Parallel Graph Computations: Foundation and Practice

Wenfei Fan

University of Edinburgh

Backgrounds

What is the right parallel graph computation model?
- **GRAPE**: automatically parallelizing graph computation
- **AAP**: adaptive asynchronous parallel

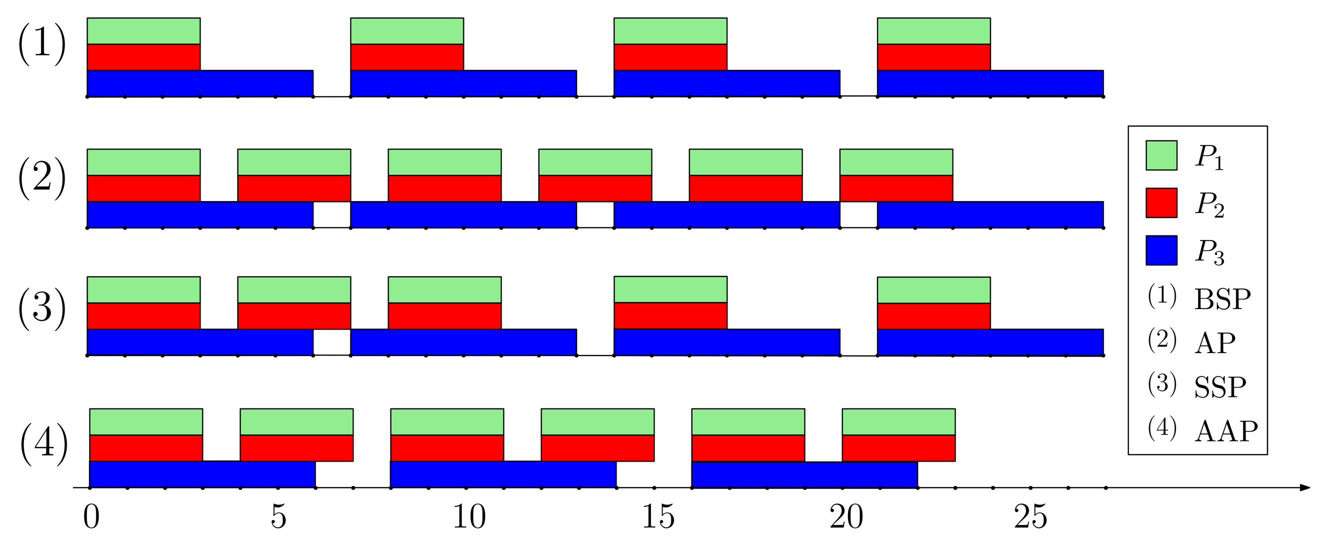
How to partition the graph under various circumstances?
- Dynamically re-partitioning a graph when adding or removing processors
- Partitioning the graph based on application scenario
- Incrementalizing existing graph partitioners

How to efficiently carry out local computation?
- Graph contraction: making big graphs small
- Incrementalize existing graph algorithms

Parallel graph computation model

GRAPE programming model: plug and play
- Partial evaluation **PEval**
- Incremental evaluation **IncEval**
- Assemble partial results

AAP: each worker maintains parameters and dynamically adjusts:
- its progress relative to other workers
- the staleness of its accumulated changes



Unique properties
- Dynamic model switch: reduce the total rounds
- Programming simplicity
- Ready to *optimally* simulate other platforms:
  - **MapReduce**, **BSP**,**AP**, **SSP** and **PRAM**
- Convergence guarantee: Church-Rosser

Outperform the state-of-the-art systems
- Outperforming **BSP**, **AP**,**SSP** by 4.3X, 14.7X and 4.7X on average
- 30X times faster than Petuum on collaborative filtering

Industrialized as GraphScope in Alibaba:
Open source: https://github.com/alibaba/GraphScope

Partitioning

Dynamic scaling
- Updating the partition $\Pi(n+k)$ from $\Pi(n)$ when add k servers with minimal migration cost, subject to balance factor b and replication factor f
- Tri-criteria optimization problem: **NP**-complete
- Approximation algorithms: extending consistent hashing on graphs

Application driven partitioning
- Finding partition $\Pi(n)$ to minimize the cost $\mathcal{C}(\mathcal{A})$ of algorithm $\mathcal{A}$
- Learned cost model: unifying computation and communication
- Hybrid partitioners: combining edge-cut and vertex-cut

Incrementalization of partition algorithms
- Tri-criteria optimization problem: **NP**-complete
- A generic approach based on fixpoint model

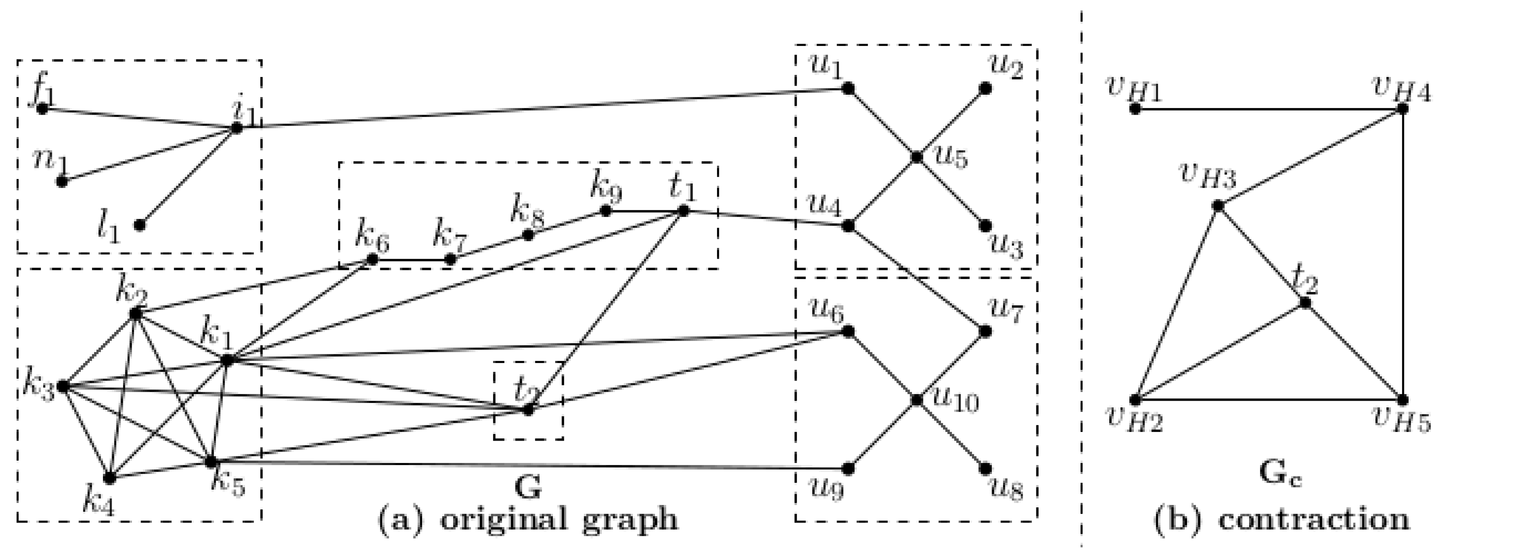
Partition transparency for algorithm $\mathcal{A}$
- Conditions: monotonic + robustness under vertex cut
- Example: **SSSP**, **WCC**, **PageRank**, etc.

$\mathcal{A}$ works for any partition without requiring any change to $\mathcal{A}$

Graph contraction

Graph contraction: making big graphs small
- Genericness: support different classes of queries at the same time
- Losslessness: compute exact answer

Contraction scheme $\langle f_C, \mathcal{S}, f_D\rangle$
- Contraction function f_C: subgraphs to supernodes
- Synopsis synopsis functions $\mathcal{S}$: annotate supernodes
- Decontraction function f_D: restore supernodes to subgraphs



Algorithms:
- Contraction following deterministic order
- Bounded Incremental contraction

Incrementalization

Incremental algorithms: $\mathcal{A}(G \oplus \Delta G) = \mathcal{A}(G) \oplus \mathcal{A}_\Delta(G, \Delta G)$

Incrementalize batch algorithms
- Generating an incremental algorithm $\mathcal{A}_\Delta$ from the batch algorithm $\mathcal{A}$
- Reusing the original logic and data structures
- Providing performance guarantee

Fixpoint model for batch algorithm $\mathcal{A}$:

$$(D_{\mathcal{A}}^{t+1},\ H_{\mathcal{A}}^{t+1}) = f_{\mathcal{A}}(D_{\mathcal{A}}^{t},\ Q,\ G,\ H_{\mathcal{A}}^{t})$$

- Status $D_{\mathcal{A}}$, status varibale $H_{\mathcal{A}}$, and step function $f_{\mathcal{A}}$

Incrementalization
- Initial scope function h: initializing status $(D_{\mathcal{A}}^0, H_{\mathcal{A}}^0)$ from ΔG and previous result
- Reusing the same $D_{\mathcal{A}}$, $H_{\mathcal{A}}$ and $f_{\mathcal{A}}$
- Reiterating from $(D_{\mathcal{A}}^0, H_{\mathcal{A}}^0)$ until convergence

Incrementalizability: every fixpoint algorithm $\mathcal{A}$ is incrementalizable
Relative boundedness: correct and bounded h + Church-rosser $\mathcal{A}$

Delivered

11 top-tier publications :
- 2021: SIGMOD [1, 2], VLDB [3, 4], TKDE [5]
- Past: 2020 [6, 7, 8, 9], 2019 [10], and 2018 [11]

To develop a package of solutions for parallel graph computations

Publications

[1] Wenfei Fan et al. "Making Graphs Compact by Lossless Contraction". In: *SIGMOD*. 2021.

[2] Wenfei Fan et al. "Incrementalizing Graph Algorithms". In: *SIGMOD*. 2021.

[3] Wenfei Fan et al. "GraphScope: a unified engine for big graph processing". In: *Proceedings of the VLDB Endowment* 14.12 (2021), pp. 2879–2892.

[4] Jingbo Xu et al. "GraphScope: a one-stop large graph processing system". In: *Proceedings of the VLDB Endowment* 14.12 (2021), pp. 2703–2706.

[5] Wenfei Fan et al. "Graph Algorithms with Partition Transparency". In: *TKDE* (2021), accpted.

[6] Wenfei Fan et al. "Adaptive Asynchronous Parallelization of Graph Algorithms". In: *TODS (Invited)* (2020).

[7] Grace Fan et al. "Extending Graph Patterns with Conditions". In: *SIGMOD*. 2020.

[8] Wenfei Fan et al. "Application Driven Graph Partitioning". In: *SIGMOD*. 2020.

[9] Wenfei Fan et al. "Incrementalization of graph partitioning algorithms". In: *PVLDB* 13.8 (2020), pp. 1261–1274.

[10] Wenfei Fan et al. "Dynamic Scaling for Parallel Graph Computations". In: *PVLDB* 12.8 (2019), pp. 877–890.

[11] Wenfei Fan et al. "Adaptive Asynchronous Parallelization of Graph Algorithms". In: *SIGMOD*. 2018, pp. 1141–1156.