PDTL: Parallel and Distributed Triangle Listing for Massive Graphs

Ilias Giechaskiel[1] George Panagopoulos Eiko Yoneki

UNIVERSITY OF CAMBRIDGE

[1]Currently at University of Oxford

September 3, 2015

Key Points

Motivation

- Graphs are becoming massive-scale
  - Billions of edges and vertices
- Triangle counting has important applications
  - Metrics for connectivity, density, quality of nodes

Outcomes

- PDTL: Parallel and Distributed Triangle Listing for Massive Graphs
  - Provable Memory, Network, CPU, I/O bounds
  - Relies on and amends MGT (Hu et al., SIGMOD 2013)
  - Outperforms state-of-the-art by $2-4\times$ using fewer resources
  - Reasonable even compared to fast in-memory algorithms
- Bring external-memory considerations to distributed systems!

Triangle Listing

Definition

Given a simple undirected graph $G = (V, E)$, list all $\{v_1, v_2, v_3\}$ such that $v_i \in V$ and $(v_i, v_j) \in E$ exactly once

Motivation

- Shortest non-trivial cycles and cliques
- Blackbox for density and connectivity metrics
  - Clustering coefficient, transitivity ratio
  - Triangular connectivity, k-truss
- Spam and fake account detection
- Massive graphs, so in-memory algorithms not sufficient

Orientation

Definition

Given $G = (V, E)$, define $G^* = (V, E^*)$ with $(u, v) \in E^*$ iff $(u, v) \in E$ and either $d(u) < d(v)$ or $d(u) = d(v)$ with $u < v$

Motivation

- Better asymptotic runtime
- Identify $\{u, v, w\}$ when $u \prec v \prec w$ with (u, v, w)
  - u is the *cone vertex*
  - (v, w) is the *pivot edge*

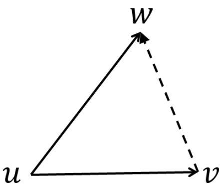

Figure: Oriented triangle

Massive Graph Triangulation (Hu et al., SIGMOD 2013)

Key Idea

1. Load next set of edges S of G^* into memory
   - Fill proportion of available memory M
   - "All-or-nothing" requirement for each vertex
2. For all $u \in V$
   - 2.1 Build hash structures on $N(u)$
   - 2.2 Find all triangles with cone u and pivot in S
   - 2.3 Release the hash structures

Our Modifications

- Work on sorted arrays, not sets
- Ignore all-or-nothing
- Same complexity!

Parallel and Distributed Triangle Listing (PDTL)

Target Environment

R machines, P processors/machine, M memory/processor

Key Insights

- Graph partitioning requires random accesses
  - Or excessive network traffic
- Duplicate graph across *each* machine
  - But make I/O-efficient accesses
- Run (part of) MGT on each processor
  - Each processor responsible for different set of (contiguous) pivot edges
  - Small memory footprint

Optimizations

- Orientation runs once and can be parallelized
- Load balance using number of in-edges

PDTL Protocol

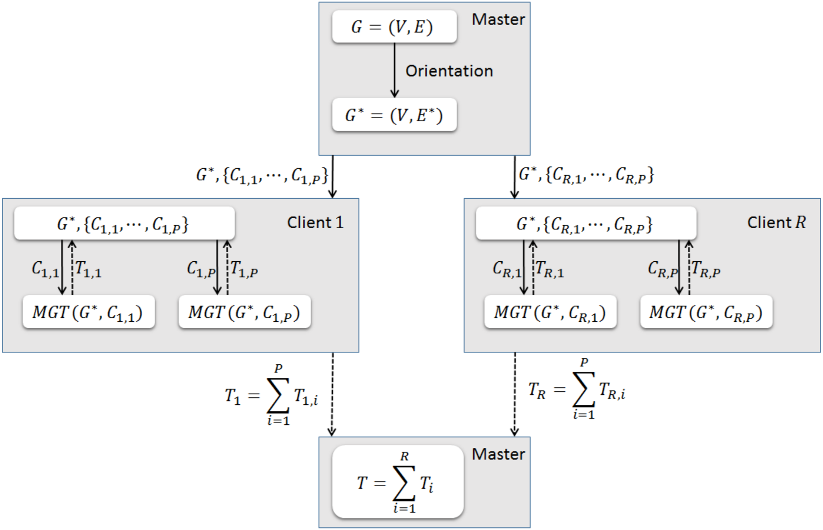

Figure: PDTL protocol

PDTL Analysis

Complexity

For triangle counting on all processors, with α as the arboricity:

- $\Theta\left(RP + R|E|\right)$ Network traffic
- $\mathcal{O}\left(RP|E| + \frac{|E|^2}{M} + \alpha|E|\right)$ CPU computations
- $\mathcal{O}\left(RP\frac{|E|}{B} + \frac{|E|^2}{MB}\right)$ I/Os

Key Decisions

- Orientation, load balancing parallelized and never re-computed
- Computation starts before transfers have finished
- Everything is included in analysis and timing

Evaluation: Introduction

Set-Up

- **Amazon EC2**: 4× `c3.8xlarge`, each with 32 CPUs, 60GB memory
- **Local Cluster**: 8 virtual nodes, each with 4 cores, 40GB memory
- **Local Multicore**: 24-core machine, 256GB memory

Datasets

Graph	Nodes	Edges	Triangles	Size
soc-LiveJournal1	4.8M	68.0M	285,730,264	365MB
com-Orkut	3.1M	117.2M	627,584,181	917MB
Twitter	61.6M	1.5B	34,824,916,864	9.4GB
Yahoo	1.4B	6.6B	85,782,928,684	59GB
RMAT-26	67.1M	1.1B	51,559,452,522	8.4GB
RMAT-27	134.2M	2.1B	114,007,006,286	17GB
RMAT-28	268.4M	4.3B	251,913,686,661	34GB
RMAT-29	536.9M	8.6B	556,443,109,053	68GB

Evaluation: PDTL Multicore Scalability

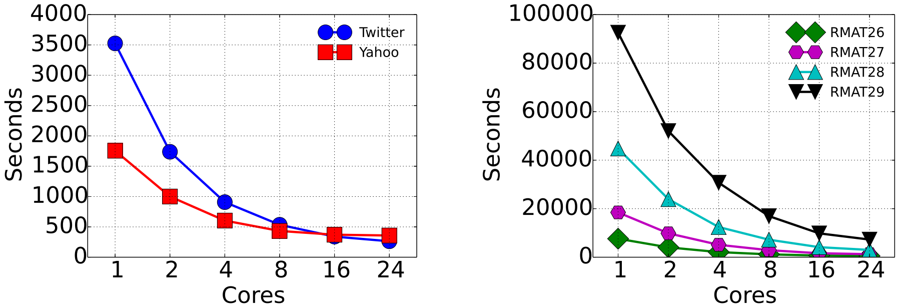

Figure: PDTL Total Time (without Orientation) in Local Multicore

Evaluation: PDTL EC2 Scalability

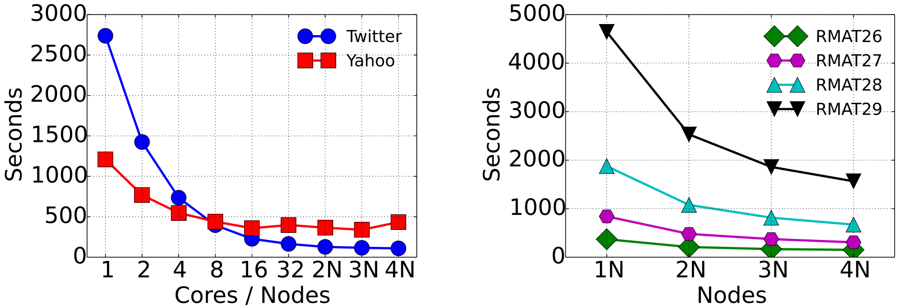

Figure: PDTL Total Time (without Orientation) in Amazon EC2

Evaluation: Load Balancing

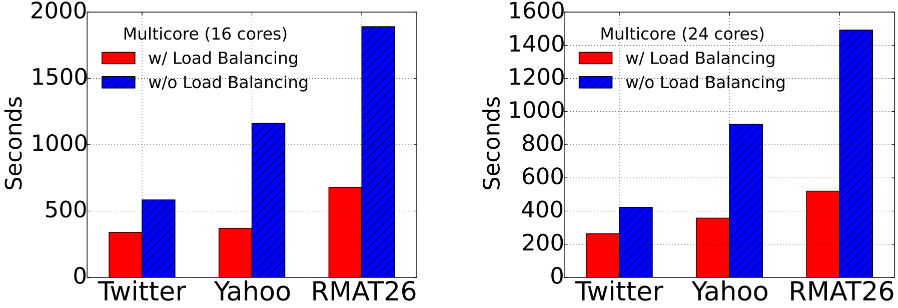

Figure: PDTL Load Balancing

Evaluation: Orientation

Pre-processing time in seconds

Graphs	d^*_{max}	PDTL	PowerGraph	OPT
LiveJ1	687	1.4	-	106.8
Orkut	535	3.6	25.7	43.6
Twitter	4,102	32.8	233.2	437.6
Yahoo	1,540	235.6	-	-
RMAT-26	2,964	29.3	213.0	910.3

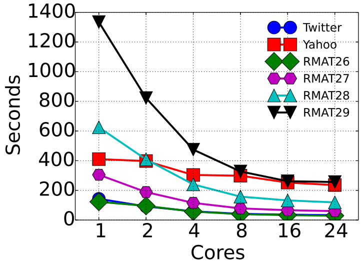

Figure: PDTL Multicore Orientation

Evaluation: PDTL vs Other Frameworks on Twitter

Distributed Frameworks

- PATRIC (Arifuzzaman et al., CIKM 2013) 564s, 200 cores, 4GB/core
  - PDTL faster using 8 cores and 1GB memory/core
  - PDTL 4$\times$ faster using 96 cores and 1GB memory/core
- CTTP (Park et al., CIKM 2014) 5520s, 40 nodes, 4GB/node
  - PDTL with 1 core and 1GB memory (MGT): 2800s

In-Memory Frameworks

- PDTL 141.8s using 128 cores and 128GB memory total
- (Sevenich et al., SNAKDD 2014)
  - 144.8s using 16*2 Hyperthread Cores @ 2.2GHz and 264GB of memory
  - 91.5s using 128*8 Hyperthread Cores @ 3.6GHz and 4TB of memory
- (Shun and Tangwongsan, ICDE 2015)
  - 55.9s using 40*2 Hyperthread Cores @ 2.4GHz and 256GB of memory
  - 78.9s using 64 Cores @ 2.4 GHz and 188GB of memory

Evaluation: PDTL More Comparisons

PDTL and (Kim et al., SIGMOD 2014) performance in local multicore

Graph	PDTL		OPT		Speed-up	
	Orient (s)	Calc (s)	DB (s)	Calc (s)	Calc	Total
Twitter	32.8	262.9	235.2	437.6	1.66×	2.28×
RMAT-26	29.3	520.4	910.3	1011.2	1.94×	3.50×

PDTL and (Gonzalez et al., OSDI 2012) performance in Amazon EC2

Graph	PDTL		PowerGraph		Speed-up	
	Calc (s)	Total (s)	Calc (s)	Total (s)	Calc	Total
Twitter	88.5	141.8	97.3	330.5	2.33×	1.86×
Yahoo	323.9	669.4	OOM	OOM	NA	NA
RMAT-26	138.6	180.6	176.7	389.7	2.16×	1.77×
RMAT-29	1533.5	1821.2	OOM	OOM	NA	NA

Conclusions

Key Contributions

- Novel PDTL algorithm
  - Framework suitable for variety of environments
  - Provable CPU, I/O, Memory, and Network efficiency
- Improved state-of-the-art by $2 - 4\times$ using fewer resources
  - Experimentally showed that algorithm is scalable
  - Orientation, balancing optimizations
  - Reasonable even compared to fast in-memory algorithms
- Focused on disk accesses in a distributed system

Key Questions

- What other problems can benefit from external-memory, distributed algorithms?
- Your questions?
  - Reach me at ilias.giechaskiel@cs.ox.ac.uk

Background

I/O Analysis

Let B denote the disk block size, and M the memory size

- Reading takes $scan(N) = \Theta(N/B)$ I/Os
  - Orientation takes $scan(|E|)$ I/Os and $\mathcal{O}(|E|)$ CPU
- Sorting takes $sort(N) = \mathcal{O}\left(N/B \log_{M/B} N/B\right)$ I/Os
  - Pre-sorting graph takes $sort(|E|)$ I/Os and $\mathcal{O}(|E| \log |E|)$ CPU

Arboricity

The arboricity $\alpha(G)$ of a graph G is the minimum number of edge-disjoint forests needed to cover the edges of G. It satisfies

$$\alpha(G) = \max_{G' \subseteq G} \left\lceil \frac{|E_{G'}|}{|V_{G'}| - 1} \right\rceil$$

where G' ranges over all subgraphs of G with ≥ 2 vertices

Bibliography I

Shaikh Arifuzzaman, Maleq Khan, and Madhav Marathe. "PATRIC: A Parallel Algorithm for Counting Triangles in Massive Networks". In: CIKM 2013.

Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. "R-MAT: A recursive model for graph mining". In: SIAM 2004.

Joseph E. Gonzalez et al. "PowerGraph: Distributed Graph-parallel Computation on Natural Graphs". In: OSDI 2012.

Xiaocheng Hu, Yufei Tao, and Chin-Wan Chung. "Massive Graph Triangulation". In: SIGMOD 2013.

Jinha Kim et al. "OPT: A New Framework for Overlapped and Parallel Triangulation in Large-scale Graphs". In: SIGMOD 2014.

Bibliography II

- Haewoon Kwak et al. "What is Twitter, a social network or a news media?" In: WWW 2010.
- Jure Leskovec. *SNAP: Stanford Large Network Dataset Collection*. http://snap.stanford.edu/data/ 2009.
- Ha-Myung Park et al. "MapReduce Triangle Enumeration With Guarantees". In: CIKM 2014.
- Martin Sevenich et al. "Fast In-Memory Triangle Listing for Large Real-World Graphs". In: SNAKDD 2014.
- J. Shun and K. Tangwongsan. "Multicore triangle computations without tuning". In: ICDE 2015.
- Yahoo! Webscope. *Yahoo! AltaVista Web Page Hyperlink Connectivity Graph, circa 2002*. http://webscope.sandbox.yahoo.com/ 2009.
- Zhi Yang et al. "Uncovering Social Network Sybils in the Wild". In: KDD 2014.