Wedge
A New Frontier for Pull-based Graph Processing

Samuel Grossman and Christos Kozyrakis
Platform Lab Retreat · June 8, 2018
Graph Processing

• Problems modelled as **objects** (vertices) and **connections between them** (edges)

• Examples:
  • Internet (pages and hyperlinks)
  • Social network (people and friendships)
  • Roads and intersections
  • Products and ratings
Graph Processing
Graph Processing

Repeat until convergence
Graph Processing
Graph Processing
Graph Processing
Graph Processing
Graph Processing

**Frontier**: set of active vertices
Graph Processing

**Frontier**: set of active vertices
Graph Processing
Graph Processing: Push and Pull

Push

Group by **source vertex**

Pull

Group by **destination vertex**
Graph Processing: Push and Pull

Push
- Better at utilizing the frontier
  - Group by source vertex
    - Dominated by atomic updates

Pull
- Higher throughput
  - Group by destination vertex
    - Dominated by reads
Graph Processing: Hybrid Frameworks

- **Pull**:
  - Start
  - Frontier
  - Full Enough?
    - Yes
    - No
    - Full
    - Enough?
      - Yes
      - Finish
      - No
      - Empty?
        - Yes
        - Finish
        - No
        - Need to write the application twice

- **Push**:
  - Start
  - Frontier
  - Full Enough?
    - Yes
    - Finish
    - No
    - Empty?
      - Yes
      - Finish
      - No
      - Need to write the application twice

✓ Get the benefits of both

✓ Need to write the application twice
My Work

1. Grazelle

Pull

↑ Throughput

Start

Yes

Frontier

Full

No

Enough?

2. Wedge

Redesigned to work for pull

No

Empty?

Yes

Finish

No
Wedge

• Software implementation of the new pull-based frontier optimization, integrated into Grazelle

• Can outperform the hybrid version of Grazelle by up to 10×

• To be open-sourced
Frontier Implementation

• Bit-mask, allocated with one bit per vertex
  • ‘1’ means active, ‘0’ means inactive

• Two exist: one is being produced while the other is consumed

• An engine sets the bit to ‘1’ for any vertex when it writes an updated value to it
  • Easy to do for both push-based and pull-based engines
Frontier Consumption

Frontier
(1 bit per vertex)

Edge List
Frontier Consumption: Push

Frontier
(1 bit per vertex)

Edge List
(Source-grouped)
Frontier Consumption: Push

Frontier
(1 bit per vertex)

Edge List
(Source-grouped)
Frontier Consumption: Push

Frontier
(1 bit per vertex)

Edge List
(Source-grouped)
Frontier Consumption: Push

Frontier
(1 bit per vertex)

Edge List
(Source-grouped)
Frontier Consumption: Push

Frontier
(1 bit per vertex)

Edge List
(Source-grouped)
Frontier Consumption: Pull

Frontier
(1 bit per vertex)

Edge List
(Destination-grouped)
Frontier Consumption: Pull

Frontier
(1 bit per vertex)

Edge List
(Destination-grouped)
Frontier Consumption: Pull

Frontier
(1 bit per vertex)

Edge List
(Destination-grouped)

Pull

00000000000010000
(1 bit per vertex)
Frontier Consumption: Pull

Frontier
(1 bit per vertex)

Edge List
(Destination-grouped)
Frontier Consumption: Pull

Frontier
(1 bit per vertex)

Edge List
(Destination-grouped)
Frontier Consumption: Pull

Frontier
(1 bit per vertex)

Edge List
(Destination-grouped)

Frontier Pull

Edge List
D

(1 bit per vertex)

0000000000010000

Pull

(Destination-grouped)
Frontier Consumption: Pull

Frontier
(1 bit per vertex)

Edge List
(Destination-grouped)

Pull

(1 bit per vertex)
Frontier Consumption: Pull

Frontier
(1 bit per vertex)

Edge List
(Destination-grouped)

Pull

Done
Graph Processing: Push vs. Pull

Running Grazelle on *uk-2007* graph
Graph Processing: Push vs. Pull

Running Grazelle on *uk-2007* graph

- **PageRank**
  - Speedup: 5×
  - No use of frontier
  - Entirely frontier-driven

- **Breadth-First Search**
  - Speedup: Logarithmic
Towards a Pull-Based Frontier
Towards a Pull-Based Frontier

Vertices 2 and 4 are added to the frontier.
Towards a Pull-Based Frontier

The active edges of the graph:
- $2 \rightarrow 4$
- $2 \rightarrow 6$
- $2 \rightarrow 8$
- $4 \rightarrow 9$

Insert vertices using the classic source orientation
Towards a Pull-Based Frontier

The active edges of the graph:
- $2 \rightarrow 4$
- $2 \rightarrow 6$
- $2 \rightarrow 8$
- $4 \rightarrow 9$
- $3 \rightarrow 6$
- $8 \rightarrow 9$

Traverse using a destination orientation
Towards a Pull-Based Frontier

The active edges of the graph:

- $2 \rightarrow 4$
- $2 \rightarrow 6$
- $2 \rightarrow 8$
- $4 \rightarrow 9$
- $3 \rightarrow 6$
- $8 \rightarrow 9$
- $+1$ million extra edges

Traverse using a destination orientation
Towards a Pull-Based Frontier

Filter out inactive edges for each vertex

The active edges of the graph:

• 2 → 4
• 2 → 6
• 2 → 8
• 4 → 9
Pull-Based Frontier Requirements

• **Insert** vertices using the classic source orientation

• **Traverse** using a destination orientation

• **Filter out** inactive edges for each vertex
Wedge Frontier

- An edge-oriented frontier bit-mask

- Each bit represents a small number of edges in the destination-grouped edge list
Wedge Frontier

• An **edge-oriented** frontier bit-mask
  ✓ **Filter out** inactive edges for each vertex

• Each bit represents a small number of edges in the **destination-grouped** edge list
  ✓ **Traverse** using a destination orientation
Hybrid Functional Overview

Start

Pull

Yes

Full Enough?

No

Push

1010001001010110

Full

Enough? Empty?

Yes

No

Finish

Yes

Empty?
Wedge Functional Overview

✓ **Insert** vertices using the classic source orientation

Pull

Start

Yes

Full Enough?

Yes

Finish

No

Empty?

No

Wedge

1010001001010110

Full

Enough? Empty?

Yes

No

Insert vertices using the classic source orientation
Wedge Operation

```csharp
foreach vertex v in source_oriented_frontier {
    // activate the vertex
    activate_vertex(v, wedge_frontier);
}
```

Sets the correct bits in the Wedge frontier
Hybrid Data Structures

**Destination-Grouped**
- Edge List
  - In-edges grouped by destination vertex

**Source-Grouped**
- Edge List
  - Out-edges grouped by source vertex
Wedge Data Structures

- **Pull**
  - **Destination-Grouped**
    - Edge List
  - • In-edges grouped by destination vertex

- **Push**
  - **Source-Grouped**
    - Edge List
  - • Out-edges grouped by source vertex
Wedge Data Structures

**Pull**

**Destination-Grouped**

Edge List

- In-edges grouped by destination vertex

**Wedge**

**Source-Grouped**

Edge Index

- Bit positions in the Wedge frontier, grouped by source
Evaluation Scope

• Grazelle + Wedge is compared with the hybrid version of Grazelle

• Three applications: Single-Source Shortest Path, Breadth-First Search, and Connected Components

• Running on a single Intel Xeon E5-2658 v3 processor
  • 12 physical cores / 24 logical cores
Single-Source Shortest Path

Grazelle + Wedge does better than just closing the performance gap!
Single-Source Shortest Path

Zoomed In: Grazelle (Hybrid) vs Grazelle + Wedge

- Grazelle (Hybrid)
- Grazelle + Wedge (Pull)
- Grazelle + Wedge (Wedge)

Relative Execution Time

dimacs-usa  livejournal  twitter-2010

10x
Breadth-First Search

Pull Engine Improvement

- Grazelle (Pull)
- Grazelle (Hybrid)
- Grazelle + Wedge

Relative Execution Time

- dimacs-usa
- livejournal
- twitter-2010
Breadth-First Search

Zoomed In: Grazelle (Hybrid) vs Grazelle + Wedge

- Grazelle (Hybrid)
- Grazelle + Wedge (Pull)
- Grazelle + Wedge (Wedge)

Relative Execution Time

- dimacs-usa: 2.1×
- livejournal: 1.2×
- twitter-2010: -7%
Connected Components

Pull Engine Improvement

- Grazelle (Pull)
- Grazelle (Hybrid)
- Grazelle + Wedge

Relative Execution Time

- dimacs-usa
- livejournal
- twitter-2010
Connected Components

Zoomed In: Grazelle (Hybrid) vs Grazelle + Wedge

- Grazelle (Hybrid)
- Grazelle + Wedge (Pull)
- Grazelle + Wedge (Wedge)

Relative Execution Time

- dimacs-usa
- livejournal
- twitter-2010
Conclusion

• **Eliminated the only benefit** that a push engine had over a pull engine; the push engine is now obsolete!

• Implemented in software and integrated into Grazelle

• Can outperform the hybrid version of Grazelle by up to 10×

• To be open-sourced
Thank You

Questions?