目录

[动态图上高效缓存的并发处理点对点查询策略 2](#_Toc146465686)

[摘要 2](#_Toc146465687)

[前言 4](#_Toc146465688)

[背景和动机 6](#_Toc146465689)

[系统概述 6](#_Toc146465690)

[基于xxxxx的优化策略 6](#_Toc146465691)

[实验评估 6](#_Toc146465692)

[相关工作 6](#_Toc146465693)

[结论 6](#_Toc146465694)

[参考文献 7](#_Toc146465695)

动态图上以数据为中心的并发处理点对点查询系统

摘要

随着图处理技术在地图导航、网络分析等领域的大范围应用，大量点对点查询作业在同一个底层图上并发运行，对现有的图查询系统提出了严重挑战。针对并发点对点查询有两个优化思路，1，加快对单个查询的响应速度；2，通过数据共享优化并行查询的数据访问效率；对于前者，我们提出了以数据为中心的并发处理点对点查询系统：它包含了一个高速地核心子图查询机制，通过维护一个由高度顶点组成的核心子图，在查询到来时快速确定上界近似值，从而加速单次查询；对于后者，我们提出了一个以数据为中心的处理基底（Substrate），它将图数据结构划分为LLC级别的分块，并通过优先级调度策略和细粒度同步策略实现多任务之间数据共享，提高并发查询的吞吐量。据我们所知，GraphCPP是第一个针对并发点对点查询场景进行优化的工作，我们将其与最先进的点对点查询系统进行对比，包括SGraph[x]、Tripoline[x]、Pnp[x]，实验表明，GraphCPP将并发点对点查询的效率提升了xxxx倍。

当前问题：摘要只写了数据共享这一个创新点，显得单薄。

GraphCPP: A Data-Centric System for Concurrent Point-to-Point Queries in Dynamic Graphs

Abstract

前言

图上的点对点查询任务指利用图这一通用数据结构，发掘两个特定对象之间的某种联系。和传统的图查询方法不同，图上的点对点查询专门针对两个特定节点间的关联或路径进行分析，而无需关心整个图或其大规模子集的复杂查询。这种有针对性的查询策略赋予了点对点查询巨大的优化潜力。利用专门设计的算法，如Point-to-Point Shortest Path (PPSP)、Point-to-Point Widest Path (PPWP) 以及 Point-to-Point Narrowest Path (PPNP)，可以在无需查询与处理不相关的其他节点或边的情况下，精确地确定两节点之间的特定路径属性。由于点对点查询在图分析中的这种高效性，它在多个领域中都已得到广泛的实践应用。如：在物流运输时，找到两个地点之间的最短路径（举例子XXXXXXXXX）；在社交网络分析时，通过查找两个用户之间的关系链，为用户推荐可能的朋友（举例子XXXXXXXXX）；在金融风险分析时，分析风险是如何从一个实体传播到另一个实体（举例子XXXXXXXXX）；这些热门应用提出了在同一个底层图上，执行大规模并发点对点查询的需求。

通常来说，要提高图上并发点对点查询的吞吐量，可以从两方面入手：1，加快单次查询的速度；2，采用高效地调度策略优化并行查询的效率；当前已经有许多针对点对点查询的解决方案都聚焦于加速单次查询的效率，如：PnP使用基于下界的剪枝方法来减少查询过程中的冗余访问。Tripoline通过维护中心节点到其它顶点的日常索引，实现无需先验知识的快速查询。Sgraph利用三角不等式原理，提出了基于“上界+下界”的剪枝方法，进一步减少点对点查询过程中的冗余访问。然而我们发现上述工作都，诶呦考虑高并发的点对点查询场景。我们在XXX节证明了在企业应用中，经常要面临高并发的查询需求，而现有系统由于冗余的数据访问开销，常常造成严重的性能瓶颈。为此，本文提出了GraphCPP，一种以数据为中心的并发处理点对点查询系统，它对单次查询和并发查询都做了专门优化。

INTRODUCTION

（**ToDo，需要写变量，需要一个图来说明核心子图机制**）单次点对点查询优化：GraphCPP提出了一个高速地核心子图查询机制。**建立索引**：它首先遍历所有分区，统计顶点度数，选择hub顶点。Hub顶点的选择需要满足以下两个标准中的至少一条：1，该顶点是完整图中度数前k的顶点；2，该顶点是所在分区中度数前q的顶点（k和q的大小需要根据图的规模和内存容量调整）；确定好hub顶点后，GraphCPP以每个hub顶点为起点，在全图上执行SSSP，记录hub顶点到其它所有顶点的距离值（对于有向图，还要统计其它所有顶点到hub顶点的距离值），我们称之为hub顶点的索引值。**执行查找**：hub顶点的索引值可以帮助我们迅速找到对应查询的安全近似值。具体来说，当一个查询到来，所有的hub顶点，加上源点和目的节点构成一个核心子图。由于前面建立索引时已经完成了统计，子图上所有边的权重是已知的。对核心子图上的源点和目的顶点进行点对点查询，得到一个距离值w。这个值不一定是最短路径值，但是为我们的遍历提供了参考，我们可以使用其来进行剪枝。**剪枝查询**：同SGraph的操作一样，GraphCPP把得到的值w作为剪枝的“上界”，所有距离值大于该上界的路径都被剪枝。同时GraphCPP利用索引值、w、三角不等式，推导出路径的下界。所有考虑下界后路径值大于w的路径也会被剪枝。

**并发查询优化**：GraphCPP提出了一个以数据为中心的处理基底（Substrate），具体包含以下几部分：1，基于优先级的图分区调度策略：在GraphCPP框架内，我们对每个计算节点的图结构数据进行了更为细粒度的划分，使其适配LLC的大小。进一步提出了一种优先级计算方法。此方法旨在紧密地将查询任务与其相关的图分区联系起来。分区所关联的任务数量增加会导致其优先级提升，使其更可能被优先缓存，从而加速计算并提升整体效率；2，基于让步的异步任务执行策略：传统的策略倾向于确保单一任务的快速完成，这种方法可能导致部分任务过早完成，而其他任务长时间等待，制约了大规模并发查询的吞吐量。

GraphCPP提出了一种基于让步的异步策略，该策略在每次迭代中优选与已缓存图分区关联的任务。在执行过程中，系统更倾向于在缓存的图分区上推进任务的进度，而非一次性完成。这种方法虽可能导致个别任务速度稍减，但显著提高了整体执行效率与吞吐量；3，细粒度的数据共享策略：传统策略中任务的数据访问彼此独立，即使它们处理的数据完全相同也无法共享。GraphCPP结合其LLC级图分区与异步机制，实现了细粒度的图分区数据共享。当具有高优先级的图分区被加载到缓存，关联任务随即被唤醒并进入异步调度。这种细粒度的数据共享方式显著降低了数据访问的冗余，从而提升了系统的性能。

综上，本文主要做出了如下贡献：

1. 揭示了现有图查询系统处理并发点对点查询任务时，冗余数据访问带来的性能瓶颈。并指出可以利用并发查询任务之间的数据访问相似性优化并发任务吞吐量。
2. 实现了GraphCPP，一个动态图上以数据为中心的并发处理点对点查询系统，它利用核心子图机制优化单次查询速度。然后利用并发任务之间的数据访问相似性，加速并发点对点查询系统的吞吐量。、
3. 我们将GraphCPP于当前最先进的点对点查询系统XXXXXX进行对比，结果表明XXXXXXXXX

背景和动机



所需图像（还没画，占位）：  
1，统计各个场景的实际并发数，证明并发查询的需求。

2，统计并行查询执行时间，说明并行执行效率很差。

3，统计大量作业访问数据的重叠性，证明“数据冗余访问”。

4，统计重叠数据占总数据的比例，证明“数据冗余访问”

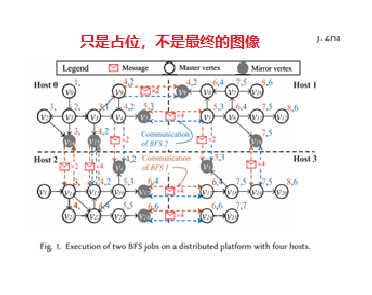
5，统计并行调度缓存错失率，说明并行调度的方案低效的原因。

现有的大多数点对点查询系统是为了优化单次查询速度而设计的，但是如图1所示，我们的统计表明，很多实际应用场景需要应对大规模的并发查询，这类场景对单次查询的速度很宽容，更加重视系统整体的吞吐量。但是如图2所示，现有系统在处理大规模并发查询时吞吐量很差。这种坏结果出现的原因是并发任务之间存在大量的冗余访问，为了定性地证明这一点，我们在XXXXX（机器配置），选取了XXXXX（现有最佳方案），在XXXXX（图数据集上），进行并行点对点查询的性能评测。

BACKGROUND AND MOTIVATION

并发任务的冗余数据访问

我用下图的例子，来说明在同一个底层图上执行并发的点对点查询，存在的冗余内存访问。如下图(参照GraphTune)所示，XXXXX结合图像描述。



对图像的修改：抛弃分区，高度顶点和普通顶点要大小和颜色区分。不同查询的路径颜色要区分。图像下面给出每个查询遍历的路径点。

如图x所示，数据重叠访问在并发任务中大量存在，对这部分数据的重复访问属于冗余访问。且如图x所示，每轮查询中冗余的数据访问占到总访问的XXXX。由于少量的高度顶点成为热门的查询路径候选点，它们被不同的查询反复加载。然而，不同任务加载的时间不同，即使在同一时间加载相同数据，在现有系统体系下也不支持这部分数据的共享。如图x所示，这部分数据在LLC中频繁换入换出，导致很高的缓存不命中率，从而导致很差的系统吞吐量。Redundant Data Access Overhead

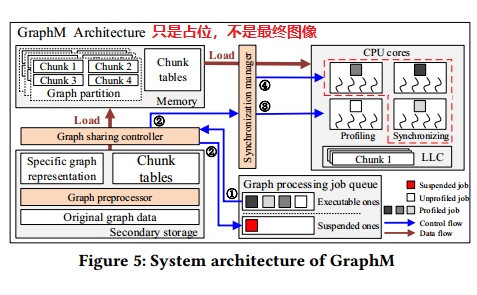
我们的启发

通过上述的实验，我们观察到了以下几点结论：

**观察1**：图上的高度顶点更可能被不同的任务重复遍历。不同的查询路径可以看做一条条线，高度顶点就是这些线段的交点，会频繁出现在不同的任务中。假如可以识别图上的高度顶点hub，并建立起hub之间的核心子图，就可以为每次查询快速确定一个近似的距离值。不一定是最优的结果，但是基于这个值我们可以对查询过程执行有效地剪枝，从而大大加快单次查询过程。

**观察2**：不同任务之间的数据访问存在相似性，它们的遍历路径有很大部分是重叠的。这点和观察1是契合的。由于不同任务访问重叠数据的时间不同，且现有的点对点查询系统并不支持任务之间细粒度的数据共享，这带来了冗余的数据访问开销。这启发我们开发高效地细粒度数据共享机制，通过支持不同任务在不同时间对相同数据进行访问共享，来减少数据访问开销，提高并发查询的吞吐量。

**Our Motivation**

系统概述

高效地核心子图查询机制

核心子图由hub顶点构成

Hub节点可以动态更新

Hub顶点除了考虑高度顶点，还考虑了每个分区

核心子图之间可以多跳运转

新查询的距离值，如果低于两个分区的核心子图值，则把这个距离值也添加

写一个公式，除了考虑距离值，还要考虑度数。

新查询的距离值只是特定分区的最短距离值

以数据为中心的调度机制

OVERVIEW OF GraphCPP

xxxxx STRATEGY xxxxx

xxxxx STRATEGY xxxxx

实验评估

我们的实验和sgraph一样是基于动态图的，sgraph采用了一种快照机制，图更新在未关闭快照上执行，图查询在已关闭快照执行。每隔一段时间将未关闭快照转为已关闭快照，并替换原有快照。

实验设置

预处理开销

整体性能对比

调度策略性能

是否开启索引子图对结果影响

可扩展性

EXPERIMENTAL EVALUATION

相关工作

**点对点查询**。现有工作对点对点查询做出了许多研究，如𝐻𝑢𝑏2 [x]提出了一种以hub为中心的专用加速器，它认为具有大量连接的顶点，即hub，扩大了搜索空间，使最短路径计算变得异常困难。它提出了Hub-Network概念，以限制hub节点的搜索范围。并使用Hub2-Labeling方法来对hub搜索过程进行在线剪枝。但是由于𝐻𝑢𝑏2定位在专用加速器，它的通用性较差。PnP观察点对点查询的遍历过程，提出了基于上界的剪枝策略，减少了不必要的顶点遍历，为点对点查询的研究提供了新的思路。Tripoline通过在日常维护一些“常设顶点”，以常设顶点为“中介”，推导两点之间近似的”上界“，这样实现了”无先验知识“的上界查询。SGraph在前两者的基础上进一步发展，利用图上的三角不等式原理提出了基于上界和下界的剪枝策略，实现了亚秒级的图上点对点查询。但是这些系统都专注于优化单次点对点查询的速度，忽略了大规模并发查询的严重负载。

**并发图计算**。许多图计算系统都对并发计算进行了研究，GraphM指出并发图计算任务之间存在的“数据访问相似性”，并提出了一种以数据为中心的调度策略，实现多任务之间的数据共享，提高了并发图计算的吞吐量。但是GraphM是单机核外图计算系统，采用BSP计算模型，并且只适用于静态图。在此基础上，CGraph[x]进一步将应用场景扩展到分布式系统上的动态图计算，并针对分布式场景优化了通信机制和负载均衡策略，但是他和GraphM一样都是核外系统，即使可以通过调度策略将磁盘访问的开销分摊到不同子图，依然不适合并发查询的高负载场景。ForkGraph实现了在内存中进行高效地并发图处理，并且采用了基于让步的调度策略，每轮迭代仅处理部分数据，加速了整体执行速度。但是该他是一个单机内存系统，并且没有为点对点查询进行优化，不适合在海量数据上执行并发点对点查询任务。

结论

致谢

RELATED WORK

CONCLUSION

ACKNOWLEDGMENTS

参考文献

1. Chen H, Zhang M, Yang K, et al. Achieving Sub-second Pairwise Query over Evolving Graphs[C]//Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2. 2023: 1-15.
2. Jiang X, Xu C, Yin X, et al. Tripoline: generalized incremental graph processing via graph triangle inequality[C]//Proceedings of the Sixteenth European Conference on Computer Systems. 2021: 17-32.
3. Xu C, Vora K, Gupta R. Pnp: Pruning and prediction for point-to-point iterative graph analytics[C]//Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems. 2019: 587-600.

废弃材料

废弃摘要内容：

在面对高并发的点对点查询需求时，由于冗余的数据访问，处理效率很低。我们观察到并发查询任务之间存在着数据访问相似性，这启发我们提出了一种以数据为中心的并发点对点查询方法。具体地，我们将图查询过程中的数据分为“图结构数据”和“任务特定数据”，前者记录了图的拓扑信息，后者记录了查询任务所要访问的图结构数据分块，不同查询独立访问任务所需的数据分块，这些分块可能重叠，但在传统的查询方案中。因此，我们采用了一种数据驱动的调度方法：在执行并发点对点查询任务时，内存/LLC中只保留一份图结构数据。多任务之间以细粒度的图数据分块为单位共享数据。一次访问，多个任务处理，以此分摊数据访问的开销，提高并发图查询的吞吐量。为了展示GraphCPP的效率，