北京航空航天大学 BEIHANG UNIVERSITY
北航计算机学院 School of Computer Science and Engineering,Buaa

单机大规模图数据查询系统优化关键技术研究

博士学位论文答辩

答辩人：朱筱可
导师：樊文飞
张日崇
专业：软件工程
培养院系：北京航空航天大学计算机学院

2025年10月14日

汇报提纲

1. 研究背景目标

2. 主要研究内容

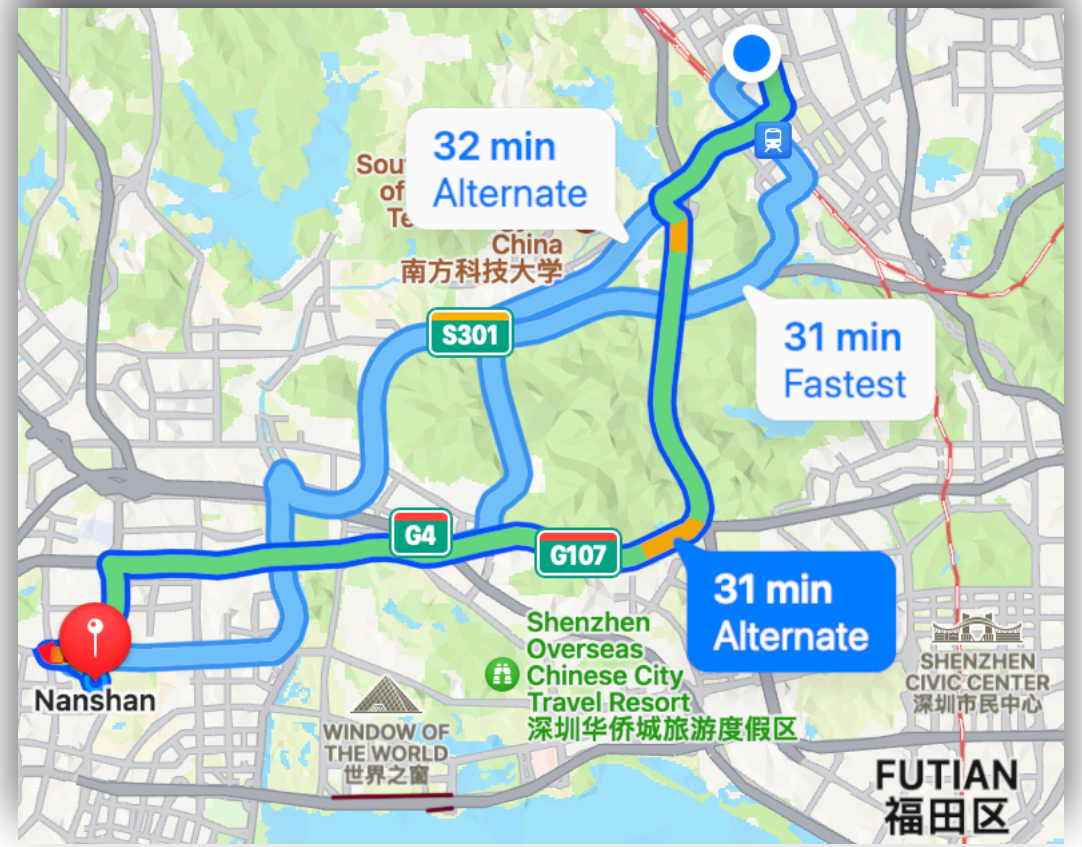
3. 论文工作总结

4. 未来工作展望

研究背景目标

研究对象：图

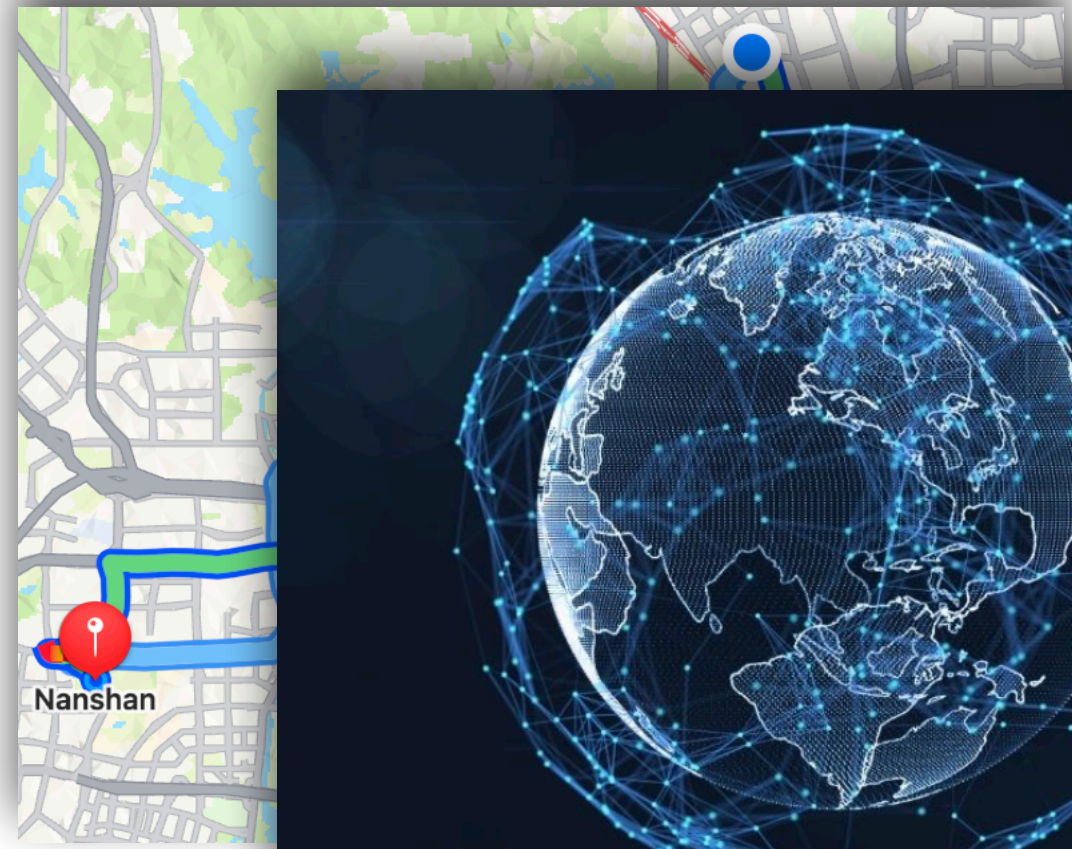
研究对象：图



路网图

我们每天都在使用图，图无处不在

研究对象：图

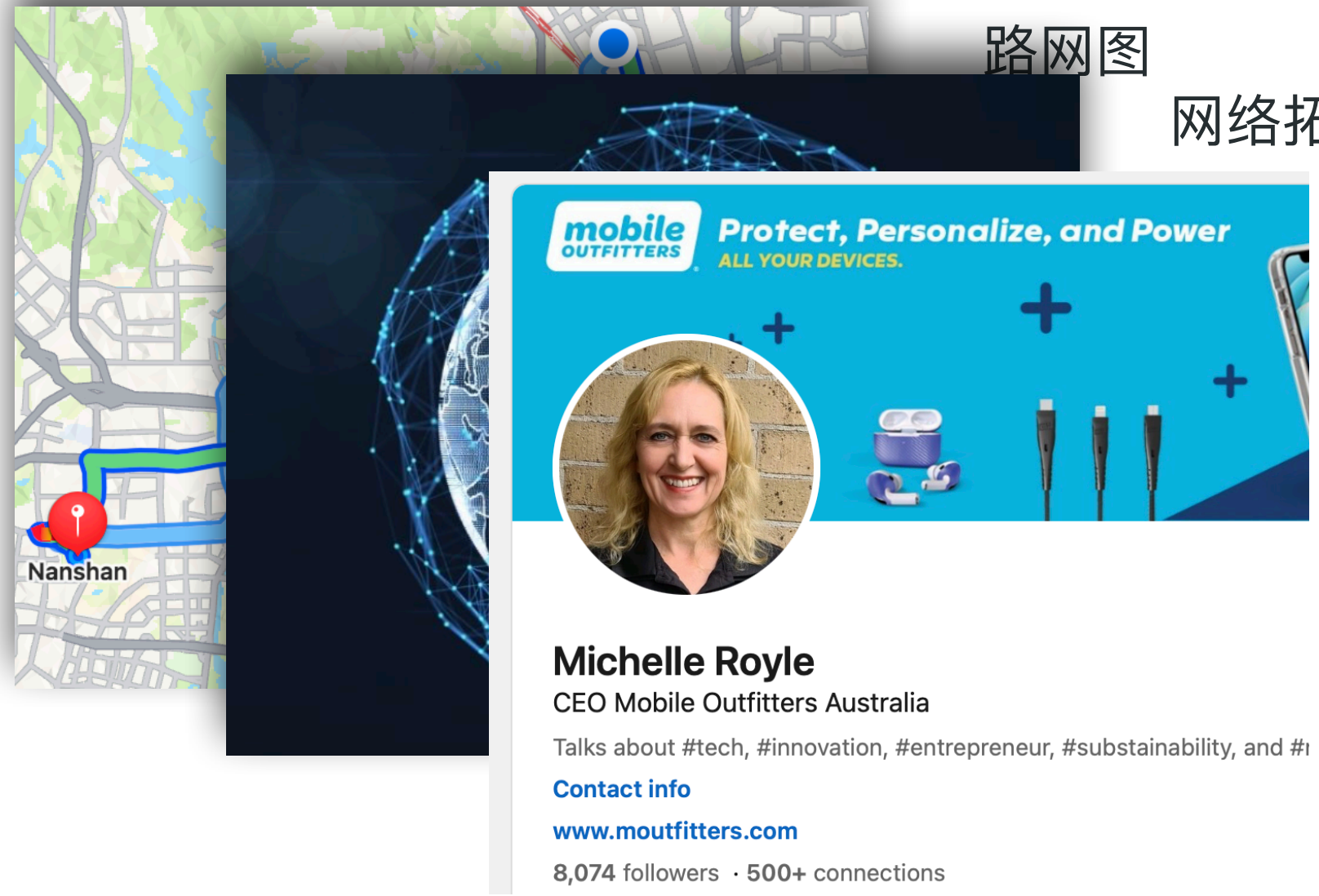


路网图

网络拓扑图

我们每天都在使用图，图无处不在

研究对象：图



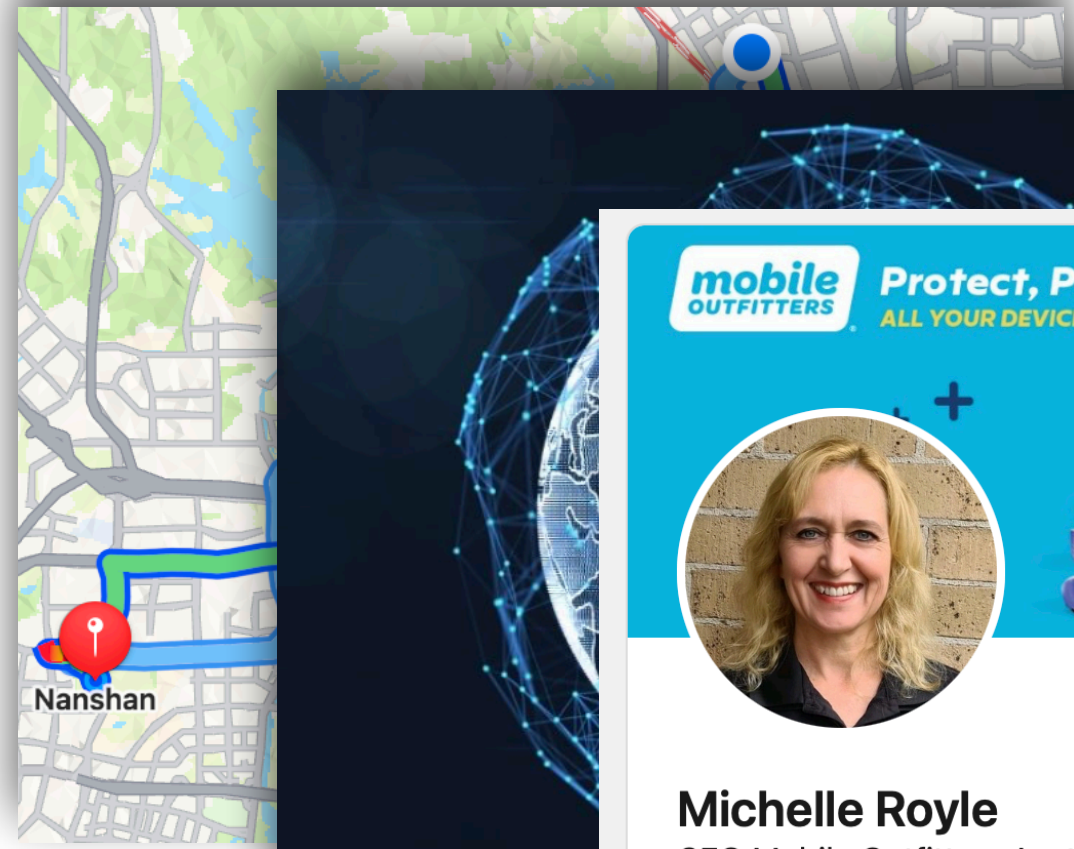
路网图

网络拓扑图

社交网络图

我们每天都在使用图，图无处不在

研究对象：图



路网图

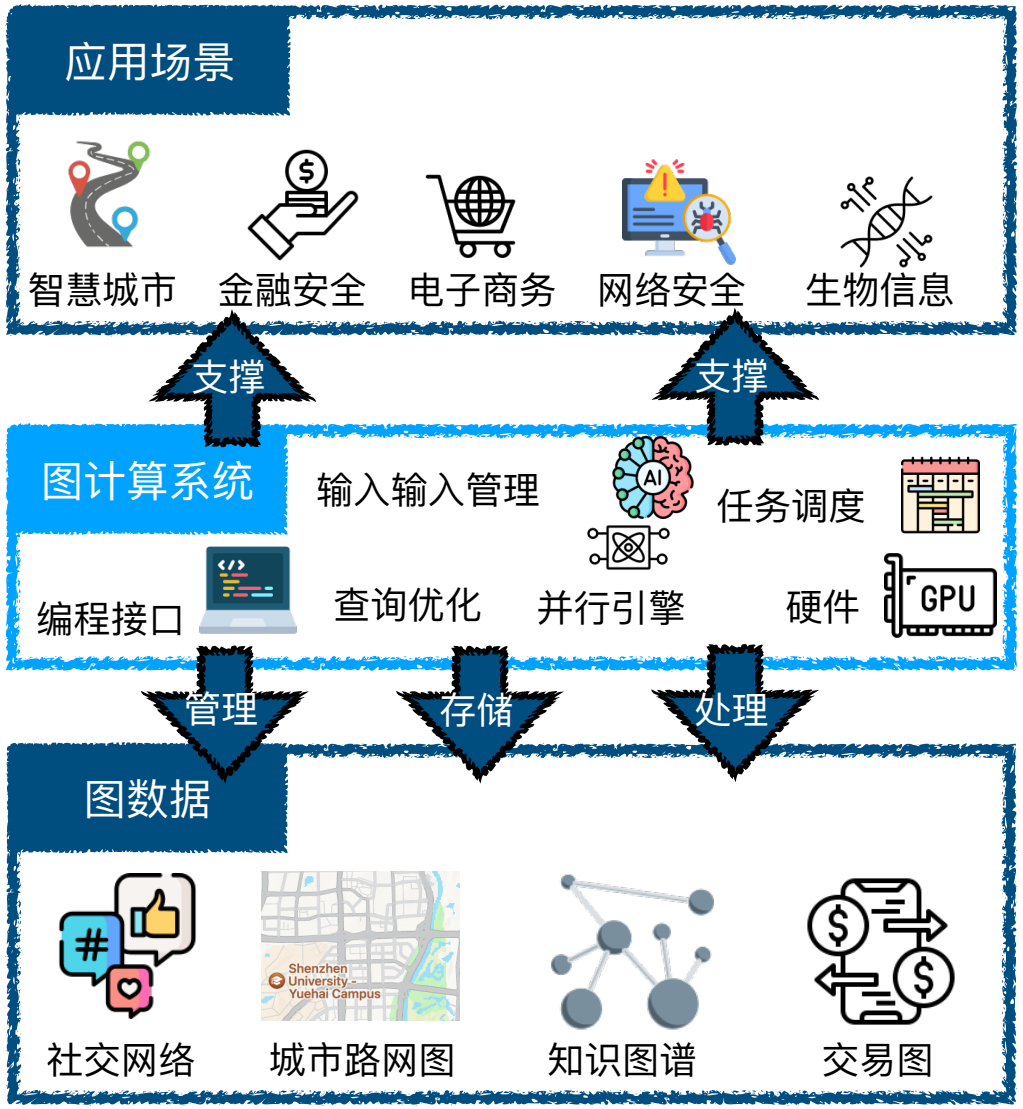
网络拓扑图

社交网络图

图谱

我们每天都在使用图，图无处不在

研究背景:图计算系统研究的重要性



图计算系统研究的重要性

Trend 8 Graph Relates Everything: Gartner predicts that by 2025, **graph technologies will be used in 80% of data and analytics innovations**, up from 10% in 2021, facilitating rapid decision making across the organization.

Gartner

《Gartner Identifies Top 10 Data and Analytics Technology Trends》

专栏一 关键基础软件补短板：突破全内存高速数据引擎、高可靠数据存储引擎、分布式数据处理与任务调度架构、**大规模并行图数据处理**等技术。

工信部《“十四五”软件和信息技术服务业发展规划》

专栏二 关键共性技术：研究跨媒体统一表征、关联理解与知识挖掘、**知识图谱构建与学习**、知识演化与推理、智能描述与生成等技术，开发跨媒体**分析推理引擎**与验证系统。

国务院《新一代人工智能发展规划》

图查询系统是支撑下游图应用的重要基础设施

研究背景：图数据查询系统的历史发展

1736年，欧拉通过解决“哥尼斯堡七桥问题”，
开创了图论这一全新数学分支

2010年，Google 提出基于点中心模型的分布式图模型Pregel
大规模图计算兴起

2016年，UC Davis提出
第一个GPU图计算框架

2020年，Alibaba 提出全展示分布式图系统 GraphScope集成OLAP、OLTP、GNN，实现多种图任务的统一架构

1960年，第一个图数据管理系统IMS诞生于IBM

2007年， 以Neo4j为代表的商用图数据库兴起

2012年，CMU推出GraphChi
开启单机外存图系统先河，
PC也能处理大规模图数据

2018年，爱丁堡大学，北航首次推出基于图中心模型基的分布式图系统GRAPE

图计算系统在计算机领域的重要性日益凸显

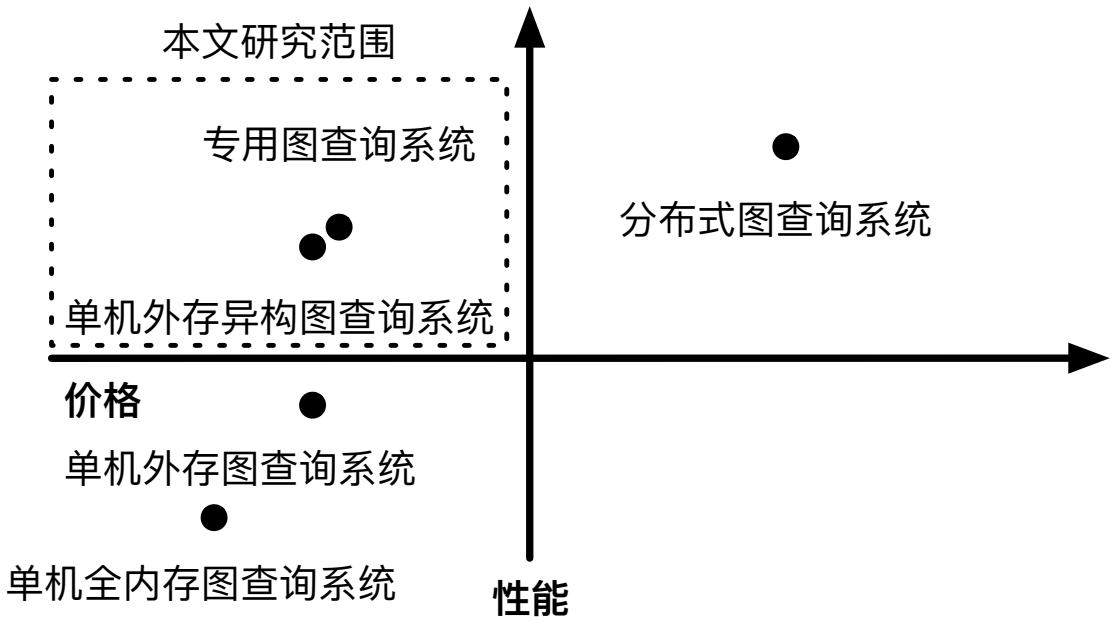
研究现状

技术类型	代表方法	核心思想	优点	存在的问题
全内存图查询系统	Ligra[PPoPP'13], Galios[SOSP'13]等	将整个图加载到内存，加速随机访问	查询时延低，适合实时分析	无法处理超出内存容量的图
分布式图查询系统	GraphScope[VLDB'21],Pregel[SIGMOD'10],GRAPE[SIGMOD'18]等	将大图分片存储 并行查询，使用集群计算提升查询能力	优越的并行性和扩展性，计算效率高	计算负载不均衡，通信开销大，运维成本高
外存图查询系统	GraphChi[OSDI'12], GridGraph[ATC'15]Mosaic[EuroSys'17]等	使用磁盘 & SSD 存储图数据，通过索引 & 预取优化查询	可处理大规模图，适用于低成本场景	I/O瓶颈，查询时延高
异构图查询系统	Gunrock[PPoPP'16],EGSM[SIGMOD'23],G2Miner[OSDI'22]等	利用 GPU / FPGA / NVM 加速计算 & 存储	并行能力&强适合高吞吐查询 & 复杂分析	基于CPU的优化逻辑不再适用，需要专门优化

本研究重点

现实需求

- ✓ **中小企业**对大规模图计算需求旺盛，但**预算有限**。
- ✓ 国企和事业单位处于**安全隐私考虑**在采用云解决方案时则更为审慎
- ✓ 对单机图系统的研究有助于分布式系统的研究，单机的研究能**反哺分布式架构**的研究

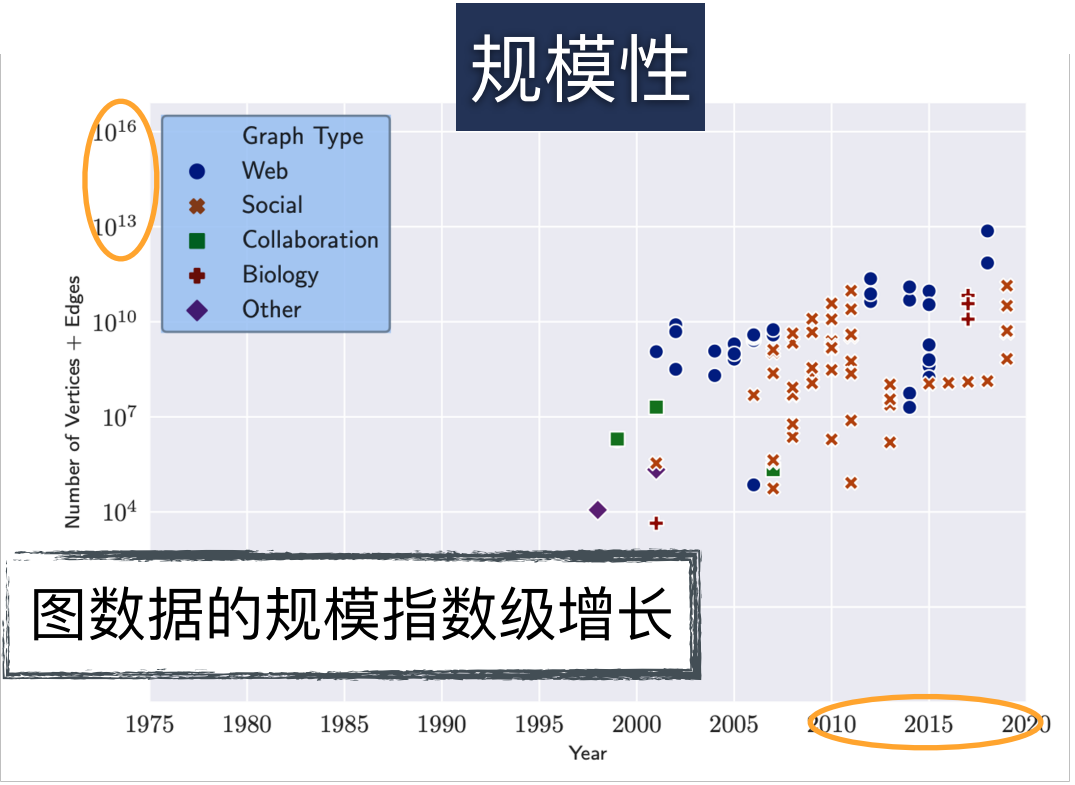


本研究聚焦于单机图查询系统

单机大规模图数据查询系统挑战

图数据处理难

规模性

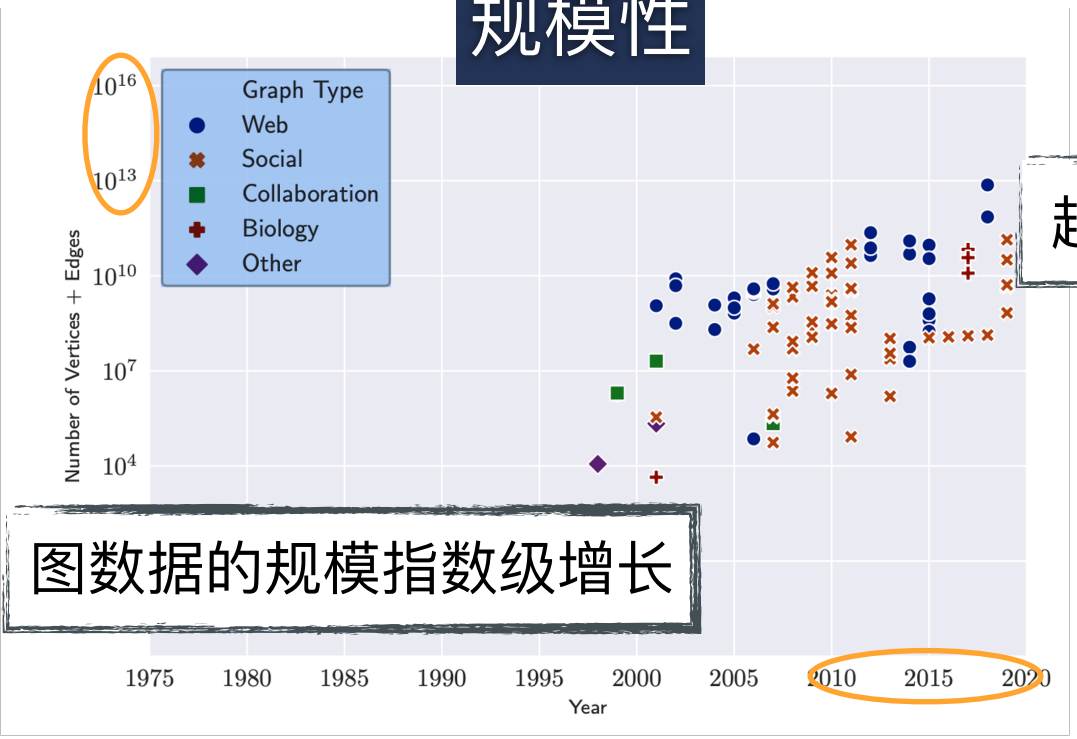


图数据的规模指数级增长

单机大规模图数据查询系统挑战

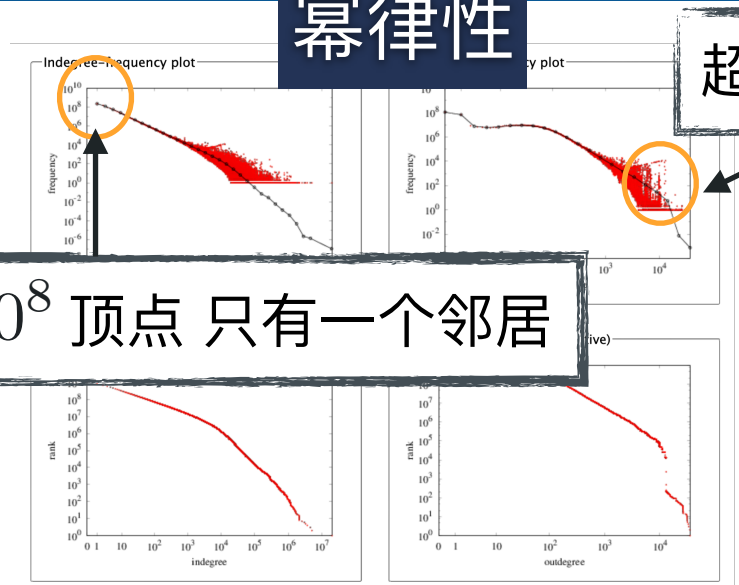
图数据处理难

规模性



图数据的规模指数级增长

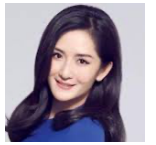
幂律性



超过 10^8 顶点 只有一个邻居

超级节点

用户 粉丝数



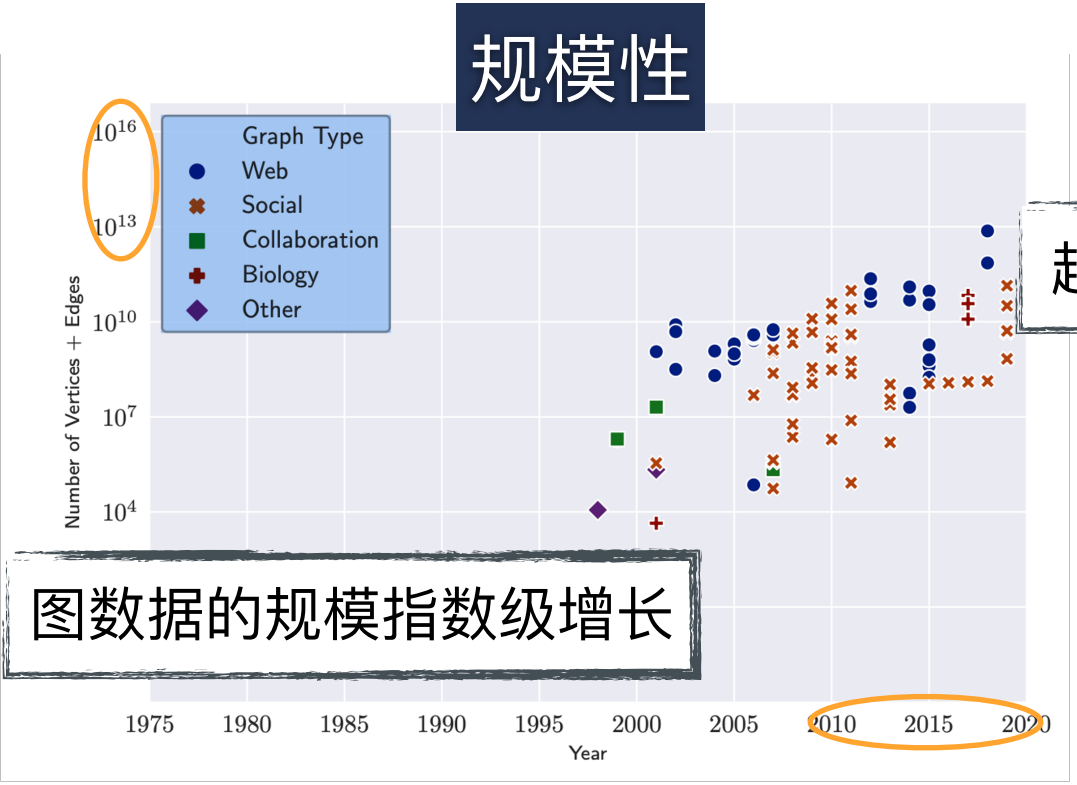
1.27亿

1000+

单机大规模图数据查询系统挑战

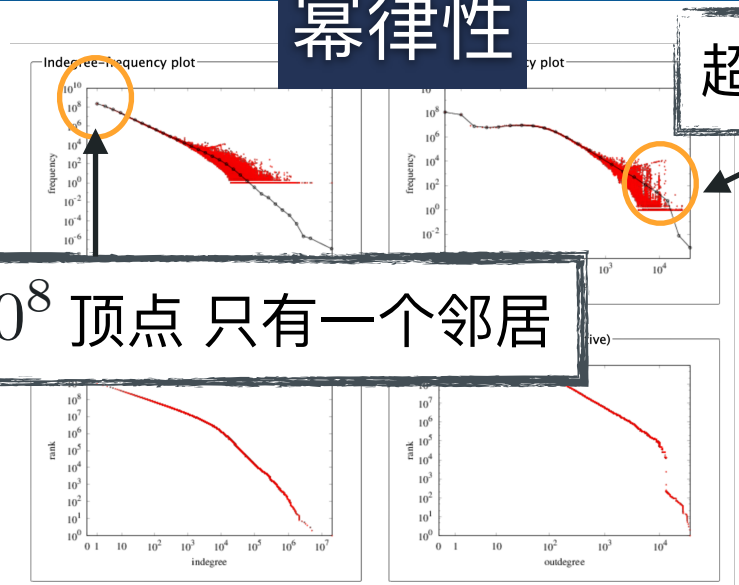
图数据处理难

规模性



图数据的规模指数级增长

幂律性



超过 10^8 顶点 只有一个邻居

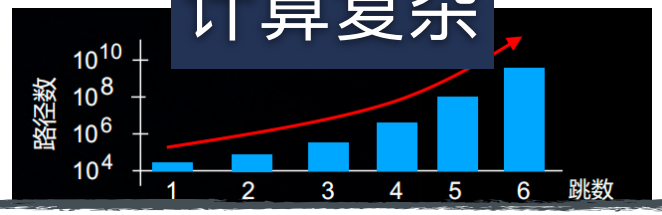
超级节点

用户 粉丝数

1.27亿

1000+

计算复杂



一次6跳环路检测在519M顶点图上
平均需要遍历超过20亿条路径

大规模、幂律分布与高计算复杂性

单机大规模图数据查询系统研究挑战与机遇

挑战：资源受限

性能维度	单机系统	分布式系统
最大并行度	CPU物理核心数（<128）	节点数 X 节点核心数（理论无上界）
内存容量上限	内存容量上限（受主板限制）	计算节点内存总和（理论无上界）
数据局部性	全内存访问延迟（<100ns）	跨节点访问（1us）

机遇一：新型硬件的快速普及

高并发算力

GPU:
2k~5k cores,
12-16GB DDR 5

10GB/s

7000MB/s

SSD:
4TGB

NVMe SSD:
256GB~768GB

12~40GB/s

CPU:
6~32 cores

高吞吐量外存设备

榨干单台机器的性能极限，使得单机也能处理大规模图数据

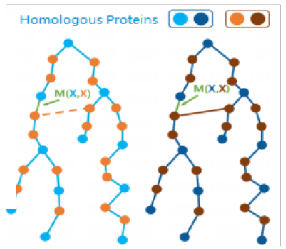
单机大规模图数据查询系统研究挑战与机遇

挑战：资源受限

性能维度	单机系统	分布式系统
最大并行度	CPU物理核心数（<128）	节点数 X 节点核心数（理论无上界）
内存容量上限	内存容量上限（受主板限制）	计算节点内存总和（理论无上界）
数据局部性	全内存访问延迟（<100ns）	跨节点访问（1us）

机遇二：专用图查询系统优化研究

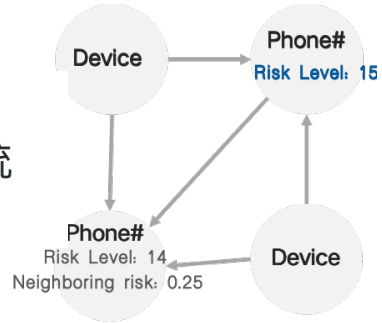
药物发现



Accuracy compared with ML models alone

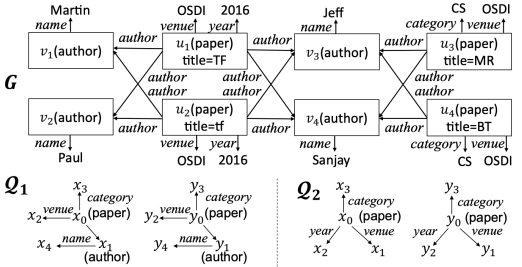
43%

反金图系统



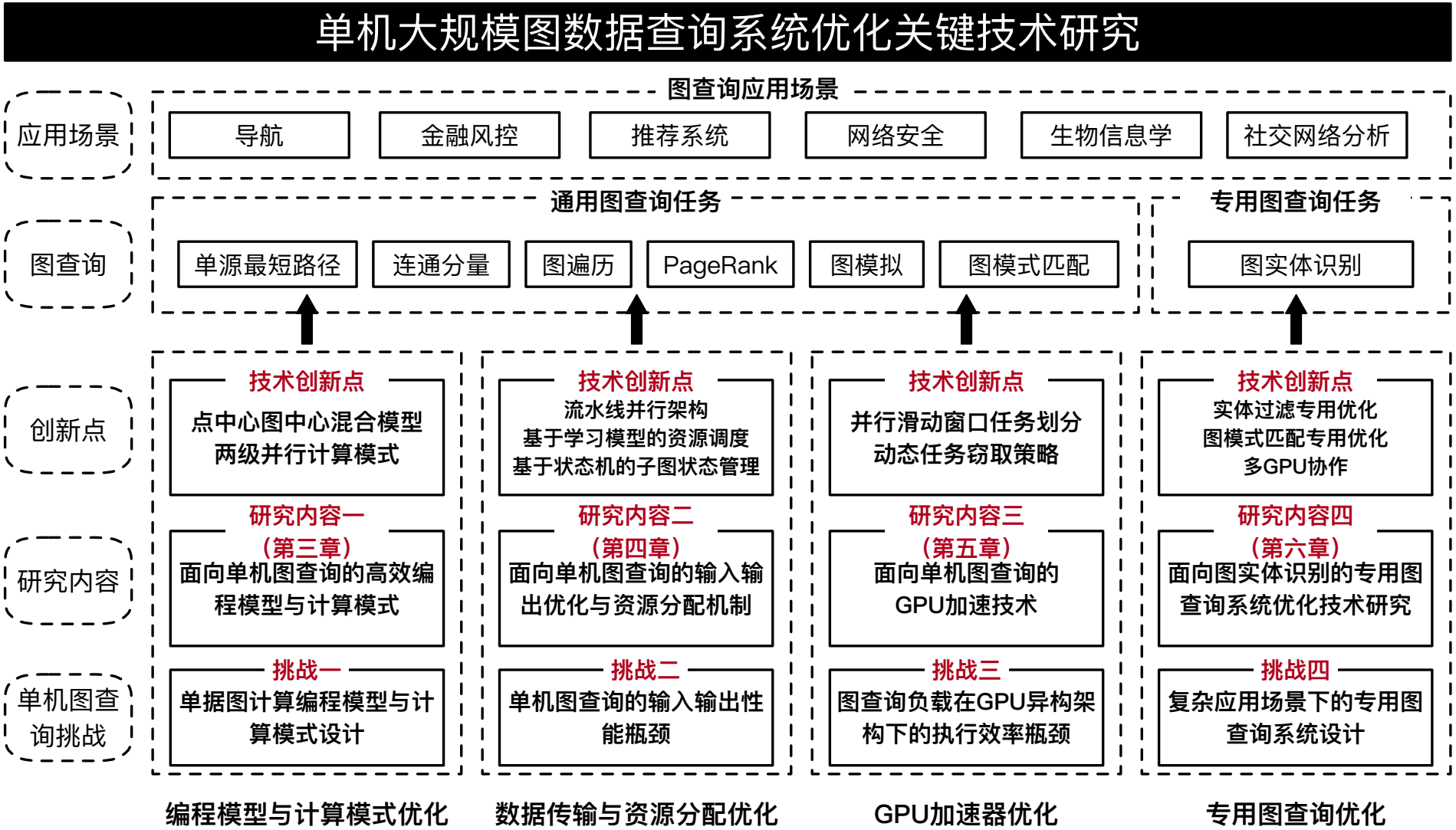
图数据清洗系统

8x



榨干单台机器的性能极限，使得单机也能处理大规模图数据

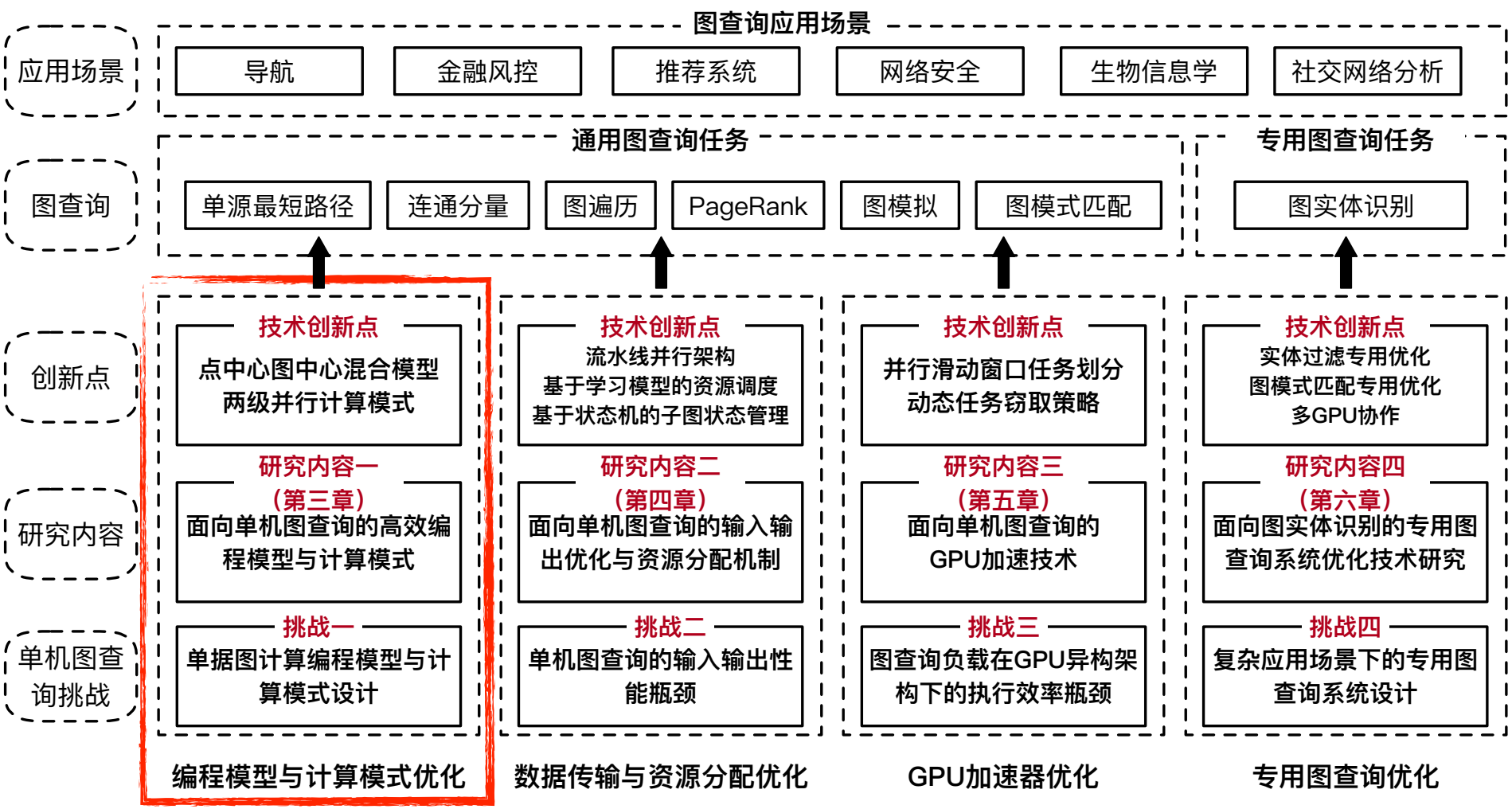
单机大规模图数据查询系统关键技术研究



主要研究内容

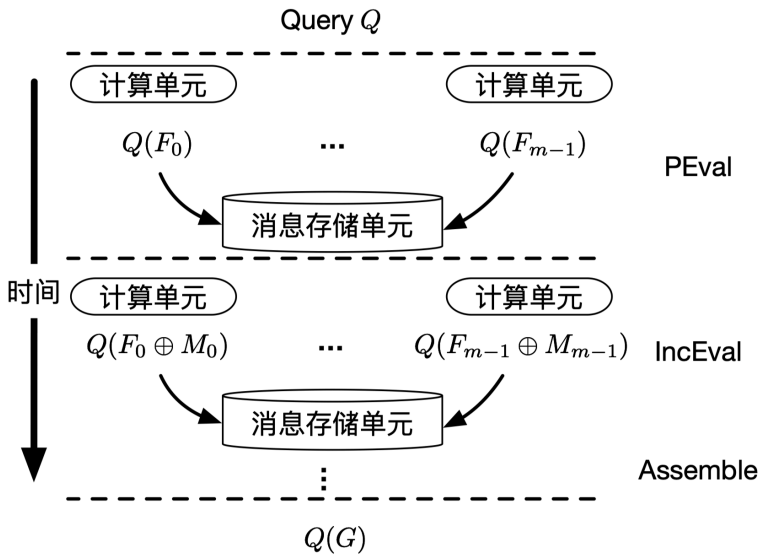
主要研究工作一

单机大规模图数据查询系统优化关键技术研究



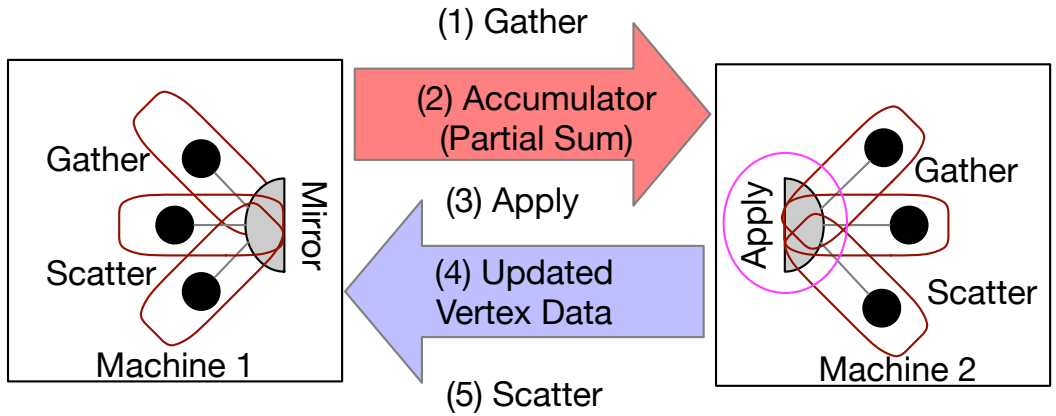
挑战一：图计算并行模型单机环境适配难

图中心模型



- ✓ 图中心模型将全图细节暴露给用户，用户可以实现更优的算法
- ✓ 基于数据划分，BSP的子图级同步并行
- ✓ 并行度 <= 子图数

点中心模型



- ✓ 点中心模型为用户暴露顶点及其邻居的细节
- ✓ 顶点级并行潜在并行度更高
- ✓ 并行度 <= 顶点数

挑战一：图计算并行模型单机环境适配难

不同的并行模型在单机环境下，具有显著性能差异

实验环境

- ✓ 查询：弱连通分量查询
- ✓ 环境：16GB DDR4 内存、Intel Xeon W-2255 （10 核 20 线程）
- ✓ 数据集: web-sk(32GB), friendster:(29 GB)

发现

- ✓ 当内存不足以容纳整个图G时，点中心模型引发更多的数据读取操作
- ✓ 当图的直径增大，点中心模型引发更多的数据传输开销

并行模型	Friendster (18亿边，29GB，平均直径5.1)			Web-sk（18亿边，32GB，平均直径13.7）		
	超步	外存读	并行度	超步	外存读	并行度
点中心模型	21	135GB	10 （取决于可用核数）	120	367GB	10 （取决于可用核数）
图中心模型	6	74GB	4（取决于子图数量）	9	82GB	4（取决于子图数量）

I/O是外存系统的性能瓶颈

挑战一：图计算并行模型单机环境适配难

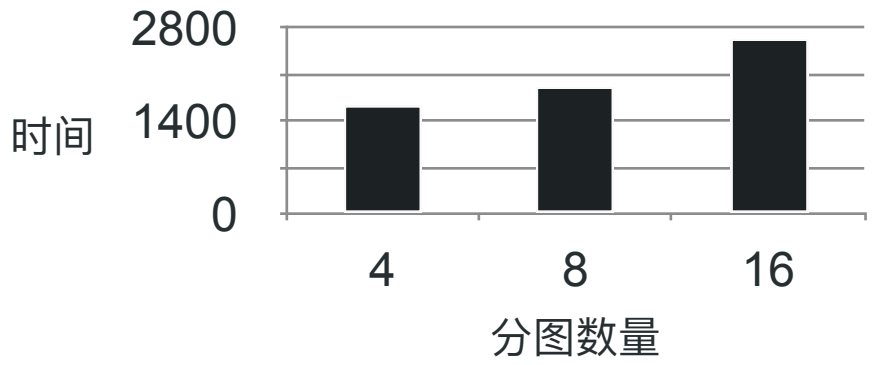
不同的并行模型在单机环境下，具有显著性能差异

实验环境

- ✓ 查询：弱连通分量查询
- ✓ 环境：64GB DDR4 内存、Intel Core i9 CPU （10 核 20 线程）
- ✓ 数据集: clueweb:(137GB GB)

发现

- ✓ 采用图中心模型会引入图碎片化问题，从而削弱系统性能。



基于数据划分并行的图中心模型，并行度高将损害系统性能

PIE+：图中心/点中心混合模型

问题

- ✓ 点中心模型在单机环境下数据传输开销大
- ✓ 图中心模型在单机环境下易产生“碎片化”问题

解决思路

- ✓ **混合模型**：支持基于数据划分并行的图中心模型，在**PEval** 和**IncEval** 插入支持操作级并行的点中心模型**EMap**和**VMap**
- ✓ **两极并行模式**：子图间并行 + 子图内并行
- ✓ **特点**：**正确性、易用性、高性能**

算法 3: 弱联通分量查询在 PIE+ 下的 PEval计算流程

Input: Subgraph $F_i = (V_i, E_i, L_i)$

Output: Set CC_i of connected components in F_i

```
1  CC_i := ∅; Π := V_i; ;                    // Π is the set of unvisited vertices
2  while Π not empty do
3      find root vertex v_r, such that L_i(v_r) = min_{u∈Π} L_i(u);
4      CC_i := CC_i ∪ {v_r}; remove v_r from Π;
5      Δ := {v_r};
6      while Δ not empty do
7          Δ := EMap(Δ, BFSRecur);
8      end
9  end
10 return CC_i;
11 Procedure BFSRecur(u, v):
12     if v ∉ Π then
13         return ∅;
14     end
15     remove v from Π; v.root := v_r; return {v};
```

图中心程序 PEval

并行执行点中心程序

点中心程序

算法 4: 弱联通分量查询在 PIE+ 下的 IncEval计算流程

Input: Subgraph $F_i = (V_i, E_i, L_i)$, connected components CC_i, incoming messages M_i

Output: Refined connected components CC_i

```
1  Δ = {v | v ∈ M_i}; VMap(Δ, UpdateRoot);
2  return refine(CC_i); ;                    // merge roots with identical labels in CC_i
3  Message Segment: M_i := {(v, L_i(v.root)) | v ∈ C_i};
4  Procedure UpdateRoot(v):
5      v_r := v.root;
6      while v_r ∉ CC_i do
7          v_r := v_r.root;
8      end
9      update v.root := v_r; L_i(v_r) := min(L_i(v_r), M_i[v]);
10     return ∅;
```

并行执行点中心程序

点中心程序

图中心程序 IncEval

通过*VC*的子图内并行丰富*GC*的子图间并行

实验分析

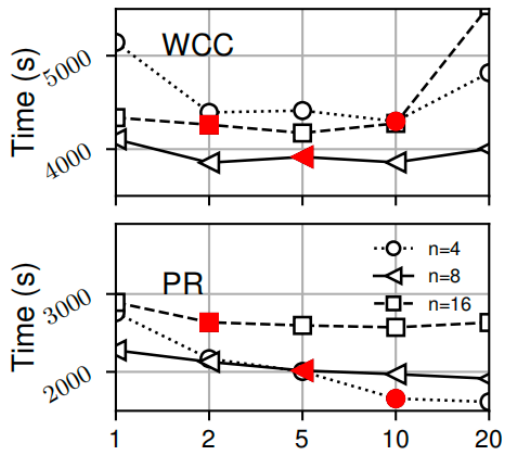
实验设计

数据集

- ✓ 环境：16GB DDR4 内存、Intel Xeon W-2255 （10 核 20 线程）
- ✓ 数据集： Friendster (|V|=65M, |E| = 1.9B, 31GB)， web-sk（|V|=50, |E|=1.8B, 31GB）
- ✓ 查询：WCC

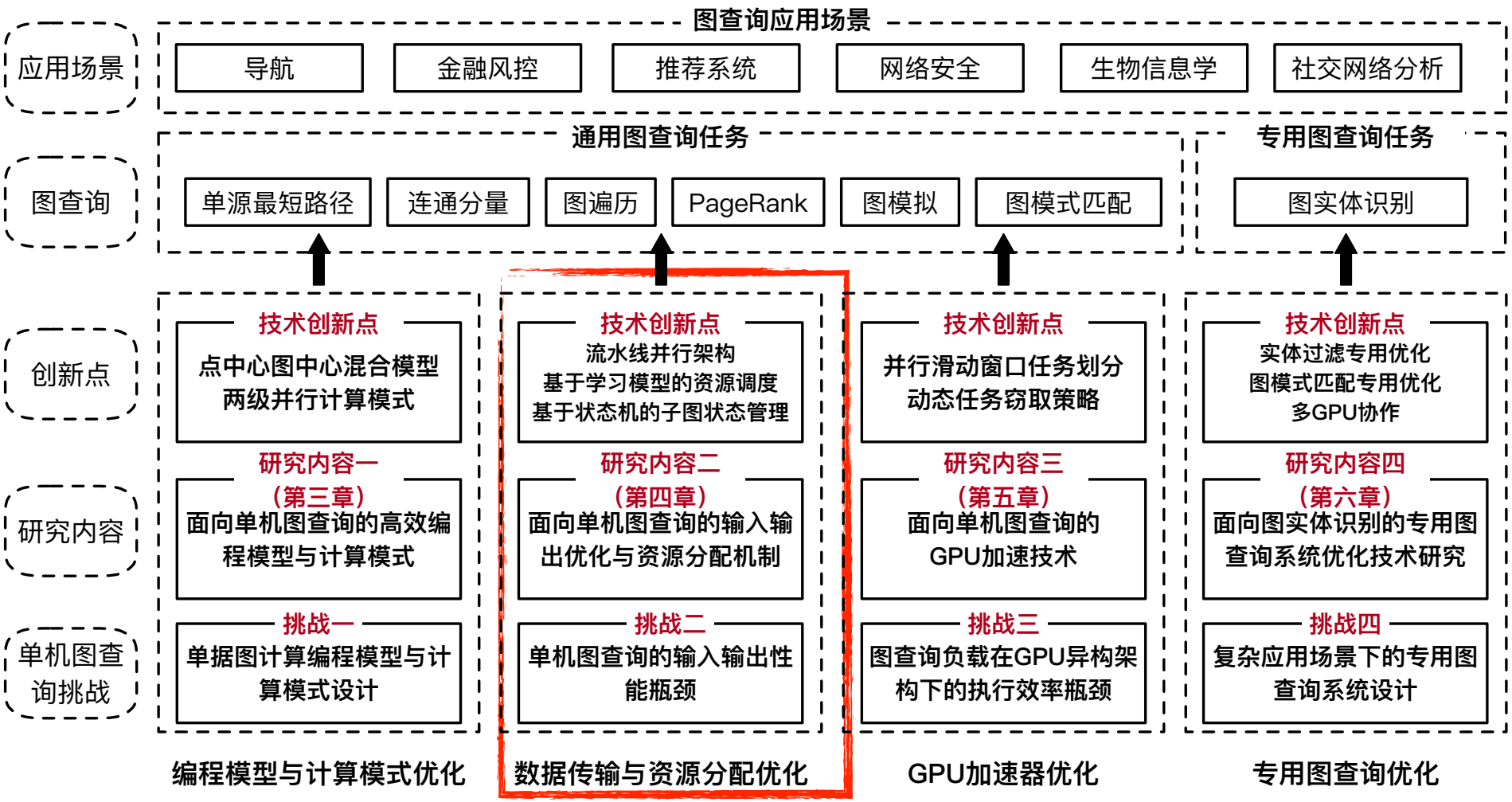
实验结果

Dataset	Metric	SSSP		WCC		PR	
		MiniGraph	GridGraph	MiniGraph	GridGraph	MiniGraph	GridGraph
friendster	# Supersteps	**8**	32	**6**	21	8	10
	Disk Read (GB)	**78**	115.1	**74**	135	**107**	160
	Shortcut I/O (GB)	**-12**	N/A	**-12**	N/A	**-10.4**	N/A
	Avg. CPU Util.	**33.74%**	4.45%	**48.2%**	6.83%	**68.46%**	62.38%
	I/O-Compute Corr.	**0.095**	-0.113	**0.163**	-0.202	**0.185**	-0.156
	Cache Hits	**45.33%**	9.59%	**48.25%**	12.04%	**34.8%**	**36.2%**
web-sk	# Supersteps	**10**	63	**9**	120	15	20
	Disk Read (GB)	**112.5**	232	**81.9**	367	**87**	232
	Shortcut I/O (GB)	**-30.9**	N/A	**-6.1**	N/A	**-20.9**	N/A
	Avg. CPU Util.	**15.76%**	5.83%	**25.04%**	5.16%	42%	**42%**
	I/O-Compute Corr.	**0.008**	0.003	**0.013**	0.009	**0.082**	-0.039
	Cache Hits	**50.89%**	6.37%	**37.42%**	11.63%	**50.22%**	46.04%



主要研究工作二

单机大规模图数据查询系统优化关键技术研究



挑战：I/O瓶颈导致的资源利用率低下

已有图查询系统

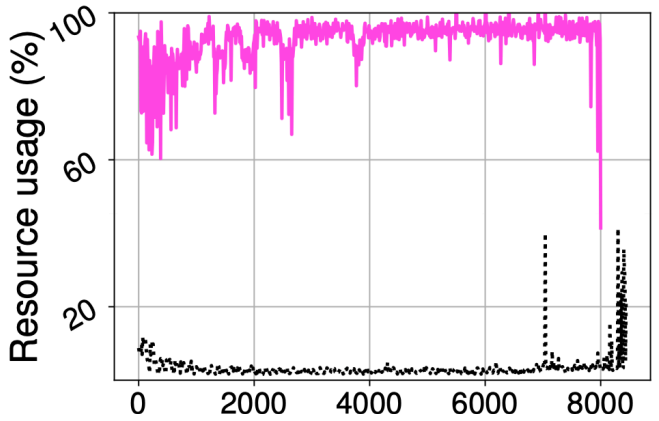
✓ 当图数据规模显著超过内存容量时，I/O操作已成为现有图查询系统的主要性能瓶颈。

实验示例

✓ 查询：弱连通分量查询

✓ 环境：64GB DDR4 内存、Intel Core i9 （10 核 20 线程）、1TB SATA SSD（顺序读取 560MB/s）

✓ 数据集: clueweb(134GB)



GridGraph 资源利用率分析： 黑色 CPU利用率、粉红色I/O带宽利用率

I/O是外存系统的性能瓶颈，如何提升资源利用率

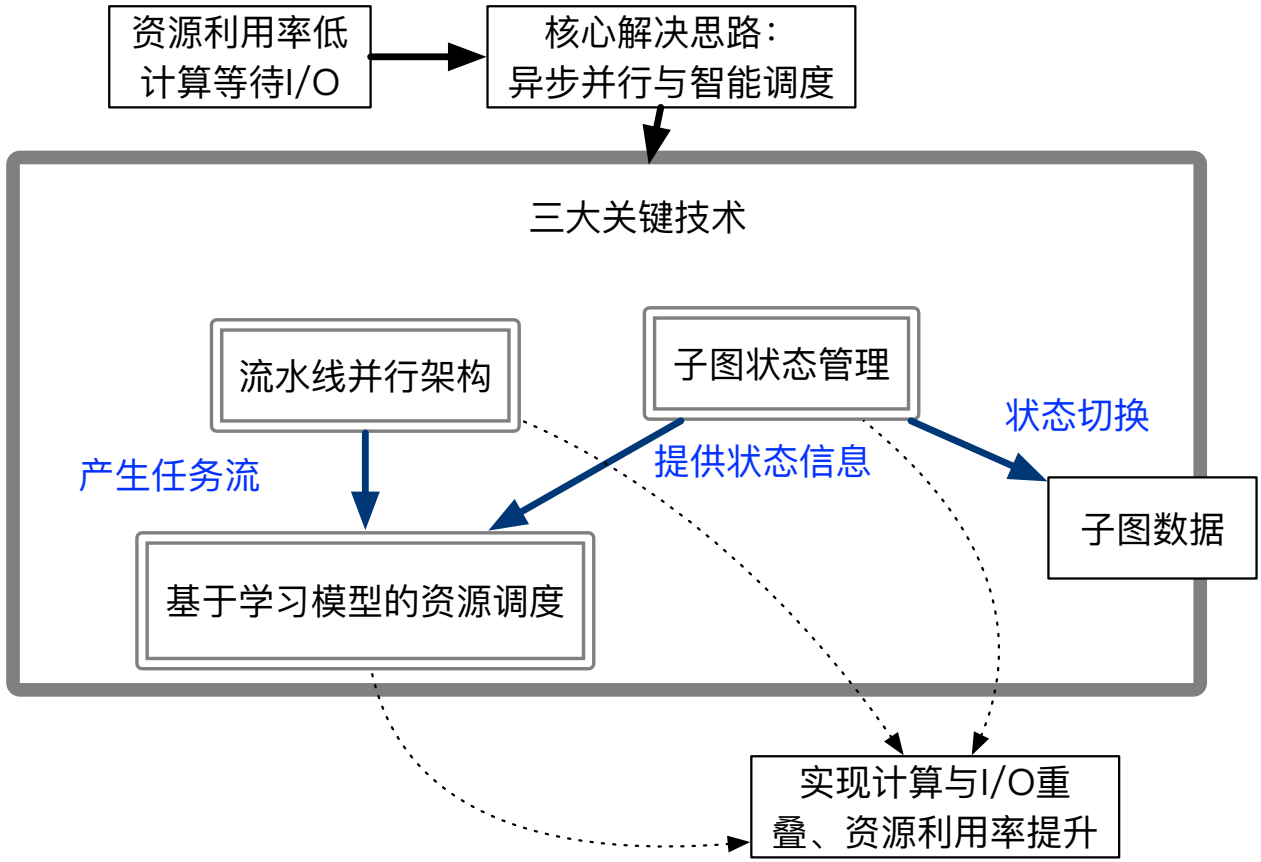
MiniGraph: 单机外存图查询系统加速引擎

问题

- ✓ 外存图查询系统资源利用率低

解决思路

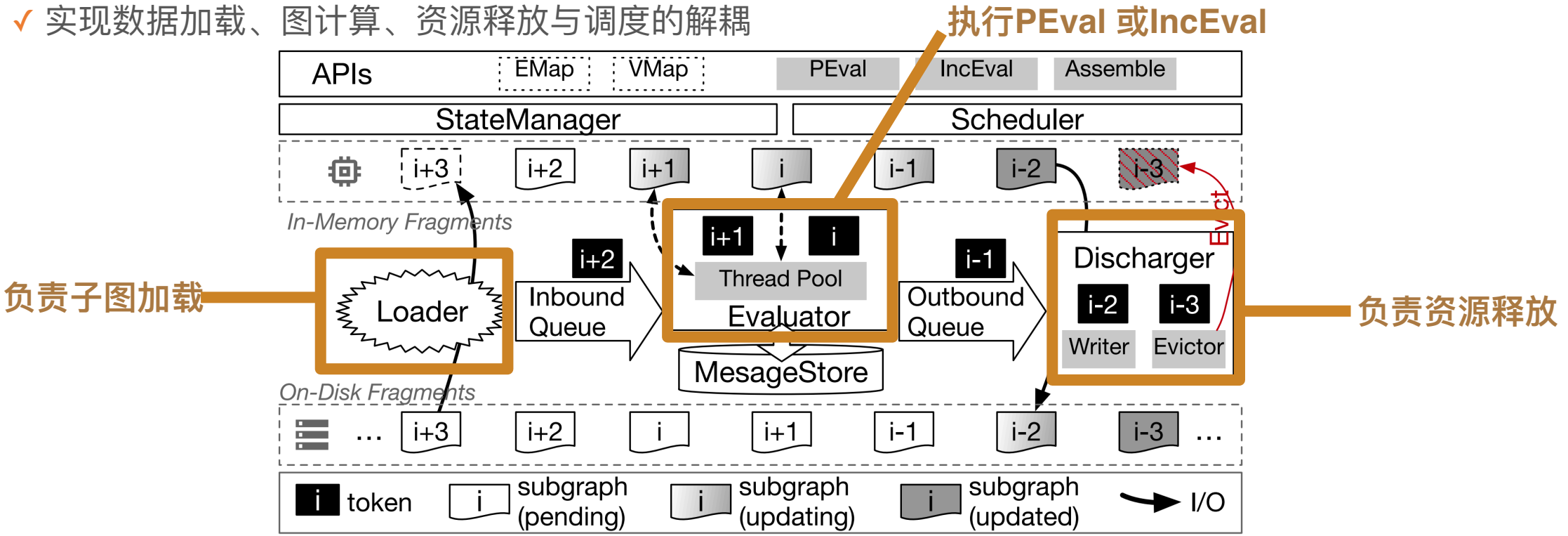
- ✓ 流水线并行架构
- ✓ 基于学习模型的资源调度
- ✓ 子图状态管理



贡献点一： 异步流水线架构

异步流水线架构

- ✓ 一种通过重叠 I/O 和 CPU 操作以“抵消”过高 I/O 成本的流水线架构
- ✓ 实现数据加载、图计算、资源释放与调度的解耦



异步流水线架构

问题定义

问题提出

- ✓ 何时加载并处理子图?
- ✓ 如何分配资源以实现两级并行度的最大化?

调度决策问题的目标函数

$$\arg\min_{\mathcal{S}} \max_{i\in[0,n)} \{t_i + C_{\mathcal{A}}(F_i, p_i)\}$$

约束条件

- ✓ 消耗的内存不能超过系统内存
- ✓ 总线程数不能超过系统线程数
- ✓ 只有当子图被加载进内存才能开始计算

调度决策问题是NP-完全的

贡献点二： 基于学习模型的资源调度

问题定义

问题提出

- ✓ 何时加载并处理子图?
- ✓ 如何分配资源以实现两级并行度的最大化?

调度决策问题的目标函数

$$\arg\min_{\mathcal{S}} \max_{i\in[0,n)} \{t_i + C_{\mathcal{A}}(F_i, p_i)\}$$

约束条件

- ✓ 消耗的内存不能超过系统内存
- ✓ 总线程数不能超过系统线程数
- ✓ 只有当子图被加载进内存才能开始计算

调度决策问题是NP-完全的

启发式解决办法

1. 从日志数据中训练模型评估执行代价

$$C_{\mathcal{A}}(F_i, p_i) = \sum_{v\in V(F_i)} [h^{seq}(\hat{x}_i(v)) + \frac{h^{para}(\hat{x}_i(v))}{\min\{p_i, \lfloor d_i \rfloor\}}]$$

2. 试探性资源分配策略

- ✓ 初始状态根据子图的特征（大小，边缘顶点数量分配）

3. 根据学习模型动态调整

- ✓ 根据学习模型，优先加载计算成本最高且满足约束条件的子图

贡献点三： 子图状态管理

问题提出

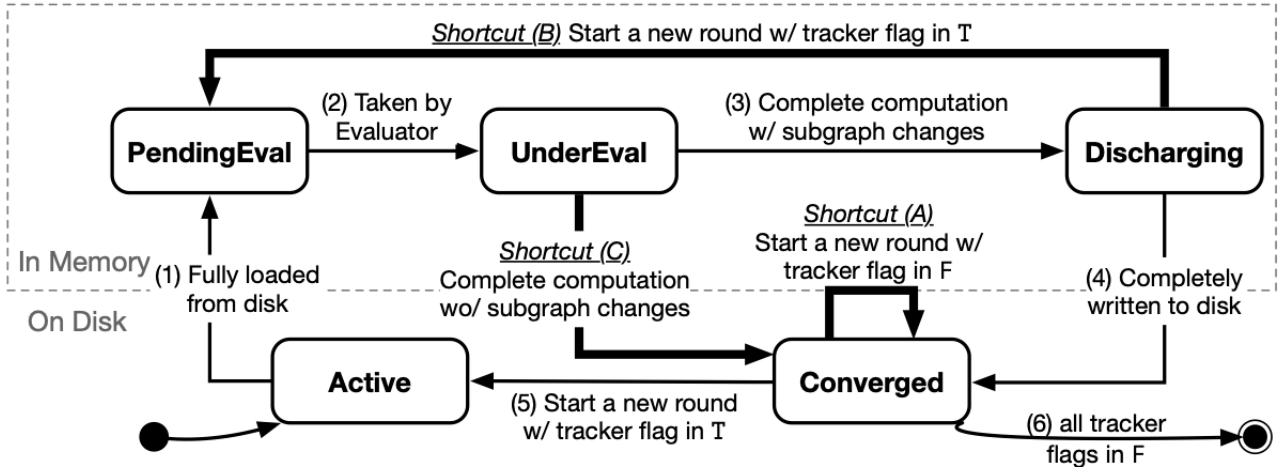
✓ 如何有效削减冗余I/O操作，并优化系统内部子图的协同管理机制

子图状态管理

✓ 采用**轻量级状态机**架构，持续追踪各子图的核心计算进度

✓ 基于**不动点检测**机制动态判断**系统停止条件**

✓ 通过状态转换过程中的决策，自动**跳过非必要**的计算与I/O操作



子图状态转换图

实验设计

数据集

Name	Type	$\|V\|$	$\|E\|$	MaxDegree	Raw Data
roadNetCA [1]	road network	2M	2.7M	23	83MB
skitter [42]	network topology	1.6M	11M	35455	142MB
twitter [8, 40]	social network	41.6M	1.5B	3M	25GB
friendster [5]	social network	65.6M	1.8B	5124	30.14GB
web-sk [55]	Web	50M	1.9B	8.5M	32GB
clueWeb [55]	Web	1.7B	7.9B	6.4M	137GB

测试环境

- ✓ Ubuntu Server 20.04 LTS
- ✓ Intel Core i9-7900X CPU @3.30GHz
- ✓ 13.75MB LLC
- ✓ 10 cores (20 hyper threads)
- ✓ 64GB of DDR4-2666 memory
- ✓ 1TB WD blue SATA SSD, whose read throughput is 560MB/s.

基线

单机系统

- ✓ GridGraph[ATC'15],GraphChi[OSDI'12], XStream[SIGOPS'13]

分布式系统

- ✓ GraphScope[VLDB'21],Gluon[PLDI'18]

图查询

- ✓ WCC
- ✓ PageRank
- ✓ SSSP
- ✓ BFS
- ✓ Random Walk
- ✓ Simulation

实验结果

Benchmark

Data	Memory Budget	#Partitions (PR/Others)	SSSP				WCC				PR			
			MiniGraph	GraphChi	GridGraph	XStream	MiniGraph	GraphChi	GridGraph	XStream	MiniGraph	GraphChi	GridGraph	XStream
roadNetCA	100%	1/1	8.66	22.5 (2.6×)	10.55 (1.2×)	**2 (0.2×)**	**2.76**	17.2 (6×)	18.22 (6.6×)	2.93 (1.1×)	**0.25**	0.91 (3.5×)	0.71 (2.7×)	2.34 (2.6×)
skitter	100%	1/1	0.53	1.64 (78.5×)	**0.35 (0.67×)**	0.69 (1.3×)	**0.16**	3.43 (115.2×)	0.33 (2.1×)	0.59 (3.9×)	**0.27**	1.27 (4.7×)	0.82 (3.0×)	0.98 (3.6×)
twitter	50% (12.5GB)	4/10	**150.8**	802.8(5.3×)	195.4(1.29×)	2365(15.6×)	**159.5**	594.8(3.7×)	186(1.2×)	1983(12.4×)	**224.2**	782.1(3.5×)	371.3(1.7×)	2183(9.7×)
friendster	50% (15.07GB)	4/10	**201.8**	535(2.7×)	293.1(1.45×)	3061(15.2×)	**171.8**	1636(9.5×)	204.7(1.2×)	2037(11.8×)	**190.104**	450.7(1.9×)	485.3(1.9×)	2685(11.3×)
web-sk	50% (16GB)	4/10	**326.4**	1140(3.5×)	917.9(2.8×)	9437 (28.9×	**172**	620.1(3.6×)	704.6(4.1×)	4056(23.5×)	**248.3**	2288(9.2×)	395(1.6×)	2903(11.7×)
clueWeb	47% (64GB)	4/10	**2514**	/	11534 (4.59×)	/	**2742**	/	11665 (4.25×)	/	**2022**	/	3803(2.1×)	/
	10% (13.7GB)	20/50	**5871**	/	/	/	**7486**	/	/	/	**2979**	/	/	/

结果概览

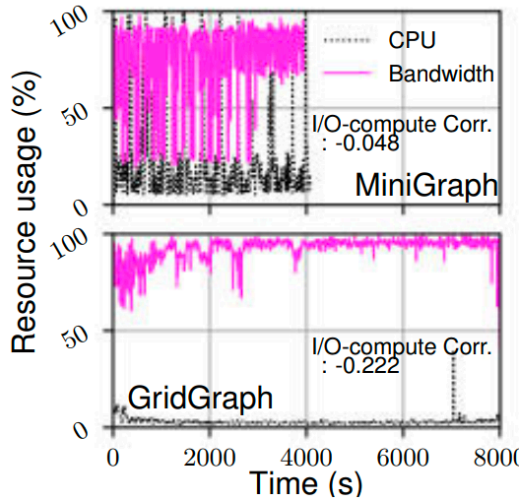
✓ MiniGraph 比基线系统（ GridGraph, GraphChi and XStream）快最多 **4.6×**, **9.5×** 和 **28.9×**

实验结果：运行时统计与资源使用情况对比

运行时统计信息

Dataset	Metric	SSSP		WCC		PR	
		MiniGraph	GridGraph	MiniGraph	GridGraph	MiniGraph	GridGraph
friendster	# Supersteps	**8**	32	**6**	21	8	10
	Disk Read (GB)	**78**	115.1	**74**	135	**107**	160
	Shortcut I/O (GB)	**-12**	N/A	**-12**	N/A	**-10.4**	N/A
	Avg. CPU Util.	**33.74%**	4.45%	**48.2%**	6.83%	**68.46%**	62.38%
	I/O-Compute Corr	**0.095**	-0.113	**0.163**	-0.202	**0.185**	-0.156
	Cache Hits	**45.33%**	9.59%	**48.25%**	12.04%	**34.8%**	**36.2%**
web-sk	# Supersteps	**10**	63	**9**	120	15	20
	Disk Read (GB)	**112.5**	232	**81.9**	367	**87**	232
	Shortcut I/O (GB)	**-30.9**	N/A	**-6.1**	N/A	**-20.9**	N/A
	Avg. CPU Util.	**15.76%**	5.83%	**25.04%**	5.16%	42%	**42%**
	I/O-Compute Corr	**0.008**	0.003	**0.013**	0.009	**0.082**	-0.039
	Cache Hits	**50.89%**	6.37%	**37.42%**	11.63%	**50.22%**	46.04%

*CPU/IO*资源利用率分析

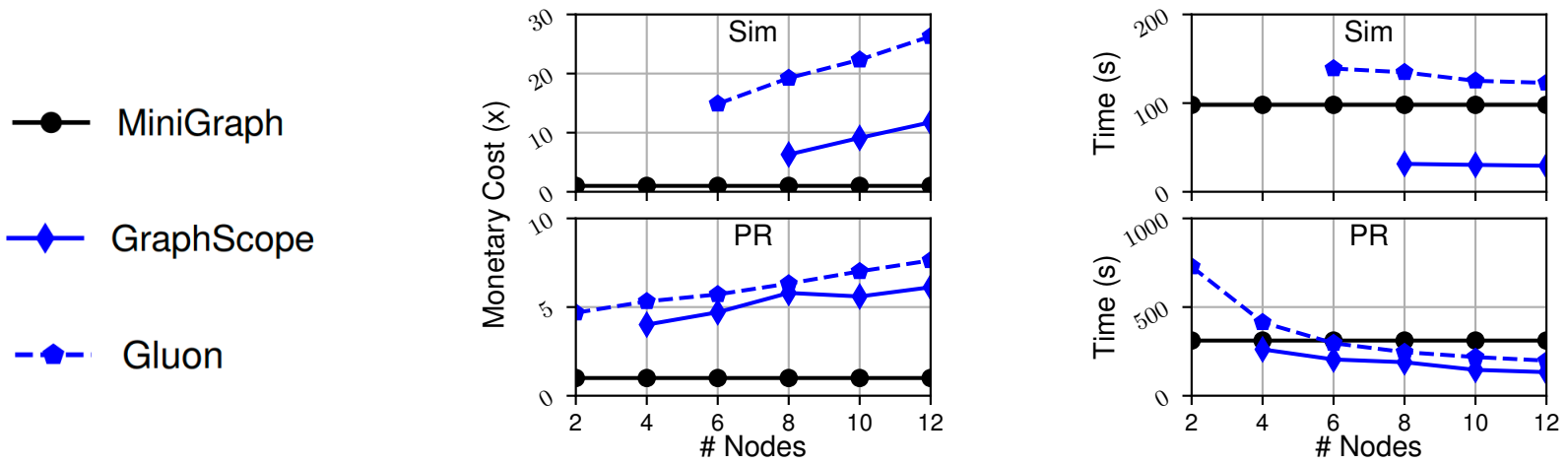


发现

- ✓ MiniGraph 在执行 SSSP 和 WCC 任务时所需的超级步数量不到 GridGraph 的 29%，磁盘读取流量也不到其 53.3%。
- ✓ 相比表现最优的基线系统 GridGraph，MiniGraph 的 CPU 利用率最高提升了 41.4%。

实验结果：对比分布式系统

MiniGraph VS distributed systems

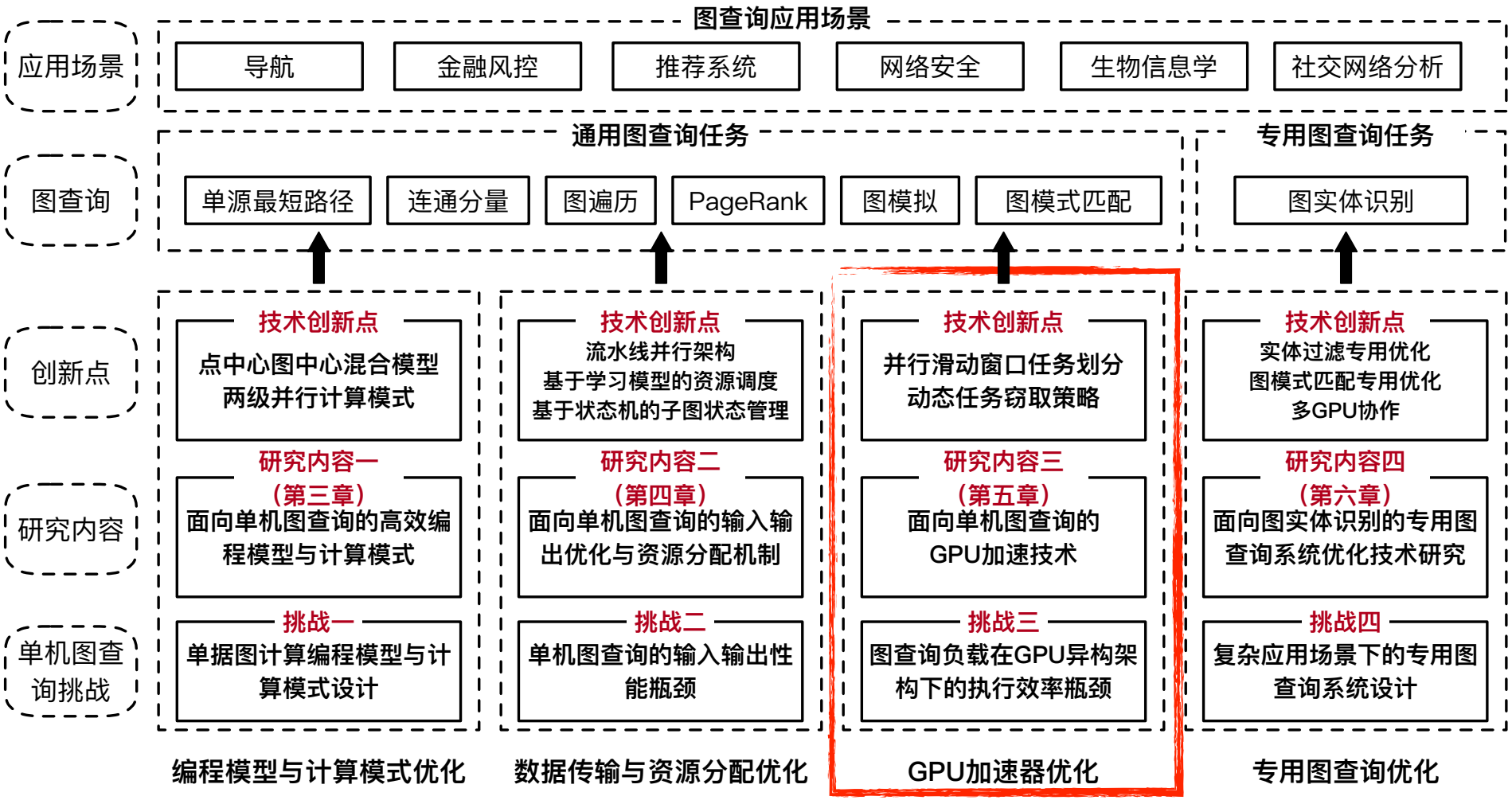


发现

✓ 在子图模拟任务中，MiniGraph 的表现优于由12台机器组成的分布式图分析系统 Gluon，同时将多机系统的金钱成本降低了3.0至13.9倍。

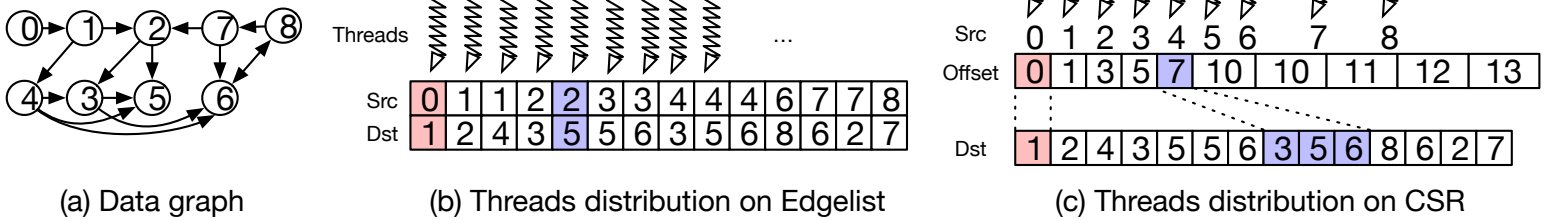
主要研究工作三

单机大规模图数据查询系统优化关键技术研究



挑战：图计算难以高效利用*GPU*

*GPU*的高效计算与图数据的倾斜性之间的矛盾

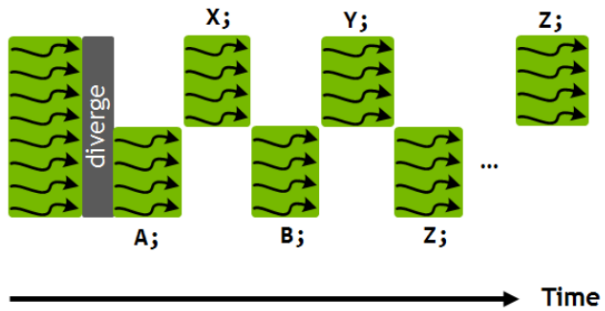


(a) Data graph　(b) Threads distribution on Edgelist　(c) Threads distribution on CSR

挑战

- ✓ 图数据的顶点度的高度倾斜
- ✓ GPU线程束负载失衡引发高同步开销

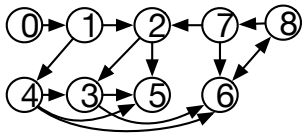
```
if (threadIdx.x < 4) {
    A;
    B;
} else {
    X;
    Y;
}
Z;
```



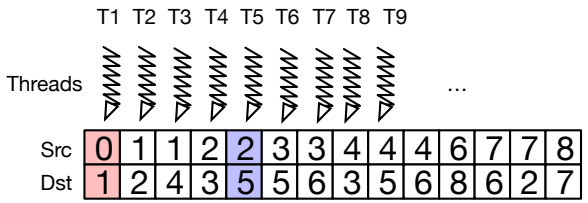
GPU SIMT 架构执行流程示例

挑战：图计算难以高效利用*GPU*

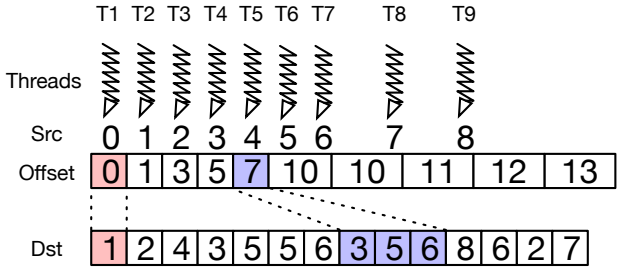
*GPU*的高效计算与图数据的倾斜性之间的矛盾



(a) Data graph



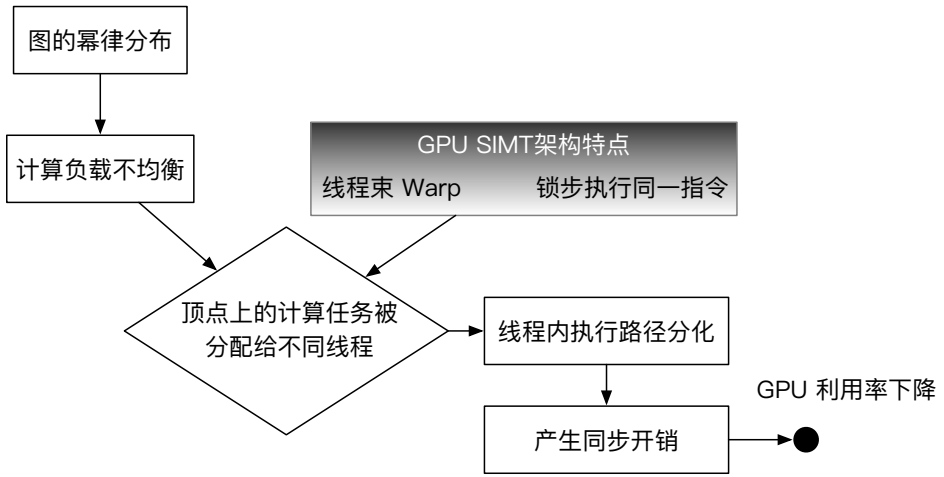
(b) Threads distribution on Edgelist



(c) Threads distribution on CSR

挑战

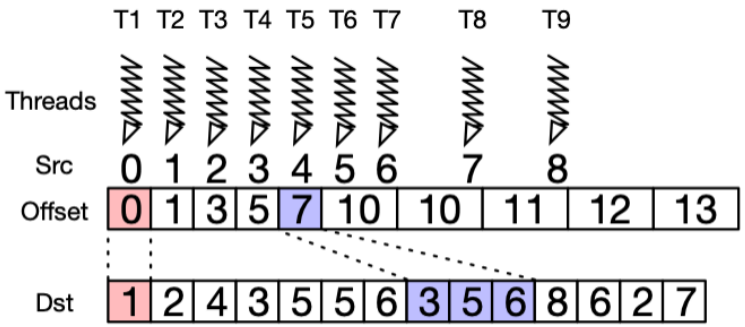
- ✓ 图数据的顶点度的高度倾斜
- ✓ GPU线程束负载失衡引发高同步开销

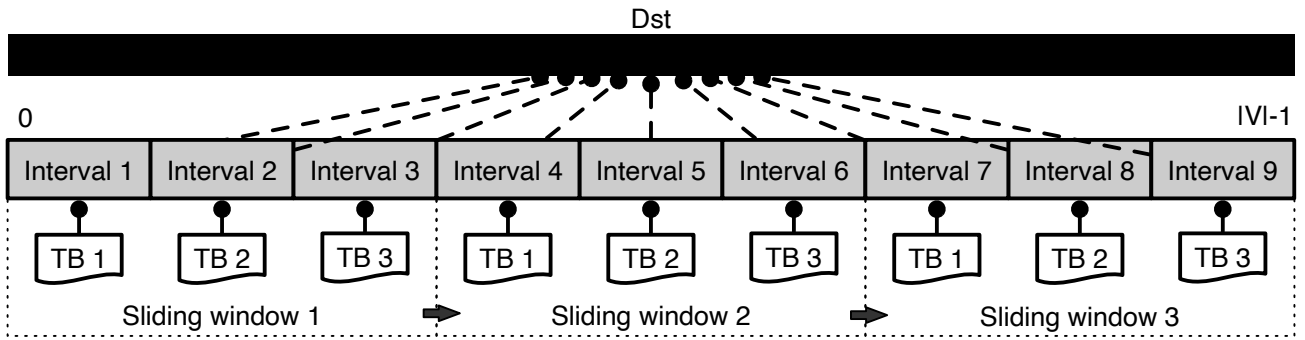


贡献点一： 基于滑动窗口的静态任务分配策略

基于滑动窗口的静态任务分配策略

- ✓ 对Src中顶点，按顶点度降序排序
- ✓ 将Src划分为 $\frac{|V|}{n_t}$ 个区间（Interval），并为GPU中的每个线程块（TB）按序分配一个区间
- ✓ 线程块（TB）处理完当前区间，移动至滑动窗口的下一个对应区间



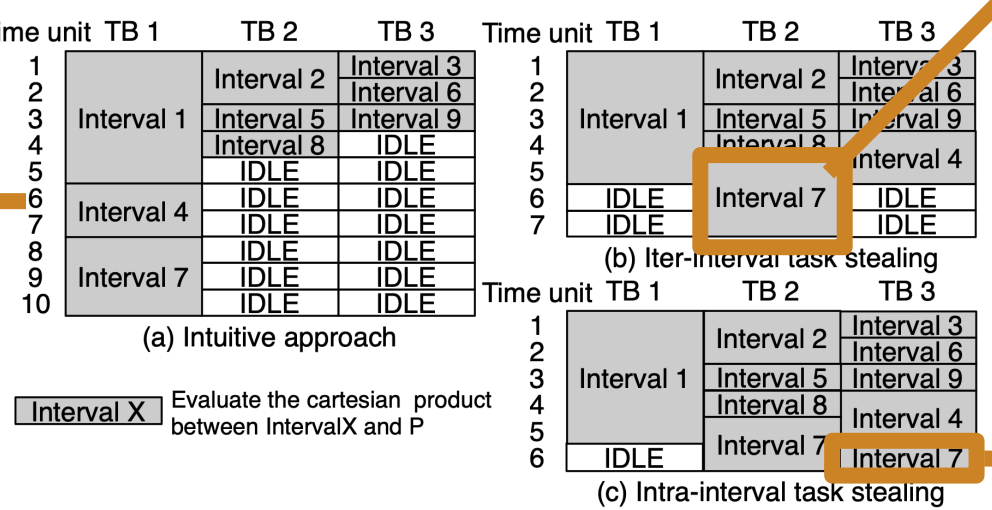


贡献点二： 基于共享内存的动态任务窃取

基于共享内存的动态任务窃取策略

- ✓ 基于滑动窗口的静态任务划分不能保证负载均衡，由于每个区间的计算量可能不一样
- ✓ 区间间任务窃取
- ✓ 区间内任务窃取

并行滑动窗口分配结果



区间间任务窃取

$$start_a = start_b + \frac{start_b + end_b}{2}$$

$$end_a = end_b$$

区间内任务窃取

尽管每个TB1-TB3被分配了同样数量的区间，但负载依然不均衡

实验设计

数据集

Dataset	Type	|V|	|E|	MaxDegree
stanford	social network	0.2M	2M	-
patents	patents	3.8M	1.7M	-
livejournal	social network	4.8M	68.9M	-
twitter	social network	41.6M	1.4B	-
friendster	social network	65.6M	1.8B	5124
web-sk	web	50M	1.9B	8.5M

基线

SOTA

- ✓ CGGraph[VLDB'22]
- ✓ Subway[EuroSys'20]

测试环境

- ✓ Ubuntu Server 20.04 LTS
- ✓ Intel Core i9-7900X CPU @3.30GHz
- ✓ 13.75MB LLC
- ✓ 10 cores (20 hyper threads)
- ✓ 64GB of DDR4-2666 memory
- ✓ Tesla V100 GPU x 4

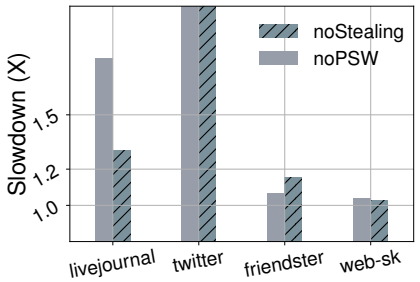
图查询

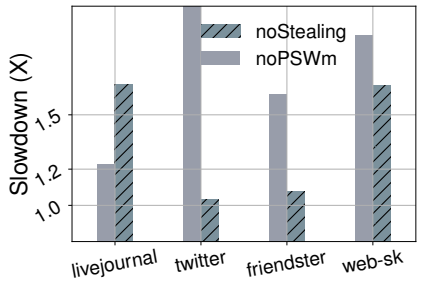
- ✓ WCC
- ✓ PageRank
- ✓ SSSP
- ✓ BFS

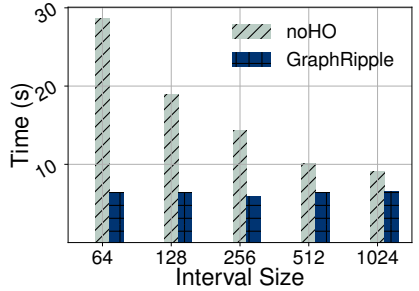
实验结果

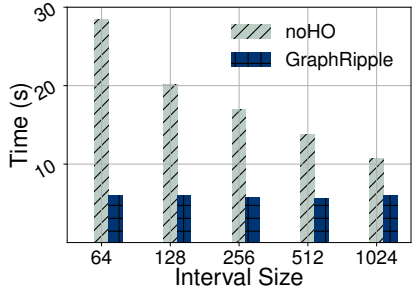
Benchmark

Data	PageRank			BFS			WCC		
	GraphRipple	CGGRaph	Subway	*GraphRipple*	CGGRaph	Subway	*GraphRipple*	CGGRaph	Subway
stanford	0.2	3.0 (15.0×)	0.4 (2.0×)	0.2	2.9 (14.5×)	0.4 (2.0×)	1.4	4.2 (3.0×)	0.4 (0.3×)
patents	0.5	3.2 (6.4×)	3.2 (6.4×)	0.4	3.1 (7.8×)	2.9 (7.3×)	0.5	3.3 (6.6×)	3.1 (6.2×)
livejournal	1.2	4.1 (3.4×)	13.7 (11.4×)	1.0	3.9 (3.9×)	12.7 (12.7×)	1.3	4.1 (3.2×)	12.6 (9.7×)
twitter	26.4	63.2 (2.4×)	>1min	18.4	20.5 (1.1×)	>1min	20.7	31.7 (1.5×)	>1min
friendster	22.7	31.6 (1.4×)	>1min	19.7	25.6 (1.3×)	>1min	21.3	35.3 (1.7×)	>1min
web-sk	32.0	34.7 (1.1×)	>1min	17.8	23.1 (1.3×)	>1min	25.4	29.3 (1.2×)	>1min







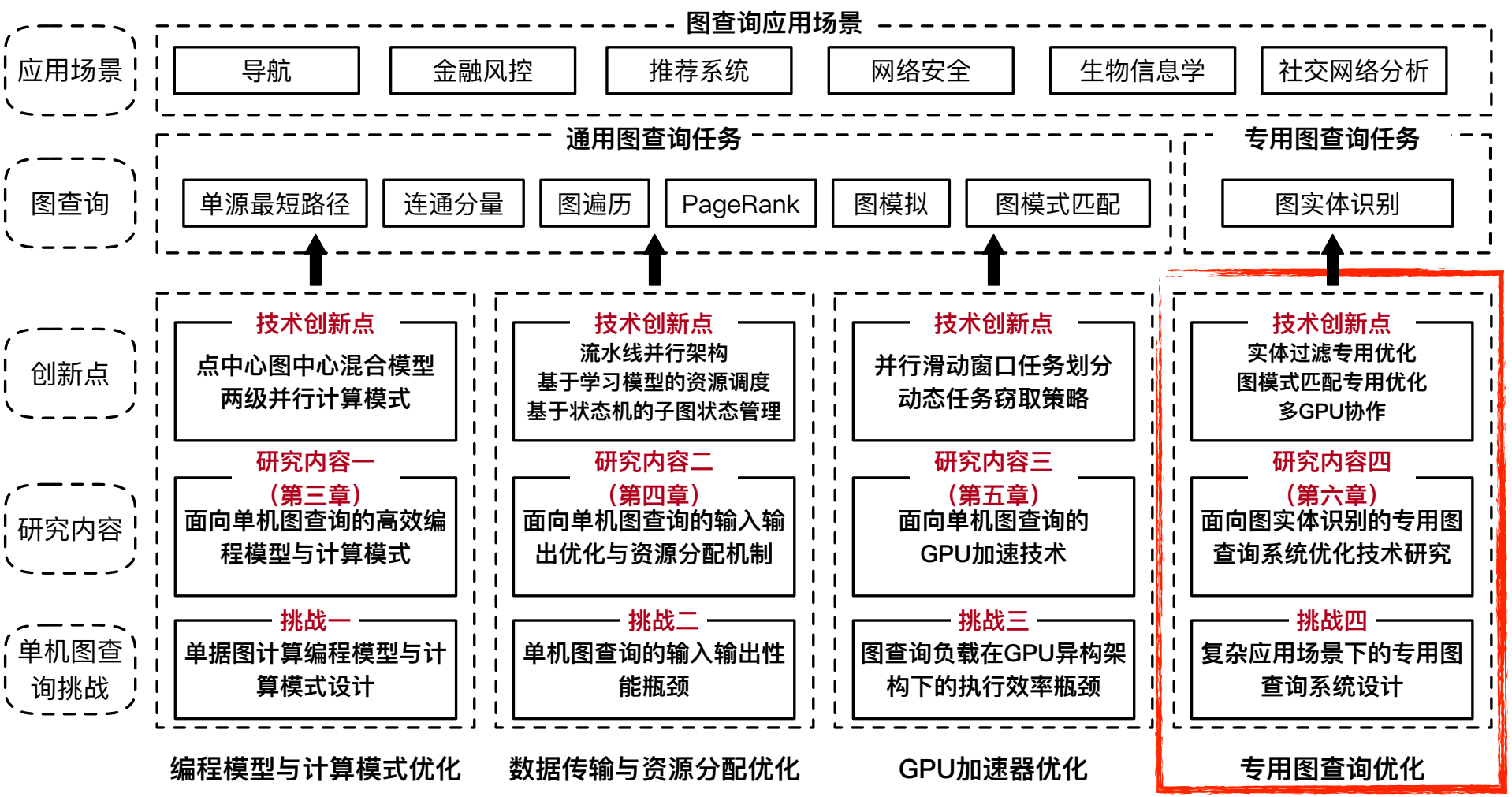


结果概览

- ✓ GraphRipple 比基线系统（ CGGraph, Subway）快最多 **15×，14.5x**
- ✓ 验证了动态任务窃取以及滑动窗口的有效性

主要研究工作四

单机大规模图数据查询系统优化关键技术研究



“大数据分析的头号问题”（*Veracity*, 真实性）

语义冲突与重复

✓ 真实性与可靠性方面的挑战

过时数据

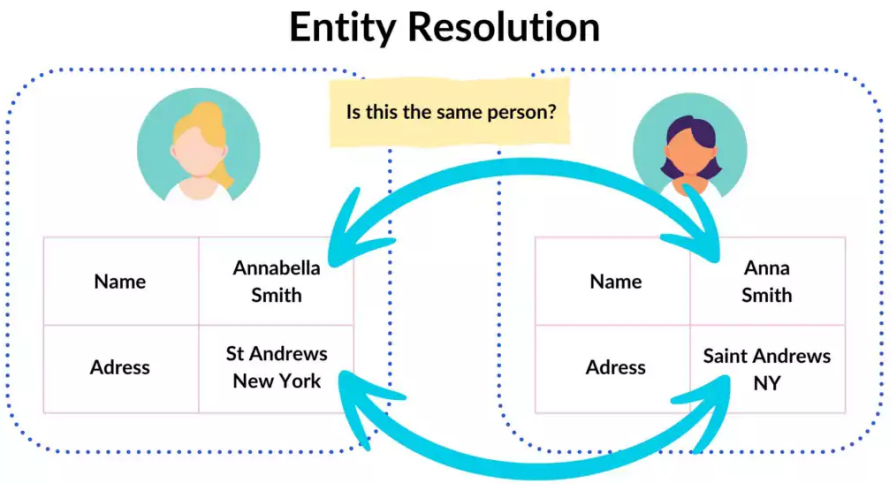
✓ 50%的存档记录在两年内变得过时

实体识别：发现数据中的实体关系

✓ 高效实体识别：数据清洗的核心驱动力

关键挑战之一：性能

好不好　　快不快



“大数据分析的头号问题”（*Veracity*, 真实性）

语义冲突与重复

✓ 真实性与可靠性方面的挑战

过时数据

✓ 50%的存档记录在两年内变得过时

实体识别：发现数据中的实体关系

✓ 高效实体识别：数据清洗的核心驱动力

关键挑战之一：性能

好不好

快不快

英国:8100万个国家保险号码，而背后对应符合条件的真实公民只有6000万

USA：数百万超过130岁的老人，其中一位甚至已达到360岁，他们都在领取社保

AHIMA American Health Information Management Association®

医疗机构中约有8%-12%的患者记录是重复的 （来源：AHIMA, American Health Information Management Association）

THOMSON REUTERS

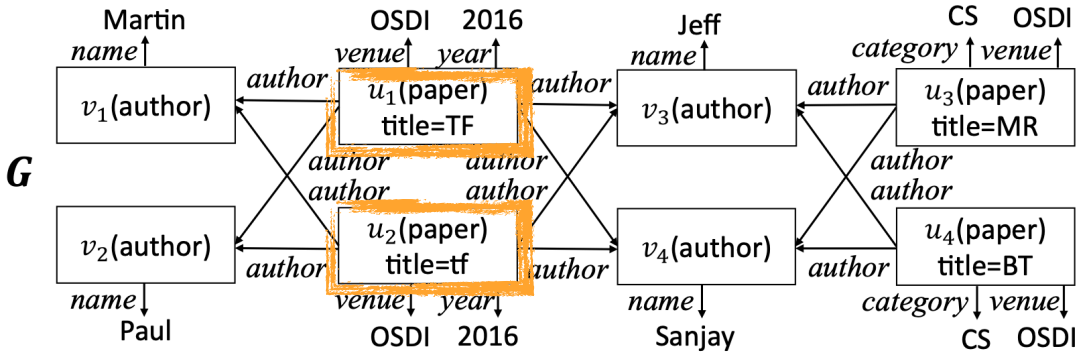
一个数据去重项目通常需要3~6个月才能完成（来源：Thomson Reuters）

本文重点关注图实体识别任务的效率问题

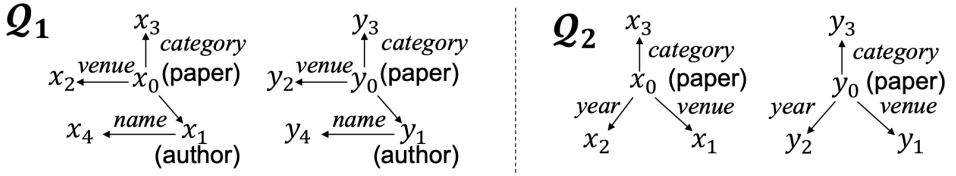
挑战：专用图查询系统优化难

图数据实体识别

目标图



图模式



匹配依赖

$$t.\text{venue} = s.\text{venue} \wedge t.\text{name} = s.\text{name} \wedge t.\text{author} = s.\text{author} \rightarrow s.id = t.id$$

$$t.\text{venue} = s.\text{venue} \wedge t.\text{category} = s.\text{category} \wedge t.\text{year} = s.\text{year} \rightarrow s.id = t.id$$

图实体识别规则

$$\mathcal{Q}[x_0, y_0](X \rightarrow p_0)$$

输入：

实体识别规则、数据图

输出：

属于同一真实世界实体的顶点对集合

$$\{v_a, v_b | (v_a, v_b) \text{ are the same real word entity}\}$$

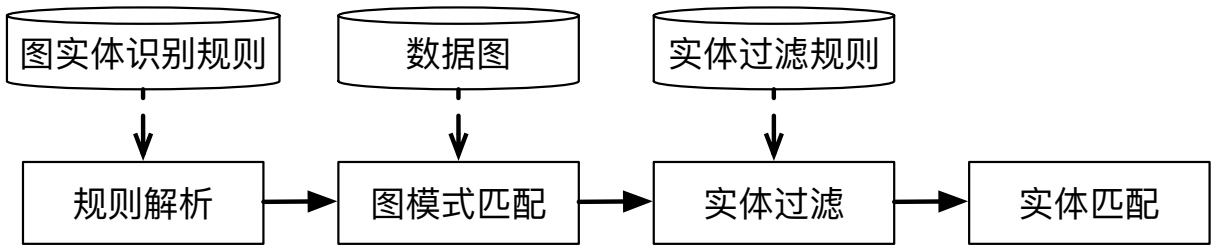
u_1 u_2 属于同一实体

挑战：专用图查询系统优化难

存在问题

- ✓ 图实体识别包含多个关联的阶段。每个阶段具有不同的优化目标、输入输出
- ✓ 数据异构性：包含拓扑结构，文本等异构数据

图数据实体识别流程

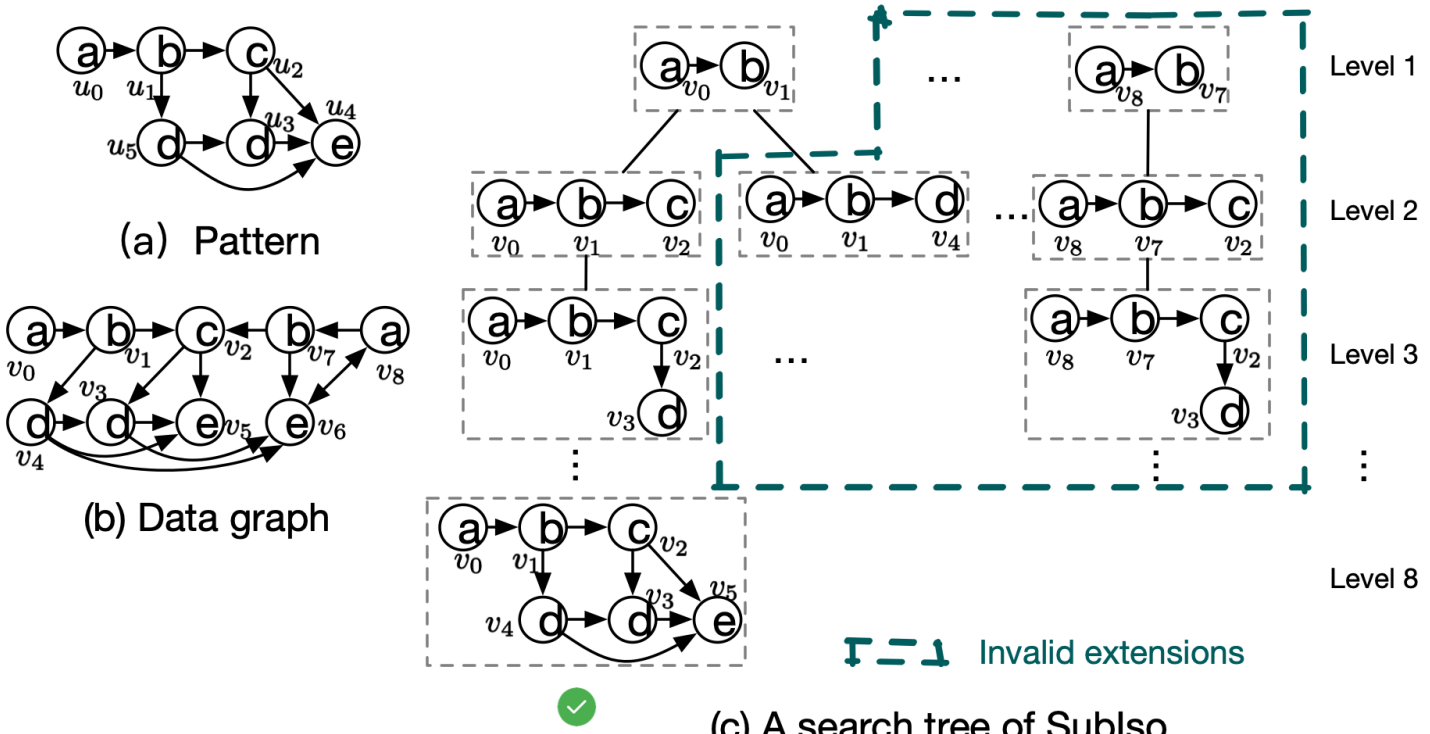


目标

针对图实体识别中各个阶段优化性能

图模式匹配：

✓ 图模式匹配是指在目标图中寻找与给定模式图结构一致的子图的过程

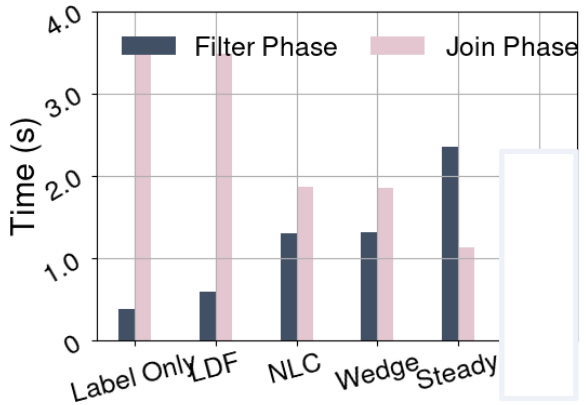


图模式匹配示例

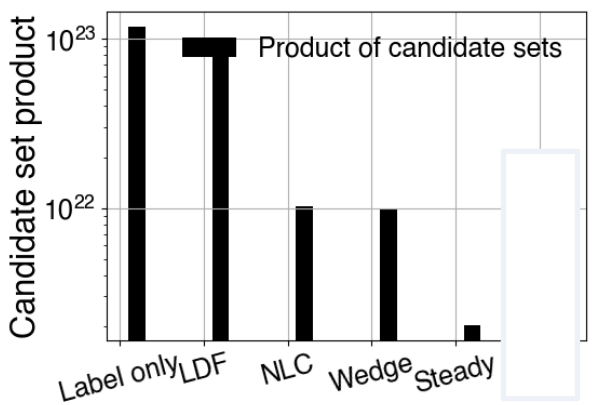
图模式匹配阶段挑战

评估*Twitter*上的剪枝能力与剪枝成本

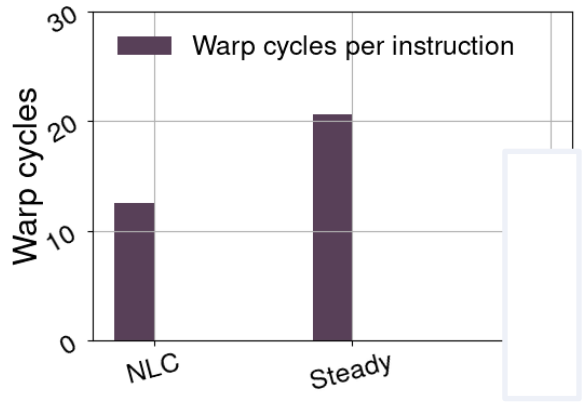
- ✓ 查询图：Q (|V|=4, |E|=3)
- ✓ 目标图：Twitter (|V|=1334392, |E|=46180342)
- ✓ 过滤方法：Label and degree filter (LDF, [SIGMOD'20]), Steady (DPIso[SIGMOD'19]), Neighbor-label (NLC, EGSM[SIGMOD'23]), Wedge (EGSM[SIGMOD'23])
- ✓ 过滤能力：Steady > Wedge > NLC > LDF > Label-only
- ✓ 过滤代价：Pruning cost: Label-only > LDF > NLC > Wedge > Steady



过滤方法对比：时间



过滤方法对比：解空间大小



过滤方法对比：GPU效率

本文方法在过滤能力与过滤代价的比值上表现最优

贡献点一：基于最小独立置换的过滤策略

观察：过滤规则

$$\phi_1 : \{L(u')|u' \in N(u)\} \nsubseteq \{L(v')|v' \in N(v)\} \rightarrow u \neq v$$

其中 $u \in P$, $v \in G$, $N(v)$ 表示 v 的邻居的集合, $L(v)$ 表示顶点 v 的标签

基于最小独立置换的过滤方法

$$Pr(\min\{\pi(\mathcal{A})\} > \min\{\pi(\mathcal{B})\}) > 0 \rightarrow \mathcal{B} \nsubseteq \mathcal{A}$$

$\mathcal{A} = \{L(v')|v' \in N(v)\}$, $\mathcal{B} = \{L(v')|v' \in N(u)\}$ 得 $u \neq v$

仅依赖最小值容易导致假阴性结果

基于 k-最小独立置换的过滤方法

$$Pr(\max\{min_k\{\pi(\mathcal{A})\}\backslash_k\{\pi(\mathcal{B})\}\} > \max\{min_k\{\pi(\mathcal{B})\}\backslash min_k\{\pi(\mathcal{A})\}) > 0 \rightarrow \mathcal{B} \nsubseteq \mathcal{A}$$

$$Pr(\min\{min_k\{\pi(\mathcal{B})\}\backslash min_k\{\pi(\mathcal{A})\} < \min\{min_k\{\pi(\mathcal{A})\}\backslash min_k\{\pi(\mathcal{B})\}\}) > 0 \rightarrow \mathcal{B} \nsubseteq \mathcal{A}$$

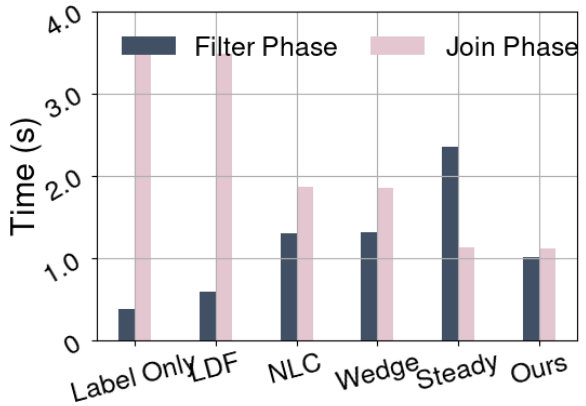
$$\mathcal{A} = \{L(v')|v' \in N(v)\}\text{，}\mathcal{B} = \{L(u')|N(u)\}\text{得 } u \neq v$$

采用最小的 k 个值，有助于减少假阴性结果的发生

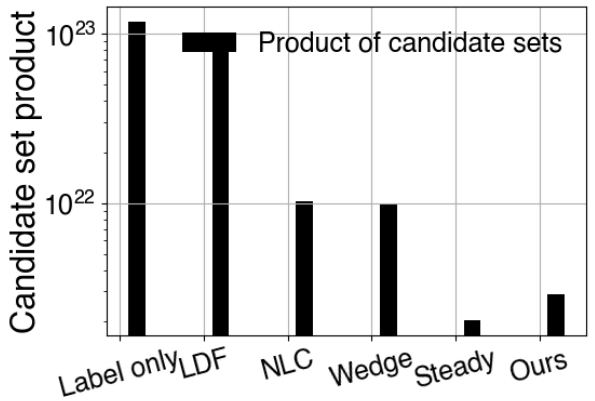
图模式匹配阶段实验结果

评估*Twitter*上的剪枝能力与剪枝成本

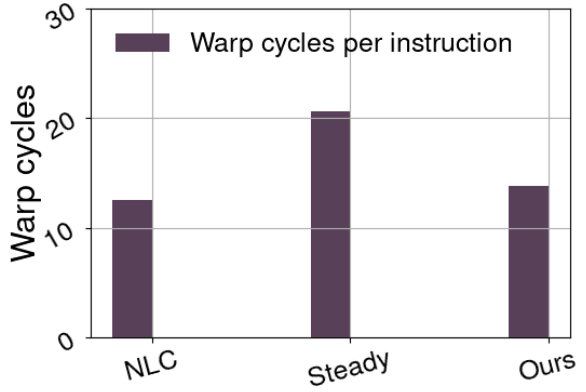
- ✓ 查询图：Q (|V|=4, |E|=3)
- ✓ 目标图：Twitter (|V|=1334392, |E|=46180342)
- ✓ 过滤方法：Label and degree filter (LDF, [SIGMOD'20]), Steady (DPIso[SIGMOD'19]), Neighbor-label (NLC, EGSM[SIGMOD'23]), Wedge (EGSM[SIGMOD'23])
- ✓ 过滤能力：Steady > Ours > Wedge > NLC > LDF > Label-only
- ✓ 过滤代价：Pruning cost: Label-only > LDF > Ours > NLC > Wedge > Steady



过滤方法对比：时间



过滤方法对比：解空间大小

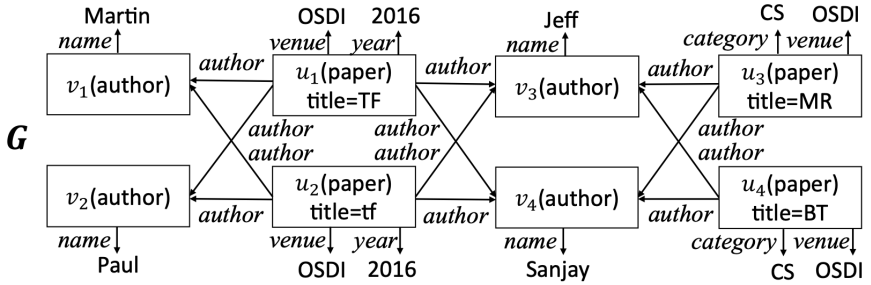


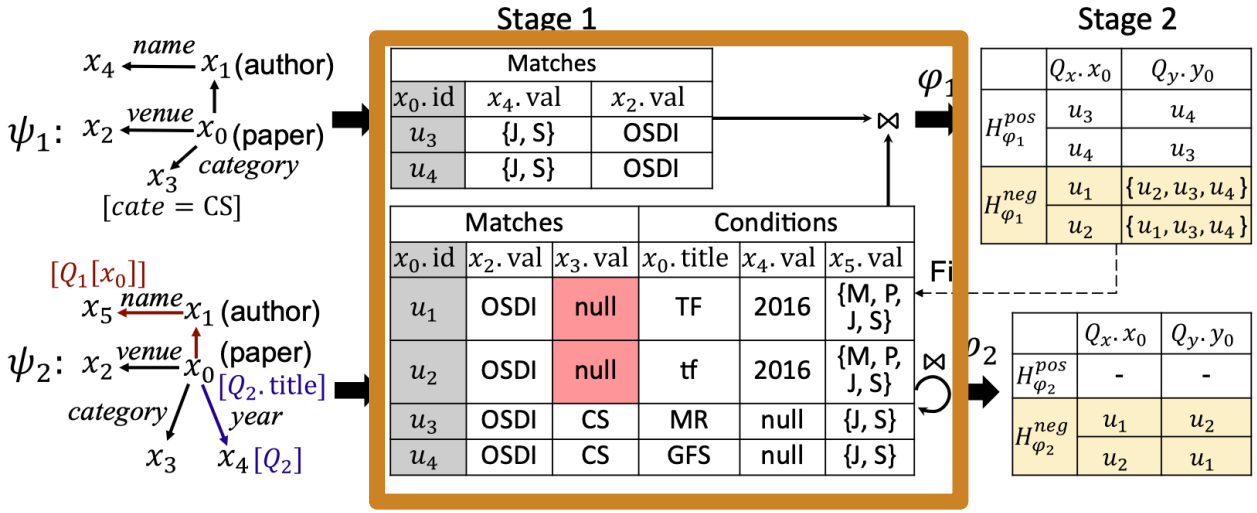
过滤方法对比：GPU效率

本文方法在过滤能力与过滤代价的比值上表现最优

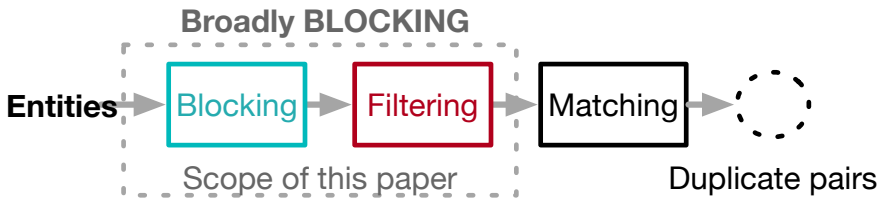
实体过滤阶段挑战

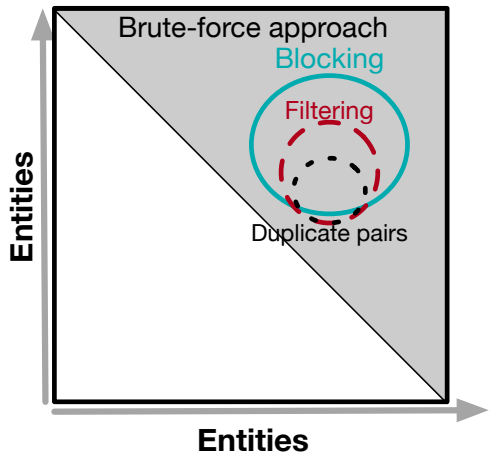
实验评估





图模式匹配阶段之后

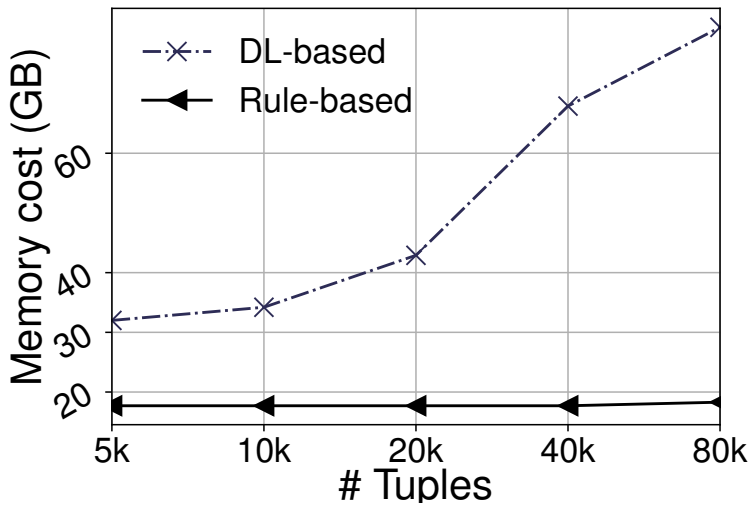




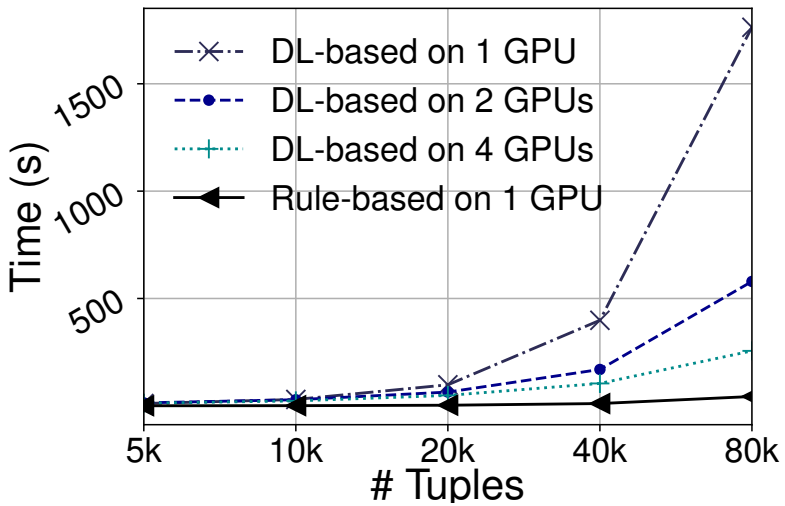
实体过滤阶段挑战

实验评估

- ✓ 目标图：DBLP (|V|=0.5M)
- ✓ 过滤方法：DeepBlocker[VLDB'21], 基于规则的简单实现的原型系统



基于规则的方法vs 基于学习的方法：空间开销



基于规则的方法vs 基于学习的方法：效率

鲜有基于规则的方法能充分利用*GPU*的大规模并行优势

贡献点二： 面向基于规则的实体过滤的执行计划优化

动机

- **谓词排序**：“描述（description）”字段提供更高的有效性和选择性，而“颜色（color）”字段的评估成本较低。那么，应该优先评估哪一个?
- 基于规则的实体过滤是以析取范式（DNF）形式组织的，即 φ_1 OR φ_2
- 实体过滤规则可能包含**共同的谓词**，因此可以复用计算结果
  - φ_1 $t.title \approx s.title \wedge t.author \approx s.author \wedge t.year = s.year \rightarrow t.id = s.id$
  - φ_2 $t.title \approx s.title \wedge t.author = s.author \wedge t.venue \approx s.venue \rightarrow t.id = s.id$

共同谓词

高辨识度，但计算代价高昂

计算效率高，但辨识度低

eid	pno	pname	price	sname	description	color	saddress
e_1	t_1	Apple Mac Air	$909	Comp. World	Apple MacBook Air (13-inch, 8GB RAM, 256GB SSD)	Gray	Barton Grove, McCulloughmouth
e_2	t_2	ThinkPad	-	Smith's Tech	ThinkPad E15, 15.6-inch full HD IPS display, Intel Core i5-1235U processor, (16GB) RAM | 512GB PCIe SSD)	Gray	Seg Plaza, Hua qiang North Road
e_2	t_3	ThinkPad	$849	Smith's Tech	Lenovo E15 Business ThinkPad, 15.6-inch full HD IPS display, 12 generation Intel Core i5, 16GB RAM, 512GB SSD	Gray	Seg Plaza, Hua qiang North Road
e_1	t_4	MacBook Air	$909	Comp. World	Apple 2022 MacBook Air M2 chip 13-inch,8 GB RAM,256 GB SSD storage gray	Gray	-
e_1	t_5	MacBook Air	$909	Comp. World	-	Gray	Barton Grove, McCulloughmouth

HyperBlocker架构

如何基于上述特性设计更高效的执行计划

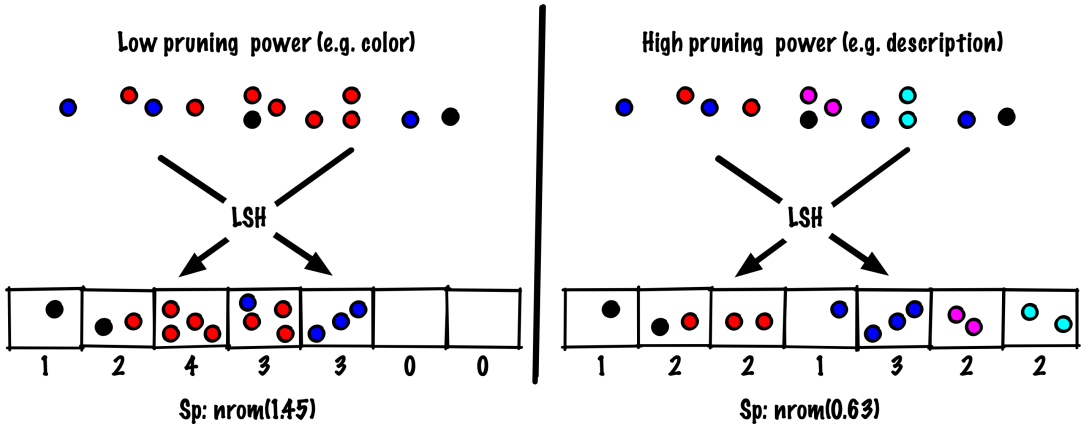
贡献点二： 面向基于规则的实体过滤的执行计划优化

输入：规则

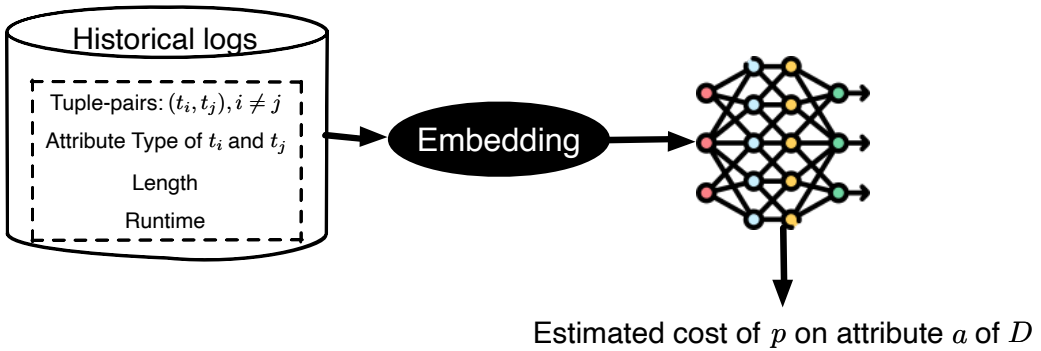
- $\varphi_1 : t.color = s.color \wedge t.price = s.price \wedge t.sname = s.sname \wedge t.pname \approx_{ED} s.pname \rightarrow t.eid = s.eid$
- $\varphi_2 : t.sname = s.sname \wedge t.description \approx_{JD} s.description \rightarrow t.eid = s.eid$
- $\varphi_3 : t.saddress \approx_{ED} s.saddress \wedge t.description \approx_{JD} s.description \rightarrow t.eid = s.eid$

输出：执行计划

- 步骤 1：对谓词和规则进行排序：**使用局部敏感哈希量化辨识度；** 使用**浅层模型和日志估算评估成本**



量化在数据 D 上满足谓词 p 的可能性



量化谓词 p 在数据集 D 上的执行效率

贡献点二： 面向基于规则的实体过滤的执行计划优化

输入：规则

- $\varphi_1 : t.color = s.color \wedge t.price = s.price \wedge t.sname = s.sname \wedge t.pname \approx_{ED} s.pname \rightarrow t.eid = s.eid$
- $\varphi_2 : t.sname = s.sname \wedge t.description \approx_{JD} s.description \rightarrow t.eid = s.eid$
- $\varphi_3 : t.saddress \approx_{ED} s.saddress \wedge t.description \approx_{JD} s.description \rightarrow t.eid = s.eid$

输出：执行计划

- 步骤 1：对谓词和规则进行排序：**使用局部敏感哈希量化辨识度；** 使用**浅层模型和日志估算评估成本**

$$\mathrm{sp}(p, D) = \mathrm{Norm}(\sqrt{\frac{1}{k}\sum_{i}^{k}(b_i - \frac{|D|}{k})^2})$$

量化在数据 D 上满足谓词 p 的可能性

$$\hat{\mathrm{cost}}_a(p, D) = \mathrm{Norm}(\sum_{(t_1,t_2)\in D\times D} \mathcal{N}(p, t_1, t_2))$$

量化谓词 p 在数据集 D 上的执行效率

$$\frac{1 - sp(p, D)}{\hat{\mathrm{cost}}_a(p, D)}$$

代价模型

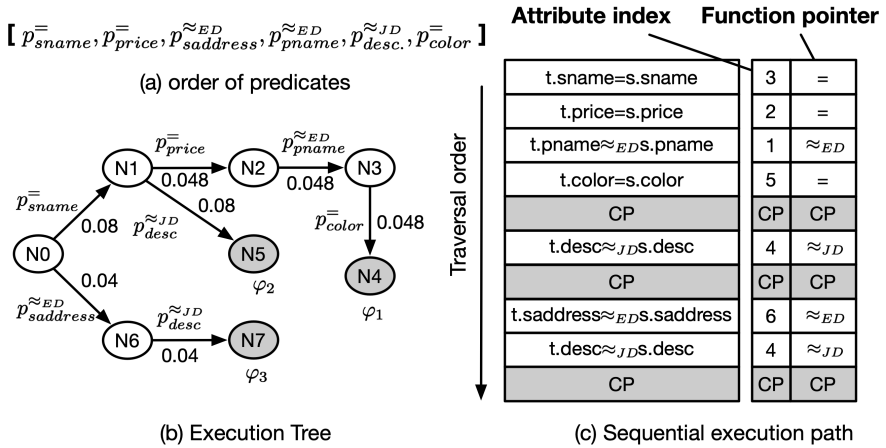
贡献点二： 面向基于规则的实体过滤的执行计划优化

输入：规则

- $\varphi_1 : t.color = s.color \wedge t.price = s.price \wedge t.sname = s.sname \wedge t.pname \approx_{ED} s.pname \rightarrow t.eid = s.eid$
- $\varphi_2 : t.sname = s.sname \wedge t.description \approx_{JD} s.description \rightarrow t.eid = s.eid$
- $\varphi_3 : t.saddress \approx_{ED} s.saddress \wedge t.description \approx_{JD} s.description \rightarrow t.eid = s.eid$

输出：执行计划

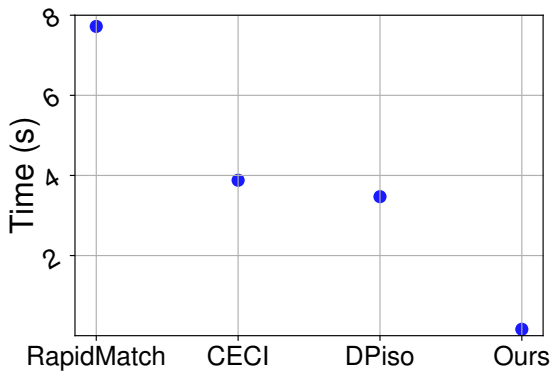
- 步骤 1：对谓词和规则进行排序：**使用局部敏感哈希量化辨识度；** 使用**浅层模型和日志估算评估成本**
- 步骤 2：构建执行树：复用共享前缀谓词的评估结果
- 步骤 3：序列化执行树：避免递归带来的线程计算差异



实验分析

实验总结

- ✓ HyperBlocker 性能优于现有实体过滤并行方案，比竞争对手平均快 **12.3 倍** 。
- ✓ 与 SOTA 实体匹配方法结合使用，可在保持准确率的同时节省至少 **30% 时间** 。
- ✓ 其执行计划优化器通过高效评估顺序和基于学习的 LSH 成本估算，使运行时间提升 **12.4 倍** ，并超越 PostgreSQL 的执行计划
- ✓ GPU 优化带来 **3.4 倍提升**，而缺少并行滑动窗口和任务窃取策略时，性能平均下降 **43.1% 和 28.8%**
- ✓ 异步流水线架构表现优异，较同步架构提速至少 **2.1 倍**



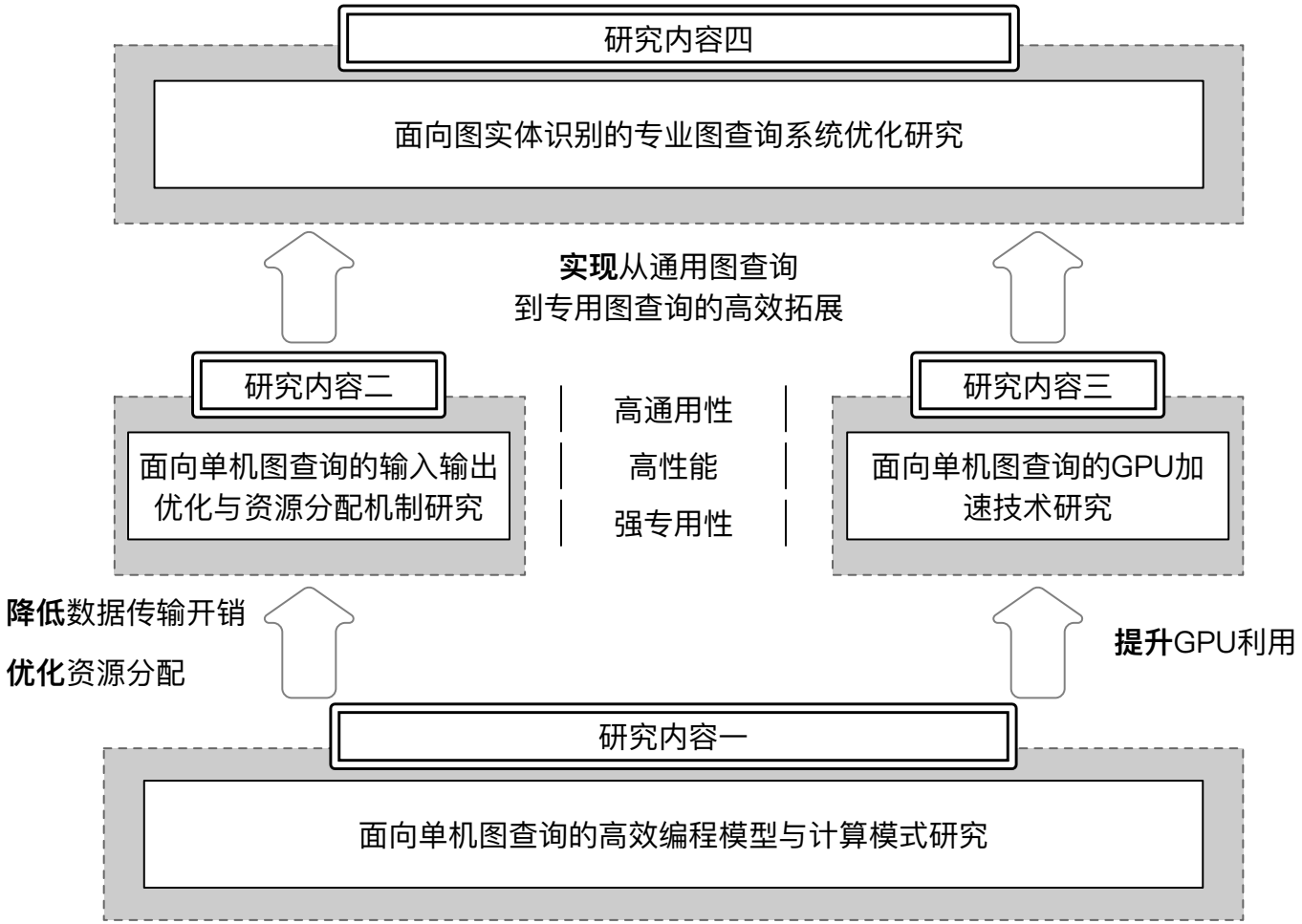
图模式匹配阶段性能

Method	Backend	Category	DBLP-ACM		IMDB		Songs		NCV	
			F1-score	Time (s)	F1-score	Time (s)	F1-score	Time (s)	F1-score	Time (s)
SparkER	CPU	Blocker	0.77 (-0.17)	11.0 (13.8×)	0.31 (-0.65)	242.9 (6.8×)	0.08 (-0.72)	203.4 (15.2×)	0.26 (-0.66)	229.3 (49.8×)
GPUDet	GPU	Blocker	0.92 (-0.02)	20.1 (25.1×)	0.94 (-0.02)	323.8 (9.1×)	0.80 (+0)	404.8 (30.2×)	0.90 (-0.02)	1252.6 (272.3×)
DeepBlocker	GPU	Blocker	0.98 (+0.04)	8.3 (10.4×)	/	>3h	/	>3h	/	>3h
HyperBlocker$_{noEPG}$	GPU	Blocker	0.94 (+0)	9.9 (12.4×)	/	>3h	0.80 (+0)	1904.1 (142×)	0.92 (+0)	2408.6 (523.6×)
HyperBlocker$_{noHO}$	GPU	Blocker	0.94 (+0)	9.5 (11.9×)	0.96 (+0)	472.6 (13.2×)	0.80 (+0)	45.0 (3.4×)	0.92 (+0)	35.9 (7.8×)
HyperBlocker	GPU	Blocker	0.94	0.8	0.96	35.7	0.80	13.4	0.92	4.6
Dedoop	CPU	Blocker+Matcher	0.90 (-0.08)	59.4 (9.4×)	0.67 (-0.29)	534.0 (15.0×)	0.80 (-0.08)	7643.4 (6.5×)	/	>3h
DisDedup	CPU	Blocker+Matcher	0.45 (-0.53)	94.0 (14.9×)	0.67 (-0.29)	644.0 (18.0×)	0.06 (-0.82)	917.0 (0.8×)	/	>3h
Ditto$_{top2}$	GPU	Blocker+Matcher	0.98 (+0)	9.0 (1.4×)	0.79 (-0.17)	6741.2 (188.8×)	0.88 (+0)	2308.6 (2.0×)	0.97 (+0.03)	381.8 (2.1×)
DeepBlocker$_{Ditto}$	GPU	Blocker+Matcher	0.99 (+0.01)	12.4 (2.0×)	/	>3h	/	>3h	/	>3h
HyperBlocker$_{Ditto}$	GPU	Blocker+Matcher	0.98	6.3	*0.96	*35.7	0.88	1179.0	0.94	180.6

整体性能

论文工作总结

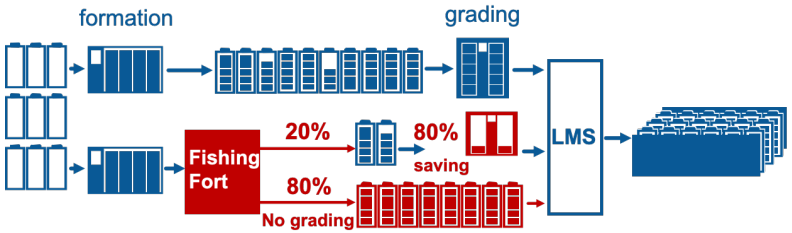
研究归纳:单机大规模图数据查询系统关键技术研究



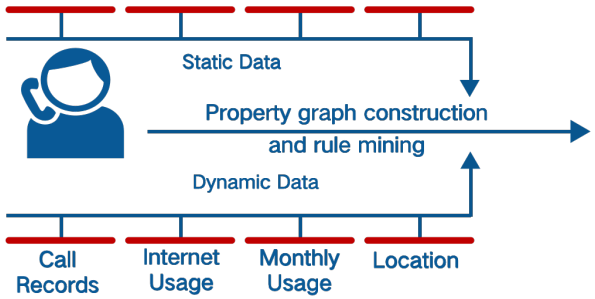
未来工作展望

未来工作展望

向更复杂的应用场景深度拓展

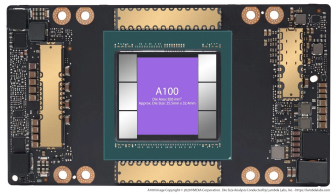


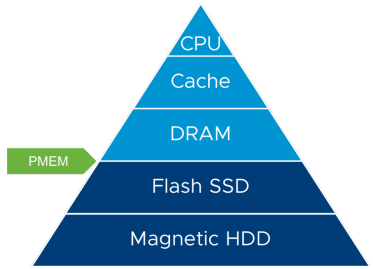
新能源

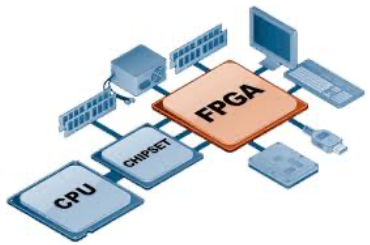


金融

新硬件





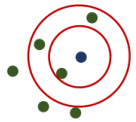


AI4 DB & Sys.

Filtering

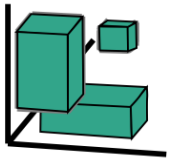
Query optimizing

Search



Resource scheduling

Index



...

主要学术成果

论著发表情况

近五年在 *CCF* 推荐国际会议和期刊上发表论文 *8* 篇

- ✓ 在CCF A 类会议上发表论文 **6** 篇，其中一作 **2** 篇，共同一作 **2** 篇
- ✓ 在其他 CCF 推荐国际会议和期刊上发表论文 **2** 篇，其中一作 **2** 篇

发明专利情况

近五年在申请发明专利 *4* 项

- ✓ 进入实质审查阶段发明专利 2 项
- ✓ 授权发明专利 2 项


Deep and Collective Entity Resolution in Parallel

ICDE 2021
(CCF A)


MiniGraph: Querying Big Graphs with a Single Machine

VLDB 2023
(CCF A)


HyperBlocker: Accelerating Rule-based Blocking in Entity Resolution using GPUs

VLDB 2025
(CCF A)


Rule-Based Graph Cleaning with GPUs on a Single Machine

SIGMOD 2025
(CCF A)

部分已发表论文

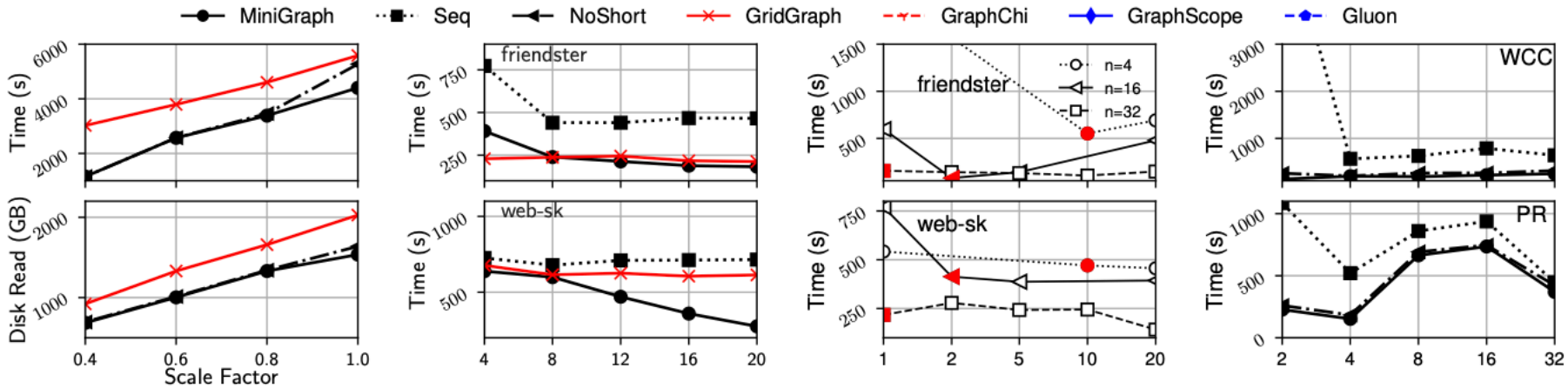
已授权专利

谢谢各位老师！敬请批评指正！

其他实验结果

Result: other results of MiniGraph

Scalability of MiniGraph



(a) Varying $|G|$: PR. (b) Varying m: WCC. (c) Varying p_i: PR. (d) Varying n: friendster.

Accuracy and effectiveness of cost model formulations

Cost model	$C_{\mathcal{A}}$	Model (a)	Model (b)
Normalized loss over S_{test}	0.16	0.22	0.22
Normalized loss over S'_{test}	0.40	0.50	0.43
Improvement web-sk (%)	39.0%	27.2%	27.3%
Improvement clueWeb (%)	30.0%	16.5%	17.1%

Result: Effectiveness of EPG

PostGreSQL的执行计划

```
SELECT * FROM DBLP, ACM
WHERE DBLP.authors ≈ ACM.authors
  AND DBLP.title ≈ ACM.title
  AND DBLP.year = ACM.year;
```

Query 1

```
SELECT * FROM DBLP, ACM
WHERE DBLP.title ≈ ACM.title
  AND DBLP.authors ≈ ACM.authors
  AND DBLP.year = ACM.year;
```

Query 2

```
SELECT * FROM DBLP, ACM
WHERE DBLP.venue ≈ ACM.venue
  AND DBLP.title ≈ ACM.title
  AND DBLP.author = ACM.author;
```

Query 3

```
SELECT * FROM DBLP, ACM
WHERE (DBLP.authors ≈ ACM.authors
  AND DBLP.title ≈ ACM.title
  AND DBLP.year = ACM.year)
OR (DBLP.venue ≈ ACM.venue
  AND DBLP.title ≈ ACM.title
  AND DBLP.authors = ACM.authors)
```

Query 4

发现

- ✓ Query2 性能优于 Query1（20%）源于 PostgreSQL 严格遵循 UDF (User Define Function) 声明顺序执行，未能基于计算代价优化执行计划
- ✓ Query4因OR运算符导致索引失效，被迫采用顺序扫描，其平均执行时间为Query2与Query3的12倍

Result: Effectiveness of EPG

Vs other execution plan

Baseline

- ✓ TupleX[SIGMOD'21]
- ✓ Query optimizer of PostgreSQL

Results

方法	Songs	DBLP-ACM	NCV	IMDB
PostgreSQL	40.6 (3.0×)	0.7 (2.3×)	13.5 (2.2×)	13036.0 (449.5×)
TupleX	14.2 (1.04×)	0.4 (1.3×)	12.4 (2.0×)	36.1 (1.2×)
EPG	13.6	0.3	6.2	29.0