DOI:10.7544/issn1000-1239.2015.20148073
52(7):1522-1530,2015

一种面向大规模数据密集计算的缓存方法

周恩强[1] 张 伟[1] 卢宇彤[1] 侯红军[2] 董 勇[1]
[1](高性能计算国家重点实验室(国防科学技术大学) 长沙 410073)
[2](中国石油集团东方地球物理勘探公司 河北涿州 072751)
(eqzhou@nudt.edu.cn)

A Cache Approach for Large Scale Data-Intensive Computing

Zhou Enqiang[1], Zhang Wei[1], Lu Yutong[1], Hou Hongjun[2], and Dong Yong[1]
[1](*State Key Laboratory of High Performance Computing* (*National University of Defense Technology*), *Changsha* 410073)
[2](*Bureau of Geophysical Prospecting*, *China National Petroleum Corporation*, *Zhuozhou*, *Hebei* 072751)

Abstract With HPC systems widely used in today's modern science computing, more data-intensive applications are generating and analyzing the increasing scale of datasets, which makes HPC storage system facing new challenges. By comparing the different storage architectures with the corresponding approaches of file system, a novel cache approach, named DDCache, is proposed to improve the efficiency of data-intensive computing. DDCache leverages the distributed storage architecture as performance booster for centralized storage architecture by fully exploiting the potential benefits of node-local storage distributed across the system. In order to supply much larger cache volume than volatile memory cache, DDCache aggregates the node-local disks as huge non-volatile cooperative cache. Then high cache hit ratio is achieved through keeping intermediate data in the DDCache as long as possible during overall process of applications. To make the node-local storage efficient enough to act as data cache, locality aware data layout is used to make cached data close to compute tasks and evenly distributed. Furthermore, concurrency control is introduced to throttle I/O requests flowing into or out of DDCache and regain the special advantage of node-local storage. Evaluations on the typical HPC platforms verify the effectiveness of DDCache. Scalable I/O bandwidth is achieved on the well-known HPC scenario of checkpoint/restart and the overall performance of typical data-intensive application is improved up to 6 times.

Key words data-intensive computing; cache; local storage; shared storage; seismic data processing

摘 要 随着高性能计算机逐步应用在大规模数据处理领域,存储系统将成为制约数据处理效率的主要瓶颈.在分析了影响数据密集型计算I/O性能若干关键因素的基础上,提出使用计算结点本地存储构建协作式非易失缓存、以分布式存储架构加速集中式存储架构的方法.该方法基于应用层协同使用分布化的本地存储资源,使用非易失存储介质构成大缓存空间,存放大规模数据分析的中间过程结果,以此实现高缓存命中率,并利用并发度约束控制等手段避免I/O竞争,充分利用本地存储的特定性能优势保证缓存加速效果,从而有效地提高了大规模数据处理过程的I/O效率.基于多平台多种I/O模式的测试结果证实了该方法的有效性,聚合I/O带宽具有高扩展性,典型数据密集应用的整体性能最大可提升6倍.

收稿日期:2014-01-21;**修回日期**:2014-07-14
基金项目:国家自然科学基金项目(61120106005);国家"八六三"高技术研究发展计划基金项目(2012AA01A301)

关键词 数据密集计算;缓存;本地存储;共享存储;地震数据处理

中图法分类号 TP311

随着科学发现的深度和广度的扩展,高性能计算(high performance computing, HPC)系统越来越多地用于大规模数据处理[1-2],例如 TH-1A 超级计算机已广泛用于大规模地震数据处理和基因数据分析等数据密集计算领域.这类适合 HPC 系统处理的数据密集应用除了具有计算密集的特征,还可被视为一类大数据问题,其突出特点是输入和输出的数据规模大,往往不能全部放入内存,需要利用 HPC 系统中多核、多处理器以及多结点并行的架构进行高并发度 I/O 和数据处理,在复杂的数据处理过程中大量的随机化磁盘访问使得问题求解过程的 I/O 时间比例明显增加,I/O 访问成为主要瓶颈,严重限制了 HPC 平台在数据密集计算领域的应用.

cache 机制是存储系统优化带宽和延迟的常用手段.I/O 路径上的各类 cache 可以吸收突发数据,平滑慢存储设备的性能差异,隐藏延迟,也可以缓存热点数据,减少慢速设备的访问次数.cache 通常采用低延迟的易失性存储介质实现,例如 DRAM,相对于目前的数据规模,这类存储介质在系统中属稀缺资源.在大数据量随机访问条件下,有限容量的 cache 往往会失去优化作用.

针对以上背景和问题,本文研究一种基于非易失存储介质的 cache 方法.实际上典型 HPC 系统在处理器和外部存储中间,除了 DRAM 这类易失性存储介质外,还存在多种非易失存储介质,例如计算结点上 I/O 总线挂接的本地磁盘、NVRAM 类新型存储介质等,这类本地化存储介质的访问延迟虽然远大于 DRAM,但具有容量大、处理器耦合紧密等优势,特定条件下其性能甚至强于 HPC 系统中的主力存储系统.高效利用本地化存储资源已经成为热点研究问题[3-4],有潜力成为将来 E 级计算机存储系统的重要组成部分,本文的研究目标即如何利用这类存储资源提高数据密集计算的 I/O 效率.

本文以地震数据处理作为大规模数据密集应用的典型代表,研究如何利用 HPC 系统中本地化存储资源作为应用程序的数据 cache 来加速数据处理过程.基于对 HPC 系统中各种 I/O 性能影响因素和本地存储性能特征的分析,提出了应用层耦合的协作式 cache 方法,该方法协同使用大容量、分布化和非易失的本地存储资源构成数据 cache,并绑定运行的作业,利用本地存储的特定性能优势来加速大规模数据的访问过程.实际测试证明了该方法的有效性,在多种访问模式下均可有效提高 I/O 效率,典型模式的 I/O 性能提升达 3~6 倍.

1 相关研究

cache 机制的重要作用是在高速存储介质中暂存数据,通过重用热点数据或重叠计算和 I/O 过程,避免或隐藏对低速存储介质的访问.大量的研究工作从数据访问模式的局部性和 cache 数据淘汰或预取算法等角度提高 cache 的优化效率[5].对于强局部性的 I/O 访问模式,通常易于优化,但对于大规模数据数据处理过程中随机性较强的 I/O 访问模式,则难以预测访问序列并优化.对于并发度高、局部性较差且数据规模较大的访问模式的场景,cache 容量的大小往往决定了 cache 机制能够带来多少性能好处,这也是本文研究的重要依据.

协同式 cache[6-7] 能够聚合使用分布的内存资源,构成更大的缓存空间,可缓解 cache 容量不足的问题,促进 cache 资源的使用.当计算方法中涉及的数据规模超过全部内存容量时,这种方法依然受限,而且使用过多占用内存也会影响计算效率.一些工作研究基于非 DRAM 的 cache 技术优化某些特定 I/O 模式.DCD[8] 技术在设备层使用磁盘来缓存磁盘操作,主要利用磁盘的顺序读写能力加速磁盘的小写操作.基于闪存介质的 cache 技术[9-10] 利用闪存的延迟优势,通过缓存算法尽可能地将延迟敏感的请求由闪存介质完成,这类技术适合在设备层实现,用于优化小粒度读写频繁、热点明显的 I/O 模式.文献[11]协同使用内存和 PCM 介质实现混合大缓存空间.文献[12]使用内存缓存优化数据处理的中间过程结果,本文则侧重使用非易失存储介质构成大缓存空间,存放大规模的中间过程结果.

对象存储架构[13] 在 HPC 系统中使用广泛,能够提供高聚合带宽,其典型代表 lustre 文件系统[14] 设计了多种 cache 机制来优化传统数值模拟应用 I/O 过程中常见的文件顺序访问模式,但该架构的访问延迟较大,更适合顺序访问大文件,本文的研究工作可以弥补对象存储架构的不足.

一些研究工作利用 HPC 本地化存储资源来缓解 I/O 瓶颈.文献[15-16]利用计算结点本地磁盘暂存

大文件,例如专门将本地存储介质用作检查点文件的存储,以缓解集中式存储架构面临的 I/O 瓶颈问题. 这类研究表明发挥本地存储资源和计算紧密耦合的优势,能够显著优化某些特定的 I/O 模式. 这些研究工作中计算结点之间相对独立,适合优化任务间数据访问无关联的 I/O 模式,本文研究本地存储资源聚合协同机制,使其能够优化数据关联性较强的大规模数据规约和数据后处理等分析类应用.

2 问题分析

HPC 系统中常见的存储资源可分为集中部署的共享存储和分布部署的本地存储,如图 1,计算结点(compute node, CN)分别通过总线和网络连接磁盘或存储结点(storage node, SN),构成不同类型的存储系统.

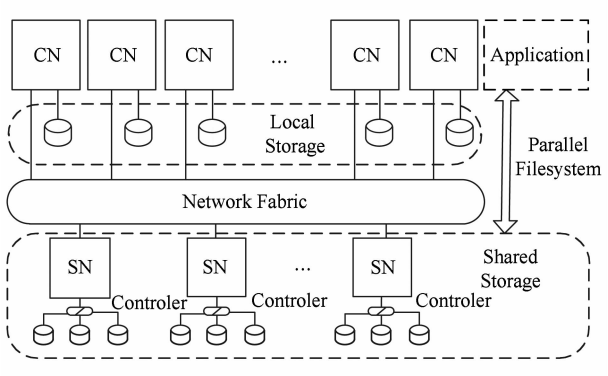


Fig. 1 Architecture of HPC storage.

图 1 HPC 存储架构示意图

定义 1. 共享存储. 采用计算和存储分离的集中式存储架构,存储介质分别通过全局文件系统、存储结点和 RAID 控制器多个层次来集中控制,所有的计算结点通过网络共享使用和并发访问.

定义 2. 本地存储. HPC 系统中计算结点独占专用的磁盘或固态盘,属于分布式存储架构,每个计算结点上通过 I/O 总线独立访问本地存储介质.

HPC 系统中应用程序任意 I/O 访问请求的完成时间可描述为

$$T_{io} = \alpha T_{baseline}, \tag{1}$$

其中,$T_{baseline}$为理想情况下存储介质的最优 I/O 请求完成时间,α 为性能影响因子,取决于 I/O 全路径多种因素,α 越大,请求完成时间越长,意味着应用程序的 I/O 效率越低. 本文忽略 HPC 系统 I/O 关键路径上对 I/O 性能影响小的因素(例如,假设内部互连网络性能足以满足 I/O 需求),将影响 α 的因素综合归纳为应用程序的 I/O 模式、文件系统和存储硬件架构 3 个方面,则 α 可表示为

$$\alpha = \phi(app, fs, arch). \tag{2}$$

以下分别分析三者对 α 的影响. 对于局部性较强的应用程序访问模式,例如顺序访问和热点数据访问使得数据可以预先聚集在 cache 中,能够隐藏 I/O 延迟,α 可有效降低. 在并发 I/O 规模增长迅速条件下,高达数万至数十万并发规模的 I/O 请求会竞争规模有限的共享存储系统,使得原来局部性较强的模式在共享存储端被破坏,表现出访问序列随机化的趋势[17]. 另一方面,典型数据密集型应用往往按逻辑关系访问数据集合,当数据的顶层逻辑关系和底层物理存储关系不匹配或 I/O 栈不同层次间数据视图不一致时[18],也会产生底层存储系统不易预测和优化的随机化 I/O 访问序列,导致存储系统效率降低,此时 α 增大.

地震数据处理是 HPC 系统用于大规模数据处理的典型代表性应用. 地震数据处理以野外采集的测量数据作为原始输入数据,由相对独立的多个计算过程迭代完成数据处理,整个处理过程中产生的中间过程数据比原始数据膨胀达数十倍以上,基于 DRAM 的 cache 机制无法发挥作用,大量中间过程数据需重复访问,但数据重复访问的时间窗口和访问序列不确定,无明显热点,底层存储系统无法预测访问序列. 这种典型 I/O 模式导致 α 在任务规模较大的情况下被放大,某些处理流程中 I/O 时间比例高达 80%以上,严重降低了 HPC 系统处理这类数据密集问题的计算效能,本文即着重研究这类由数据访问序列随机化带来的性能问题.

图 2 模拟上述模式评估文件系统对存储效率的影响. 测试中设定大数据量排除内存 cache 的影响,

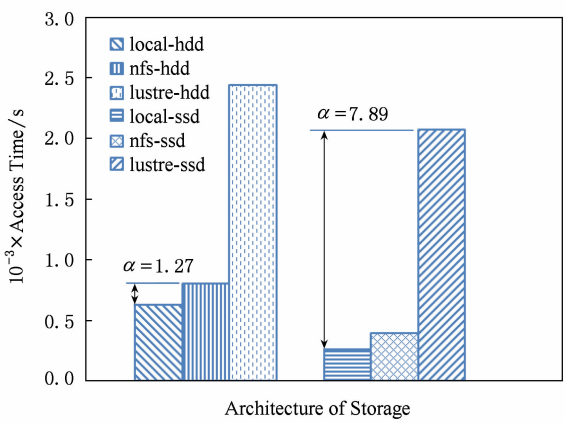


Fig. 2 Comparison of overhead with different file system.

图 2 文件系统软件开销分析

采用长度为12 KB的常用地震道体作为单次访问的数据块,模拟地震数据处理的读写混合模式.以本地磁盘性能做为性能基线,则该测试中最小的影响因子 $\alpha_{\text{nfs-hdd}}\approx 1.27$,最大 $\alpha_{\text{lustre-ssd}}\approx 7.89$.

这说明了该应用模式条件下,不同的文件系统对不同存储介质I/O请求效率的影响差异很大,lustre这类文件系统适合构建较大规模共享存储系统,适合大文件顺序访问,而对于访问序列随机化的数据处理模式,即使部署固态盘存储介质,带来的好处远小于较大的软件协议延迟开销,反而复杂性较低的NFS的 $\alpha_{\text{nfs-hdd}}$ 较小.

图3采用本地分布和集中共享2种架构形态部署8个磁盘进行以上测试,分析存储架构对性能的影响.本地存储架构下的磁盘由8个结点独立控制,数据在8个磁盘理想化均衡分布,共享架构的磁盘由单个存储结点集中控制,由磁盘阵列控制器以RAID5冗余的方式分布数据,文件系统条件相同.

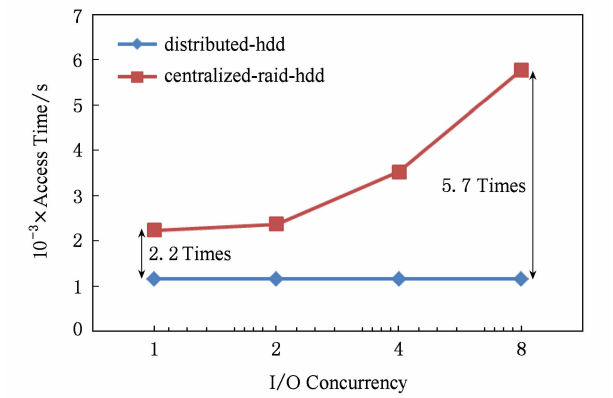


Fig. 3 Overhead of different storage architecture.

图3 存储架构性能开销分析

盘阵控制器的实际性能强弱影响共享存储系统 $\alpha_{\text{central-hdd}}$ 的绝对值,假设以本地单磁盘性能为性能基线,理论上 $\alpha_{\text{central-hdd}}\leqslant 1$.测试中 $\alpha_{\text{central-hdd}}$ 则从单I/O流的3.5增长至8个I/O流的5.7,随着I/O并发度增长逐渐增大,说明了集中式存储架构的I/O请求处理效率在特定条件下低于本地存储架构,而且受I/O并发度影响较大[19].

基于以上分析,对于访问序列随机化、热点不突出的固定数据密集处理模式,α 可细化为

$$\alpha = \phi_1(fs)\phi_2(arch)\phi_3(concurrency), \quad (3)$$

即文件系统协议的复杂度、存储架构和并发度是影响I/O请求效率的3个重要因素.对于大规模并行条件下的共享存储系统,这3个因素相互独立又相互影响,例如共享存储条件下单个磁盘的并发度不易控制[20],分布式存储架构下文件的统一管理导致文件系统协议比集中式架构复杂,NFS协议虽然简单,但难以支撑高并发度I/O.

3 基于本地存储的协同式cache

本文提出基于本地存储构建应用层cache的办法来协调三者的关系,利用目标应用模式下本地存储 α_{local} 低于共享存储 α_{central} 的特征,降低该应用I/O访问请求的 α 影响因子,达到提高典型数据密集处理模式I/O效率的目标.

从功能上,本地存储暂时无法替代共享存储,但从存储架构角度分析,本地存储在特定应用背景下具有性能优势.由图2和图3的分析可知,计算任务独立使用本地磁盘,多任务均衡匹配多个本地磁盘,计算和存储均衡可扩展等理想条件下,本地化存储避免了集中共享存储架构[19]的弱点,具有性能优势.

定义3. 本地存储的优势访问模式.即在随机化访问模式、访问协议简化、数据均衡分布、I/O并发度可控和较大的并发I/O规模等条件下本地存储具备性能优势的数据访问模式.

HPC系统中大规模数据处理的典型场景下本地存储优势访问模式具备形成条件,例如访问模式和任务并发规模,而其他条件则可采用必要的系统设计来满足.

密集数据处理作业的运行时间周期可细化分解为

$$T_{\text{total}} = T_{\text{raw\_input}} + T_{\text{temp\_input}} + T_{\text{processing}} + T_{\text{temp\_output}} + T_{\text{final\_output}}, \quad (4)$$

其中,$T_{\text{raw\_input}}$ 和 $T_{\text{final\_output}}$ 分别为原始输入和最终结果I/O时间,$T_{\text{temp\_input}}$ 和 $T_{\text{temp\_output}}$ 分别为若干个计算迭代过程的中间数据I/O时间.中间数据存在随机性重复访问的特点,非常适合长期存放在cache中,假设存在足够大的cache空间保证需重复处理的中间数据有较高的命中率,必然会给数据密集算带来极大的好处.本地存储处于处理器和全局共享存储之间,容量相对较大,例如单个磁盘容量可达结点内存容量的数十倍,聚合使用将形成一个“巨大cache”.该cache的存储介质和全局存储相同,通过控制对该数据cache的I/O访问,使其工作在本地存储的性能优势访问模式下,访问该cache大多数I/O请求的影响因子 α_{local} 将低于访问共享存储的 α_{central},即可获得加速效果.该过程中,本地存储承担降低 α 的作用,共享存储则主要承担数据结果永久存储的作用.

这种利用本地存储架构优势的 cache 方法,本文简称为 DDCache(distributed disks based file cache). DDCache 利用本地存储资源容量充足、非易失等特点,保证较高的 cache 命中率,基于式(3)的结论,多方面保持 DDCache 的性能优势,即可提升大规模数据处理作业的整体 I/O 性能. 3.1 节描述 DDCache 的关键设计技术.

3.1 DDCache 核心设计和实现

DDCache 在 I/O 路径上处于应用程序和共享存储之间,系统资源调度器给并行作业分配计算结点后,这些计算结点的本地存储即可构成作业的数据 cache. 图 4 为 DDCache 组成结构图,DDCache 采用用户态实现技术,关键模块以动态库形式存在. DDCache 的管理服务运行在独立的结点上,负责存储 DDCache 的基本信息,例如 cache 标识、cache 状态和 cache 结点列表. 客户端主要负责提供数据访问和辅助管理接口,提供 cache 空间视图,负责定位文件数据的存储位置. 客户端以 I/O 库的形态运行在计算任务的进程上下文. 服务端以常驻守护进程的形式存在,通过上下文区分 DDCache 请求. 客户端和服务端之间在结点内部通过共享内存传输数据,在结点间采用网络完成数据传输.

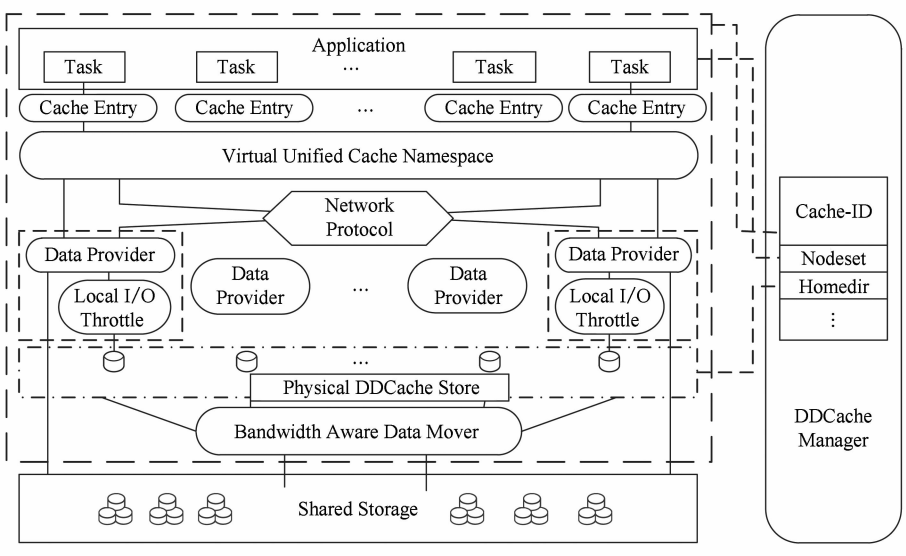


Fig. 4 Architecture of DDCache.

图 4 DDCache 结构图

每个作业都可创建自己的数据 cache,每个 DDCache 有唯一的标识和上下文,上下文用以区分资源归属,可描述为二元组(*nodes*,*home*),*nodes* 为计算结点集合,*home* 为该 DDCache 在本地存储上的专有目录,这些信息由管理服务器负责记录和管理. DDCache 内部采用 Client/Server 架构,在计算结点上对等分布. 应用程序通过兼容标准文件访问语义的 cache 接口使用虚拟 cache 空间,透明地访问分布在各个结点上的缓存数据.

3.2 DDCache 关键技术

本地存储能够加速共享存储的 I/O 访问的 2 个核心问题是:

1) 本地存储构成大容量单一 cache 空间;

2) 本地存储必须工作在优势访问模式下.

为了解决上述关键问题,DDCache 采用以下设计:

1) 应用级独占式 cache 设计

cache 被多个应用程序共享使用时,多种模式相互干扰会降低 cache 效率[21]. 基于 HPC 系统中作业普遍独占计算结点的资源使用模式,DDCache 采用应用级独占式设计,每个 DDCache 由作业创建和独占使用,作业内部任务分布共享 cache 数据,附属于多个作业的 DDCache 相互间无竞争和干扰. 并行作业的主任务负责创建 DDCache,将 cache 标识广播给子任务,通过 cache 标识获取 DDCache 上下文后,才能访问 cache 内数据. 这种紧耦合的使用的模式避免了作业间 I/O 请求竞争本地存储资源的问题,DDCache 资源应用程序可控,降低了 cache 数据管理的复杂性.

2) 协同式虚拟 cache 空间

DDCache 结点集合的每个成员有唯一的索引标识,相互协同将本地存储空间虚拟聚合成 DDCache 空间. 文件是构成虚拟 cache 空间的基本单元,DDCache 缓存的文件集合构成了一个大的目录树,

该空间包含了作业的数据集.DDCache 的每个文件分为数据和元数据 2 部分存储,元数据的存储结点以文件全路径名为键值通过散列算法映射生成,再通过元数据链接到数据,使得虚拟空间管理具有高可扩展性,避免集中管理的软件控制开销.

为了保证数据均衡分布和保持本地存储的优势访问模式,DDCache 采用本地 I/O 约束策略控制 cache 内数据的访问.首先,利用亲和调度规则约束文件数据存储位置,数据仅存储在任务所在结点,而不是任意分布,这使得作业内部生成的数据能够自然均衡分布.其次,对于读请求采用请求分流控制策略,事先设定本地存储的最优 I/O 并发度,本地存储未工作在最优状态且共享存储可用时,数据读请求将绕过 DDCache 直接调度到共享存储完成.假设应用程序的 I/O 是均衡的,DDCache 虚拟空间的数据分布和调度约束规则使得整个数据集的文件布局具有本地化、均衡化且可扩展的特征,单结点磁盘的 I/O 并发度和计算任务数适度匹配,避免过度并发.

3) cache 数据管理策略

DDCache 以大容量和长缓存周期来换取高命中率,采用不同的策略优化不同类型数据访问.DDCache 利用应用级 cache 接口来区分数据文件类型.对于原始输入数据,利用共享存储系统顺序读取性能高的特点,采用一次性读入 DDCache 的贪婪预取策略[22],后续访问可直接基于 DDCache 存取数据;对于已存在于 DDCache 的中间过程数据,采用被动替换策略,基于生存周期或应用程序的暗示确定数据丢弃时机,保持数据在处理周期内有效缓存.

DDCache 利用计算任务的 I/O 空闲周期将数据写回到共享存储以保证数据可靠性.为了提高数据移动效率,避免 I/O 竞争,DDCache 写回过程采用全局 I/O 约束控制机制,假设本地存储和共享存储的最优 I/O 并发度分别是 C_l 和 C_g,DDCache 内含 n 个结点,则每个结点的写回最佳并发度为 $\min[C_l, C_g/n]$.这种方法避免了过度并发引发的本地或共享存储效率下降现象.

4) 应用层协同的粗粒度一致性维护方法

DDCache 将数据写回到全局存储后,在 DDCache 内留有 1 份副本.为降低一致性维护的复杂性,DDCache 采用基于阶段乐观的一致性模型,即乐观假设在一个处理阶段内作业对 cache 数据没有读写互斥访问,允许副本不一致,以此降低 cache 数据的写回次数.事实上大部分 HPC 系统的 I/O 场景符合这一假设.例如,地震数据处理场景下,数据生成和数据分析处理通常分属不同的处理阶段,发生读写冲突的概率为零,只要在阶段结束时保证数据一致性,即可满足数据处理场景的一致性需求.DDCache 由应用程序在阶段处理结束时保证数据副本的一致性,避免为了适应各种场景的一致性要求而增加协议复杂性代价[23].

4 DDCache 实现和性能评估

DDCache 适用于磁盘和闪存等存储介质,本文采用普遍使用的磁盘来评估 DDCache 性能.表 1 为测试平台配置,平台 1 为天河-2 超级计算机中部署了本地磁盘的结点分区,平台 2 为典型地震数据处理集群.为了公平准确评估性能差异,测试平台中共享存储的磁盘数和本地存储的磁盘数尽可能一致,数量差异在 10%以内.

Table 1 Comparison of Evaluation Platform

表 1 测试平台配置对比

Platform	Compute Node	Local Storage	Global Storage	Interconnect
1	2048	2048 SATA Disk Ext3	SATA Disk RAID6 Lustre	80 Gbps TH-Express
2	64	64 SATA Disk Ext3	SATA Disk RAID5 NFS	1 Gbps Ethernet

我们在平台 1 上采用 HPC 应用中常见的检查点 I/O 模式来评估 DDCache 聚合 I/O 带宽的扩展能力,使用地震数据并行 I/O 模拟程序用来测试随机访问性能.在平台 2 上采用国内东方地球物理公司的 GeoEast 地震数据处理软件来评估 DDCache 对真实数据密集处理场景的 I/O 加速效果.为了准确对比本地存储和全局存储的性能差异,测试中关闭分流调度.测试中假设磁盘的最佳并发度为 1,I/O 任务并发数与之持平,每个结点的任务数和磁盘数相同.

1) DDCache 扩展能力评估

图 5 显示了 DDCache 和共享存储的扩展性差异.测试过程中数据量随并发任务数线性增长,任务交叉均跨结点读取检查点文件,避免结点内存 cache 对性能的影响.对于顺序访问为主的 CR 模式,当结点数在 256 以内时,共享存储的 I/O 带宽呈较好的增长趋势,说明任务规模适度的情况下共享存储可发挥较好的顺序访问性能,而 DDCache 的约束调度

策略使其仅使用部分结点本地存储,此时共享存储性能优于本地存储. 当结点规模继续增大时,由于I/O并发数超出了共享系统可以良好支撑的范围,I/O冲突概率逐渐升高,磁盘访问的局部性被干扰,共享存储效率开始下降,而DDCache的并发度均衡匹配磁盘,拥塞概率低,1 024结点时性能超出共享存储,2 048结点时带宽优势明显.

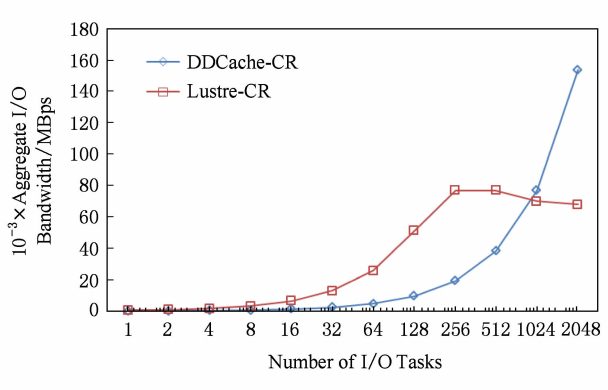


Fig. 5 Performance of checkpoint/restart.

图5 检查点模式I/O测试

地震数据并行I/O模拟程序测试设定100%的请求为完全无局部性的随机访问,数据量伴随任务数线性增长. 当结点数在128～256之间时,图6结果显示共享存储效果好于DDCache,原因在于小规模条件下存储结点的内存cache仍能够发挥作用,而且DDCache使用的磁盘数少于共享存储. 伴随任务数扩展,数据量增大,共享存储有限的内存cache逐渐失去作用,DDCache则表现出良好的I/O加速能力,在2 048规模时,DDCache的效率约为共享存储的6倍. 性能变化曲线说明DDCache方法对于大规模随机数据访问模式的有效性. DDCache处理原始数据时,先从共享存储预取数据到DDCache缓存,尽管增加了一个预取过程,但大粒度顺序预取的开销远低于同数据量随机访问的开销,因此DDCache处理原始数据的时间仅略多于直接处理缓存内中间过程数据的时间. DDCache的预取模式适合应用于实际的数据处理场景,即由共享存储存储原始数据、部分关键中间结果和最终结果数据,保证可靠存储,DDCache负责预取并加速数据处理过程.

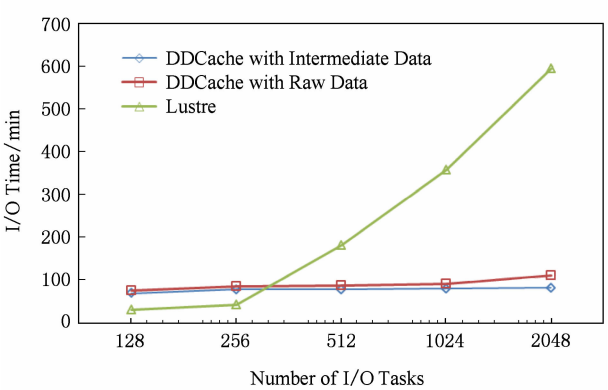


Fig. 6 Performance of synthetic seismic benchmark.

图6 地震数据并行I/O模拟测试

2) DDCache实际数据密集场景性能评估

图7和图8显示了DDCache对于真实数据密集场景的加速效果,测试过程中完全基于DDCache生成和读取地震数据,顺序访问和随机访问各占50%. 图7显示64个并发任务条件下,随着数据规模的增大,DDCache的性能优势逐步提升,1.5 TB数据量时I/O效率提升4倍. 图8显示数据量固定为750 GB,扩展至128个任务时,共享存储开始受限于并发数性能明显下降,DDCache仍能够获得性能好处,数据处理能力提升3.3倍.

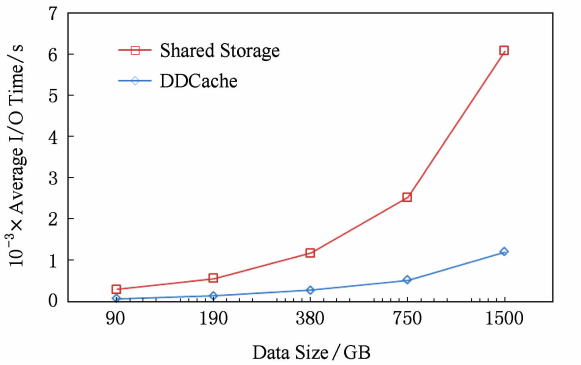


Fig. 7 Performance of different data size.

图7 扩展数据量测试

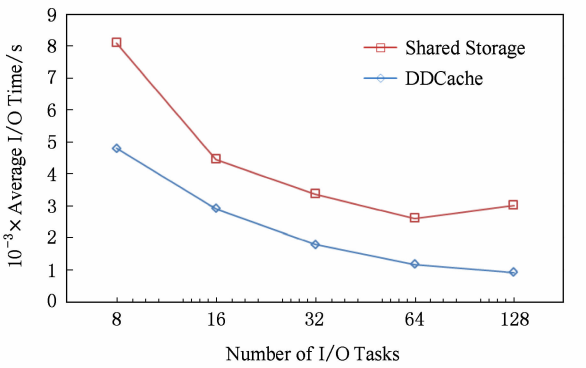


Fig. 8 Performance of different tasks.

图8 扩展任务数测试

该测试中因结点本地磁盘空间大于需处理的数据量,DDCache能够保持100%访问命中率. 测试中DDCache阶段性将数据写回共享存储,尽管平台1采用千兆以太网配置,网络延迟相对较大,带宽相对较低,但DDCache加速效果依然明显. 以上测试证明了DDCache方法对于优化数据量较大、并发数较

高和随机访问操作较频繁的数据密集 I/O 模式的有效性.

以上测试中 DDCache 表现出良好的扩展能力和加速效果,其性能趋势符合本文第 2 节的分析结论,达到了以磁盘加速磁盘的效果.闪存类存储的访问延迟远优于磁盘,更适合部署为本地存储,该条件下 DDCache 的加速效果将更加突出.

5 总结和展望

I/O 瓶颈问题已成为制约超级计算机处理大规模数据密集计算问题的主要因素,本文分析了 HPC 系统中影响 I/O 效率的多种因素,提出了基于计算结点本地存储资源来提高 I/O 效率的协同式非易失 cache 方法.该方法通过多种手段充分利用本地存储资源的性能优势,来加速 HPC 系统中大规模数据的访问过程,基于磁盘平台的应用测试证明了该方法的有效性和突出的加速效果.本文的工作已经应用于实际地震数据处理系统,DDCache 原理同样适用于其他新型存储介质,后续工作将利用天河-2 系统的闪存类存储介质的适度部署,进一步探索本地存储在更大规模高性能计算机系统中的作用和优化方法.

参 考 文 献

[1] Raicu I, Foster I, Zhao Y, et al. Towards data intensive many-task computing [C] //Proc of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies. New York: ACM, 2012: 28-73

[2] Lang S, Carns P, Latham R, et al. I/O performance challenges at leadership scale [C] //Proc of the Conf on High Performance Computing Networking, Storage and Analysis, SC'09. New York: ACM, 2009: 1-12

[3] Strande S M, Cicotti P, Sinkovits R S, et al. Gordon: Design, performance, and experiences deploying and supporting a data intensive supercomputer [C] //Proc of the 1st Conf on the Extreme Science and Engineering Discovery Environment. New York: ACM, 2012: 1-8

[4] Dong X, Xie Y, Muralimanohar N, et al. Hybrid checkpointing using emerging nonvolatile memories for future exascale systems [J]. ACM Trans on Architecture and Code Optimization (TACO), 2011, 8(2): 1-29

[5] Xu J, Sun B, Si C. The survey of cache management in the shared storage environment [C] //Proc of 2012 Int Conf of Internet Technology and Secured Transaction. Piscataway, NJ: IEEE, 2012: 139-142

[6] Dahlin M, Wang R, Anderson T, et al. Cooperative caching: Using remote client memory to improve file system performance [C] //Proc of the 1st Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 1994: 267-280

[7] Niu Xinzheng, She Kun, Qin Ke, et al. A cooperative caching optimized policy of mobile peer-to-peer networks [J]. Journal of Computer Researc and Development, 2008, 45(4): 656-665 (in Chinese)
(牛新征, 佘坤, 秦 科, 等. p2p 网络的协作缓存优化策略[J]. 计算机研究与发展, 2008, 45(4): 656-665)

[8] Yang Q, Hu Y. DCD-disk caching disk: A new approach for boosting I/O performance [C] //Proc of the 23rd Annual Int Symp on Computer Architecture. New York: ACM, 1996: 169-178

[9] Koller R, Marmol L, Rangaswami R, et al. Write policies for host-side flash caches [C] //Proc of the 11th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2013: 45-58

[10] Byan S, Lentini J, Madan A, et al. Mercury: Host-side flash caching for the data center [C] //Proc of the 28th Symp on Mass Storage Systems and Technologies (MSST 2012). New York: ACM, 2012: 1-12

[11] Ramos L, Bianchini R. Exploiting phase-change memory in cooperative caches [C] //Proc of the 24th Int Symp on Computer Architecture and High Performance Computing. Piscataway, NJ: IEEE, 2012: 227-234

[12] Qi Kaiyuan, Han Yanbo, Zhao Zuofeng, et al. MapReduce intermediate result cache for concurrent data stream processing [J]. Journal of Computer Research and Development, 2013, 50(1): 111-121 (in Chinese)
(齐开元, 韩燕波, 赵卓峰, 等. 支持高并发数据流处理的 MapReduce 中间结果缓存[J]. 计算机研究与发展, 2013, 50(1): 111-121)

[13] Gibson G. Object storage architecture [EB/OL]. 2009 [2014-01-18]. http://www.panasas.com

[14] Braam P. Lustre file system [EB/OL]. 2005 [2014-01-18]. http://www.lustre.org

[15] Bronevetsky G, Moody A. Scalable I/O systems via node-local storage: Approaching 1TB/sec file I/O, LLNL-TR-415791 [R]. Livermore, CA: Lawrence Livermore National Laboratory, 2009: 1-6

[16] Zhou Enqiang, Lu Yutong, Shen Zhiyu. Implementation of checkpoint system towards large scale parallel computing [J]. Journal of Computer Research and Development, 2005, 42(6): 987-992 (in Chinese)
(周恩强, 卢宇彤, 沈志宇. 一个适合大规模集群并行计算的检查点系统[J]. 计算机研究与发展, 2005, 42(6): 987-992)

[17] Seelam S, Kerstens A, Teller P J. High Performance Computing and Communications [M]. Berlin: Springer, 2007: 718-731

[18] Lofstead J, Polte M, Gibson G, et al. Six degrees of scientific data: Reading patterns for extreme scale science IO [C] //Proc of the 20th Int Symp on High Performance Distributed Computing (HPDC 2011). New York: ACM, 2011: 49-60

[19] Chen Y. Towards scalable i/o architecture for exascale systems [C] //Proc of the 2011 ACM Int Workshop on Many Task Computing on Grids and Supercomputers. New York: ACM, 2011: 43-48

[20] Varki E, Merchant A, Xu J, et al. Issues and challenges in the performance analysis of real disk arrays [J]. IEEE Trans on Parallel and Distributed Systems, 2004, 15(6): 559-574

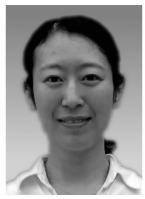
[21] Wong T M, Wilkes J. My cache or yours?: Making storage more exclusive [C] //Proc of the 2002 USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2002: 161-175

[22] Lu Kai, Jin Shiyao, Lu Xicheng. Properly greedy cache prefetch integrated algorithm in the parallel file system [J]. Chinese Journal of Computers, 1999, 22(11): 1172-1177 (in Chinese)
(卢凯, 金士尧, 卢锡城. 并行文件系统中适度贪婪的 cache 预取一体化算法[J]. 计算机学报, 1999, 22(11): 1172-1177)

[23] Vilayannur M, Nath P, Sivasubramaniam A. Providing tunable consistency for a parallel file store [C] //Proc of the 3rd USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2005: 1-2

Zhou Enqiang, born in 1972. Received his MSc degree from the National University of Defense Technology. Associate professor. His main research interests include high performance computing, parallel I/O and network storage system.

Zhang Wei, born in 1982. Received his PhD from the National University of Defense Technology. Assistant professor. His main research interests include high performance computing, parallel I/O and network storage system.



Lu Yutong, born in 1969. Received his PhD from the National University of Defense Technology. Professor. Member of China Computer Federation. Her main research interests include high performance computing and fault tolerant computing.

Hou Hongjun, born in 1972. Senior engineer in the Bureau of Geophysical Prospecting, China National Petroleum Corporation. His main research interests include seismic data processing, big data analysis.

Dong Yong, born in 1979. Received his PhD from the National University of Defense Technology. Associated professor. Member of China Computer Federation. His main research interests include high performance computing, parallel I/O and network storage system.