GLADE: A Scalable Framework for Efficient Analytics

Florin Rusu (University of California, Merced)
Alin Dobra (University of Florida)

Big Data Analytics

- Big Data
  - Storage is cheap ($100 for 1TB disk)
  - Everything is stored
    - Actions on web
    - Bank and credit card transactions
    - Network logs
  - Diverse formats
  - Not clean
- Analytics
  - Extract useful information from data for decision making
  - More than SQL queries and data cubes
  - Statistics
  - Linear algebra
  - Machine learning and data mining

Big Data Analytics Solutions

- Parallel databases + extended SQL
  - User-Defined Functions (UDF)
  - User-Defined Aggregates (UDA)
- Map-Reduce
  - Distributed file system
  - Simplified execution model (Map + Reduce operator)
  - User code executed in each phase
  - SQL-like languages built on top of Map-Reduce
    - Pig Latin, Hive, Sawzall
- Dryad
  - Execution model is arbitrary DAG
  - DryadLINQ – SQL operators embedded into C#
  - SCOPE – SQL-like scripting language
- Dremel
  - Aggregate queries over nested columnar data
  - Execution model is multi-level tree with partial aggregation

GLADE

- Associative-decomposable analytical tasks
- Execute only user code
- UDA iterator-based interface
- Input
  - tuples from a column-oriented relational store
- Output
  - state of UDA computation
- Execution model
  - multi-level tree with partial aggregation at each level
- **Remarkable execution time**

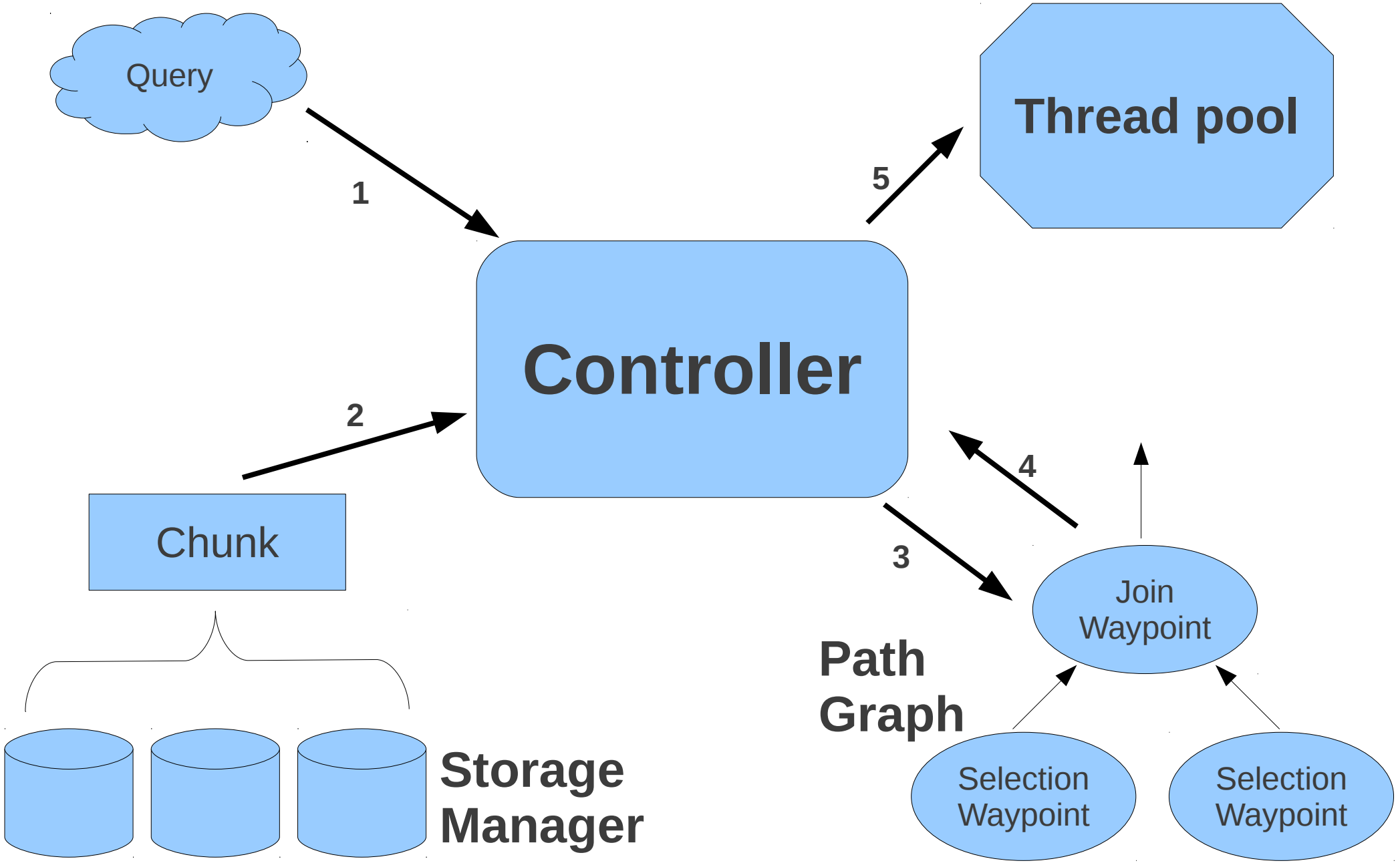
Outline

- DataPath Query Execution Engine
  - Architecture
  - Waypoints & work-units
  - Query execution
- Generalized Linear Aggregates (GLA)
  - User-defined aggregates (UDA)
- GLADE
  - Architecture
  - Task execution
  - Experimental results

DataPath

- Multi-query analytical execution engine
  - Shared scan & processing
- Push-driven execution
  - Data pushed through operators at speed read from disk
- Chunk-at-a-time processing
  - Sequential scan (large pages)
  - Vectorized execution (loop unrolling)
- Runtime code generation, compilation, loading, and execution
  - M4 templates instantiated into C code with query attribute types and arithmetical & logical expressions

Architecture



Waypoints & Work-units

- Waypoint
  - Controller for specific operation
    - Selection, Join, etc.
  - State
  - Tasks = ***work-units***
- Work-unit
  - Code generated & loaded at runtime
- Selection waypoint
  - No state
  - Work-units: Filter
- Join waypoint
  - State: hash table
  - Work-units: BuildHash, MergeHash, Probe

Query Execution

- When a new query arrives in the system, the code to be executed is first generated, then compiled and loaded into the execution engine. Essentially, the waypoints are configured with the code (work-units) to execute for the new query.
- Once the storage manager starts to produce chunks for the new query, they are routed to waypoints based on the query execution plan (path graph).
- If there are available threads in the system, a chunk and the work-unit selected by its waypoint are sent to a worker thread for execution.
- When the worker thread finishes a work-unit, it is returned to the pool and the chunk is routed to the next waypoint.

Outline

- DataPath Query Execution Engine
  - Architecture
  - Waypoints & work-units
  - Query execution
- Generalized Linear Aggregates (GLA)
  - User-defined aggregates (UDA)
- GLADE
  - Architecture
  - Task execution
  - Experimental results

User-Defined Aggregates (UDA)

```
AGGREGATE Avg(Next Int) : Real {
    TABLE state(tsum Int, cnt Int);
    INITIALIZE : {
        INSERT INTO state VALUES (Next, 1);
    }
    ITERATE : {
        UPDATE state
        SET tsum = tsum + Next, cnt = cnt + 1;
    }
    TERMINATE : {
        INSERT INTO RETURN
        SELECT tsum / cnt FROM state;
    }
}
```

```
public class Avg {
    int sum;
    int count;
    public void Init() {
        sum = 0;
        count = 0;
    }
    public void Accumulate(int newVal) {
        sum = sum + newVal;
        count = count + 1;
    }
    public void Merge(Avg other) {
        sum = sum + other.sum;
        count = count + other.count;
    }
    public double Terminate() {
        return (double)sum / count;
    }
}
```

Generalized Linear Aggregates (GLA)

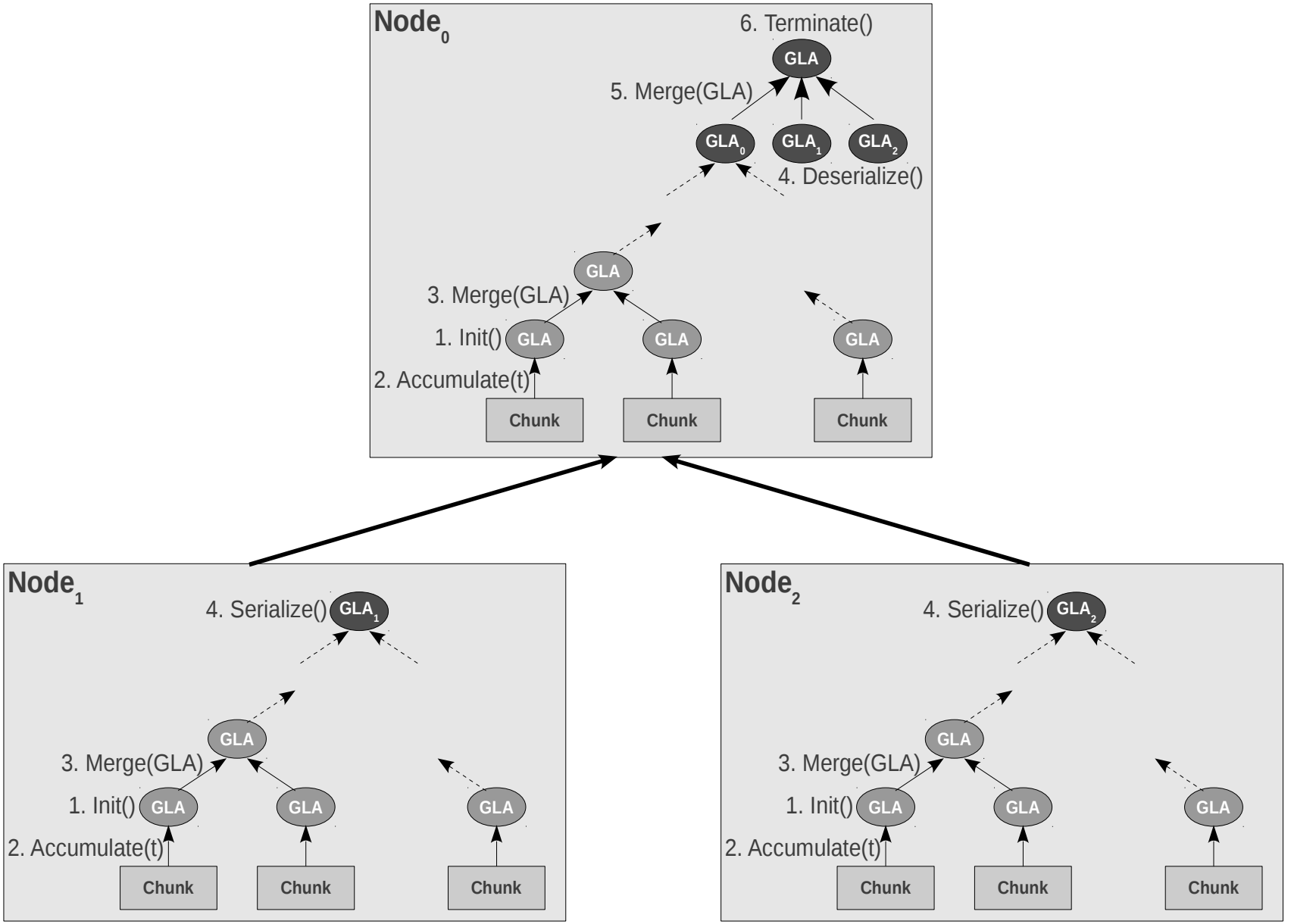
- Aggregates computed through sequential passes over data (multiple)
- State
- UDA interface
  - Init
  - Accumulate
  - Merge
  - Terminate
- **Serialize/Deserialize**
- **Define and invoke in C++**
  - **Direct access to state**

```
public class Avg {
    int sum;
    int count;
    public void Init() {
        sum = 0;
        count = 0;
    }
    public void Accumulate(int newVal) {
        sum = sum + newVal;
        count = count + 1;
    }
    public void Merge(Avg other) {
        sum = sum + other.sum;
        count = count + other.count;
    }
    public double Terminate() {
        return (double)sum / count;
    }
    ARCHIVER_SIMPLE_DEFINITION()
}
```

Outline

- DataPath Query Execution Engine
  - Architecture
  - Waypoints & work-units
  - Query execution
- Generalized Linear Aggregates (GLA)
  - User-defined aggregates (UDA)
- GLADE
  - Architecture
  - Task execution
  - Experimental results

GLADE Architecture



GLA Distributed Engine (GLADE)

- User defines GLA
- User invokes GLA with data on DataPath
  - Execute right near data
  - Take full advantage of parallelism
    - Single machine & across multiple machines
- GLA Waypoint
  - State: List of GLAs
  - Work-units
    - Accumulate
    - Merge
- User gets back computed GLA

Query Execution

- The coordinator generates the code to be executed at each waypoint in the DataPath execution plan. A single execution plan is used for all the workers.
- The coordinator creates an aggregation tree connecting all the workers. The tree is used for in-network aggregation of the GLAs.
- The execution plan, the code, and the aggregation tree information are broadcasted to all the workers.
- Once the worker configures itself with the execution plan and loads the code, it starts to compute the GLA for its local data.
- When a worker completes the computation of the local GLA, it first communicates this to the coordinator – the coordinator uses this information to monitor how the execution evolves. If the worker is a leaf, it sends the serialized GLA to its parent in the aggregation tree immediately.
- A non-leaf node has one more step to execute. It needs to aggregate the local GLA with the GLAs of its children. For this, it first deserializes the external GLAs and then executes another round of Merge work-units. In the end, it sends the combined GLA to the parent.
- The worker at the root of the aggregation tree calls the method Terminate before sending the final GLA to the coordinator who passes it further to the client who sent the job.

Experimental Data

- CREATE TABLE UserVisits (
  - sourceIP VARCHAR(16), // **internally INT (4 bytes)**
  - destURL VARCHAR(100),
  - visitDate DATE, // **internally INT (4 bytes)**
  - adRevenue FLOAT,
  - userAgent VARCHAR(64),
  - countryCode VARCHAR(3),
  - languageCode VARCHAR(6),
  - searchWord VARCHAR(32),
  - duration INT);
- **155 million tuples, approx. 20GB**

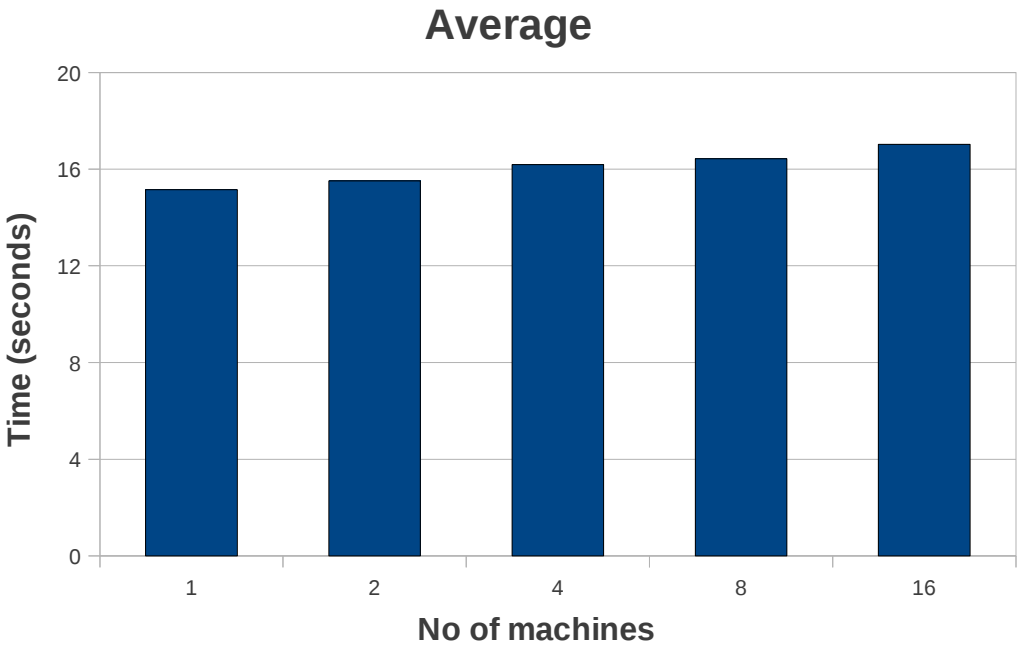
Experimental Setup

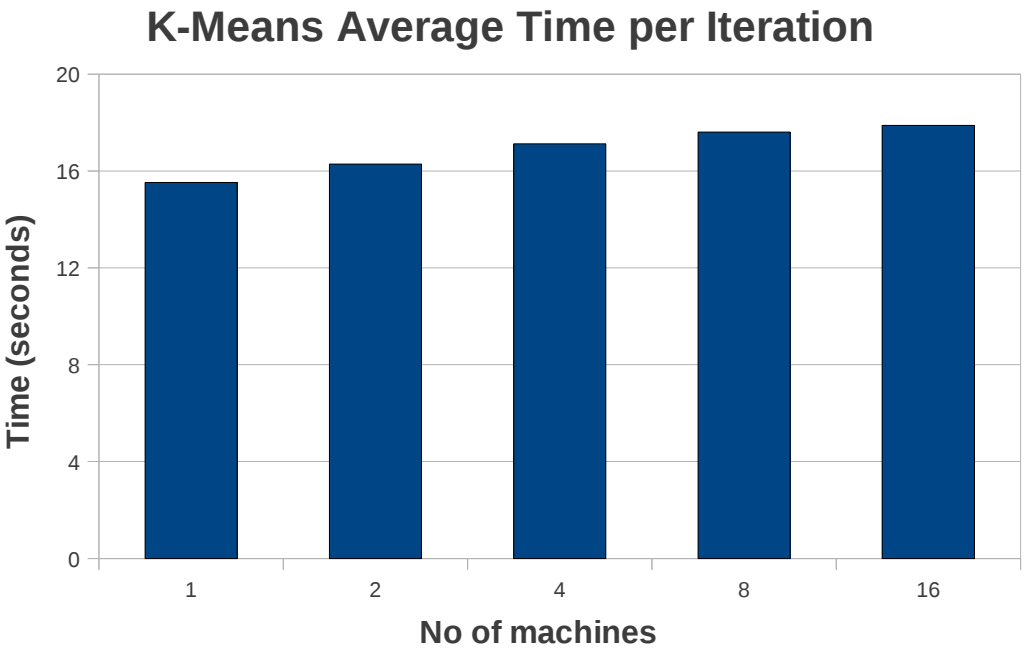
- 17 node cluster
  - 1 coordinator node, 16 worker nodes
  - CPU: 4 AMD Opteron cores @ 2.4GHz
  - Memory: 4GB
  - Disk: 1 hard-disk @ 50MB/s bandwidth
  - Network: 1Gb/s (125GB/s) switch
  - Ubuntu 7.4 Server, kernel 2.6.20-16 (32-bit)
- Data
  - One UserVisits instance (20GB) per node
  - Total maximum 320GB

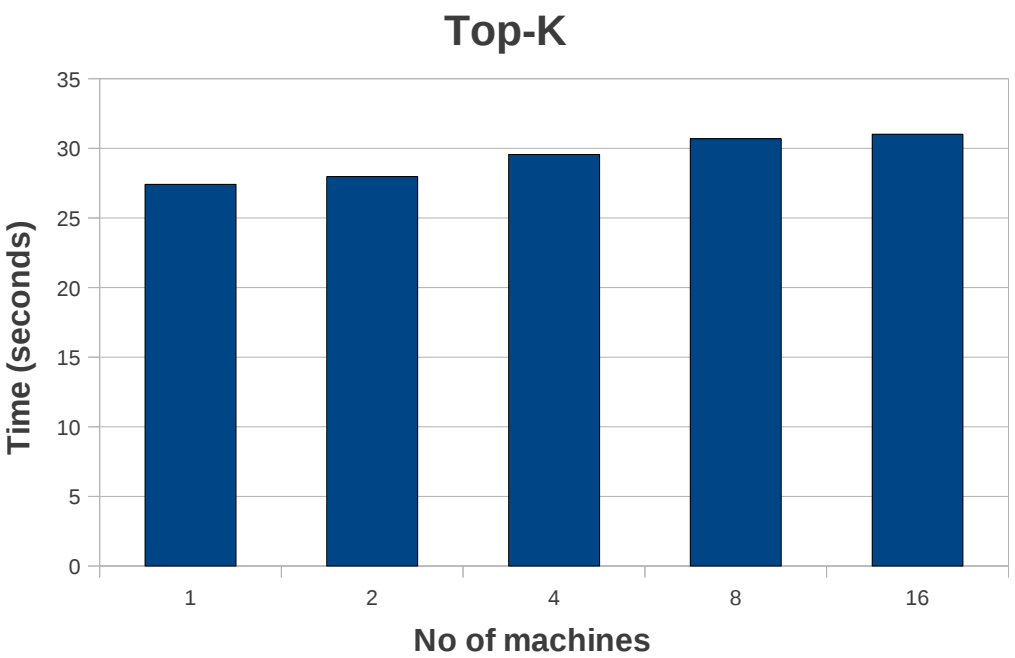
Experimental Tasks

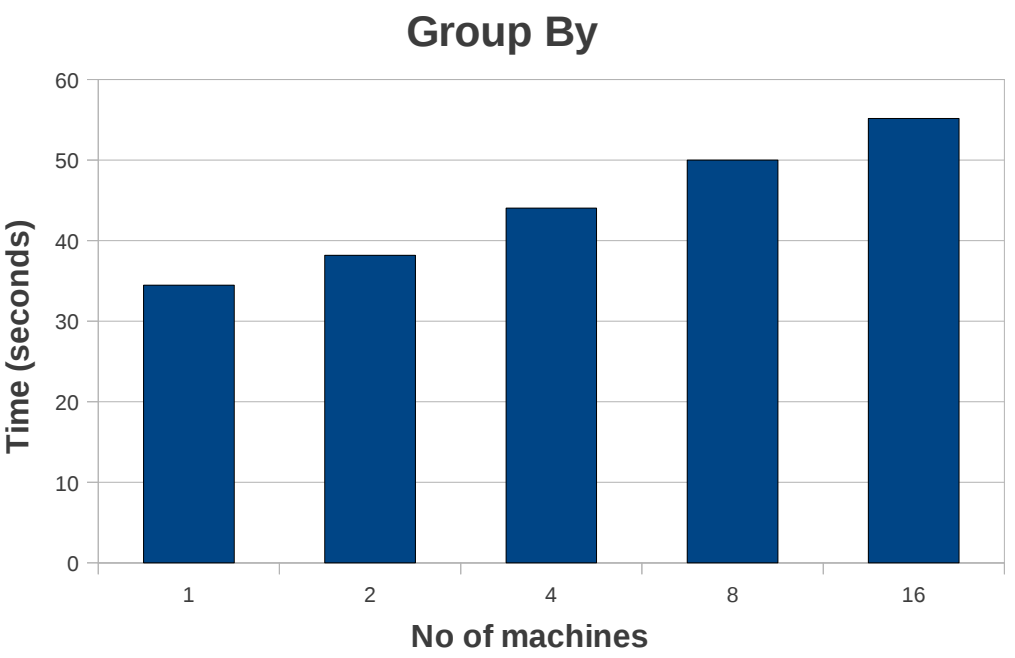
- **Average**
  - computes the average time a user spends on a web page
  - SELECT AVG(duration) FROM UserVisits
- **K-Means**
  - calculates the five most representative (5 centers) ad revenues
- **Top-K**
  - determines the users who generated the largest one hundred (top-100) ad revenues on a single visit
  - SELECT TOP 100 sourceIP, adRevenue FROM UserVisits ORDER BY adRevenue DESC
- **Group By**
  - computes the ad revenue generated by a user across all the visited web pages
  - SELECT SUM(adRevenue) FROM UserVisits GROUP BY sourceIP

Experimental Results







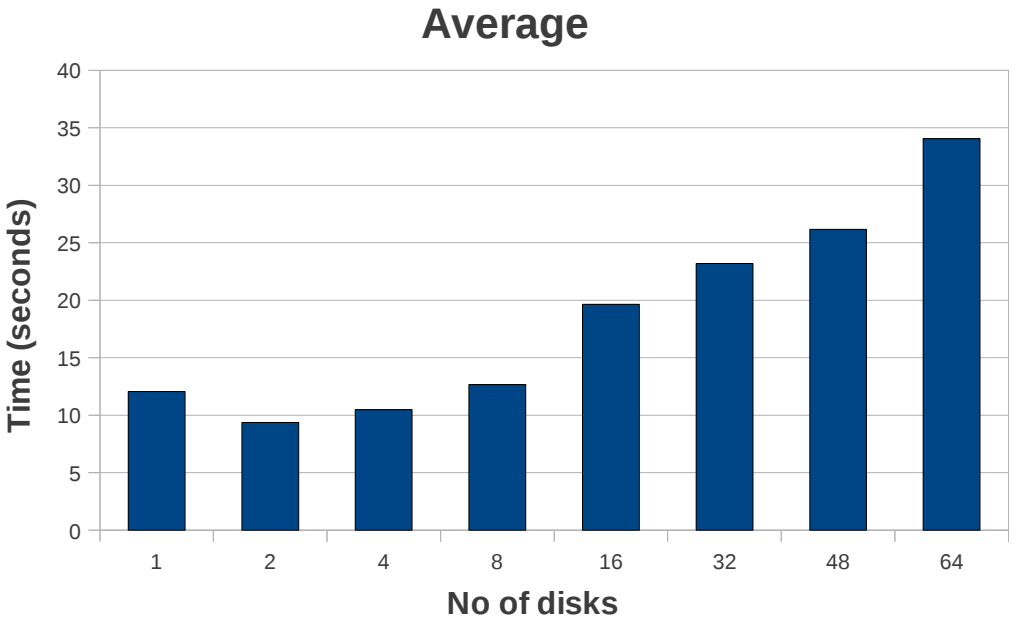


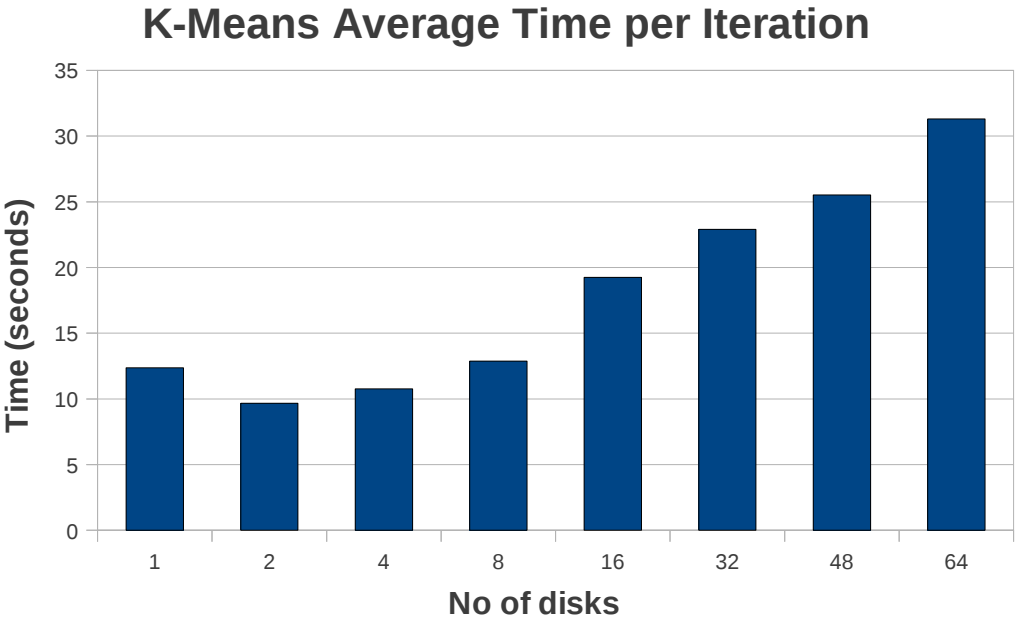
Conclusions and Future Work

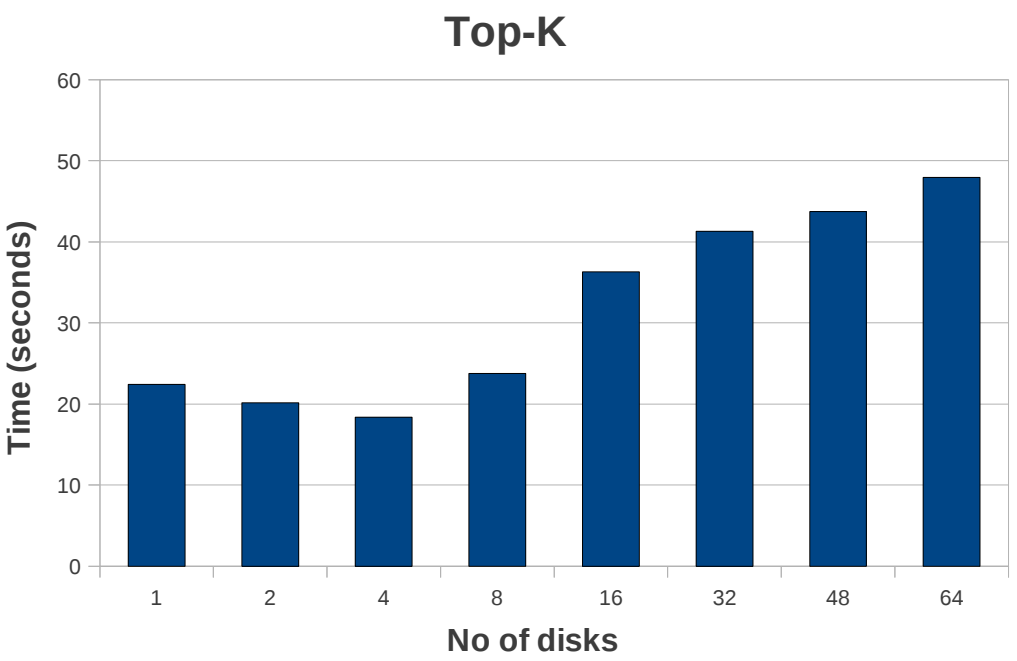
- GLA
  - Complex aggregates
  - Intuitive interface
  - User code
- GLADE
  - Parallel
  - Efficient
  - Scalable
- Library of template aggregates
- High-level language
- Interface between GLAs
- Approximate query processing & online aggregation
- Fault-tolerance

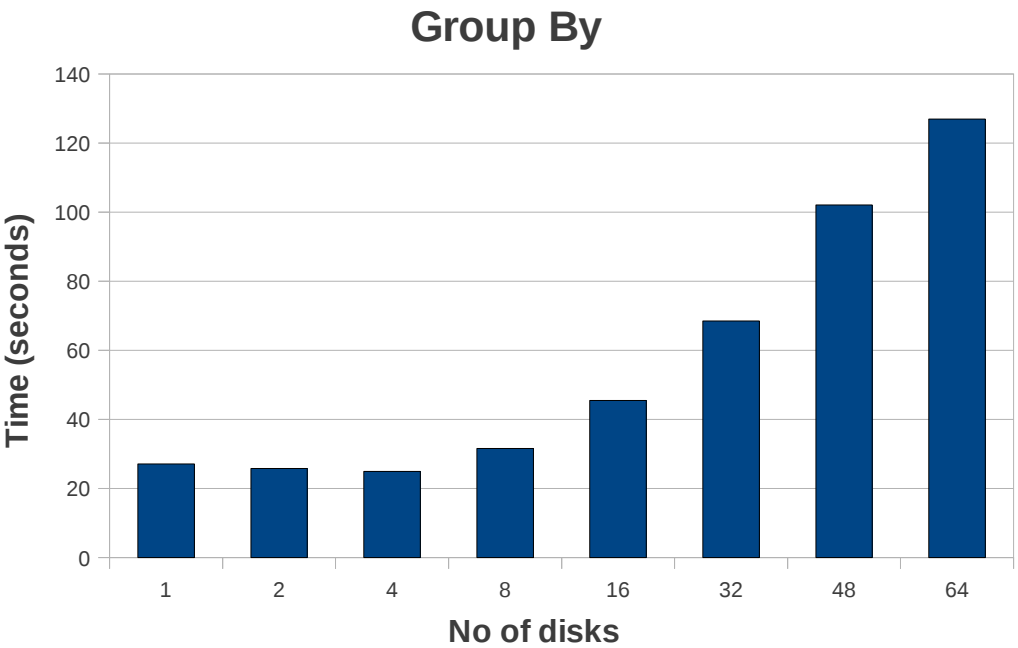
Questions ???

Experimental Results









GLA vs. Map-Reduce

- GLA
  - State well-defined
  - Interface
    - Init
    - Accumulate
    - Merge
    - Terminate
  - Turing complete
  - Runtime executes only user code
  - Only point-to-point communication
  - More intuitive
  - Algorithms for complex analytics already defined
- Map-Reduce
  - No state (only key-value pairs)
  - Interface
    - Map
    - Combine
    - Reduce
  - Turing complete
  - Runtime executes sorting & grouping
  - All-to-all communication
  - Aggregate computation difficult to express
    - Extra merging job
    - Single reducer
    - Extra communication

Hadoop vs GLADE