计算机科学
COMPUTER SCIENCE

面向远程内存图数据库的应用感知分离式存储设计

李纯羽, 邓龙, 李永坤, 许胤龙

引用本文

李纯羽, 邓龙, 李永坤, 许胤龙. 面向远程内存图数据库的应用感知分离式存储设计[J]. 计算机科学, 2025, 52(1): 151-159.

LI Chunyu, DENG Long, LI Yongkun, XU Yinlong. Application-aware Disaggregated Storage Design for Remote Memory Graph Database [J]. Computer Science, 2025, 52(1): 151-159.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.231200073

面向远程内存图数据库的应用感知分离式存储设计

李纯羽[1]　邓　龙[1]　李永坤[1,2]　许胤龙[1,2]
1 中国科学技术大学计算机科学与技术学院　合肥 230026
2 高性能计算安徽省重点实验室　合肥 230026
(lichunyu@mail.ustc.edu.cn)

摘　要　图数据在各种应用中日益普及,其因涵盖多种实体类型和存在丰富的关联关系而备受关注。对于图数据库用户而言,高效的图查询服务是保障系统性能的关键因素。随着数据量增加,单机图数据库很难满足将所有数据存储在内存中的需求,而分布式图数据库在拓展性和资源利用率方面受到挑战。基于RDMA的远程内存系统的引入为克服这些挑战提供了一种新的选择,通过分离计算和存储资源,实现了更为灵活的内存使用方式。然而,在使用远程内存的情况下如何最大程度地优化图查询性能成为了当前研究的重点问题。文中首先分析了利用操作系统分页机制透明使用远程内存构建图数据库存在的问题,并在应用层次上设计了远程内存图数据库的存储模型。根据不同数据的特点和访问模式,设计了属性图在远程内存中的存储结构,优化了数据布局和访问路径。实验结果表明,在本地内存受限的情况下,与透明使用远程内存相比,应用感知的设计方式的端到端性能最高提升了12倍。

关键词:图查询;图数据库;图存储;远程内存;属性图模型
中图分类号　TP311

Application-aware Disaggregated Storage Design for Remote Memory Graph Database

LI Chunyu[1],DENG Long[1],LI Yongkun[1,2] and XU Yinlong[1,2]
1 School of Computer Science and Technology,University of Science and Technology of China,Hefei 230026,China
2 Anhui Province Key Laboratory of High Performance Computing,Hefei 230026,China

Abstract　Graph data is becoming more widespread across various applications due to its features to represent multiple entity types and rich associated relationships. For users of graph databases,efficient graph query service is crucial to ensure system performance. As the amount of data grows,the single-machine graph database is difficult to meet the demand of storing all the data in memory,while distributed graph databases,which spread the data across the memory of multiple machines,face challenges in scalability and resource utilization rate. A new solution to these challenges is the introduction of RDMA-based remote memory systems. These systems separate the tasks of computing and storing graph data,offering a more flexible way to use memory. Yet,a big challenge in current solutions is how to ensure the performance of graph query when using remote memory. This study takes a close look at the challenges that come up when building remote memory graph databases on general-purpose far memory platforms,which use remote memory transparently. It suggests a new approach,where the design of the remote memory graph database is aware of how it's being used. This design method creates a storage model that understands the different types of property graph data and how it's accessed. Specifically,the study explains how to make sure the data is arranged and accessed in the best way. Experimental results show that when the local memory is limited,the awareness method outperforms the transparent approach,giving a 12x improvement in graph query performance.

Keywords　Graph query,Graph database,Graph storage,Remote memory,Property graph model

1　引言

图数据在各类应用中越来越常见,这类数据的最大特点是包含多种类型的实体,实体之间存在丰富的关联关系。其中,社交网络是最常见的图数据之一,用户及其喜好以及用户之间的相关关系可以方便地用图模型表示。除此之外,商业关系、风险控制关系、知识图谱等都可以使用图数据表示,例如Neo4j[1],JanusGraph[2],TigerGraph[3],Nebula Graph[4]等常见的图数据库已经在各种应用中得到广泛部署和使用[5]。

对于图数据库使用者,图查询是最常见的工作负载

到稿日期:2023-12-11　返修日期:2024-05-13
基金项目:国家自然科学基金(62172382)
This work was supported by the National Natural Science Foundation of China(62172382).
通信作者:李永坤(ykli@ustc.edu.cn)

之一[6]。在知识图谱、社交网络等应用中，对数据的探索通常表现为图查询。常见图查询的访问模式是从一个或多个顶点出发，查询满足筛选条件的多跳范围内的邻居顶点。是否提供高效的图查询服务是图数据库性能的重要指标之一[7]。图查询的实现和优化在工业界和学术界的图数据库工作中受到了广泛关注[8-9]。

如果数据能够全部载入到内存中，图数据库可以获得最优的查询性能。然而，随着图数据量的增加，单机图数据库无法满足将所有数据存储在内存中的需求。一种纵向扩展的方法是使用存储容量更大、价格更低的存储介质(如磁盘)作为下一层存储。然而，由于CPU访问内存的速度远快于与磁盘交换数据的速度，这种扩展方式会导致应用性能严重下降。横向拓展为分布式系统是另一种选择，即通过增加节点将单机内存系统拓展为分布式系统。然而，这种方式涉及数据划分和计算任务划分，使得应用性能难以实现很好的线性扩展，节点间通信开展也随节点数量增加而增大。此外，分布式应用程序需要以完整节点为粒度扩展，每次扩展涉及固定的处理器和内存资源，无法动态调整内存大小。

除了以上两种拓展方式之外，考虑到数据密集型应用对内存资源的需求，设计数据密集型应用的一个重要趋势是将内存资源分离到远程节点并提供远程内存访问。这种远程内存架构通过将计算与存储资源分离，允许应用程序更加灵活地使用内存，提高集群的内存利用率[10]。如图1所示，这类系统往往由解耦的计算节点和存储节点通过高速网络连接构成，使用远程内存直接访问(Remote Direct Memory Access，RDMA)技术。更进一步地，计算节点内存资源受限，存储节点几乎没有计算能力。

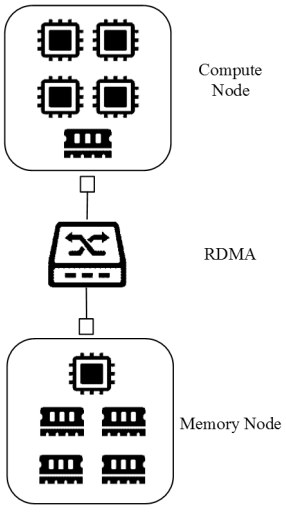


图1 远程内存架构图

Fig. 1 Architecture diagram of remote memory system

由于计算节点资源池化和硬件独立部署，远程内存架构能够实现内存资源解耦合，带来资源利用率提升、故障隔离和弹性增强等收益，能有效提升数据中心资源的灵活性，从而在云数据中心具有广泛的应用前景。已经提出的远程内存架构系统在提高系统内存利用率、提升任务总吞吐量、减小系统能耗等方面都取得了显著的效果[10]。与使用磁盘等存储介质拓展为单机系统相比，使用远程内存带来的性能下降更少，远程内存的灵活使用也能满足应用对内存规模的各种不同需求。与分布式系统相比，远程内存架构能够提高内存利用率，支持灵活按需使用内存。

拓展远程内存的一种直接方法是用远程内存空间替换原来的交换空间[11-15]，通过本地内存受限时引发缺页异常的方式，由操作系统内核决定换出或换入的内存数据。尽管这种方式对应用程序透明，无需改变应用程序的设计，但它带来了显著的操作系统上下文切换开销。由于操作系统无法了解应用程序的工作负载特征，当在远程内存系统运行图查询工作负载时，这种方式无法提供最佳性能。另一方面，图查询具有显著的数据访问特征，不同部分的数据格式不同，部分数据的访问频率比其他部分高很多。因此，我们可以构建应用感知的远程内存图数据库，在应用层为不同部分设计不同的存储结构，并根据访问频率将不同部分的数据存储在不同位置的内存中。

本文的主要贡献包括两个方面：

1)分析了现有的透明远程内存使用方式在构建图数据库时遇到的问题，并提出了一种应用感知的远程内存图数据库存储模型。根据图查询的访问模式和访问频率，在应用层次设计不同部分的存储结构，并根据远程内存架构的特点优化了各部分的访问流程。

2)在不同规模数据集下测试的结果表明，本文实现的应用感知的远程内存图数据库原型系统相比透明使用远程内存的方式实现了最高12倍的端到端查询性能提升。

本文第2章分析了透明使用远程内存时可能遇到的问题与挑战，以及我们选择以应用感知方式使用远程内存的原因；第3章介绍了本系统的设计、系统架构以及系统实现的相关细节；第4章通过实验分析了设计的效果；第5章介绍了相关工作；最后总结全文。

2 问题与挑战

目前，一种通用的使用远程内存的方式是应用无感知使用远程内存，通过拓展操作系统的分页机制，在不修改应用实现的情况下将其扩展为远程内存架构[11-15]。本章主要介绍典型的属性图存储结构，并分析将图数据库以应用无感知方式拓展至远程内存面临的问题与挑战。

2.1 典型属性图存储结构

在众多图表示方法中，带标签的属性图(Labeled Property Graph，LPG)是图数据库中最通用且表达性最强的图模型之一[16]。如图2所示，属性图模型包含了图拓扑结构(顶点、边等)及顶点和边包含的属性。顶点被赋予一个或者多个标签(类型)，同时顶点可以对应属性信息。标签将顶点分成若干集合，表示其在数据集中的作用。边(关系)将顶点连接起来并对图结构起到决定性的作用。带标签的属性图中的边往往是有向的，由一个起始顶点和结束顶点决定。同时，边也可以有对应的标签和若干属性。

使用属性图模型的图数据库通常采用邻接链表的存储结构，其中变长的数组或链表被用于存储邻居边，将顶点或边嵌套在对象中[9]。顶点对象除了包含邻居边外，还包括指向顶点属性等数据的指针。类似地，边对象也包含指向其他数据(如属性)的指针。为便于属性查询和数据划分，部分图数据

库采用键值对结构存储属性信息[17]。为了提高图查询速度，大多数图存储结构中都包含了顶点和边的索引[18]。

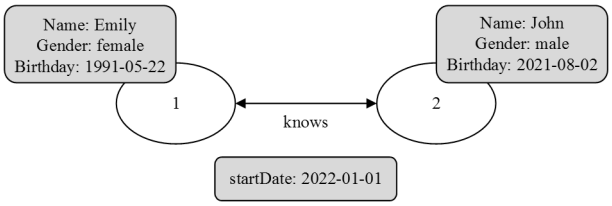


图 2 带标签属性图示例

Fig. 2 Example of labeled property graph

2.2 应用无感知透明使用远程内存

当前的主流做法是通过拓展操作系统分页机制，允许应用程序无感知地透明使用远程内存。如图 3 所示，这种方法对操作系统内存管理模块进行修改，以支持透明地使用远程内存。新的内存管理模块将虚拟地址空间的抽象扩展到远程节点内存，将远程内存作为本地内存的下一存储层。

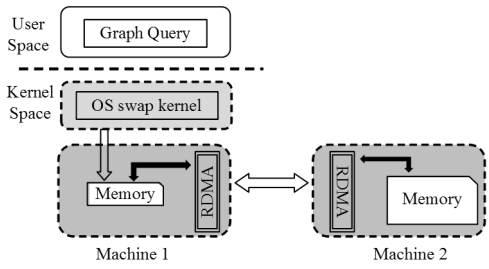


图 3 应用无感知使用远程内存

Fig. 3 Application unaware using remote memory

将使用 2.1 节中典型的图存储结构的图数据库运行在远程内存分页系统中，可以直接构建支持远程内存的应用无感知图数据库系统。在远程内存分页系统的协助下，图数据存储在本地节点或远程节点对于应用程序而言没有差异。当本地内存被占满时，部分页面被交换到远程内存中。当应用程序访问被交换出去的页时，操作系统通过调用 Page Fault 将在远程内存中的页交换回来，并再次映射到进程的页表中。图数据库应用在指定需要访问的内存地址后，无法感知是否经历了上述访问远程内存的步骤。因此，我们将这种基于远程分页系统的拓展方式称为应用无感知方式。由于修改发生在操作系统层，因此这种方法具有很强的通用性，应用程序无需修改任何代码即可在远程内存系统架构下运行。

2.3 应用无感知方式存在的问题

尽管应用无感知使用远程内存具有通用性，但对于数据密集型应用而言，随着应用使用远程内存比例的增加，换入换出操作变得更加频繁，无法支持高效图查询。具体而言，基于操作系统分页机制透明使用远程内存的应用存在以下问题：

1）处理 Page Fault 的额外开销

当应用程序请求访问的地址被换出到远程内存中，会触发操作系统缺页异常处理程序，将远程内存中的页复制到本地。除了实际传输数据的 IO 开销之外，Page Fault 的处理过程涉及查询缓存、回收页、修改页表等步骤，会产生额外的软件开销。即使是当前性能最优的远程分页系统，一次远程访问的软件开销导致的延迟也超过了总延迟的 50%[13]。对于远程内存图数据库，缺页异常不再是偶发的，其会随着远程数据量的增加变得更加频繁。尽管 Hermit[13]，Fastswap[14] 等工作引入了后台回收、异步、任务重叠等方式，以降低处理缺页异常的软件开销，在大部分数据放在远程内存的情况下仍无法避免图查询性能的下降。

2）读放大严重

Linux 分页机制以页为粒度换入换出，因此单次远程内存访问至少 4kB 的数据。图查询中远程访问涉及的数据包含顶点、边、邻居、属性等，数据规模可能从若干字节到上百字节。因此，以页为单位的数据访问将导致严重的读放大问题。然而，RDMA 技术本身对于数据传输的大小并无限制，以页为单位进行数据访问是由于操作系统内存管理机制的限制，可以通过应用程序感知的方式避免。

3）数据布局难以匹配图查询访问特征

除了分页机制的限制之外，我们进一步分析图数据库查询负载在无感知远程内存系统中遇到的挑战发现，现有的无感知方式导致数据布局难以匹配图查询工作负载的数据访问特征。

在只读查询工作负载下，不同数据的访问频率和访问模式有相当大的区别。与图拓扑信息相比，属性信息访问频率更低，但透明使用远程内存的方式难以在换出页时考虑到其冷热、访问模式等问题，导致被频繁访问的页难以固定在本地内存。除了图数据外，图数据库的计算引擎部分同样需要占用部分内存，在不加区分的情况下，计算引擎占用的内存页同样可能被换出到远程内存中，进一步增加查询执行的延迟。图 4 给出了在不同远程内存比例下执行图查询的端到端延迟结果，实验配置见 4.1 节。

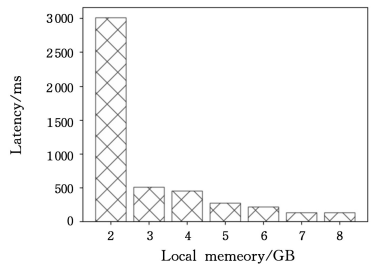


图 4 查询延迟与本地内存大小关系

Fig. 4 Query latency versus local memory size

使用 LDBC SNB[19]（社交网络基准）测试中的复杂交互式查询 IC4，该语句的语义为在指定类型为人的起始顶点之后，查询满足条件的该起始人的朋友创建的帖子相关的标签。通过修改本地内存限制来改变远程内存比例，统计该交互式查询语句端到端的延迟结果，测试结果如图 4 所示。当内存为 8GB 时，所有的图数据均在本地内存中，此时查询延迟为理论最优。当内存为 2GB 即本地内存限制为总内存需求的 25%时，大部分图数据均在远程内存中。

从延迟测试结果中也可以看出，随着本地内存逐渐减少，性能的下降趋势比较平缓。然而，当限制本地内存为总内存需求的 25%时，查询性能大幅度降低。由于此时无法感知应用的访问特征，图查询中频繁访问的数据可能被换出到远程内存，因此查询过程中涉及的远程内存访问次数大幅度增加。

2.4 构建图应用感知的远程内存系统

基于以上分析，我们认为在远程内存图数据库中，在应用层设计图存储结构，通过应用感知方式管理和访问图数据，可以优化图查询工作负载的性能。在数据加载阶段，应用将首先在内存节点分配并注册所需内存。我们根据查询工作负载的特点设计图数据的存储结构和访问方式。计算节点通过单边 RDMA 访问远程内存中的图数据，在无需远程节点 CPU 参与的情况下访问指定的远程内存地址。

由于在应用层面管理远程内存，因此避免了使用缺页异常方式访问远程内存，从而跳过了内核处理缺页异常的软件开销。另外，远程数据访问无需再以页为单位，可以执行完全符合数据规模的 IO。应用感知的方式允许根据数据冷热、访问模式等特点组织和管理图数据，从而获得更理想的查询性能。

3 系统设计

本章将详细介绍基于 RDMA 的远程内存图数据库系统的设计。首先介绍属性图存储的设计原则，然后分别介绍属性图存储设计和数据布局优化，最后介绍系统架构和系统实现的具体情况。

3.1 设计原则

基于带标签属性图模型的图数据库可以使用不同的底层存储模型。直接使用图结构存储的图数据库被称为原生图数据库[18]，它们专为图存储和计算而设计，特点是使用“无索引邻接”，可以更高效地获取顶点的连接信息。Neo4j，TigerGraph 等主流图数据库产品均属于原生图数据库，我们同样构建原生图数据库。在众多图的存储结构中，邻接链表是一种能够有效平衡内存开销和访问性能的数据结构，我们在邻接链表的基础上设计远程内存数据库的图存储结构。

与无感知透明使用远程内存的方式不同，我们希望考虑不同部分属性图数据的访问模式，并结合 RDMA 技术的特性，尽可能降低新增的远程内存访问对图查询性能的影响。主要遵循以下两个设计原则。

1）图拓扑存储与属性分离

在属性图查询过程中，访问模式通常为从顶点出发，访问若干邻居及其指定属性。与属性数据相比，图拓扑部分（顶点和边）的访问频率更高。此外，属性数据往往为变长的字符串格式，而拓扑部分可以使用相对固定的结构组织。考虑到访问模式和访问频率，在设计属性图存储时遵循图拓扑与属性分离的原则。

2）避免指针追踪

在传统的图数据库设计中，为了平衡访问与更新开销，设计者往往使用更加灵活的链表结构存储图拓扑。然而，在远程内存系统架构下，我们希望将大部分属性图数据存储在远程内存节点中。当邻居数据放在远程时，为了避免指针追踪以及频繁访问小数据，我们尽可能避免使用指针连接，以避免指针追踪的开销[20]。

3.2 属性图存储设计

遵循以上设计原则，本节主要介绍属性图各部分的存储结构。对于一个带标签的属性图模型，可以将其表达的信息表示为 $G=\langle V,E,P,T\rangle$，其中 G 表示属性图，V 表示节点集合，E 表示关系（边）集合，P 表示属性集合，T 表示标签集合。属性图各部分的存储结构如图 5 所示。

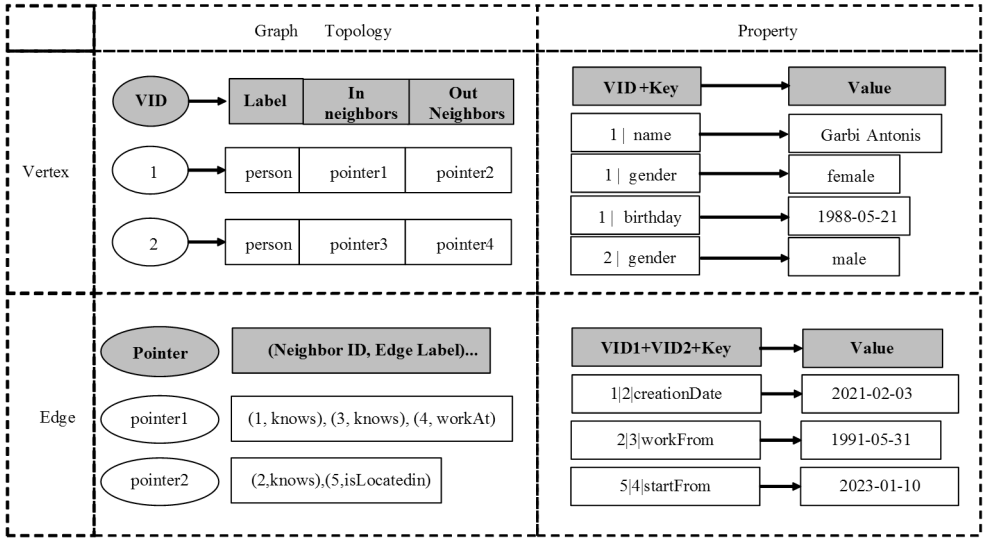


图 5 带标签属性图各部分存储结构

Fig. 5 Storage structure of labeled property graph

1）顶点信息

邻接表结构中，顶点信息可以视为边的索引。我们采用固定且简单的顶点结构。对于每一个顶点，该结构记录了顶点的类型、出边和入边邻居数组的位置。顶点信息记录了顶点 ID 与这一固定存储结构的映射关系。

2）顶点属性

对于属性图数据库中属性部分的存储，目前的一种主流方法是将其视为键值对集合[21-22]，采用键值存储结构。在查询过程中，属性又与图拓扑结构绑定在一起。因此，对于顶点属性，我们以顶点或边 ID 连接属性名称作为键。由于大部分属性值为变长类型的字符串，因此我们将存储字符串位置的指针作为值。为存储顶点属性的键值对，我们使用远程内存访问友好的哈希表结构，具体实现采用了 Cluster Hashing[23]。

3）边信息

不同于当前主流数据库中使用复杂的数据结构（如有序的树结构或链表等）存储图上的邻居信息，在远程内存系统中，我们的目标是尽可能减少查询中需要的追踪远程指针的次数，从而在尽可能少的远程访问中获得顶点的邻居信息。为此，我们选择使用变长邻居数组替代邻接链表。由于在包含多跳访问的图遍历过程中，访问顶点的邻居边时通常指定边的类型[17]，因此数组的每一项为邻居 ID 和边类型的二元组，从而支持快速筛选满足条件的邻居边。

4）边属性

与顶点属性类似，边属性同样可以看作是键值对的集合，采用键值结构进行存储。不同之处在于，其中键的部分由起点和终点 ID 加上属性名组成。我们同样使用哈希表存储这些键值对。

3.3 图查询友好的数据布局

在确定了属性图各部分数据存储结构之后，另一个目标是结合图查询负载的数据访问模式规划数据布局，从而实现应用感知的查询优化。

我们支持主流图查询语言 Gremlin[24] 实现交互式图查询，以 LDBC SNB 测试中的复杂交互式查询 IC3 为例，分析典型的图查询访问数据的特征。该语句的含义为在指定起始顶点的两跳朋友中，筛选在给定的时期内在指定的两个国家发表了评论的人员。图 6 给出了使用 Gremlin 实现该查询的示例及该查询访问数据的模式。访问流程中给出了依次查询的顶点、边以及每个顶点访问的属性和属性筛选条件。

在访问流程中，我们发现在典型的多跳图查询中，需要按顺序执行邻居访问和属性查询步骤。因此，我们分别优化邻居访问和属性查询。

```
g.V().has("id",$ID).as('a').union(
        both("knows"),
        both("knows").both("knows")
    ).where(neq('a')).dedup().not(
        out("isLocatedIn").out("isPartOf").
        or(has("name",$COUNTRY1),
            has("name",$COUNTRY2))).as("person").in("hasCreator").
    has("creationDate",between(&START_DATE,$END_DATE)).
    or(out("isLocatedIn").has("name",$COUNTRY1),
        out("isLocatedIn").has("name",$COUNTRY2)).select("person").groupCount()
```

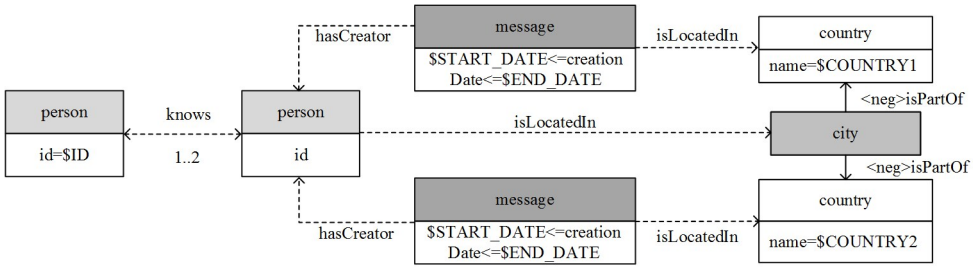


图 6 查询语句 IC3 的 Gremlin 实现及其数据访问模式图

Fig. 6 Gremlin implementation of IC3 and its data access pattern

1）邻居访问

典型的步骤如 IC3 中的 both，in，out 等，它们表示访问入边或出边。访问邻居需要首先查询顶点信息获得数组指针，再通过指针完成访问及筛选。为了加速查询，我们将顶点信息存储在计算节点内存中，仅需一次网络请求即可完成邻居访问。由于顶点结构简单，这部分的空间开销是可以接受的。

2）属性查询

典型的步骤如 IC3 中的若干 has 步骤，用于筛选满足条件的顶点或边。图拓扑的访问可以被视为属性值的索引，通过图的连接关系快速找到满足条件的邻居。在完成邻居访问后，查询属性值仍需经过访问属性列表、查询哈希表和访问指针等步骤。其中，访问属性列表步骤指根据类型查询该顶点或边包含的属性的名称，以避免查询语句中指定的属性不存在。另外，Gremlin 中部分属性查询步骤支持不指定参数即不提供具体的属性名称，对于这类查询，必须先访问该顶点或边的所有属性列表。为了减少查询属性名的远程内存访问，我们在计算节点中存储了类型与其对应的属性名列表（schema）的映射关系。该映射关系存储开销小，且可以实现对属性查询步骤的优化。

根据上述的分析与设计，如图 7 所示，我们在计算节点内存和远程内存中存储了与属性图查询相关的各部分数据。

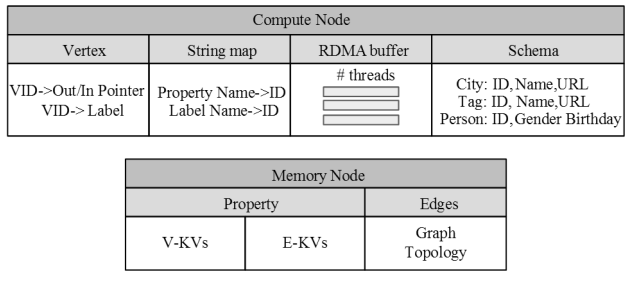


图 7 计算和存储节点数据布局

Fig. 7 Data layout of computing and memory nodes

计算节点内存包含顶点信息、属性名列表、RDMA 缓存以及部分字符串映射。其中，字符串映射充当属性图的元数据，包含属性名称和类型名称等字符串的映射，RDMA 缓存

用于在远程通信中发送和接收数据。存储在计算节点内存的数据具有固定结构、频繁访问和小存储开销的特点，能够有效降低图查询访问远程内存的次数。内存节点存储了邻接边信息、顶点属性和边属性，这部分数据具有较大的存储开销和相对较低的访问频率。通过设计不同的存储结构，能够减轻计算节点的内存压力，并更灵活地利用远程内存。

在存在多个存储节点的情境中，由于所有图数据(包括顶点、边、顶点属性和边属性)都采用了整型编码作为 ID，我们能够通过一致性哈希将数据打散并划分到多个内存节点中。在访问图数据时，计算节点指定 ID，根据 ID 值的哈希计算结果定位对应的内存节点。计算节点与每个内存节点在启动阶段可以建立独立的 RDMA 链路。通过查询索引，计算节点可以直接在相应的内存节点上访问边，或者查询相应内存节点中的属性值。

3.4 系统实现

我们在开源图数据库 Grasper[17] 的基础上实现了远程内存图数据库原型，其系统架构如图 8 所示。该原型系统由解耦合的计算节点和远程内存节点组成，通过与图查询客户端的交互执行图查询语句并返回结果。计算节点包含图查询语言处理接口、解释器、执行计划生成和执行器等模块。计算引擎通过访问图存储层接口实现图查询中的数据访问。

在原型系统中，我们采用了 Grasper 的计算引擎，而其他部分则是新增实现。计算节点记录的字符串映射、属性列表和顶点信息均使用本地哈希表进行存储。内存节点的 CPU 在完成内存注册和数据加载后无需再参与后续查询过程。图的拓扑结构和属性按照 3.2 节中描述的边和属性结构进行存储。如图 8 中虚线框部分所示，将计算引擎对不同部分图数据的访问封装在多个访问接口中，每个访问接口都调用 RDMA 通信模块，以实现对内存节点的数据访问。

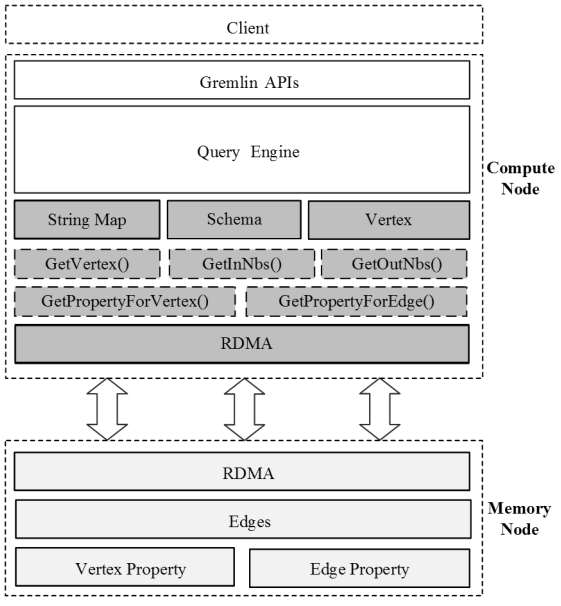


图 8 系统架构

Fig. 8 System architecture

为实现计算节点与内存节点的通信，引入了 RDMA 通信模块，采用单边传输的方式，绕过远程节点的 CPU 直接访问内存。RDMA 通信模块采用线程级通信通道，N 个 SendBuffer 和 N 个 RecvBuffer 形成一个 RDMA 缓存区，其中 N 是计算引擎使用的工作线程数。每个 $SendBuffer[i]$ 负责相应工作线程 i 的 $RDMA$ 请求发送，$RecvBuffer[i]$ 保存从内存节点接收到的消息并提供给工作线程 i。线程级 RDMA 通信信道避免了加解锁操作带来的额外开销，其充分利用 RDMA 带宽，以实现低延迟和高吞吐的查询性能。

RDMA Doorbell batch[25] 机制支持批处理多个 RDMA 读请求，允许在一次远程访问中包含多个读的目的地址。在采用 BFS 方式的图查询[7]中，每次需要访问的邻居数量可能较大。通过批处理多个 RDMA 读请求，可以降低远程内存访问的次数，提高查询性能。

4 实验

本章将应用感知的远程内存图数据库与无感知方式进行对比，并基于各项性能指标对系统设计进行评估。首先介绍实验环境、数据集及系统配置；然后使用查询延迟、吞吐量和尾延迟等评估指标对比两个系统，分析导致性能结果的原因，以验证系统设计的有效性；最后测试各部分数据的内存占用情况。

4.1 实验配置

实验在多核和 RDMA 环境下进行，配置了两块 Intel(R) Xeon(R)Gold 5218R CPU @ 2.10GHz20 核心 CPU，共 80 个物理线程，每个 NUMA 节点的内存容量为 96 GB。操作系统为 Ubuntu20.04LTS，Linux 内核版本为 5.5.0，GCC 版本为 9.4.0。服务器采用 Mellanox 100Gb ConnectX-6 Dx RDMA 网卡，通过局域网内的 RDMA 网络进行通信。

实验采用 LDBC SNB[19](社交网络基准)测试，该基准测试是业界权威的图数据库和图数据管理系统的参照标准之一。LDBC 定义了一系列尺度因子(Scale Factor)来衡量数据的大小。我们关注属性图查询工作负载，使用 3 个不同尺度因子生成的数据集，每个规模数据集的顶点、边及属性数量如表 1 所列。在实验中，选择了两类共 8 个典型查询：1)交互式复杂只读查询 IC1-IC4，这些查询从指定顶点出发，范围涉及 3～5 跳邻居的查询和属性筛选，因此访问的图数据量相对较大；2)交互式简单查询 IS1－IS4，这类查询只涉及属性或一跳邻居。

表 1 LDBC-SNB 数据集

Table 1 LDBC-SNB dataset

Dataset	# V	# E	# V Property	# E Property
SF-1	3.96×10^6	23.03×10^6	23.97×10^6	21.63×10^6
SF-3	11.41×10^6	69.42×10^6	69.52×10^6	67.23×10^6
SF-10	36.49×10^6	231.37×10^6	223.19×10^6	228.59×10^6

我们对比了 3 种内存使用方式在图查询性能上的差异，分别是：

1)Local Only(本地内存)：所有数据存储在本地内存中。

2)Transparent Remote(透明远程内存)：基于分页机制，实现对远程内存的透明使用，应用无感知。我们在当前最优的远程分页系统之一 Canvas[11] 上运行了 Grasper。Canvas

是在 Linux 内核版本 5.5.0 的基础上进行修改，支持使用远程内存的分页系统。我们通过 Linux 的 Cgroups 机制限制本地内存大小，实验过程中不限制远程内存的大小，通过调整本地内存大小来控制远程内存的使用比例。

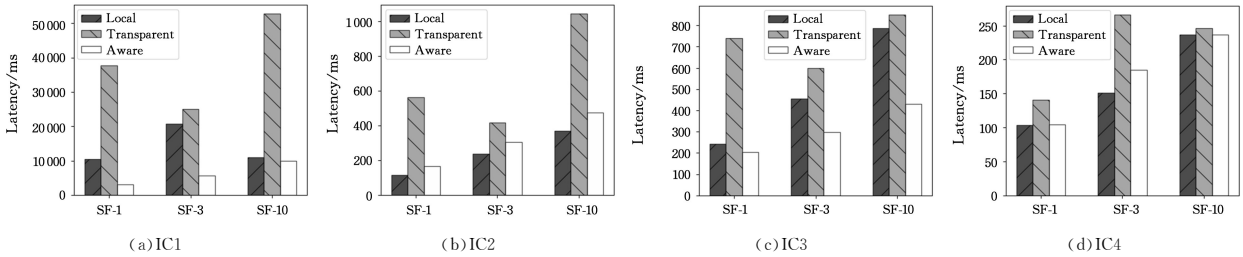
3）Aware Remote（应用感知远程内存）：本文实现的应用感知的远程内存图数据库原型。Transparent Remote 和 Aware Remote 本地内存限制相同，都提供充足的远程内存。鉴于计算引擎部分同样需要占用本地内存，我们将该限制设置为 Local Only 方式的 35%。

4.2 延迟比较

本节比较了 Local Only，Transparent Remote 和 Aware Remote 这 3 种方法的图查询端到端延迟情况，对不同规模图数据下的查询性能进行比较。使用的查询语句是 IC1 到 IC4，重点测试多跳查询负载下的查询延迟。

延迟比较结果如图 9 所示，对于每个数据集的每个查询请求，在本地内存受限的情况下，与 Transparent Remote 相比，Aware Remote 的图查询端到端延迟更短，最高可达到 12 倍的性能优势。这是因为我们使用应用感知的方式优化查询的关键路径，即邻居查找和属性查询。计算节点内存中的顶点信息和属性列表等被频繁访问的数据，不会被换出到远程内存，与 Transparent Remote 相比减少了大量远程访问。

即使限制本地内存，与 Local Only 相比，Aware Remote 的性能下降也在可接受范围内。除了由于 RDMA 访问的性能优势外，我们将类型、属性列表等结构单独保存、独立查询，而 Grasper 实现中需要通过查询属性或访问链表才能获得这些数据。



(a)IC1 (b)IC2 (c)IC3 (d)IC4

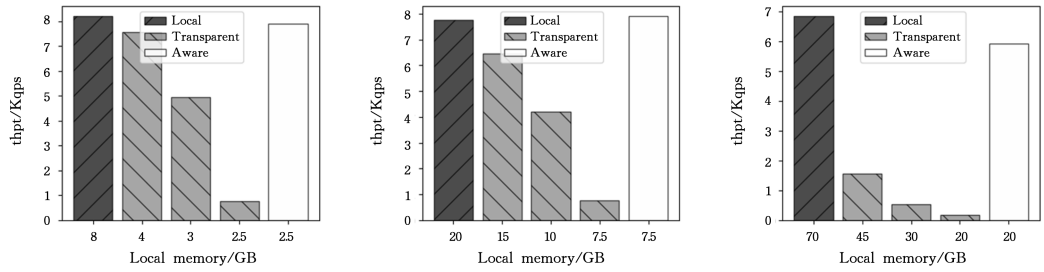
图 9 延迟比较（IC1－IC4）

Fig. 9 Latency comparison（IC1－IC4）

4.3 吞吐率和尾延迟比较

在面向只读查询负载的图数据库中，除了端到端延迟之外，查询吞吐率同样是重要的性能指标。为了测试查询的并发性能，我们使用了 LDBC SNB 基准测试中的简单类型查询（IS）作为查询模板，测试了系统每秒钟能够处理的查询数量。图 10 对比了 Local Only，Transparent Remote 和 Aware Remote 系统吞吐率的结果。

更进一步地，对于 Transparent Remote 方式，我们改变本地内存限制，内存限制大小如图 10 中横坐标所示；Aware Remote 方式的内存限制与 Transparent Remote 最小的内存限制相同。结果表明，在不同大小数据集下，Aware Remote 的系统吞吐率均优于 Transparent Remote 方式，并至少达到 Local Only 吞吐率的 87%。随着数据集规模增加，Transparent Remote 吞吐率受限的情况逐渐明显，在本地内存相同的情况下，使用 SF-10 数据集，Aware Remote 方式实现了 5.9 kps 的查询吞吐率，而 Transparent Remote 的吞吐率仅为每秒 190 个查询。除了由于查询中极少存在热数据的远程访问外，Aware Remote 原型系统的 RDMA 通信模块使用多线程方式提升系统的并发能力，因此获得了更高的查询吞吐率。



(a)SF-1 (b)SF-3 (c)SF-10

图 10 吞吐比较（IS1－IS4）

Fig. 10 Throughput comparison（IS1－IS4）

除了以上指标之外，在 Transparent Remote 系统中，由于关键路径上访问的数据被换出到远程内存，因此部分查询的尾延迟严重增加。更进一步地，我们画出使用 SF-10 数据集各查询语句延迟的累积分布函数（CDF）曲线，以分析图查询的尾延迟情况。

实验结果如图 11 所示，其中每张图代表一个查询语句。结果表明，Transparent Remote 系统的第 99 百分位延迟是 50 百分位延迟的最高 26 倍，长尾延迟现象严重。这是因为热数据被换出，即使是简单查询，也会被远程访问严重阻塞。相比之下，在 Aware Remote 系统中，图查询不会受到操作系统换出内存及内存回收的影响，累积分布函数曲线表明未出现长尾现象。该结果表明，应用感知方式在处理大量查询时不会出现饥饿现象，在处理高并发方面更加有优势。

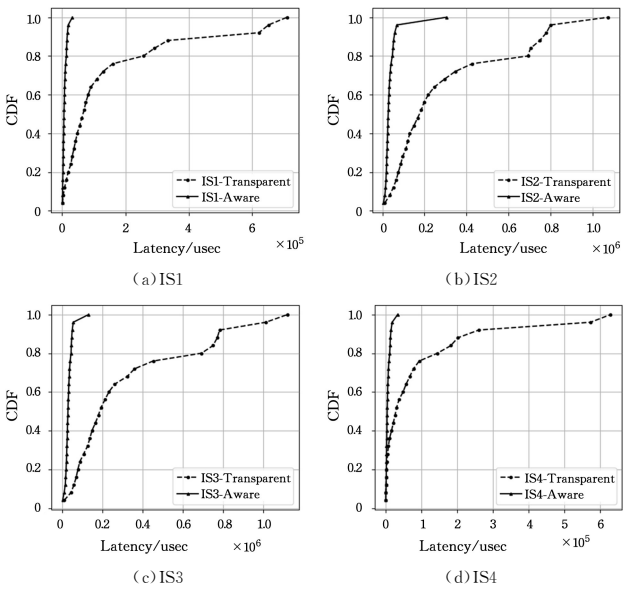


图 11 尾延迟比较(IS1－IS4)

Fig. 11 Tail latency comparison(IS1－IS4)

4.4 内存开销分析

本节主要分析本系统的各部分数据内存占用情况。在启动和数据加载阶段,系统根据配置文件在远程内存节点分配指定大小的内存空间用于存储图数据。图 12 详细给出了顶点数据、边(邻居)数据以及顶点和边的属性数据的空间占用情况。其中顶点数据包含了顶点 ID 到顶点类型和邻居数组地址的映射,保存在计算节点的内存中。属性列表和字符串映射等同样被记录在计算节点内存中,但是这部分数据与图数据集的规模无关,而且内存占用有限,因此未做统计。

如图 12 所示,由于顶点数据尺寸小且结构固定,这部分存储开销较小,仅占属性图数据内存存储总开销的 5%左右,但由于其可能多次出现在图查询的数据访问关键路径中,因此我们得以用相对较小的空间开销换取更好的查询性能。这种分配和利用内存的策略有助于在远程内存架构下实现高效的图查询,同时最大限度地减少不必要的内存占用。通过合理配置内存,能够在保证查询性能的同时,尽可能少地占用本地内存。

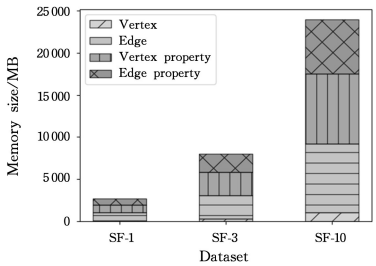


图 12 属性图存储开销统计

Fig. 12 Memory usage statistics of property graph

5 相关工作

5.1 基于 RDMA 的分布式图数据库

分布式图数据库侧重于通过拓展节点实现性能的有效拓展。由于 RDMA 网络的性能优势,许多分布式图数据库专门针对 RDMA 网络进行设计和优化。

A1[9]是基于远程内存管理平台 FaRM[26]实现的分布式图数据库。FaRM 为上层提供了一致性事务支持和基于对象的内存管理接口。A1 使用固定的数据结构表示顶点和边,该结构包含指向与顶点和边关联的属性和其他元素的指针。A1 通过 RPC 调用支持分布式图查询。G-Tran[22]是另一个支持 RDMA 的分布式图数据库,其使用非共享方式存储图的拓扑结构,而将属性数据存储在共享内存中,同时支持单边 RDMA 远程访问其他节点的属性信息。此外,G-Tran 重点关注图数据库事务的特性。ByteGraph[21]设计了更高效的邻接链表存储结构,在邻接链表中使用更大的内存分配粒度。GDI[27]根据图数据库关键模块设计了可移植可编程的图数据库接口,并基于图数据库接口设计了适用于分布式内存 RDMA 体系结构的分布式内存图数据库。

这些分布式图数据库在存储结构上考虑了远程内存访问的特点,实现了部分结构的单边 RDMA 访问,减少了额外的远程访问数量。然而,由于它们的分布式系统特征,每个节点都有计算能力,以完整节点为粒度增加资源,因此不能充分适应基于远程内存的远程内存架构特征,并且难以实现灵活的资源拓展和内存使用。

5.2 远程内存图计算系统

近期的研究将图计算系统扩展到远程内存,代表性的工作如下:Fargraph[28-29]分析了图计算应用工作负载,将频繁访问的热数据放在本地,通过并行计算与传输过程尽可能隐藏远程访问的延迟,从而实现了低延迟的远程内存图计算系统。FAM-Graph[30]优化了远程内存下的图处理系统,对于 CSR 格式的图数据,将顶点部分存储在本地内存,边数据存储在远程节点,充分利用图算法中顶点读取远远超过边的特点。它通过批量处理和流水线等优化实现了使用 OS 级别远程内存运行时库 1～6 倍的图算法性能和 1/20 的本地内存。

这些工作与本文有一些相似的思想,主要都是通过分析工作负载和数据访问冷热来合理安排存储位置。然而,图计算系统主要使用 CSR 格式存储图,不包含属性等数据,因此在基于属性图模型的图系统中可能不太适用。

结束语 本文提出了一种创新性的应用感知的远程内存图数据库设计方法。通过深入分析基于操作系统 swap 机制透明使用远程内存存在的性能问题和图查询访问数据的特征,为远程内存图应用设计了一种高效的属性图存储结构,并提出了针对性的缓存和访问路径优化方法。实验结果表明,相比于透明使用远程内存的方法,本文提出的应用感知设计在端到端图查询性能上取得了最多 12 倍的显著提升。

未来希望将原型系统拓展至支持多计算节点和多内存节点的场景,探讨远程内存图数据库在系统设计和可拓展性等方面的问题,进一步研究数据密集型应用在分离式内存架构下的性能、资源利用率和弹性容错等方面的问题。

参 考 文 献

[1] Neo4J, Inc. Neo4J[OL]. 2007[2023-11-21]. https://neo4j.com/.

[2] JanusGraph. JanusGraph[OL]. 2017[2023-11-21]. https://janusgraph.org/

[3] DEUTSCH A, XU Y, WU M, et al. Tigergraph: A native MPP graph database[J]. arXiv:1901.08248, 2019.

[4] WU M, YI X, YU H, et al. Nebula Graph: An open source dis-

tributed graph database[J]. arXiv:2206.07278,2022.
[5] LIU Q,LI Y,DUAN H,et al. Knowledge Graph Construction Techniques[J]. Journal of Computer Research and Development,2016,53(3):582-600.
[6] SU L,QIN X,ZHANG Z,et al. Banyan:a scoped dataflow engine for graph query service[J]. arXiv:2202.12530,2022.
[7] SAHU S,MHEDHBI A,SALIHOGLU S,et al. The ubiquity of large graphs and surprising challenges of graph processing:extended survey[J]. The VLDB journal,2020,29:595-618.
[8] CHEN H,WU B,DENG S,et al. High performance distributed OLAP on property graphs with grasper[C]// Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. 2020:2705-2708.
[9] BURAGOHAIN C,RISVIK K M,BRETT P,et al. A1:A distributed in-memory graph database[C]// Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. 2020:329-344.
[10] EWAIS M,CHOW P. Disaggregated Memory in the Datacenter:A Survey[J]. IEEE Access,2023,11:20688-20712.
[11] WANG C,QIAO Y,MA H,et al. Canvas:Isolated and Adaptive Swapping for {Multi-Applications} on Remote Memory[C]// 20th USENIX Symposium on Networked Systems Design and Implementation(NSDI 23). 2023:161-179.
[12] RUAN Z,SCHWARZKOPF M,AGUILERA M K,et al. {AIFM}:{High-Performance},{Application-Integrated} far memory[C]// 14th USENIX Symposium on Operating Systems Design and Implementation(OSDI 20). 2020:315-332.
[13] QIAO Y,WANG C,RUAN Z,et al. Hermit:{Low-Latency},{High-Throughput}, and Transparent Remote Memory via {Feedback-Directed} Asynchrony[C]// 20th USENIX Symposium on Networked Systems Design and Implementation(NSDI 23). 2023:181-198.
[14] AMARO E,BRANNER-AUGMON C,LUO Z,et al. Can far memory improve job throughput? [C]// Proceedings of the Fifteenth European Conference on Computer Systems. 2020:1-16.
[15] GU J,LEE Y,ZHANG Y,et al. Efficient memory disaggregation with infiniswap[C]// 14th USENIX Symposium on Networked Systems Design and Implementation(NSDI 17). 2017:649-667.
[16] ANGLES R. The Property Graph Database Model[C]// AMW. 2018.
[17] CHEN H,LI C,FANG J,et al. Grasper:A high performance distributed system for OLAP on property graphs[C]// Proceedings of the ACM Symposium on Cloud Computing. 2019:87-100.
[18] BESTA M,GERSTENBERGER R,PETER E,et al. Demystifying graph databases:Analysis and taxonomy of data organization,system designs,and graph queries[J]. ACM Computing Surveys,2023,56(2):1-40.
[19] ANGLES R,ANTAL J B,AVERBUCH A,et al. The LDBC social network benchmark[J]. arXiv:2001.02299,2020.
[20] AGUILERA M K,KEETON K,NOVAKOVIC S,et al. Designing far memory data structures:Think outside the box[C]// Proceedings of the Workshop on Hot Topics in Operating Systems. 2019:120-126.
[21] LI C,CHEN H,ZHANG S,et al. ByteGraph:a high-performance distributed graph database in ByteDance[J]. Proceedings of the VLDB Endowment,2022,15(12):3306-3318.
[22] CHEN H,LI C,ZHENG C,et al. G-tran:a high performance distributed graph database with a decentralized architecture[J]. Proceedings of the VLDB Endowment,2022,15(11):2545-2558.
[23] WEI X,SHI J,CHEN Y,et al. Fast in-memory transaction processing using RDMA and HTM[C]// Proceedings of the 25th Symposium on Operating Systems Principles. 2015:87-104.
[24] RODRIGUEZ M A. The gremlin graph traversal machine and language(invited talk)[C]// Proceedings of the 15th Symposium on Database Programming Languages. 2015:1-10.
[25] KALIA A,KAMINSKY M,ANDERSEN D G. Design guidelines for high performance {RDMA} systems[C]// 2016 USENIX Annual Technical Conference(USENIX ATC 16). 2016:437-450.
[26] DRAGOJEVIĆ A,NARAYANAN D,CASTRO M,et al. {FaRM}:Fast remote memory[C]// 11th USENIX Symposium on Networked Systems Design and Implementation(NSDI 14). 2014:401-414.
[27] BESTA M,GERSTENBERGER R,FISCHER M,et al. The Graph Database Interface:Scaling Online Transactional and Analytical Graph Workloads to Hundreds of Thousands of Cores[C]// Proceedings of the International Conference for High Performance Computing,Networking,Storage and Analysis. 2023:1-18.
[28] WANG J,LI C,WANG T,et al. Excavating the potential of graph workload on rdma-based far memory architecture[C]// 2022 IEEE International Parallel and Distributed Processing Symposium(IPDPS). IEEE,2022:1029-1039.
[29] WANG J,LI C,LIU Y,et al. Fargraph+:Excavating the parallelism of graph processing workload on RDMA-based far memory system[J]. Journal of Parallel and Distributed Computing,2023,177:144-159.
[30] ZAHKA D,GAVRILOVSKA A. FAM-Graph:Graph analytics on disaggregated memory[C]// 2022 IEEE International Parallel and Distributed Processing Symposium(IPDPS). IEEE,2022:81-92.

LI Chunyu, born in 1999, postgraduate. Her main research interests include graph database and disaggregated memory architecture.

LI Yongkun, born in 1986, professor, is a member of CCF(No. 33184M). His main research interests include I/O optimization for data-intensive computing and storage systems, parallel and distributed systems, key-value systems, and cloud computing, etc.

(责任编辑:何杨)