Massivizing Graph Processing = Scalable, High-Performance, Reliable, etc., Yet Efficient Graph-Processing Systems

Speakers also include: Dick Epema, Tim Hegeman, Wing-Lung Ngai, and Stijn Heldens.

Presentation developed jointly with Ana Lucia Varbanescu.
Several slides developed jointly with Yong Guo.
Logistics

- Afternoon: 16:00-18:30
- Room: DW-PC-3
Graphs Are at the Core of Our Society: The LinkedIn Example

The State of LinkedIn

400 million Q3 2015

A very good resource for matchmaking workforce and prospective employers

Vital for your company’s life, as your Head of HR would tell you

Vital for the prospective employees

Tens of “specialized LinkedIns”: medical, mil, edu, gov, ...

Sources: Vincenzo Cosenza, The State of LinkedIn, via Christopher Penn, http://www.shiftcomm.com/2014/02/state-linkedin-social-media-dark-horse/
LinkedIn’s Service/Ops Analytics

The State of LinkedIn

3-4 new users every second

By processing the graph: opinion mining, hub detection, etc.

100+ million questions of customer retention, of (lost) customer influence, of ...
Why Analytics?

The State of LinkedIn

3-4 new users every second

but fewer visitors (and page views)

Periodic and/or continuous analytics at full scale

Sources: Vincenzo Cosenza, The State of LinkedIn, http://vincos.it/the-state-of-linkedin/
How to do Analytics? Graph Processing @large

A Graph Processing Platform

ETL (Extraction, Transf, Loading)

Active Storage (filtering, compression, replication, caching)

Distribution to processing platform

Algorithm

Interactive processing not considered in this presentation.
Streaming not considered in this presentation.
Graph-processing is at the core of our society

The Data Deluge vs. Analytics

Graph processing @large
Which to select? What to tune? What to re-design?
A performance comparison of graph-processing systems
Take-home message
The data deluge: large-scale graphs

Social network
~1 billion vertices
~100 billion connections

Web graph
~50 billion pages
~1 trillion hyperlinks

Brain network
~100 billion neurons
~100 trillion connections

Source: Smith, CHI’10; Blog webpage; Gigandet et al., PLoS ONE 3(12)]
The data deluge: graphs everywhere!

LinkedIn: 400M users, ??? edges

Tweet: 270M MAU, 200+ avg followers, >54B edges

Yahoo!: 1.2B MAU, 0.8B DAU, 200+ avg followers, >240B edges

Friendster: ???

TU Delft
The data deluge: graphs everywhere!

LinkedIn: 1.2M followers, 132k employees
company/day: 40-60 posts, 500-700 comments

Oracle: 1.2M followers, 132k employees

Twitter: 270M MAU, 200+ avg followers, >54B edges

Facebook: 1.2B MAU, 0.8B DAU, 200+ avg followers, >240B edges

friendster: 1.2M followers, 132k employees

Delft University of Technology
The data deluge vs. Analytics

Data-intensive workload
10x graph size ➔ 100x—1,000x slower

LinkedIn
270M MAU
Oracle 1.2M followers
1.2M followers, 132k employees

Twitter
1.2B MAU 0.8B DAU
200+ avg followers
>240B edges
The data deluge vs. Analytics

Data-intensive workload
10x graph size $\Rightarrow$ 100x—1,000x slower

Compute-intensive workload
more complex analysis $\Rightarrow$ ?x slower

Oracle 1.2M followers
270M MAU
>240B edges

Linkedin

Friendster
The data deluge vs. Analytics

Data-intensive workload
10x graph size → 100x—1,000x slower

Compute-intensive workload
more complex analysis → ?x slower

Dataset-dependent workload
unfriendly graphs → ??x slower
Your network is so large...

Sorry, but your network is too large to be computed, we are working to increase the limit, stay tuned!
The “sorry, but…” moment

Supporting multiple users
10x number of users ➔ ???inx slower
Data-intensive workload
10x graph size $\Rightarrow$ 100x—1,000x slower

Compute-intensive workload
more complex analysis $\Rightarrow$ ?x slower

Dataset-dependent workload
unfriendly graphs $\Rightarrow$ ??x slower

Supporting multiple users
10x number of users $\Rightarrow$ ?????x slower
The data deluge vs. Analytics

The Big Problem: Graph Processing @large

What would you do?
Graph-processing is at the core of our society
The data deluge vs. Analytics

Graph Processing @Large

Which to select? What to tune? What to re-design?
A performance comparison of graph-processing systems
Take-home message
Graph Processing @large

A Graph Processing Platform

Ideally, N cores/disks → Nx faster

Distribution to processing platform

Parallel/Distributed/Heterogeneous

Ideally, N cores/disks → Nx faster

Interactive processing not considered in this presentation. Streaming not considered in this presentation.
Graph processing systems

Performance

- Dedicated Systems
- Generic
- Custom

Development Effort

TU Delft
Delft University of Technology
System diversity (CPU-based)
System diversity (GPU-enabled)

- medusa-gpu
  Medusa: Simplified Graph Processing on GPUs

- mapgraph
  Massively Parallel Graph processing on GPUs

- Gunrock
  High-performance Graph Primitives on GPU

- TOTEM
- VertexAPI2

Dedicated Systems

Generic
Graph-Processing Platforms

Platform: the combined hardware, software, and programming system that is being used to complete a graph processing task

Which to choose?
What to tune?
What to re-design?
The data deluge vs. Analytics

Which to choose?
What to tune?
What to re-design?

What would you do?
Graph-processing is at the core of our society
The data deluge vs. Analytics
Graph processing @large

Graphalytics:
Which system to select? What to tune? What to re-design?

A performance comparison of graph-processing systems
Take-home message
What is the performance of graph-processing platforms?

- **Graph500**
  - Single application (BFS), Single class of synthetic datasets

- **Few existing platform-centric comparative studies**
  - Prove the superiority of a given system, limited set of metrics

- **GreenGraph500, GraphBench, XGDBench**
  - Issues with representativeness, systems covered, metrics, ...
What is the performance of graph-processing platforms?

- **Metrics Diversity**
- **Graph Diversity**
- **Algorithm Diversity**

Graphalytics = comprehensive benchmarking suite for graph processing across many platforms

Why is “LDBC” here?
Graphalytics = A Challenging Benchmarking Process

Methodological challenges
Challenge 1. Evaluation process
Challenge 2. Selection and design of performance metrics
Challenge 3. Dataset selection and analysis of coverage
Challenge 4. Algorithm selection and analysis of coverage

Practical challenges
Challenge 5. Scalability of evaluation, selection processes
Challenge 6. Portability
Challenge 7. Result reporting

LDBC council.org
Graph processing systems

- Systems for graph processing
- Separate users from backends
- Think Totem, Medusa, ...
- Think Giraph, GraphLab, PGX

Custom

- Specify application
- Choose the hardware
- Implement & optimize
- Think Graph500

Dedicated Systems

- Use existing large scale distributed systems
- Mapping is difficult
- Parallelism is “free”
- Think MapReduce

Generic

Graphalytics: unified view and benchmarking of tens of systems, with little effort
Graphalytics, in a nutshell

- An LDBC benchmark*
- Advanced benchmarking harness
- Many classes of algorithms used in practice
- Diverse real and synthetic datasets
- Diverse set of experiments representative for practice
- Granula for manual choke-point analysis
- Modern software engineering practices
- Supports many platforms
- Enables comparison of community-driven and industrial systems

http://graphalytics.ewi.tudelft.nl
https://github.com/tudelft-atlarge/graphalytics/
Benchmarking Harness

Iosup et al. LDBC Graphalytics: A Benchmark for Large Scale Graph Analysis on Parallel and Distributed Platform (submitted).
Graphalytics = Representative Classes of Algorithms and Datasets

- 2-stage selection process of **algorithms** and datasets

<table>
<thead>
<tr>
<th>Class</th>
<th>Examples</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph Statistics</td>
<td>Diameter, Local Clust. Coeff., PageRank</td>
<td>20</td>
</tr>
<tr>
<td>Graph Traversal</td>
<td>BFS, SSSP, DFS</td>
<td>50</td>
</tr>
<tr>
<td>Connected Comp.</td>
<td>Reachability, BiCC, Weakly CC</td>
<td>10</td>
</tr>
<tr>
<td>Community Detection</td>
<td>Clustering, Nearest Neighbor, Community Detection w Label Propagation</td>
<td>5</td>
</tr>
<tr>
<td>Other</td>
<td>Sampling, Partitioning</td>
<td>&lt;15</td>
</tr>
</tbody>
</table>

+ weighted graphs: Single-Source Shortest Paths (~35%)

Graphalytics = Distributed Graph Generation w DATAGEN

- Rich set of configurations
- More diverse degree distribution than Graph500
- Realistic properties, e.g., clustering coefficient and assortativity

Graphalytics

Person Generation  Edge Generation  “Knows” graph serialization  Activity Generation  Activity serialization

Level of Detail
### Graphalytics = Diverse Automated Experiments

<table>
<thead>
<tr>
<th>Category</th>
<th>Experiment</th>
<th>Algo.</th>
<th>Data</th>
<th>Nodes/Threads</th>
<th>Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Baseline</strong></td>
<td>Dataset variety</td>
<td>BFS, PR</td>
<td>All</td>
<td>1</td>
<td>Run, norm.</td>
</tr>
<tr>
<td></td>
<td>Algorithm variety</td>
<td>All</td>
<td>R4(S), D300(L)</td>
<td>1</td>
<td>Runtime</td>
</tr>
<tr>
<td><strong>Scalability</strong></td>
<td>Vertical vs. horiz.</td>
<td>BFS, PR</td>
<td>D300(L), D1000(XL)</td>
<td>1—16/1—32</td>
<td>Runtime, S</td>
</tr>
<tr>
<td></td>
<td>Weak vs. strong</td>
<td>BFS, PR</td>
<td>G22(S)—G26(XL)</td>
<td>1—16/1—32</td>
<td>Runtime, S</td>
</tr>
<tr>
<td><strong>Robustness</strong></td>
<td>Stress test</td>
<td>BFS</td>
<td>All</td>
<td>1</td>
<td>SLA met</td>
</tr>
<tr>
<td></td>
<td>Variability</td>
<td>BFS</td>
<td>D300(L), D1000(L)</td>
<td>1/16</td>
<td>CV</td>
</tr>
<tr>
<td><strong>Self-Test</strong></td>
<td>Time to run/part</td>
<td>--</td>
<td>Datagen</td>
<td>1—16</td>
<td>Runtime</td>
</tr>
</tbody>
</table>
Coarse-grained vs Fine-grained Evaluation (1)

Coarse-grained Method
system viewed as a black-box

Algorithms, Datasets, Resources

Graph processing system

Coarse-grained metrics
(Overall Execution Time)

Fine-grained Method
system viewed as a white-box

Algorithms, Datasets, Resources

IO operations
Processing
Overheads

Fine-grained metrics
(Stage 3 time, straggler tasks)

Fine-grained evaluation method is more comprehensive
Coarse-grained vs Fine-grained Evaluation

Abstract

Coarse-grained Method
knowledge at conceptual level

Graph Processing Systems
Distributed Infrastructure

Few, coarse-grained results

Granular

Fine-grained Method
knowledge at technical level

Fine-grained evaluation method is more comprehensive
... but more time-consuming, esp. to implement

Fine-grained Method

Few, coarse-grained results

Many, fine-grained results

GraphLab
Oracle Labs
PGX
Graphalytics = Portable Performance Analysis w Granula

Minimal code invasion + automated data collection at runtime + portable archive (+ web UI) \(\rightarrow\) portable bottleneck analysis
### Implementation status

<table>
<thead>
<tr>
<th>MR 2</th>
<th>Gigraph</th>
<th>Graph X</th>
<th>Power Graph</th>
<th>GraphLab</th>
<th>Neo4j</th>
<th>PGX D</th>
<th>Graph Mat</th>
<th>Open G</th>
<th>TOTEM</th>
<th>Map Graph</th>
<th>Medusa</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCC</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>--</td>
<td>G</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>BFS</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>V</td>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td>WCC</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>V</td>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td>CDLP</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>P'Rank</td>
<td>--</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>V</td>
<td>--</td>
<td>G</td>
<td>G</td>
<td>V</td>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td>SSSP</td>
<td>--</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>G</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

G=validated, on GitHub  
V=validation stage

https://github.com/tudelft-atlarge/graphalytics/
## Implementation status

<table>
<thead>
<tr>
<th></th>
<th>MR 2</th>
<th>Giraph</th>
<th>GraphX</th>
<th>Power Graph</th>
<th>Graph Lab</th>
<th>Neo4j</th>
<th>PGX.</th>
<th>Graph Mat</th>
<th>Open G</th>
<th>TOTEM</th>
<th>Map Graph</th>
<th>Medusa</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCC</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>--</td>
<td>G</td>
<td>G</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>BFS</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>V</td>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td>WCC</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>V</td>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td>CDLP</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>P'Rank</td>
<td>--</td>
<td>G</td>
<td>G</td>
<td>V</td>
<td>--</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>V</td>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td>SSSP</td>
<td>--</td>
<td>G</td>
<td>G</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

G=validated, on GitHub
V=validation stage

Benchmarking and tuning performed by vendors
Graphalytics Capabilities: An Example

Graphalytics enables deep comparison of many systems at once, through diverse experiments and metrics.

- Diverse datasets
- Diverse algorithms (PR, BFS)
- Diverse metrics (Edges per second, (Vertices + Edges) per second)
Which system is the best?
It depends…
Algorithm + Dataset + Metric

OK, but … why is this system better for this workload for this metric?
Granula Visualizer
Portable choke-point analysis for everyone!
Graphalytics = Modern Software Engineering Process

Graphalytics code reviews
- Internal release to LDBC partners (first, Feb 2015; last, Feb 2016)
- Public release, announced first through LDBC (Apr 2015)
- First full benchmark specification, LDBC criteria (Q1 2016)

Jenkins continuous integration server
SonarQube software quality analyzer

https://github.com/tudelft-atlarge/graphalytics/
Graph-processing is at the core of our society
The data deluge vs. Analytics
Graph processing @large
Graphalytics: Which system to select? What to tune? What to re-design?

A Performance Comparison of Graph-Processing Systems

Take-home message
System diversity (GPU-enabled)

- medusa-gpu
  Medusa: Simplified Graph Processing on GPUs

- mapgraph
  Massively Parallel Graph processing on GPUs

- Gunrock
  High-performance Graph Primitives on GPU

- TOTEM

- VertexAPI2

Dedicated Systems

Generic
Medusa

- Enables the use of GPUs for graph processing
  - Single-node, multiple GPUs
  - In-memory processing

- Simple API that hides GPU programming
  - Edge- / vertex-granularity that enables fine-grained parallelism.
  - API calls are grouped in kernels
  - Kernels are scheduled on one or multiple GPUs

- Run-time for communicating with the GPU
Totem

- Enables use of GPUs (**T-G**)
- Enables *single-node* heterogeneous (**T-H**) computing on graphs

- Programming requires expert knowledge of all types of systems
  - C+CUDA+API for specifying applications
  - Based on BSP

- Partitions the data (edge-based) between CPUs and GPUs
  - Based on processing capacity
  - Minimizing the overhead of communication
    - Buffer schemes, aggregation, smart partitioning
MapGraph

• Target at high performance graph analytics on GPUs.

• Single GPU available and Multi-GPU ready
  • Also available in a CPU-only version

• API based on the Gather-Apply-Scatter (GAS) model as used in GraphLab.
  • Productivity-oriented API
PageRank [algorithm]

- Medusa, MapGraph fail on larger datasets, with MG failing earlier
- Medusa better for small datasets
- Totem the only system able to process all tried datasets
PageRank [full]

- Medusa better?
- Totem better, when entire processing considered (execution time vs. algorithm runtime)
Lessons learned from GPU-based systems

Brave attempts to enable the use of GPUs *inside* graph processing systems

Every system has its own quirks
- Lower level programming allows more optimizations, better performance
- Higher level APIs allow more productivity

No clear winner, performance-wise

Challenge:
- Distributed accelerated graph-processing
System diversity (CPU-based)
Hadoop (Generic)

The most popular MapReduce implementation
   Generic system for large-scale computation

Pros:
   Easy to understand model
   Multitude of tools and storage systems

Cons:
   Express the graph application in MapReduce
   Costly disk and network operations
   No specific graph processing optimizations
Hadoop2 with YARN (Generic)

Next generation of Hadoop
- Supports old MapReduce jobs
- Designed to facilitate multiple programming models (frameworks, e.g., Spark)
- Separates resource management (YARN) and job management
  - MapReduce uses resources provided by YARN
Stratosphere (Generic)

Now Apache Flink (now 6M$ investment from Intel)

Nephele resource manager
  Scalable parallel engine
  Jobs are represented as DAGs
  Supports data flow in-memory, via network, or via files

PACT job model
  5 second-order functions (MapReduce has 2):
  Map, Reduce, Match, Cross, and CogGroup
  Code annotations for compile-time plans
  Compiled as DAGs for Nephele
Pregel: dedicated graph-processing + Apache Giraph (Dedicated)

Proposed a **vertex-centric** model for graph processing
- Graph-to-graph transformations

**Front-end:**
- Write the computation that runs on all vertices
- Each vertex can vote to halt
  - All vertexes halt => terminate
- Can add/remove edges and vertices

**Back-end:**
- Uses the BSP model
- Message passing between nodes
  - Combiners, aggregators
- Checkpointing for fault-tolerance
GraphLab (Dedicated)

Distributed programming model for machine learning
  Provides an API for graph processing, C++ based
  (now Python)
All in-memory
Supports asynchronous processing
GraphChi is its single-node version,
Dato as GraphLab company
Neo4J (Dedicated)

Very popular graph database
  Graphs are represented as relationships and annotated vertices
Single-node system
  Uses parallel processing
  Additional caching and query optimizations
  All in-memory
The most widely used solutions for medium-scale problems
Cluster version in development
PGX.D (Dedicated)

Designing for beefy clusters
  Fully exploits the underlying resources of modern beefy cluster machines
Low-overhead communication mechanism
  Lightweight cooperative context switching mechanism
Support for data-pulling (also data-pushing)
  Intuitive transformation of classical graph algorithms
Reducing traffic and balancing workloads
  Several advanced techniques: Selective Ghostnodes, edge based partitioning, edge chunking

Attend presentation of SC15 article!
PGX.D: Programming Model

High level programming model for Neighborhood Iteration Tasks

```cpp
foreach(n: G.nodes)
    foreach(t: n.Nbrs)
        n.foo += t.bar

class my_task_pull : public innbr_iter_task {
    void run(..) {
        read_remote(get_nbr_id(), bar);
    }
    void read_done(void* buffer,..) {
        int foo_v = get_local<int>(node_id, foo);
        int bar_v = get_data<int>(buffer);
        set_local(node_id, foo_v + bar_v, foo);
    }
}
```
GraphMat (Dedicated)

- Vertex programming as front-end and sparse matrix operations as back-end
  - “Matrix level performance with vertex program productivity”
  - Unifying vertex programming with linear algebra is new
**Example**

A Vertex Program (Single Source Shortest Path) ~ Giraph

**SEND_MESSAGE** : message := vertex_distance

**PROCESS_MESSAGE** : result := message + edge_value

**REDUCE** : result := min(result, operand)

**APPLY** : vertex_distance = min(result, vertex_distance)
BFS: results for all-2-all

No platform runs fastest for all graphs, but **Hadoop is the worst performer.** Not all platforms can process all graphs, but **Hadoop processes everything.**
Giraph: results for all algorithms, all data sets

Storing the whole graph in memory helps Giraph perform well. Giraph may crash when graph or number of messages large.
Lessons learned*

Performance of graph processing is function of (Dataset, Algorithm, Platform, Deployment)

All current platforms have important drawbacks (crashes, long execution time, tuning, etc.)
Best-performing is not only low response time
Scalability with cluster size/number of cores varies per system
Ease-of-use of a platform is very important
Graph-processing is at the core of our society
The data deluge vs. Analytics
Graph processing @large
Graphalytics: Which system to select? What to tune? What to re-design?
A performance comparison of graph-processing systems

Take-Home Message
Graphs are relevant for our society

Graph processing is challenging → many systems
• GPU/CPU,
• parallel/distributed, …

Which system to use? What to tune?
• Benchmarking with Graphalytics
• Performance = f(Data,Algo,Platform,HW)
Reading List


Graphalytics: Granula Overview

1. Modeling
   - Concepts
   - Feedback

2. Archiving
   - Information

3. Visualizing

Granular
Fine-grained Method

https://github.com/tudelft-atlarge/granula/
Granula Modeller

Job

Operation

Operation [Actor @ Mission]

Info [StartTime]

Info [EndTime]

Info [...................]

Visual

Visual

Visual

Operation

Operation

Operation

Time-consuming, expert-only, done only once
Granula Archiver

Performance Model

Performance Analyzer

Logging Patch

Graph Processing System

Granula Archiver

Job Performance Archive

Time-consuming, minimal code invasion, automated data collection at runtime, portable archive
Granula Visualizer
Portable choke-point analysis for everyone!
PGX.D: System Design Overview

Fast Network Connection

M1
- Task Manager
- Data Manager
- Communication Manager
- M2

Worker thread
Copier thread

Task
Task
Task
... Task

Edge chunking

Ghostnodes
Local Graph
Distributed Property Graph
Edge-Partitioning
Graph Loader

Task
Task
Task
... Task

Task
Task
Task
... Task

Edge chunking

Communication Manager

Copier thread
Worker thread
Worker thread
Worker thread

Fast Network Connection
Icons