MSSG: A Framework for Massive-Scale Semantic Graphs

Timothy D. R. Hartley[1], Umit Catalyurek[1,2],
Füsun Özgüner[1]
[1]Dept. of Electrical & Computer Engineering
[2]Dept. of Biomedical Informatics
The Ohio State University

Andy Yoo, Scott Kohn, Keith Henderson
Lawrence Livermore National Laboratory

Motivation

- Graph data is growing in size
  - Kolda et al. (2004) estimate emerging graphs have 10^{15} entities!
  - Data will be dynamic
- Large-scale data
  - Out-of-core data structures
  - Parallel computer (shared memory / cluster)
- Cluster architecture
  - Commodity hardware is still cheap
  - High-speed interconnection networks are becoming commonplace

Related work

- External Memory Data structures
  - Good online performance
    - B tree
  - Good I/O performance
    - Buffer tree (Arge 1996)
- Parallel Graph
  - Efficient memory usage
    - Frontier BFS (Korf et al. 2005)
  - Efficient scale-free search
    - Prioritize hub vertices (Adamic et al. 2001)
- Middleware
  - TPIE, River

Objectives

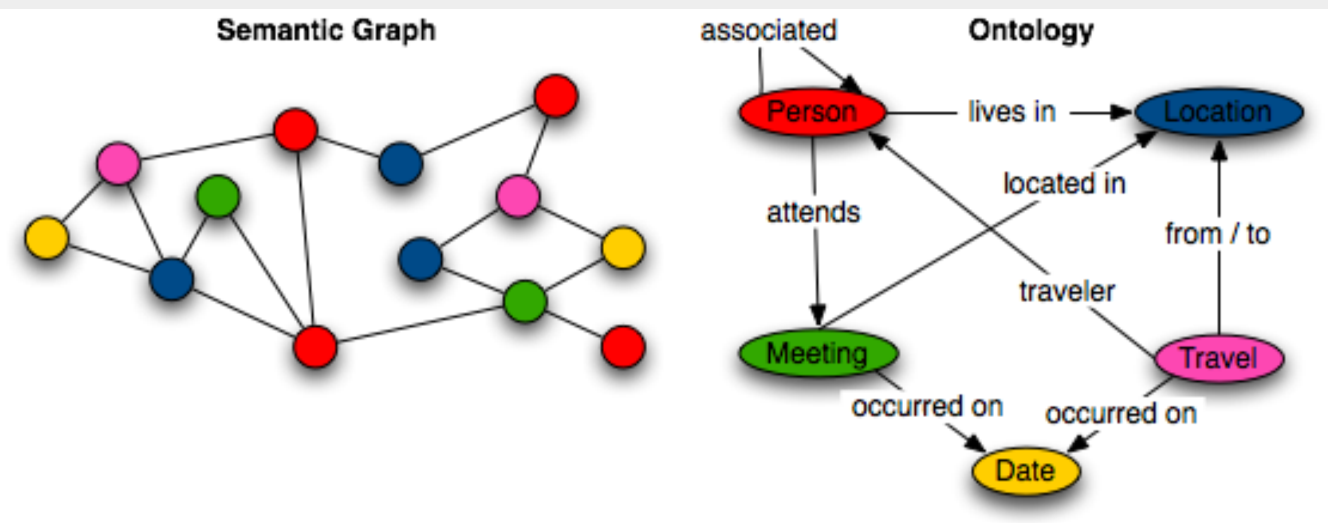
- Design and implement a flexible, easy-to-use API and associated middleware platform for analyzing massive-scale semantic graphs

Outline

- Scale-free semantic graphs
- Massive data
- Design: MSSG architecture and services
- Implementation: MSSG prototype
- Experimental setup and results
- Conclusion
- Future Work

Semantic graphs

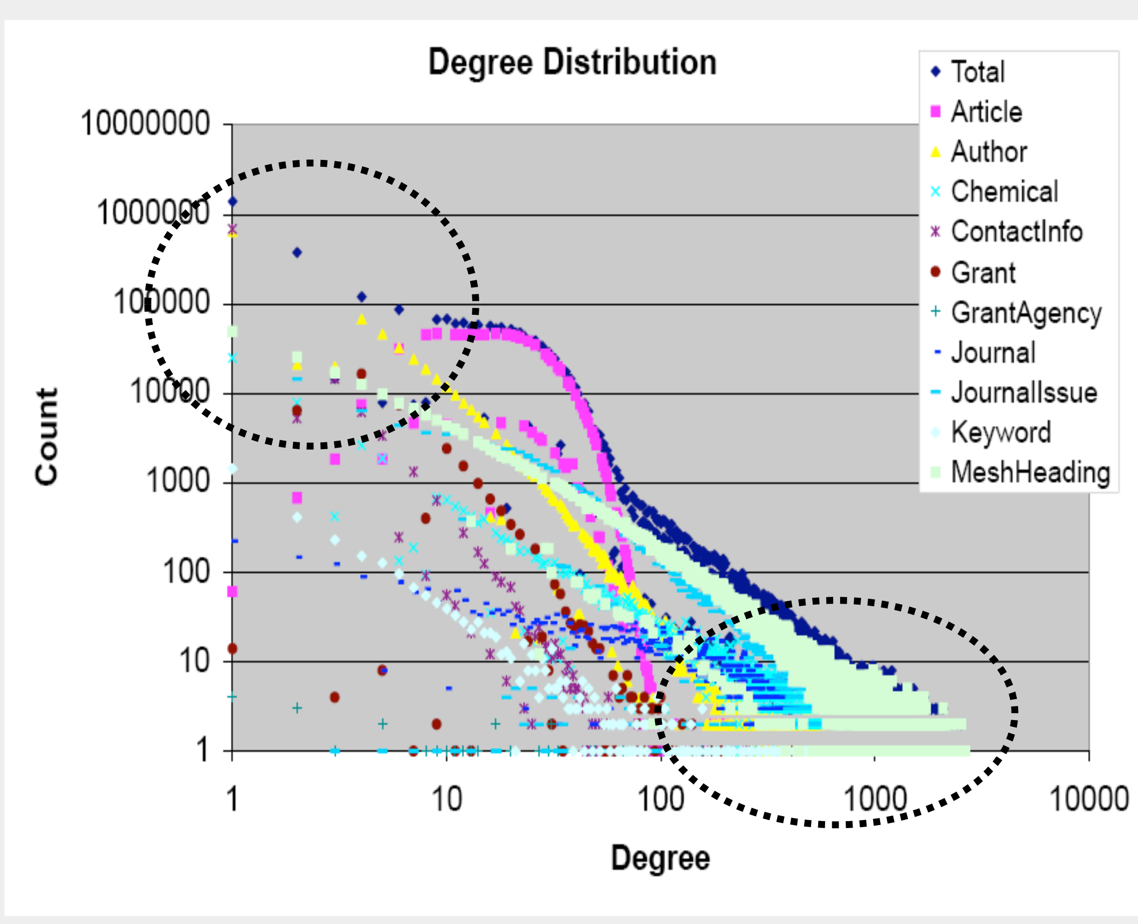
- Vertices/Edges have type information
- Topology restricted by ontological information
- Useful to model real interaction networks
  - Social networks



multiscale computing laboratory
osu department of biomedical informatics

Scale-free graphs

- Roughly follow power-law
- Small-world phenomenon
- Many vertices have low degree
- A few 'hub' vertices have large degree
- Pubmed Extraction

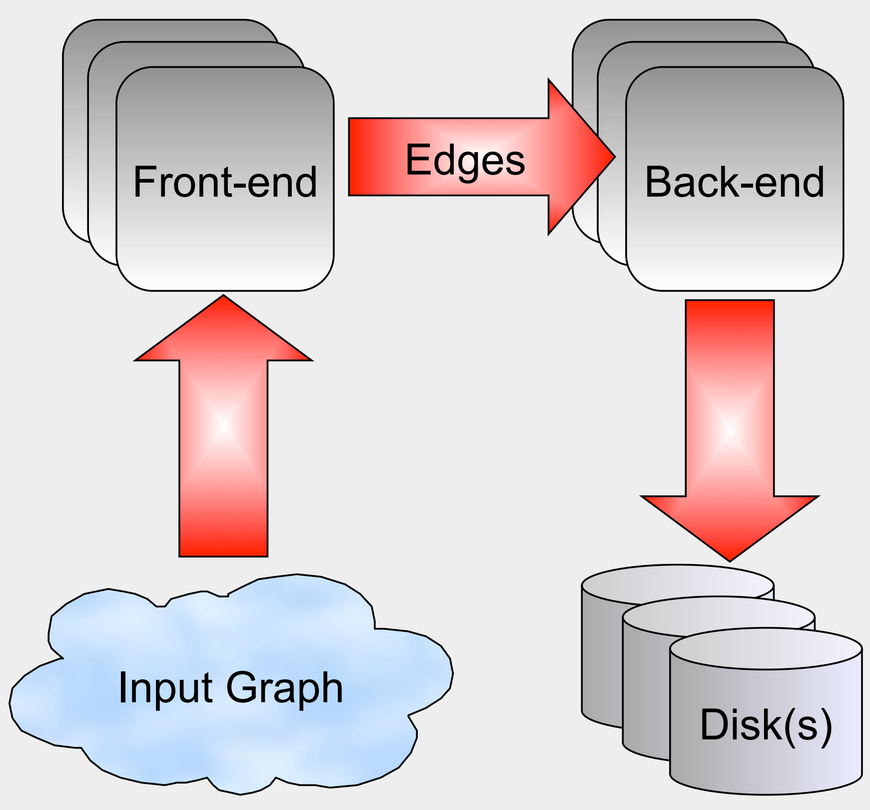


Massive Data?

- Massively multithreaded SMP
  - Cray MTA-2
- Massively parallel cluster
  - IBM Bluegene/L
- Advantages
  - High performance
- Disadvantages
  - Expensive!
  - Algorithm tightly coupled with data distribution

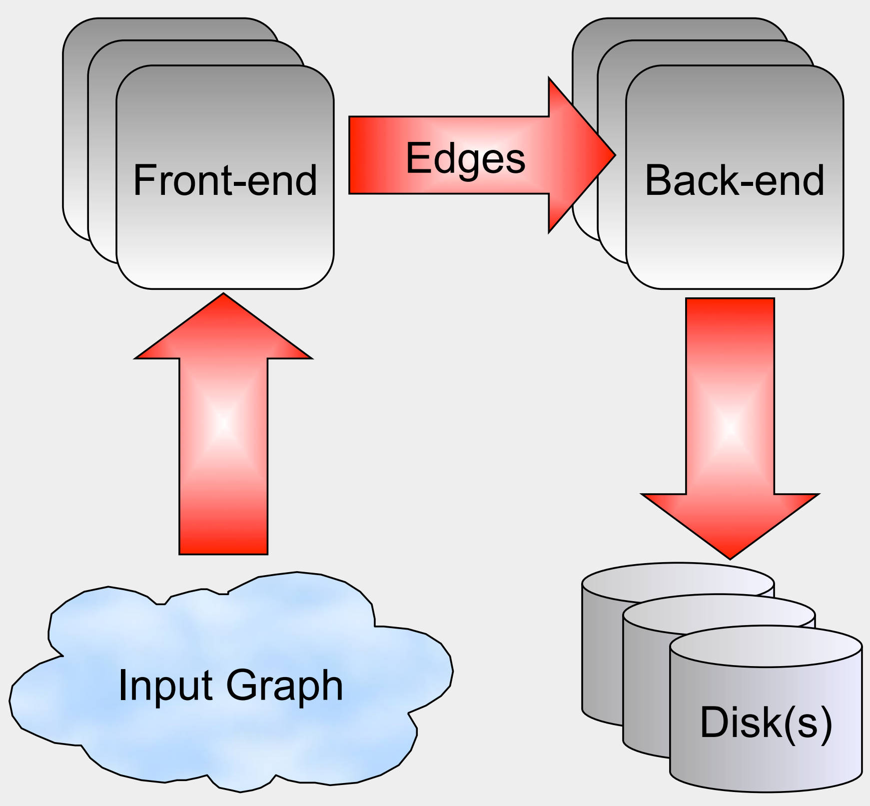
MSSG architecture

- Scalable
  - Parallel layout
    - Multiple front-end nodes
    - Multiple back-end nodes
  - External memory
    - Back-end nodes
- Practical
  - Target graphs will be dynamic
    - Streaming updates



MSSG architecture (continued)

- Services
  - Analysis
    - Graph Query Service
  - Storage
    - Ingestion Service
    - Graph Database Service

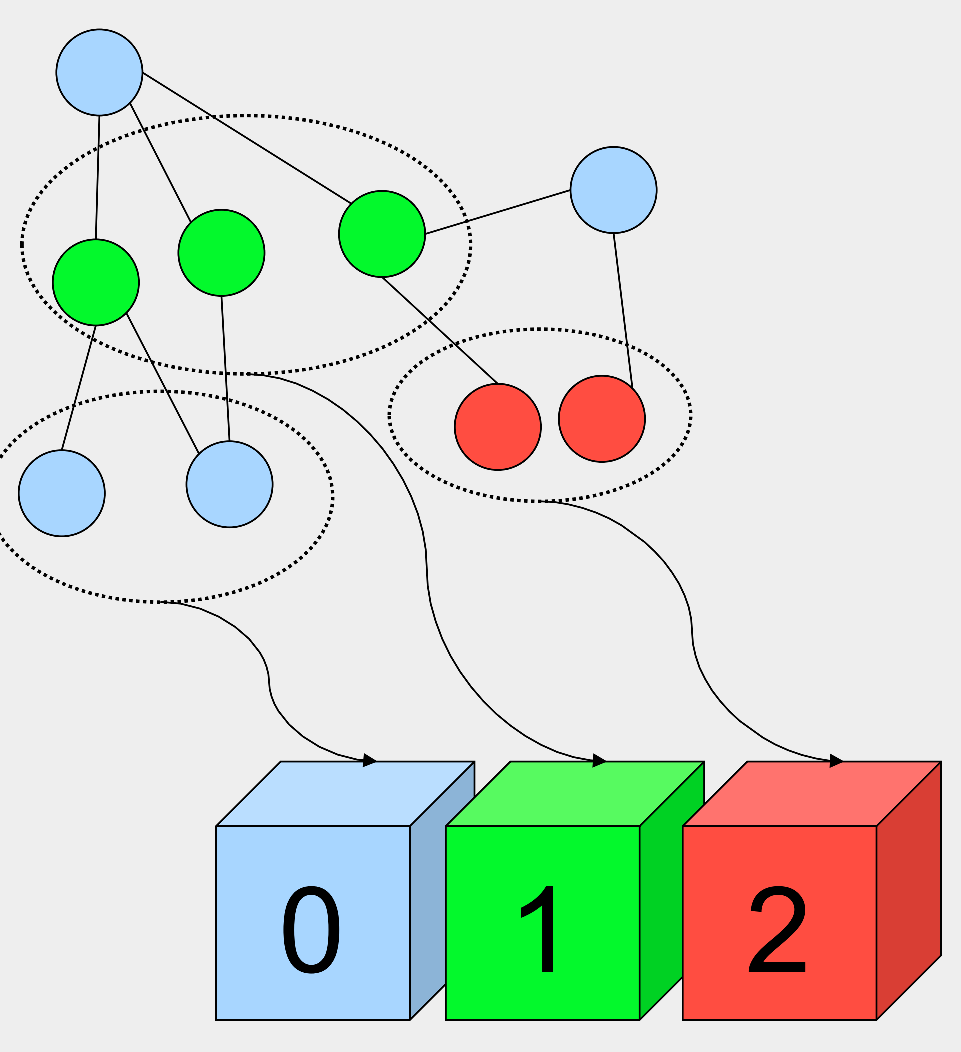


Graph Query service

- Queries come in via user-interface
- Posted to database back-end nodes
- Orchestrated by the query service
- Implementation possibilities
  - BFS
  - Best-first search
  - Pattern search
  - Neighborhood quality quantification

Ingestion service

- Edges streamed from ingestion front-end node(s) to database back-end node (s)
  - Window size important
    - Amortize disk / communication latency
- Ingestion node(s) must partition the graph
  - Plug-in architecture



Graph Database service

- Exposes simple interface
  - Get adjacency list for vertex
  - Store vertex metadata (e.g. visited at level *x*)
- Plug-in architecture to allow various database types to be used
  - In memory
    - Array
    - HashMap
  - Out-of-core
    - BerkeleyDB
    - Commodity database installation (MySQL)
    - Streaming Graph
    - GrDB

Streaming Graph details

- Active Disk research
  - Netezza streaming database
- Finding adjacency list of a vertex requires full scan
  - Read a chunk of the graph from disk
  - Pick which edges match vertex
  - Return full list of adjacent vertices
- Slow for single adjacency list lookup
- Fast when fringe expansion touches large portion of graph
  - Lower seek overhead
- Good as worst-case bound

GrDB: Scale-free graph storage

- Wide variability in vertex degree
- Design decisions
  - Fixed record size
    - Wasted space
    - MSSG targets streaming graphs
  - Variable record size
    - Efficient space usage
    - Complex
  - Multiple fixed record files
    - Efficient space usage
    - Simple

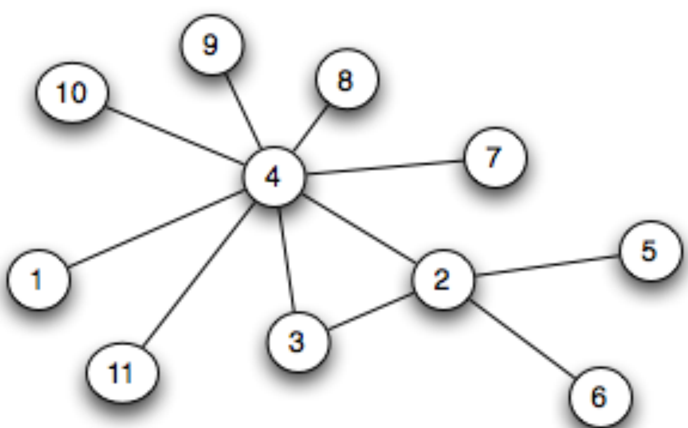
GrDB (continued)

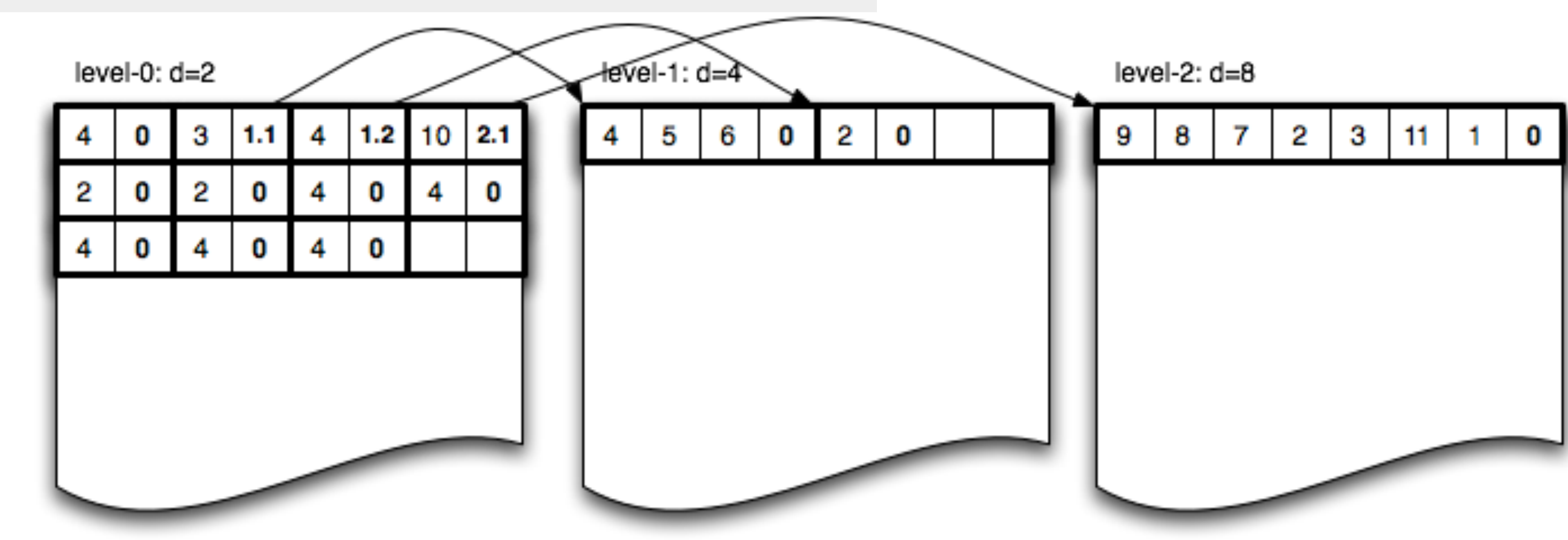
- Targeted to scale-free graphs
- *File-levels*
  - Record sizes chosen to match scale-free graph vertex degree distribution

    $$d_l = d^d$$

  - File level 0
    - 2 records
  - File level 1
    - 4 records
- Records grouped together into sub-blocks
- Sub-blocks grouped into Disk-blocks
  - Disk-block = unit of I/O

GrDB (continued)





MSSG Prototype

Java

DataCutter

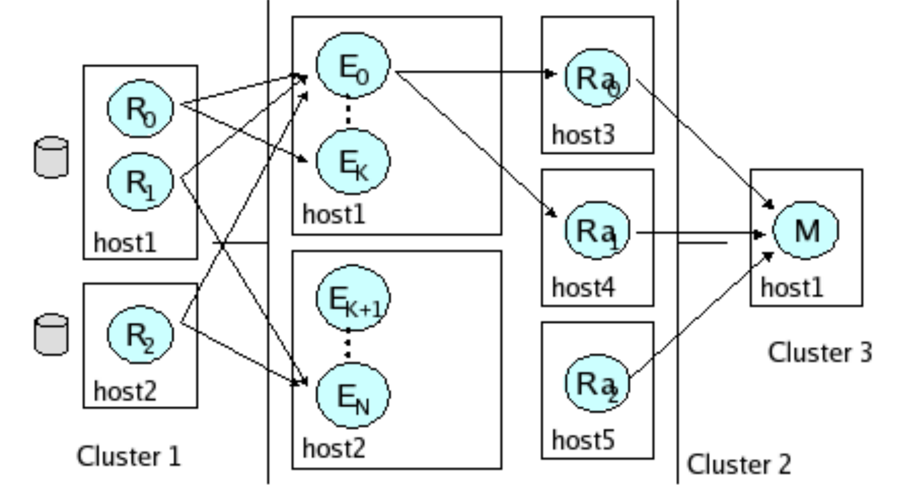
MPI

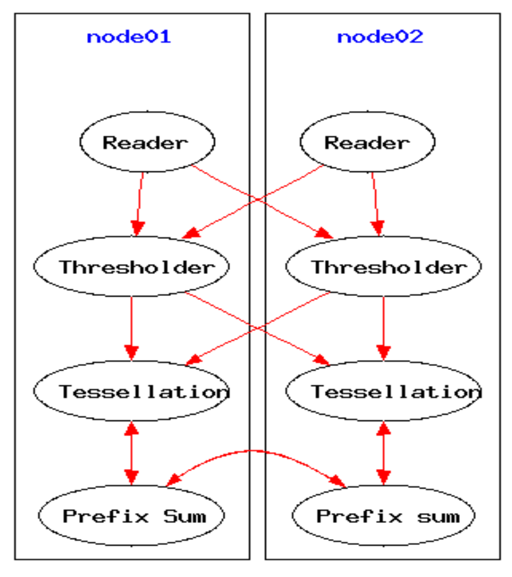
MSSG Prototype

- MPI
  - Fast, scalable parallel communication
  - High-speed interconnect support
- DataCutter
  - Easy-to-use filter-based API
  - Rapid development
  - Robust processing model
- Java
  - Rapid development
  - Fast execution time

DataCutter

- Component-Framework for task- and data-parallel manipulation of large scientific data
  - Transparent copies of filters
  - C++/Java/Python filters
  - Each filter runs as a thread
- Filter-stream metaphor of data processing
  - Data is streamed from producer to consumer filters
- Provide grid-based distributed computation and application-specific storage access
- Filters form a parallel workflow across any number of heterogeneous nodes





Experimental setup

- 24 nodes - dual 2.4GHz AMD Opteron 250
  - 8 GB RAM per node
  - 500 GB local disks in RAID 0 per node
  - Infiniband
- Graphs
  - Pubmed-S: 3,751,921 vertices and 27,841,781 edges
  - Pubmed-L: 26,676,177 vertices and 519,630,678 edges
  - Syn-2B: 100 Million vertices and 2 Billion edges
- Metrics
  - Search time (s)
  - Aggregate Edges/s processed

Experimental Results: Pubmed-S

Ingestion Performance

Time (s)

275
250
225
200
175
150
125
100
75
50
25
0

1 frontend node
4 frontend nodes

ARRAY
HASH
GRDB
BDB
MYSQL

GraphDB Service Implementations

Experimental Results: Pubmed-S

Search Performance

Time (s): 0, 5, 10, 15, 20, 25, 30, 35, 40

Path length: 1, 2, 3, 4, 5, 6

ARRAY
HASH
GRDB
BDB
MYSQL

Experimental Results: Pubmed-L

Ingestion Performance

Time (s)

1000
900
800
700
600
500
400
300
200
100
0

4
8
16

Number of Processors

ARRAY
HASH
GRDB
STREAM
BDB

Experimental Results: Pubmed-L

Weighted Average Search Performance

Time (s)

240
220
200
180
160
140
120
100
80
60
40
20
0

4
8
16

Number of Processors

ARRAY
HASH
GRDB
STREAM
BDB

Experimental Results: Pubmed-L

Weighted Average Seach Performance

Edges/s processed (in Millions)

30
25
20
15
10
5
0

4
8
16

Number of processors

ARRAY
HASH
GRDB
STREAM
BDB

multiscale computing laboratory
osu department of biomedical informatics

Experimental Results: Syn-2B

Search Performance

Ttime (s)

140
130
120
110
100
90
80
70
60
50
40
30
20
10
0

4
5
6
7

Path length

ARRAY
HASH
GRDB
GRDB - OOC

Experimental Results: Syn-2B

Search Performance

Edges/s processed (in Millions)

32.5
30.0
27.5
25.0
22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0

4
5
6
7

Path length

ARRAY
HASH
GRDB
GRDB - OOC

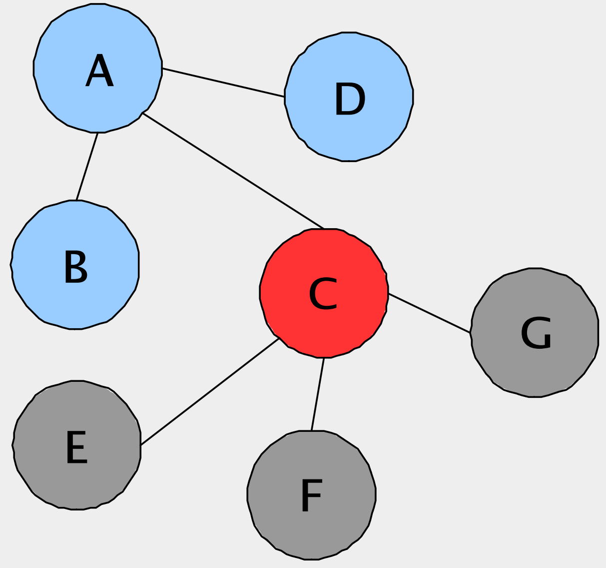
Conclusions and Future Work

- One of the first parallel, out-of-core BFS algorithms
- Good first step
- One trillion edge graph
  - Expected ingestion with GrDB in roughly 77 hours
  - Expected average search in 10s of minutes
- Future work
  - I/O-efficient hash / index structure needed
  - More performance testing
  - Larger graphs

Thank you!

Breadth-first search

- Serialized version
  - Use queue for frontier vertices
- Parallel version
  - Use global queue
    - High synchronization overhead
  - Use local queue
    - Must decide vertex partitioning



Breadth-first search (continued)

```
while (goal not found)
    while (fringe empty)
        fringe <- chunk from other node
        if (goal found by other node)
            quit search
        expand (fringe)
        if (goal found by this node)
            quit search
        send fringe to other nodes
    level = level + 1
```