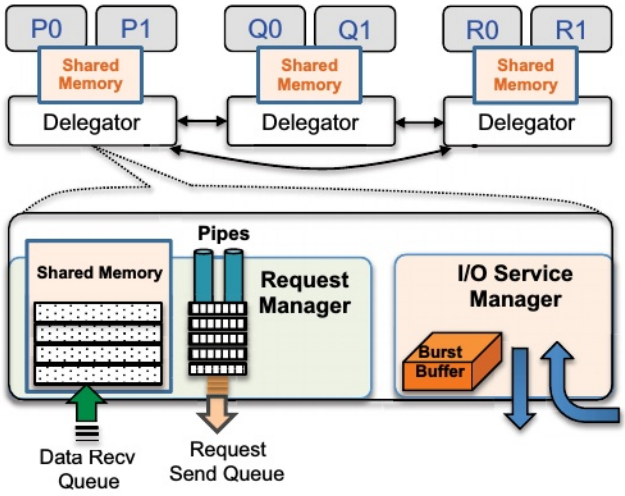


图5:在三个计算节点P、Q和R上共存的I/O委托示意图，每个节点有2个进程。

扩展可以将一个范围查询分解为多个小范围查询，每个范围服务器一个。然后将这些小查询并行发送到所有范围服务器以检索所有kvm。在服务器端，对于其范围内的小范围查询，通过键值存储中的单个顺序扫描检索该范围内的所有kvm。有了这种并行优化，任何查询组合都可以通过对所有服务器的并行范围查询和在每个键值服务器上的单个本地扫描操作来完成。

B. co -locate I/O Delegation

与BurstFS本地存储数据的写操作相比，BurstFS中的读操作可能需要将数据从远程突发缓冲区传输到发起读操作的进程。为了保证读操作的效率，我们需要为读操作支持快速和可扩展的数据传输。许多并行编程模型(如MPI[28]和PGAS[20])采用的一种常见方法是让每个进程对持久文件和存储服务守护进程进行读函数调用。由于BurstFS的生命周期有限，仅为单个作业的生命周期，因此BurstFS对I/O服务有特殊要求。一个实现选项可能是使用持久的I/O守护进程来支持BurstFS;然而，这将导致计算和内存资源的浪费。另一种实现选择可能是利用并行程序中从父进程派生出的简单I/O服务线程。然而，使用这种方法，服务线程只能服务于同一程序中的进程的I/O需求，而不能服务于批处理作业中的后续或并发程序。

在BurstFS中，我们通过一种称为*同址I/O委托的机制引入了可伸缩的读服务*。我们在每个节点上启动一个I/O代理进程，一个*委托者*。委托者与批处理作业中的应用程序解耦，并在所有计算节点上启动。委托代理共同为作业中的所有应用程序提供数据服务。

如图5所示，三个计算节点上的进程将把它们所有的I/O活动委托给同一节点上的委托者。每个委托者由两个主要组件组成:一个*请求管理*器和一个*I/O服务管理器*。这样，传统的用于I/O服务的客户机-服务器模型在所有委托者之间被转换为对等模型。在这种安排下，各个进程不再直接与I/O服务器通信，而是通过它们的I/O委派者。这导致了跨计算节点的网络通信通道和相关资源的总数的极大减少。每个委派器中的I/O服务管理器致力于服务来自对等委派器的传入读请求。I/O服务管理器利用机会合并请求，从本地存储中管道数据检索，并将数据传输回请求委托者(详见第三- c节)。

委托者的请求管理器由两个主要的数据结构组成:*请求发送队列*和*数据接收队列*，如图5所示。请求发送队列是一个循环列表，条目的数量可配置。当未满时，它通过命名管道接收来自所有客户端进程的读请求。请求根据目的地委派者进行排队。对同一个委托者的请求被链接在一起，这将多个请求合并为一个单一的网络消息。数据接收队列驻留在同一个节点上跨委托进程和客户进程构建的共享内存池中。对于每个I/O请求，在接收队列中创建一个未完成的请求条目。从远程委派者返回的数据直接存放在共享内存池中，并在接收队列中搜索匹配的未完成请求条目。当找到匹配项时，未完成的请求被标记为完成。通过管道发送一个额外的确认，通知客户端进程使用数据。

请求管理器监视共享内存池的使用水平。当它高于一个可配置的阈值(默认75%)时，委托者(1)通知进程消耗其数据的紧急需求，(2)限制向远程委托者注入请求。请求管理器还根据发送队列中读取请求的接收数据监视入口带宽。当入口带宽饱和时，请求管理器创建额外的网络通信通道来发送请求和接收数据。

C. 服务器端读取聚类和管道

如第II-B节所述，对于多维变量，进程可以对每个数据日志中的零散数据段发出许多小的、不连续的读请求。各种I/O库和工具为这种非连续读访问提供了特殊的支持。例如，POSIX lio_listio允许批量传输读请求;Or-angeFS支持批量读请求。虽然能够将小请求组合成一个列表或一个大请求，但这些技术主要从客户端工作，并依赖于磁盘调度器等底层存储系统来预取或合并请求以实现快速数据检索。然而，

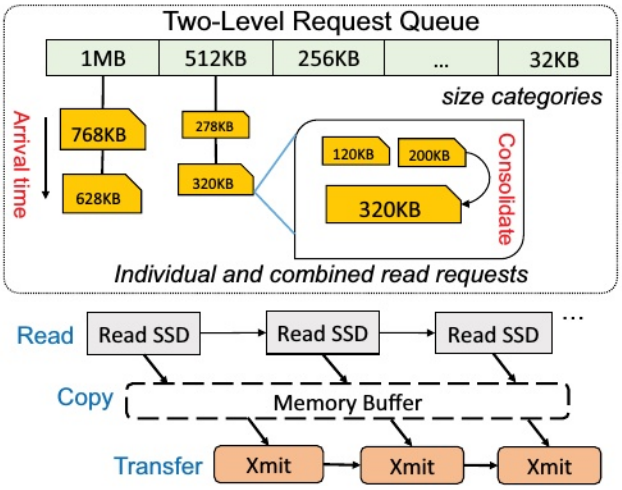


图6:服务器端读取聚类和流水线。

目前仍然缺乏能够全局优化这些来自所有进程的批处理读请求的分布式文件系统。

作为批处理作业中的临时文件系统，BurstFS通过委托直接管理科学应用程序对数据集的访问。因此，除了利用现有的客户端批处理读取技术外，BurstFS还可以利用它在服务器端读取请求的可见性(通过I/O服务管理器)来进一步提高性能。为此，我们在I/O服务管理器中引入了一种称为*服务器端读集群和管道(SSCP)*的机制，以提高BurstFS的读性能。

SSCP解决了几个并发的，有时是相互冲突的目标:(1)需要检测读请求之间的空间局域性，并将它们组合起来进行大型连续读。(2)以及为执行进度尽快服务按需读请求的需求。如图6所示，SSCP提供了两个关键组件来实现这些目标，一个用于读取集群的*两级请求队列*和一个用于快速数据移动的三阶段管道。

在两级请求队列中，SSCP首先创建请求大小的几个类别，从32KB到1MB不等(见图6)。传入的请求将单独插入到适当的大小类别中，或者如果与其他请求相邻，则与现有的连续请求结合，然后插入到合适的大小类别中。如图所示，两个连续的120KB和200KB的请求由服务管理器合并。在每个大小类别中，所有的请求都根据它们的到达时间排队。合并请求将使用其最老成员的到达时间。为了获得最佳的调度效率，请求规模最大的类别优先提供服务。在同一类别内，最早的请求将首先得到服务。BurstFS对每个类别的等待时间强制一个阈值(默认为5ms)。如果有任何类别的服务时间超过这个阈值，BurstFS将从该类别中选择最老的读请求进行服务，并重置该类别的等待时间。

I/O服务管理器创建一个内存池来tem-

暂存传出的数据。这有助于重新安排网络传输的数据段，并允许制定一个管道。图6显示了三个阶段的数据移动管道:读取、复制和传输。在读取阶段，I/O服务管理器根据上述调度策略从请求列表中挑选一个请求，将请求的数据从本地突发缓冲区读取到内存缓冲区中的插槽中。在复制阶段，将内存缓冲区中的数据准备为远程委派者的外发回复，然后从内存缓冲区复制到网络数据包中。内存缓冲区内的数据可能需要被分成多个回复给不同的远程委托者。然后，I/O服务管理器创建多个网络回复，为每个委托者创建一个。在传输阶段，I/O服务管理器可以将同一个远程委托的一个或多个网络应答打包成一个网络消息(最大1MB)，并将其传输(图6中的Xmit)给委托。

IV. 实验评价

A. 试验台

我们的实验是在劳伦斯利弗莫尔国家实验室(LLNL)的催化剂簇[3]上进行的，该簇由384个节点组成。每个节点配备2个12核Intel Xeon E5-2695v2处理器、128 GB DRAM和一个由PCIe ssd组成的800 GB突发缓冲区。

配置:我们重点比较BurstFS与两个当代文件系统:OrangeFS 2.8.8和并行日志结构文件系统2.5 (PLFS[15])。作为一种典型的并行文件系统(PFS)，OrangeFS在多个存储服务器上分配每个文件，以实现具有高聚合带宽的并行I/O。在我们的实验中，我们在分配给作业的所有计算节点上建立OrangeFS服务器实例，以管理所有节点本地ssd。PLFS旨在通过以日志结构的方式将随机、分散、N-1的写入转换为顺序的N-N写入，从而加速N-1写入。每个进程写入的数据作为日志文件存储在后端PFS中。在我们的实验中，我们使用OrangeFS(在节点本地ssd上)作为PLFS的后端PFS。我们使用PLFS的MPI接口进行读写。

从2.0版开始，PLFS支持突发缓冲区。在支持突发缓冲区的PLFS中(在本文的其余部分中称为“PLFS突发缓冲区”)，进程将其*金属链接*存储在后端PFS上，而不是将日志文件写入后端PFS，这些金属链接指向其日志文件在突发缓冲区中的实际位置。这允许每个进程将其日志文件写入突发缓冲区，而不是后端PFS。在我们的实验中，我们让每个进程写入其节点本地SSD，并将位置记录在存储在中心范围Lustre并行文件系统上的metalink中。这种配置可以提供可扩展的写入带宽。为了从PLFS突发缓冲区读取数据，每个节点本地SSD必须作为全局文件系统(例如NFS)挂载在所有其他计算节点上，这需要系统管理员的支持。BurstFS的主要目标是从用户空间完全控制它，包括挂载文件系统。因此，由于建立使用PLFS突发缓冲区进行读操作的交叉挂载环境需要管理员的干预，我们只评估了PLFS突发缓冲区的写可伸缩性，并将此结果包含在Section IV-B中。

基准:我们采用了显示三种检查点/重启I/O模式的微基准(见图II-A)。请注意，N-1跨行模式是II-B节中描述的二维科学I/O。

为了评估BurstFS支持科学应用的潜力，我们使用从MPI-Tile-IO[33]和BTIO[44]中提取的I/O工作负载来评估BurstFS。MPI-Tile-IO是一种广泛采用的基准，用于模拟可视化和数值应用中存在的工作负载。二维数据集被划分为多个tile，每个process渲染一个tile内的像素。BTIO是由美国宇航局高级超级计算部门开发的，它将一个三维数组划分为平方数的进程，每个进程处理多个笛卡尔子集。在这两个工作负载中，所有进程首先将数据写入共享文件，然后再将数据读回内存中。为了评估对多个应用程序的批处理作业的支持，我们采用了交错或随机(IOR)基准测试[26]来读取同一作业中Tile-IO和BTIO程序提供的数据。

B. 整体的写/读性能

我们首先评估BurstFS的总体写/读性能。在这个实验中，在每个节点上放置了16个进程，每个进程按照N-1跨步、N-1分段或N-N模式写入64MB数据。在每个进程写完所有数据后，我们使用fsync强制将所有写操作同步到节点本地SSD。我们将OrangeFS上的条带大小设置为1MB，并将传输大小固定为1MB以与条带大小保持一致，并且每个文件在OrangeFS中的所有节点上都是条带的。此配置为OrangeFS提供了优于其他调优选项的最佳读/写带宽(例如，默认条带大小为64KB)。

图7比较了PLFS突发缓冲器(PLFS- bb)、PLFS和OrangeFS的写带宽。在所有三种写模式中，BurstFS和PLFS突发缓冲区都随进程数线性缩放。这是因为两个系统中的进程都在本地写，每个节点本地SSD的写带宽已经饱和。虽然我们也观察到OrangeFS和PLFS的线性可扩展性，但它们的带宽增长速度要慢得多。这是因为PLFS和OrangeFS在多个节点上对它们的文件进行分条处理，当不同的进程向同一个远程节点写入数据时，由于争用，这会导致带宽降低。平均而言，对于N-1分段、N-1跨步和N-N模式，BurstFS的性能分别是OrangeFS的3.5倍、2.7倍和1.3倍。三种模式的性能分别是PLFS的1.6倍、1.6倍和1.5倍。

我们观察到，对于所有三种模式，PLFS最初在小进程计数(16和32进程)下提供的带宽高于BurstFS。经过进一步调查，我们发现这是因为，在内部，PLFS将N-1写入转换为N-N写入。但是，当调用fsync强制将这些N-N文件写入PLFS的后端文件系统时(即:

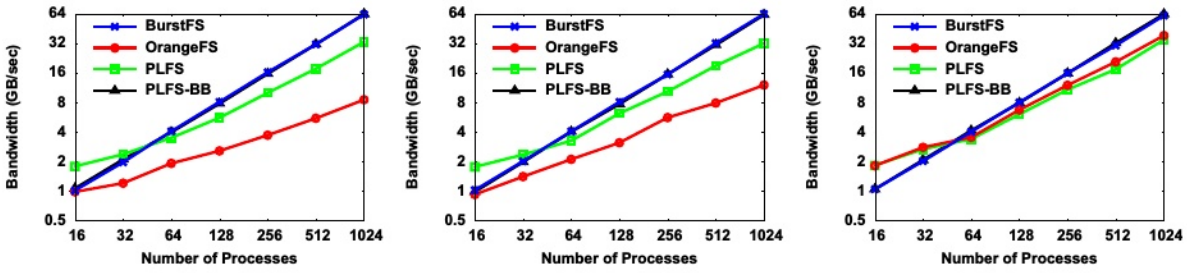


图7:不同写入模式下的BurstFS与PLFS和OrangeFS的比较。

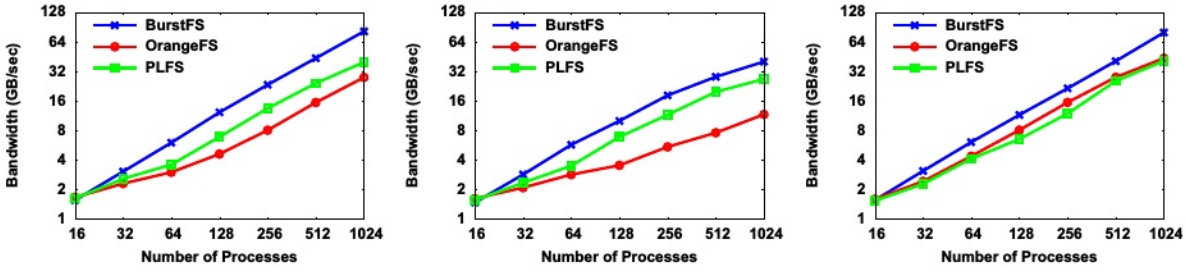


图8:不同读取模式下的BurstFS与PLFS和OrangeFS的比较。

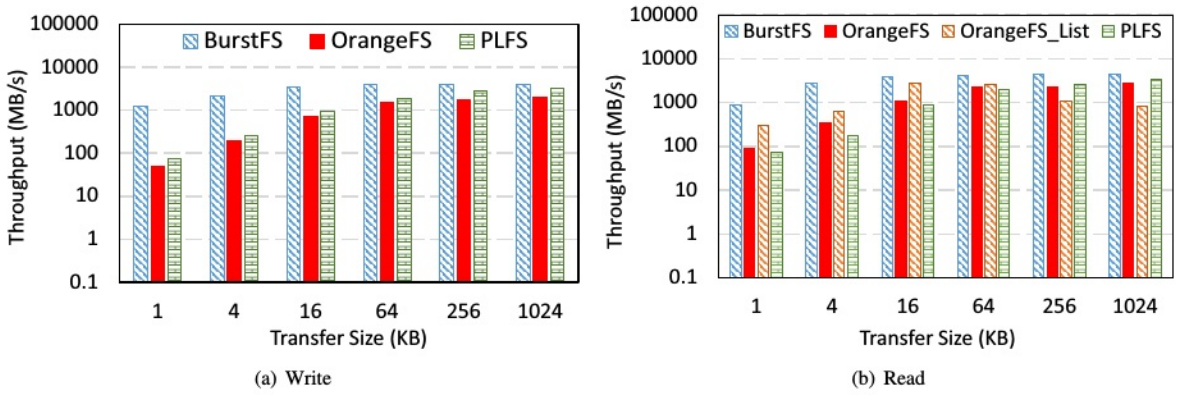


图9:不同传输规模下的BurstFS与PLFS和OrangeFS的比较。

OrangeFS)， OrangeFS在fsync返回之前不会将文件完全刷新到ssd上。测量到的带宽甚至比本地文件系统上SSD的总带宽还要高。

图8比较了BurstFS与or - angfs和PLFS的读带宽。每个进程在N-1步进、N-1分段和N-N模式下读取64MB数据。对于N-1次跨步读取，我们首先使用N-1次分段写入创建一个共享文件，然后使用N-1次跨步读取读取所有数据。通过这种方式，每个进程需要从多个日志中读取数据，如第II-B节所述。为了将这种模式下的不连续读请求聚类，我们使用POSIX lio_listio将读请求批量传输到BurstFS。在OrangeFS的情况下，当我们启用它的列表I/O操作时，我们观察到带宽比没有列表I/O操作的配置低两倍。这是因为OrangeFS列表I/O对大的读操作没有好处。因此，对于这个实验，我们只报告N-1跨步模式在OrangeFS中没有列表I/O支持的性能。

从图8(a)中可以看出，对于BurstFS, N-1分段读取的带宽随着进程数呈线性增长，因为每个进程直接从其本地节点读取所有数据。相比之下，PLFS和OrangeFS都需要从远程节点读取数据，从而失去了局部性的好处。另一方面，图8(b)中N-1步进读取的带宽在BurstFS下的增长速度要比分段读取慢得多。这是因为跨越读取模式由于远程burst的all-to-all读取而导致更高的争用

缓冲区。具有N-1步进读取的BurstFS仍然具有更好的可伸缩性，并且优于OrangeFS和PLFS。这是因为BurstFS委托不是单独为每个请求提供服务，而是将来自众多进程的读请求聚集在一起，并通过一个三阶段读管道为它们提供服务。平均而言，对于N-1分段、N-1跨步和N-N模式，BurstFS的性能分别是OrangeFS的2.2倍、2.5倍和1.4倍。对于三种模式，它的性能分别是PLFS的1.6倍、1.4倍和1.6倍。

C. 不同传输大小的性能影响

图9显示了传输大小对BurstFS带宽的影响。我们将重点放在N-1步进I/O上，因为它是一种具有挑战性的I/O模式。与第IV-B节中的实验类似，对于BurstFS跨行读操作，我们首先使用N-1分段写创建一个共享文件，然后使用N-1跨行模式回读数据。这样，BurstFS就不会从本地读取中获益。

图9(a)中的结果展示了64个进程写入共享文件时，传输大小对写入带宽的影响。通过让每个进程在本地写数据，BurstFS优于OrangeFS和PLFS，并且在较小的传输大小下提供了出色的性能改进，例如，在1KB时，与OrangeFS和PLFS相比，性能分别提高了24.4倍和16.7倍。这是因为PLFS和OrangeFS都要承受竞争写入和重复数据传输到共享远程ssd的成本。

图9(b)显示了传输大小对读取带宽的影响。对于小的读请求，OrangeFS提供了列表I/O支持，这样就可以将一个读请求列表合并到一个函数调用中。这种类型的读取操作的结果如图9(b)所示为OrangeFS列表。从图中可以看出，虽然OrangeFS List增强了小读的性能，但仍然低于BurstFS。这是因为BurstFS中的服务器端集群和流水线的额外好处。总体而言，与OrangeFS、or - angfs List和PLFS相比，BurstFS的性能分别提高了10.2倍、3倍和12.3倍。

D. 元数据性能分析

如第III-A1节所述，BurstFS在分布式键值存储上分布全局元数据索引。在文件打开过程中，PLFS中的每个进程都需要通过读取和组合其他进程的元数据来构建共享文件的全局视图。在这一步之后，所有的查找操作都在本地进行。为了评估我们的设计的好处，我们比较了BurstFS的元数据查找时间与PLFS的元数据查找时间(即PLFS在文件打开期间用于索引构建的总时间和读取期间用于本地查找的总时间)，以及MDHIM函数的原始查找时间。我们使用MDHIM中的游标和批处理get函数来检查查找性能。每个游标操作都会触发每个键值对的往返传输，查找一个范围可以调用多个游标操作，如第III-A3节所述。总的查找时间明显高于其他情况。例如，图10(a)中的4KBca se需要81秒。所以我们在图中省略了游标操作的查找时间。

图10(a)比较了BurstFS、PLFS和MDHIM批get(记为MDHIM)的查找时间。在这三种情况下，我们启动了32个进程，每个都是为了查找在N-1步幅模式下写入的64MB数据的位置。总的数据量是传输大小和段数的乘积。因此，较小的传输大小将导致更多的段，因此更多的索引。从图中我们可以看到，随着转移大小的增加，所有案例的查找时间都在下降。这是因为查找元数据的次数减少了。BurstFS的查找时间明显快于PLFS。这验证了BurstFS中的可伸缩元数据索引技术可以快速地为共享文件建立元数据的全局视图。相比之下，PLFS中的每个进程都必须加载写过程中生成的所有索引，并为读构建全局索引，这种全对全的加载占据了查找时间。BurstFS还优于MDHIM，因为它支持并行范围查询(参见第III-A3节)，因此在范围服务器上只进行一次顺序扫描就可以最大限度地减少读取操作的数量。平均而言，与PLFS和MDHIM相比，BurstFS分别减少了77%和58%的查找时间。

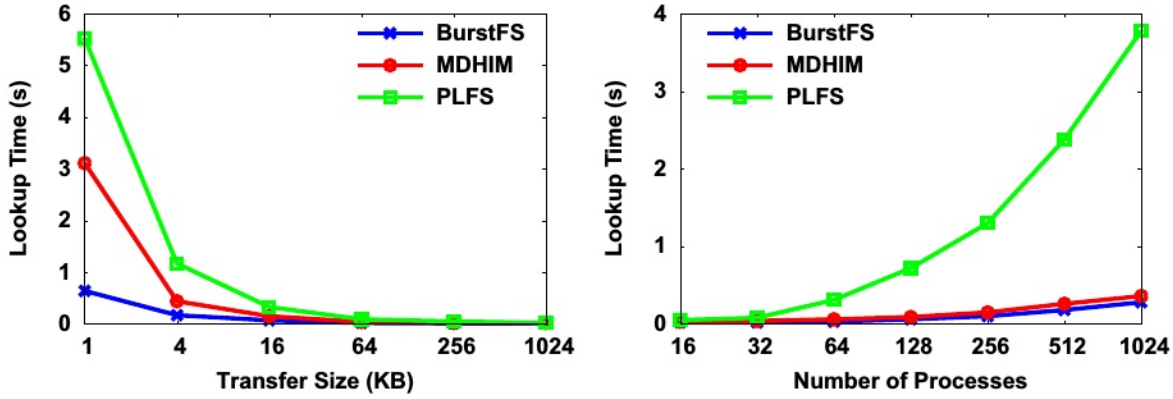
图10(b)显示了进程计数不断增加时的元数据性能。在这个测试中，每个进程查找64 MB的数据，以64 KB的传输大小写入。更多的进程会导致更多的查找操作。如图所示，随着进程数的增加，PLFS的查找时间急剧增加。相反，BurstFS和MDHIM的查找时间随着进程的增加而缓慢增加，这是因为元数据使用了分布式键值存储。

E. Tile-IO测试

图11显示了Tile-IO BurstFS的性能。在这个实验中，一个32GB的全局数组被划分在256个进程上。每个进程首先将其tile写入共享文件的几个不连续区域，然后将其读回本地内存。对于写操作，BurstFS通过直接将数据写入本地ssd而优于OrangeFS和PLFS。对于读取，尽管所有三个文件系统都受益于缓冲区缓存，但BurstFS仍然表现最好，因为每个进程都在本地读取数据。总的来说，BurstFS在读写方面分别比OrangeFS提高6.9倍和2.5倍，在读写方面分别比PLFS提高7.3倍和1.4倍。

F. BTIO测试

图12显示了问题大小为d的BTIO工作负载下BurstFS的性能。在本实验中，$408 \times 408 \times 408$全局数组被分解为64个进程。与Tile-IO类似，每个进程首先将自己的单元写入共享文件的几个不连续区域，然后将它们读回本地内存。由于采用了三维分区，每个进程的传输大小(2040B)比Tile-IO (32KB)要小得多，因此BTIO下的PLFS和OrangeFS的I/O带宽都比前者下降得快



(a) Metadata performance with varying transfer sizes

(b) Metadata Performance with varying process counts

图10:根据传输大小和进程计数分析元数据性能。

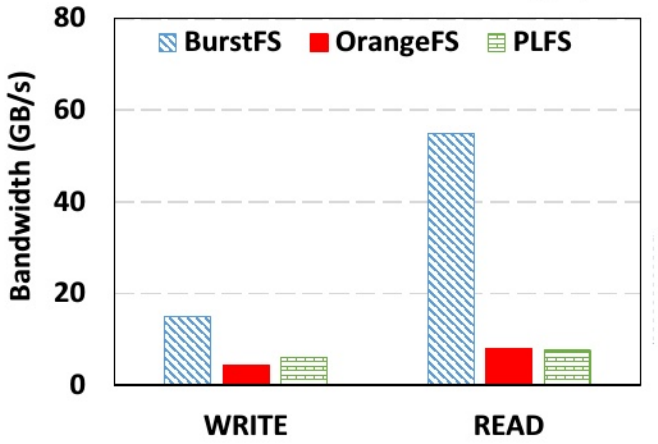


图11:Tile-IO性能。

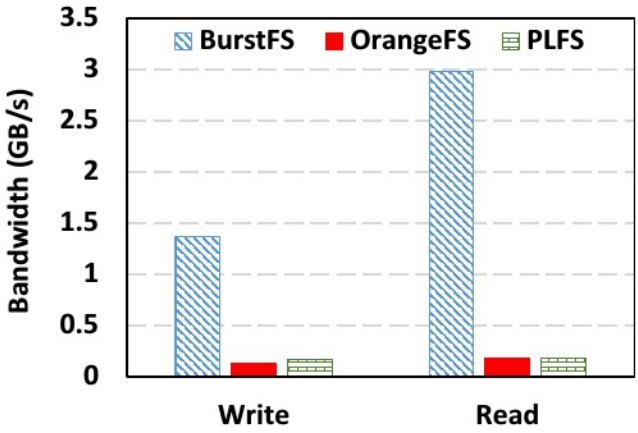


图12:BTIO性能。

Tile-IO。BurstFS通过本地读取和服务器端读取集群的优势来维持这种小消息工作负载。总的来说，与OrangeFS相比，它在读和写方面的性能分别提高了15.6倍和9.5倍。在读和写方面，它的性能也分别比PLFS高16.2倍和7倍。

G. IOR试验

为了评估批处理作业中不同程序之间对数据共享的支持，我们使用IOR进行了测试。我们运行IOR时，不同数量的进程读取由另一组进程从Tile-IO程序写入的共享文件。两个MPI程序中的进程在同一个作业中启动。每个节点承载16个Tile-IO进程和16个IOR进程。一旦Tile-IO进程完成了对共享文件的写入，该文件将由IOR进程使用N-1分段读取模式回读。我们保持与Tile-IO相同的IOR传输大小。由于读取模式与Tile-IO的初始写入模式不匹配，因此每个进程需要从远程节点上的多个日志中读取。我们将每个瓦片的大小固定为128MB，并将沿轴的瓦片数量固定为4，然后增加沿X轴的瓦片数量。因此，X轴上的瓦片数量将随着读取进程的数量增加而增加。图13比较了BurstFS与PLFS和OrangeFS的读带宽。PLFS和OrangeFS都容易受到较小传输大小(32KB)的影响。BurstFS保持高带宽，因为在本地组合小请求和服务器端读取集群和管道。平均而言，在读取Tile-IO产生的数据时，BurstFS的性能分别是OrangeFS和PLFS的2.3倍和2.5倍。

我们还使用两个BTIO类D和e，在BTIO生成的数据集上评估IOR的读取带宽。对于类D，我们使用64个进程将408 ×408 ×*408*数组写入共享文件。对于类E, 225个进程将1020 ×*1020* ×*1020*数组写入共享文件。在这两种情况下，共享文件然后由IOR进程使用N-1分段读取模式回读。图14显示了读取带宽。由于传输尺寸小得多(D类为2040B, 类E为2720B)，使用BTIO的OrangeFS和PLFS的带宽远低于Tile-IO。而BurstFS的性能也会受到小

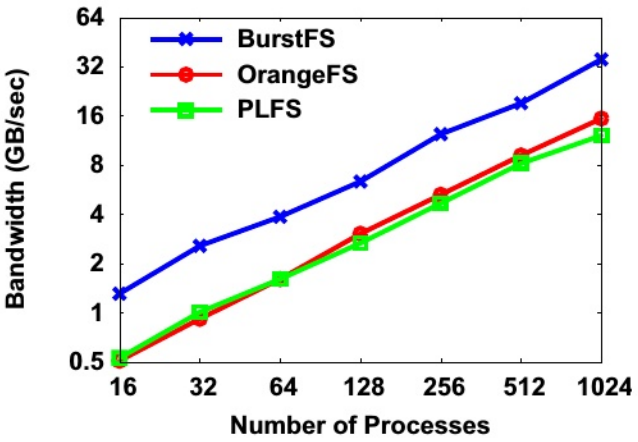


图13:Tile-IO写入共享文件时IOR的读带宽。

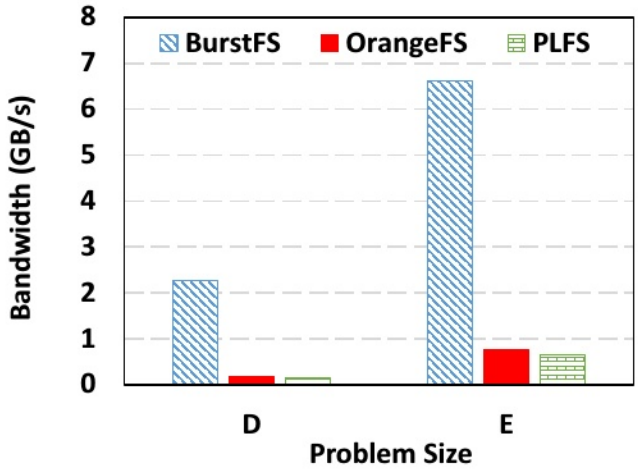


图14:BTIO写入共享文件时IOR的读带宽。

传输大小，它为这些小请求提供了更好的带宽。平均而言，在读取由BTIO生成的数据时，与OrangeFS和PLFS相比，BurstFS的性能分别提高了10倍和12.2倍。

V. 相关工作

猝发缓冲区的重要性体现在许多下一代超级计算机[2,8,9,10,11,12]的蓝图中，并在支持软件方面进行了广泛的投资。DataWarp[5]、IME[7]和aBBa[1]是Cray、DDN和EMC三个正在进行的项目。它们的潜在效益已经从各种研究角度进行了探索[25,35,42]。所有这些工作都针对远程、共享的突发缓冲器。相比之下，我们的工作集中在节点本地突发缓冲器上，这是一种同样重要的架构，目前缺乏标准化的支持软件。与远程突发缓冲区的工作相比，我们的工作提供了检查点/重启的线性可伸缩性，因为大多数I/O请求都是本地服务的。PLFS突发缓冲区[14]支持节点本地突发缓冲区(参见章节IV-A)，可以提供快速，可扩展的写入性能。它依赖全局文件系统(如Lustre、NFS)来管理metalinks，如果metalinks的数量很大，这可能是一种开销。此外，从PLFS突发缓冲区读取数据需要在所有计算节点上挂载每个节点本地突发缓冲区。BurstFS不同于PLFS突发缓冲区，因为它是作为一个独立的文件系统构建的。BurstFS使用其委托的集体服务实现可伸缩的读性能。此外，BurstFS专门用于管理节点本地突发缓冲区，而PLFS突发缓冲区同时支持节点本地突发缓冲区和远程共享突发缓冲区。

checkpoint/restart带来的I/O带宽需求随着计算能力的增长而增长。SCR [29]， CRUISE[32]和FusionFS[48]是值得注意的努力，旨在解决这种不断增加的I/O挑战，并通过让每个进程将单个文件写入节点本地存储(N-N)来实现线性写带宽。与这些工作不同的是，BurstFS支持N-1和N-N I/O模式，并为这两种模式提供可扩展的读/写带宽。多维I/O一直是并行文件系统的一个具有挑战性的工作负载。单个进程发出的小的、非连续的读/写请求可以极大地限制并行文件系统的带宽。为了解决这个问题，已经开发了几种最先进的方法。PLFS通过将小的、不连续的N-1写入转换为连续的、连续的N-N写入来加速它们[15]。然而，PLFS(没有突发缓冲区支持)仍然依赖于后端并行文件系统来存储来自N-N写入的单个文件。当两个文件在同一个存储服务器上被分条存储时，争用就会发生。相比之下，BurstFS提供了一个独立的文件系统服务。它通过本地写解决写竞争，并针对读密集型工作负载进行了优化。两阶段I/O[38]是另一种被广泛采用的优化小的、非连续的I/O工作负载的方法。所有的进程将它们的I/O请求发送到聚合器，聚合器将它们合并成大型的、连续的请求。BurstFS的读服务与两阶段I/O有一些相似之处:它的委托类似于将两阶段I/O中使用的I/O聚合器变成一个服务。然而，有两个关键的区别。首先，BurstFS的合并直接在文件系统而不是聚合器上进行。这避免了从聚合器到客户端进程的额外传输。其次，整合是由每个授权者单独完成的，没有额外的同步开销。

跨应用程序数据共享是一个令人生畏的话题，因为许多当代编程模型(例如MPI, PGAS)为每个程序定义了单独的名称空间。一种广泛采用的方法是利用现有的分布式系统，如分布式文件系统(如Lustre [17]， Or-angeFS [16]， HDFS[37])和分布式键值存储(如Memcached [30]，Dynamo [21]， BigTable[19])。然而，这些服务通常远离计算进程，产生有限的带宽。此外，启动、拆除和管理的沉重开销使得它们不适合与批处理作业中的应用程序共存。在

另一方面，一些服务程序被开发为在批处理作业中与应用程序一起运行。Docan等人[22]开发了DART，这是一种通信框架，能够通过位于与仿真应用程序(在同一作业中)不同的一组节点上的独立服务进程实现数据共享。他们后来的工作DataSpaces[22]扩展了原来的设计。在这两项研究中，应用程序进程以一种临时的方式向服务进程写入和读取数据。每个操作都需要一个单独的网络传输。相比之下，BurstFS中的委托被设计为与同一节点上的应用程序进程共存的I/O代理进程。所有的写操作都是本地的。读被延迟给I/O委派者，这提供了许多优化读操作的机会。

VI. 结论

在本文中，我们研究了数据管理对节点本地突发缓冲器的需求，这是一个关键的话题，因为节点本地突发缓冲器在下一代大规模超级计算机的设计中。我们管理节点本地突发缓冲区的方法是BurstFS，这是一个短暂的突发缓冲区文件系统，与批处理作业具有相同的生存期，专为HPC I/O工作负载的高性能而设计。BurstFS可以由同一作业中的多个应用程序使用，可以像检查点/重新启动一样顺序使用，也可以像集成应用程序一样并发使用。我们在BurstFS中实现了几种技术，它们极大地有利于具有挑战性的HPC I/O模式:可扩展的元数据索引、同址I/O委托以及服务器端读取集群和管道。这些技术确保了可扩展的元数据处理和快速的数据传输。我们的性能结果表明，BurstFS可以有效地支持各种具有挑战性的I/O模式。特别是，它可以支持跨分布式、节点本地的共享文件工作负载，性能非常接近于非共享文件工作负载。BurstFS对于并行写入和读取带宽也是线性扩展的，并且在很大程度上优于最先进的技术。

鸣谢。

我们非常感谢EMC的John Bent博士对配置和运行具有突发缓冲支持的PLFS的指导。我们也感谢来自匿名评论者的富有洞察力的评论。

根据合同DE-AC52-07NA27344，这项工作是在美国能源部的赞助下由劳伦斯利弗莫尔国家实验室进行的。llnl - conf - 681480。这项工作还得到了美国国家科学基金奖1561041的部分支持。

参考文献。

[1] Active Burst Buffer Appliance. http://www.theregister.co.uk/2012/09/21/emc abba//.

[2] Aurora. http://aurora.alcf.anl.gov/.

[3] Catalyst. http://computation.llnl.gov/computers/catalyst.

[4] Characterization and optimization of memory-resident mapre-duce on hpc systems.

[5] Datawarp. http://www.cray.com/products/storage/datawarp.

[6] Hyperion. https://hyperionproject.llnl.gov/index.php.

[7] Infinite Memory Engine. http://www. ddn. com/products/infinite-memory-engine-ime/.

[8] NERSC-8. https://www.nersc.gov/users/computational-systems/cori/.

[9] Sierra. https://www. llnl. gov/news/next-generation-supercomputer-coming-lab.

[10] Summit. https://www.olcf.ornl.gov/summit/.

[11] Theta. https://www. alcf. anl. gov/articles/alcf-selects-projects-theta-early-science-program.

[12] Trinity. http://www.lanl.gov/projects/trinity.

[13] TSUBAME2. http://tsubame. gsic. titech. ac. jp/en/hardware-architecture.

[14] John Bent, Sorin Faibish, Jim Ahrens, Gary Grider, John Patchett, Percy Tzelnic, and Jon Woodring. Jitter-Free Co-Processing on a Prototype Exascale Storage Stack. In *IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–5. IEEE, 2012.

[15] John Bent, Garth Gibson, Gary Grider, Ben McClelland, Paul Nowoczynski, James Nunez, Milo Polte, and Meghan Wingate. PLFS: A Checkpoint Filesystem for Parallel Applications. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. ACM, 2009.

[16] Michael Moore David Bonnie, Becky Ligon, Mike Marshall, Walt Ligon, Nicholas Mills, Elaine Quarles Sam Sampson, Shuangyang Yang, and Boyd Wilson. OrangeFS: Advancing PVFS.

[17] Peter J Braam and R Zahir. Lustre: A Scalable, High Perfor-mance File System. *Cluster File Systems, Inc*, 2002.

[18] Michael J Brim, David A Dillow, Sarp Oral, Bradley W Settle-myer, and Feiyi Wang. Asynchronous Object Storage with QoS for Scientifi c and Commercial Big Data. In *Proceedings of the 8th Parallel Data Storage Workshop*, pages 7–13. ACM, 2013.

[19] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A Distributed Storage System for Structured Data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Berkeley, CA, USA, 2006. USENIX Association.

[20] Cristian Coarfa, Yuri Dotsenko, John Mellor-Crummey, Franc¸ois Cantonnet, Tarek El-Ghazawi, Ashrujit Mohanti, Yiyi Yao, and Daniel Chavarr´ıa-Miranda. An Evaluation of Global Address Space Languages: Co-array Fortran and Unified Paral-lel C. In *Proceedings of the Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 36– 47. ACM, 2005.

[21] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gu-navardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vo-gels. Dynamo: Amazon's Highly Available Key-Value Store. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 205–220. ACM, 2007.

[22] Ciprian Docan, Manish Parashar, and Scott Klasky. DataSpaces: An Interaction and Coordination Framework for Coupled Sim-ulation Workflows. In *Proceedings of the 19th ACM Interna-tional Symposium on High Performance Distributed Computing*, HPDC '10, pages 25–36, New York, NY, USA, 2010. ACM.

[23] Hugh Greenberg, John Bent, and Gary Grider. MDHIM: A Par-allel Key/Value Framework for HPC. In *7th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 15)*, 2015.

[24] Jiahua He, Arun Jagatheesan, Sandeep Gupta, Jeffrey Bennett, and Allan Snavely. DASH: a Recipe for a Flash-Based Data In-tensive Supercomputer. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11. IEEE Computer Society, 2010.

[25] Ning Liu, Jason Cope, Philip Carns, Christopher Carothers, Robert Ross, Gary Grider, Adam Crume, and Carlos Maltzahn.

On the Role of Burst Buffers in Leadership-Class Storage Systems. In *IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–11. IEEE, 2012.

[26] LLNL. IOR Benchmark. https://github.com/LLNL/ior.

[27] Jay Lofstead, Milo Polte, Garth Gibson, Scott Klasky, Karsten Schwan, Ron Oldfield, Matthew Wolf, and Qing Liu. Six Degrees of Scientific Data: Reading Patterns for Extreme Scale Science IO. In *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, pages 49–60. ACM, 2011.

[28] Ewing Lusk, S Huss, B Saphir, and M Snir. MPI: A Message-Passing Interface Standard, 2009.

[29] Adam Moody, Greg Bronevetsky, Kathryn Mohror, and Bro-nis R De Supinski. Design, Modeling, and Evaluation of a Scal-able Multi-Level Checkpointing System. In *2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11. IEEE, 2010.

[30] Rajesh Nishtala, Hans Fugal, Steven Grimm, Marc Kwiatkowski, Herman Lee, Harry C Li, Ryan McElroy, Mike Paleczny, Daniel Peek, Paul Saab, et al. Scaling Memcache at Facebook. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 385–398, 2013.

[31] Sarp Oral, David A Dillow, Douglas Fuller, Jason Hill, Dustin Leverman, Sudharshan S Vazhkudai, Feiyi Wang, Youngjae Kim, James Rogers, James Simmons, et al. OLCFs 1 TB/s, Next-Generation Lustre File System.

[32] Raghunath Rajachandrasekar, Adam Moody, Kathryn Mohror, and Dhabaleswar K Panda. A 1 PB/s File System to Check-point Three Million MPI Tasks. In *Proceedings of the 22nd International Symposium on High-Performance Parallel and Distributed Computing*, pages 143–154. ACM, 2013.

[33] R. B. Ross. Parallel I/O Benchmark Consortium.

[34] Robert B Ross, Rajeev Thakur, et al. PVFS: A Parallel File System for Linux Clusters. In *Proceedings of the 4th annual Linux Showcase and Conference*, pages 391–430, 2000.

[35] Kiminori Sato, Kathryn Mohror, Adam Moody, Todd Gamblin, Bronis R De Supinski, Naoya Maruyama, and Shingo Matsuoka. A User-Level Infiniband-Based File System and Checkpoint Strategy for Burst Buffers. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, pages 21–30. IEEE, 2014.

[36] Frank B Schmuck and Roger L Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. In *FAST*, 2002.

[37] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop Distributed File System. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10. IEEE, 2010.

[38] Rajeev Thakur, William Gropp, and Ewing Lusk. Data Sieving and Collective I/O in ROMIO. In *Frontiers of Massively Parallel Computation, 1999. Frontiers' 99. The Seventh Symposium on the*, pages 182–189. IEEE, 1999.

[39] Yuan Tian, Scott Klasky, Hasan Abbasi, Jay Lofstead, Ray Grout, Norbert Podhorszki, Qing Liu, Yandong Wang, and Weikuan Yu. EDO: Improving Read Performance for Scientific Applications through Elastic Data Organization. In *2011 IEEE International Conference on Cluster Computing*, pages 93–102. IEEE, 2011.

[40] Teng Wang, Kathryn Mohror, Adam Moody, Weikuan Yu, and Kento Sato. BurstFS: A Distributed Burst Buffer File System for Scientific Applications. In *Poster Presented at the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2015.

[41] Teng Wang, Sarp Oral, Michael Pritchard, Bin Wang, and Weikuan Yu. Trio: Burst buffer based i/o orchestration. In *2015 IEEE International Conference on Cluster Computing*, pages 194–203. IEEE, 2015.

[42] Teng Wang, Sarp Oral, Yandong Wang, Brad Settlemyer, Scott Atchley, and Weikuan Yu. BurstMem: A High-Performance Burst Buffer System for Scientific Applications. In *Big Data (Big Data), 2014 IEEE International Conference on*, pages 71– 79. IEEE, 2014.

[43] Brent Welch, Marc Unangst, Zainul Abbasi, Garth A Gibson, Brian Mueller, Jason Small, Jim Zelenka, and Bin Zhou. Scal-able Performance of the Panasas Parallel File System. In *FAST*, pages 1–17, 2008.

[44] Parkson Wong and R der Wijngaart. NAS Parallel Benchmarks I/O Version 2.4. *NASA Ames Research Center, Moffet Field, CA, Tech. Rep. NAS-03-002*, 2003.

[45] Jiangling Yin, Jun Wang, Jian Zhou, Tyler Lukasiewicz, Dan Huang, and Junyao Zhang. Opass: Analysis and Optimization of Parallel Data Access on Distributed File Systems. In *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*, pages 623–632. IEEE, 2015.

[46] W. Yu, J.S. Vetter, R.S. Canon, and S. Jiang. Exploiting Lustre File Joining for Effective Collective I/O. In *7th Int'l Conference on Cluster Computing and Grid (CCGrid'07)*, Rio de Janeiro, Brazil, May 2007.

[47] W. Yu, J.S. Vetter, and H.S. Oral. Performance Characterization and Optimization of Parallel I/O on the Cray XT. In *22nd IEEE International Parallel and Distributed Processing Symposium (IPDPS'08)*, Miami, FL, April 2008.

[48] Dongfang Zhao, Zhao Zhang, Xiaobing Zhou, Tonglin Li, Ke Wang, Dries Kimpe, Philip Carns, Robert Ross, and Ioan Raicu. FusionFS: Toward Supporting Data-Intensive Scientific Applications on Extreme-Scale High-Performance Computing Systems. In *Big Data (Big Data), 2014 IEEE International Conference on*, pages 61–70. IEEE, 2014.