小 型 微 型 计 算 机 系 统
Journal of Chinese Computer Systems
DOI:10.20009/j.cnki.21-1106/TP.2025-0034
2026 年 2 月 第 2 期
Vol.47 No.2 2026

PGCA-RAG:面向大语言模型检索增强的并行图缓存架构

郭宇豪,陈庆奎

(上海理工大学 光电信息与计算机工程学院,上海 200093)
E-mail:chenqingkui@usst.edu.cn

摘 要: 目前大语言模型在检索增强(Retrieval-Augmented Generation,RAG)领域大部分工作都着重于提高检索能力提高召回精度,以及结合大语言模型对知识图谱进行构建.但是在面对实际工程化任务时,RAG 系统往往会面临高并发场景,目前的 RAG 系统存在知识图谱数据检索速度慢,系统并发能力差等问题.为了克服这些局限性,本文提出了一种面向大语言模型检索增强的并行图缓存架构(PGCA-RAG),和一种面向大语言模型检索增强的动态缓存图数据模型知识图缓存单元(KGCU),能够在 RAG 系统中表现出良好的适用性,并基于 VoltDB 实现了并行缓存调度算法.通过仿真对比实验结果表明相比于现有架构,PGCA-RAG 在多轮查询检索速度指标上提高了 82.8%,同时在并发测试中性能保持领先,较好的证明了该并行图缓存架构的合理性和有效性.

关 键 词: 并行架构;图缓存;大语言模型;检索增强;知识图谱
中图分类号: TP391 **文献标识码:** A **文 章 编 号:** 1000-1220(2026)02-0336-07

PGCA-RAG:a Parallel Graph Caching Architecture for Large Language Model Retrieval-augmented Generation

GUO Yuhao,CHEN Qingkui
(College of Optoelectronic Information and Computer Engineering,University of Shanghai for Science and Technology,Shanghai 200093,China)

Abstract: At present,most of the work in the field of Retrieval-Augmented Generation (RAG) of Large Language Model focuses on improving retrieval ability to improve recall precision,as well as combining with LLM for the construction of knowledge graphs. However,RAG systems often face high concurrency scenarios when facing real engineering tasks,and current RAG systems suffer from slow knowledge graph data retrieval and poor system concurrency. In order to overcome these limitations,this paper proposes a parallel graph caching architecture for large language model retrieval enhancement(PGCA-RAG),and a dynamic caching graph data model knowledge graph cache unit(KGCU) for large language model retrieval enhancement,which show good applicability in RAG systems,and implement a parallel caching scheduling algorithm based on VoltDB. Finally,The simulation comparison experiment results show that compared with the existing architecture,the architecture proposed improves the speed index of multi-round query retrieval by 82.8%,and at the same time,the performance in the concurrency test is significantly ahead of the others,which is a better proof of the reasonableness and validity of the parallel graph caching architecture proposed.

Keywords: parallel architecture;graph caching;large language models;retrieval enhancement;knowledge graphs

0 引 言

近些年随着生成式人工智能领域的快速发展,大语言模型(LLM)呈现爆炸式增长,模型参数已经达到千亿级别[1,2],在面对各种自然语言处理下游任务时,如文本生成[3],摘要任务[4]等,表现出了卓越的性能.然而,尽管 LLMs 在处理各种自然语言任务方面取得了巨大进展,但幻觉生成[5,6]和知识近因问题在现有的 LLMs 中仍然普遍存在,即使是最优秀的 LLMs 之一如 GPT-4[7],为了缓解上述问题,校准调优策略已在现有工作中得到了广泛应用[8],此外整合外部知识源为大语言模型提供可靠的信息补充同样可以大大缓解上述问题[9,10].

由于 LLMs 存在幻觉问题以及特定专业领域知识的缺乏,严重影响了其在高风险场景中的应用,如医疗领域和企业私有数据的知识密集型任务,目前一种潜在的范式是知识图谱检索增强生成(Graph Retrieval-augmented Generation,Graph RAG),将可提供精确事实知识的知识图谱(Knowledge Graphs,KGs)与 LLMs 进行集成[11,12]以此来提高 LLMs 生成内容的质量,而目前 Graph RAG 大量的相关工作主要围绕构建知识图谱提高检索精度[13-15],在面对工程化使用场景中,能够快速的从知识图谱中检索数据同样是至关重要的,目前知识图谱存储方案主要包括:关系存储和原生图存储[16],当查询并发量较大,选择性较大时,两种存储方案查询性能都明显下降,同时当用户的查询具有一定的重复性时,上述存储方案存在一定的局限性.

因此基于上述研究,本文提出了一种面向大语言模型检

收稿日期:2025-01-14 收修改稿日期:2025-03-13 基金项目:国家自然科学基金项目(61572325)资助;上海重点科技攻关项目(19DZ1208903)资助. 作者简介:**郭宇豪**,男,2000 年生,硕士研究生,研究方向为大模型、知识图谱、计算机集群;**陈庆奎**(通信作者),男,1966 年生,博士,教授,博士生导师,CCF 高级会员,研究方向为网络计算与并行计算、人工智能、大规模物联网等.

索增强的并行图缓存架构,并针对不同的知识图谱子图数据特征提出了一种适用于 Graph RAG 系统的数据模型:知识图缓存单元,能够更好的适应高并发场景,同时架构支持采用分布式设计,缓存节点能够灵活的进行调整,具有良好的可扩展性. 综上所述本文的主要贡献包括两个方面:

1)提出了面向大语言模型检索增强的知识图缓存单元(Knowledge Graph Cache Unit,简称 KGCU 模型):一种适用于 Graph RAG 系统的动态缓存图数据模型,该数据模型不但良好的映射了知识图谱数据,同时结合了 Graph RAG 系统的特点对用户级数据进行了整合,能够极大的提高系统访问效率.

2)提出了一种新的检索增强并行图缓存结构(Parallel Graph Cache Architecture for Retrieval-augmented Generation,简称 PGCA-RAG 模型),同时采用了一种新颖的并行缓存调度算法.

1　相关工作

1.1　知识图谱存储

知识图谱的存储方式可以大致分为两种:基于关系型结构的存储方式和原生图结构的图存储方式. S. Harris 等人[17]最早提出基于三元组表的关系型数据库存储方案,将图数据表示为具有主语、谓语、宾语结构的三元组数据,但会存在多跳查询时产生自连接的问题. Z 等人[18]提出了将每行数据存储一个主语的所有谓语和宾语的水平表存储方式,但面对真实大规模图数据时,每一行数据往往会产生大量的空值列,严重影响了关系表的查询性能,为了改进该方案,K. Wilkinson[19]等人提出了属性表关系存储方案,该方案基于水平表存储方式将同类型主语存储在同一张表中,解决了空值过多的问题,但在大规模知识图谱数据集中,主语的类别数目往往成千上万,因此可能造成所创建的表数目超过关系数据库的存储上限. D. J. Abadi 等人[20]提出了垂直分区关系存储方案,与属性表不同的是该方案针对每一种谓语创建一张两列的表. 而 G. Weikum[21],P. Karras[22]等工作提出采用六元组索引关系存储方式,针对三元组的所有列创建六张表,可以大大缓解三元组存储方式中所产生的自连接问题. W. Sun 等人[23]通过分析属性表关系存储方案和垂直分区关系存储方案的特性,提出了创建直接哈希表,反向哈希表的存储方案,能够大大提高知识图谱查询过程,但面对工作负载不断变化的场景下,仍具有一定的局限性. 在面对复杂的图数据查询场景下,原生图结构的图存储方式也被广泛应用,Neo4j 是目前最流行的属性图存储[24],由于其无索引邻接特性遍历邻接顶点时查询效率很高,但面对高并发场景以及源数据发生变化情况下,Neo4j 图数据库则具有一定的局限性,其他流行的原生图数据库如 OrientDB[25]支持在复杂场景下对图进行分片,JanusGraph[26]能够较好的支持大型图计算,在不同业务场景中则具有一定的可用性.

1.2　图缓存

原生图结构的图存储方式由于其灵活的数据模型以及高效处理复杂查询的特点,得以在各类应用中得到广泛应用,而由于子图同构的 np 完全问题,图查询处理在应用中可能非常耗时,基于此 Wang 等人[27]首次提出 GraphCache:用于子图/超图查询的图缓存系统,并针对该架构提出了一种新的混合图替换策略和缓存准入控制机制,作为可插拔缓存组件支持多种子图同构算法和 filter-then-verify(FTV)子图查询方法,在查询处理时间方面取得了较好的性能,基于 GraphCache 系统架构 Wang[28]提出 GraphCache + 来解决图形缓存的一致性问题. Zhang 等人[29]提出了 Cagra 缓存优化内存图框架,采用 CSR Segmenting 将顶点分解为适合上一级缓存的段,并根据段将图划分为子图,从而与缓存进行交互. Yuan 等人[30]提出了 GCache,由在线阶段和离线阶段组成,离线阶段提供了一个基于二分图聚类的缓存模型,在线阶段根据 LRU 和 MRU 策略缓存输出图形聚类. D. Tsoumakos 等人[31]则通过缓存 SPARQL 查询结果来解决大型图索引问题,将规范标签算法与动态规划器集成在一起,以生成最佳连接执行计划,检查原始三元索引和缓存查询结果的利用率.

2　PGCA-RAG 模型

本节将详细介绍所提出的模型的具体实现过程,包括 PGCA-RAG 模型整体架构和 KGCU 模型. 缓存节点采用 VoltDB 实现,知识图谱源数据采用 Neo4j 进行存储. 系统将用户的查询和查询到的相关知识进行整合作为上下文信息,

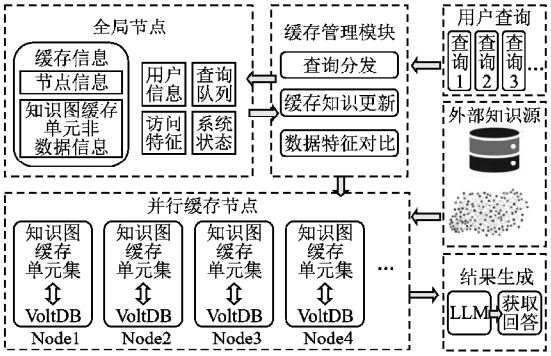

图 1　PGCA-RAG:并行图缓存架构的总体框架示意图

Fig. 1　PGCA-RAG: schematic diagram of the general framework of the parallel graph cache architecture

送入 LLMs 中进行答案生成并返回给用户,同时用户也可以根据反馈回答满意度来更新关键词权重,整体架构如图 1 所示.

2.1　知识图缓存单元数据模型

本文使用 VoltDB 内存关系型数据库作为模型实现的基础,大型网络数据往往包含着隐藏信息,如果能尽可能正确的发现图中所包含的子社群,能大大的提高数据访问的效率,常见的社区检测算法[32,33]从图本身试图尽可能准确的找出准确的聚类信息,但聚类出的数据在使用过程中往往不能真实的与访问特征相匹配,尤其在面对访问时延要求比较高的图谱访问场景中,聚类数据跟访问特征相匹配是非常重要的,因此引入了一种新的社区检测算法,基本理念是依据用户行为动态生成知识聚类社区,使得提炼出的的子图社区更加具有真实可用性,社区检测贯穿于端到端的系统中,同时与内存关系数据库结合,形成动态 KGCU 模型.

2.1.1 知识图缓存单元模型定义

在 KGCU 模型中存储对应的图结构化数据,衍生模块数据,以及调度信息等,系统在访问过程中针对不同数据进行调度,不同模块的数据描述定义如下:

定义 1. 知识主词:设图 $G=(V,E)$,$V(G)$为图中节点的集合,$E(G)$为图中有向边的集合,其中 $e=\{u,v\}$ 表示两个节点之间的有向边,同时每个节点存在属性集 $P=<p1,p2,p3,\cdots,pn\text{-}1,pn>$,在任意垂直领域知识图谱中,定义存在一类核心实体 $R(R\subseteq V)$,即为知识主词,如医疗领域知识图谱中的疾病类型实体,而如何快速准确的确定在垂直领域图谱中哪一类实体为知识主词,可以从知识主词的两大特性出发:

1)领域核心性:该类实体是垂直领域的核心研究对象,如医疗领域中的"疾病",金融领域中的"股票",可以根据领域权威文献或行业术语体系中是否将其作为分类主干来确定其是否能够作为知识主词.

2)网络中心性:该类实体在知识图谱中与其他实体,属性,关系的连接密度最高,可以根据计算实体的度中心性和介数中心性来进行量化.

对于特殊使用场景同样可以根据需求将特殊的一类实体作为知识主词,在属性集 P 中添加属性 $p0$,表示知识模型中唯一的序列号,同时在知识特征和知识轨迹中每条记录均包含 $p0$.

定义 2. 推荐词:设推荐词 $S(S\subseteq V)$ 为图中非主词节点的集合,即在知识图谱中除知识主词所代表的一类核心实体之外,其余所有类型的实体,例如医疗知识图谱中除了疾病实体之外的药品,食物,科室等等,用于匹配用户查询知识主词的邻接信息,推荐词表的数量与图中非知识主词节点类型的数量相匹配,其中属性集 P 由图中对应节点实体的属性集合组成.

定义 3. 知识编码:定义知识编码 X 为图 G 中由知识主词节点 R 出发的有向边的类型,$R(R\subseteq V)$,$S(S\subseteq V)$其中 $R\cap S=\varnothing$,存在(R,X,V)其对应知识主词与目标实体节点的关系链. 例如在医疗知识图谱中,将以疾病实体类型为源节点的所有关系类型作为知识编码,即以疾病为主语的三元组数据中存在的所有谓语类型.

定义 4. 知识特征:设知识特征 KF 为知识主词 R 及 R 对应的访问频度组成的集合,在结构化存储中 KF 即为知识主词及其对应的访问频度所组成的两列表数据,不同的知识主词集合或不同的访问频度分布都代表了不同的语义网络,通过 KF 表征该 KGCU 所包含的数据(除调度信息之外)对应的隐含语义网络特征,以此作为缓存节点更新时与具有相同两列表结构的访问特征进行相似度匹配的依据.

定义 5. 知识轨迹:定义知识轨迹 H 是由多个有序对(R,X)或(R,P)构成的集合通过合并转化形成的,在访问知识图谱数据过程中,通常以图节点为中心出发访问其属性集或邻接点,每个用户问题中包含的所有由知识主词 R 和对应知识编码 X 或属性集 P 组成的集合通过合并形成的字段即为知识轨迹,例如在医疗领域中用户的问题为"最近有些头痛和失眠的症状,可以吃些什么药物缓解?",通过解析出问题所包含的失眠和头痛两个实体以及推荐药品的关系链,该知识轨迹即为"失眠-推荐药品-头痛-推荐药品",可以根据知识轨迹来对用户的问题进行特征分析,在关系数据库中将 H 映射为字符类型的数据,当访问该记录时,根据对应的映射规则进行解析以此与新的知识轨迹进行匹配.

定义 6. 知识图缓存单元:该整体结构由 3 部分组成,如图2右侧部分所示,左上部分是将知识图谱中的实体属性以及关系映射成的知识编码,知识主词,以及推荐词的结构化数据,右上部分则为根据上述三者以及用户访问行为衍生出的三大数据模块,其中三元组信息用于匹配知识主词和推荐词之间的关系,知识轨迹存储在历史信息模块之中,用于匹配用户的查询访问,知识特征模块则用于表达该 KGCU 的语义特征. 下方则存储 KGCU 的统计信息和淘汰相关信息用于进行全局调度.

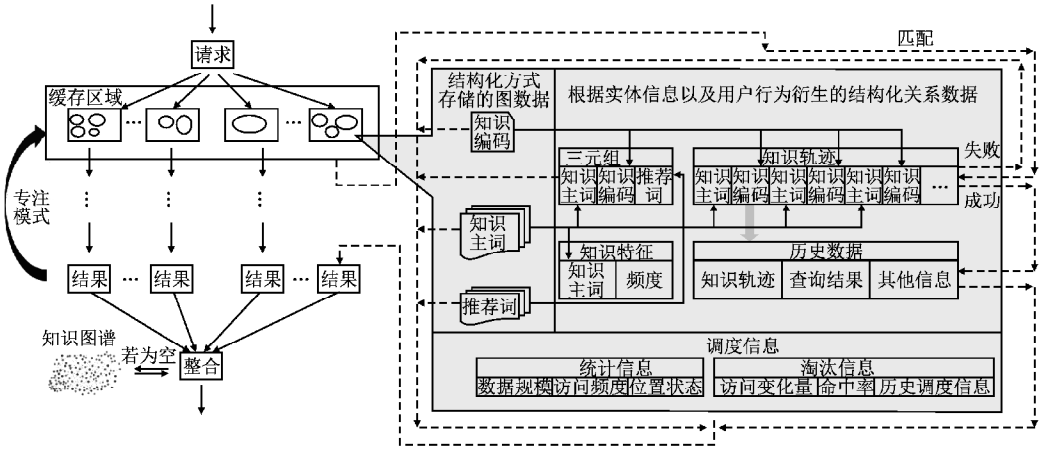

图 2 并行知识图缓存单元结构图

Fig. 2 Parallel knowledge graph cache unit structure diagram

2.1.2 知识调度方法

系统接受用户的查询作为输入,进而对请求进行解析提取其所包含的知识轨迹,解析由实体和关系组成的知识轨迹此步骤涉及信息抽取领域中的命名实体识别任务和关系抽取任务,本文对两大任务均采用了准确率高且计算资源需求相对较低的规则化方法,以降低对于实验结果带来干扰的可能性. 命名实体识别任务采用基于词典和规则方法,词典来自知

识图谱的所有实体信息,关系抽取任务采用基于规则化的模式匹配方法,预先根据知识图谱存在的属性关系定义了固定的模式来匹配用户非结构化查询中存在的关系,例如对于医疗知识图谱,固定的模式包括但不限于:症状、忌口、吃什么药、如何治疗、如何预防、治疗周期等等. 随后从缓存调度管理模块所指定查询的 KGCU 中匹配存在的知识轨迹记录,如果匹配成功则会直接将匹配到的结果进行返回,否则将在模型中的图结构化数据进行查询,若仍然不存在该请求所需的查询数据,系统将会根据该请求对应的轨迹信息转化为对应的图查询语言 Cypher 语句从知识图谱源数据中进行查询. 整体流程如图 2 右侧虚线所示. 通过端到端的请求处理产生的聚类数据来动态的更新 KGCU 的方法,相比于传统的社区检测算法能够生成更加真实的图聚类数据.

2.2　集群节点结构

每个 KGCU 模型的大小通常由热点事件中用户的访问行为所决定,为了最大化利用缓存节点的性能,系统支持设置节点最大 KGCU 容量,也就是说在每个缓存节点中可以同时存在多个 KGCU 并行化使用,这样大大提高集群机器的性能利用率,如图 1 和图 2 左侧缓存区域所示. 每个 KGCU 在首次构建时都会生成独一无二的索引. 在系统的运行过程中无论其处于缓存中或处于磁盘中,系统都可以通过该索引在全局调度节点中找到该 KGCU 的相关信息,全局调度节点负责存储所有 KGCU 的非数据相关信息,包括索引信息,状态,以及所处的位置等,便于在对其进行搜索的时候能够准确的找到其位置,其次全局调度节点还负责存储所有用户的相关信息,基于此结构来实现 KGCU 的并行化访问.

3　并行缓存调度算法

系统采用集群的方式实现如图 2 左侧所示,在缓存区域中的每个并行缓存节点对应若干个规模不定的 KGCU 模型,所实现的并行缓存调度算法主要分为并行访问调度算法和并行知识图缓存单元调度算法两大部分.

3.1　并行访问调度算法

将基于访问所产生的系统行为归纳到并行访问调度算法

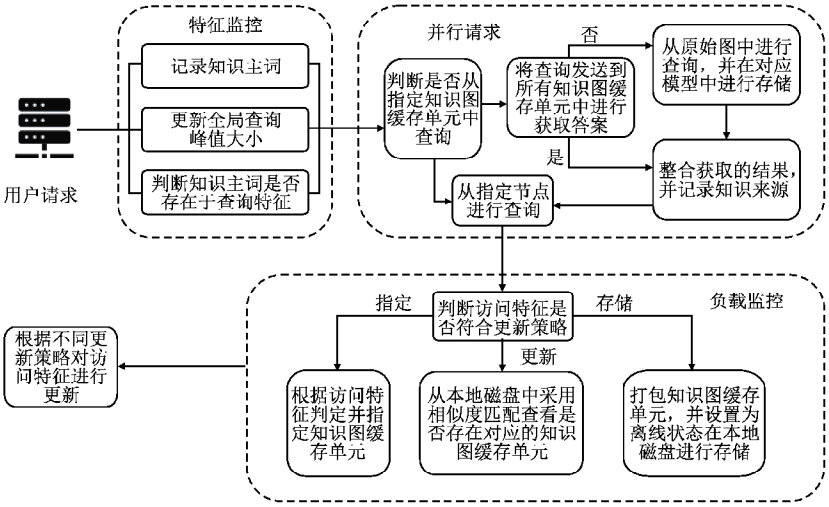

图 3　并行访问调度算法总体框架示意图

Fig. 3　Schematic diagram of the overall framework of parallel access scheduling algorithm

框架下,如图 3 所示,主要包括 3 个部分:1)特征监控;2)并行请求;3)负载监控.

3.1.1　特征监控

对缓存节点更新策略提供可靠的依据,需要准确的记录系统工作内容以及各个节点的状态,系统在全局调度节点中进行该项工作,同时外部的访问都经过该节点进行处理以及请求转发,在本步骤中需要对 3 个模块内容进行记录更新:

1)QF(Query Feature)模块:该模块采用与 KGCU 中知识特征相同的数据结构进而实时记录用户访问的特征.

2)TPV(Time Period Visits)模块:该模块记录每个时间段系统访问量,用于在处理负载监控的时候判定调度行为.

3)NQ(Nearby Query)模块:该模块记录每个邻近访问结果来源.

3.1.2　并行请求

能够最大限度的发挥集群节点的性能应用于图缓存是非常重要的,系统中每个缓存节点分为 BN(Busy Node)和 FN(Free Node)状态,当该节点中存储 KGCU 时或该节点用于构建新 KGCU 模型时,此时该节点状态为 BN,当节点未被利用时为 FN. 系统实时监控 TPV 模块信息,根据设定的 FT(Fluctuation Threshold ,波动阈值)参数来判断系统状态,同时系统在转发请求时分为两种状态:

1)当系统访问变化量小于 FT 时,此时系统处于 Global 全局模式,系统在处理用户的请求时会将请求分发到所有 BN 节点中,如果所有 BN 节点都不存在该提问对应的外部知识,则系统会将查询转化为 Cypher 从 NodeKG 进行查询,同时将对应的知识映射存储在空闲节点中.

2)当系统访问变化量大于 FT 时,此时系统处于 Single 专注模式(其中该模式对应两种不同情况将在 3.1.3 中讲解),系统在处理用户访问时会在指定节点的 KGCU 之间进行访问操作.

3.1.3　负载监控

本小节将详细介绍并行图缓存架构负载监控细节以及缓存更新策略,缓存节点的状态与用户的访问行为息息相关,通过监控 TPV 模块以及系统的访问特征将系统的更新策略分

为 4 种情况：

1）当系统访问变化量处于上升趋势并且大于 FT 时，此时系统需要根据 NQ 模块进行决策，如果该模块中所记录的访问结果来源为知识图谱源数据，则系统需要从本地磁盘中对所有 KGCU 模型中的知识特征进行相似度计算，查询是否存在符合该阶段的 KGCU 模型并将其部署到缓存中. 如果磁盘中不存在，系统将根据访问行为构建该阶段的 KGCU 模型，此时系统为 Single 专注模式. 本文采用混合评分机制对 QF 模块和 KGCU 模型中知识特征进行相似度计算，确保能够精准的判断两者之间的相似性，如公式（1）所示：

$$similarity = 0.5 \times \frac{q \cap d}{q \cup d} + 0.5 \times \frac{\sum_{i=1}^{n} \left[\left| \frac{|q'i - q'd|}{l} - 1 \right| \right]}{l} \tag{1}$$

其中 q 为 QF 模块中的知识主词集合，d 为知识特征中的知识主词集合，q′和 d′分别为 QF 模块和知识特征根据知识主词 value 降序排序之后值赋为下标索引的列表，l 为 QF 模块和知识特征长度的较小值.

2）当系统访问变化量处于上升趋势并且大于 FT，但访问特征中的 NQ 判定所记录的访问特征为缓存，则系统将对 BN 节点进行遍历，同时对知识特征和 QF 模块进行相似度计算，选取相似度最高的 KGCU 模型设置为 Single 专注模式下指定的缓存数据来源，系统将根据访问的进行对所包含的知识进行更新迭代.

3）当系统访问变化量处于下降趋势并且大于 FT，说明系统结束 1 阶段和 2 阶段，此时 Single 专注模式所对应的 KGCU 模型已经更新构建完成，将系统设置为 Global 全局模式. 当下次缓存节点对该模型进行替换时，系统则会对其所包含的数据进行打包存储在本地磁盘中.

4）无论系统访问变化量处于上升或下降趋势，当其处于 FT 范围之内时，说明此时系统处于 Global 状态下，则不对缓存节点进行更新，该阶段系统处理访问过程中不存在于缓存节点中的知识则会存储在空闲节点中.

上述 4 种情况分别对应于系统运行过程中的不同状态，按照该负载策略能够更加充分的发挥并行缓存节点的性能，高质量的处理系统可能面临的高并发访问，同时能够随着系统的运行动态检测出图数据的知识子图社群.

3.2 并行知识图缓存单元调度算法

对不同缓存节点所拥有的 KGCU 数量以及规模进行调度对于提高访问效率是至关重要的，当某个缓存节点所拥有的 KGCU 规模较大数量较多时，则用户访问所产生的时延将大部分来自于该节点，基于此结合在全局调度节点中储存的信息来对缓存区域中的 KGCU 进行调度，以便更好的均衡不同缓存节点所部署的 KGCU 模型规模差距较大或数量差距较大的问题. 系统需要选取调度的缓存节点以及需要调度的 KGCU 模型，其中选取缓存节点计算公式如公式（2）所示：

$$node = Min(\sum_{x} x_{qn}) \tag{2}$$

其中 x 为缓存节点，qn 为 KGCU 的知识主词数量

选取进行调度的 KGCU 模型计算公式如公式（3）所示：

$$knowledge\ cell = Min\left[\sqrt{\frac{\left((\sum_{x'} x'_{qn} - \frac{\sum_{x=1}^{n} \sum_{x} x_{qn}}{n} \right)^2}{n}} \right] \tag{3}$$

其中 x 为调整前的缓存节点，x′为调整后的缓存节点.

当系统处于 Global 状态下，系统会自动进行调度尝试，将所选取的 KGCU 移动到所选取的缓存节点上，以便于尽可能的均衡各个缓存节点的访问性能，提高系统的访问效率.

4 仿真实验与评估

本章在不同节点数量和不同热点事件数量的情况下进行充分实验，对所提出的并行图缓存架构和基础架构进行对比分析.

4.1 实验环境与方法

所有实验均在 8 节点 Sugon A620r-G 服务器上进行，每个节点均配有 AMD Opteron（tm） Processor 6320 8 核 CPU，16G 内存，280G SSD，系统版本为 Ubuntu 22.04，其中节点 1 为全局调度节点，除了存储用户信息以及 KGCU 非数据信息之外，还负责接收处理用户请求以及转发请求从图数据节点或缓存节点查询数据，节点 2 为知识图谱源数据节点，实验所采用的知识图谱数据源来自[https://github.com/liuhuanyong/QASystemOnMedicalKG]，包含八千余种疾病，4 万余知识实体，约 30 万实体关系具有代表性的医疗知识图谱. 节点 3～8 为缓存节点，每个节点对应一个由 VoltDB 组成的缓存，因此本实验最大缓存数量为 6，为了便于准确测试所提出的架构的性能，本文在一定时间范围内针对一部分数据进行大量访问来模拟在真实场景下热点事件的发生，所模拟的用户访问采用 Abacha 等人[34]所提出的方法并根据所用图谱中所包含的疾病来生成，系统将一个热点事件下所对应的数据及其特征构建为一个 KGCU 模型. 在实际应用场景中不同 KGCU 所包含的数据规模不同，本文采用由所包含的知识主词数量表示 KGCU 数据规模大小的方法，具体而言，KGCU 数据规模大小为 10000 数量级即该 KGCU 所包含的知识主词数量为 10000，同时采用传统朴素 Graph RAG 即将知识图谱作为数据源的朴素 RAG 作为实验测试基准. 实验所采用 FT 的大小为 0.5，在工程化实际使用过程中 FT 大小需要根据不同使用场景来设定.

4.2 实验结果与分析

4.2.1 图缓存架构有效性验证

为了测试图缓存架构相比朴素 Graph RAG 架构所带来的性能提升，本文预先采用不同数量级的访问规模构建了不同数据规模大小的 KGCU，同时针对两种架构进行了多轮测试，该实验中图缓存架构采用单节点同时 KGCU 容量为 1 来测试，以此来充分证明两种架构之间的差异性以及 KGCU 模型的有效性，实验结果如图 4 所示，对于数据规模大小不同的 KGCU 访问测试中，朴素 Graph RAG 在多轮测试中均保持了相对稳定的速度，而单节点 PGCA-RAG 在多轮测试中访问时延大幅度降低，但随着 KGCU 数据规模的提高，PGCA-RAG 访问时延存在小幅度上升，实验结果证明所提出的图缓存架构在面对热点事件时具有优势.

4.2.2 并行架构有效性验证

系统往往在不同阶段会面临不同的热点事件，同时存在同一时间段内用户对不同热点事件访问的情况，为了测试提出的并行架构在分布式多集群节点中对比单节点缓存是否具

有优势,本文在不同数量的热点事件下分别采用单节点,2 节点,4 节点,6 节点架构进行测试,该实验每个集群节点的 KGCU 容量为 1,同时已经预先将每个热点事件所产生的访

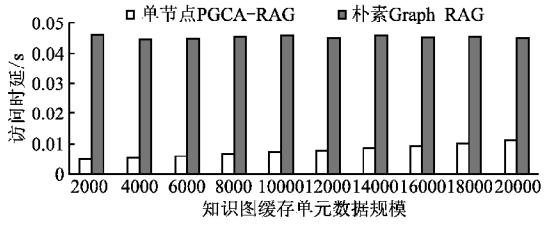

图 4　单节点 PGCA-RAG 与朴素 Graph RAG 方法对比结果

Fig. 4　Comparison results of single node PGCA-RAG with simple Graph RAG method

问构建为不同的 KGCU 模型存储在本地磁盘,每一轮访问为 10000 次,实验结果如图 5 所示,在不同集群节点数量情况下,当热点数量事件大于缓存数量时,会出现事件排队的情况,即系统所有缓存节点都对应应用于某热点事件,当新的热点事件出现时,则会出现没有缓存能够应用于该事件,该部分访问则会自动从原生图数据中查询,因此造成访问时延出现

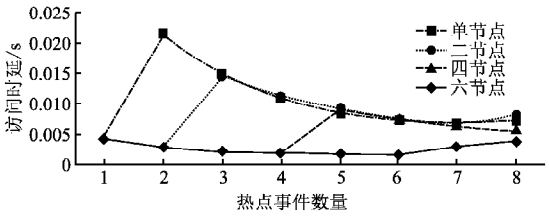

图 5　不同缓存节点数量架构下对比结果

Fig. 5　Comparison results for architectures with different numbers of distributed nodes

上升的情况. 随着分布式集群节点数量的提高,系统能够处理的热点事件的数量也随之提高,因此采用多节点架构能够较好的提升在应对不同情况的访问行为下系统的承载处理能力.

4.2.3　单节点多知识图缓存单元并行测试

每个缓存节点中支持多个 KGCU 并行存在进行访问使用,通常来说,一个缓存节点对应于一个 KGCU 往往访问效率是最高的,但可能会造成大量的机器资源浪费,因此本文测试了 PGCA-RAG 架构在单缓存节点情况下不同数量的 KGCU 所带来的访问时延差异以及机器资源利用效率,以此来探究对于实验所用的集群资源在不同数量的 KGCU 情况下的差异. 该实验采用数据规模为 10000 数量级的 KGCU 进行测试,经过完整的实验观察发现,缓存节点在不同数量的 KGCU 情况下 CPU 利用率均维持在 12%,因此在计算机器资源利用效率(Machine Resource Utilization Efficiency, MRUE)指标时不考虑 CPU 利用率,具体计算方法如公式(4)所示:

$$MRUE = \frac{r_i - r_{min}}{r_{max} - r_{min}} \tag{4}$$

其中 r 为每秒查询率(Queries-Per-Second, QPS)与内存利用率比值的集合.

实验结果如图 6 所示,在不同数量的 KGCU 情况下,系统的访问时延随着节点中 KGCU 模型数量的增加而上升,而机器资源利用效率随着 KGCU 模型数量的增加大幅提高,因

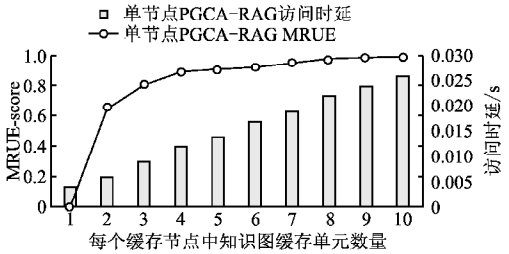

图 6　同数量知识图缓存单元对比结果

Fig. 6　Comparison results of knowledge graph cache unit with different quantities

此建议系统缓存节点的 KGCU 容量为设置为 4 ~ 6,能够较好的兼顾访问时延和机器资源利用效率.

4.2.4　并发能力测试

在工程化使用场景中,系统往往会面临高并发的情况,为了测试 PGCA-RAG 在面对并发访问情况下的性能,本文对比了朴素 Graph RAG 和单节点 PGCA-RAG 并发压力测试结果,以此来验证所提出的架构与朴素 Graph RAG 的差距. 该实验采用数据规模为 10000 数量级的 KGCU,节点 KGCU 容量为 1,同时采用 6 台服务器来模拟从 100 ~ 2000 的并发规模. 通过模拟在真实场景下不同用户并发访问来对系统进行测试,实验结果如图 7 所示,在实验的过程中随着并发数量的上升,朴素GraphRAG所采用的策略在并发数量大于400时

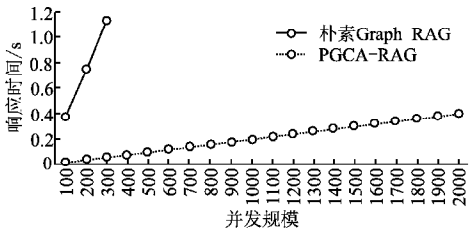

图 7　不同架构并发压力测试

Fig. 7　Concurrent stress testing for different architecture

会出现访问假死系统崩溃的情况,此时模拟访问的客户端持续处于等待的状态. 在并发数量能够承受的范围内,朴素 Graph RAG 采用的策略响应时间大幅上升,而所采用的 PGCA-RAG 系统在并发规模 2000 内响应时间逐渐上升,平均响应时间相比于朴素 Graph RAG 保持了优势,因此实验结果表明所提出的系统架构在高并发场景下保持了一定程度的优势,在真正工程化场景中具有较高的可用性.

5　总　结

为了提高在 RAG 系统中用户的访问体验,本文通过分析 RAG 系统结构中不同组件的特点以及现有检索增强方法,提出了一种面向大语言模型检索增强的并行图缓存架构. 通过对图数据的结构进行分析以及结合用户侧访问行为,提出了一种适用于 RAG 系统的动态缓存数据模型 KGCU,同时适配

于所提出的并行图缓存架构.研究结果表明,在面对不同热点事件的访问以及不同压力程度的并发测试环境中,所提出的系统的访问效率上优于目前传统的朴素 Graph RAG,同时能够灵活的调整缓存架构来提高系统的承载能力,然而并行架构系统所面临的整合应用能力依旧欠佳,未来的工作将探索类似于简化 RAG 缓存系统的构建过程,为检索增强实现更为高效通用的缓存解决方案.

References:

[1] Touvron H, Lavril T, Izacard G, et al. Llama: open and efficient foundation language models[J]. arXiv preprint arXiv:2302.13971, 2023.

[2] Liu A, Feng B, Xue B, et al. Deepseek-v3 technical report[J]. arXiv preprint arXiv:2412.19437, 2024.

[3] Cho J, Lei J, Tan H, et al. Unifying vision-and-language tasks via text generation[C]//International Conference on Machine Learning, 2021: 1931-1942.

[4] Zhang T, Ladhak F, Durmus E, et al. Benchmarking large language models for news summarization[J]. Transactions of the Association for Computational Linguistics, 2024, 12: 39-57, doi: 10.1162/tacl_a_00632.

[5] Huang L, Yu W, Ma W, et al. A survey on hallucination in large language models: principles, taxonomy, challenges, and open questions[J]. ACM Transactions on Information Systems, 2024, 43(2): 1-55.

[6] Ji Z, Lee N, Frieske R, et al. Survey of hallucination in natural language generation[J]. ACM Computing Surveys, 2023, 55(12): 1-38.

[7] Achiam J, Adler S, Agarwal S, et al. Gpt-4 technical report[J]. arXiv preprint arXiv:2303.08774, 2023.

[8] Ouyang L, Wu J, Jiang X, et al. Training language models to follow instructions with human feedback[J]. Advances in Neural Information Processing Systems, 2022, 35: 27730-27744, doi: 10.48550/arxiv.2203.02115.

[9] Nakano R, Hilton J, Balaji S, et al. Webgpt: browser-assisted question-answering with human feedback[J]. arXiv preprint arXiv: 2112.09332, 2021.

[10] Li J, Cheng X, Zhao W X, et al. Halueval: s large-scale hallucination evaluation benchmark for large language models[J]. arXiv preprint arXiv:2305.11747, 2023.

[11] Lewis P, Perez E, Piktus A, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks[J]. Advances in Neural Information Processing Systems, 2020, 33: 9459-9474, doi: 10.48550/arxiv.2005.11401.

[12] Petroni F, Rocktäschel T, Lewis P, et al. Language models as knowledge bases? [J]. arXiv preprint arXiv:1909.01066, 2019.

[13] Robertson S, Zaragoza H. The probabilistic relevance framework: BM25 and beyond[J]. Foundations and Trends in Information Retrieval, 2009, 3(4): 333-389.

[14] Wang L, Yang N, Huang X, et al. Text embeddings by weakly-supervised contrastive pre-training[J]. arXiv preprint arXiv: 2212.03533, 2022.

[15] He X, Tian Y, Sun Y, et al. G-retriever: retrieval-augmented generation for textual graph understanding and question answering[J]. arXiv preprint arXiv:2402.07630, 2024.

[16] Qi Z, Wang H, Zhang H. A dual-store structure for knowledge graphs[J]. IEEE Transactions on Knowledge and Data Engineering, 2020, 35(2): 1104-1118, doi: 10.1109/TKDE.2021.3093200.

[17] Harris S, Gibbins N. 3store: efficient bulk RDF storage[J/OL]. Practical and Scalable Semantic Systems, 2004: 1-15, https://api.semanticscholar.org/CorpusID:11115232.

[18] Pan Z, Heflin J. DLDB: extending relational databases to support semantic web queries[C]//1st International Workshop on Practical and Scalable Semantic Systems, 2003: 1-14.

[19] Wilkinson K, Wilkinson K. Jena property table implementation[J]. Computer Science, 2006, 140: 35-46.

[20] Abadi D J, Marcus A, Madden S R, et al. SW-Store: a vertically partitioned DBMS for semantic web data management[J]. The VLDB Journal, 2009, 18: 385-406, doi: 10.1007/s00778-008-0125-y.

[21] Neumann T, Weikum G. RDF-3X: a RISC-style engine for RDF[J]. Proceedings of the VLDB Endowment, 2008, 1(1): 647-659.

[22] Weiss C, Karras P, Bernstein A. Hexastore: sextuple indexing for semantic web data management[J]. Proceedings of the VLDB Endowment, 2008, 1(1): 1008-1019.

[23] Sun W, Fokoue A, Srinivas K, et al. Sqlgraph: an efficient relational-based property graph store[C]//Proceedings of the ACM SIGMOD International Conference on Management of Data, 2015: 1887-1901.

[24] Neo4j[EB/OL]. https://neo4j.com/download/, 2024.

[25] Ritter D, Dell'Aquila L, Lomakin A, et al. OrientDB: a NoSQL, open source MMDMS[C]//British International Conference on Databases, 2021: 10-19.

[26] JanusGraph[EB/OL]. https://janusgraph.org/, 2022.

[27] Wang J, Ntarmos N, Triantafillou P. Graphcache: a caching system for graph queries[C]//International Conference on Extending Database Technology, 2017: 13-24.

[28] Wang J, Ntarmos N. Triantafillou P. Ensuring consistency in graph cache for graph-pattern queries[C]//6th International Workshop on Querying Graph Structured Data, 2017: 1-8.

[29] Zhang Y, Kiriansky V, Mendis C, et al. Making caches work for graph analytics[C]//International Conference on Big Data, 2017: 293-302.

[30] Yuan Y, Lian X, Chen L, et al. Gcache: neighborhood-guided graph caching in a distributed environment[J]. IEEE Transactions on Parallel and Distributed Systems, 2019, 30(11): 2463-2477.

[31] Papailiou N, Tsoumakos D, Karras P, et al. Graph-aware, workload-adaptive SPARQL query caching[C]//Proceedings of the ACM SIGMOD International Conference on Management of Data, 2015: 1777-1792.

[32] Blondel V D, Guillaume J L, Lambiotte R, et al. Fast unfolding of communities in large networks[J]. Journal of Statistical Mechanics: Theory and Experiment, 2008, (10): P10008, doi: 10.1088/1742-5468/2008/10/P10008.

[33] Traag V A, Waltman L, Van Eck N J. From Louvain to Leiden: guaranteeing well-connected communities[J]. Scientific Reports, 2019, 9(1): 1-12.

[34] Abacha A B, Zweigenbaum P. MEANS: a medical question-answering system combining NLP techniques and semantic web technologies[J]. Information Processing & Management, 2015, 51(5): 570-594.