FlashGraph: Processing Billion-Node Graphs on an Array of Commodity SSDs

Da Zheng, Disa Mhembere, Randal Burns, Joshua Vogelstein, Carey E. Priebe, Alexander S. Szalay
Johns Hopkins University
Overview

- Large-scale graph analysis using a commodity SSD array.
- We built a *semi-external memory* [1] graph processing framework called FlashGraph.
  - Achieve performance comparable to in-memory graph engines.
  - Scale to graphs with billions of vertices and hundreds of billions of edges.

Motivation

- Graph analysis is ubiquitous:
  - Many real-world problems are modeled with graphs.
Challenges in Graph analysis

- Many real-world graphs are massive.
  - Facebook’s social graph has billions of vertices.
  - Web page graph has tens of billions of vertices.
- Seemingly random vertex connection.
  - Small random memory access.
  - Hard to explore data locality.
  - Efficient graph analysis requires RAM.
- Power-law distribution in vertex degree.
  - Load balance.
Alternatives for scaling

• **Option 1:** Buy a machine with large RAM.
  - Examples: SNAP, Galois, Ligra
  - Problems: Price & Scalability.

• **Option 2:** Scale out in a cluster.
  - Examples: Pregel, GraphLab/PowerGraph, GraphX
  - Problem: Network is the bottleneck.

• **Option 3:** Scale with hard drives.
  - Examples: GraphChi, X-Streams
  - Problem: Slow.
Why Scale with SSDs?

- **SSD properties:**
  - Throughput: over 1M random IOPS.
  - Latency: 20μs.
  - Much cheaper and much larger than RAM.

- **Questions:**
  - How much can we approach to the performance of in-memory graph analysis?
  - How do we maximize the benefits SSDs bring for graph analysis?
Challenges in using SSDs

- High OS overhead in fast I/O.
  - Heavy locking overhead in the OS block subsystem.
  - We tackle it with SAFS, a user-space filesystem optimized for a large SSD array [1].

- High latency and low throughput compared with RAM.
  - Latency: three orders of magnitude higher.
  - Throughput: almost an order of magnitude lower.

Design principles for I/O optimization:
- Reduce I/O.
- Overlap I/O and computation.
- Perform sequential I/O.

Architecture

- Three layers for graph analysis using a large SSD array.
  - SAFS: a user-space file system for delivering maximal I/O performance.
  - FlashGraph: schedules vertex programs to optimize I/O.
  - Graph algorithms: part of vertex computation in the page cache.
Semi-external memory

- Algorithmic vertex state in memory.
- Edge lists of vertices on SSDs.
- Scalability:
  - In proportion to the ratio of edges to vertices in a graph.
- Advantage:
  - vs. distributed memory:
    - Enable in-memory vertex communication.
  - vs. external memory:
    - Reduce reads and avoid writes to SSDs.
I/O access in FlashGraph

- Access edge lists only required by an application.
  - Reduce I/O.

- Conservatively merge I/O requests.
  - Increase sequential I/O.
  - Merge criteria:
    - Two edge lists are on the same SSD page.
    - Two edge lists are on adjacent SSD pages.
  - Never increase the amount of data read from SSD.
I/O merging in FlashGraph

- The most common case:
  - A vertex only accesses its own edge list.
Applications

- **Class 1**: subset of vertices access their own edge lists.
  - Breadth-first search (BFS).
  - Betweenness centrality (BC).

- **Class 2**: all vertices access their own edge lists.
  - PageRank (PR).
  - Weakly connected components (WCC).

- **Class 3**: a vertex accesses multiple edges lists.
  - Triangle counting (TC).
  - Scan statistics (SS).
FlashGraph Performance

- FlashGraph has performance comparable to Galois.
- Ext-mem FlashGraph has performance comparable to in-mem FlashGraph.
- Ext-mem FlashGraph significantly outperforms PowerGraph.
FlashGraph Scalability

- Largest graph publicly available: 3.5B vertices and 129B edges.

Previous Results:
- Google Pregel used **300 multi-core machines** for SSSP in +10 min on **1B vertex** and **127B edge** graph.
- Microsoft Trinity used **14 12-core machines** for BFS in +10 min on **1B vertex** and **13B edge** graph.

FlashGraph uses one machine with 32 CPU cores.

We can scale to a much larger graph!
Conclusion

- SSD-based graph analysis has performance comparable to in-memory counterparts.
- FlashGraph offers unprecedented opportunities to perform massive graph analysis efficiently with commodity hardware.
Thank you!

• Da Zheng: dzheng5@jhu.edu
• FlashGraph: https://github.com/icoming/FlashGraph
Scale out

- Graph analysis in distributed memory.
- PageRank on EC2 m2.4xlarge.
- Problems:
  - Network is the bottleneck.
Execution model

- A graph is partitioned for parallel processing.
- Each thread schedules and executes vertex programs independently.
- Vertex status:
  - Running.
  - Active.
  - Inactive.
FlashGraph programming interface

- **Vertex centric**
  - Users write programs from the perspective of vertices

```cpp
class vertex {
    void run(graph_engine &g);
    void run_on_vertex(graph_engine &g, page_vertex &v);
    void run_on_message(graph_engine &g, vertex_message &msg);
    void run_on_iteration_end(graph_engine &g);
};
```

Asynchronous

Run in the page cache asynchronously

Asynchronous