USENIX ATC 2018

CGraph: A Correlations-aware Approach for Efficient Concurrent Iterative Graph Processing

Yu Zhang[1], Xiaofei Liao[1], Hai Jin[1], Lin Gu[1], Ligang He [2], Bingsheng He[3], Haikun Liu[1]

[1] SCTS/CGCL, Huazhong University of Science and Technology, China

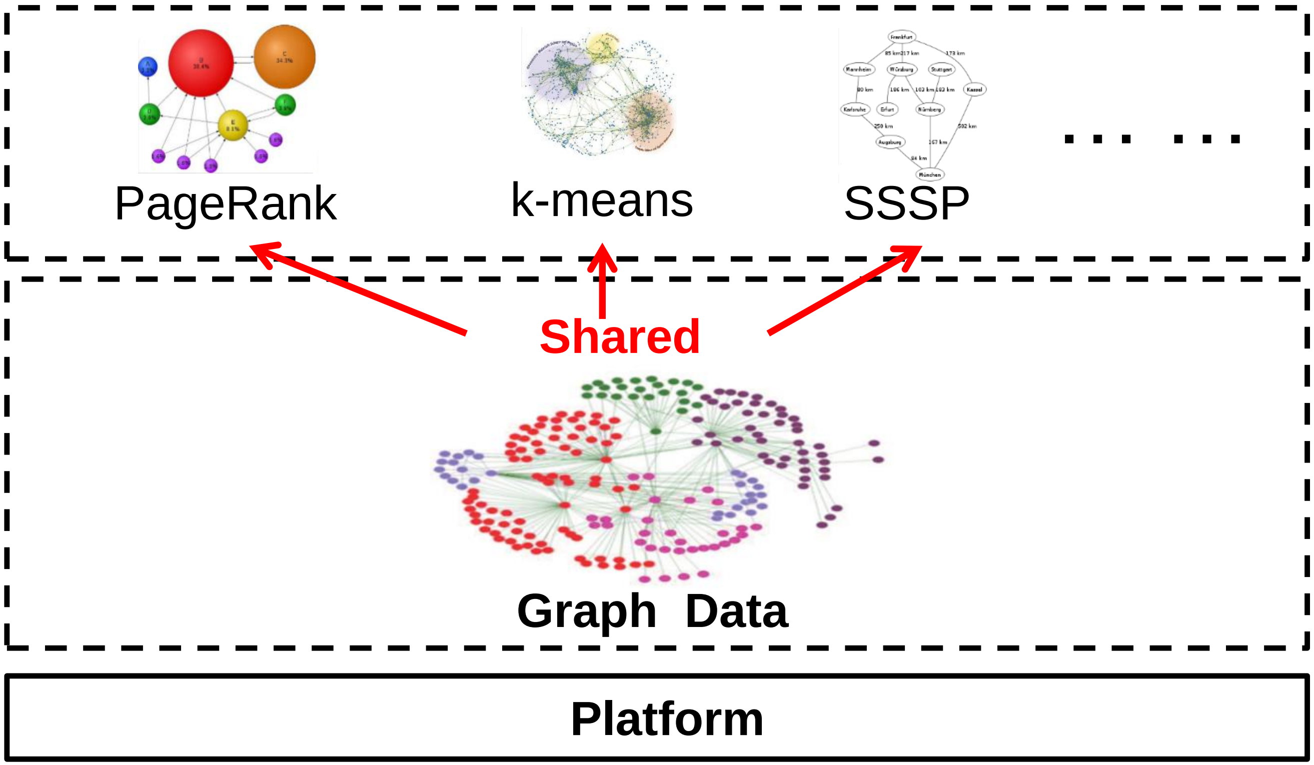
[2] University of Warwick, UK

[3] National University of Singapore, Singapore

Part 1

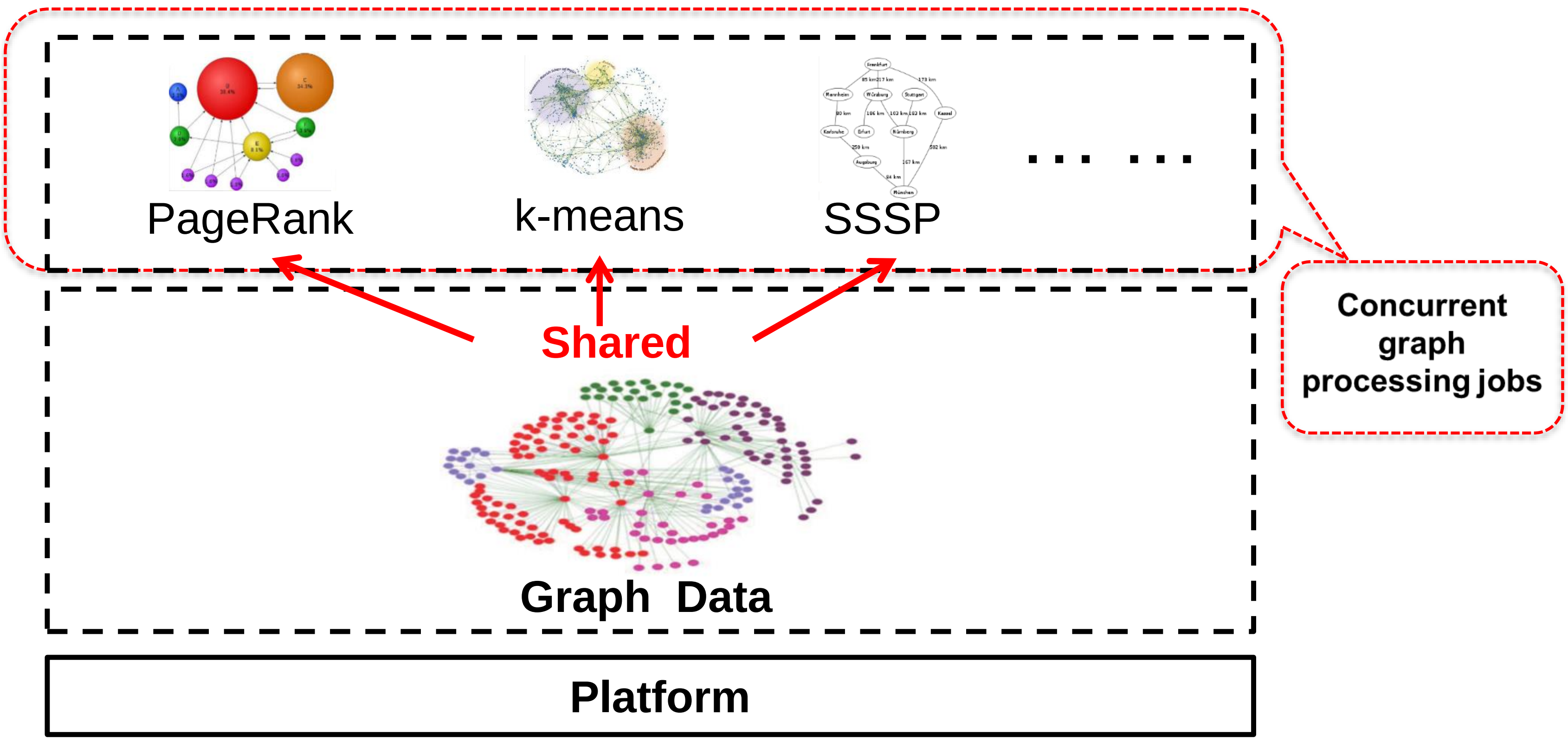
Background and Challenges

What is CGP Job

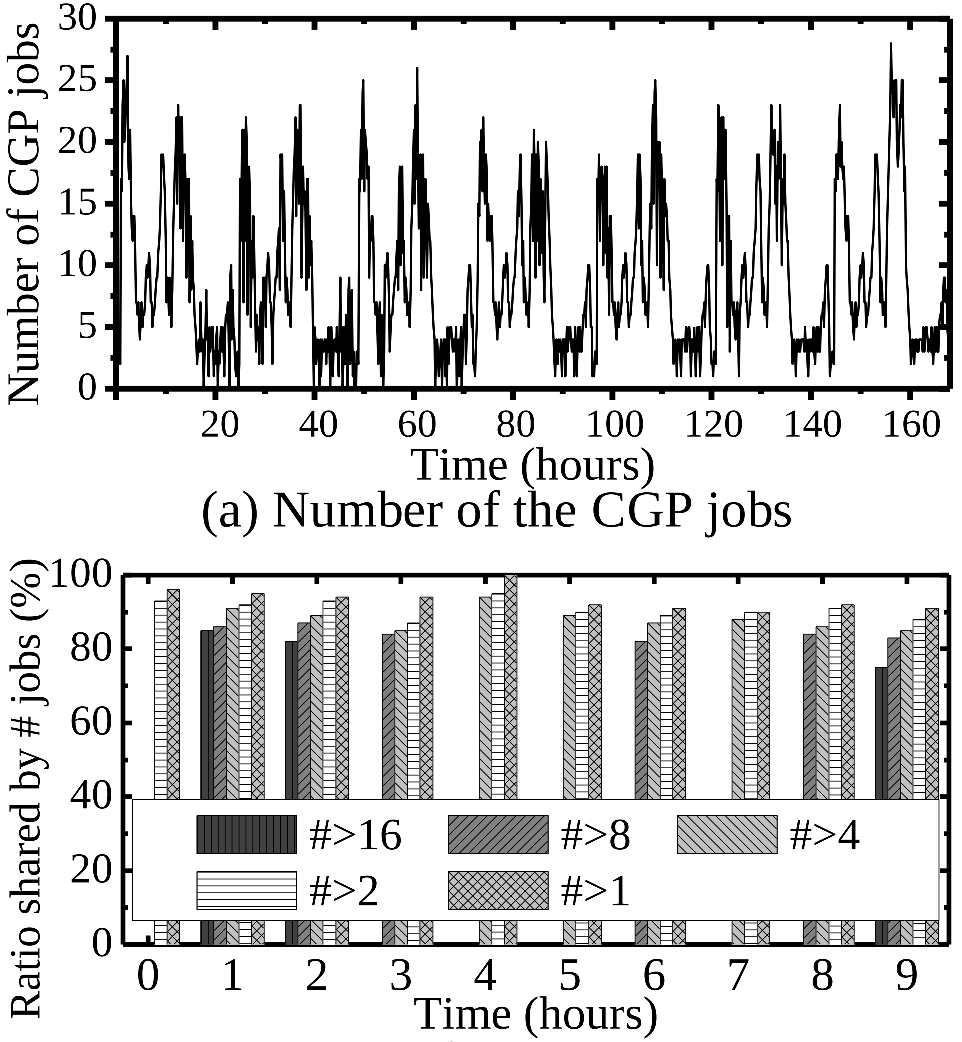


Many concurrent graph processing jobs are daily executed over the same graph (or its different snapshots) to provide various information for different products

What is CGP Job



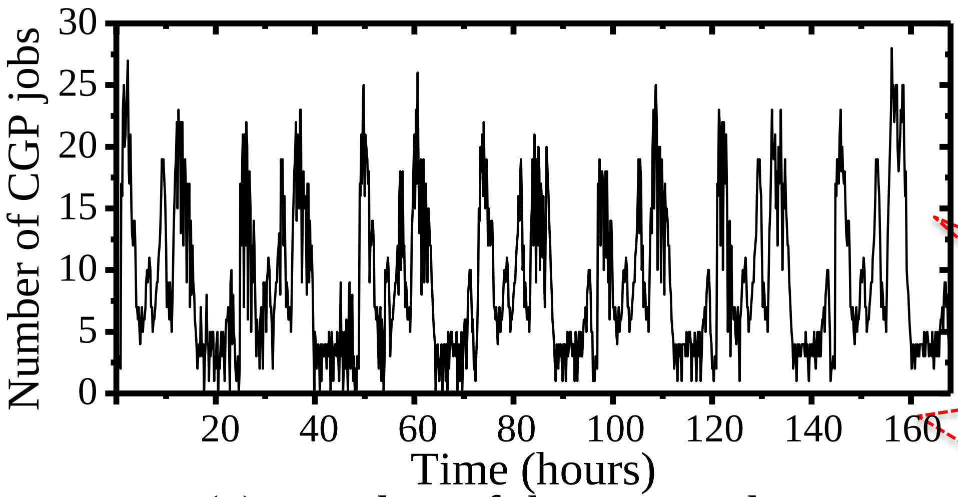
What is CGP Job



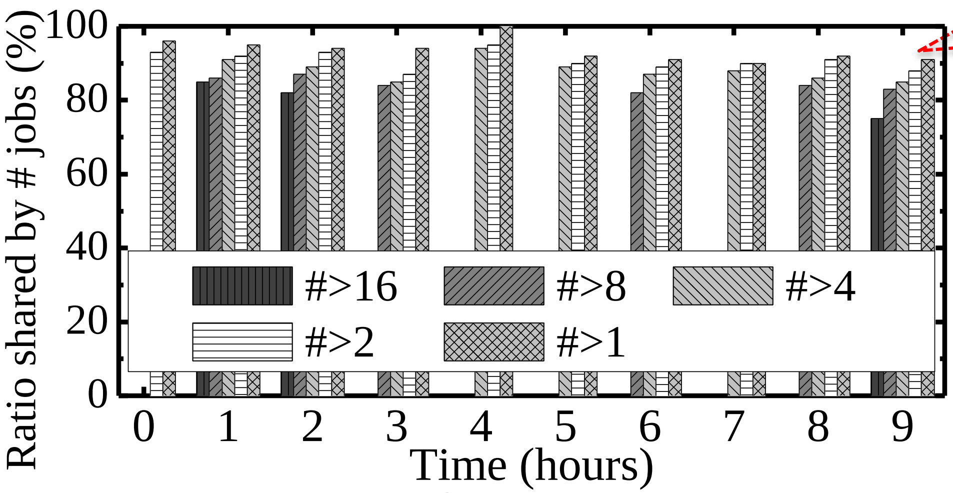
(b) Ratio of shared graph data

The information traced over a large social network

What is CGP Job



(a) Number of the CGP jobs

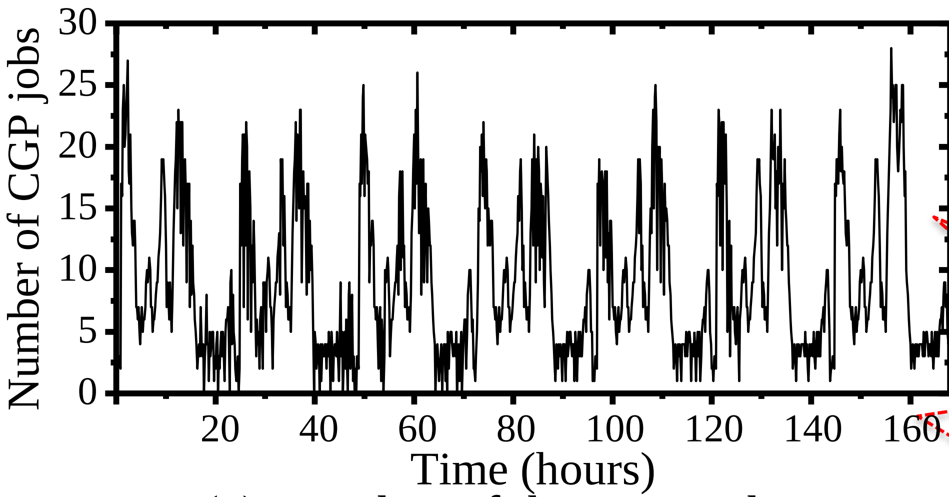


(b) Ratio of shared graph data

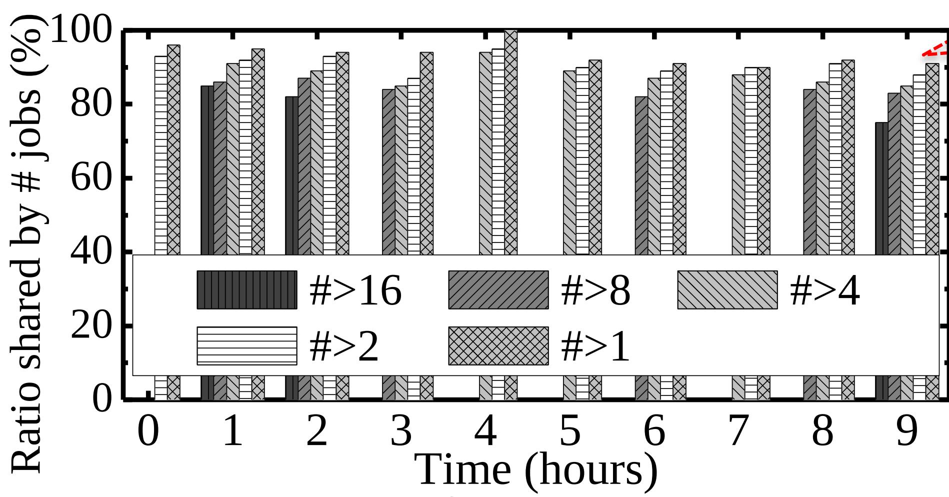
The information traced over a large social network

More than 20 CGP jobs to concurrently analyze the same graph at the peak time

What is CGP Job



(a) Number of the CGP jobs

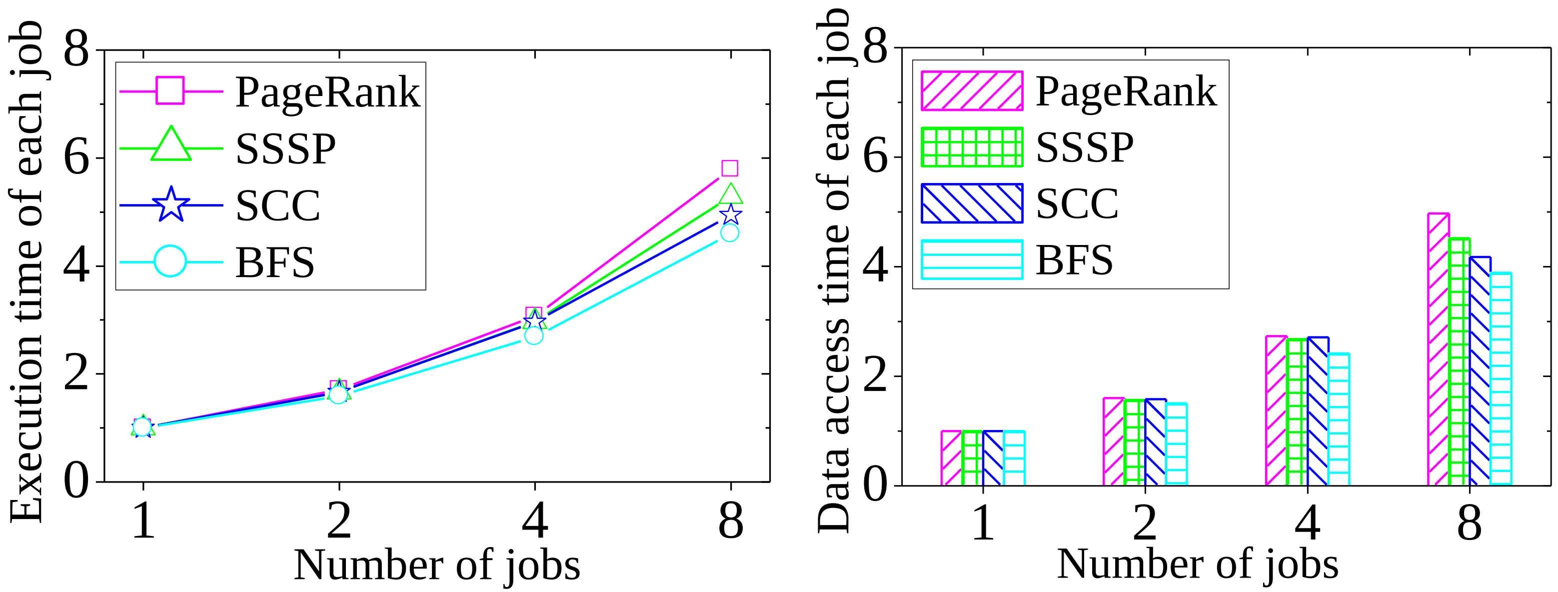


(b) Ratio of shared graph data

The information traced over a large social network

Serious cache interference
and memory wall

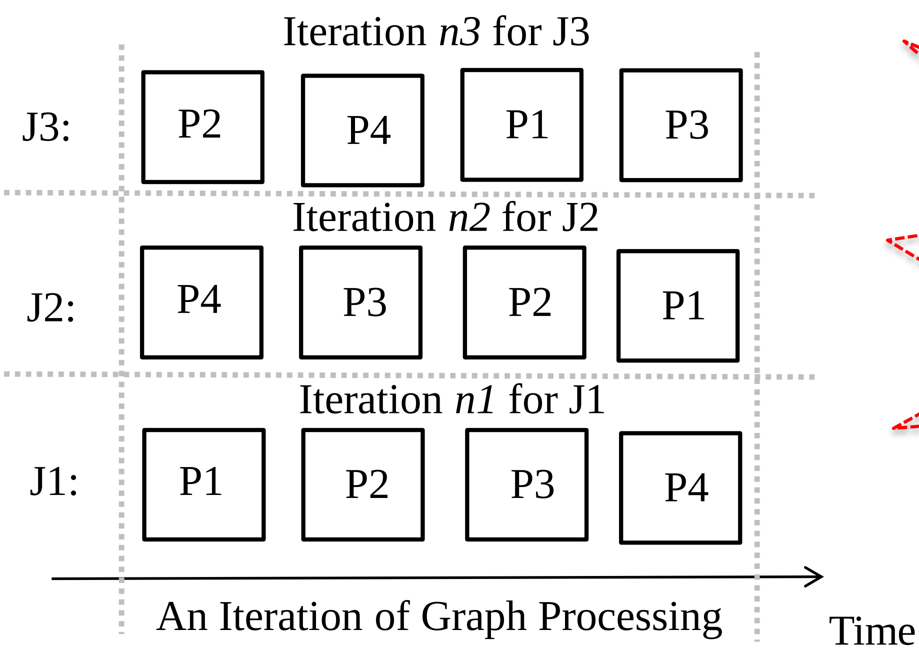
Challenges: Data Access Problems in the CGP Jobs



(a) Average execution time (b) Average data access time

The average execution time of each job is significantly prolonged as the number of jobs increases due to higher data access cost

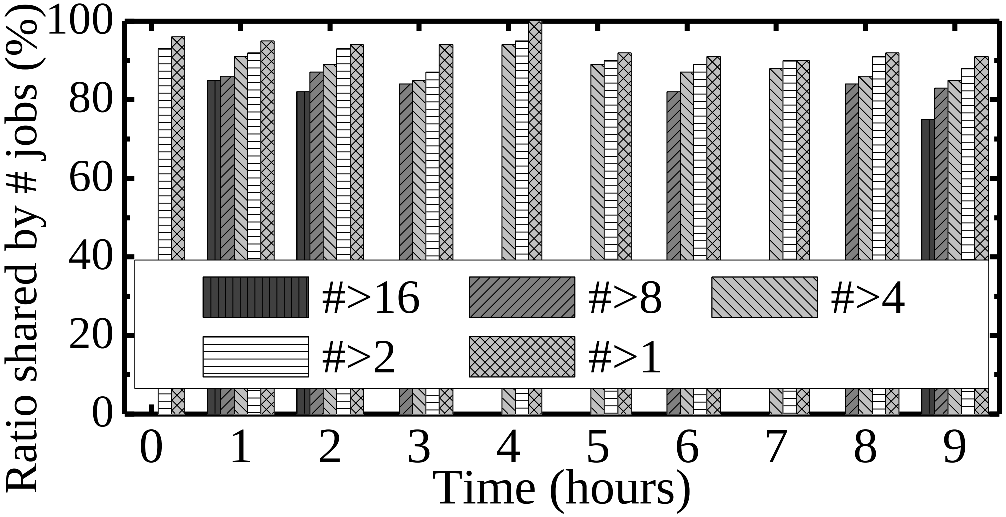
Challenges: An Example



Reason: The CGP jobs contend for data access channel, memory and cache

- The CGP jobs access the shared graph partitions in an individual manner along different graph paths
- The processing time of each partition is various for different jobs

Motivations

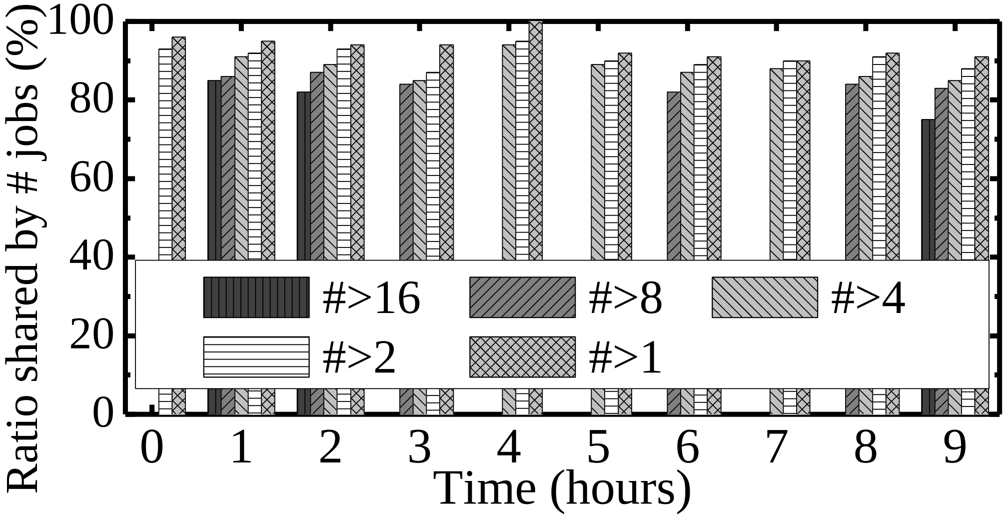


Observations:

-Spatial correlation

-Temporal correlation

Motivations

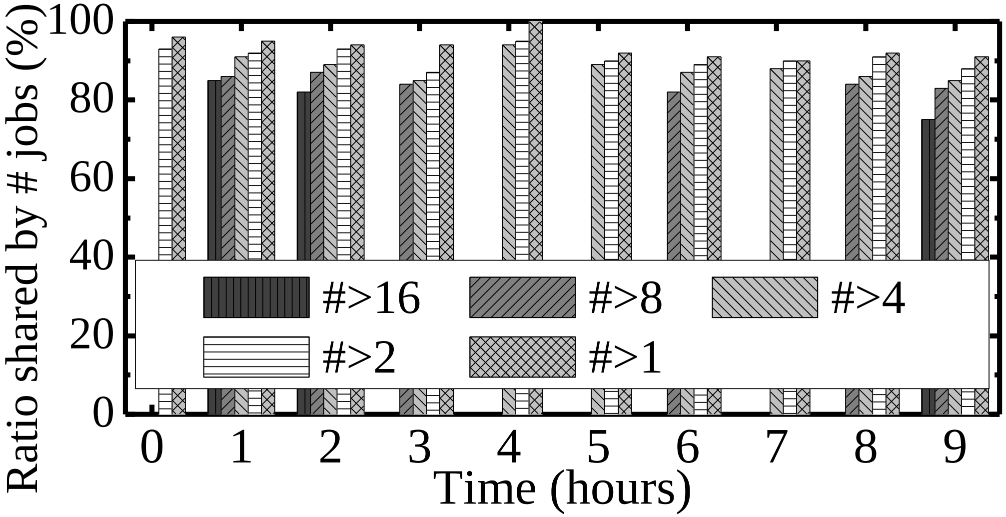


Observations:

-Spatial correlation: The intersections of the set of graph partitions to be handled by different CGP jobs in each iteration are large (more than 75% of all active partitions on average).

-Temporal correlation

Motivations

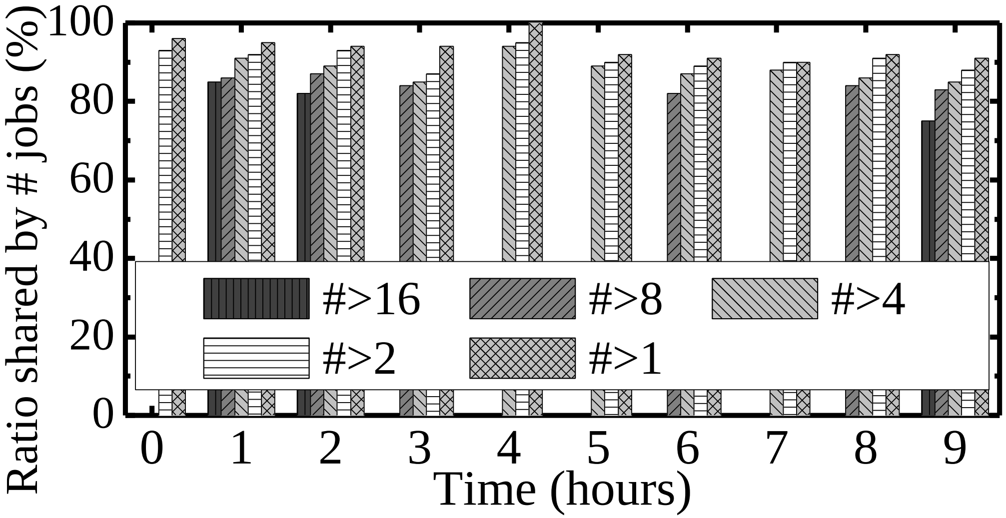


Observations:

-Spatial correlation: The intersections of the set of graph partitions to be handled by different CGP jobs in each iteration are large (more than 75% of all active partitions on average).

-Temporal correlation: Some graph partitions may be accessed by multiple CGP jobs (may be more than 16 jobs) within a short time duration.

Motivations



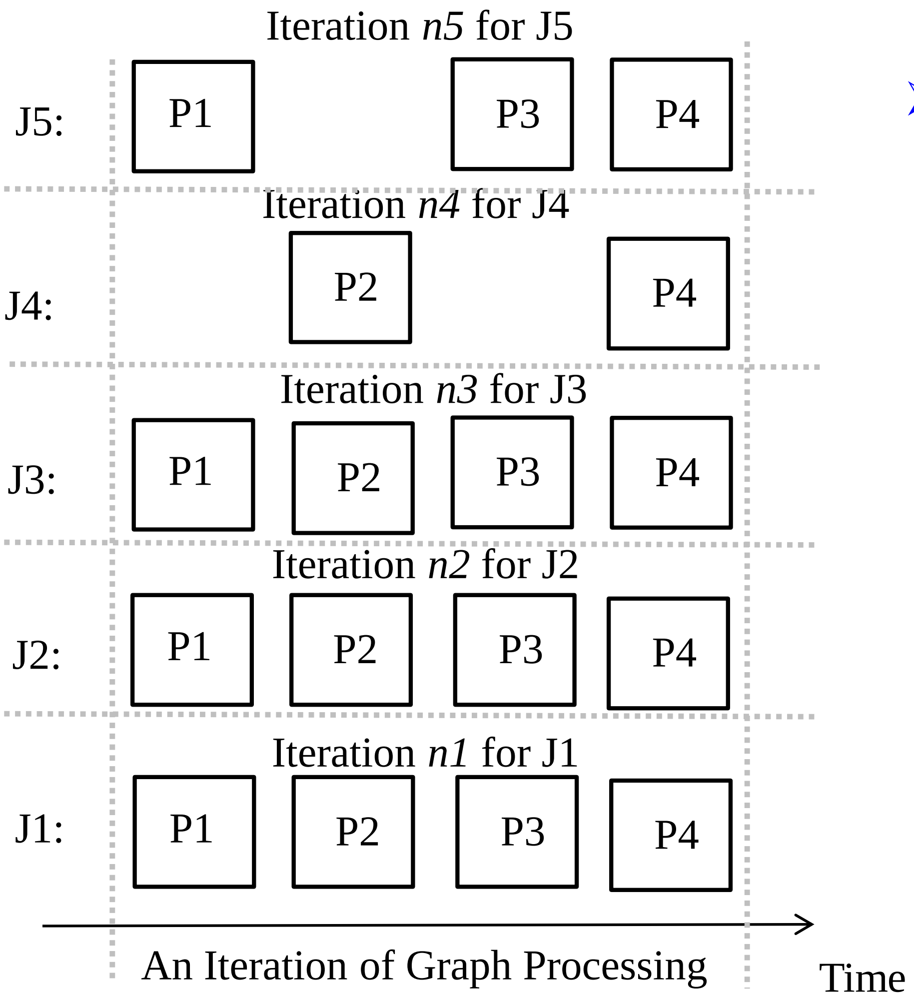
Observations:

-Spatial correlation: The intersections of the set of graph partitions to be handled by different CGP jobs in each iteration are large (more than 75% of all active partitions on average).

-Temporal correlation: Some graph partitions may be accessed by multiple CGP jobs (may be more than 16 jobs) within a short time duration.

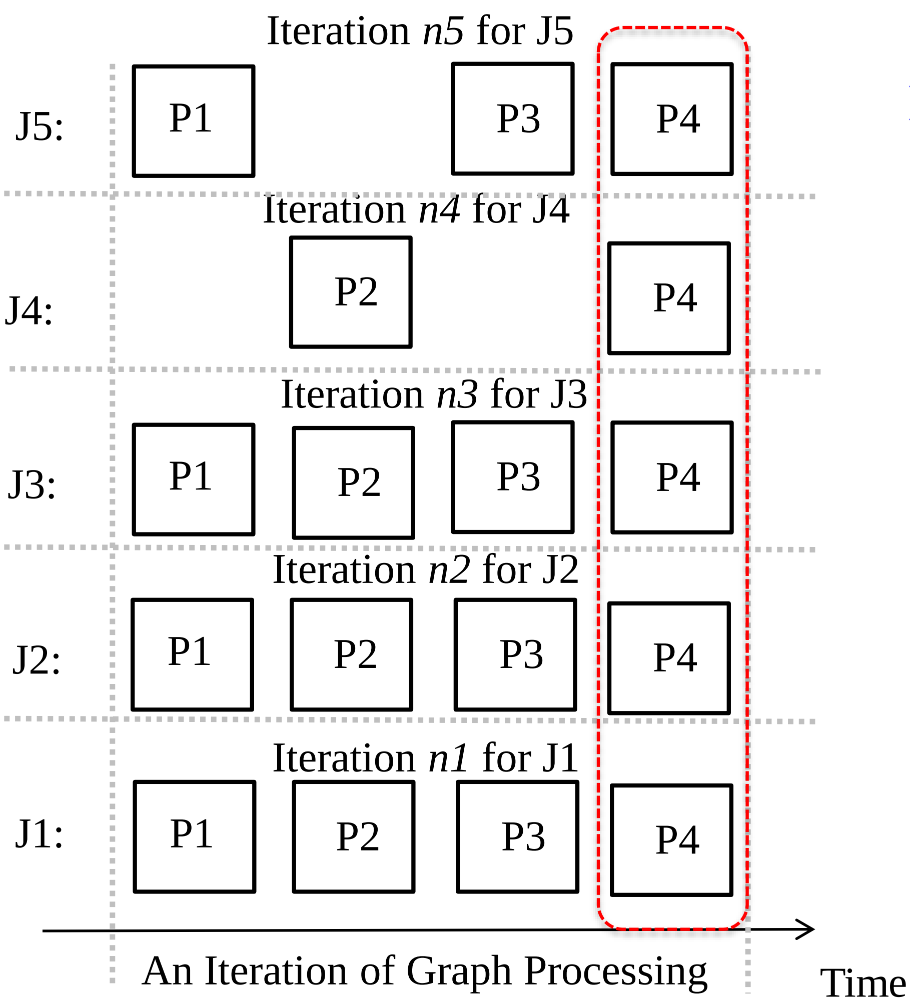
Develop a solution for efficient use of cache/memory and the data access channel to achieve a higher throughput by fully exploiting the spatial/temporal correlations

Motivations: An Example



- Spatial Correlations
  - Load the shared partitions for the related jobs along a common order to provide opportunity to consolidate the accesses to the shared graph structure and store a single copy of the shared data in the cache to serve multiple CGP jobs at the same time.
- Temporal Correlations

Motivations: An Example



- **Spatial Correlations**
  - Load the shared partitions for the related jobs along a common order to provide opportunity to consolidate the accesses to the shared graph structure and store a single copy of the shared data in the cache to serve multiple CGP jobs at the same time.
- **Temporal Correlations**
  - Take into account the temporal correlations, e.g., the usage frequency of the graph partitions, when loading them into the cache

Part 2

Related Work

Existing Graph Processing Systems

Single graph processing

GraphChi | X-Stream | GridGraph | NXgraph | CLIP | …

Existing Graph Processing Systems

Single graph processing

GraphChi | X-Stream | GridGraph | NXgraph | CLIP | …

Mainly focus on single graph processing job

- Higher sequential memory bandwidth
- Better data locality
- Less redundant data accesses
- Less memory consumption

…

Existing Graph Processing Systems

Single graph processing

Concurrent graph processing

GraphChi | X-Stream | GridGraph | NXgraph | CLIP | …

Mainly focus on single graph processing job

- Higher sequential memory bandwidth
- Better data locality
- Less redundant data accesses
- Less memory consumption

…

Part 3

Our Approach:

A Correlations-aware Data-centric Execution Model

Main Goals

Minimize the redundant accessing and storing cost of the shared graph structure data（occupies more than 70% of the total memory of each job）by fully exploiting the spatial/temporal correlations between the CGP jobs

Data-centric LTP Execution Model

- **Traditional approach**:

$$D_1 = (V_1, \boxed{S_1}, E_1, W_1)$$

$$D_2 = (V_2, \boxed{S_2}, E_2, W_2)$$

$$\vdots$$

$$D_J = (V_J, \boxed{S_J}, E_J, W_J)$$

Most graph structure data G=(V, E, W) is the same for different CGP jobs

Data-centric LTP Execution Model

- **Traditional approach**:

$$D_1 = (V_1, \boxed{S_1}, E_1, W_1)$$

$$D_2 = (V_2, \boxed{S_2}, E_2, W_2)$$

$$\vdots$$

$$D_J = (V_J, \boxed{S_J}, E_J, W_J)$$

Most graph structure data G=(V, E, W) is the same for different CGP jobs

- **Load-Trigger-Pushing (denoted by LTP) model**:

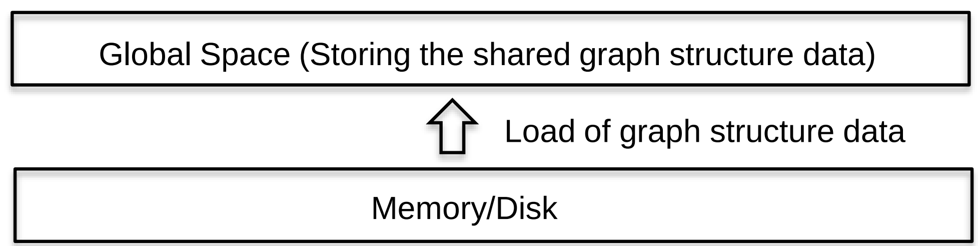
D = (V, S, E, W)

G = (V, E, W), where $G = \cup_i G^i$

$S_1, S_2, \ldots, S_J$

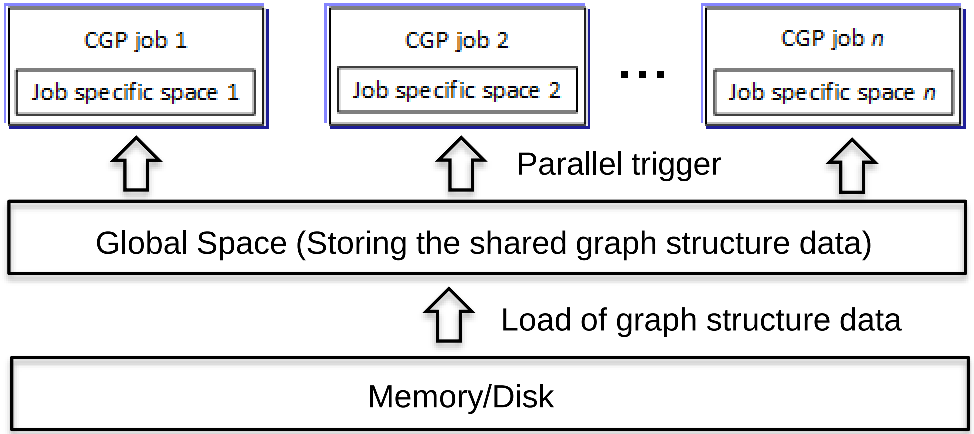
Data-centric LTP Execution Model

- **Load-Trigger-Pushing (denoted by LTP) model**:
  - Graph **L**oading:
    $L^i \leftarrow L(G^i, \cup_{j \in J} S_j^i)$



Data-centric LTP Execution Model

- **Load-Trigger-Pushing (denoted by LTP) model**:
  - Graph **L**oading:
    $$L^i \leftarrow L(G^i, \cup_{j \in J} S_j^i)$$
  - **T**rigger and Parallel Execution:
    $$S^{i_{new}} \leftarrow \cup_{j \in J} T_j(G^i, S_j^i)$$



Data-centric LTP Execution Model

- **Load-Trigger-Pushing (denoted by LTP) model**:
  - Graph **L**oading:

    $$L^i \leftarrow L(G^i, \cup_{j \in J} S^i_j)$$

  - **T**rigger and Parallel Execution:

    $$S^{i_{new}} \leftarrow \cup_{j \in J} T_j(G^i, S^i_j)$$

  - State **P**ushing:

    $$S_j^{new} = \cup_i S_j^{i_{new}}$$

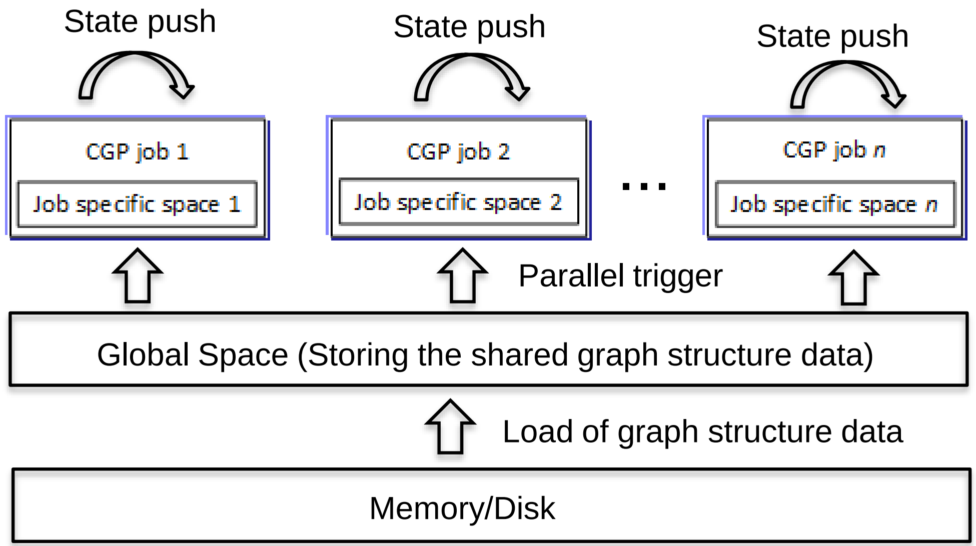
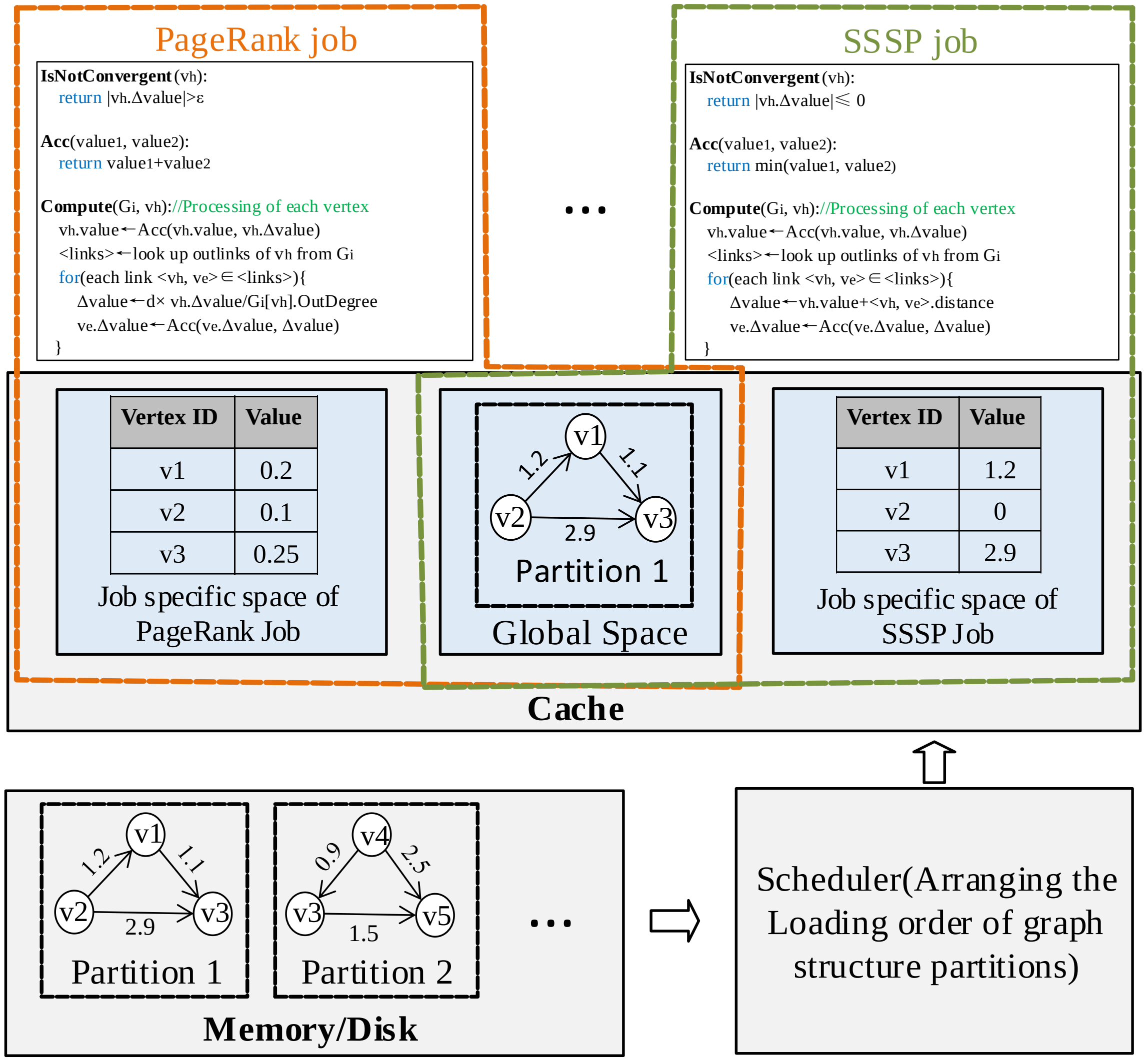
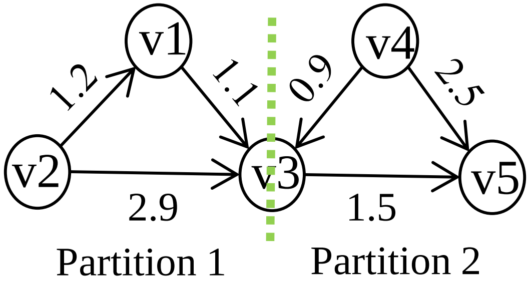


Illustration of Our LTP Model



Implementations: Graph Storage for Multiple CGP Jobs



PageRank Job

Vertex ID	Value
v1	0.2
v2	0.1
v3	0.25

Vertex ID	Value
v3	0.05
v4	0.1
v5	0.3

Private Table Partitions

...

SSSP Job

Vertex ID	Value
v1	1.2
v2	0
v3	2.9

Vertex ID	Value
v3	∞
v4	∞
v5	∞

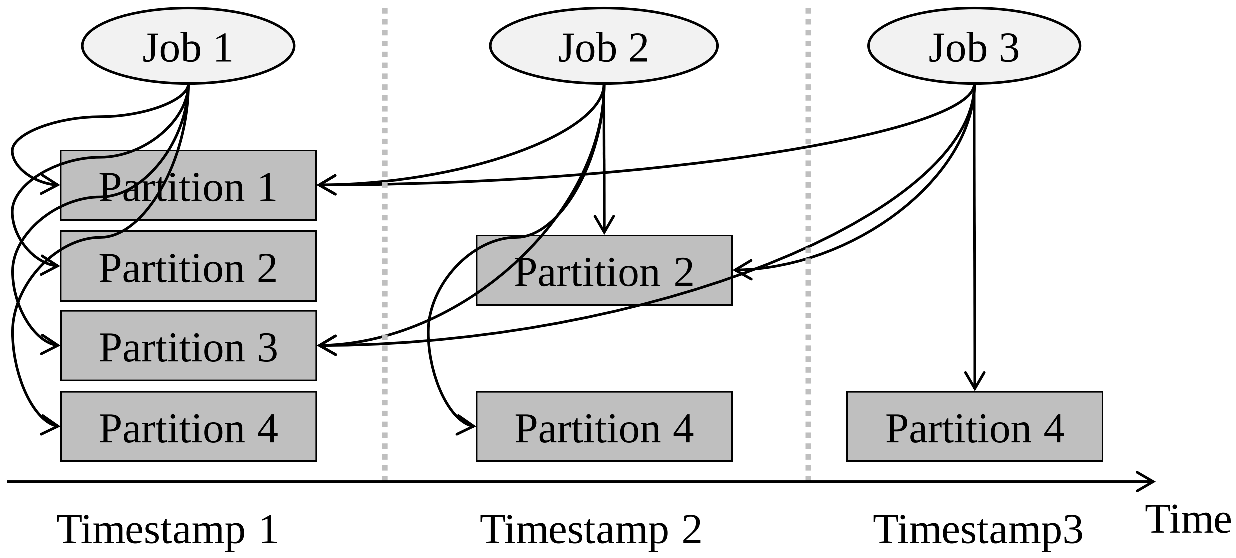
Private Table Partitions

Vertex ID	Edge List	Flag	Master Location	Information Associated with Its Edges
v1	v3	Master	Partition 1	1.1
v2	v1, v3	Master	Partition 1	1.2, 2.9
v3	Ø	Master	Partition 1	Ø

Vertex ID	Edge List	Flag	Master Location	Information Associated with Its Edges
v3	v5	Mirror	Partition 1	1.5
v4	v3, v5	Master	Partition 2	0.9, 2.5
v5	Ø	Master	Partition 2	Ø

Graph Structure Partitions

Implementations：Details to Store Evolving Graph Structure



Implementations：Load of Partitions

(a) There is only one job J1

Partition 1	J1		
Partition 2	J1		
Partition 3	J1		
Partition 4	J1		

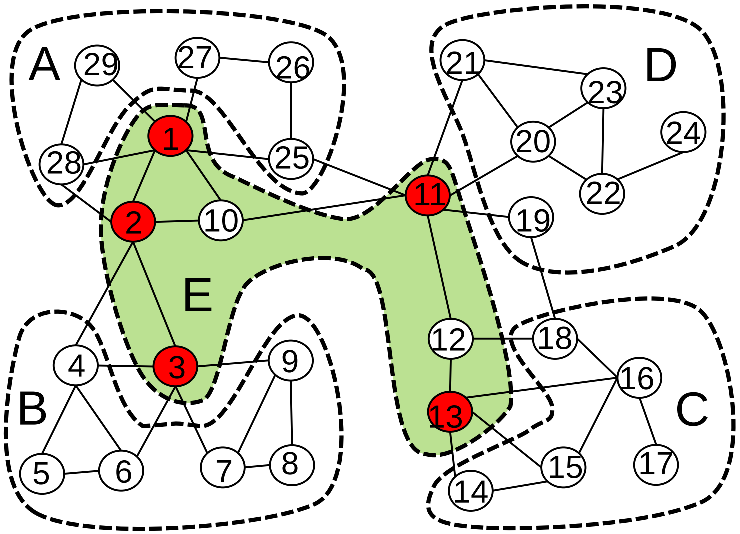
(b) J2 has been submitted

Partition 1	J1	J2	
Partition 2	J1		
Partition 3		J2	
Partition 4	J1		

(c) J3 has been submitted

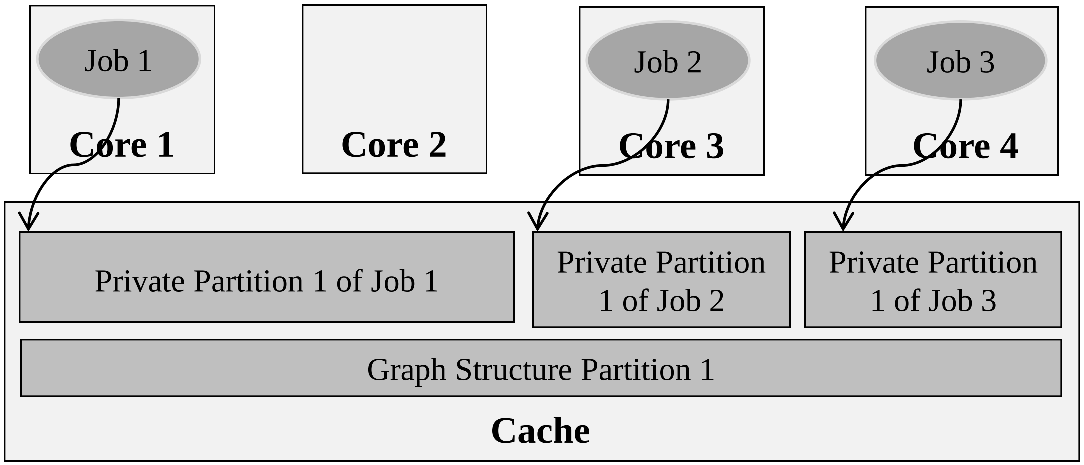
Partition 1	J1	J2	J3
Partition 2	J1		
Partition 3	J1	J2	J3
Partition 4	J1		

Implementations： Load of Partitions

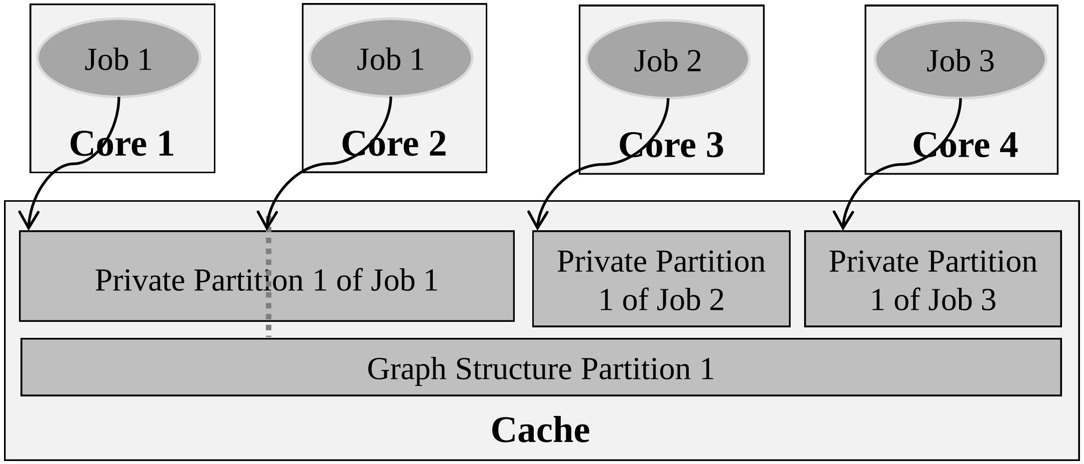


A core-subgraph based scheduling algorithm can be used to maximize the utilization ratio of each partition loaded into the cache

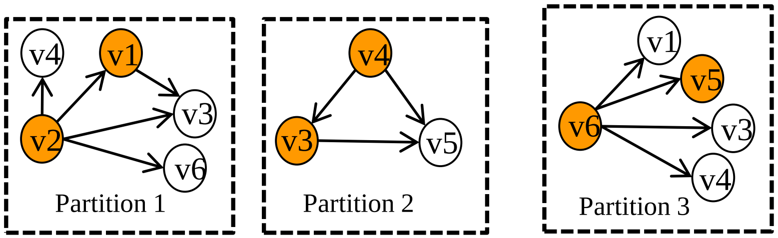
Implementations：Parallel Processing of Graph Partition



Implementations： Parallel Processing of Graph Partition



Implementations：Vertex State Synchronization



Synchronization from **Mirrors** to **Master**

Non-optimized:

P1:v3->P2:v3

P1:v6->P3:v6

P1:v4->P2:v4

⋮

Optimized:

P1:v3->P2:v3

P1:v4->P2:v4

P1:v6->P3:v6

⋮

Synchronization from **Master** to **Mirrors**

Non-optimized:

P2:v3->P1:v3

P2:v3->P3:v3

P2:v4->P1:v4

P2:v4->P3:v4

⋮

Optimized:

P2:v3->P1:v3

P2:v4->P1:v4

P2:v3->P3:v3

P2:v4->P3:v4

⋮

Part 4

Performance Evaluation

Evaluation

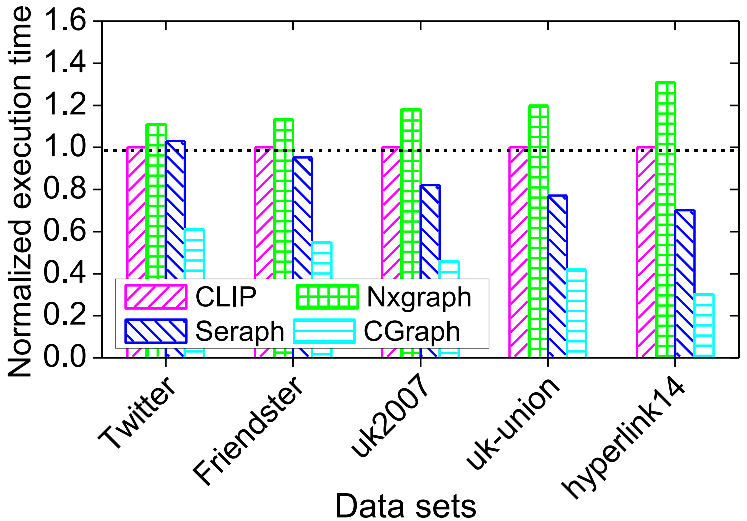
Experimental setup

- **Machine information**
  -CPU: 4-way 8-core Intel Xeon CPU E5-2670; each CPU has 20 MB LLC
  -Main Memory: 64 GB
- **Typical graph algorithms**
  -PageRank, SSSP, SCC, BFS
- **Data sets**

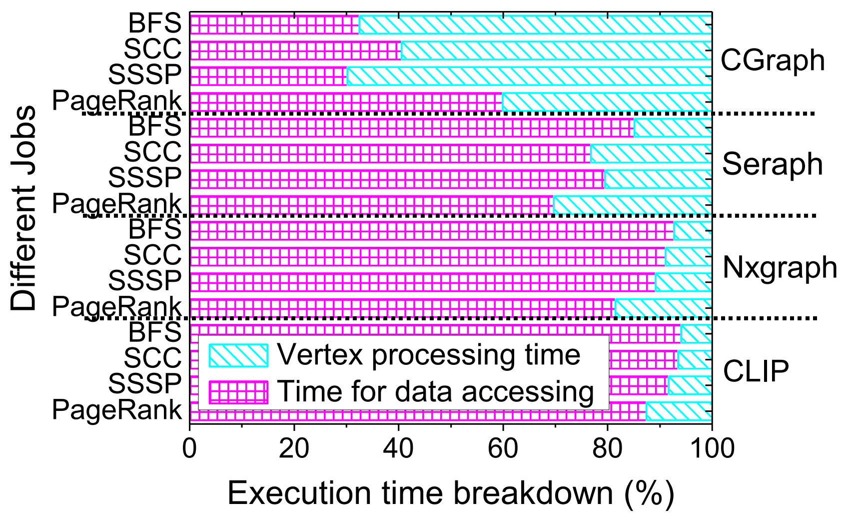
Properties of data sets

Data sets	Vertices	Edges	Sizes
Twitter	41.7 M	1.4 B	17.5 GB
Friendster	65 M	1.8 B	22.7 GB
uk2007	105.9 M	3.7 B	46.2 GB
uk-union	133.6 M	5.5 B	68.3 GB
hyperlink14	1.7 B	64.4 B	480.0 GB

Evaluation

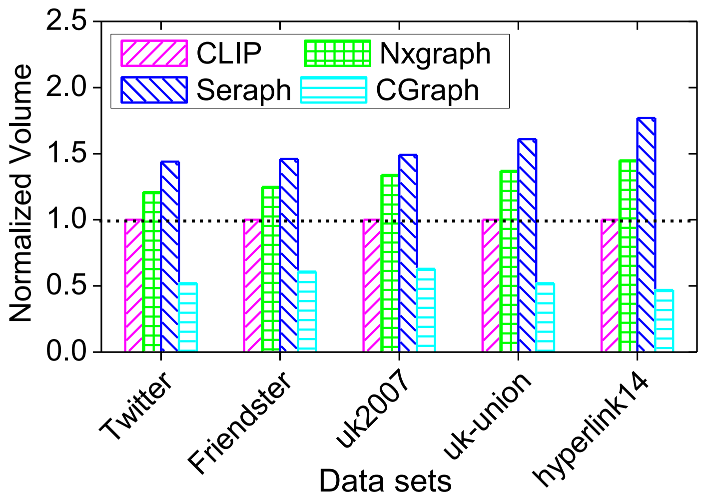


Total execution time for the four jobs with different solutions

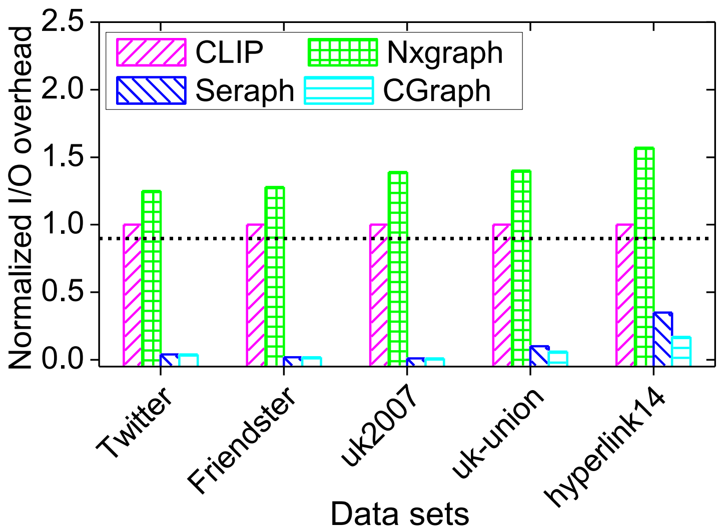


Execution time breakdown of different jobs on hyperlink14

Evaluation

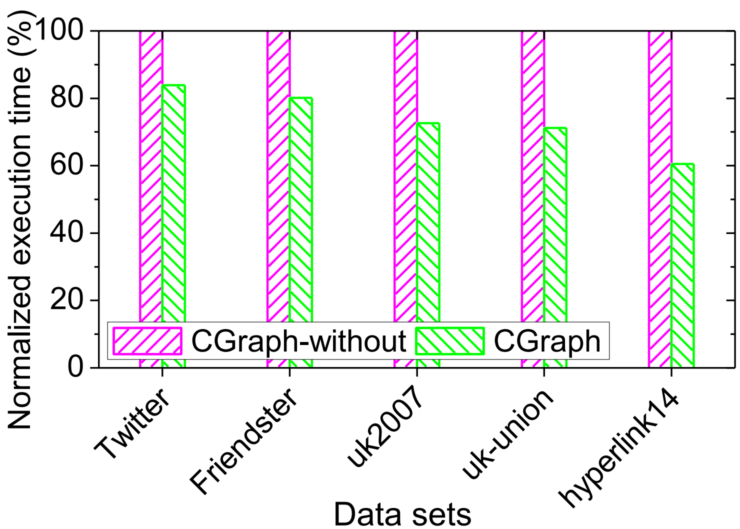


Volume of data swapped into the cache for the four jobs

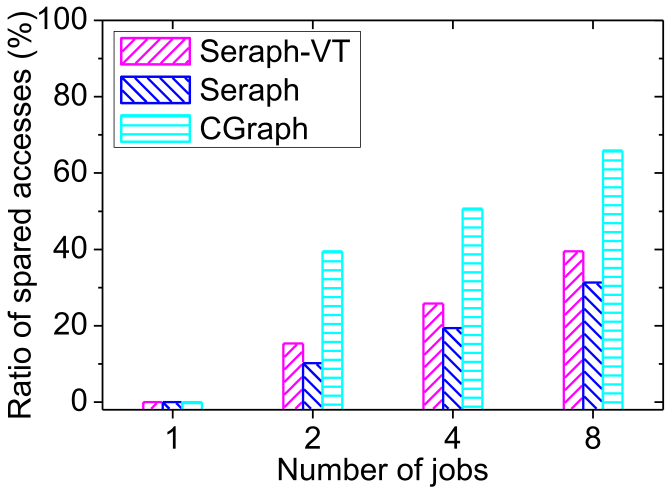


I/O overhead for the four jobs with different solutions

Evaluation



Execution time for the four jobs without/with our scheduler



Ratio of spared accessed data on hyperlink14

Part 5

Conclusions

Conclusions

- What **CGraph** brings in graph processing
  - **Analysis of temporal/spatial correlations in concurrent graph processing**
  - **A novel data-centric LTP model for concurrent graph processing**
  - **A core-subgraph based scheduling scheme**
- Future work
  - **How to further optimize the approach for evolving graph analysis**
  - **How to ensure QoS for some real-time CGP jobs**
  - **How to extend it to a distributed platform and also heterogeneous platform consisting of GPU, FPGA and even ASIC for higher throughput.**

Thanks!