doi:10.3772/j.issn.1002-0470.2017.06.005

基于非易失存储器件的内存键值存储系统的性能研究①

魏 巍②[*,**,***] 蒋德钧[*,**] 熊 劲[*,**] 陈明宇[*,**]

([*]中国科学院计算机体系结构国家重点实验室 北京 100190)

([**]中国科学院计算技术研究所 北京 100190)

([***]中国科学院大学 北京 100049)

摘 要 分析了互联网应用为满足后端存储系统高性能要求而引用的内存键值存储系统的应用特点,指出:为了永久保存数据,这些系统还需要在后端将数据从易失性的内存拷贝到慢速的非易失存储设备中;将新型的非易失存储器件(NVM)引入内存键值存储系统可减少其性能开销;根据 NVM 的特征,内存键值系统可采用两种架构:将 NVM 替代磁盘作为二级存储设备和将 NVM 替代 DRAM 直接作为主存储设备。基于上述分析,实现了两种 NVM 架构的内存键值存储系统,并通过实验分析,总结出了内存键值存储系统选择 NVM 架构的原则,这些原则可有效指导内存键值存储系统在采用当前以及未来 NVM 器件时,对架构的选择。其次,还通过理论和实验分析,得出了不同架构下的内存键值系统在软件层的主要开销,指出了未来针对这些系统的软件设计的优化方向。

关键词 内存键值存储系统,非易失存储器件(NVM),持久内存,性能分析,数据持久化机制

0 引 言

互联网应用(例如电子商务、搜索引擎等)往往需要能快速地从海量数据中搜索和存储数据,及时地响应海量用户的并发请求。然而,基于慢速的磁盘设备的传统存储系统很难满足这些应用的需求。为此,当前很多互联网公司都在后端部署内存键值存储系统(in-memory key-value stores)。例如,目前已经超过 58 个互联网网站(如 Twitter, Github, StackOverflow 等)在后端采用 Redis[1] 作为存储系统的一部分[2]。

内存键值存储系统将数据直接存储在内存中,避免了应用运行时对慢速的二级存储设备(如磁盘、固态磁盘等)的访问。然而,为了能永久保存数据,当前内存键值存储系统需要在后端将数据从易失性的内存拷贝到非易失存储设备中。由于当前二级存储设备速度较慢,这种持久化操作严重影响了快速的内存键值存储系统的性能[3-14]。幸运的是,新型非易失存储器件(non-volatile memory, NVM)已经被广泛研究。概况来讲,NVM 具有扩展性高、延迟接近动态 RAM(dynamic random access memory, DRAM)、能够字节粒度寻址且能长时间保存数据的优势。鉴于 NVM 的优势,可将 NVM 引入内存键值存储系统解决数据移动影响性能的问题。

根据 NVM 的特征,将 NVM 引入内存键值存储系统可选择两种不同的架构。第一种架构利用 NVM 低延迟的特征,将 NVM 挂载在内存总线上替代磁盘(或固态硬盘(SSD))作为二级存储设备(以下称为存储架构)。另一种架构直接将 NVM 作为

① 863 计划(2015AA015303),国家自然科学基金(61502448, 61521092),山东省计算机网络重点实验室开放基金(SDKLCN-2013-01)和山东省自然科学基金(ZR2016FM41)资助项目。

② 女,1989 年生,博士;研究方向:基于新型非易失存储介质的内外存管理系统;E-mail:weiwei01@ict.ac.cn

(收稿日期:2017-01-27)

持久化内存替代 DRAM(以下称为内存架构)。

目前,很多工作分别研究这两种 NVM 架构的软件系统设计。研究 NVM 存储架构的工作主要集中在针对 NVM 特征设计高效的文件系统。BPFS[3] 是最早的针对挂载在内存总线上的 NVM 设备的文件系统。根据 NVM 的速度显著高于磁盘的特征,BPFS 删除了页缓存机制和合并小粒度随机写为大粒度顺序写的机制。SCMFS[4] 更加深入地考虑传统文件系统中不适合 NVM 特征的冗余机制。它构建在虚拟地址空间上,借用传统内存系统的分配机制和页表映射机制,避免了因传统文件系统复杂的分配机制而带来的开销。Aerie[5] 提供了创建在用户态上的可信的文件系统服务,使得大部分访问操作可不必进出内核,减少开销。最近提出的 PMFS[6] 不仅实现了上述文件系统的优化目标,还针对 NVM 特征实现了轻量级日志机制,进一步提高了 NVM 文件系统的性能。

针对 NVM 内存架构的工作主要集中在设计高效的持久化内存对象系统。早期的持久化内存对象系统(如 Mnemosyne[7],NV-Heap[8])包含两层:上层对应用提供类似传统内存接口的持久化内存接口(如 pmalloc[7],pmmap[7] 等)用于直接持久化数据;底层依然采用文件系统管理 NVM 的物理空间。这类系统使上层应用能直接访问 NVM 中的持久化数据,不需要通过系统调用进入文件系统。但是这类系统的底层依然包含文件系统,导致两个系统(内存系统和文件系统)同时管理 NVM 的数据,开销较大。因此,近期的持久化内存对象系统(如 pVM[9],HEAPO[10])删除了文件系统,直接采用内存对象系统统一管理 NVM 虚拟空间和物理空间。这类系统在传统易失性内存管理系统的基础上增加了保证 NVM 数据和空间一致性的机制。

综上所述,当前研究工作主要集中在针对 NVM 的两种架构设计高效的软件系统。与这些工作不同,鉴于内存键值存储系统可分别采用这两种 NVM 架构,本文主要研究选择哪种架构可使内存键值存储系统获得更高的性能。为此,本研究首先调研了目前研究NVM的主要工作[6,11-21],从中总结出了当前 NVM 的具体参数特征范围,为实验中 NVM 的参数设定提供依据。其次,在理论上分别分析采用不同架构的内存键值存储系统的软件开销来源。然后,根据最新的研究工作分别实现了采用 NVM 存储架构的 Redis 系统和采用 NVM 内存架构的 Redis 系统。其中,针对 NVM 存储架构,又分别实现了采用命令日志的方法和采用检查点方法两种不同数据持久化机制的内存键值系统。综上,本研究共实现了三种基于 NVM 的内存键值存储系统,并通过实验分析在不同负载和不同 NVM 配置下这三个系统的性能,总结出在具体场景中,内存键值存储系统应采用何种架构和何种数据持久化机制可获得高性能的原则。这些原则可有效指导内存键值存储系统在采用当前以及未来 NVM 器件时,对架构以及数据持久化机制的选择。最后,通过实验分析出不同架构的内存键值系统在软件层上的主要开销,指出未来针对这些系统的软件设计的优化方向。

1 新型非易失存储器件

当前工艺界正在广泛研究各种新型非易失存储器件。例如,三星、英特尔和镁光从 2006 年开始分别研制基于相变存储器[15](phase-change-memory,PCM)的存储器件,并随后发布了几款器件原型[16]。松下、惠普、镁光和索尼等公司从 2008 年开始发布基于阻变式存储器(resistive random access memory,ReRAM[17])的存储器件原型[18]。英特尔和镁光在 2015 年联合发布了 3D xPoint[20] 器件。此外,Qualcomm、Crossbar、富士通、中芯国际等国内外公司均在研发自己的非易失存储器件(NVM)。

总体来讲,这些新型 NVM 的共同特征是扩展性高、延迟接近动态 RAM(DRAM)、能够字节粒度寻址且能长时间保存数据。本研究调研了目前研究 NVM 的主要工作,从中总结出了当前 NVM 的具体参数特征范围。与 Flash 相比,NVM 的读、写速度最高分别快 20、1000 倍,且能够字节粒度寻址。另一方面,与 DRAM 相比虽然 NVM 的写速度慢 5 ~ 10 倍,但是其密度高了 4 ~ 8 倍,且不需要刷新操作。

表 1　不同器件的特征[6,11-21]

	密度	读延迟	写延迟	字节粒度寻址	非易失
DRAM	1x	1x	1x	是	否
Flash	4x	40x	5000x	否	是
NVM	4x ~ 8x	2x ~ 3x	5x ~ 10x	是	是

2　基于 NVM 的内存键值存储系统

鉴于 NVM 的优势,可将 NVM 引入内存键值存储系统解决数据移动影响性能的问题。本节首先介绍基于 NVM 的内存键值存储系统的两种架构。

根据 NVM 的特征,将 NVM 引入内存键值存储系统可选择两种不同的架构。如图 1(a)所示,第一种架构利用 NVM 低延迟的特征,将 NVM 挂载在内存总线上替代磁盘(或 SSD)作为二级存储设备(以下称为存储架构)。在这种架构下,内存键值存储系统依然利用 DRAM 作为主存储设备,只是在后台将数据从 DRAM 拷贝到 NVM 以保证数据的持久性。该架构有两个好处:(1)利用 NVM 显著提高二级存储设备的性能,缩短数据移动的时间;(2)兼容当前内存键值存储系统,不需要代码修改。但是,这种架构没有避免数据移动,以及由数据移动带来的数据形式转换、元数据冗余(需要内存系统和文件系统)等软件开销。

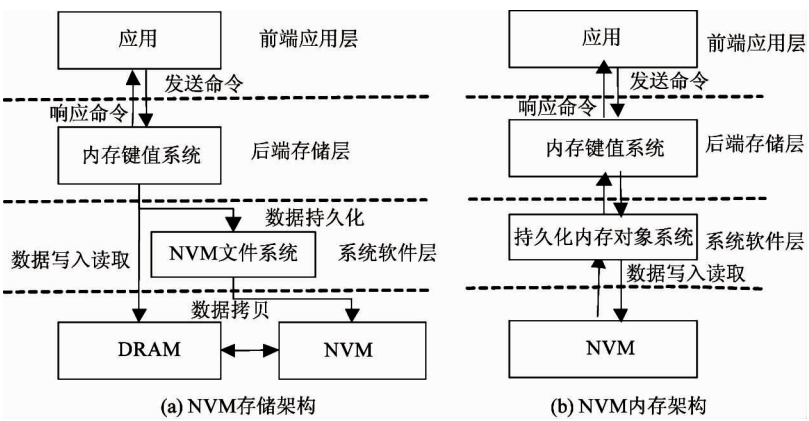

图 1　两种基于 NVM 的内存键值系统架构

如图 1(b)所示,另一种架构直接将 NVM 作为持久化内存替代 DRAM(以下称为内存架构)。在这种架构下,内存键值存储系统将数据直接存储到 NVM 中。因为 NVM 具有非易失特性,这种架构完全避免了数据移动,也因此避免了数据移动带来的软件开销。但是,这种架构直接将数据存储在延迟较高(与 DRAM 相比)的 NVM 上,同样影响性能。

下面,本文将从理论上分析上述两种内存键值存储系统的主要开销来源,为实验分析提供指导。

2.1　NVM 存储架构的内存键值存储系统软件开销

在这类系统中,NVM 替代磁盘作为二级存储设备(如图 1(a)所示)。内存键值存储系统将数据全部存储在 DRAM 构成的内存中。为响应前端应用发送的存储/读取命令,内存键值存储系统只需要在 DRAM 中进行存储/读取操作。这些操作在硬件上只涉及 DRAM,在软件上只涉及系统软件层的内存系统(即库层、内核层的内存管理系统),因此延迟较低。然而,为了保证数据的持久性,这类系统需要在后端将数据从易失性的 DRAM 拷贝到非易失的 NVM 上。当前,主要有两种方法执行持久化操作。

如图 2 所示,一种方法(以下简称命令日志)在每次执行写命令时,将写命令追加写入日志文件。当系统宕机重启后,通过读取日志文件并重新执行里面的命令来恢复数据。可见,该方法的主要开销是"将写命令追加写入日志文件"这一操作。如图 2 所示,该操作主要包含三个步骤:将命令追加写入内存缓冲区、将内存缓冲区写入文件以及将文件同步到 NVM 上。在后两个步骤中,内存键值存储系统

要借助文件系统来完成。传统文件系统为了隐藏磁盘的高延迟,通常在内存中建立页缓存。因此,内存键值存储系统调用 write 接口将缓冲区内容写入文件后,还需要调用 fsync 操作保证文件数据从页缓存同步到 NVM 上。由于和磁盘/SSD 相比,NVM 速度较高,当前基于 NVM 的文件系统通常删除页缓存,调用 write 时直接将数据写入到 NVM 上。所以,在这类系统中,不需要再执行 fsync 操作。综上,这类系统的持久化操作主要开销来源如表 2 所示。

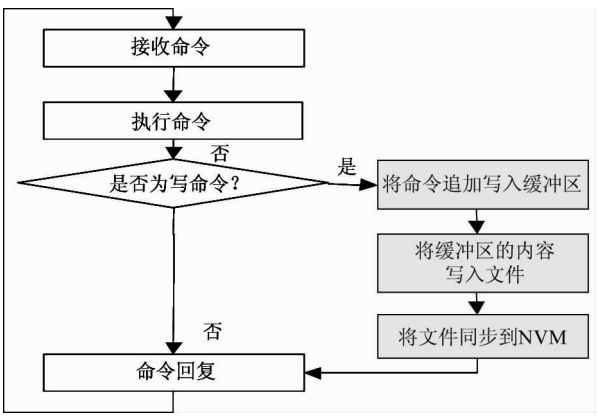

图 2　写命令日志方法下持久化操作

如图 3 所示,另一种持久化操作的方法是定时建立检查点(即 checkpoint),将内存键值存储系统的数据镜像全部写入到文件上。当系统宕机重启后,通过读取检查点文件可将内存键值存储系统恢复到建立检查点的时刻。在建立检查点之后的数据更新则无法恢复。当前的内存键值系统可利用多线程机制在后台执行 checkpoint,此过程不会中断对前端应用的服务[22]。该方法的主要开销在"建立检查点"这一操作。如图 3 所示,该操作主要包含三个步骤:将内存键值存储系统的内存数据进行形式转换(即序列化)、将转换后的数据写入镜像文件以及将文件同步到 NVM 上。由前所述,在基于 NVM 的文件系统中不需要同步操作。因此,采用建立检查点的系统的主要开销来源如表 2 所示。

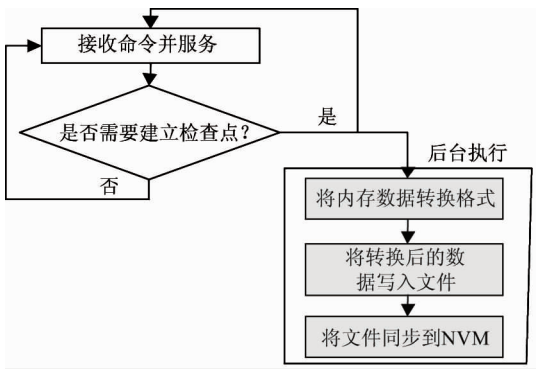

图 3　检查点方式下持久化操作

表 2　NVM 存储架构的内存键值存储系统的开销来源

持久化操作方法	步骤	开销来源
写命令日志方式	将写命令写入缓冲区(write _ to _ buffer)	数据在 DRAM 中拷贝(memcpy)
	将缓冲区内容写入文件(write)	元数据加锁(metadata _ lock)
		元数据更新(metadata _ update)
		记日志(log)
		将数据从缓冲区(DRAM)拷贝到文件(NVM)(copy _ data)
		分配空间(page fault)
建立检查点方式	数据格式转换(serialization)	数据格式转换(serialization)
	将转换后的数据写入文件(write)	同"将缓冲区内容写入文件"

2.2　NVM 内存架构的内存键值存储系统软件开销

在这类系统中,NVM 直接替代 DRAM 作为内存键值存储系统的的主存储设备(如图 1(b)所示)。为响应前端应用发送的存储/读取命令,内存键值存储系统只需要在 NVM 中进行存储/读取操作。这些操作在硬件层只涉及 NVM,在软件层只涉

及持久化内存对象系统。鉴于 NVM 的特征和 DRAM 相似,该类内存键值存储系统不需要执行如 2.1 节所述的数据拷贝来保证持久化,但要容忍和 DRAM 相比 NVM 较慢的速度。

另外,虽然该类内存键值存储系统可避免持久化操作,但是为了宕机后能继续服务应用,还需要保证存储在 NVM 中的数据的宕机一致性[23]。为此,软件层上内存键值系统可依然采用写命令日志的方式;底层的持久化内存对象系统也需要记录元数据更新日志(即与文件系统层日志相同)。宕机重启后持久化内存对象系统通过日志恢复元数据,上层内存键值系统再通过写命令日志恢复其在 NVM 中的数据。与 2.1 所述的写命令日志操作不同,该类内存键值存储系统只需将写命令追加到从 NVM 内存中开辟的持久化缓冲区,无需再将数据写入文件。最后,与传统易失性内存系统不同,持久化内存对象系统需要保证 NVM 空间的一致性。为此,当前持久化内存对象系统[20,21]在每次分配或释放空间(包含虚拟、物理空间)时,将改变(如 VMA、页表、页描述符的改变)记录到在 NVM 中分配的特定区域。在宕机重启后,通过该区域数据恢复空间一致性。综上所述,该类内存键值存储系统在软件层上的开销来源如表 3 所示。

表 3　NVM 内存架构的内存键值存储系统开销来源

操作	涉及的硬件	开销来源
将写命令写入持久化缓冲区(write _ to _ buffer)	NVM	数据在 NVM 内存中拷贝(memcpy)
涉及元数据更新	NVM	日志(log)
涉及虚拟、物理空间的更新	NVM	分配虚拟空间时记录日志(alloc _ vma)
		分配物理页时记录日志(page fault)

3　实验方法

如前节所述,基于 NVM 的存储架构和内存架构各有优缺点,很难从理论上直接分析出内存键值存储系统选择何种架构可获得更高的性能。为此,本文将通过实验评测得出结论。本研究选择当前应用较为广泛的内存键值系统 Redis 和两种先进的软件管理系统 PMFS[6]、pVM[9],实现了基于 NVM 存储架构的 Redis 系统(Redis + PMFS)和基于 NVM 内存架构的 Redis 系统(Redis + pVM)。其中,基于 NVM 存储架构的 Redis 系统,本文又分别实现了采用命令日志的方法和采用检查点的方法。因此,本文共评测了以下 3 种系统:(1)基于 NVM 存储架构利用日志持久化的 Redis 系统(称为 Storage-Log,简称 SL 系统);(2)基于 NVM 存储架构利用检查点进行持久化的 Redis 系统(称为 Storage-Checkpoint,简称 SC 系统);(3)基于 NVM 内存架构的 Redis 系统(称为 PM-Log,简称 PL 系统)。

实验机器采用英特尔双路至强 E5 v3 平台,共 24 个核。该机器为 NUMA 架构,共两个内存节点,每个内存节点的容量为 8GB。其中与运行 Redis 的核在同一 socket 的内存节点作为 DRAM,另一个远端节点被模拟为 NVM。本文利用英特尔的性能监控器监测运行 Redis 的核对远端内存节点发送 load 指令的次数。之后,每隔固定时间向该核发中断使其空转一段时间,从而模拟出 NVM 比 DRAM 高的读延迟。每次空转的时间根据这段时间内该核的读访问次数以及模拟的 NVM 的延迟值算出。鉴于当前机器结构下有硬件 buffer 可以缓存 store 指令,本文没有模拟 NVM 的写延迟。相反,通过英特尔提供的寄存器,限制内存控制器内每秒执行命令的次数模拟 NVM 的低带宽。上述模拟 NVM 的方法与之前的研究工作中[3,4,9]采用的方法相同。根据表 1 显示的 NVM 特征,本文设置了 4 种 NVM 延迟(Latency,以下简称 L),分别是与 DRAM 延迟相同以及是 DRAM 延迟的 2 倍(2 × L)、3 倍(3 × L)和 4 倍(4 × L);设置了 4 种 NVM 带宽,分别是 DRAM 带宽(Bandwidth,以下简称 BW)的 100%、50%、20% 和 10%。通过组合共设置了 13 种 NVM 配置。

本文采用 YCSB[24]中 zipfian 分布的负载来测试这三种系统。如前节所述,三种系统的主要性能开销来源均与写命令相关,而与读命令不相关。因此,本文设计了以读为主(90% 读 10% 写)和以写为主

(10% 读 90% 写)两种负载。为了消除网络对结果的影响,将 YCSB 与 Redis 运行在同一台机器,但不同的 socket 中(避免影响模拟 NVM)。通过前期实验测试,研究发现在上述的机器环境下,当客户端为 8、总操作数为 1000 万时,三种系统性能达到峰值。因此,本文将正式实验时的客户端设置为 8,总操作数为 1000 万。

4 结果分析

本节主要分析实验结果。通过分析,本节将回答两个问题:(1)在不同 NVM 的配置下,采用哪种架构的内存键值存储系统性能较高?(2)三种内存键值存储系统的主要软件开销分别在哪里?

通常,在类似互联网这种交互式场景下的后台系统的性能评价指标有两个:吞吐率和每个请求的延迟。如前文所述,本文研究的内存键值存储系统被广泛布署到互联网网站的后台。因此,本文采用吞吐率和请求的平均延迟作为衡量三个系统性能的评价指标。具体来说:吞吐率指系统每秒钟完成的请求数,吞吐率越高,代表系统性能越高;请求的平均延迟指系统完成一个请求的平均延迟,平均延迟越低,代表系统性能越高。本文分别分析了在不同负载以及不同配置下,三个系统的吞吐率和请求的平均延迟。为了分析方便,本文将三个系统在不同 NVM 配置下的吞吐率和请求的平均延迟分别与理想系统(即采用 DRAM 且不保证数据持久化的系统)下的结果进行归一化处理。

4.1 读为主负载下性能分析

图 4 和图 5 分别显示了读为主负载下 NVM 的不同配置下三个系统的归一化的吞吐率和归一化的请求平均延迟(即与理想系统在读为主负载下的吞吐率和请求平均延迟的相对值)。

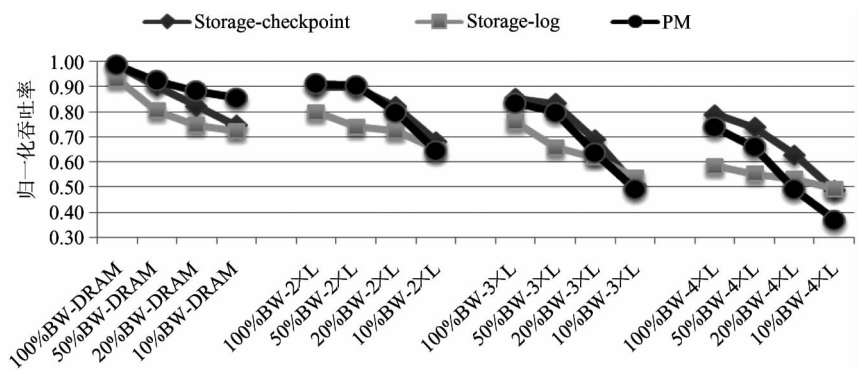

图 4 10% 写 90% 读负载下不同 NVM 配置下系统的吞吐率

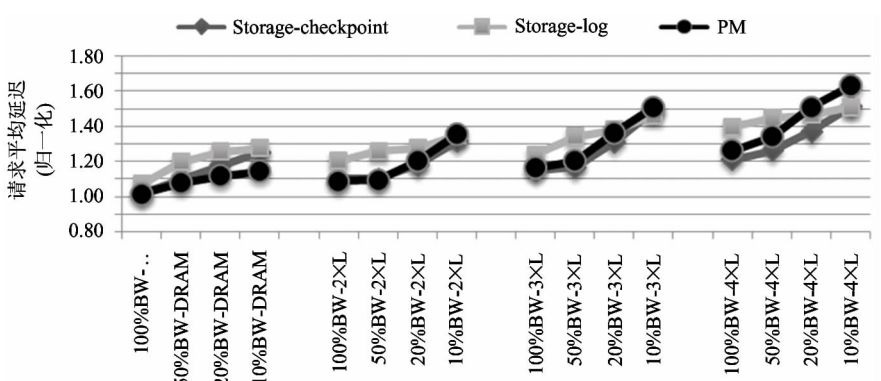

图 5 10% 写 90% 读负载下不同 NVM 配置下系统的请求平均延迟

首先分析在 NVM 配置与 DRAM 均相同时(即在理想硬件条件下)三个系统的性能,从而分析出在以读为主的负载下三个系统的软件实现对系统性能的影响。从图 4 中,发现在理想硬件配置下(即 NVM 的延迟和带宽均和 DRAM 相同),SC 系统、SL 系统和 PL 系统的吞吐率与理想系统的吞吐率相比,分别减少了 2%,7% 和 1%。类似地,图 5 显示,SC 系统、SL 系统和 PL 系统的请求平均延迟与理想系统下的平均延迟相比,分别增加了 1%,6% 和 0.5%。上述现象表明,在以读为主的负载下,三个系统的软件实现对性能的影响小。这是因为与理想系统相比,三个系统的软件开销来源于写操作(如第 2 节所述),对于读操作较重的负载的影响较小。

其次分析了不同 NVM 配置下,三个系统的性能,观察到如下现象。

现象 1:SC 系统在所有 NVM 配置下的性能都比 SL 系统高。如图 4 所示,SC 系统的吞吐率最高比 SL 系统提高 21%(此时 NVM 配置为 100% BW-4 × L),平均提高 8%。类似地,SC 系统的请求平均延迟最低比 SL 系统低 19%(此时 NVM 配置为 100% BW-4 × L),平均低 7%(图 5 所示)。出现上述现象的原因是:Redis 在后台执行检查点,不会中断服务;但若采用日志的方法,则每次执行写命令时,都需先写日志,这导致在 SL 系统中每次执行写命令的时间增加。因此 SL 系统性能比 SC 系统差。但是采用检查点的方法会丢失一段时间内的操作结果,而采用日志的方法可保证已执行完的操作结果不丢失。

现象 2:当 NVM 配置为:带宽大于等于 20% 且延迟小于等于 3 × L 时,PL 系统的性能比 SL 系统高。在上述的 NVM 配置范围内,PL 系统的吞吐率比 SL 系统最高可提高 17%(此时 NVM 配置为 50% BW-2 × L),平均提高 10%。类似地,在上述 NVM 配置范围内,PL 系统的请求的平均延迟比 SL 系统最低可减少 16%(此时 NVM 配置为 50% BW-2 × L),平均减少 6%。相反,当 NVM 的配置变为:带宽小于等于 10% 且延迟大于等于4 × L时,PL 系统的性能低于 SL 系统。例如,在 NVM 配置为 10% BW-4 × L时,PL 系统的性能与 SL 系统的性能差距最大。此配置下,PL 系统的吞吐率比 SL 系统降低 12%,相反 PL 系统的请求平均延迟比 SL 系统高 11%。上述现象出现的原因包含软件开销和 NVM 配置两个方面。如前节理论分析,采用 NVM 内存架构的系统的软件层开销低于采用 NVM 存储架构的系统,但要容忍与 DRAM 相比 NVM 较高的读、写延迟。

现象 3:PL 系统只在两个 NVM 的配置(20% BW-DRAM 和 10% BW-DRAM)下性能高于 SC 系统。在 NVM 配置为 10% BW-DRAM 时,PL 系统性能与 SC 系统相比优势最大。在此配置下,PL 系统的吞吐率比 SC 系统高 11%,请求平均延迟比 SC 系统低 9%。当 NVM 的配置变为带宽大于等于 50% 且延迟小于等于2 × L时,PL 系统的性能(指吞吐率和请求的平均延迟)和 SC 系统的性能持平。相反,当 NVM 的配置变为带宽低于 50% 或延迟大于2 × L时,PL 系统的性能低于 SC 系统。例如,在 NVM 配置为 20% BW-4 × L下,PL 系统的吞吐率比 SC 系统减少 14%,请求的平均延迟增加 10%。

根据上述观察到的现象 1 ~ 现象 3,本文提出对于以读为主的前端应用,为提高性能,内存键值系统可分为以下三种情况选择 NVM 架构:(1)当系统不需要保证每次已执行完的操作的结果持久化时,无论 NVM 的配置高低,都采用 NVM 存储架构并利用检查点的方法保证持久化;(2)当系统需要严格保证已执行完的操作结果不丢失时,对于 NVM 配置为带宽小于 20% 且延迟大于3 × L时,可采用 NVM 存储架构并利用 log 的方法保证持久化;(3)对于 NVM 配置为带宽大于等于 20% 且延迟小于等于 3 × L时,可采用 NVM 内存架构并利用 log 的方法保证持久化。

4.2　写为主的负载下性能分析

图 6 和图 7 分别显示了写为主负载下 NVM 的不同配置下三个系统的归一化吞吐率和归一化的请求平均延迟(即与理想系统在写为主负载下的吞吐率和请求平均延迟的相对值)。

与 4.1 小节类似,首先分析在 NVM 配置与 DRAM 均相同时三个系统的性能,从而分析出在写为主负载下三个系统的软件实现对系统性能的影

响。从图 6 中,发现在 NVM 理想配置下,SC 系统、SL 系统和 PL 系统的吞吐率与理想系统相比,分别减少了 36%,41% 和 29%。类似地,图 7 显示,在 NVM 理想配置下,SC 系统、SL 系统和 PL 系统的请求平均延迟与理想系统相比,分别增加了 34%,36% 和 27%。上述现象表明,在以写为主的负载下,三个系统的软件实现对性能的影响较为明显。这与第 2 节所述的三个系统的软件开销来源于写操作的分析相符。本文将在 4.3 小节具体分析三个系统在以写为主的负载下的软件开销。

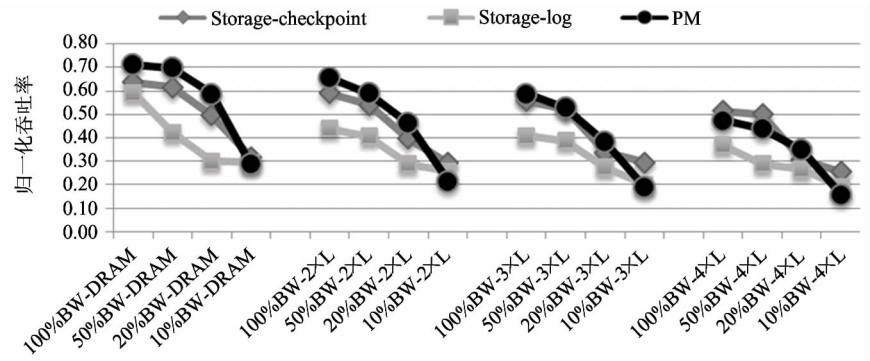

图 6 90% 写 10% 读负载下不同 NVM 配置下系统的吞吐率

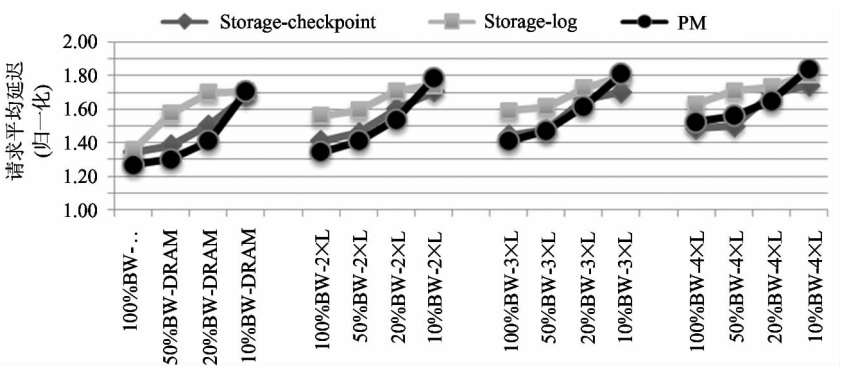

图 7 90% 写 10% 读负载下不同 NVM 配置下系统的请求平均延迟

其次分析了 NVM 不同配置下,三个系统的性能。从分析中观察到如下现象。

现象 4: 与以读为主的负载下现象类似,SC 系统在 NVM 所有配置下的性能都比 SL 系统高。整体看,如图 6 所示,SC 系统的吞吐率最高比 SL 系统提高 21%(此时 NVM 配置为 50% BW-4 × L),平均提高 9%。如图 7 所示,SC 系统的请求的平均延迟与 SL 系统相比,最高减少了 18%(此时 NVM 配置为 50% BW-4 × L),平均减少 7%。

现象 5: 当 NVM 配置为带宽大于等于 20% 时,PL 系统的性能高于 SL 系统。在上述 NVM 配置下,PL 系统的吞吐率与 SL 系统相比,最高提高 29%(此时 NVM 配置为 20% BW-DRAM),平均提高 10.2%。类似地,在上述 NVM 配置下,PL 系统的请求平均延迟与 SL 系统相比,最高减少 25%(此时 NVM 配置为 20% BW-DRAM),平均减少 8%。相反,当 NVM 配置为带宽小于 10% 时,PL 系统性能低于 SL 系统。同时,随着 NVM 延迟的增加,PL 系统的性能和 SL 系统性能之间的差距更加明显。例如,在 NVM 配置为 10% BW-4 × L下,PL 系统的吞吐率比 SL 系统减少 6%,但请求的平均延迟增加 3%。

现象 6: 在 NVM 配置为带宽大于等于 20% 且延迟小于等于 3 × L时,PL 系统的性能优于 SC 系统。例如,在 NVM 配置为 50% BW-2 × L时,PL 系统的吞吐率比 SC 系统提高 5%,请求的平均延迟减少 4%。上述现象表明:当 NVM 配置较优时,即使利用日志

的方法保证持久化,采用 NVM 内存架构的系统性能也优于采用 NVM 存储架构系统(采用检查点的方法)的性能。相反,当 NVM 配置变为带宽小于等于 10% 或延迟大于等于4 × L时,PL 系统的性能低于 SC 系统。例如,在 NVM 配置为 10% BW-4 × L配置下,PL 系统的吞吐率比 SC 系统低 10%,但请求的平均延迟比 SC 系统高 9%。

根据上述观察到的现象 4 ~ 现象 6,本文提出对于以写为主的前端应用,为提高性能,内存键值系统可分为以下三种情况选择 NVM 架构:(1)当 NVM 的配置为带宽大于等于 20% 且延迟小于等于 DRAM 的 3 倍时,可采用 NVM 内存架构,相反,当 NVM 的配置为带宽小于 10% 或延迟大于等于 DRAM 的 4 倍时,可采用 NVM 存储架构;(2)如果系统不需要保证每次操作均持久化时,可采用检查点的方法;(3)否则采用日志的方法。

4.3　软件开销分析

如 4.2 节所述,在以写为主的负载下,三个系统的软件实现对性能的影响较大。结合第 2 章所述的理论分析和实验结果,下文详细分析不同系统下的软件开销。图 8 分别显示了在以写为主的负载以及 NVM 理想配置下(即消除硬件对性能的影响),三个系统中与持久化有关的软件操作(具体操作名称见表 2 和表 3)的时间占据系统运行时整个 CPU 运行周期的百分比。如图 8 所示,SL 系统中元数据更新的操作所占比例最大,为 13.62%。这是因为 SL 系统对 NVM 文件是大量的小粒度写,对元数据的操作开销较为明显。因此,未来在针对 SL 系统的底层文件系统设计时,可优化小粒度写机制,减少元数据更新开销。在 SC 系统中序列化的开销最大,占据整个 CPU 运行周期的 11.86%。这是因为系统需要将内存中的所有数据结构序列化,当数据量较多时,占用时间较多。因此,未来针对 SC 系统可优化检查点机制,减少每次序列化的数据量。在 PL 系统中,page fault 的开销最多,占据整个 CPU 运行周期的 12.82%。这是因为在每次分配新页(page fault)时,持久化内存对象系统都需要额外将新页的信息记录到在 NVM 中分配的特定区域,用于宕机恢复。这个操作延长了每次分配页的时间。又因为在写操作较多的负载下,page fault 发生的次数较多。所以,在整体上 page fault 的开销较高。未来针对 PL 系统可优化空间数据持久化机制,减少其对访问路径上的开销。

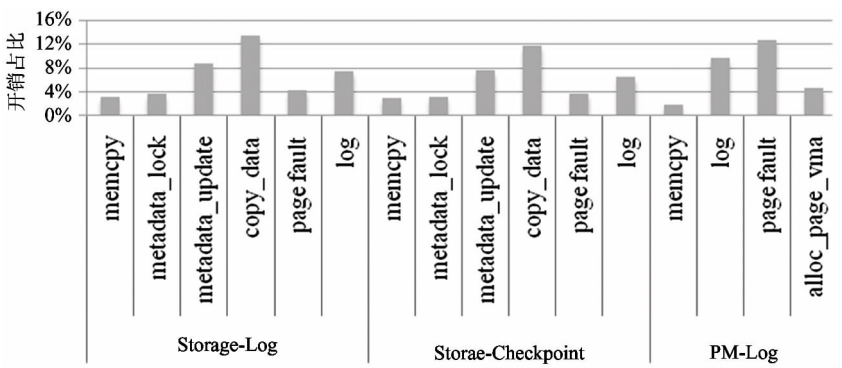

图 8　90% 写 10% 读负载下三个系统软件开销分析

5　结　论

本文实现了基于 NVM 的两种不同架构的内存键值存储系统:基于 NVM 的存储架构和基于 NVM 的内存架构。其中,基于 NVM 的存储架构,本文又分别实现了采用命令日志的方法和采用检查点方法两种不同的保证一致性机制的内存键值系统。综上,本文共实现了三种基于 NVM 的内存键值存储系统并通过实验分析,总结出了在不同负载和不同 NVM 配置下内存键值存储系统选择 NVM 架构以及数据持久化机制的原则。这些原则可有效指导内存

键值存储系统在采用当前以及未来 NVM 器件时，对架构和数据持久化机制的选择。其次，本文还通过理论和实验分析给出了这三种内存键值系统在软件层上的主要开销，指出了未来针对这三种系统的软件层的优化方向。

后续研究是优化持久化内存对象系统下的宕机一致性保证机制，并将其应用到内存键值系统中，进一步减少系统的软件开销。

参考文献

[1] Redis. Introduction to redis. https://redis.io: Redis, 2009

[2] Redis. Who's using redis? https://redis.io/topics/whos-using- redis/who39s-using-redis: Redis, 2017

[3] Condit J, Nightingale E B, Frost C, et al. Better I/O through byte-addressable, persistent memory. In: Proceedings of ACM Symposium on Operating Systems Principles, Montana, USA, 2009. 133-146

[4] Wu X, Reddy A L N. SCMFS: a file system for storage class memory. In: Proceedings of Conference on High Performance Computing Networking, Storage and Analysis, Seattle, USA, 2011. 1-11

[5] Volos H, Nalli S, Panneerselvam S, et al. Aerie: flexible-file-system interfaces to storage-class memory. In: Proceedings of the 9th European Conference on Computer Systems, Amsterdam, Netherlands, 2014. 1-14

[6] Dulloor S R, Kumar S, Keshavamurthy A, et al. System software for persistent memory. In: Proceedings of the 9th European Conference on Computer Systems, Amsterdam, Netherlands, 2014. 15:1-15:15

[7] Volos H, Tack A J, Swift M M. Mnemosyne: lightweight persistent memory. In: Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems, Newport Beach, USA, 2011. 91-104

[8] Coburn J, Caulfield A M, Akel A, et al. NV-Heaps: making persistent objects fast and safe with next-generation, non-volatile memories. *ACM SIGPLAN Notices*, 2012, 47(4):105-118

[9] Kannan S, Gavrilovska A, Schwan K. pVM: persistent virtual memory for efficient capacity scaling and object storage. In: Proceedings of European Conference on Computer Systems, London, UK, 2016. 13:1-13:16

[10] Wang H T, Jung J, Won Y. HEAPO: Heap-Based Persistent Object Store. *ACM Transactions on Storage*, 2014, 11(1): 3:1-3:21

[11] Zhang Y, Swanson S. A study of application performance with non-volatile main memory. In: Proceedings of International Conference on Massive Storage Systems and Technology, Santa Clara, USA, 2015. 1-10

[12] Chen F, Mesnier M P, Hahn S. A protected block device for persistent memory. In: Proceedings of International Conference on Massive Storage Systems and Technology, Santa Clara, USA, 2014. 1-12

[13] Nazarian H. Crossbar resistive memory: The future technology for nand flash. http://www.crossbar- inc.com/assets/resource/whitepaper/Crossbar-ReRAM-Technology-Whitepaper.pdf: Crossbar, 2017

[14] Qureshi M K, Srinivasan V, Rivers J A. Scalable high performance main memory system using phase-change memory technology. *ACM SIGARCH Computer Architecture News*, 2009, 37(3): 24-33

[15] Nirschl T, Philipp J B, Happ T D, et al. Write strategies for 2 and 4-bit multi-level phase-change memory. In: Proceedings of IEEE International Electron Devices Meeting, Washington, USA, 2007. 461-464

[16] Wikipedia. Phase-change memory. https://en.wikipedia.org/wiki/Phase-change_memory#cite_note-samsung-17: Wikipedia, 2017

[17] Kawahara A, Azuma R, Ikeda Y, et al. An 8 Mb multi-layered cross-point ReRAM macro with 443 MB/s write throughput. *IEEE Journal of Solid-State Circuits*, 2013, 48(1): 178-185

[18] Wikipedia. Resistive random-access memory, https://en.wikipedia.org/wiki/Resistive_random-access_memory#Crossba: Wikipediar, 2017

[19] Shiga H, Takashima D, Shiratake S, et al. A 1.6 GB/s DDR2 128 Mb chain FeRAM with scalable octal bitline and sensing schemes. *IEEE Journal of Solid-State Circuits*, 2010, 45(1): 142-152

[20] Intel. Intel® Optane™ Technology — Breakthrough Memory Technology. http://www.intel.com/content/www/us/en/architecture-and-technology/optane-technology-unveiled-video.html: Intel, 2016

[21] Yang J J, Strukov D B, Stewart D R. Memristive devices

for computing. *Nature Nanotechnology*, 2013, 8(1): 13-24

[22] Redis. Redis persistency. https://redis.io/topics/persistence:redis: Redis, 2017

[23] Ren J, Zhao J, Khan S, et al. ThyNVM: enabling software-transparent crash consistency in persistent memory systems. In: Proceedings of International Symposium on Microarchitecture, Hawaii, USA, 2015. 672-685

[24] Cooper B F, Silberstein A, Tam E, et al. Benchmarking cloud serving systems with YCSB. In: Proceedings of ACM Symposium on Cloud Computing, New York, USA, 2010. 143-154

Study of the performance of in-memory key-value stores with non-volatile memory

Wei Wei[*,**,***], Dejun Jiang[*,**], Jin Xiong[*,**], Mingyu Chen[*,**]

([*]State Key Laboratory of Computer Architecture, Chinese Academy of Sciences, Beijing 100190)

([**]Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

([***]University of Chinese Academy of Sciences, Beijing 100049)

Abstract

The application characteristics of in-memory key-value stores applicable to Internet applications to meet the high performance requirement of the back end storage system was analyzed. To avoid data losses in volatile DRAM memory, existing in-memory stores persist data into slow disk-based storage. Emerging non-volatile memories (NVMs), such as PCM and ReRAM, can sit on the memory bus due to their similar performance as DRAM. Moreover, the inherent data durability in NVMs helps to accelerate data persisting when deploying in-memory key-value stores on NVMs. According to NVMs' characteristics, in-memory key-value stores can exploit two architectures, using NVM to replace Disk or using NVM to replace DRAM. Based on the above analysis, two types of NVM-based in-memory key-value stores were implemented, and then the experiments were conducted and some principles that effectively guide to choosing an appropriate architecture when in-memory key-value stores use different NVM devices. Also, the main software overhead in the two types of stores was concluded, and some future optimizing suggestions was given.

Key words: in-memory key-value stores, non-volatile memory (NVM), persistent memory, performance analysis, data persistent mechanimses