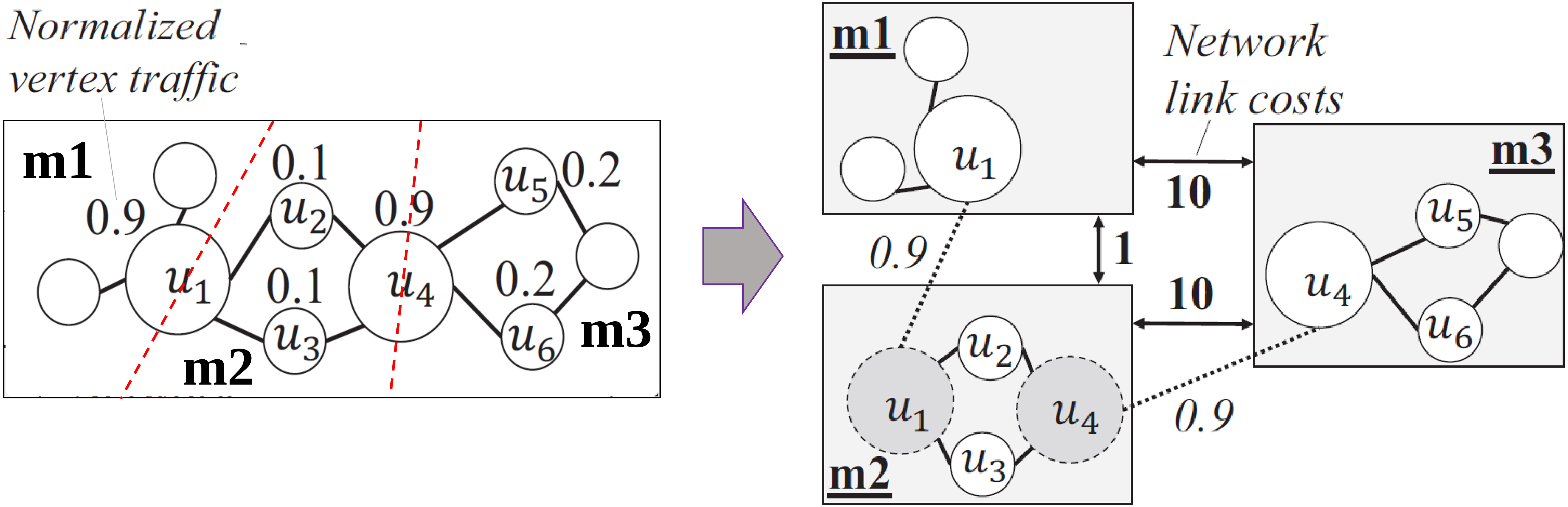
GrapH: Heterogeneity-Aware Graph Computation with Adaptive Partitioning

Christian Mayer, Muhammad Adnan Tariq, Chen Li, Kurt Rothermel

Institute of Parallel and Distributed Systems, University of Stuttgart, Germany

2016 IEEE 36th International Conference on Distributed Computing Systems

Partition graph with vertex-cut

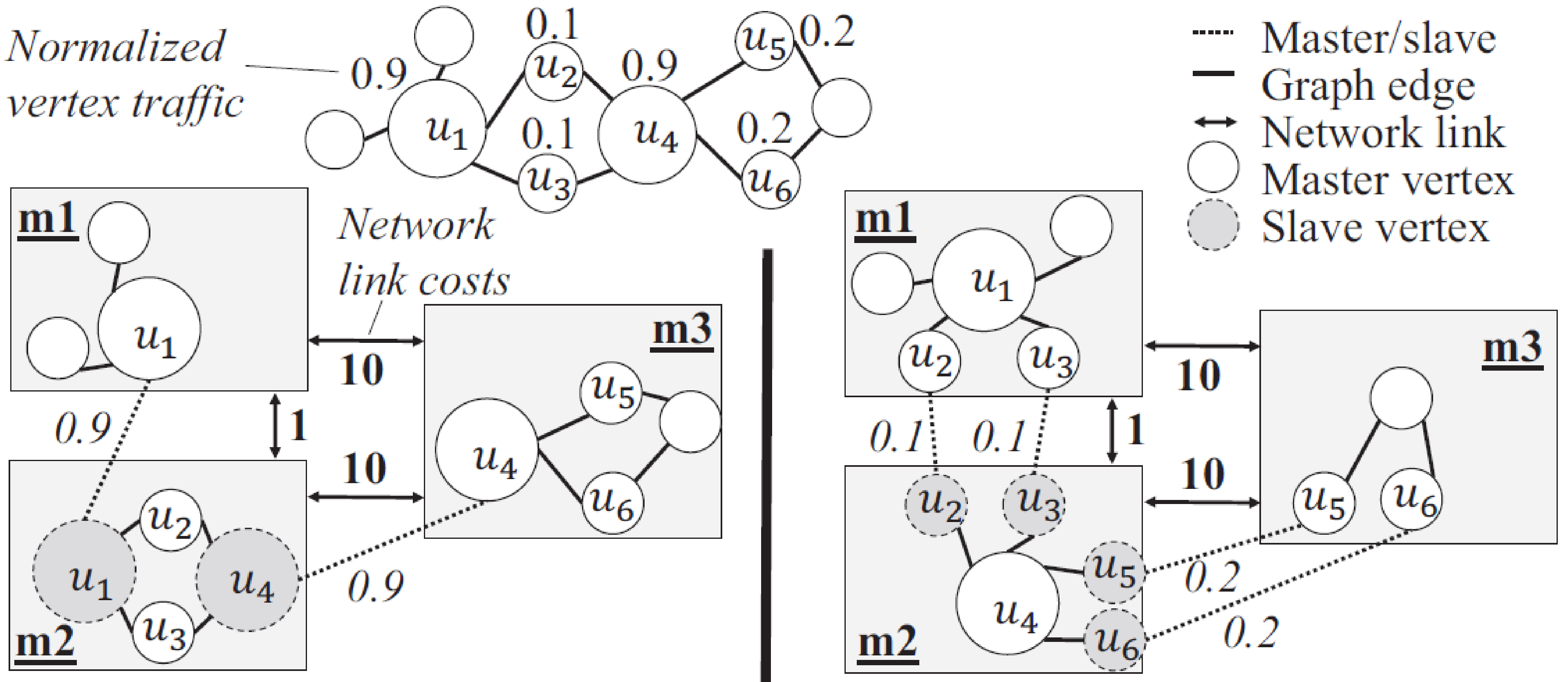


Motivation

- Many high-effect Vertex-centric graph processing systems use graph partitioning algorithms assuming:
  - uniform **vertex traffic** exchanged between graph vertices
  - homogeneous underlying **network costs**.
- However, in real-world scenarios:
  - **vertex traffic** and **network costs** are heterogeneous.

➔ **suboptimal** partitioning decisions and **inefficient** graph processing.

Motivation: Traffic- & network-aware vertex-cut



(a) **Traditional vertex-cut**

replication degree: 4

costs: $0.9 \times \mathbf{1} + 0.9 \times \mathbf{10} = 9.9$

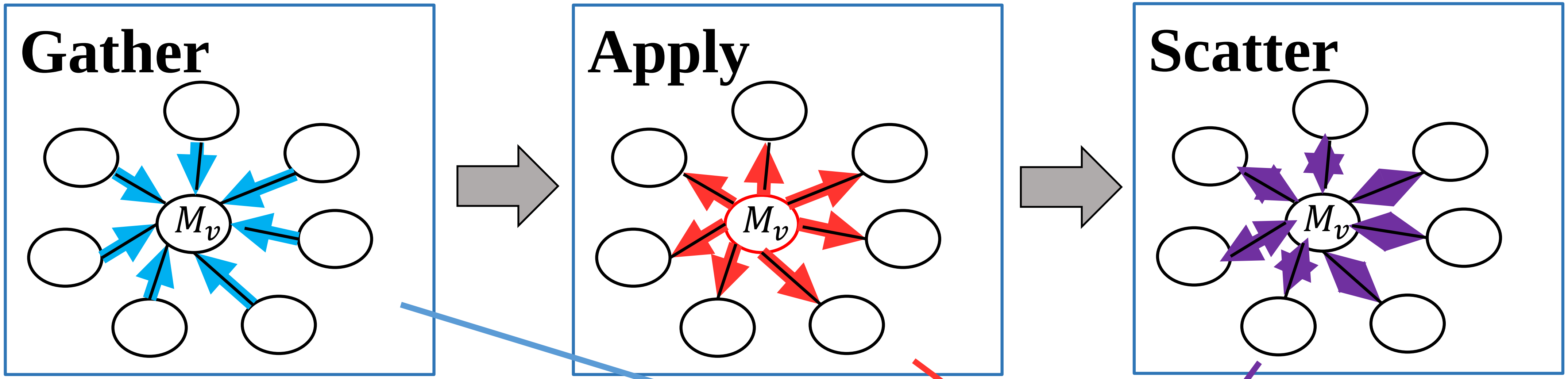
(b) **Traffic- & network-aware vertex-cut**

replication degree: 8

costs: $2(0.1 \times \mathbf{1}) + 2(0.2 \times \mathbf{10}) = 4.2$

Distributed vertex computation model

- organized in iterations
- three phases, Gather, Apply and Scatter (GAS), in each iteration.



Vertex Traffic $$t^v(i) = \frac{1}{|R_v(i)|} \sum_{r \in R_v(i)} (g_r^v(i) + a^v(i) + s)$$

Goal

1. (Mainly) Optimal dynamic assignment of edges to machines minimizing overall communication costs:

Dynamic Assignment $$a_{opt} = \operatorname*{argmin}_a \sum_i \sum_{v \in V} \sum_{m \in R_v^a(i)} t^v(i)\ T_{m,\mathcal{M}_v}$$

2. Machine load *Lm*(*i*), the summed vertex traffic, is bounded by a small balancing factor *λ* > 1:

Load of machine m $$L_m(i) = \sum_{v \in V_m} t^v(i) < \lambda \frac{\sum_{v \in V} t^v(i)}{|M|}.$$

Hardness

- Dynamic network- and traffic-aware partitioning problem is NP-hard.

$$a_{opt} = \underset{a}{\operatorname{argmin}} \sum_{i} \sum_{v \in V} \sum_{m \in R_v^a(i)} t^v(i)\; T_{m,\mathcal{M}_v}$$ is NP-hard

$\because$ the reduce problem: Network- and traffic-unaware partitioning problem is NP-hard

$a_{opt} = \operatorname{argmin}_a \sum_i \sum_{v \in V} \sum_{m \in R_v^a(i)} 1*1 = \operatorname{argmin}_a \sum_{v \in V} |R_v^a|$ is NP-hard

Solution

Consist two phases:

- H-load:
  - a partitioning algorithm for pre-partitioning the graph

- H-move:
  - a dynamic algorithm for runtime refinement using migration of edges.

H-load

Consist two phases:

1. Group partitions into *c* clusters and map edges to partitions such that replicas preferentially lie in the same cluster

Each edge (*u*, *v*) is assigned to a partition *p* as follows:

1) **If no** replica of *u* or *v* on any partition

➔ assign (*u*, *v*) to the least loaded partition.

2) **If exist** partitions containing replicas of ***u and v***

➔ assign (*u*, *v*) to the least loaded of those partitions.

3) **Otherwise**, choose partition *p* such that the new replica preferentially lies in the same cluster as already existing replicas.

H-load

2. Find a good mapping of partitions to machines

Use iterated local search algorithm to greedily minimize (communication) costs.

1) Initially, partitions are randomly mapped to machines.
2) Then iteratively the following method:
    a) Find two machines, if an exchange of partition assignments would lower total communication costs.
    b) If an improvement is found, it is applied immediately.
    c) Perturb a local optimal solution by randomly exchanging two assignments to avoid convergence to local minima.

H-move

- Idea:
  - Each machine locally migrate *bag-of-edges* (in parallel) after each GAS iteration.
    - *bag-of-edges* is the set of edges to be migrated.
  - Finally, if no further improvements can be performed, migration is switched off.

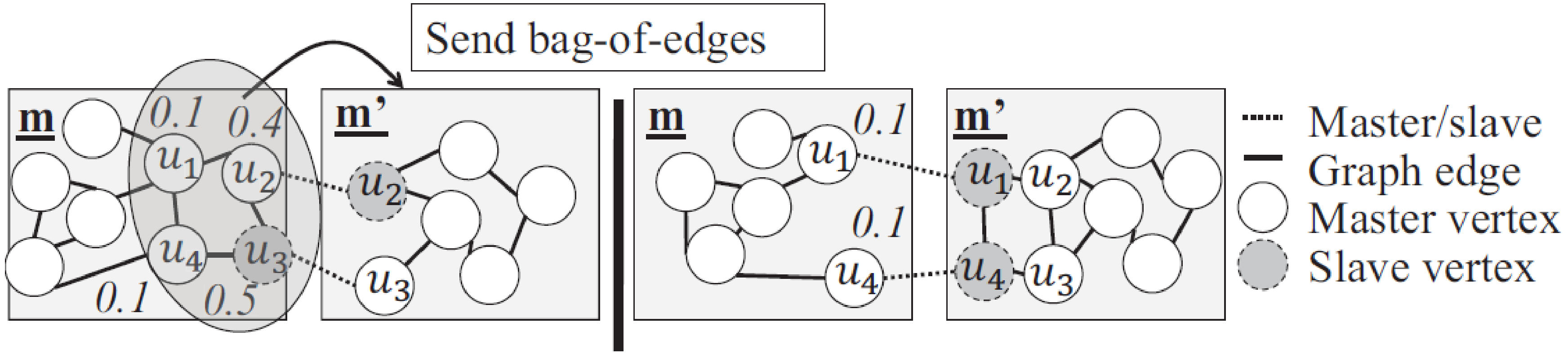


Fig. 4. Example of bag-of-edges migration to reduce inter-machine traffic.

H-move - **Migration algorithm**

Algorithm 1 Migration algorithm on machine m.

$waitForActivation()$
$m' \leftarrow selectPartner()$
$b \leftarrow bagOfEdges(m')$
$lock(b)$
$b \leftarrow updateLocked(b)$
$\Delta c \leftarrow c_{+} - c_{-}$
if $\Delta c < 0$ **then**
    $migrateBag(b)$
$releaseLocks(b)$

H-move - **Determining the bag-of-edges**

Algorithm 2 Determining the bag-of-edges to exchange.

function $bagOfEdges(m')$:
$bag \leftarrow []$
$candidates \leftarrow sort(adjacent(m'))$
while $hasCapacity(m', bag)$ **do**
  $v \leftarrow candidates.removeFirst()$
  $b \leftarrow \{(u, v) | u \neq v\}$
  $\Delta c \leftarrow c_+ - c_-$
  if $\Delta c < 0$ **then**
    $bag \leftarrow bag + b$
return bag

Capacity $C = (L_{m'} - L_m)/2.$

Evaluation - setup

- To get the graph in real world, implemented the three graph algorithms:
  - **PageRank,** denoted as PR
- compared migration strategies with static vertex-cut partitioning approaches:
  - **hashing of edges** (Hash) and **PowerGraph** (PG).
- Implemented GrapH in the Java programming language
- GrapH consists of a master machine and multiple client machines
- The master receives a sequence of graph processing queries $q1$, $q2$, $q3$, ... consisting of user specified GAS algorithms.
- All machines communicate directly via TCP/IP.
- Use two computing clusters with homogeneous and heterogeneous network costs.

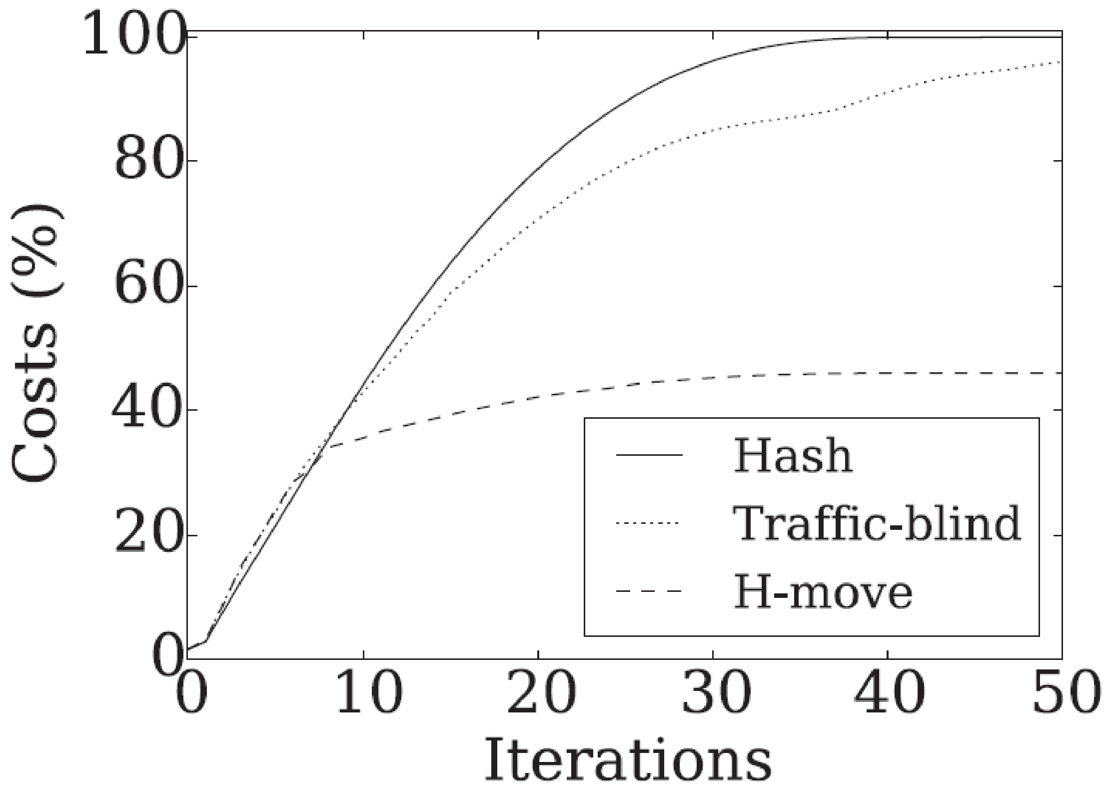
Evaluation - ***Setup***

Machine placement	Incoming traffic	Outgoing traffic
Same AZ	0.00-0.01 $/GB	0.00-0.01 $/GB
Different AZ, same region	0.01 $/GB	0.01 $/GB
Different region	0.00 $/GB	0.02 $/GB
Internet	0.00 $/GB	0.00-0.09 $/GB

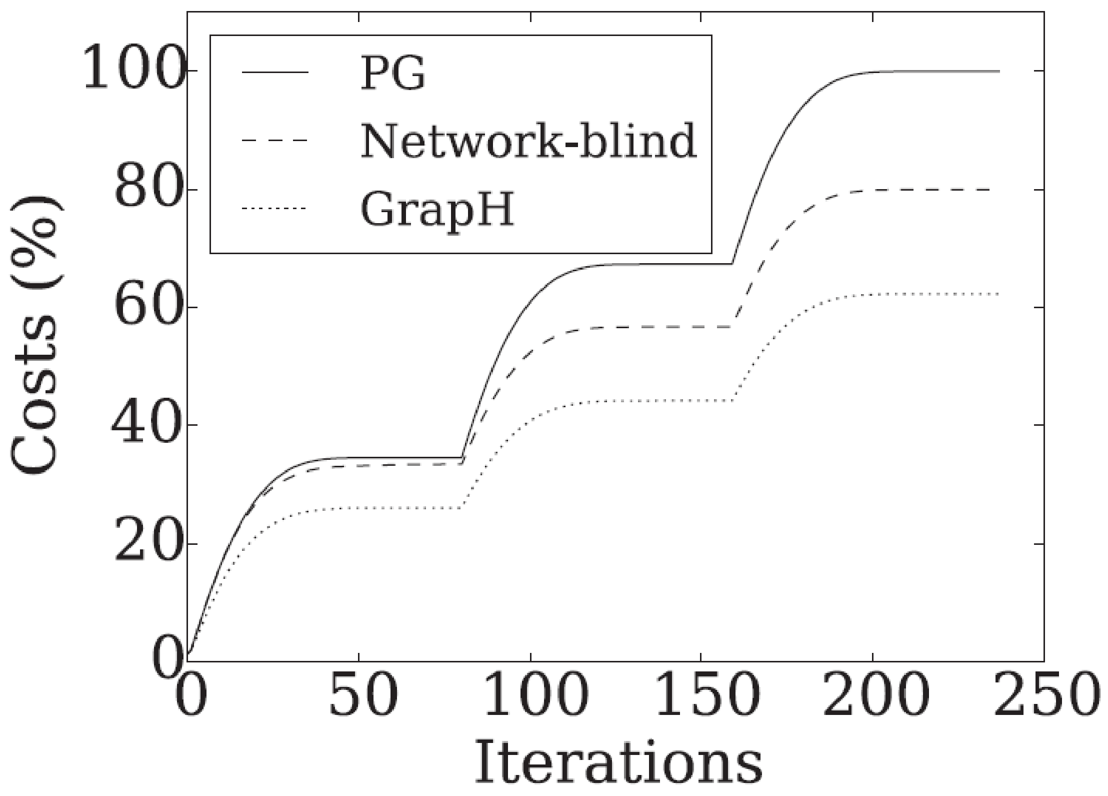
TABLE I
HETEROGENEOUS COMMUNICATION COSTS FOR AMAZON EC2 CLOUD INSTANCES (AUGUST 2015).

- The homogeneous computing cluster (ComputeC) consists of 12 machines, each with 8 cores (3.0GHZ) and 32GB RAM, interconnected with 1 Gbps ethernet.
- The heterogeneous computing cluster (CloudC) is deployed in the Amazon cloud using 8 geographically distributed EC2 instances (1 virtual CPU with 3.3 GHz and 1 GB RAM) that are distributed across two regions, US East (Virginia) and EU (Frankfurt), and four different availability zones. .
- As network costs between these instances, we used the real monetary costs charged by Amazon (Tab. I).

Evaluation - ***Communication costs***

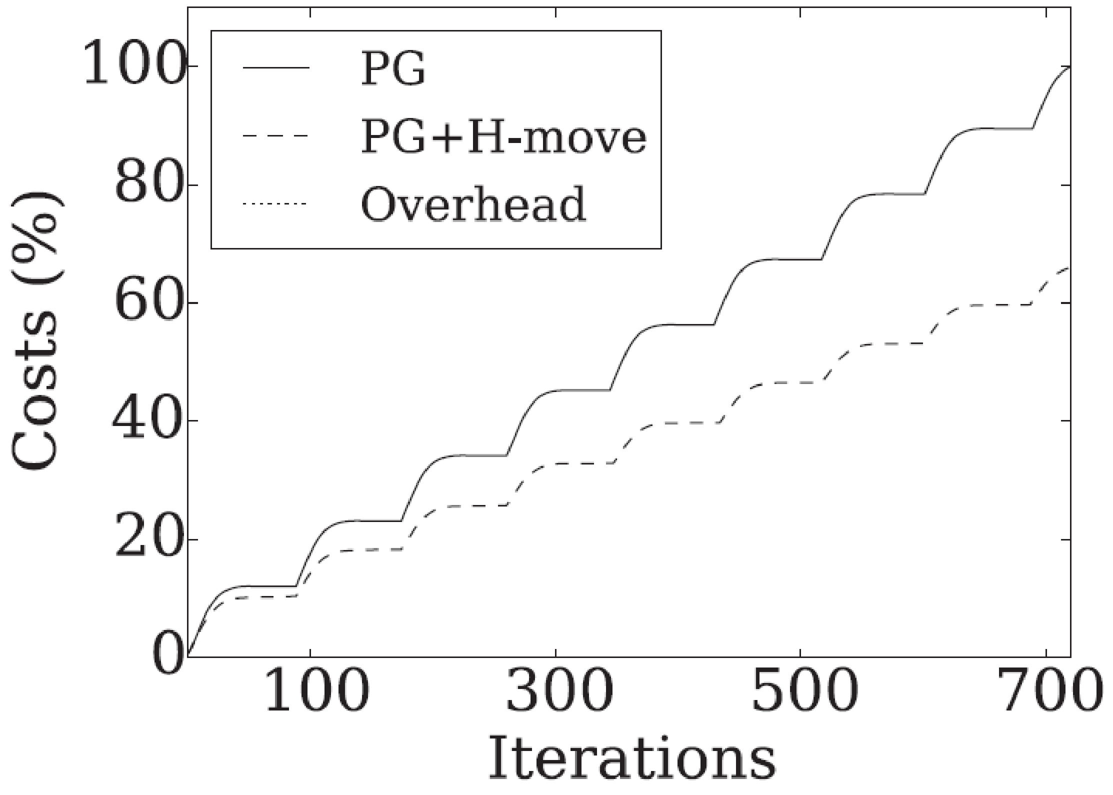


(a) PR on *Twitter*(Traffic-awareness)

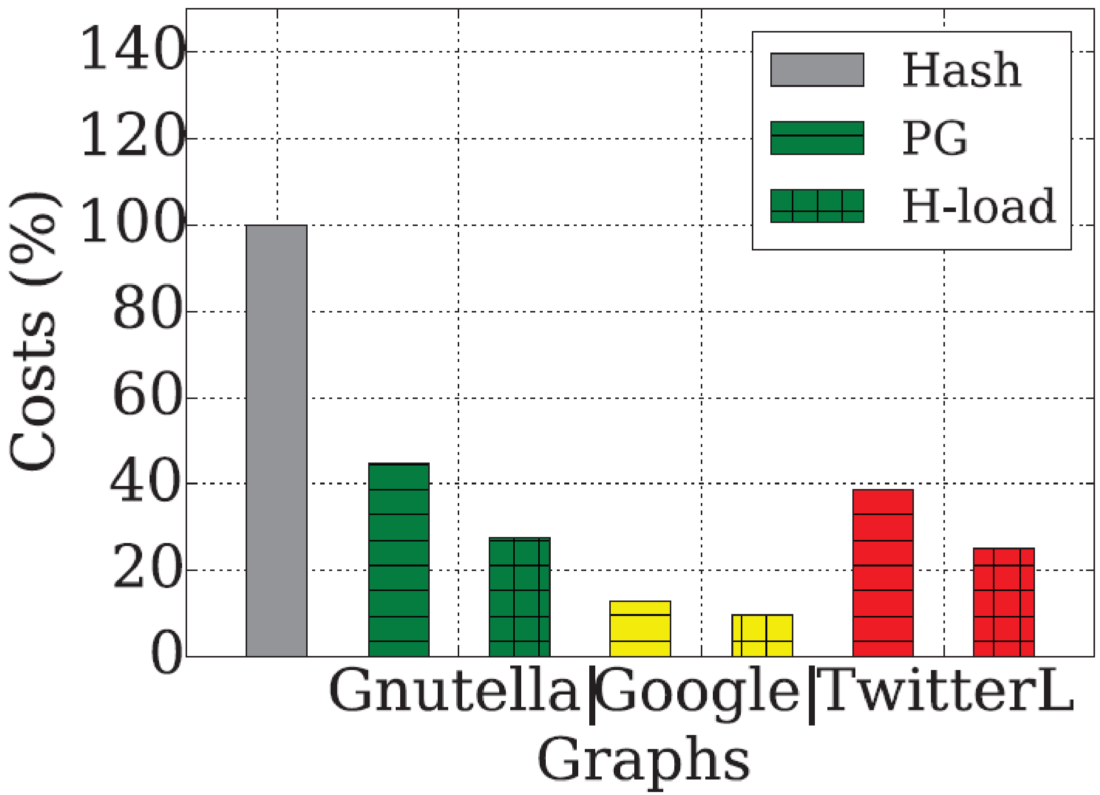


(b) PR on *GoogleWeb*

Evaluation - ***Communication costs***

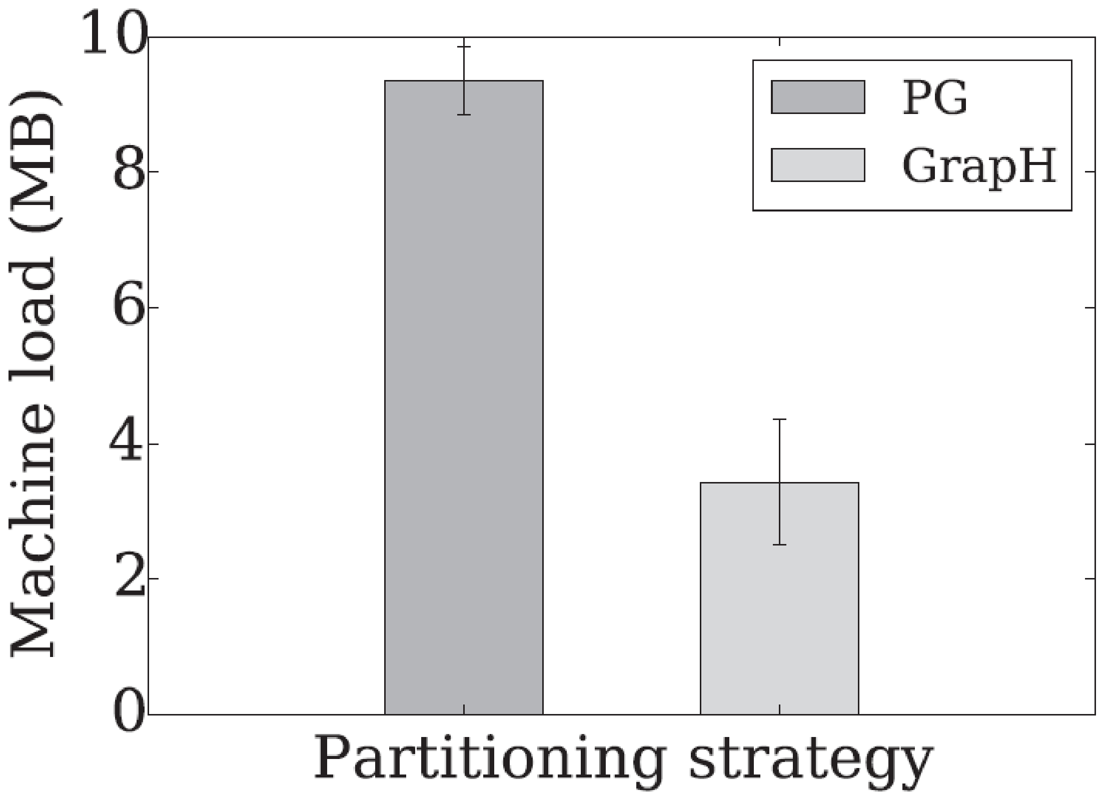


(c) PR on *GoogleWeb*.



(f) Pre-partitionings

Evaluation – ***Load balancing***



(c) PR on *GoogleWeb*.

Conclusion

- Modern graph processing systems use vertex-cut partitioning methods assume:
  - **uniform** vertex traffic
  - **homogeneous** network costs

  do not hold for many real-world applications.
- GrapH considers
  - **dynamic** vertex traffic
  - **diverse** network costs

  By adaptively minimizing communication costs of the vertex-cut at runtime.
- Evaluation show that GrapH **outperforms** PowerGraph's vertex-cut partitioning algorithm by more than 60% communication costs.