图划分在混合内存系统的实现与性能优化

李 琪[1)] 钟 将[1)] 李 雪[2)]

[1)](重庆大学计算机学院 重庆 400030)

[2)](昆士兰大学信息技术与电子工程系 布里斯班 4072 澳大利亚)

摘 要 图划分是大图数据并行计算的基础,目前主要采用分布式算法实现大图划分.非易失存储器(Non-Volatile Memory,NVM)速度接近动态随机存储器(Dynamic Random Access Memory,DRAM),且具有低功耗、高密度、低时延等优点,本文针对分布式图划分算法难以分析和调试等问题,设计了基于混合内存的单机图划分算法框架.作者提出了基于邻边结构的图划分结果动态缓存管理策略(AeFdy),以提高缓存区邻居节点的搜索效率.在 17 种真实应用数据上的实验结果表明,采用新方法的平均图划分速度是基于邻点结构算法的 4.9 倍.本文还针对 NVM 寿命有限的问题,设计了基于内存页读写特征的迁移算法,实现了 NVM 写操作受限条件下的迁移优化方案.相对于 Linux Swap、M-CLOCK、Dr.Swap 混合内存管理策略,使用 AeFdy 策略的性能分别提升了 128.5%、87.4%与 50.4%.仿真实验结果表明,本文设计的混合内存管理方法实现了 NVM+DRAM 高效协同.

关键词 复杂网络;非易失存储器;流划分;混合内存;内存计算;平衡图划分

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2019.02481

Graph Partitioning in the Implementation and Performance Optimization of A Hybrid Memory System

LI Qi[1)] ZHONG Jiang[1)] LI Xue[2)]

[1)](*College of Computer Science, Chongqing University, Chongqing* 400030)

[2)](*School of Information Technology and Electrical Engineering, University of Queensland, Brisbane*, 4072 *Australia*)

Abstract The tremendous scale of modern graph datasets has rapidly increased the demand for efficient algorithms for graph analysis. A standard approach distributes the graph over a cluster of computer nodes, but the performing computations on a distributed graph is expensive if the large amount of data has to be moved. Graph partitioning is the precondition of distributed graph computing framework, which is a key problem in improving the performance of distributed graph computing. Streaming graph partitioning is more efficient compared with offline partitioning, it has been developed continuously in the application of graph partitioning in recent years. Because of the limitation of memory capacity, the single commodity type computer is difficult to partition and optimize the massive graphs. Existing methods mainly use distributed cluster to process these large graph partitioning, while distributed computational resources have become more accessible, developing distributed graph partitioning algorithms still remains challenging, especially to non-experts. NVM storage has the advantages of low power consumption, high density, low latency and byte-addressable, which is the construction of high performance storage system and

收稿日期:2017-07-31;录用日期:2018-06-28.本课题得到国家重点研发计划(2017YFB1402400)、重庆市社会事业与民生保障科技创新专项(cstc2017shmsA0641)、国家"八六三"高技术研究发展计划项目基金(2015AA015308)、重庆市重点产业共性关键技术创新专项(cstc2017zdcy-zdyxx0047)、重庆市技术创新与应用示范(产业类重点研发)项目(cstc2018jszx-cyzdX0086)、中央高校项目(2018CDYJSY-0055)资助.**李 琪**,博士研究生,主要研究方向为图挖掘、图计算.E-mail:liqi0713@foxmail.com.**钟 将**(通信作者),博士,教授,博士生导师,主要研究领域为数据挖掘、并行计算、自然语言处理.E-mail:zhongjiang@cqu.edu.cn.**李 雪**,博士,教授,博士生导师,主要研究领域为社交网络分析、分布式高速时变数据流的知识挖掘.

an important means to improve the performance big data system. But NVM storage also has some disadvantages, such as writing power consumption is higher than reading, write latency is higher than read, and the write counts of NVM is limited. The asymmetry problem about read and write in NVM should be paid more attention when using the hybrid NVM and DRAM memory. In this work, we explore to partition the large graphs under single compute node with hybrid NVM and DRAM memory. We propose a management strategy based on adjacent edge structure for dynamic cached data (AeFdy). This strategy converts the cached data structure from the adjacent vertex structure in the original streaming algorithms to the adjacent edge structure. The experimental results on 7 real-world graphs show that the average partitioning time of the new method is 4.9 times faster than that of original method. At the same time, the strategy evaluates the data pages in NVM and DRAM media by different models according to the characteristics of the streaming algorithm based on adjacent edge structure, places data pages in different memory media to reduce system migration operation times and improve the system performance. To demonstrate, the AeFdy strategy is simulated in the Linux kernel. Compared with the existing hybrid memory management strategy, such as Linux Swap, M-CLOCK and Dr. Swap, AeFdy improves the system performance by 128.5%, 87.4% and 50.4% respectively.

Keywords complex network; non-volatile memory; streaming partitioning; hybrid memory; memory computing; balanced graph partitioning

1 引 言

以动态随机存储器(Dynamic Random Access Memory,DRAM)为主存的存储系统存在两点不足:(1)DRAM 存储设备容量小[1-3].由于 DRAM 介质密度较低,其制程工艺也很难再有提升;(2)DRAM 介质的数据易失性.在使用闪存或磁盘等非易失性设备进行数据持久化时,内外存速度不匹配导致存储性能瓶颈.上述两个问题导致在进行大规模数据计算和实时分析时需要频繁地在主存和外存之间迁移数据,难以满足应用对系统的实时交互需要.

新型非易失性存储器(Non-Volatile Memory,NVM)[1,4-5]可以作为主存或外存来缓解传统存储体系结构的问题.(1)NVM 具有可字节寻址的特性,它可以直接挂载在内存总线上被访问,处理器可以通过 load/store 指令访存 NVM 上的数据[6-7],而不需要通过耗时的 I/O 操作来访存数据;(2)NVM 在访问时延和读写性能等方面与 DRAM 相近,静态功耗低于 DRAM[8];(3)NVM 具有非易失性的特点,它可以对数据进行持久化存储.但是 NVM 也存在读写不对称的特性,它的写性能较差,写功耗高,写操作次数有限.例如,PCM 设备的写次数通常只有 $10^8\sim10^9$[1,9].常用的非易失性存储介质主要包括相变存储器[4](Phase Change Memory,PCM)、电阻式/阻变存储器[10](Resistive Ram,RRAM)、石墨烯存储器[11](Graphene Memory,GM).不同存储介质特性比较见表 1.

表 1 几种存储介质的特性比较[6,21]

存储介质	非易失性	擦写单位	读时延/ns	写时延/ns	工艺制程/nm	特征尺寸	空闲功耗/(nJ/b)	写功耗/(nJ/b)	寿命
DRAM	否	位	<10	<10	20	$6\sim10F^2$	高	≈0.1	$>10^{15}$
PCM	是	位	10～100	20～120	≈5	$4\sim8F^2$	低	1	$10^8\sim10^9$
RRAM	是	位	10～50	10～50	≈11	$4\sim14F^2$	低	≈0.1	$10^8\sim10^{10}$
NAND	是	块	10^4	$10^5\sim10^6$	≈16	$4\sim11F^2$	低	0.1～1	$10^4\sim10^6$

图 1 中给出了 NVM 在计算机存储体系结构中的两类应用模式.一是 NVM 作为主存,包括使用 NVM 和 DRAM 混合内存[12-13]、单独使用 NVM 做内存[6,9,14]、使用 DRAM 做写缓存区等研究工作[15],还有一些研究将 NVM 作为交换分区[16-17],即内存的一个扩展;二是 NVM 作为存储级内存[18-20],如基于 NVM 的文件系统和数据库系统等.

另外,针对 NVM 写寿命的优化问题[16,22],通过

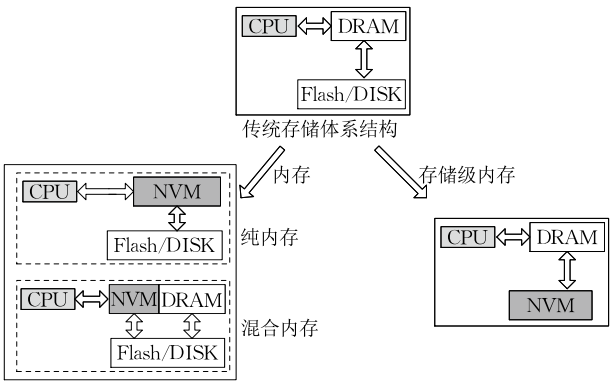

图 1　NVM 在存储体系结构中的应用

磨损均衡算法来优化 NVM 的写分布，减少 NVM 写操作次数，最终达到延长 NVM 使用寿命的目的.

如何利用 DRAM+NVM 混合内存系统实现大图数据高效划分是本文的主要研究内容. 已有相关工作中，文献[23]解决的是图计算框架 Graphlab，Galois 等在混合内存中的实现，文献[13,24]主要以图的存储数据结构为优化对象，提高宽度优先搜索(Breadth First Search，BFS)，以及图着色(graph coloring)算法在混合内存中的效率且减少系统能耗. 文献[25]主要利用 NVM 的低延迟性与高密度存储性对大图算法执行效率进行加速. 但是已有工作没有考虑到 NVM 寿命问题(即 NVM 写次数有限)，其次不同图算法优化方法不同，很难有通用的设计.

图划分是经典的组合优化问题[26]，广泛应用于图像分割[27]、数据挖掘[18]、VLSI 设计[28]等领域. 本文的研究动机主要有两点：(1) 以 DRAM 为主存的单计算节点容量有限，难以处理大规模图的划分问题. 例如由大脑神经构成的图包含 890 亿个顶点(神经细胞)及 100 万亿条边(树突)[25]. 全球最大的社交网络 Facebook 目前月活跃用户数量超过了 22 亿①，社交网络图有上万亿条边. 虽然可以利用分布式系统实现大图划分的问题，但是分布式图划分算法存在高网络时延以及难以分析和调试等问题，且单计算节点相比于分布式更容易管理；(2) NVM 读写操作不对称且写操作次数有限，需要设计优化调度 DRAM 和 NVM 存储资源. 图划分算法属于读写密集型计算，如果将 NVM 作为主存势必会减少 NVM 的使用寿命且大量的写操作也会降低系统性能. 考虑到 NVM 介质存储容量大，读写延迟低及 DRAM 介质寿命不受写次数的影响，所以本文采用 NVM+DRAM 的混合内存协同方案. 本文的主要贡献如下：

(1) 设计了基于混合内存的流式图划分框架. 提出了一种基于邻边结构的动态缓存数据管理机制(AeFdy)，该机制有效地管理划分过程中的缓存数据. 相对于现有的基于邻点结构的流算法，划分效率明显提升. 同时该机制不仅可以应用在线性贪婪流划分中，对其它流划分算法同样适用，如 Fennel 流算法，三角算法等，具体的流算法在文献[29]有详细的介绍.

(2) 设计了 NVM 和 DRAM 之间的优化调度机制. 该机制针对存储介质的不同特征，结合基于邻边结构流算法的划分过程对 NVM 及 DRAM 介质中的数据页分别使用不同的模型进行评估，将具有不同访问特征的数据页保存在合适的内存空间中，以减少系统的迁移操作次数，从而提升系统性能.

(3) 最后在 Linux 内核中对 AeFdy 机制进行了仿真实验. 并与其它内存管理机制分别做了对比，实验结果验证了 AeFdy 机制的有效性.

本文在第 2 节介绍现有的图划分算法，由于流式划分相较于离线划分的高效性，所以以线性权重贪婪流算法为例重点分析流式划分的过程并讨论其不足；第 3 节介绍邻边结构，给出其对应的缓存数据管理方案，并对复杂度进行分析；第 4 节分析内存数据页的访问特征. 结合基于邻点结构流式划分的过程，给出 NVM 及 DRAM 介质中的内存页之间的迁移优化方案；第 5 节给出实验结果并进行讨论分析；最后，在第 6 节总结本文的研究工作并对未来研究做出展望.

2　流式图划分模型

图的 k-划分属于 NP 完全问题[30-31]，通常很难在有限的时间内找到最优解. 从 20 世纪 90 年代初期至今，国内外研究者不断对图划分及其相关问题进行深入研究，提出了许多性能较好的离线划分算法. 有谱方法[32]、几何方法[33]、启发式方法[34]、多层划分方法[35-36]，这些算法主要是将图一次性载入内存，反复迭代运算，频繁访问顶点与边，由于较高的时间复杂度和对内存的容量要求，很难应用在大规模的图划分中. 例如最经典的 KL(Kernighan-Lin)[37]算法，复杂度为 $O(n^2 \log n)$，对于只有上万顶点的图，也很难处理. 随后出现了 KL 算法的改进版本——FM(Fiduccia- Mattheyse)[21]算法，虽然时

① https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/

间复杂度相对于 KL 有所降低，但是 FM 算法仍无法处理顶点数为百万以上的图，而且 FM 算法很容易陷入局部最优，划分效果很不稳定.

相较于离线划分，流式图划分算法只需访问一次图中所有的顶点，划分效率明显提升. 最新的 Fennel[38]算法在划分质量上接近甚至超过了 Metis[36]划分，在划分效率上，划分大约 14 亿条边的图 Twitter-2010 需要 42 min 左右，Metis 耗时超过了 8 个小时[39]. 而且流算法也能够有效处理动态的图数据，例如 Facebook，Skype 和 Twitter 每天都会有新的账户的创建和删除，用户的每次登入都会通知其它的在线联系人，如果他的大部分好友与他不在同一个服务器上，它们之间将通过不同网络基础设施之间进行通信，通信量明显增多，平衡流算法很好的解决了这一问题，每次只要计算代价函数将新节点放入相应最优值的服务器上，这样通信量就会明显的降低.

设图数据 $G=(V,E)$. V 为图 G 的点集. E 为图 G 的边集. $|V|=n$，$|E|=m$. k 为子区的数量. 图划分就是将图 G 按照某种规则 π 划分为 k 个子区 $(\pi\rightarrow P=(S_1,S_2,\cdots,S_k))$，子集 $S_i\subseteq V$，$1\leq i\leq k$，且 $S_i\cap S_j=\varnothing$. 而平衡图划分就是使划分之后子区之间的割边数(边的两顶点不在同一个子区)尽量的少，且每个子区负载均衡(负载均衡包括点平衡与边平衡，本文主要指点平衡，即每个子区的点数量几乎相等，负载系数 $\rho\approx1.0$). 割边率用 λ 表示.

$$\lambda=\frac{\#\text{割边数}}{\#\text{总边数}},\ \rho=\frac{\#\text{负载最大的子区}}{\#\text{平均负载}} \tag{1}$$

$S^t=\{S_1^t,\cdots,S_k^t\}$ 表示在 t 时刻 k 路划分的状态，其中 S_i^t 表示在 t 时刻子区 S_i 的点集合. 流式图划分就是将图中顶点按照某种规则排序(如广度优先拓朴排序，深度优先拓朴排序等)，根据此时的划分状态 S^t 及当前点的邻居点 $N(v)$，依次处理点队列中的顶点. 不同流算法的区别是由邻点计算当前点归属子区的启发式方法不同，例如最小非邻居节点流划分算法(Non-Neighbors，NN)，将点 v 分配到子区 S_i 的规则是最小化 $|S_i\backslash N(v)|$. 确定性贪心流划分算法(Deterministic Greedy，DG)，规则是最大化 $|N(v)\cap S_i|$. 指数权重确定性贪婪流划分算法(Exponentially Weighted Deterministic Greedy，EDG)，规则是最大化 $|N(v)\cap S_i|(1-\exp(|S_i|-n/k))$. 流式划分依据为不完整的局部信息，但是随着已分配完成的顶点数量增多，计算当前点可利用的信息量也在不断的增加.

在文献[38]中作者已经证明了深度优先遍历，广度优先遍历和随机遍历对最后划分性能的影响是很小的. 在三种遍历方式中，随机搜索遍历在大规模图顶点排序中是最简单的方式，也符合动态图数据更新的真实情况. 因此，本文图中点队列采用的是随机排序.

Stanton 和 Kliot[29]分析了一系列的启发式流算法的性能，在这些流方法中，性能最好的是线性权重贪婪流算法(Linear Weighted Deterministic Greedy，简称 LDG)，式(2)介绍了将点 v 分配到子区 S_{ind} 的规则.

$$S_{\text{ind}}=\underset{i\in\{1,\cdots,k\}}{\arg\max}\{|N(v)\cap S_i^t|w(t,i)\},$$
$$w(t,i)=1-|S_i^t|/(n/k) \tag{2}$$

从式(2)可以看出，每个顶点只计算一次. 方程前半部分函数 $|N(v)\cap S_i^t|$ 表示在 t 时刻，子区 S_i 中含有点 v 的邻居点数量. 为了使子区负载均衡，在方程后面乘以惩罚函数 $w(t,i)$，惩罚拥有点过多的分区. 选择函数值最大的子区，将点 v 分配到此子区中. 流式划分由于高效的划分管理，近年来得到了不断的发展[40-41]. 但是流方法存在两个问题：(1)划分过程中的邻点缓存数据结构不利于查找，对于查找操作效率较低；(2)由于单计算节点内存容量的限制，划分规模巨大的图数据只能在内存与外存之间来回迁移数据，划分效率明显降低.

针对以上两个问题，我们分别提出了邻边结构及混合内存数据页迁移策略，具体过程在以下章节中详细介绍.

3 动态缓存管理

为了提高流式划分的效率，文献[42-43]采用分布式流算法对图进行划分，划分的效率明显提升，但是增加了硬件与程序的复杂性. 文献[44]提出了重复流概念，在相同的算法框架下，对图进行多轮流处理，划分质量比单次流划分提高很多，但是增加了时间复杂度. 本节对动态缓存中数据结构进行了改进(动态缓存数据是指计算过程中的缓存数据，不包括原图数据，具体指表 2 和表 3 缓存数据内容). 已有流划分中的动态缓存区中的数据是以邻点结构保存，我们采用邻边结构保存，并且对输入邻接图进行了预处理，利用邻接图的对称性，只需输入上三角矩阵，输入规模降低为原图的一半，提高了内存空间的利用率，且不影响划分质量. 并相应地提出邻边结构

的缓存数据管理策略，相对于基于邻点结构的流算法，在单计算节点环境中算法的效率可得到明显提升.

3.1　邻边结构

图 2 用示例介绍了邻点与邻边的定义. 首先介绍基于邻点结构的线性权重贪婪流算法划分过程. 图 3 为示例图 $G(V,E)$ 划分为 3 个子区(S_1, S_2, S_3). 算法首先将图中的顶点随机排成队列，根据顶点的先后依次进行分配. 分配每一顶点，将此点及对应的分区信息($v_i \rightarrow S_i$)保存在内存中. 计算过程如表 2 所示，在 T 时刻分配完 ID 为 1 的顶点，将顶点 v_1 分配到子区 S_1，随后在缓存中保存点 v_1 及对应的分区信息 S_1. 在 $T+1$ 时刻计算 ID 为 3 的顶点(v_3)的所属子区，首先要在缓存中查找已经分配完成的此点邻居点信息(v_3 的邻居点只有 v_1 完成分配，所以只能依据点 v_1 的所属子区信息分配 v_3). 再根据邻点分区信息($v_1 \rightarrow S_1$)计算当前的顶点归属子区，按照同样的规则分配后续的顶点，直到图中所有的点都分配完成，算法结束.

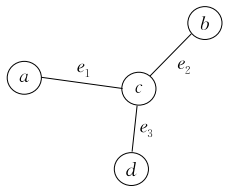

图 2　边 e_1, e_2, e_3 为点 c 的邻边，点 a, b, d 为点 c 的邻点

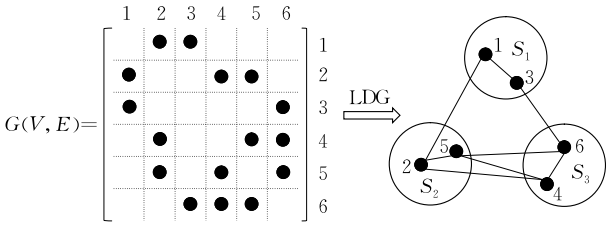

图 3　示例图 G 划分为 3 个子区

表 2　从时刻 T 到时刻 $T+5$ 对图 G 进行流划分过程中动态缓存区中的数据变化(邻点结构)

时刻	对应时刻处理的顶点 ID	处理之后缓存数据内容
T	1	$v_1 \rightarrow S_1$
$T+1$	3	$v_1 \rightarrow S_1$, $v_3 \rightarrow S_1$
$T+2$	2	$v_1 \rightarrow S_1$, $v_3 \rightarrow S_1$, $v_2 \rightarrow S_2$
$T+3$	6	$v_1 \rightarrow S_1$, $v_3 \rightarrow S_1$, $v_2 \rightarrow S_2$ $v_6 \rightarrow S_3$
$T+4$	5	$v_1 \rightarrow S_1$, $v_3 \rightarrow S_1$, $v_2 \rightarrow S_2$ $v_6 \rightarrow S_3$, $v_5 \rightarrow S_2$
$T+5$	4	$v_1 \rightarrow S_1$, $v_3 \rightarrow S_1$, $v_2 \rightarrow S_2$ $v_6 \rightarrow S_3$, $v_5 \rightarrow S_2$, $v_4 \rightarrow S_3$

通过对已有流算法过程的分析，可以发现，输入图是以邻接矩阵的形式载入内存，如图 4 左图，每行内容包括源点及邻居点. 原始流算法分配当前的顶点，由于从邻接矩阵中只能得出此点的邻居点，无法判断出哪些邻居点已经分配完成，所以要在动态缓存中寻找，由邻接矩阵与动态缓存内容共同确认已经分配完成的邻居点.

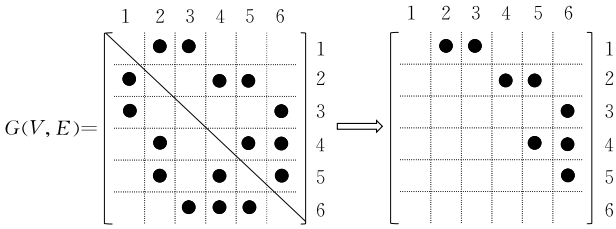

图 4　输入上半角邻接矩阵示例图

据此，我们将缓存数据的结构转换为邻边形式. 分配当前的顶点，只需查找此点为键的字典条目，通过键查找到值，值对应着此点已经分配完成的所有邻点分区信息，通过值可以直接计算出此点所属的子区. 计算过程如表 3 所示，在 T 时刻处理 ID 为 1 的顶点(v_1)，分配完成之后，将此点的分区信息作为值，邻点做为键($v_2:S_1$, $v_3:S_1$)保存入动态缓存中. 在 $T+1$ 时刻，计算顶点 v_3，由于在动态缓存中已经保存了点 v_3 的邻点分区情况($v_3:S_1$)，所以根据此信息直接计算出此点所属的分区，按照同样的规则分配后续的顶点，直到所有的顶点分配完成，算法结束.

表 3　从时刻 T 到时刻 $T+5$ 对图 G 进行流划分过程中动态缓存区中的数据变化(邻边结构)

时刻	对应时刻处理的顶点 ID	处理之后缓存数据内容
T	1	$\{v_2:S_1; v_3:S_1\}$
$T+1$	3	$\{v_6:S_1; v_2:S_1\}$
$T+2$	2	$\{v_6:S_1; v_5:S_2; v_4:S_2\}$
$T+3$	6	$\{v_5:S_2, S_3; v_4:S_2, S_3\}$
$T+4$	5	$\{v_4:S_2, S_3, S_2\}$
$T+5$	4	NULL

每次分配完当前顶点，其所在上半角矩阵中的邻点都是未分配的点. 例如，图 4 左图上半角矩阵中，当算法分配点 v_3 时，可以直接依据动态缓存区其已经分配完成的邻居点的所属子区信息($v_3:S_1$)对其计算. 分配完点 v_3，上半角矩阵中其邻点 v_6 是未分配的顶点. 所以将点 v_6 作为键，v_3 所属的子区信息为值($v_6:S_1$)保存入动态缓存中，供后续的点提取使用. 所以只需输入邻接矩阵的上半角，如图 4 右图所示. 该方法使输入图规模减少一半，提高了内存空间利用率.

3.2 更新过程

算法执行过程中，对于动态缓存中的数据，AeFdy 策略提供了两种操作，分别是添加操作及删除操作.

添加操作 $appenddate(N(v),P(v))$：每次点 v 分配完毕，要将此点的邻居点 $N(v)$ 作为键，点 v 的分区信息 $P(v)$ 作为值以字典条目的形式保存在缓存中. 如果动态缓存中已经存在点 v 的某一邻居点 v_1 为键的条目，则直接在其值域追加点 v 的分区信息 $P(v)$，如果不存在，则新建条目 $item(v_1)$.

删除操作 $deletedate(item(v))$：每次点 v 分配完毕，动态缓存中以此点为键的条目已变成冗余数据则直接将其删除. 例如在表 3 中 $T+1$ 时刻，处理完 ID 为 3 的顶点之后，动态缓存中以此点为键的条目 $(v_3:S_1)$ 对于后续顶点的计算已无价值，所以将其删除，主要是为了提升内存空间的利用率.

相应的伪代码如算法1所示，其中 $ldgalgorithm(v,N(v))$ 为线性权重贪婪流划分算法函数名，输入的参数为点及其邻居点，具体参考图 4 中的输入示例图. 函数 $rs(G(V,E))$ 作用是将图 G 中的顶点按照随机序列排序，函数 $f(S^t)$ 为流模型，输入参数 S^t 为 t 时刻划分的状态. 函数 $dycachepool()$ 表示动态缓存中的数据集合，随着不断执行更新与删除操作，数据也是动态变化的.

算法 1. 线性权重贪婪流划分 $ldgalgorithm(v,N(v))$.

输入：图 $G(V,E)$，子区数 k

输出：划分结果 $P(V)$

```
1. FOR v IN rs(G(V,E))
2.     S_ind → f(S^t)
3.     appenddate(N(v),P(v))
4.     IF item(v) in dycachepool()
5.         deletedate(item(v))
6.     ELSE
7.         CONTINUE
8.     END IF
9. END FOR
```

3.3 复杂度分析

3.3.1 空间复杂度分析

图是以邻接矩阵的形式表示，但是保存方式有很多种，如邻接表方式（图 5 左图），边列表方式（图 5 右图）等，不同方式保存图的规模大小不同，本文输入图规模是按照邻接表方式计算.

设图规模为 $Size_G$，已有流算法首先将图载入内存，所需空间为 $Size_G$，动态缓存区中存放着点及对应的分区信息，最大缓存空间为 $2\times n$，总的消耗内存最大空间大约为 $Size_G+2\times n$.

源点ID	邻点$N(v_{ID})$
1	2 3
2	1 4 5
3	1 6
4	2 5 6
5	2 4 6
	3 4 5

源点ID	目标点ID
1	2
1	3
2	4
2	5
3	6
4	5
4	6
5	6

图 5 示例图 G 的邻接表（左图）及边列表方式（右图）

本文方法输入图规模为原图的一半，即 $1/2\times Size_G$. 在没有删除操作的情况下，动态缓存区中保存所有的邻接边信息，缓存最大规模达到 $Size_G$，但是在有删除操作的情况下，动态缓存区最大规模达不到 $Size_G$，实验结果（图 6）显示低于 $1/2\times Size_G$. 综上所述，在没有删除操作的情况下，基于邻边结构的流算法最大消耗内存空间为 $3/2\times Size_G$. 在有删除操作的情况要小于 $Size_G$. 本文提出的邻边结构方法在空间使用率上要高于邻点结构.

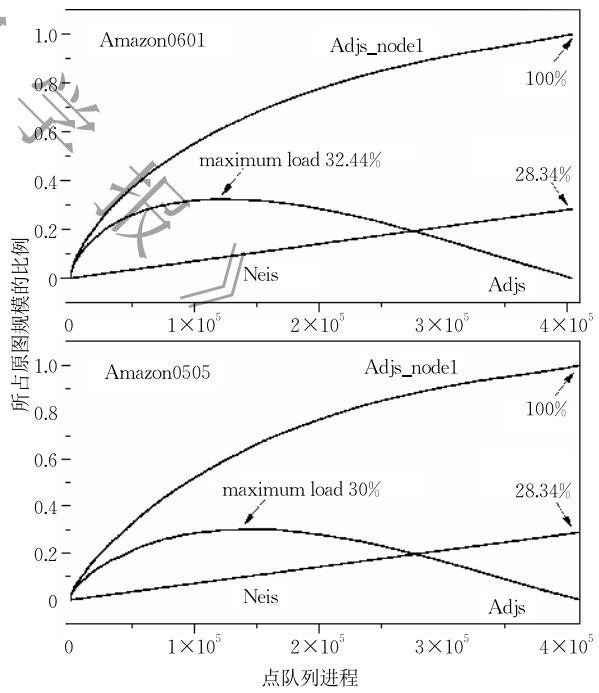

图 6 缓存规模随着点队列进程的变化情况

图 6 为邻边结构（Adjacent Edge Structure，Adjs），邻点结构（Neighbor Structure，Neis），及没有引入删除操作的邻边结构（Adjacent Edge structure without deleteoperation，Adjs_nodel）分别在图 Amazon0601，Amazon0505 测试的结果. 为了直观

地查看结果，我们将动态缓存区中数据规模转换为原图规模的百分比(例如原图规模为 500 M，动态缓存区规模为 100 M，那么动态缓存区规模占原图规模的 20%)．可以看出，在使用邻点结构及没有引入删除操作的邻边结构时，动态缓存数据规模随着处理顶点数的增多而不断的增加．使用邻边结构时，其缓存数据规模到达一定的峰值之后反而不断的减小，最大峰值比使用邻点结构的最大规模要高几个百分点，但是总的消耗内存要比邻点结构小．

3.3.2　时间复杂度分析

假设某顶点 v_i 的度为 $Deg(v_i)$，算法在某时刻分配点 v_i 时，其已经分配完成的此点邻居点数量用符号 $Deg'(v_i)$ 表示(例如表 3 中 $T+1$ 时刻，在处理 ID 为 3 的顶点时，此点的邻居点 v_1 已经处理完毕，所以已经分配完成的 v_3 邻居点数量为 1，即 $Deg'(v_1)=1$)，分区数为 k．采用基于邻点结构的流算法分配当前顶点，首先查找已分配完成的此点邻居点，此步骤时间复杂度为 $Deg(v_i)$，再根据这些邻居点所属的分区信息进行划分，此步骤复杂度为 $k\times Deg(v_i)$，分配完图中所有顶点所需要的时间复杂度为 $\sum_{i=1}^{n}Deg(v_i)+k\sum_{i=1}^{n}Deg'(v_i)$，即 $2m+k\sum_{i=1}^{n}Deg'(v_i)$．引入邻边结构的流算法分配当前的顶点，只需要查找此点为键的条目，每次查找的时间复杂度为 1，再依据此条目中的值直接进行划分，分配完图中所有点所需要的时间复杂度为 $n+k\sum_{i=1}^{n}Deg'(v_i)$，其中 $(2m\gg n)$，时间复杂度明显降低．

本节通过对采用邻边结构的流算法时间复杂度进行了理论分析，并与基于邻点结构的已有流算法进行了对比，在第 5.4 节会通过具体实验衡量划分时间．

4　动态缓存迁移优化

混合内存管理机制的核心内容是数据页的迁移策略．数据页的迁移包括 DRAM 中的数据页迁移到 NVM 中(DN 迁移)，NVM 中的数据迁页移到 DRAM 中(ND 迁移)．图 7 左图展示了 NVM 在混合内存中所处的位置，它和 DRAM 挂载在同一内存总线上并共享内存地址空间．

图 7 右图展示了用户进程虚拟地址空间到混合内存页框的映射关系，以及 DN 迁移过程，具体 DN 迁移过程如下所示：

(1) 在 NVM 空间中分配一个空闲的页框；

(2) 把 DRAM 中的数据拷贝到新分配的 NVM 页框中；

(3) 重新建立该用户进程虚拟地址到物理地址的映射关系，即更新用户进程页表中该数据页对应的页表项内容，将其指向 NVM 页框；释放原 DRAM 页框．

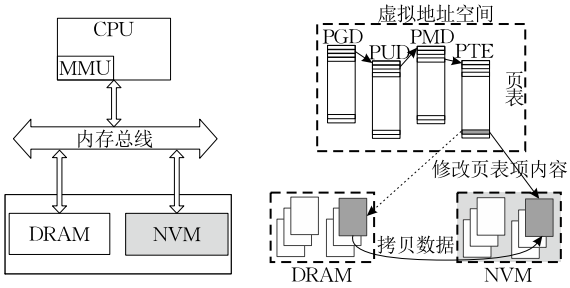

图 7　混合内存系统架构(左)与数据迁移过程(右)

ND 迁移过程与 DN 迁移操作相反．但是 NVM 介质与 DRAM 介质特性存在差异，所以 NVM 空间数据页的分配也与 DRAM 不同，我们将在 4.3 节具体介绍 ND 迁移过程．本章节首先分析应用程序访问内存数据页的特点，得出写不均衡的现象，这是迁移策略设计的基础，然后具体描述两种迁移策略的设计细节．

4.1　内存数据的访问特征

我们对 MiBench[9] 中的一组测试程序访问内存数据时的特征进行分析，这组测试程序包括 SHA(安全散列算法)、CRC32(循环冗余校验算法)、dijkstra(迪杰斯特拉算法)、stringsearch(字符串查找工具)、jpeg(JPEG 编解码程序)、ghostscript(文本编辑处理工具)．这些应用基本覆盖了嵌入式应用和桌面应用的大部分领域，因此特征分析的结果具有普遍性．另外，我们使用 cachegrind[10] 工具记录上述应用程序运行时访问过的所有内存数据页的访问方式，通常是读操作或写操作．

将 cachegrind 统计得到的结果进行处理，按页框被写的次数进行归类，分别统计被写过 1～10、11～100、101～1000 和大于 1000 次的页框个数．统计结果如图 8 所示．横轴表示不同的测试程序，纵轴代表这些不同写次数的页框分别占对应程序总写页框个数的比例．只有 stringsearch 中被写次数小于 10 次的页框个数所占比例为 73%，其它应用程序中所占的比例都在 80%以上．

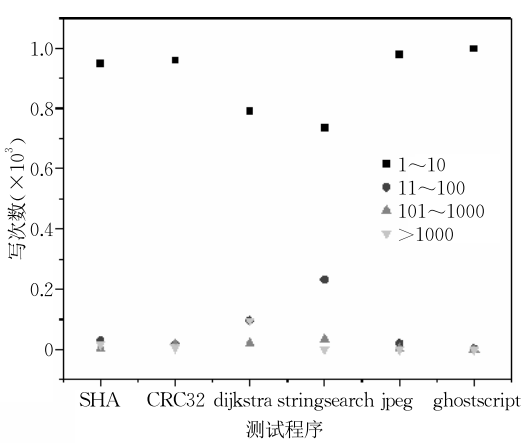

图 8 测试程序写操作比例

基于以上实验结果，可以发现应用程序在内存数据页上的写操作分布是不均匀的，尤其大部分页框只被写过 1 次，以 CRC32 这个应用程序为例，只被写过 1 次的页框数量几乎占到了总写页框个数的 91%. 如果考虑还有部分数据页仅进行读操作，这种不均匀的写操作更加明显. 针对这个写操作不均衡现象，如果允许 NVM 介质发生一定数量的写操作，尤其是那些只被写 1 次的数据页，可以在很大程度上减少数据页迁移操作的次数. 程序运行时间是性能的一个重要指标，如果直接写 1 次 NVM 数据页的开销大于将其迁移到 DRAM 再进行写操作的开销，那么系统的性能将会下降，导致对 NVM 直接写策略没有意义，所以必须验证一点，写 1 次 NVM 数据页的开销是否小于将其迁移到 DRAM 然后再进行写操作的开销.

我们通过具体实验来说明相应的时间开销，在 Linux 内核代码中设置时间戳以计算 ND 迁移操作的开销，通过实验我们计算出 ND 迁移操作的平均时间开销为 29.9 μs. 我们根据处理器访问 DRAM 的时延来计算系统访问 NVM 数据页的开销. 访问 DRAM 时延是指处理器将其缓存行中的数据同步回 DRAM 所用的时间，处理器每次写一个 4 KB 数据页需要进行 64 次(4 KB/64 B)缓存行同步操作. 本文实验平台的处理器写一个数据页到 DRAM 的开销为 529.6 ns. 由此可以计算出数据页 ND 迁移操作的时间开销为 30.4296 μs(29.9 μs+0.5296 μs)，根据文献[6-7]，访问 NVM 介质的时延是 DRAM 的 7 倍，可得出处理器直接写 1 次 NVM 数据页的开销为 3.7 μs. 综上所述，数据页 ND 迁移操作及在 DRAM 对数据页进行写操作的总时间开销是直接写 1 次 NVM 数据页开销的 8.22 倍. 证明了写 1 次 NVM 数据页的时间开销小于将其迁移到 DRAM 然后再进行写操作的开销，也说明了对 NVM 数据页可以进行一些直接写操作的优势，能够提升系统性能.

由此我们可以认为当系统访问 NVM 中的数据页时，如果访问方式为读操作，就直接对该数据页进行访问；如果访问方式为写操作，则允许在 NVM 中进行部分写操作，以减少大量的迁移操作，进而提升系统的性能. 如果系统初始化在 NVM 中为应用程序分配页面，这种策略不具有通用性，它只有在事先知道某个应用程序的访问特征时才有较好的效果. 如果初始化失败，会带来大量的迁移操作. 因此，为了考虑通用性，在我们现有的混合内存管理策略中，总是先从 DRAM 空间为应用程序分配页面，当 DRAM 空间不足时才会从 NVM 中分配空间.

4.2 DN 迁移策略

以 DRAM 为主存的存储系统中，当 DRAM 空间不足时，操作系统会根据数据的特点将部分数据迁移出内存来加载其它的数据. Linux 操作系统使用最近最少使用(Least Recently Used，LRU)算法来完成数据页的替换工作. LRU 算法依据数据的历史访问记录来进行淘汰数据，其核心思想是“如果数据最近被访问过(热数据)，那么在未来的时间里被访问的概率也更高”. 当需要置换一页时，选择在之前一段时间里最久没有使用过的页面(冷数据)予以置换. LRU 算法在 DRAM 中尽量保留更多的热数据，有利于减少 I/O 操作. 但是，LRU 算法只考虑了数据被使用的冷热程度而没有考虑被访问数据的读写特性，因此它在混合内存架构中不再适用.

通常情况下，系统是无法感知应用程序的读/写访问特征. 由于动态缓存数据是实时更新的，根据 3.2 节动态缓存更新过程，对于 DRAM 中的只读条目，这些条目被读一次就会转为冗余数据，为了提高内存利用率，由删除操作将其删除，所以无法从读特征来评价数据页的冷热程度. 基于邻边结构流式划分的特点，我们根据数据被访问的写特征，重新对 DRAM 中的数据页进行了分类，将其定义为以下 3 种.

(1) 热写数据页(HWDP). 数据页近期发生较多次数的写操作. 写次数越多，说明其中以点为键的条目更新越频繁，写的概率较大.

(2) 冷写数据页(CWDP). 数据页在近期只发生少量的写操作，被写的几率在热写数据页与读数

据页之间.

(3)读数据页(RDP).该数据页近期没有发生写操作,我们认为被读的概率较大.

系统将读数据页迁移到 NVM 中,对 NVM 介质影响最小(这些数据页读的概率比较大).LRU 算法会把读数据页保留在 DRAM 介质中或者把热写数据页迁移到 NVM 中,严重影响了 NVM 寿命.我们提出相对最近最少写算法(Relative Least Recently Written,RLRW),与现有的最近最少写算(Least Recently Written,LRW)[45-47]不同,LRW 会记录每个页面的直接写次数,根据写的次数对页面进行分类.用图 3 示例说明,顶点 v_2 计算完毕,创建顶点 v_5 为键的条目,系统会对此条目所在的页面直接写操作.计算完点 v_6 对 v_5 为键的条目进行更新,对其所在的页面再次进行直接写操作,按照 LRW 算法,此页面连续几次发生写操作会将其转为热数据,此页面会保留在 DRAM 中,但是此条目在后续计算中只会进行读操作不会发生写操作,与实际情况相反.

RLRW 会根据邻边结构的特点,将 DRAM 中每个数据页被写的频繁程度与点的度数结合起来,共同决定此页面的冷热程度.设单个条目在某个页面的直接写次数为 W_{item_i},此条目中键的度为 $Degree_{v_i}$.那么此条目在此页面相对写次数 W_{R_i} 应该为

$$W_{R_i}=W_{\text{item}_i}+Degree_{v_i}/X \tag{3}$$

如果在某段时间内对此页面中的 n 个条目进行连续写操作,那么此数据页的相对写次数 W_{total} 为

$$W_{\text{total}}=\sum_{j=1}^{n}W_{R_j} \tag{4}$$

根据数据页最近被写的次数,系统在 DRAM 空间不足时选择合适的候选页迁移到 NVM 中.RLRW 算法将 DRAM 中的数据页保存在 3 个不同的链表中:

(1)活动链表,该链表包含热写数据页.

(2)非活动链表,该链表包含冷写数据页.

(3)读链表,该链表包含读数据页.

图 9 展示了数据页在 RLRW 链表中的迁移情况及迁移条件.RLRW 链表通过 *PG_write* 和 *PG_active* 来标记每个数据页的状态.如果数据页没有发生写操作,则它一直是读数据页,将被保存在读链表中(*PG_write*=0,*PG_active*=0).当读链表中的数据页被写一次,该数据页就会被迁移至非活动写链表(*PG_write*=1,*PG_active*=0).当该数据页连续被写多于 1 次时,该数据页将从非活动写链表被迁移至活动写链表成为热写数据页(*PG_write*=1,*PG_active*=1).当活动写链表中的热写数据最近发生一次读操作,该数据页就会被迁移至非活动写链表且 *PG_active* 重置为零,当该数据页最近发生多余一次的读操作时,该数据页就会被迁移至读链表中,同时 *PG_write* 重置为零.

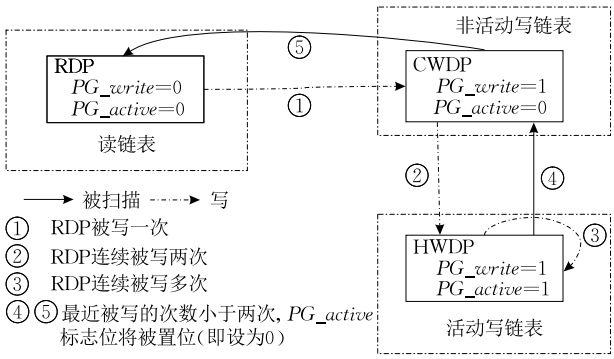

图 9　数据页在 RLRW 链表中的迁移

当 DRAM 空间不足时会引发 DN 迁移,AeFdy 机制中的 RLRW 算法对 DRAM 介质中的数据页迁移规则如下:首先会选择读链表中的读数据页(RDP)进行迁移,因为将这部分页迁移到 NVM 上对系统的性能所带来的负面影响最小.如果回收 DRAM 这部分数据页所释放的空间无法满足系统的分配需求,RLRW 算法会对非活动链表中的冷写数据页(CWDP)进行回收.如果回收这部分数据页所释放的空间仍然不足,RLRW 会释放活动链表中的热写数据页(HWDP)进行回收.

4.3　ND 迁移策略

现有关于 NVM 管理机制的研究大多数都是以 NVM写次数最小化为研究目标[48-50],如Dr. Swap[17]、M-CLOCK[14]机制.本文设计的 AeFdy 机制中的 ND 迁移策略允许对 NVM 进行少量的写操作.在 4.1 节中已经分析过应用程序访问内存数据页表现出写操作不均衡的现象.ND 策略具体描述为:当对 NVM 中的数据页进行读操作时,直接对数据页进行读取,当对数据页进行写操作时,允许 NVM 页框发生少量的写操作,来保证大部分数据页不发生迁移操作.我们设置一个写操作次数的上限 W 来进行判断,当 NVM 空间中某个数据页达到这个写次数上限时,我们就认定这个数据页属于写操作次数很多的页,将其迁移到 DRAM 再进行写操作.算法描述如算法 2 所示.

算法 2. ND 迁移策略.

输入:NVM 中的数据页 p,数据页 p 的写次数 N_p,访问方式 AM,阈值 W

输出:数据页 p 的访问过程

```
IF AM=read
    在 NVM 中读取数据页 p
ELSE
    IF N_p<W
        在 NVM 中写数据页 p
        N_p=N_p+1
    ELSE
        将数据页 p 从 NVM 迁移到 DRAM
        N_p=0
        在 DRAM 中写数据页 p
    END IF
END IF
```

接下来我们通过理论分析阈值 W. 数据页从 NVM 页框迁移到 DRAM 页框的时间开销假设为 E_{ND}. 直接写一次 NVM 数据页的开销为 E_{WN}. 直接写一次 DRAM 数据页的开销为 E_{WD}. 为了使系统性能达到最优,对于 NVM 中的某一数据页,对其写 T 次的开销应该小于此数据页从 NVM 页框迁移到 DRAM 中所消耗的时间(即 $E_{WN}\cdot T\leq E_{ND}$),且在此数据页中写 $T+1$ 次的时间开销大于此数据页从 NVM 页框迁移到 DRAM 中的开销(即 $(T+1)\cdot E_{WN}\geq E_{ND}$). 假设该数据页被写过的次数为 i,那么写这个 NVM 数据页的总开销 E_{total} 计算如下:

$$E_{total}=\begin{cases} i\cdot E_{WN}, & i\leq W \\ E_{ND}+E_{WD}\cdot(i-W)+E_{WN}\cdot W, & i>W \end{cases} \tag{5}$$

我们分两种情况来讨论最优解. (1) 如果 $i\leq W$,即对该数据页的写次数没有超过阈值,那么最优解应该是直接在 NVM 上写这一数据页 i 次;(2) 如果 $i>W$,即对该数据页写次数超过阈值,最优解应该是在第一次写该数据页时就把它从 NVM 页框迁移到 DRAM 中,以后的写操作都在 DRAM 中进行. 因此,最优解 $optimal_{total}$ 可以表示为

$$optimal_{total}=\begin{cases} E_{WN}\cdot i, & i\leq W \\ E_{ND}+E_{WD}\cdot i, & i>W \end{cases} \tag{6}$$

由式(6)可知,当 $i\leq W$ 时,此策略能够满足最优解. 对于 $i>W$ 的情况,为了衡量最优解与实际得到的解之间的差异,我们将所得到的解除以最优解,用 $Rate$ 表示为

$$\begin{aligned} Rate&=\frac{E_{ND}+E_{WD}\cdot(i-W)+E_{WN}\cdot W}{E_{ND}+E_{WD}\cdot i} \\ &=1+\frac{E_{WN}-E_{WD}}{E_{ND}+E_{WD}\cdot i}\cdot W \end{aligned} \tag{7}$$

$Rate$ 是一个关于 i 的单调递减函数. 在式(6)中 i 的取值为 $i>W$,所以所得值与最优值相差最大时,$i=W+1$. 上一节我们已经测得 $E_{ND}=29.9\ \mu s$、$E_{WD}=529.6\ ns$、$E_{WN}=3.7\ \mu s$,将这些数值代入式(5)、(6),得出 W 的值为 8. 具体含义为:当对 NVM 中某一数据页写操作多于 8 次时,先对此数据页直接写 8 次,再将其迁移到 DRAM 介质中写剩余的次数,显然这种情况是系统开销最多的. 这里得出的阈值 W 是通过理论的方法计算得出,本文将在 5.3 节通过实验方式来确定 ND 迁移策略中的阈值 W.

5 实验与结果分析

我们首先进行模拟环境的搭建,用于实现管理混合内存页面的内存管理机制. 接着用具体实验来分析讨论 ND 迁移中阈值 W,最后从系统性能、数据页迁移操作次数,NVM 写操作次数等方面来衡量 AeFdy 策略. 同时我们也列出了其它混合内存管理机制(M-CLOCK[14]、Dr.Swap[17]、Linux Swap[51]),并与本文所提的 AeFdy 机制进行对比.

M-CLOCK 算法是对经典时钟算法[52-53]的一种改进,它通过脏标记(D 指针)和访问标记(C 指针)来记录每个 DRAM 数据页被访问的情况. 当一个候选数据页被重新访问时,算法就会通过两个指针来确定此页面是否为热脏页. 当 DRAM 空间消耗已满时,D 指针就会在标记的热脏页面里选择最低写频繁页迁移到 NVM 中,如果无法找到,那么就会由 C 指针在所标记的干净页面里选择所需页面进行迁移. 通过此方法,M-CLOCK 将 DRAM 中的读冷数据页迁移到 NVM 介质中,使对 NVM 的写操作次数保持最小化.

Dr.Swap 算法是将 NVM 存储区作为交换区使用,当 NVM 数据页需要被读时,系统可直接进行读操作,而当 NVM 数据页需要被写时,需先将其迁移到 DRAM 中,然后再进行写操作. 这种方式会导致大量的迁移操作,使性能受到严重影响. 另外,在选择 DRAM 数据页替换时,该研究也未充分考虑内存数据页的访问特征,有一部分写密集型数据页面频繁的在 DRAM 和 NVM 之间进行迁移操作,使得系统发生了过多无用的开销.

Linux 系统使用 Swap 策略来缓解物理内存不足的情况. 当系统的物理内存不足时,系统会将内存中的一部分数据释放出来,以供当前运行的程序使

用.这些被释放的数据被临时保存到 Swap 外存空间中,当需要这些数据时,再从 Swap 空间恢复数据到内存中.这些操作会阻塞系统真正的内存访问,因此开销非常大.

5.1　实验环境

从 Linux4.0 版本开始,内核就加入了对使用 NVM 设备的支持,它可以将一段物理内存空间注册成一个 NVM 设备,并将这段内存的物理地址映射成虚拟地址供开发者使用.我们通过这种方式将系统内存的一部分模拟成 NVM 空间,其结构如图 10 所示.

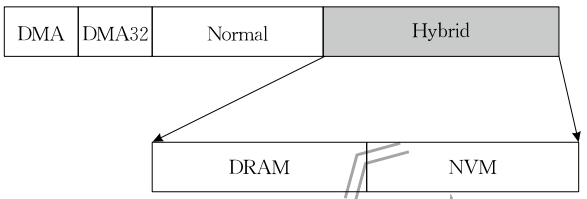

图 10　混合内存的分区

在 64 位 Linux 系统中,内存有三个区域,分别是 ZONE_DMA,ZONE_DMA32,ZONE_NORMAL.我们在操作系统初始化时保留一部分内存空间,并将其注册为 NVM 区域.需要说明的是,操作系统在正常使用内存时,不会使用到 NVM 上的页框,NVM 与其它三个子区是互相独立的,这样有利于我们单独管理模拟空间中的页框,防止操作系统常规的分配、回收和交换等流程使用模拟空间.图划分应用程序在正常使用内存时,一般包括申请内存空间、读写内存中的数据以及释放内存空间等过程.我们在 Linux 内核中主要实现了四个功能调用,分别是申请混合内存空间(*DN_MALLOC*())、读写混合内存中的数据页(*DN_READ*()、*DN_WRITE*())和释放混合内存空间(*DN_FREE*()).

需要说明的是,应用程序申请内存空间后,就可以直接通过虚拟地址进行读写.但是在我们的模拟实验中,需要记录 NVM 每个页框的访问特征,如果直接通过虚拟地址进行读写,不经过内核层,将无法捕捉到 NVM 页框的读写次数,所以我们实现 *DN_READ*()和 *DN_WRITE*()函数接口进行读写操作.我们在 X86_64 架构的 Lenovo 单机下进行实验,使用 Linux4.4 版本的内核,在实验环境中,操作系统的内存大小为 8 GB,混合内存区由容量为 512 MB 的 DRAM 和 512 MB 的 NVM 组成.实验使用 DRAM 模拟 NVM 空间.具体环境配置参数如表 4 所示.

表 4　实验环境配置

项目	配置
处理器	Intel(R) Core(TM) i3-3220 CPU@3.30 Hz
操作系统	Ubuntu15.04
内核版本	Linux4.4
系统内存	4 GB
混合内存区	512 MB DRAM+512 MB NVM(模拟)

不同的 NVM 存储介质对划分时间及 ND 迁移策略中阈值的设定都有影响,但是可以用本文的方法通过具体 NVM 存储介质的参数计算阈值.本文实验采用的是学术界和工业界中比较成熟的相变存储器(Phase Change Memory,PCM),具体参数引用参考文献[6-7],PCM 的访问延迟是 DRAM 的 7 倍,DRAM 的访问延迟是 8.275 ns,NVM 的写延迟为 57.925 ns,NVM 的读延迟与 DRAM 的访问延迟相等,直接写一次 NVM 数据页的开销为 3.707 s.

5.2　实验数据集

实验所选用的是真实世界的图数据集,且综合平衡了图的规模和种类,包括社交网,时序网,引文网络等.表 5 列出了本文所用到的实验图数据,所有的图数据来自斯坦福大学网络分析项目[26].本文主要考虑的是连通图.为了防止零切割边的出现,重边,自环,度为零的点都已经被移去.

表 5　实验数据集①②

图	顶点数	边数	类型
soc-LiveJournal1	4847571	43369619	Social
Soc-Pokec	1632803	22301964	Social
Twitter−2010	41652230	1468365128	Socail
com-LiveJournal1	3997962	34681189	Trust
Friendster	100199	14067887	Socail
com-Orkut	3072441	117185083	Trust
com-Youtube	1134890	2987624	Trust
amazon0601	403394	3387388	Product
amazon0505	410236	3356824	Product
amazon0312	400727	1234877	Product
email-Enron	36692	183831	Communication
wiki-Talk	2394385	4659565	Communication
cit-Patents	3774768	16518948	Citation
sx-stackoverflow	2601977	63497050	Temporal
as-Skitter	1696415	11095298	Autonomous
web-BerkStan	685230	6649470	Web
Web-Stanford	281903	1992636	Web

5.3　ND 迁移中阈值的设定

在 4.3 节中,我们通过理论的方法对 ND 迁移策略中的阈值进行了计算,在本小节中,我们将通过具体实验来讨论阈值的设定.实验对象选用社交网

① http://snap.stanford.edu/data/index.html
② http://twitter.mpi-sws.org/data-icwsm2010.html

络 soc-LiveJournal1. 实验设定不同阈值 W，分别统计在不同阈值条件下，系统发生的 ND 迁移操作次数和 NVM 写操作次数. 其中，NVM 写操作次数包括程序直接对 NVM 写操作的次数以及由 DN 迁移操作引起的对 NVM 写操作的次数.

在图 11 和图 12 中，横轴代表不同的阈值($W=2,4,\cdots,50$)，竖轴分别表示 NVM 写操作次数和 ND 迁移操作次数. 图中还标出不同阈值下具体的操作次数. 当 W 较小时，系统发生的 ND 迁移操作和 NVM 写操作次数都很多. 如阈值为 2 时，迁移操作次数在所有阈值中是最多的. 这主要是因为 NVM 中大量的数据页在写过一次后就会被迁移到 DRAM 介质中，加剧了 DRAM 空间的消耗，系统又会将数据页从 DRAM 中迁移到 NVM 中，这些过程导致大量的迁移操作及对 NVM 的写操作. 当阈值为 8 时，NVM 写操作次数达到了最小值. 随着阈值的继续扩大，ND 迁移次数急剧下降，这是因为阈值的扩大使得对 NVM 数据页的写操作都直接落在 NVM

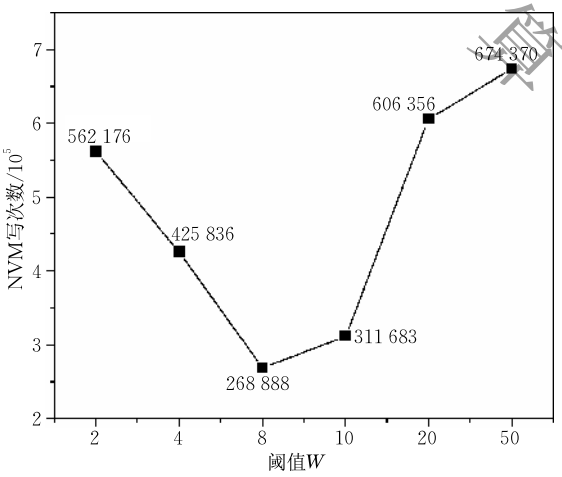

图 11 不同阈值下的 NVM 写操作次数

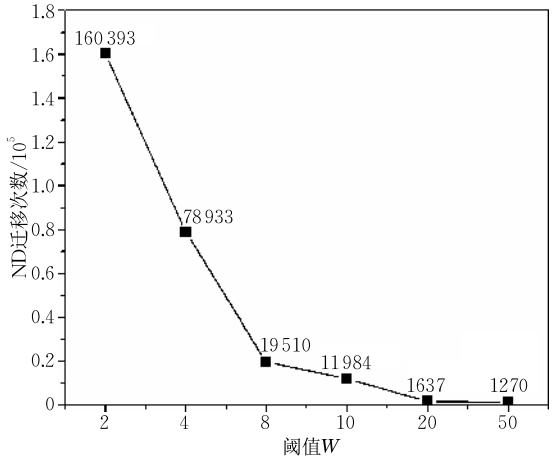

图 12 不同阈值下的 ND 迁移操作次数

页框中，造成了 NVM 写操作次数逐渐增加. 当阈值为 20 时，系统对 NVM 的写操作次数就已经超过了阈值为 2 时的写次数.

通过以上分析，较小的阈值和较大的阈值都不满足实际的需求. 在后续实验中，ND 迁移策略中的阈值 W 都设置为 8，这是因为相对于其它阈值，此阈值使系统发生的 NVM 写操作次数最少，并且此时 ND 迁移操作次数也得到了大幅的降低.

5.4 系统性能

本节主要衡量 AeFdy 策略的性能. 我们选用文件系统性能的自动化测试工具 filebench[19] 中的一组测试程序，这组测试程序包括 multi-stream-write、fileserver、webproxy、randomrw、videoserver，对 AeFdy 中迁移策略的性能与其它混合内存管理策略进行比较分析，然后使用不同网络图在不同存储架构上从划分时间方面衡量 AeFdy 策略性能. 需要说明的是，AeFdy 的 DN 迁移策略中的数据页面的冷热程度是由网络中顶点度决定的，不适合非网络应用，因此在用 filebench 测试工具集时，对于 AeFdy 中的 DN 迁移策略采用类似的 LRW 算法[45-47]. 在 5.7 节也对比了 LRW 与 RLRW 的性能.

测试程序完成所有内存数据访问所消耗的时间包括读、写 DRAM 和 NVM 数据页的时间，以及进行迁移操作的时间. 这一时间用 t_{total} 表示，具体描述如下：

$$t_{\text{total}}=(N_{\text{DN}}+N_{\text{ND}})\times E_{\text{mig}}+N_{\text{RD}}\times E_{\text{RD}}+N_{\text{WD}}\times E_{\text{WD}}+N_{\text{RN}}\times E_{\text{RN}}+N_{\text{WN}}\times E_{\text{WN}} \quad (9)$$

其中 N_{DN}、N_{ND} 分别表示测试程序运行过程中，系统发生的 DN 迁移和 ND 迁移操作次数，E_{mig} 表示迁移操作的时间. N_{RD}、N_{WD} 分别表示测试程序直接读、写 DRAM 内存数据页的次数. N_{RN}、N_{WN} 分别表示测试程序直接读、写 NVM 内存数据页的次数. 读、写 DRAM 内存数据页的时间和读 NVM 内存数据页的时间相等，即 $E_{\text{RD}}=E_{\text{WD}}=E_{\text{RN}}$. 写 NVM 内存数据页的时间为写 DRAM 内存数据页时间的 7 倍，即 $E_{\text{WN}}=7\times E_{\text{WD}}$. DN 迁移操作的时间与 ND 迁移操作的时间相等. 迁移操作的时间为写 NVM 内存数据页时间的 8 倍，即 $E_{\text{mig}}=8\times E_{\text{WN}}$.

我们将 Linux 策略运行每组测试程序的时间作为基准参照，即实验结果中各方案运行测试程序的时间与 Linux 策略运行测试程序时间的比值，比值越小，说明该方案相较于 Linux 策略运行的时间就越短，相应的系统性能就越好. 图 13 为测试程序的运行时间归一化之后的结果. 与使用 Linux、Dr.Swap 和 M-CLOCK 这三种策略的系统相比，使

用本文的策略分别将性能提高了 128.5%、87.4%、50.4%. 表 6 与表 7 分别为系统发生的迁移操作次数及 NVM 写操作次数. 通过对数据分析可知，影响系统性能的主要因素是系统所产生的迁移操作次数，其次是 NVM 的写操作次数. 在 fileserver 这组实验中，M-CLOCK 和本文策略所产生的迁移操作次数及 NVM 写操作次数都比较接近，因此这两种策略在性能方面差异不大. 使用 Linux 策略的系统在性能方面表现最差，因为该系统发生的迁移操作次数最多. 而在 webproxy 这组实验中，本文策略的性能提升最高，这也是因为相对于其它策略，系统减少的迁移操作比例最高.

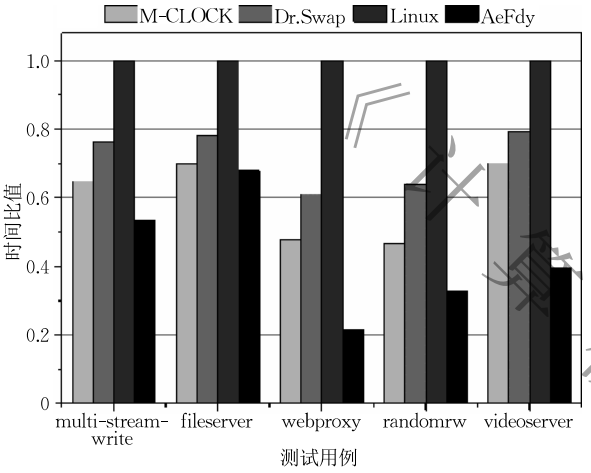

图 13　归一化后的系统运行时间

表 6　迁移操作次数

测试程序	AeFdy	M-CLOCK	Dr.Swap
muti-stream-write	34644	79479	163449
fileserver	34467	40404	91983
webproxy	8789	43369	65670
randomrw	9309	27631	52976
videoserver	8795	46078	66327

表 7　NVM 写操作次数

测试程序	AeFdy	M-CLOCK	Dr.Swap
muti-stream-write	448175	487230	109064
fileserver	345593	357117	62143
webproxy	62280	66104	27528
randomrw	28063	33208	21968
videoserver	81264	105997	32688

接着在不同存储结构上对网络图进行划分，对比划分所消耗的时间. 其它存储结构分别为 DRAM＋磁盘(DISK)、DRAM＋固态硬盘(SSD). 当动态缓存数据规模大于 DRAM 最大容量时，系统会将数据迁移到 DISK 或者 SSD 外存中，需要时再将这些数据从外存重新载入内存. 由于实验条件的限制，其中一部分网络图的规模要小于 DRAM 容量，算法运行过程中缓存数据将无法迁移，为了达到实验目的，我们将 DRAM 中动态缓存数据的容量分别设置为对应图规模的 5%，9%，13%，17%，超过指定的容量，数据将被迁移到其它存储介质中. 系统在三种不同的存储架构中都采用的是邻边结构对实例图进行流划分. 表 8 列出了网络图在不同存储架构中的划分时间，本文混合内存架构平均要比磁盘存储结构减少约 72%，最高减少 85%的划分时间，比固态硬盘结构平均要减少约 54%，最高减少 65%的划分时间. 但是随着 DRAM 容量的增加，三种结构的划分时间都在减少，因为更多的缓存数据保存在 DRAM 中，减少了迁移次数. 由于在混合内存系统中影响系统性能的主要因素是系统所产生的迁移操作次数，其次是 NVM 的写操作次数，所以我们在 5.5 节及 5.6 节单独分析了在混合内存环境中使用 AeFdy 策略对网络图进行划分过程中所产生的迁移操作次数及 NVM 写操作次数.

表 8 中 DRAM-only 表示动态缓存数据都保存

表 8　在不同存储架构中的划分时间

图	DRAM＋NVM				DRAM＋DISK				DRAM＋SSD				DRAM-only	
	5%	9%	13%	17%	5%	9%	13%	17%	5%	9%	13%	17%	100% (Adjs)	100% (Neis)
soc-LiveJournal	**145.20**	**112.31**	**107.13**	**98.89**	962.50	812.00	713.01	628.25	424.80	368.04	312.47	214.90	**82.61**	379
Soc-Pokec	**105.65**	**92.52**	**86.34**	**72.69**	453.00	381.90	314.96	256.63	234.20	196.54	156.90	119.80	**36.10**	132
com-LiveJournal	**251.54**	**210.96**	**184.12**	**160.47**	851.09	782.71	711.31	625.09	518.79	475.15	403.68	347.04	**65.63**	312
as-skitter	**80.07**	**71.26**	**63.90**	**48.75**	289.10	209.80	182.47	132.70	180.80	152.68	128.30	98.10	**21.09**	96
com-Orkut	**464.20**	**428.14**	**355.73**	**253.29**	1369.54	1164.57	1046.96	968.60	1023.52	925.40	861.00	765.80	**139.24**	582
cit-Patents	**183.27**	**147.07**	**127.13**	**109.23**	647.30	599.01	529.58	473.40	420.36	387.59	310.70	259.07	**47.84**	211
sx-stackoverflow	**202.15**	**196.22**	**161.09**	**156.57**	819.78	782.32	737.46	624.80	546.25	487.39	413.40	359.58	**52.71**	298
web-BerkStan	**30.07**	**26.15**	**22.85**	**20.78**	148.27	120.08	102.21	85.00	85.16	76.90	61.97	51.70	**11.03**	42
Twitter	**237.74**	**226.84**	**205.64**	**152.00**	528.64	501.50	452.19	385.10	417.29	329.54	287.30	213.47	**146.43**	236
Friendster	**86.34**	**77.51**	**70.57**	**65.86**	217.90	181.80	164.73	110.30	169.62	125.00	95.08	87.00	**58.51**	95

在 DRAM 中,实验目的是在同等环境下将邻边结构与邻点结构在划分效率上进行对比.引入邻边结构的流算法相对于邻点结构,划分时间效率平均提高了 4.9 倍,最高接近 6 倍.图 14 及图 15 为分别使用邻边结构和邻点结构的流算法划分具有不同数量边的图(表 5 中的图数据集)与划分时间之间的关系.由结果图可以看出,划分具有百万数量级边的图,邻点结构最多需要接近 400 s,最少也需要 100 s,而邻边结构最多只需要接近 100 s,且大部分集中在 40 s 附近.划分本文规模最大的网络图 Twitter-2010,使用邻边结构的流算法需要 20 min 左右,使用邻点结构的流算法需要约 42 min.说明了邻边结构的优越性.

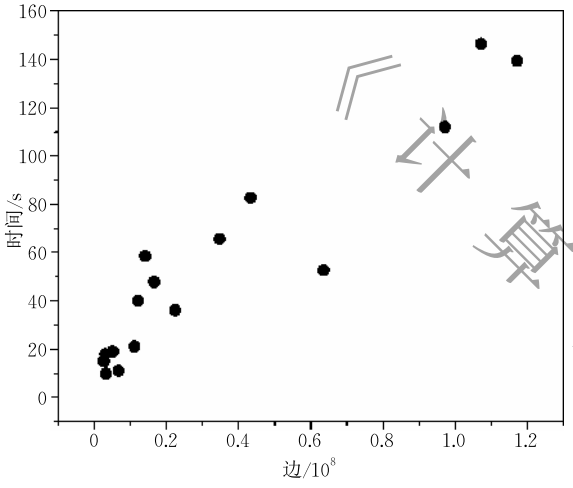

图 14 邻边结构:边数量与时间的关系

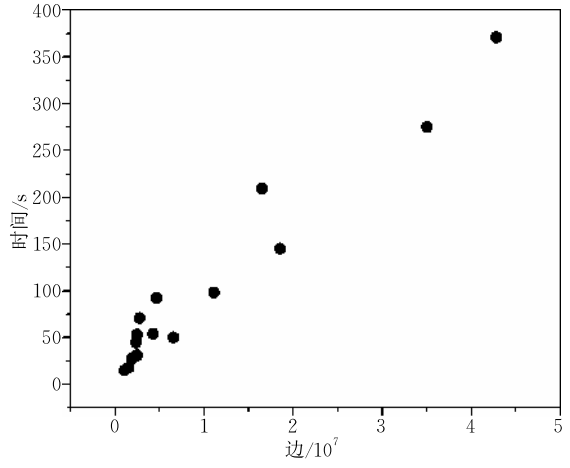

图 15 邻点结构:边数量与时间的关系

5.5 迁移操作次数

迁移操作次数包括数据页从 NVM 页框迁移到 DRAM 页框(ND 迁移)的次数,以及数据页从 DRAM 页框迁移到 NVM 页框(DN 迁移)的次数.

三种图数据在不同策略中产生的迁移操作次数结果如图 16 所示.与 M-CLOCK 和 Dr.Swap 策略相比,本文提出的 AeFdy 策略平均减少了 49.57% 和 74.6% 的迁移操作次数.主要是因为 AeFdy 机制允许直接在 NVM 页框中访问数据页(少量的写操作),与其它策略相比,提高了 NVM 空间的使用率,减少了数据页的 ND 迁移操作,同时也减少了 DRAM 空间不足的情况,从而间接减少了数据页的 DN 迁移操作.与 AeFdy 策略不同的是,在 Dr.Swap 策略中,应用程序只要尝试对 NVM 数据页进行写操作,该数据页就会被系统迁移到 DRAM 介质中,然后再对该数据页进行写操作.M-CLOCK 策略仅在 DRAM 空间不足时,才会允许对 NVM 介质中的数据页进行一次写操作,这两种策略都会发生大量的 ND 迁移操作及 DN 迁移操作.

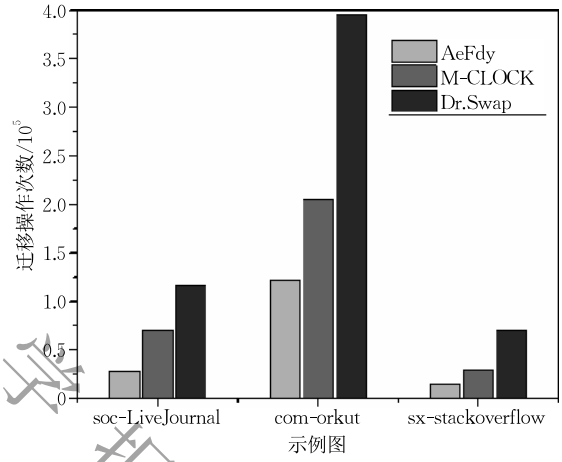

图 16 不同策略下的迁移操作次数

5.6 NVM 写操作次数

NVM 写操作包括系统直接对 NVM 数据页的写操作以及由 DN 迁移引起的对 NVM 的写操作.结果如图 17 所示.3 种策略对 3 个网络图划分的结

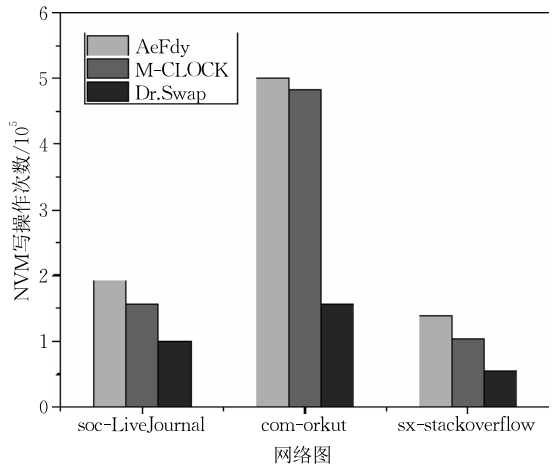

图 17 NVM 写操作次数

果中，Dr.Swap 策略所产生的 NVM 写操作次数都是最少的，主要是因为 Dr.Swap 策略没有对 NVM 数据页的直接写操作，它所产生的写操作是由于当 DRAM 内存消耗殆尽必须将数据页迁移到 NVM 所产生的间接写操作次数.

而对于另外两种策略，理论上 M-CLOCK 允许直接对 NVM 介质的写操作次数更少. 因为 M-CLOCK 允许直接对 NVM 写一次，而 AeFdy 允许直接对 NVM 介质写多次. 但是实验结果表明 AeFdy 机制在所有测试网络图中，对 NVM 写操作次数都少于 M-CLOCK 策略，主要原因有两点：(1) 在运行测试程序过程中，程序所占内存的空间都超过了 DRAM 的容量，混合内存系统的 DRAM 空间在大多数都处于即将耗尽的状态，所以必须将数据页迁移到 NVM 中并允许对这些数据页直接写一次. 在 4.1 小节中我们已经分析过内存数据页被写一次的比例很高，导致更多的 NVM 中直接被写一次的数据页迁移到 DRAM 中，也间接导致了更多的数据页从 DRAM 迁移到 NVM 中；(2) 由于 M-CLOCK 策略产生更多的 DN 迁移及 ND 迁移，所以对 NVM 介质间接产生的写操作次数较多.

5.7　RLRW 性能分析

RLRW、LRU 以及 LRW 是三种不同的页面置换策略. 我们分别使用这三种策略，记录系统所发生的迁移操作次数. 如果系统发生的迁移操作次数越少，说明置换策略所选择的数据页越准确，反之迁移操作次数越多，说明置换策略性能越差.

实验结果如图 18 所示，相对于 LRU 策略，使用 LRW 替换策略平均减少了 9% 的迁移操作次数，使用 RLRW 替换策略平均减少了约 10.1% 的迁移操作次数. 相对于 LRW 策略，使用 RLRW 替换策略平均减少了 0.9% 的操作次数. 主要是因为 RLRW 策略不仅考虑了数据页中直接写的次数同时也考虑了流算法中邻边结构的特点，将写概率最小的数据页迁移到 NVM 空间，保留更多的写数据页在 DRAM 中，减少了系统的迁移操作次数. 结果说明了 RLRW 策略更适合于混合内存的图划分应用.

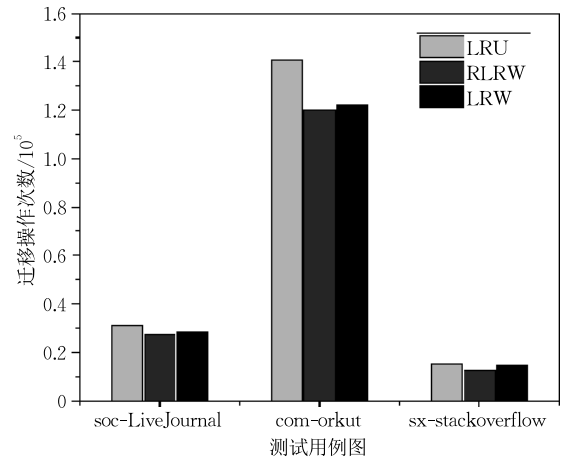

图 18　三种替换策略中的迁移操作次数

6　结　论

图可以用来表达实体之间复杂的关系，有着广泛的实际应用，如生物信息学、社交网络、Web 分析、网络计算等都可以用图的思维对问题进行建模与分析. 这类问题所构造的图规模通常十分庞大. 而大图的计算不仅是计算密集型，同时也是存储密集型，如何在可接受的代价内对大图进行有效计算，是亟待解决的问题. 这就使得复杂网络的研究不得不借助于高效的计算工具来实现. 并行计算技术是现在最成熟、应用最广、最可行的计算加速技术之一. 因此研究复杂网络计算的并行加速技术具有十分重要的意义，而如何在各处理器间分配任务是并行计算性能好坏的关键. 图划分技术就是解决这一问题的有效手段之一.

本文将流式图划分应用在混合内存架构中，并提出了高效的缓存数据管理机制 AeFdy. AeFdy 的核心思想在于：根据不同内存介质的特性以及基于邻边结构流划分的过程，将具有不同访问特征的数据页保存在合适的存储介质中，以减少迁移操作次数及写操作次数，同时将流图划分过程中的缓存数据转换成邻边结构便于查找，提高算法效率. 支撑 AeFdy 机制的一个重要条件是应用程序访问内存数据页存在写操作不均衡这一现象，据此，本文提出保留一部分写操作在 NVM 介质中进行，直到该数据页的写操作次数达到阈值才将其迁移到 DRAM 空间中. 最后实验证明所提方法的有效性. 本文主要探讨图划分在混合内存中的应用，对单机条件下大图计算具有一定的借鉴意义.

7　未来工作

分布式系统的研究与广泛应用，对图划分算法的效果和执行效率提出了严峻挑战，在本文的问题研究中，我们主要关注混合内存系统的读写操作与迁移操作，而没有考虑到其它方面对性能的影响，如静态功耗、NVM 数据一致性以及语义网络的划分

等问题,这都是实际环境中所面对的真实问题,下一步的研究是探索开发适合在混合内存运行中的大图计算框架,提升在单机环境下的大图计算效率.

参 考 文 献

[1] Mao Wei, Liu Jing-Ning, Tong Wei, et al. A review of storage technology research based on phase change memory. Chinese Journal of Computers, 2015, 38(5): 944-958 (in Chinese)
(冒伟, 刘景宁, 童薇等. 基于相变存储器的存储技术研究综述. 计算机学报, 2015, 38(5): 944-958)

[2] Ruocco S, Le D K. Efficient persistence of financial transactions in NVM-based cloud data centers//Proceedings of the International Conference on Cloud Computing Research and Innovation. Singapore, 2016: 25-36

[3] Stitt G, Grattan B, Villarreal J, et al. Using on-chip configurable logic to reduce embedded system software energy//Proceedings of the Field-Programmable Custom Computing Machines. Napa, USA, 2002: 143-151

[4] Wong H S P, Raoux S, Kim S B, et al. Phase change memory. Proceedings of the IEEE, 2010, 98(12): 2201-2227

[5] Jung M, Shalf J, Kandemir M. Design of a large-scale storage-class RRAM system//Proceedings of the International ACM Conference on International Conference on Supercomputing. Eugene, USA, 2013: 103-114

[6] Raoux S, Burr G W, Breitwisch M J, et al. Phase-change random access memory: A scalable technology. IBM Journal of Research & Development, 2008, 52(4.5): 465-480

[7] Zhou P, Zhao B, Yang J, et al. A durable and energy efficient main memory using phase change memory technology//Proceedings of the 36th International Symposium on Computer Architecture. Texas, USA, 2009: 14-23

[8] Sharma A, Tyagi V V, Chen C R, et al. Review on thermal energy storage with phase change materials and applications. Renewable & Sustainable Energy Reviews, 2009, 13(2): 318-345

[9] Zhang W, Li T. Exploring phase change memory and 3D die-stacking for power/thermal friendly, fast and durable memory architectures//Proceedings of the 2009 18th International Conference on Parallel Architectures and Compilation Techniques. Raleigh, USA, 2009: 101-112

[10] Schonhals A, Mohr J, Wouters D J, et al. 3-bit resistive RAM write-read scheme based on complementary switching mechanism. IEEE Electron Device Letters, 2017, 38(4): 449-452

[11] Alaabdlqader H S, Sleiman A, Sayers P, et al. Graphene oxide-based non-volatile organic field effect memory transistors. IET Circuits Devices & Systems, 2018, 9(1): 67-71

[12] Lee M, Kang D H, Kim J, et al. M-clock: Migration-optimized page replacement algorithm for hybrid DRAM and PCM memory architecture//Proceedings of the 30th Annual ACM Symposium on Applied Computing. Marrakech, Morocco, 2015: 2001-2006

[13] Li Q, Zhao Y, Hu J, et al. MGC: Multiple graph-coloring for non-volatile memory based hybrid scratchpad memory//Proceedings of the 16th Interaction Between Compilers and Computer Architectures (INTERACT). New Orleans, Louisiana, 2012: 17-24

[14] Lee B C, Ipek E, Mutlu O, et al. Architecting phase change memory as a scalable DRAM alternative//Proceedings of the 36th Annual International Symposium on Computer Architecture. Austin, USA, 2009: 2-13

[15] Lee H G, Baek S, Nicopoulos C, et al. An energy- and performance-aware DRAM cache architecture for hybrid DRAM/PCM main memory systems//Proceedings of the International Conference on Computer Design. Amherst, USA, 2011: 381-387

[16] Chen C H, Hsiu P C, Kuo T W, et al. Age-based PCM wear leveling with nearly zero search cost//Proceedings of the 49th Design Automation Conference. San Francisco, USA, 2012: 453-458

[17] Zhong K, Zhu X, Wang T, et al. Dr. Swap: Energy-efficient paging for smartphones//Proceedings of the 2014 International Symposium on Low Power Electronics and Design. CA, USA, 2014: 81-86

[18] Boukhobza J, Rubini S, Chen R, et al. Emerging NVM: A survey on architectural integration and research challenges. ACM Transactions on Design Automation of Electronic Systems, 2018, 23(2): 1-32

[19] Lee S W, Moon B, Park C, et al. Accelerating in-page logging with non-volatile memory. Bulletin of the Technical Committee on Data Engineering, 2013, 33(4): 41-47

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

[21] Fiduccia C M, Mattheyses R M. A linear-time heuristic for improving network partitions//Proceedings of 19th Design Automation Conference. Las Vegas, USA, 1982: 175-181

[22] Luo H, Zhuge Q, Shi L, et al. Accurate age counter for wear leveling on non-volatile based main memory. Design Automation for Embedded Systems, 2014, 17(9): 543-564

[23] Malicevic J, Dulloor S, Sundaram N, et al. Exploiting NVM in large-scale graph analytics//Proceedings of the 3rd Workshop on Interactions of NVM/FLASH with Operating Systems and Workloads. New York, USA, 2015: 1-9

[24] Iwabuchi K, Sato H, Yasui Y, et al. NVM-based hybrid BFS with memory efficient data structure//Proceedings of the IEEE International Conference on Big Data. CA, USA, 2015: 529-538

[25] Shantharam M, Iwabuchi K, Cicotti P, et al. Performance evaluation of scale-free graph algorithms in low latency non-

volatile memory//Proceedings of the Parallel and Distributed Processing Symposium Workshops. Lake Buena Vista, USA, 2017: 1021-1028

[26] Xu Jin-Feng, Dong Yi-Hong, Wang Shi-Yi, et al. Summary of large-scale graph partitioning algorithms. Telecommunications Science, 2014, 30(7): 100-106(in Chinese)
(许金凤，董一鸿，王诗懿等. 大规模图数据划分算法综述. 电信科学，2014，30(7)：100-106)

[27] Masood S, Sheng B, Li P, et al. Automatic choroid layer segmentation using normalized graph cut. IET Image Processing, 2018, 12(1): 53-59

[28] Owik A, Ko M. Partitioning of VLSI circuits on subcircuits with minimal number of connections using evolutionary algorithm//Proceedings of the International Conference on Artificial Intelligence and Soft Computing. Zakopane, Poland, 2006: 470-478

[29] Stanton I, Kliot G. Streaming graph partitioning for large distributed graphs//Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Beijing, China, 2012: 1222-1230

[30] Andreev K, Racke H. Balanced graph partitioning//Proceedings of the 16th Annual ACM Symposium on Parallelism in Algorithms and Architectures. Barcelona, Spain, 2004: 120-124

[31] Li Qi, Zhong Jiang, Li Xue. DyBGP: A dynamic-balanced algorithm for graph partitioning based on heuristic strategies. Journal of Computer Research and Development, 2017, 54(12): 2851-2857(in Chinese)
(李琪，钟将，李雪. 基于启发策略的动态平衡图划分算法. 计算机研究与发展，2017，54(12)：2851-2857)

[32] Talu M F. Multi-level spectral graph partitioning method. Journal of Statistical Mechanics Theory & Experiment, 2017, 2017(9): 093406

[33] Arora S, Rao S, Vazirani U. Geometry, flows, and graph-partitioning algorithms. Communications of the ACM, 2008, 51(10): 96-105

[34] Benlic U, Hao J K. An effective multilevel tabu search approach for balanced graph partitioning. Computers & Operations Research, 2011, 38(7): 1066-1075

[35] Wang L, Xiao Y, Shao B, et al. How to partition a billion-node graph//Proceedings of the 2014 IEEE 30th International Conference on Data Engineering. Chicago, USA, 2014: 568-579

[36] Karypis G, Kumar V. Analysis of multilevel graph partitioning //Proceedings of the 1995 ACM/IEEE Conference on Supercomputing. San Diego, USA, 1995: 29

[37] Kernighan B W, Lin S. An efficient heuristic procedure for partitioning graphs. Bell Labs Technical Journal, 1970, 49(2): 291-307

[38] Tsourakakis C, Gkantsidis C, Radunovic B, et al. Fennel: Streaming graph partitioning for massive scale graphs// Proceedings of the 7th ACM International Conference on Web Search and Data Mining. New York, USA, 2014: 333-342

[39] Wang Zhi-Gang, Gu Yu, Bao Yu-Bin, Yu Ge. OnFlyP: An online distributed partition algorithm for large scale graphs based on edge-exchange model. Chinese Journal of Computers, 2015, 38(9): 1838-1851(in Chinese)
(王志刚，谷峪，鲍玉斌，于戈. OnFlyP：基于定向边交换的分布式在线大图划分算法. 计算机学报，2015，38(9)：1838-1851)

[40] Shi Z, Li J, Guo P, et al. Partitioning dynamic graph asynchronously with distributed fennel. Future Generation Computer Systems, 2017, 71: 32-42

[41] Kao E, Gadepally V, Hurley M, et al. Streaming graph challenge: Stochastic block partition//Proceedings of the High PERFORMANCE Extreme Computing Conference. Waltham, USA, 2017: 1-12

[42] Battaglino C, Pienta P, Vuduc R. Grasp: Distributed streaming graph partitioning//Proceedings of the High PERFORMANCE Graph Mining Workshop. Sydney, Austrilia, 2015

[43] Sajjad H P, Payberah A H, Rahimian F, et al. Boosting vertex-cut partitioning for streaming graphs//Proceedings of the IEEE International Congress on Big Data. San Francisco, USA, 2016: 1-8

[44] Nishimura J, Ugander J. Restreaming graph partitioning: Simple versatile algorithms for advanced balancing//Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. CA, USA, 2013: 1106-1114

[45] Ramos L E, Gorbatov E, Bianchini R. Page placement in hybrid memory systems//Proceedings of the International Conference on Supercomputing. AZ, USA, 2011: 85-95

[46] Wei W, Jiang D, Mckee S A, et al. Exploiting program semantics to place data in hybrid memory//Proceedings of the 2015 International Conference on Parallel Architecture and Compilation (PACT). New York, USA, 2015: 163-173

[47] Gill B S, Modha D S. WOW: Wise ordering for writes-combining spatial and temporal locality in non-volatile caches//Proceedings of the 4th Conference on USENIX Conference on File and Storage Technologies-Volume 4. CA, USA, 2005: 10

[48] Toyoda T. Information processing device that extends service life of non-volatile semiconductor memory and recording medium. USA, 2018.1.23

[49] Nazarian H, Nguyen S. Non-volatile memory with overwrite capability and low write amplification. USA, 2017.2.21

[50] Hu J, Xue C J, Tseng W C, et al. Minimizing write activities to non-volatile memory via scheduling and recomputation// Proceedings of the 2010 IEEE 8th Symposium on Application Specific Processors (SASP). DC, USA, 2010: 101-106

[51] Mauerer W. Professional Linux Kernel Architecture. Professional Linux Kernel Architecture. Wiley Pub. Birmingham, UK: Wiley, 2008

[52] Chen K, Jin P, Yue L. A survey on phase change memory-aware cache management. International Journal of Multimedia & Ubiquitous Engineering, 2016, 11(1): 293-310

[53] Lee S, Bahn H, Noh S H. CLOCK-DWF: A write-history-aware page replacement algorithm for hybrid PCM and DRAM memory architectures. IEEE Transactions on Computers, 2014, 63(9): 2187-2200

LI Qi, Ph.D. candidate. His current research interests include graph data mining and graph calculation.

ZHONG Jiang, Ph. D., professor. His research interests include data mining, parallel computing and natural language processing.

LI Xue, Ph. D., professor. His research interests include opinion analysis from social media, big data analytics, knowledge discovery from sequences, mining distributed, high-speed, time-variant data streams.

Background

Graph partitioning is becoming increasingly challenging as graphs grow in size. The graphs consist of terabytes of compressed data when stored on disks and are all far too large for a single commodity type machine to efficiently perform computations. Currently, the main use of distributed cluster systems to deal with large graph partitioning, although distributed computing resources have become more accessible, but the development of distributed partitioning algorithms still has challenges, especially for non-experts.

The streaming partitioning has been applied in graph partitioning in recent years because it is more efficient than offline partitioning. The latest Fennel algorithm performance is better than offline software METIS. This paper explores the use of the hybrid NVM and DRAM memory to solve streaming partitioning for large graphs in a commodity type machine. NVM storage has the advantages of low power consumption, high density, low latency and byte-addressable, but also has some disadvantages, such as writing power consumption is higher than reading, write latency is higher than read, and the write counts of NVM is limited.

The paper chiefly solves the following two problems: (1) Using the hybrid NVM and DRAM memory, to solve the problem of massive graph partitioning; (2) saving the data pages with different access characteristics in the appropriate memory space according to the characteristics of different memory media and the characteristics of streaming algorithm, so as to reduce the number of system migration operations and improve the system performance.

This work is supported by the National High Technology Research and Development Program (863 Program) of China under Grant (2015AA015308), the National Key Research and Development Program of China (2017YFB1402401), the Social Undertakings and Livelihood Security Science and Technology Innovation Funds of CQ CSTC (cstc2017shmsA0641), the Key Industries Common Key Technologies Innovation Projects of CQ CSTC (cstc2017zdcy-zdyxx0047), the Fundamental Research Funds for the Central University (2018CDYJSY0055), and the Technology Innovation and Application Demonstration project of CQ (cstc2018jszx-cyzdX0086).