Scaling Big Data Processing with Utility-aware Distributed Data Partitioning

Prof. Dr. Ling Liu
CERCS, School of CS
Georgia Institute of Technology

http://www.istc-cc.cmu.edu/

Intel Science & Technology
Center for Cloud Computing

Contributing Students

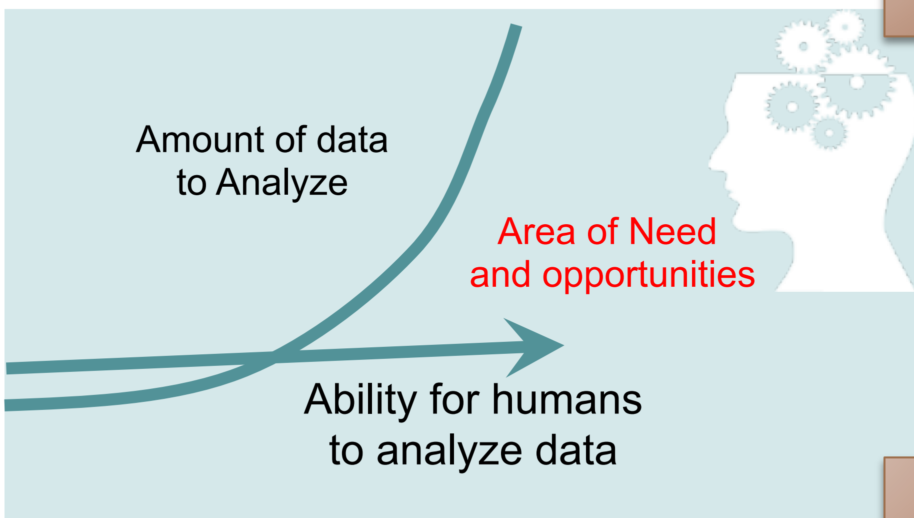
- Students on this project
  - Kisung Lee (ISTC GRA)
  - Emre Yigitoglu
  - Qi Zhang
  - Yang Zhou

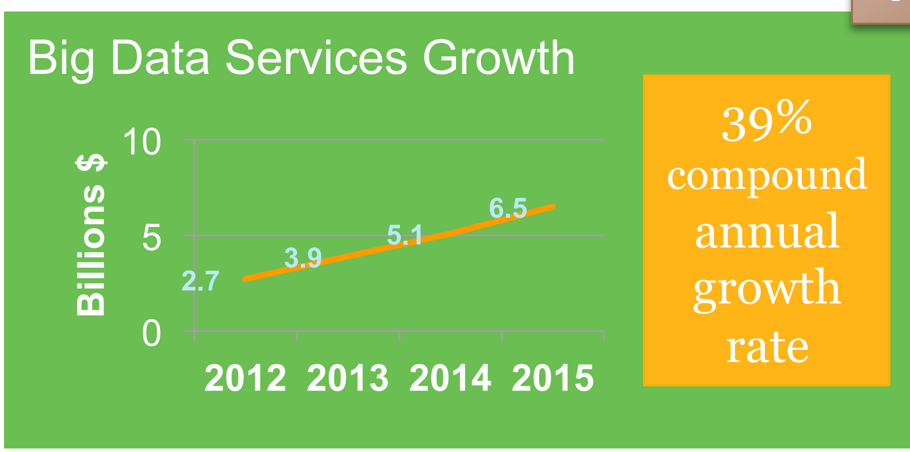
- Related Publications
  - VLDB 2014, IEEE SC 2013, VLDB 2013, ACM SIGKDD2013, IEEE ICWS 2013, IEEE Cloud 2013

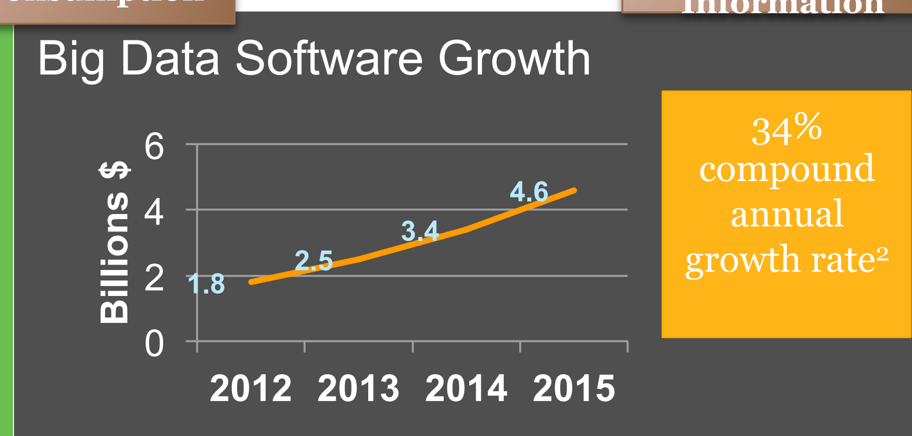
Outline

- Graph Models for Big Data
- Graph Queries v.s. Iterative graph algorithms
- Customizable Distributed Graph Partitioning Framework for Graph Queries
- Ongoing/Future Work

Data grows faster than intelligence







IDC Market Analysis, Worldwide Big Data Technology and Services 2012-2015 Forecast , 2012

Large Scale Data Analysis

- **Graph abstractions**
  - popular data structure to analyze large and complex datasets
  - graph mining can derive implicit/hidden spatial-temporal correlations among data objects
- **Many applications** can benefit from graph abstractions and graph analysis
  - Internet, Social networks, Semantic Web (RDF), senor networks, petascale simulation
- **Challenges**
  - data size (volume), heterogeneity (variety), velocity and data quality
  - Problem complexity and Computation complexity

Characterizing Graph Computation

- **Two broad classes of problems:**
  - **Graph queries** to find matchings (e.g., subgraph matchings)
  - **Iterative Algorithms** to find clusters, orderings, paths, patterns, …
- **Graph Kernel**
  - traversal, shortest path algorithms, spanning tree algorithms, topological sort, …
- **Many factors can influence the choices of graph analytic algorithms and performance optimization techniques**
  - graph sparsity (edge/vertex ratio), diameter, graph heterogeneity, vertex degree distribution, directed/undirected, simple/multi/hyper graph, problem-specific or domain specific characteristics

Scaling Graph Analysis

Common Techniques

- **Compression**
  - Compact storage on disk and compact data structure in memory
- **Data placement (disk, memory)**
  - Balance computation with storage
  - maximize sequential access and minimize random access to edges and/or vertices
- **Indexing (vertex, edge)**
  - utilizing sequential access to reduce unnecessary random access
- **Caching (multiple levels)**
  - Performance gain for repeated vertex/edge access
- **Parallel Computation (multiple levels)**
  - Multi-threads, Multi-cores, Disk and memory optimization, Cluster-computing
  - Minimizing parallel overhead (minimizing communications & maximize local computation)

Scaling Graph Analytics: Iterative Algorithms

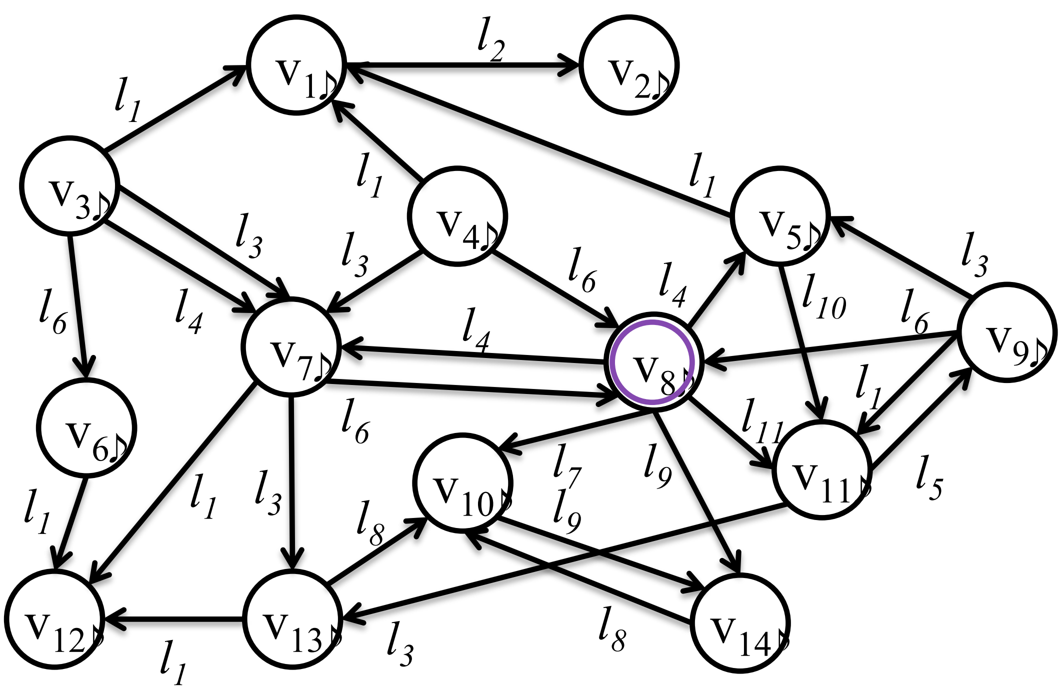
- **Indexing (hash index on source vertex)**
  - Pregel (SIGMOD 2010), GraphChi (ODSI 2012), X-Stream (SOSP 2013)
- **Data Placement/Partition**
  - Disk placement (GraphChi, Pregel)
  - Memory placement (GraphChi, Pregel, X-Stream)
  - Locality (balance computation with storage)
    - Vertex centric, disk-resident shards and streaming shards (GraphChi)
    - Edge centric, memory-constrained streaming partitions (X-Stream)
- **Caching (memory, SSD)**
  - X-Stream (SOSP 2013) (vertex and edge level)
- **Parallel Processing**
  - Data partitioning (vertex or edge)
  - Bulk Synchronous Parallel (BSP) programming model (local computation, communication, barrier synchronization)
  - Minimizing communications, minimizing overhead of barriers by overlapping communication w/ computation

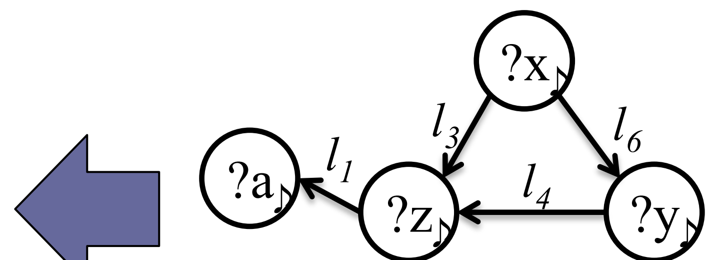
Scaling Graph Analytics: Graph Queries

- **Indexing (source vertex, edge, destination vertex)**
  - RDF-3X (VLDB 2010), BitMat (WWW 2011),TripleBit (VLDB 2013)
- **Compression (vertices, edges)**
  - RDF stores, some graph databases
- **Data Placement**
  - Disk placement (index permutation)
  - Memory placement (sequential access + Index based random access)
- **Caching**
  - query level
- **Parallel Processing**
  - data partitioning (vertex, edge)
  - maximizing parallelism while minimizing communication overhead

Graph Queries (Pattern matching)

- **Graph pattern queries** are subgraph matching problems
  - One of the most fundamental graph operations
- **Executing** a graph pattern query
  - Find a set of **subgraphs** in a given graph, which match the given graph query pattern if we can substitute the query variables with vertices and edges in the graph.
  - Variables are denoted by a prefix **"?"**





Q4: *(?x, l_3, ?z), (?x, l_6, ?y), (?y, l_4, ?z), (?z, l_1, ?a)*

Processing Graph Queries: Challenges

- Graph datasets often exhibit **higher data correlations**
  - Entities (vertices) are **correlated** through both direct and indirect links (edges)
  - High **heterogeneity**
    - heterogeneous types of entities (vertices)
    - heterogeneous types of links (edges)
  - Highly **skewed distribution** (some high degree vertices, many low degree vertieces)

- Graph computations often **exceed the processing capacity** of conventional hardware, software systems and tools
  - Intermediate results size exceeds the available memory
  - Fail to deliver the computation within acceptable latency
    - Time complexity with respect to Disk IO, Network IO

Distributed Processing of Graph Queries

- Demand **partitioning** a big graph into small partitions and **distributing** the partitions over a cluster of worker nodes

- **Existing** Graph Partitioning Algorithms
  - **Random** Partitioning
    - Well-balanced but high overhead for most graph computations due to a large amount of cross node coordination
  - **Hash** Partitioning
    - Poor performance for many graph operations due to high overhead of cross node coordination and data shipping.
  - **Min-Cut** Partitioning
    - High partitioning overhead

Dataset	#vertices	#edges	avg. #out	avg. #in	**METIS**
Freebase	51,295,293	100,692,511	4.41	2.11	> 26 hours
DBLP	25,901,515	56,704,672	16.66	2.39	> 7 hours

Our Approach

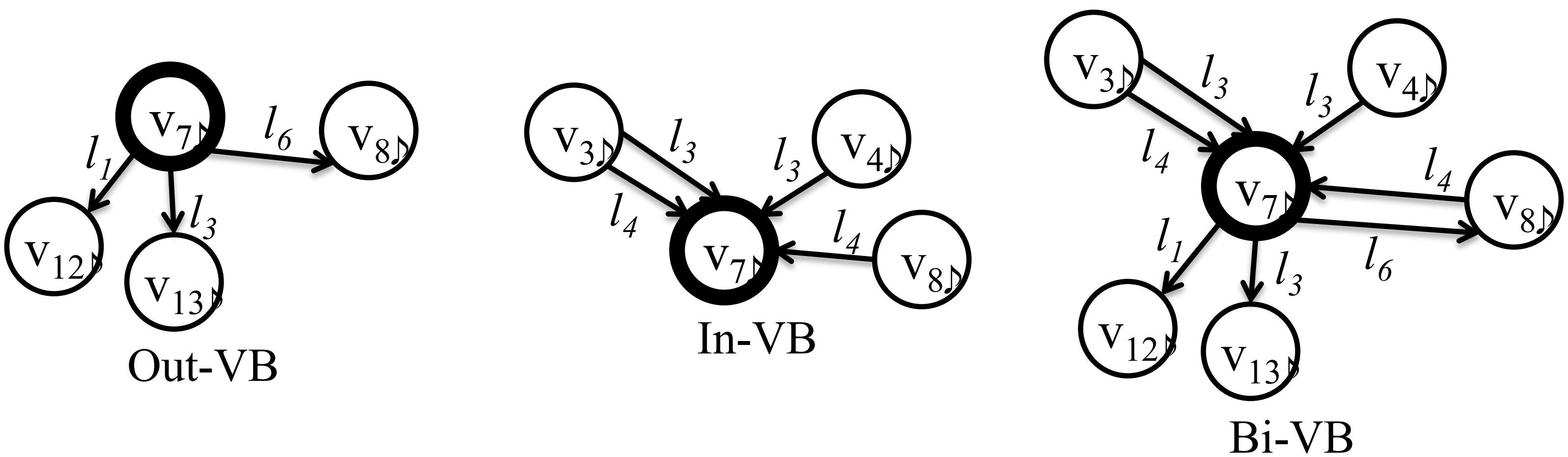
- **VB-Partitioner:** a customizable distributed graph partitioning framework
  - **Goal:** improve distributed graph processing efficiency by
    - **maximizing** intra-partition (local) processing capacity and
    - **minimizing** inter-partition communication cost (cross-worker coordination and data shipping)

  - **Main Features**
    - **Data Partitioning**
      - Constructing **Vertex Blocks** to capture general graph processing locality
      - Constructing **k-hop Extended Vertex Blocks** to distribute vertex blocks with better query locality
      - **Partitions a graph by grouping** its Vertex Blocks based on structural correlation to maximize parallelism in graph processing
      - Introduce **optimization** techniques to reduce the size of each partition

    - **Computation Partitioning:**
      - **Partition-aware** processing of graph queries with maximum parallelism while mimimizing inter-partition communication overhead

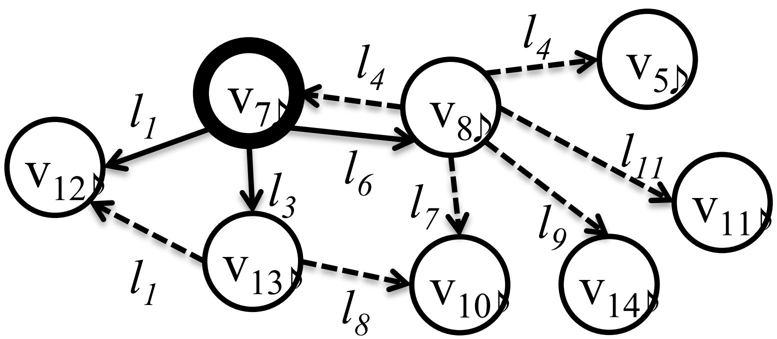
Constructing Vertex Blocks (VB)

- **V**ertex **B**lock (VB)
  - Consists of **an anchor vertex** and its **one-hop** vertices and edges connected to the anchor vertex
    - We call the one-hop neighbor vertices the **affiliated** vertices.
  - We place the **whole VB** (its vertices and edges) in the **same** partition
- **Three types** of vertex blocks to capture general graph processing locality
  - **Out**-VB: include only out-edges and the corresponding affiliated vertices
  - **In**-VB: include only in-edges and the corresponding affiliated vertices
  - **Bi**-VB: include all connected edges and the corresponding affiliated vertices

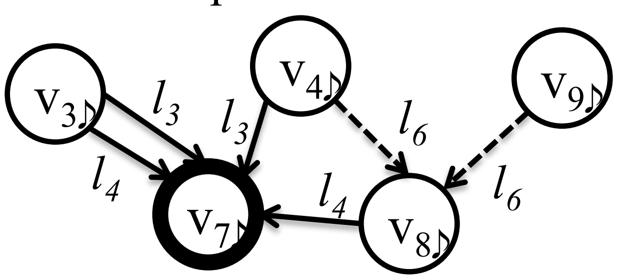


Extended Vertex Blocks (EVBs)

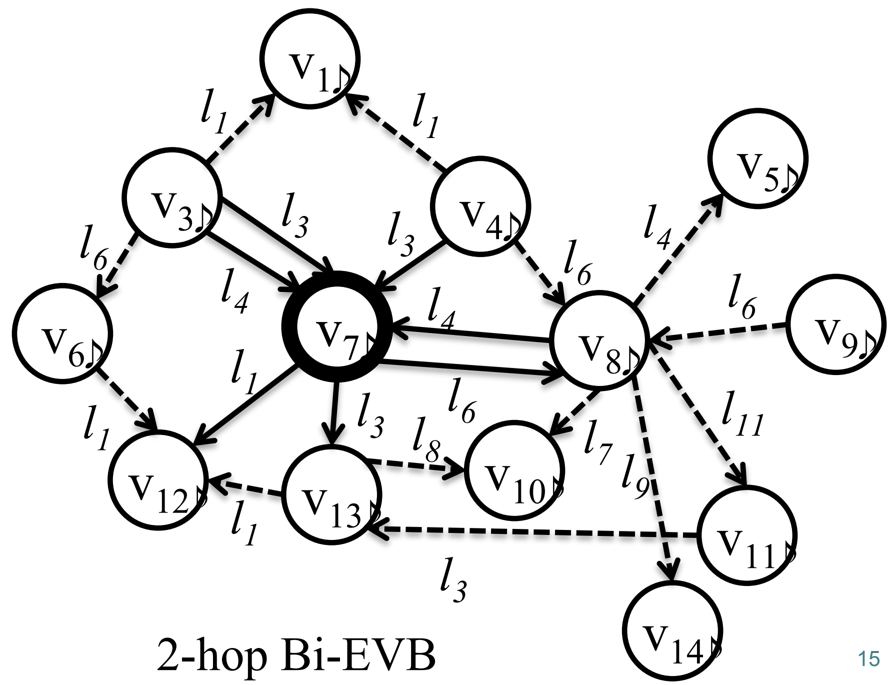
- K-hop **Extended Vertex block** (EVB)
  - Consists of **an anchor vertex** and vertices and edges which are **reachable in k-hops** from its anchor vertex.
  - When k=1, an EVB is a VB.
  - Can be seen as an **extension** of an VB with the same anchor vertex by k-hop neighbor-based expansion.
- **Three types** of EVBs to distribute vertex blocks with access locality
  - Defined in terms of which direction the EVB is expanded from its VB: **Out**-EVB, **In**-EVB, **Bi**-EVB



2-hop Out-EVB



2-hop In-EVB of



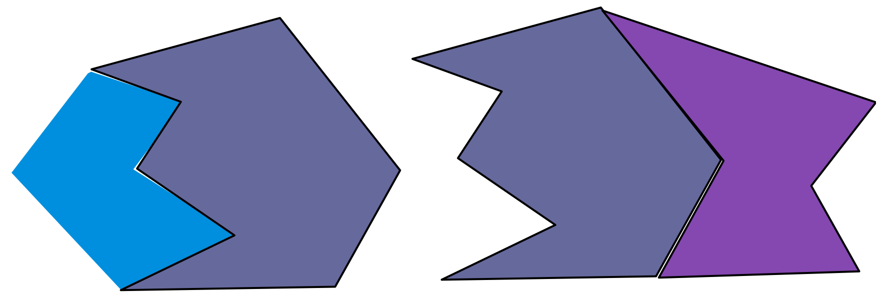
2-hop Bi-EVB

Extended Vertex Blocks (EVBs)

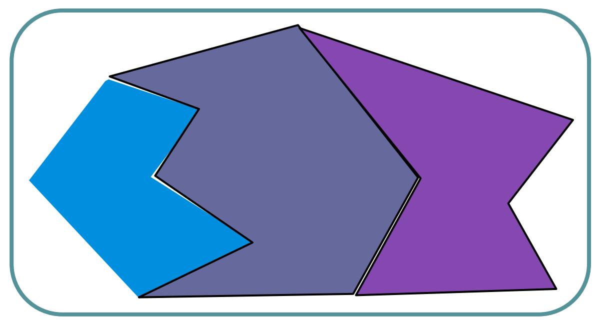
- K-hop extended vertex blocks
  - K is system supplied by default and can be tuned by users
  - Larger K → high degree of edge replication
  - K=1 → Vertex Block
    - No edge replication
  - K>1 Extended Vertex Blocks
    - Edge replication (in-edge, out-edge and bi-direction)
    - K=3 sufficient for queries over most RDF datasets

Mapping VBs/EVBs to Partitions

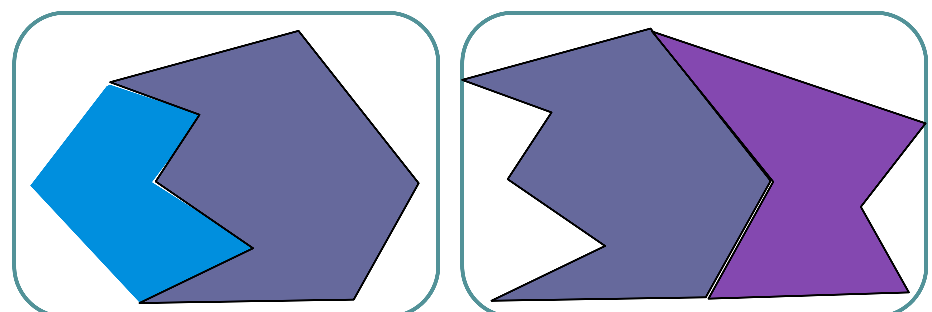
- Strategies
  - Each VB/EVB should be mapped/placed to a partition
  - **Highly correlated** VBs/EVBs should be mapped/placed to the same partition
- Goals
  - **Balanced partitions**
    - One big partition in the imbalanced partitions can be a performance bottleneck
  - **Reduced replication**
    - Smaller partitions usually mean faster local query processing
    - We need to group EVBs sharing many edges
  - **Fast grouping time**
    - To reduce the overhead of partitioning



EVB1 EVB2



Same partition



Different partitions

VB/EVB Grouping and Optimizations

- Grouping Methods
  - **Random** Grouping/Placement
  - **Hash** based Grouping/Placement
    - Hash on anchor vertex of EVBs
  - **Min-cut** Graph based Grouping/Placement

- Domain Specific Optimizations
  - **Selective k-hop Edge Replication**
    - only replicate edges along selective branches
  - **Prefix based pre-partition optimization**
    - Prefix-based hashing prior to constructing vertex blocks
      - Example: URI-hierarchy for RDF, pages from the same domain

Partition-aware Query Processing

- Given a query Q over the k-hop VB-partitioning (in-edge, out-edge or bi-direction), three steps for query processing:
  - Determine whether Q can be processed at each worker node using intra-partition processing directly
    - compute the (in-edge, out-edge or bi-direction) radius of the query
    - **if radius(Q)<=K, then Q can be processed using intra-partition processing directly.**
  - If not, decompose Q into subqueries such that all subqueries can be processed using intra-partition processing.
  - Merging the intra-partition processing results using Hadoop MapReduce.

Experiments

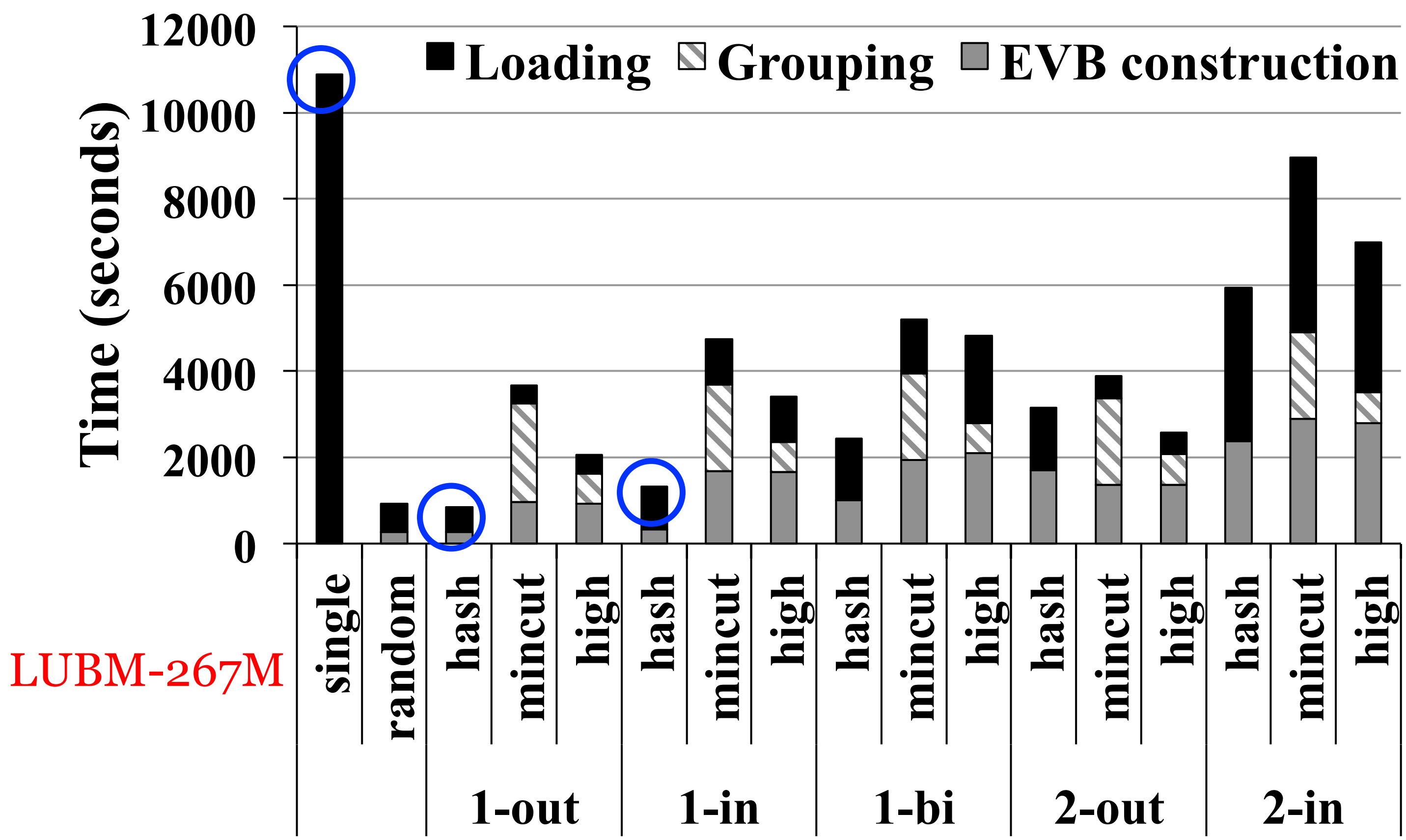
- **Settings**
  - 21 machines (one is the master) on Emulab
    - Each has 12GB RAM, one Xeon E5530, two 250GB disks
  - Hadoop version 1.0.4
  - RDF-3X version 0.3.5 as a local graph processing engine
  - METIS version 5.0.2 for minimum cut-based VB grouping

- **Datasets**

Dataset	#vertices	#edges	avg. #out	avg. #in
Freebase	51,295,293	100,692,511	4.41	2.11
DBLP	25,901,515	56,704,672	16.66	2.39
DBpedia	104,351,705	287,957,640	11.62	2.82
LUBM-267M	65,724,613	266,947,598	6.15	8.27
LUBM-534M	131,484,665	534,043,573	6.15	8.27
LUBM-1068M	262,973,129	1,068,074,675	6.15	8.27
SP2B-100M	55,182,878	100,000,380	5.61	2.11
SP2B-200M	111,027,855	200,000,007	5.49	2.08
SP2B-500M	280,908,393	500,000,912	5.31	2.04

Partitioning and Loading Time

- EVB construction and grouping are implemented using Hadoop cluster
  - Loading time indicates partition loading time of the local graph engine (RDF-3X)
  - Minimum cut-based grouping includes input conversion to METIS input format and METIS running step. The input conversion is implemented also using Hadoop cluster

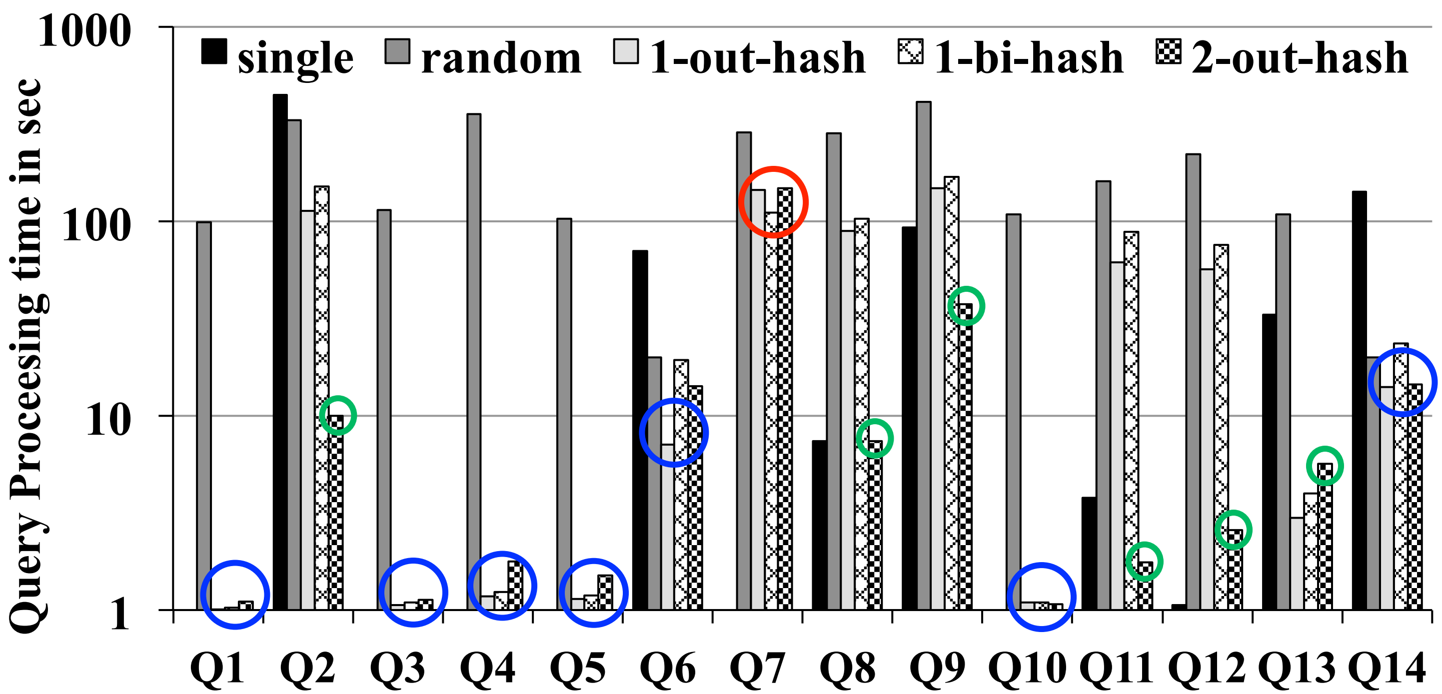


✓ Significantly reduce the graph loading time compared to single server-based approach

✓ Out-edge VBs are faster than in-edge VBs because they are well-balanced

Effect of different types of EVBs on Performance

- Show the huge benefit of **intra-partition** processing
  - For star-like queries, it is usually fast in our partitions
  - For complex queries, 2-out has the best performance
    - Except in Q7 which requires **inter**-partition processing
      - Intermediate result size: 1.2GB >> final result size: 907B
      - But still much faster than random partitioning



LUBM-267M (65,724,613, 266,947,598)

Ongoing / Future Work

- How to handle updates efficiently

- How to support iterative graph algorithms efficiently
  - Shared Memory
    - Reducing parallel overhead of barriers
  - Distributed Memory
    - Optimizing message buffer sizes, #messages
    - overlapping communication w/ computation

Questions