Distributed Multithreaded Breadth-First Search on Large Graphs using DXGraph

Stefan Nothaas
Department of Computer Science, Heinrich-Heine-University Düsseldorf, Germany

HPGDMP 2016, November 12th 2016, Salt Lake City
Application Domains

• Large-scale interactive applications and online graph computations
• Example: Facebook
  • Over one billion users
  • Over 150 TB of data (2011)
  • 70% of all objects are smaller than 64 byte (2011)
Challenges & Objectives

• Challenges
  • “Data explosion”: Billions of small data objects
  • Data evolving and expanding
  • Interactive applications
    • Fast object lookup and retrieval
    • Irregular access patterns
• Objectives
  • Low latency
  • Scalability
  • Efficient handling of small objects
DXRAM - In-Memory Key-Value Storage

- Distributed system for clusters in data centers
  - Data center with 1000 machines, 64 GB RAM each
  - Total of 62.5 TB fast memory
- All data always in RAM
- Key-value data tuples: "chunks"
- Optimized for handling billions of small chunks
- Persistency through logging to raw device (SSD aware)
- Parallel distributed fast recovery
DXRAM - Architecture

- DXRAM Core
  - Engine
  - Components
  - Services (API)
- Custom applications

![DXRAM Architecture Diagram]
DXRAM - Node Types

- **Superpeer** Overlay
  - Fast node lookup with custom Chord-like overlay
  - Superpeers do not store chunks but all global meta-data (modified B-Tree)
  - Meta-data replicated on successors
  - 5 to 10% of all nodes are superpeers
  - Every superpeer knows every other superpeer
DXRAM - Node Types

- **Peers** store chunks
  - Every peer is assigned to one superpeer
  - Key: 64 bit globally unique sequential chunk ID (CID)
  - Value: Byte buffer
DXRAM - Memory Management

- Custom allocator designed for many small chunks
- Paging like address translation
  - Chunk location lookup in $O(1)$
  - Tables created on demand

- Average metadata overhead $\sim 5\%$ (avg. payload size: 64 bytes)
- Example: 64 GB for key-value store $\Rightarrow \sim 1$ billion chunks per node
DXRAM - Foundation for Graph Computation

- DXRAM provides
  - Low latency
  - Scalability
  - Efficient handling of small objects

⇒ Foundation for graph processing

- What else do we need for graph processing?
  - Utilize CPU resources on storage nodes
  - Move computations to data ⇒ locality
DXCompute

- Extends DXRAM Core
- Services to run computations on peers
- Benefit from locally stored chunks

**JobService**
- Deploy light weight jobs to single nodes
- Scheduling by work stealing

**MasterSlaveService**
- Aggregate nodes to compute groups
- Deploy compute tasks to group
DXCompute - MasterSlaveService

- Peers form a compute group
- Master: one peer as coordinator
- Slaves: further peers as distributed workers
- Tasks are submitted to compute groups
- Groups can grow
- Access to other nodes outside group (storage)
- Task context on execution
  - Compute group ID
  - Own slave ID
  - List of node IDs of every other slave
  - Total number of slaves
DXGraph

• DXGraph extends DXCompute
• Uses JobService or MasterSlaveService
• Algorithms for graph processing
• Graph data loading
• Natural representation of graph data as objects: Vertex, Edge, Attribute
DXGraph - Breadth-First-Search

- Implementation as specified by the Graph500 benchmark
- Stress test for system: Highly random access
- Standard top-down combined with bottom-up approach (reducing number of visited vertices)
- Compute task: Implements BFS
  - Distributed and multithreaded implementation
  - Delegates processing of non local vertices to owner node
  - Lock-free bitmap based frontier data structure
  - Low overhead synchronization between BFS levels
BFS: DXGraph, Grappa and GraphLab

- Scale 24 RMAT graph (Graph500 generator)
- Private cluster, 4 nodes connected by Gigabit Ethernet

Total memory consumption

Average BFS execution times
DXGraph's BFS on Hilbert

- HPC system of our university:
  - BULL: Cluster architecture, 112 nodes with 24 cores and 128 GB RAM each
  - Running DXGraph's BFS implementation on BULL cluster
  - Goals: Scalability, Low memory overhead ⇒ storing many small objects

- Graph sizes tested: Scale 28 (64 GB) to 32 (1 TB)
- Random but equally distributed to 8 to 104 compute nodes
DXGraph's BFS on Hilbert - Results

BFS Average Execution Times

Execution Times in sec

Number of BFS Nodes
DXGraph's BFS on Hilbert - Results
Conclusions & Outlook

• Conclusions
  • DXRAM: Distributed in-memory key-value store for many small objects
    • ~ 5% metadata overhead
  • DXGraph: Fast and scalable BFS implementation on 104 nodes
    • Graph: 1 TB, ~4.3 billion vertices, ~137 billion edges
    • Double the nodes $\Rightarrow$ Half the execution time
    • Up to 1 billion traversed edges per second

• Outlook
  • Extending system evaluation
  • Graph framework
  • Infiniband