On Smart Query Routing: For Distributed Graph Querying with Decoupled Storage

Arijit Khan

Nanyang Technological University (NTU), Singapore

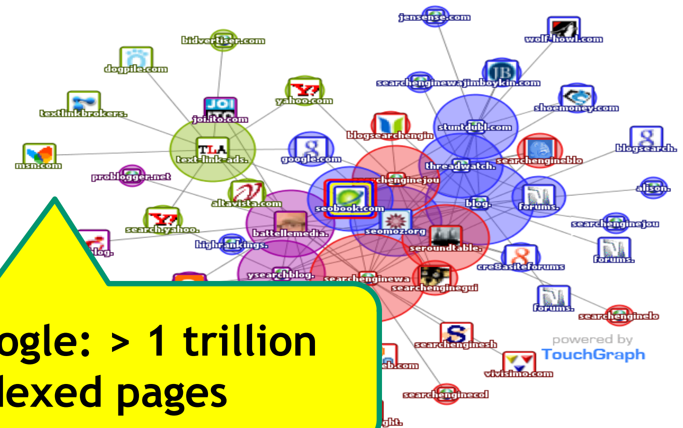
Gustavo Segovia

ETH Zurich, Switzerland

Donald Kossmann

Microsoft Research, Redmond, USA

Big Graphs

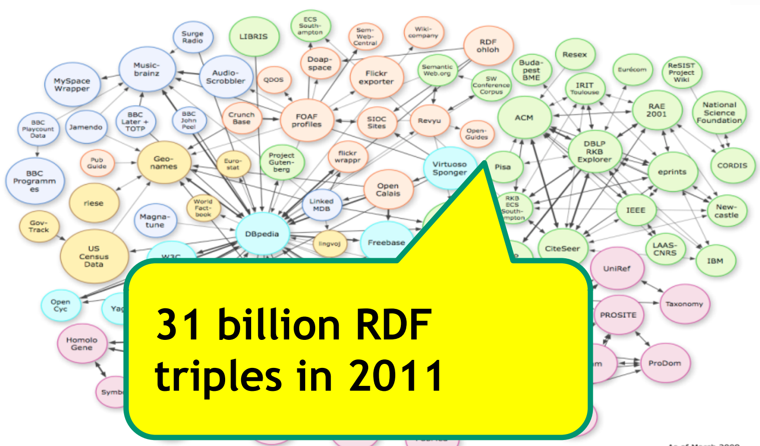


Google: > 1 trillion indexed pages

Web Graph

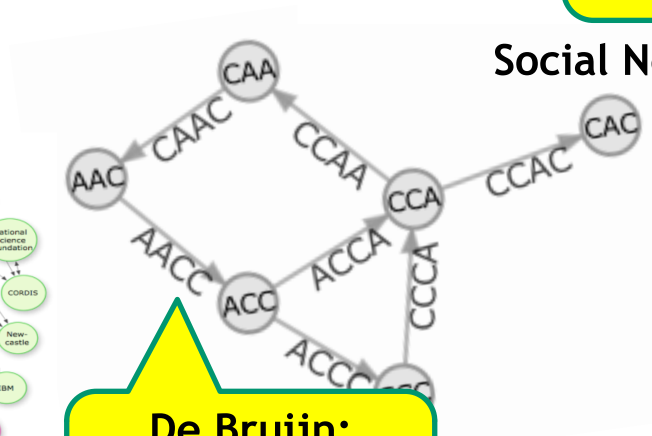
Facebook: > 800 million active users

Social Network



31 billion RDF triples in 2011

Information Network



De Bruijn:
4^k nodes
(k = 20, … , 40)

Biological Network

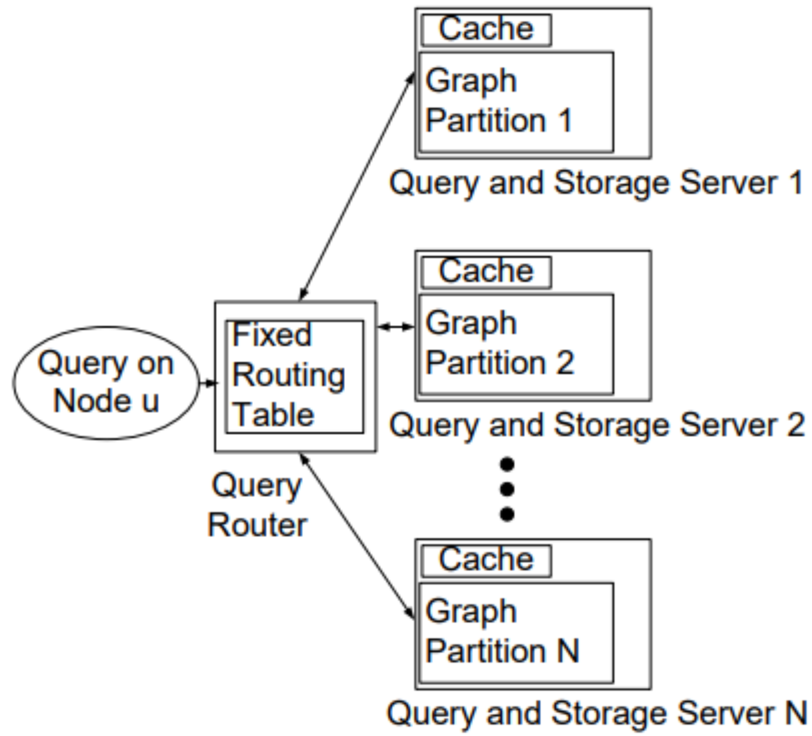
NETFLIX

100M Ratings,
480K Users,
17K Movies

Graphs in Machine Learning

Background: Distributed Graph Querying Systems

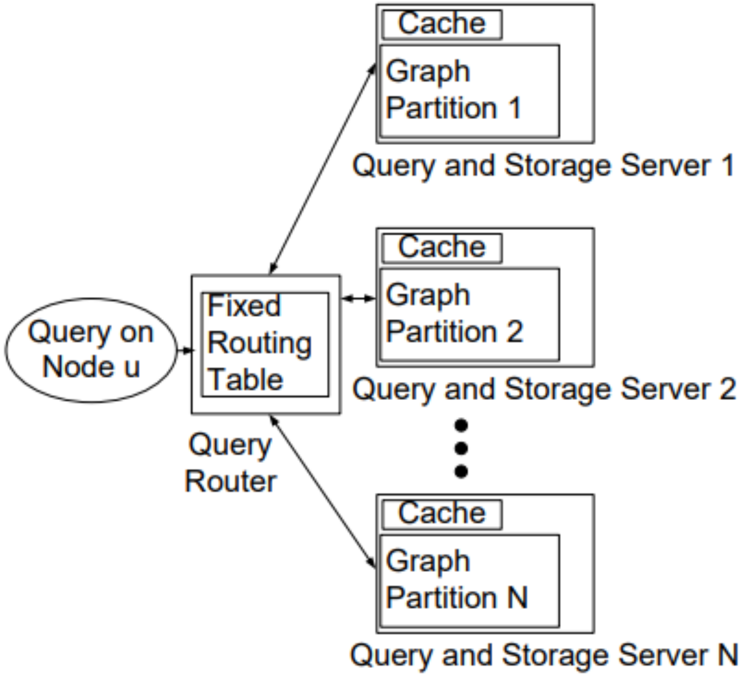
- First, partition the graph, and then place each partition on a separate server, where query answering over that partition takes place.



State-of-the-art distributed graph querying systems (e.g., SEDGE [SIGMOD'12], Trinity [SIGMOD'13], Horton [PVLDB'13])

Background: Distributed Graph Querying Systems

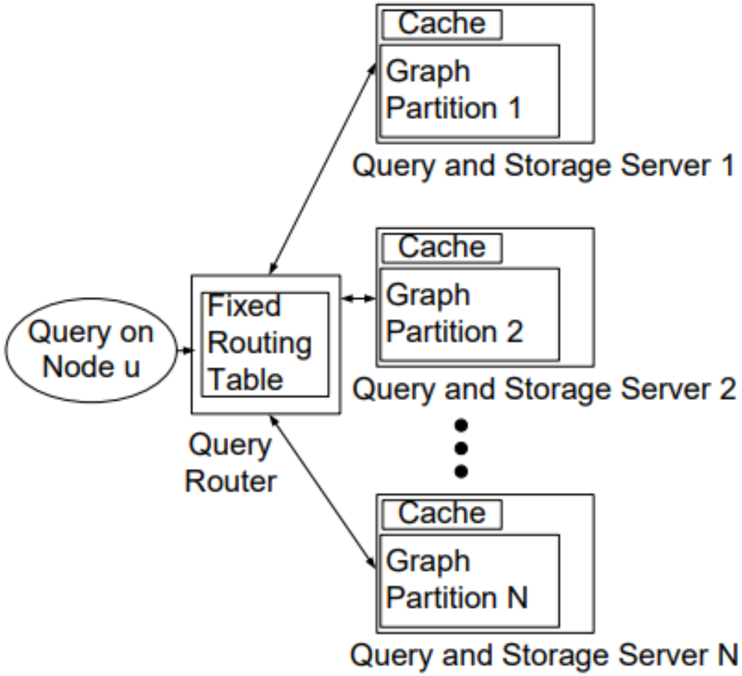
- **Disadvantages**
  - Fixed Routing (less flexible)
  - Balanced Graph Partitioning and Re-Partitioning



State-of-the-art distributed graph querying systems

Background: Distributed Graph Querying Systems

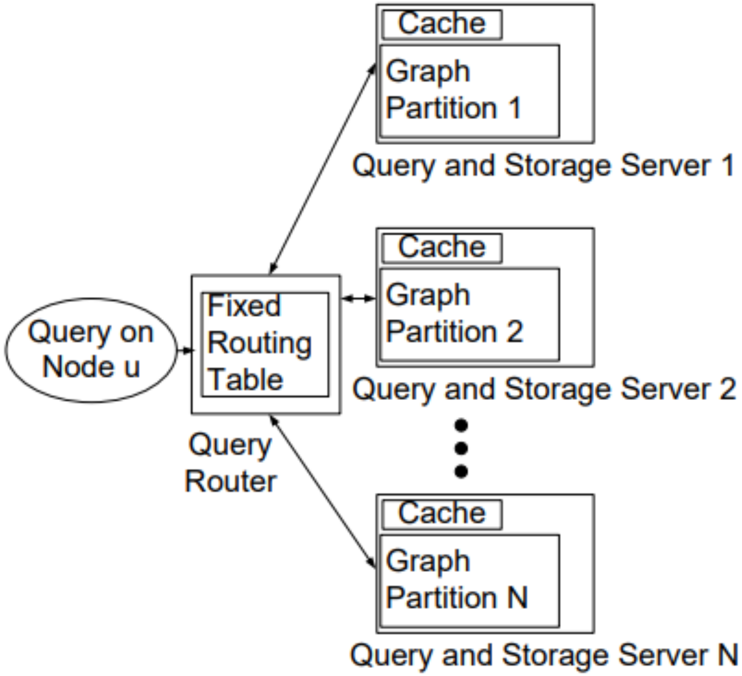
- **Disadvantages**
  - Fixed Routing (less flexible)
    - The server which contains the query node can only handle that request → the router maintains a **fixed routing table** (or, a fixed routing strategy, e.g., modulo hashing).
    - Less flexible with respect to **query routing** and **fault tolerance**, e.g., adding more machines will require updating the data partition and/or the routing table.
  - Balanced Graph Partitioning and Re-Partitioning



State-of-the-art distributed graph querying systems

Background: Distributed Graph Querying Systems

- **Disadvantages**
  - Fixed Routing (less flexible)
  - Balanced Graph Partitioning and Re-Partitioning
    - (1) workload balancing to maximize parallelism, (2) locality of data access to minimize network communication → **NP-hard, difficult in power-law graphs.**
    - later updates to graph structure or variations in query workloads → **graph re-partitioning/ replication** → online monitoring of workload changes, re-partitioning of the graph topology, and migration of graph data across servers are **expensive.**



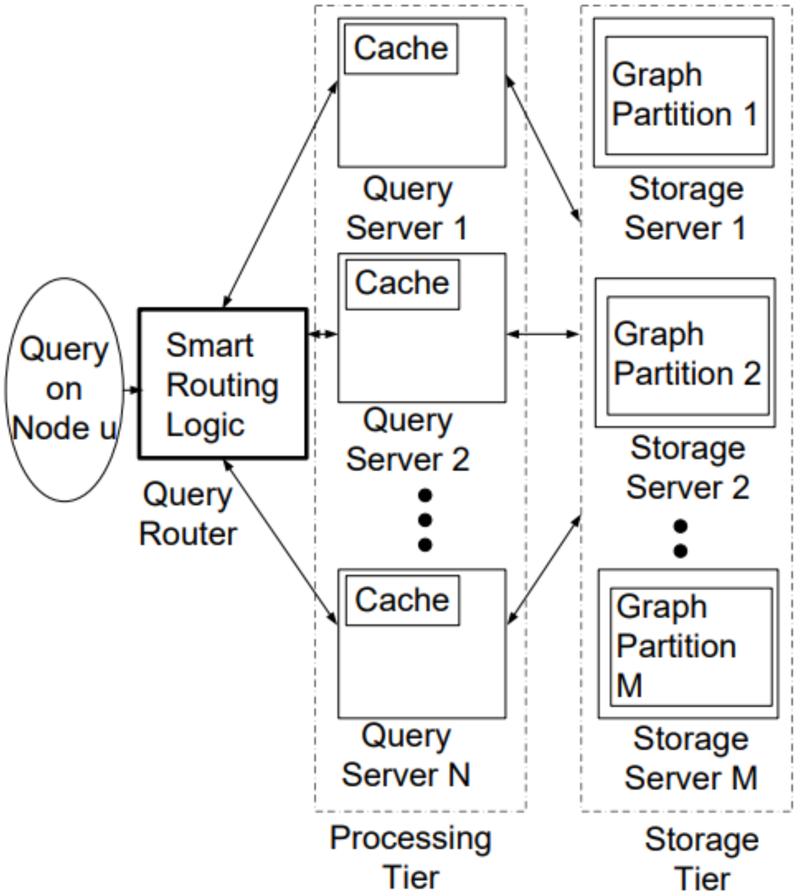
State-of-the-art distributed graph querying systems

Roadmap

- Distributed graph querying and graph partitioning
- **Decoupled graph querying system**
- Related work
- Smart graph query routing
- Experimental results
- Conclusions

Decoupled Graph Querying System

- we decouple query processing and graph storage into two separate tiers.
- This decoupling happens at a *logical* level.



Decoupled architecture for graph querying

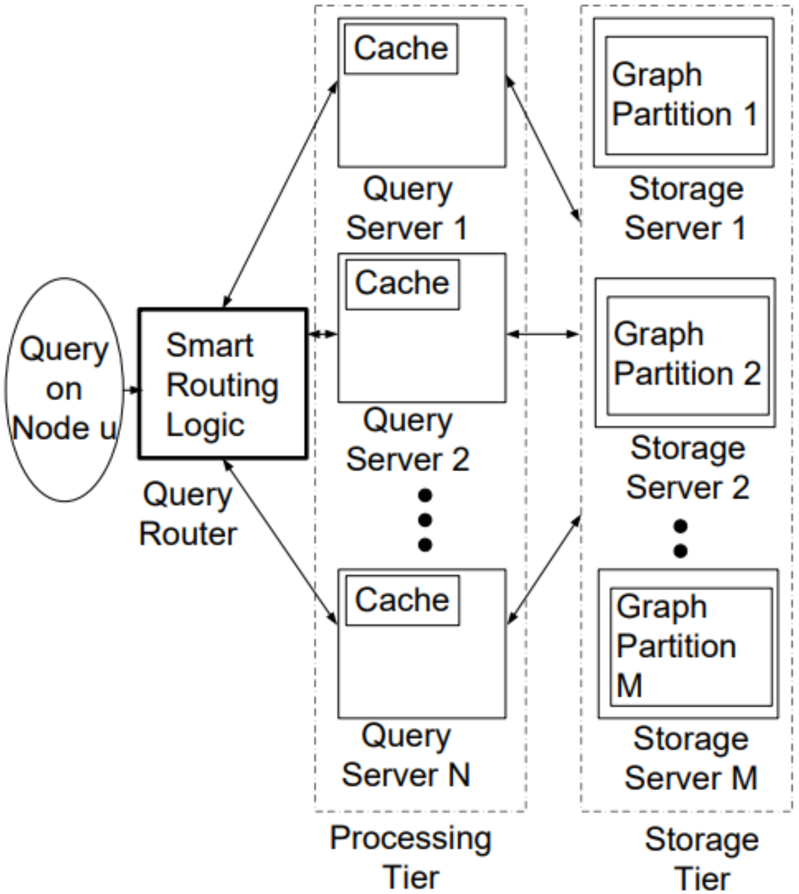
Decoupled Graph Querying System

- **Benefits**

  - Flexible routing

  - Less reliant on good partitioning across storage servers

    [Due to our **smart query routing** strategy – will be discussed soon!]



Decoupled architecture for graph querying

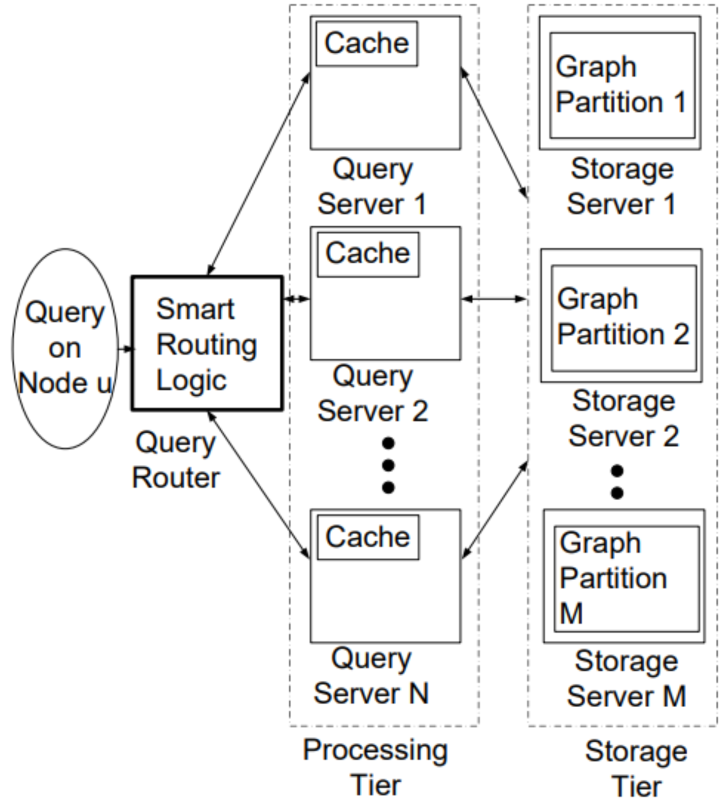
Decoupled Graph Querying System

- **Benefits**

➢ Flexible routing

- A query processor no longer assigned a fixed part of the graph → equally capable of handling any request → facilitating **load balancing** and **fault tolerance**.

- The query router can send a request to any of the query processors → more **flexible query routing**, e.g., more query processors can be added (or, a query processor that is down can be replaced) without affecting the routing strategy.



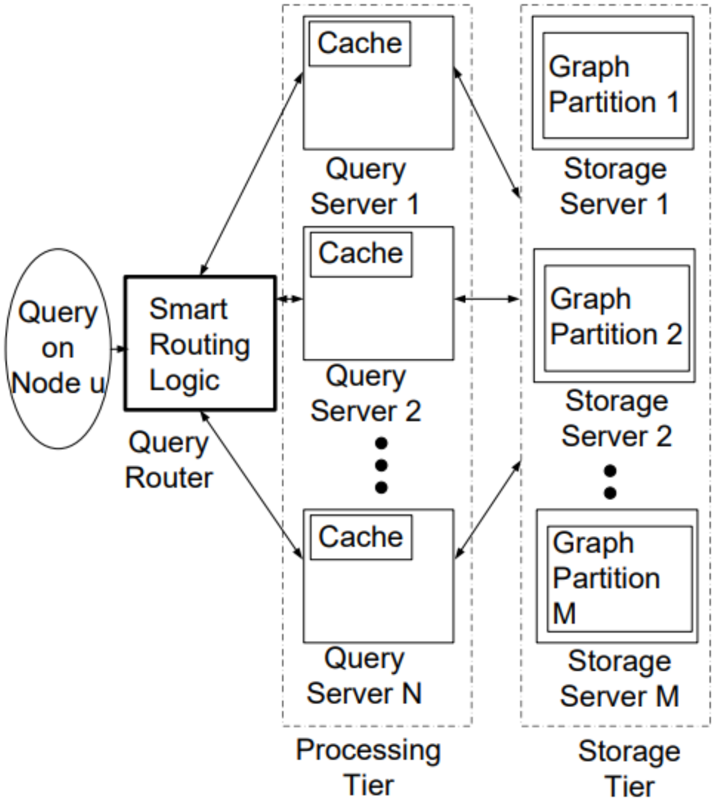
Decoupled architecture for graph querying

Decoupled Graph Querying System

- **Benefits**

➢ Flexible routing

- Each tier can be **scaled-up independently**.
- A certain workload is processing intensive → allocate more servers to the processing tier.
- Graph size increases over time → add more servers in the storage tier.
- Decoupled architecture, being **generic**, can be employed in many existing graph querying systems.



Decoupled architecture for graph querying

Roadmap

- Distributed graph querying and graph partitioning
- Decoupled graph querying system
- **Related work**
- Smart graph query routing
- Experimental results
- Conclusions

Related Work: Decoupling Storage and Query Processors

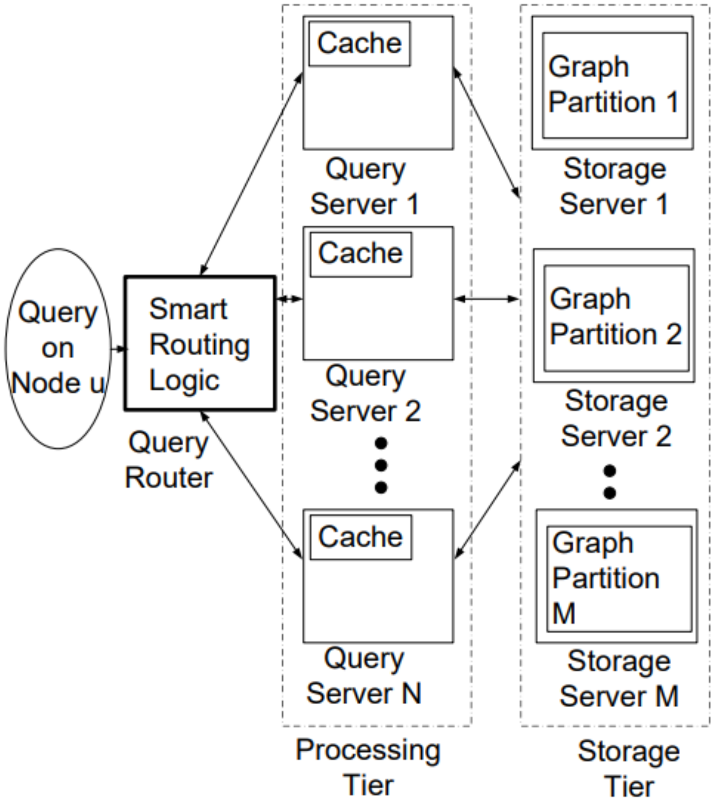
- Facebook's *Memcached* [NSDI'13]
- Google's *F1* [PVLDB'13]
- *ScaleDB* [http://scaledb.com/pdfs/TechnicalOverview.pdf]
- Loesing et. al. (*On the Design and Scalability of Distributed Shared-Data Databases*) [SIGMOD'15]
- Binnig et. al. (*The End of Slow Networks: It's Time for a Redesign*) [PVLDB'16]
- Shalita et. al. (*Social Hash: An Assignment Framework for Optimizing Distributed Systems Operations on Social Networks*) [NSDI'16]

Decoupled Graph Querying System

- **Disadvantages**

➢ Query processors may need to communicate with the storage tier via the network → **additional penalty to the response time** for answering a query.

➢ May cause **high contention rates** on either the network, storage tier, or both.



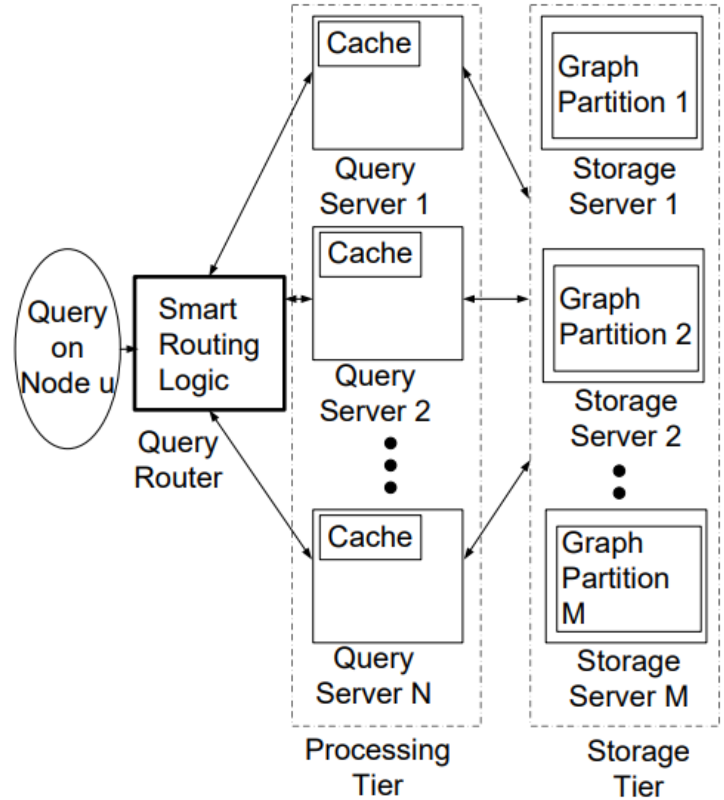
Decoupled architecture for graph querying

Our Contribution: Smart Query Routing

We design a **smart query routing logic** to utilize the cache of query processors over such decoupled architecture.

More cache hits → **reduce communication** among query processors and storage servers.

More cache hits → **less reliant on good partitioning** across storage servers.



Decoupled architecture for graph querying

Roadmap

- Distributed graph querying and graph partitioning
- Decoupled graph querying system
- Related work
- **Smart graph query routing**
- Experimental results
- Conclusions

h-Hop Traversal Queries

- Online, *h*-hop queries: explore a small region of the entire graph, and require fast response time.

- Start with a query node, and traverse its neighboring nodes up to a certain number of hops (i.e., $h = 2, 3$).

Examples

- *h*-hop neighbor aggregation

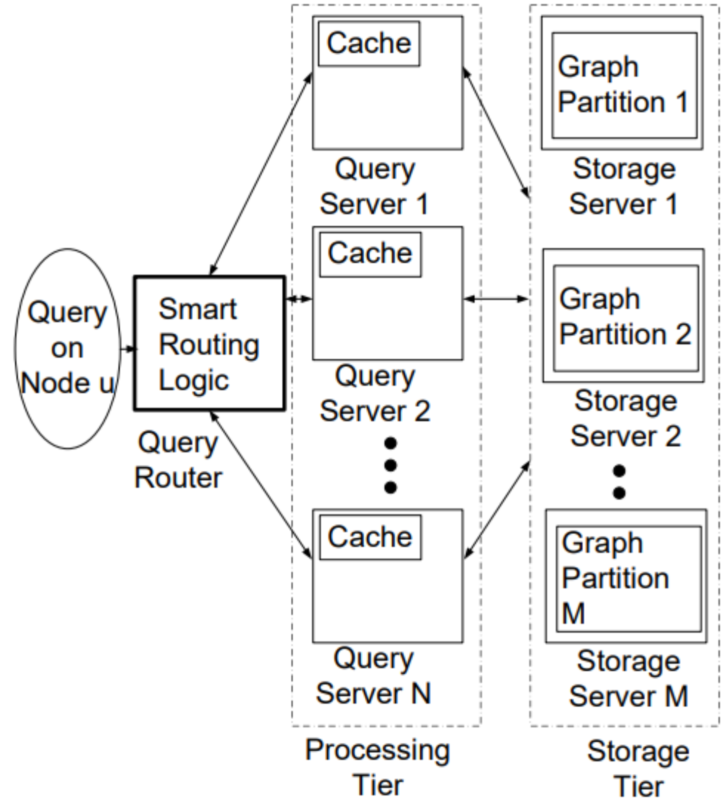
- *h*-step random walk with restart

- *h*-hop reachability

- More complex queries, e.g., node labeling and classification, expert finding, ranking, discovering functional modules, complexes, and pathways

Objectives for Smart Query Routing

- Leverage each processor's cached data
- Balance workload even if skewed or contains hotspot
- Make fast routing decisions
  [a small constant time, or << O(n)]
- Have low storage overhead in the router
  [a small fraction of the input graph size]



Decoupled architecture for graph querying

On Smart Query Routing: For Distributed Graph Querying with Decoupled Storage:
Arijit Khan (NTU Singapore), G. Segovia, and D. Kossmann

Challenges in Smart Query Routing

- Objectives are **conflicting**

  - For **maximum cache locality**, router can send all queries to the same processor (assuming no cache eviction) → **imbalanced workload** in processors → **lower throughput.**

  - router could inspect the cache of each processor before making a good routing decision → network delay. Hence, **router must infer what is likely to be in each processor's cache.**

Smart Routing Objectives

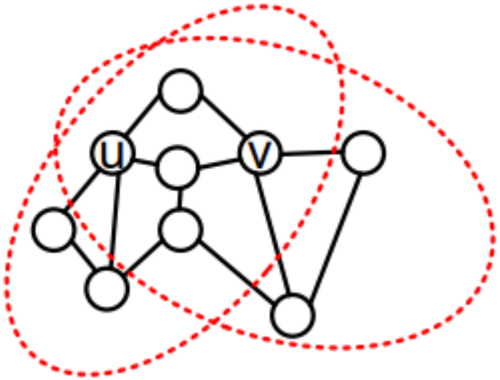
- Leverage each processor's cached data
- Balance workload even if skewed or contains hotspot
- Make fast routing decisions
- Have low storage overhead in the router

Challenges in Smart Query Routing

Smart Routing Objectives are conflicting!

- **Topology-Aware Locality**
  - **successive queries on nearby nodes must be sent to the same processor**. It is likely that *h*-hop neighborhoods of these nodes significantly overlap.
  - How the router knows about nearby nodes without storing the entire graph topology?

    - use **landmark, graph embedding**



2-hop neighborhoods of *u* and *v* overlap significantly

Challenges in Smart Query Routing

Smart Routing Objectives are conflicting!

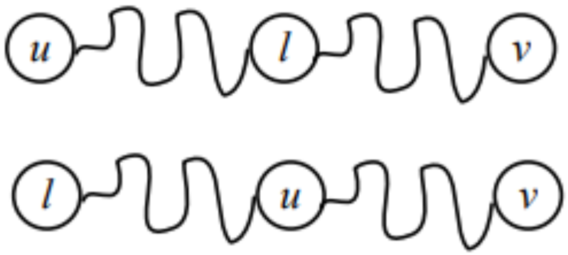
- **Query Stealing**

  - Always Routing queries to processors that have the most useful cache data → workload imbalance if skew/ query hotspot → **lower throughput.**

  - We perform **query stealing at router** → Whenever a processor is idle and is ready to handle a new query, if it does not have any other requests assigned to it, the router may "steal" a request and send to it which was intended for another processor.

  - **Query stilling by maintaining topology-aware locality** (as much as possible).

Smart Routing-1: Landmark



$$d(u, v) \leq d(u, l) + d(l, v)$$

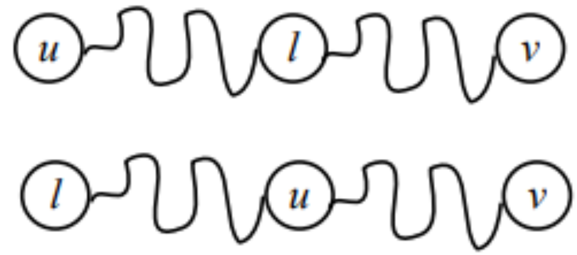
$$d(u, v) \geq |d(u, l) - d(l, v)|$$

If two nodes are close to a given landmark, they are likely to be close themselves.

Smart Routing-1: Landmark

Pre-processing

- Select a small set of L nodes as landmarks.
- Compute distance of every node to landmarks.
- **Assign landmarks to query processors**: Every processor is assigned a "pivot" landmark with the intent that pivot **landmarks are as far from each other as possible**. Each remaining landmark is assigned to the processor which contains its closest pivot landmark.



$$d(u,v) \leq d(u,l) + d(l,v)$$

$$d(u,v) \geq |d(u,l) - d(l,v)|$$

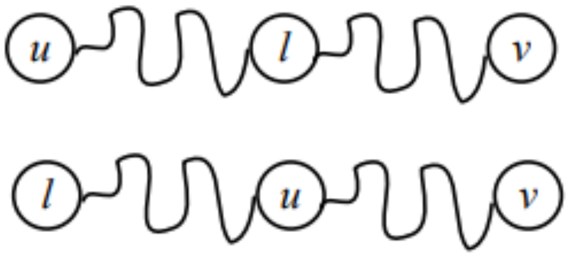
If two nodes are close to a given landmark, they are likely to be close themselves.

- **The distance of a node u to a processor p is defined as the minimum distance of u to any landmark that is assigned to processor p.**
- This distance information is stored in the router, which requires *O(nP)* space and *O(nL)* time to compute.

Smart Routing-1: Landmark

Online Routing

- a query on node *u* → the router verifies the pre-computed distance *d(u, p)* for every processor *p* → selects the one with the smallest *d(u, p)* value.
- Routing decision time: *O(p)*



$$d(u,v) \le d(u,l) + d(l,v)$$

$$d(u,v) \ge |d(u,l) - d(l,v)|$$

If two nodes are close to a given landmark, they are likely to be close themselves.

Load-balancing via Query-stealing

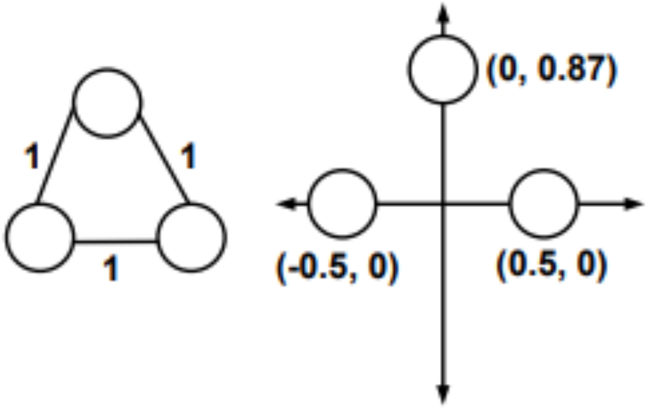
$$d^{LB}(u,p) = d(u,p) + \frac{\text{Processor Load}}{\text{Load Factor}}$$

- Route to smallest load-balanced distance.
- Nearby nodes are routed in similar way, maintaining topology-aware locality.

Smart Routing-2: Embed

Pre-processing

- Embed a graph in a lower D-dimensional Euclidean plane.
- The hop-count distance between graph nodes are approximately preserved via their Euclidean distance.
- Storage = $O(nD)$, time = $O(|L|^2 D + n|L|D)$



Graph embedding in 2D Euclidean plane

A benefit of embed routing is that the pre-processing is independent of the system topology, allowing more processors to be easily added at a later time.

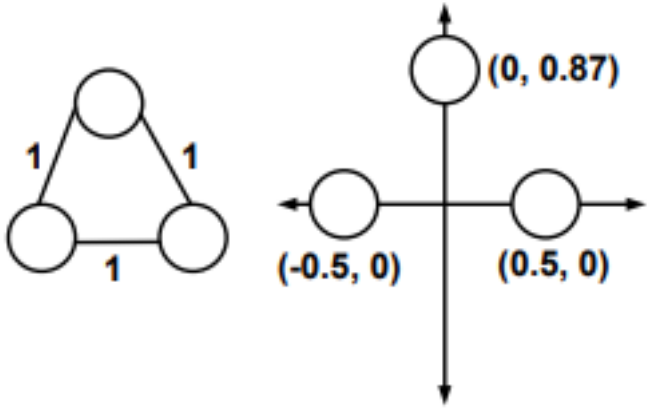
Smart Routing-2: Embed

Online Routing

- **Exponential moving average** to compute the mean of the processor's cache contents.

$$\text{MeanCo-ordinates}(p) = \alpha \cdot \text{MeanCo-ordinates}(p) + (1 - \alpha) \cdot \text{Co-ordinates}(v)$$

- Router finds the distance between a query node *u* and a processor *p*, denoted as *d(u, p)*, and defined as the distance of the query node's co-ordinates to the historical mean of the processor's cache contents.
- Route query on *u* to processor *p* with minimum *d(u, p)*.
- Routing decision time: *O(PD)*



Graph embedding in 2D Euclidean plane

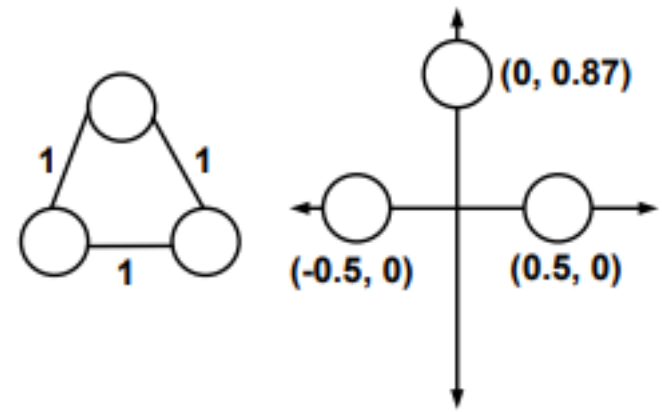
Smart Routing-2: Embed

Online Routing

- Exponential moving average to compute the mean of the processor's cache contents.

$$\text{MeanCo-ordinates}(p) = \alpha \cdot \text{MeanCo-ordinates}(p) + (1-\alpha) \cdot \text{Co-ordinates}(v)$$

- Router finds the distance between a query node *u* and a processor *p*, denoted as *d(u, p)*, and defined as the distance of the query node's co-ordinates to the historical mean of the processor's cache contents.
- Route query on *u* to processor *p* with minimum *d(u, p)*.
- Routing decision time: *O(PD)*



Graph embedding in 2D Euclidean plane

Load-balancing via Query-stealing

$$d^{LB}(u,p) = d(u,p) + \frac{\text{Processor Load}}{\text{Load Factor}}$$

Roadmap

- Distributed graph querying and graph partitioning
- Decoupled graph querying system
- Related work
- Smart graph query routing
- **Experimental results**
- Conclusions

Experimental Setup

Graph Datasets

Dataset	# Nodes	# Edges	Size on Disk (Adj. List File)
WebGraph	105 896 555	3 738 733 648	60.3 GB
Friendster	65 608 366	1 806 067 135	33.5 GB
Memetracker	96 608 034	418 237 269	8.2 GB
Freebase	49 731 389	46 708 421	1.3 GB

Cluster Configuration

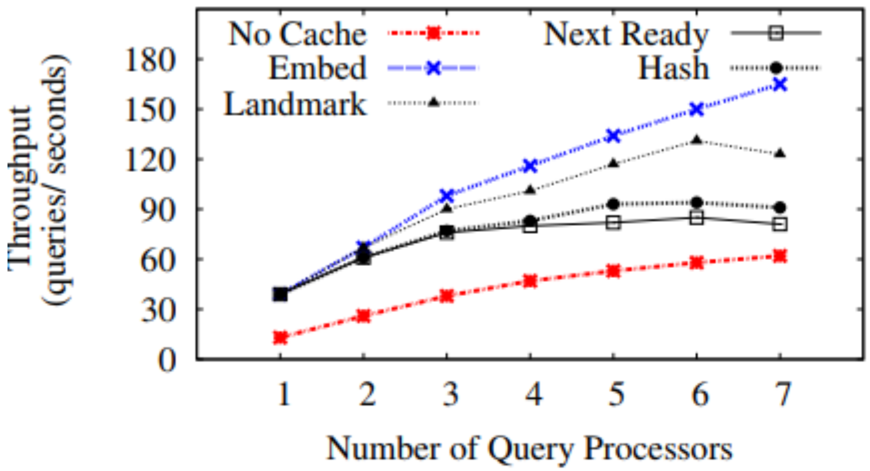
- 12 servers each having 2.4 GHz Intel Xeon processors, 0 – 4GB cache.
- interconnected by 40 Gbps Infiniband, and also by 10 Gbps Ethernet.
- Use a single core of each server with the following configuration: 1 server as router, 7 servers in the processing tier, 4 servers in the storage tier; and communication over Infiniband with remote direct memory access (RDMA).
- RAMCloud as storage tier.
- Graph is stored as adjacency list – every node-id is key, and the corresponding value is an array of its 1-hop neighbors.
- The graph is partitioned across storage servers via RAMCloud's default and inexpensive hash partitioning scheme, MurmurHash3 over graph nodes.

List of Experiments

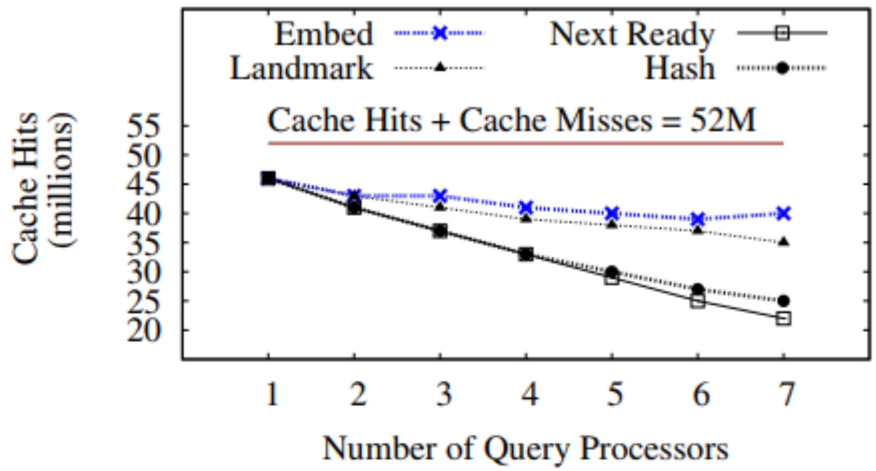
- Comparison with distributed graph systems (SEDGE [SIGMOD'12] with Giraph [SIGMOD'10], GraphLab [VLDB'12]) that use smart graph partitioning and re-portioning - *Our method achieves up to an order of magnitude higher throughput even with inexpensive hash partitioning of the graph!*
- **Scalability with number of processors** and storage servers
- **Impact of cache size**
- Impact of graph updates
- Sensitivity w.r.t. different parameters: query locality and hotspot, h-hop queries, load factor, smoothing parameter, embedding dimensionality, landmark numbers, minimum distance between a pair of landmarks

Performance Metrics	Query efficiency, Query throughput, Cache hit rates
Baseline Routing Methods	Next ready, No cache, Modular hash with query stealing

Performance with Varying Number of Query Processors



(a) Throughput

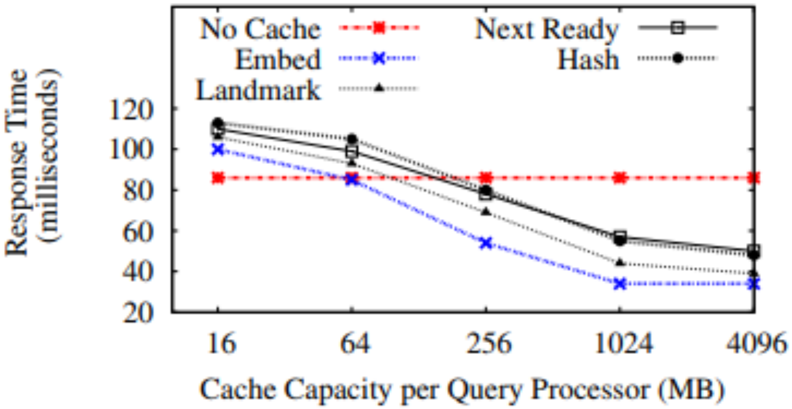


(b) Cache Hits

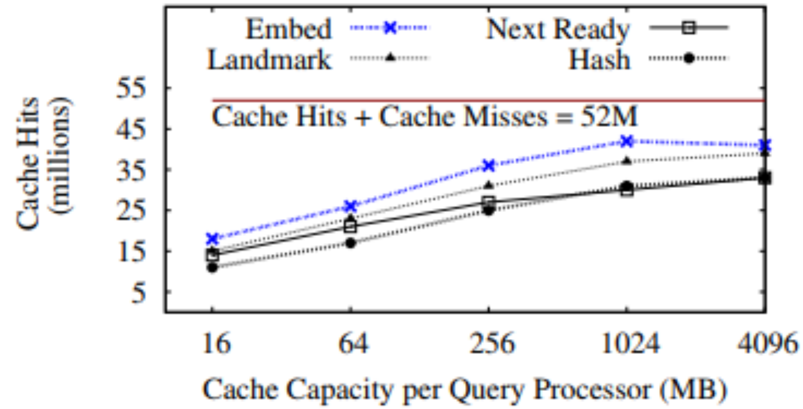
Embed routing is able to sustain almost same cache hit rate with many query processors. Hence, its throughput scales linearly with query processors.

On Smart Query Routing: For Distributed Graph Querying with Decoupled Storage: **Arijit Khan (NTU Singapore)**, G. Segovia, and D. Kossmann

Impact of Cache Sizes

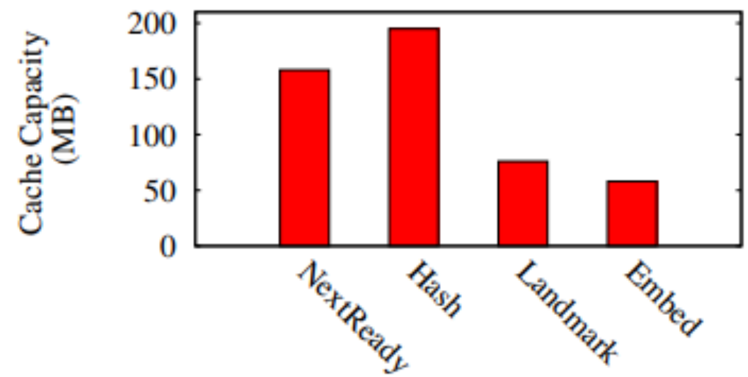


(a) Response time



(b) Cache hits

Both smart routings – Embed and Landmark – utilize the cache well; and for the same amount of cache, they achieve lower response time compared to baseline routings.



(c) Min. cache required to reach No-Cache's response time

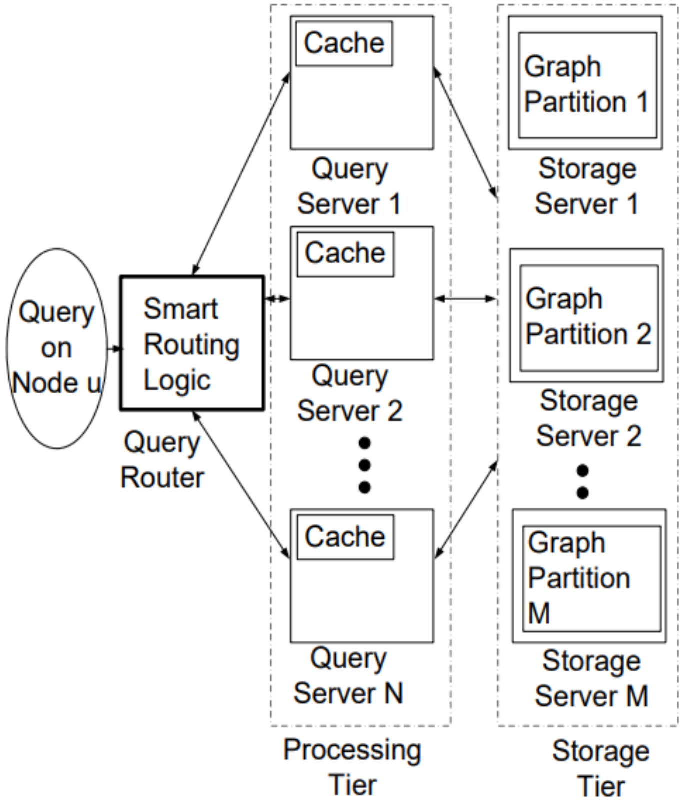
On Smart Query Routing: For Distributed Graph Querying with Decoupled Storage:
Arijit Khan (NTU Singapore), G. Segovia, and D. Kossmann

Roadmap

- Distributed graph querying and graph partitioning
- Decoupled graph querying system
- Related work
- Smart graph query routing
- Experimental results
- **Conclusions**

Conclusions

- **Decoupled graph querying system**
- **Smart query routing** to achieve higher cache hits for *h*-hop traversal queries
-
  - **emphasize less on expensive graph partitioning and re-partitioning** across storage tiers
  - provide **linear scalability in throughput with more number of query processors**
  - **works well in the presence of query hotspots**
  - **adaptive to workload changes and graph updates**.



Decoupled architecture for graph querying