GLADE: A Scalable Framework for Efficient Analytics

Florin Rusu
University of California, Merced

UCMERCED

Motivation and Objective

- Large scale data processing
  - Map-Reduce is standard technique
    - Targeted to distributed index computation
  - Hadoop is standard system
    - **Scalable, but inefficient**
- Design and implement a system for aggregate computation
  - Large scale data
  - Exact and approximate
  - Complex aggregates
  - Architecture-independent
  - **Efficient and scalable**

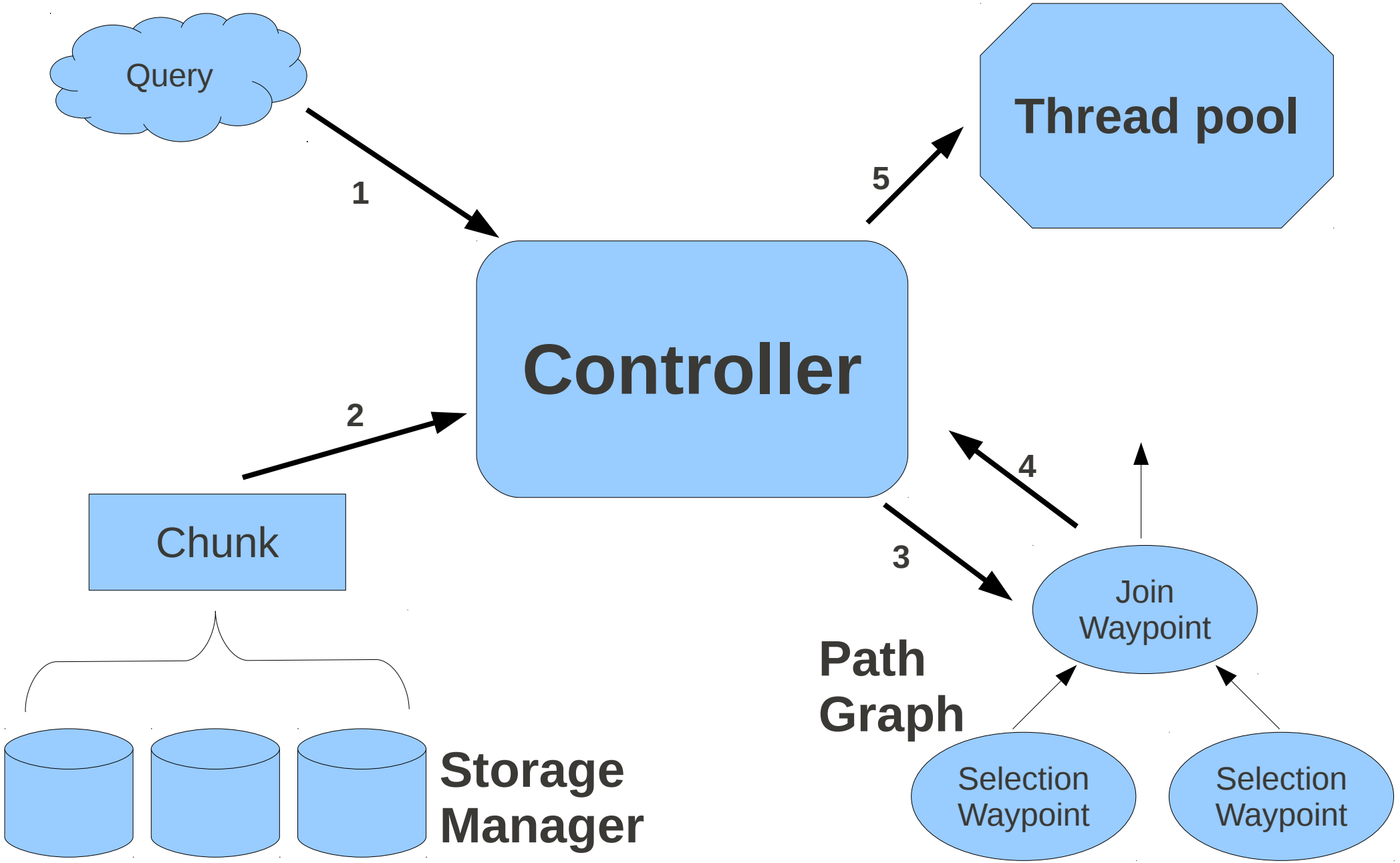
Outline

- DataPath Query Execution Engine
  - Storage Manager
  - Waypoints & Work-units
- Generalized Linear Aggregates (GLA)
- GLADE
  - Shared-Memory
  - Shared-Nothing
- GLA vs. Map-Reduce

DataPath [Arumugam et al. 2010]

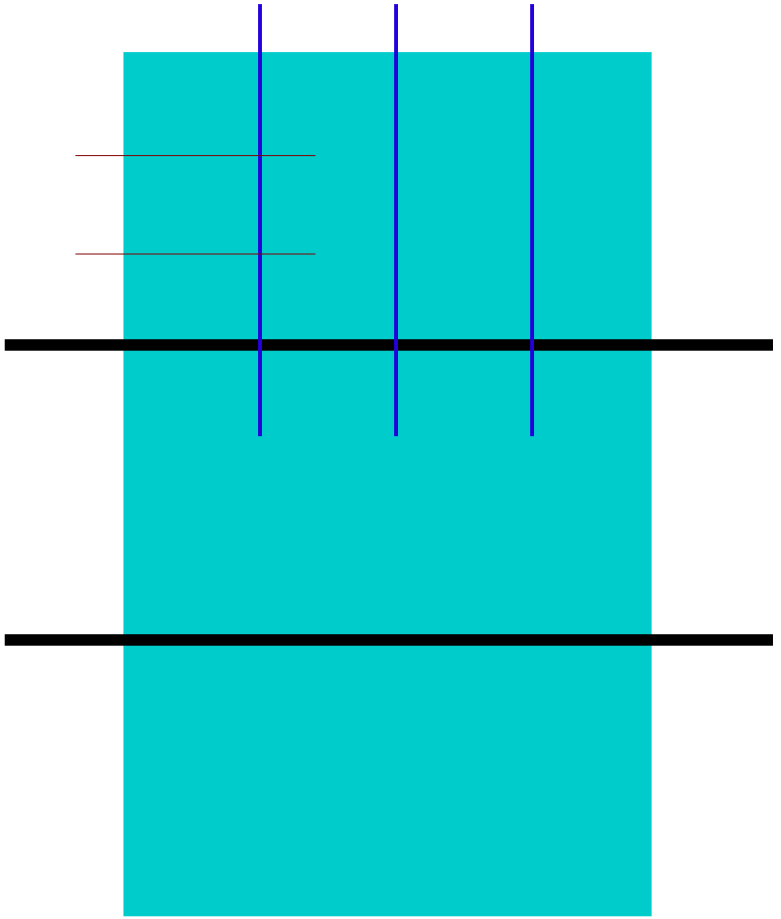
- Multi-query analytical execution engine
  - Shared scan & processing
- Push-driven execution
  - Data pushed through operators at speed read from disk
- Chunk-at-a-time processing [MapReduce 2004]
  - Sequential scan (large pages)
  - Vectorized execution (loop unrolling) [MonetDB/X100 2007]
- Runtime code generation, compilation, loading, and execution
  - M4 templates instantiated into C code with query attribute types and arithmetical & logical expressions

Architecture



Storage Manager

- Table (relation) layout
  - Horizontal partitioning into ***chunks***
- Chunk layout
  - Vertical partitioning on ***columns***
  - Limit on number of rows (2 million)
- Column layout
  - ***Page*** size (512KB)
  - Fixed size
    - Array of data
  - Variable size
    - Array of indexes
    - Data



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
  - Storage Manager
  - Waypoints & Work-units
- Generalized Linear Aggregates (GLA)
- GLADE
  - Shared-Memory
  - Shared-Nothing
- GLA vs. Map-Reduce

User-Defined Aggregates (UDA)

[Wang and Zaniolo 2000, Cohen 2006]

```
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 = 1;
    }
    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 = 1;
    }
    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
  - Storage Manager
  - Waypoints & Work-units
- Generalized Linear Aggregates (GLA)
- GLADE
  - Shared-Memory
  - Shared-Nothing
- GLA vs. Map-Reduce

GLADE

- GLA Distributed Engine
- User defines GLA
- User invokes GLA with data on DataPath
  - Execute right near data
  - Take advantage of full parallelism
    - Single machine (GLADE Shared-Memory)
    - Multiple machines (GLADE Shared-Nothing)
- User gets back computed GLA
- Hadoop + Map-Reduce :: **DataPath + GLA**

GLADE Shared-Memory

- GLA Waypoint
  - State
    - List of GLAs
  - Work-units
    - Accumulate
    - Merge

- When a chunk needs to be processed, the GLA Waypoint extracts a GLA state from the list and passes it together with the chunk to the Accumulate work-unit. The task executed by the work-unit is to call Accumulate for each tuple such that the GLA is updated with all the tuples in the chunk. If no GLA state is passed with the work-unit, a new GLA is created and initialized (Init) inside the work-unit, such that a GLA is always sent back to the GLA Waypoint.
- When all the chunks are processed, the list of GLA states has to be merged. Notice that the maximum number of GLA states that can be created is bounded by the number of threads in the pool. The merging of two GLA states is done by the Merge work-unit which calls Merge on the two states.

Experimental Data [Pavlo et al. 2009]

- 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**

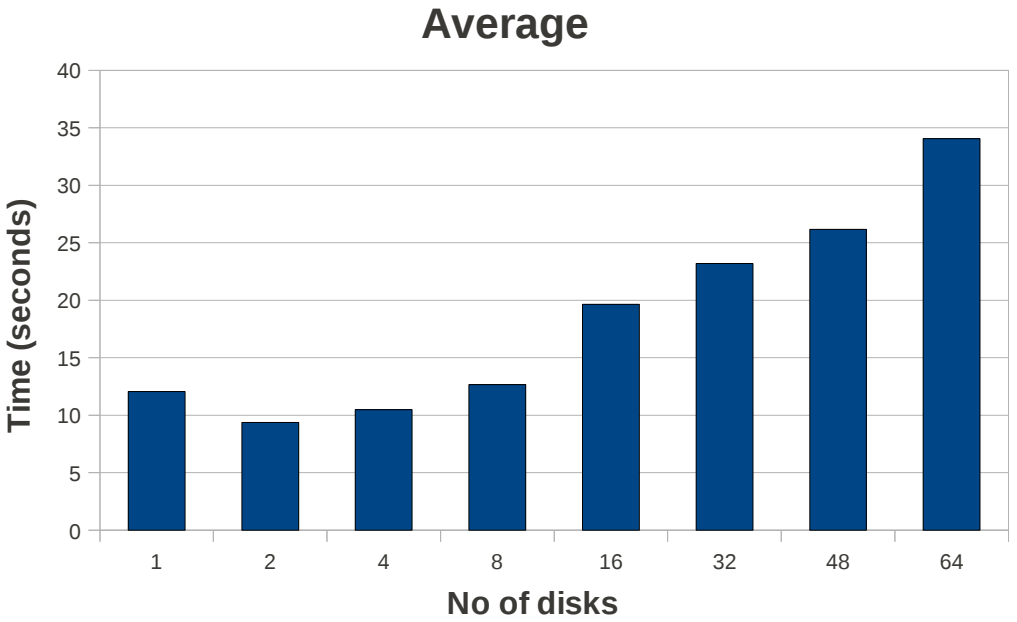
Experimental Tasks

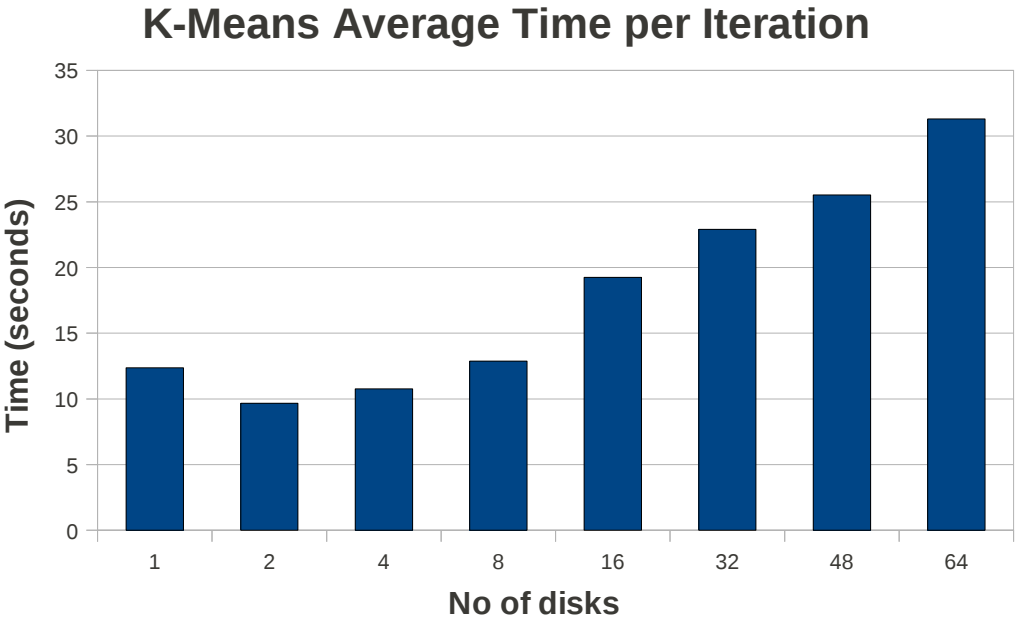
- **Average**
  - computes the average time a user spends on a web page
  - SELECT AVG(duration) FROM UserVisits
- **Group By**
  - computes the ad revenue generated by a user across all the visited web pages
  - SELECT SUM(adRevenue) FROM UserVisits GROUP BY sourceIP
- **Top-K**
  - determines the users 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
- **K-Means**
  - calculates the five most representative (5 centers) ad revenues

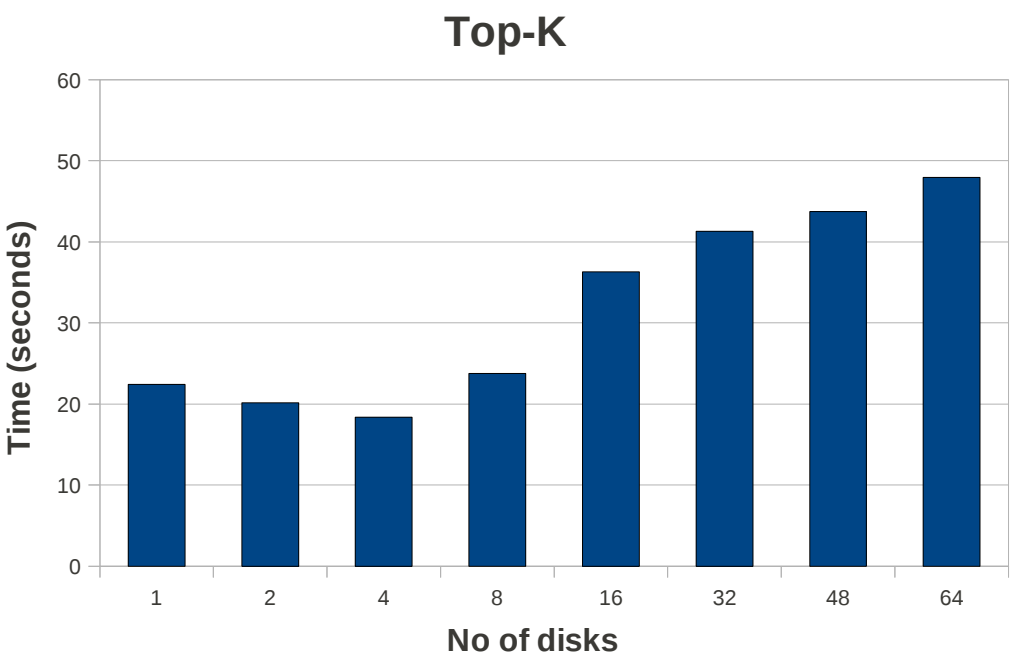
Experimental Setup

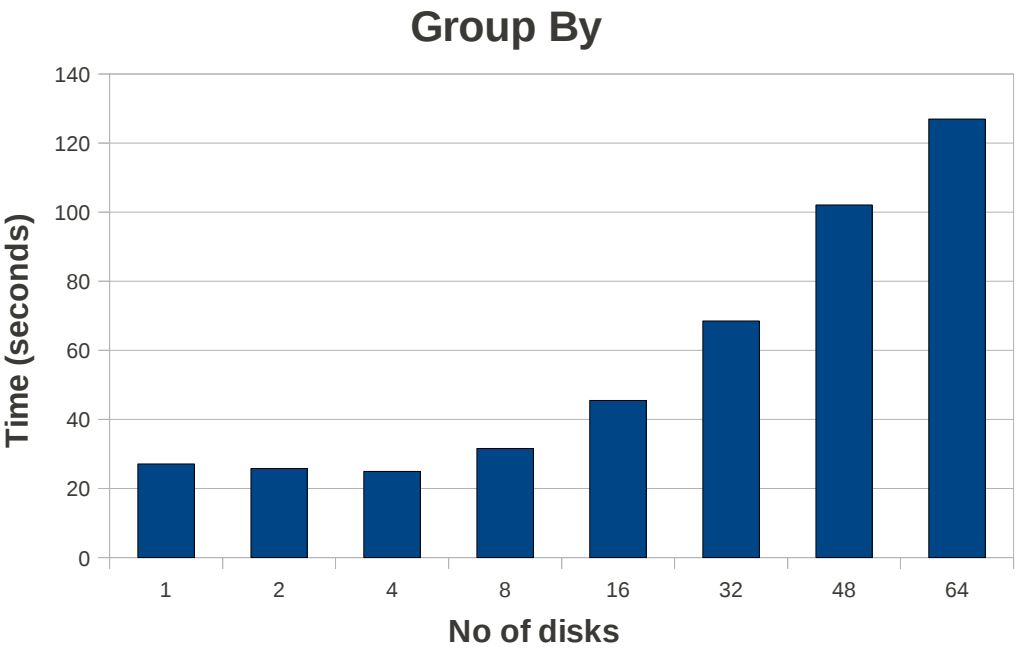
- Mid-level server
  - CPU: 48 AMD Opteron cores @ 1.9GHz
  - Memory: 256GB
  - Disk: 76 hard-disks
    - 3 RAID controllers
    - 50MB/s disk bandwidth
  - Ubuntu 10.04 SMP Server, kernel 2.6.32-26 (64-bit)
- Data
  - One UserVisits instance (20GB) per disk
  - Total maximum 1.3TB

Experimental Results

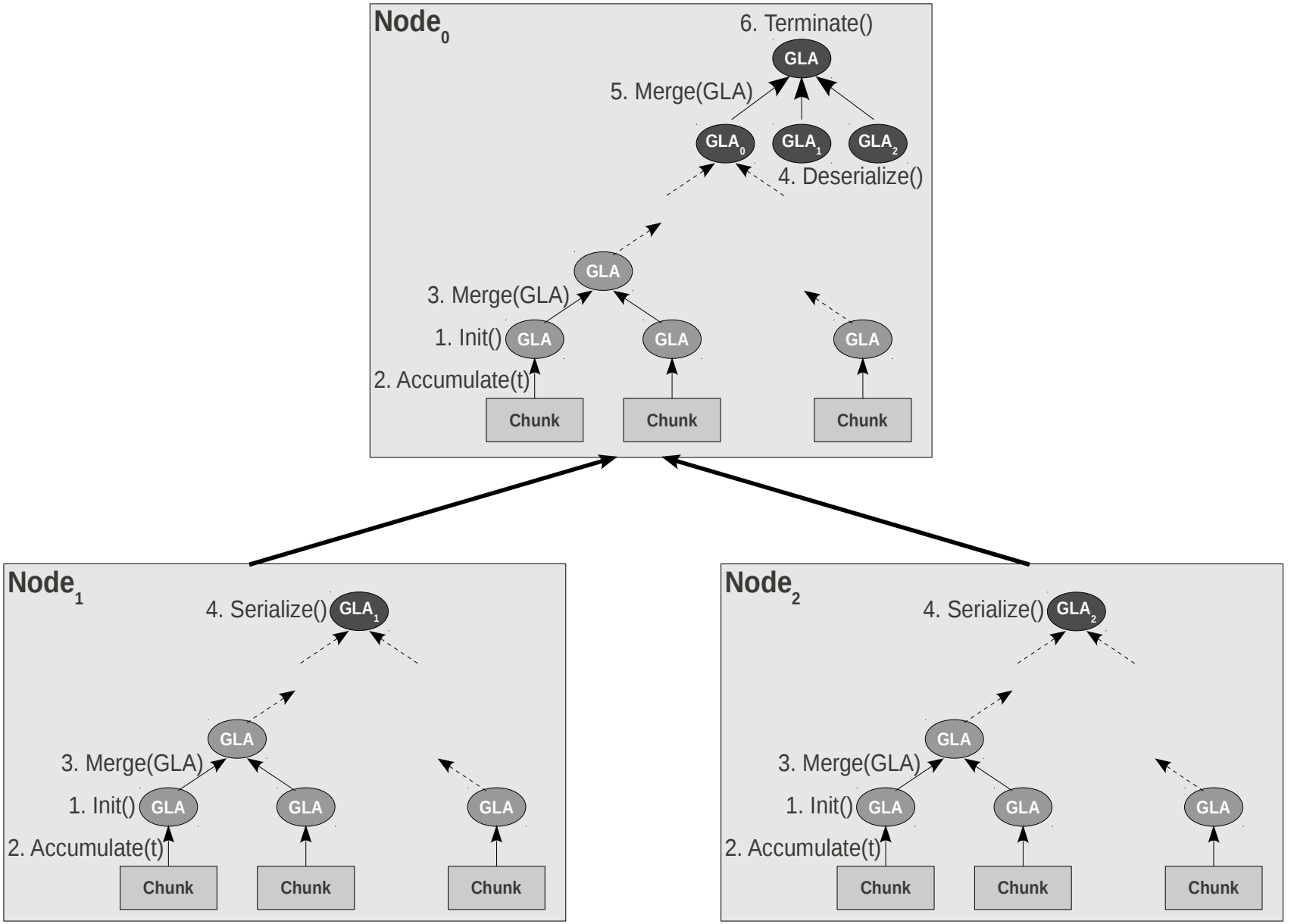








GLADE Shared-Nothing



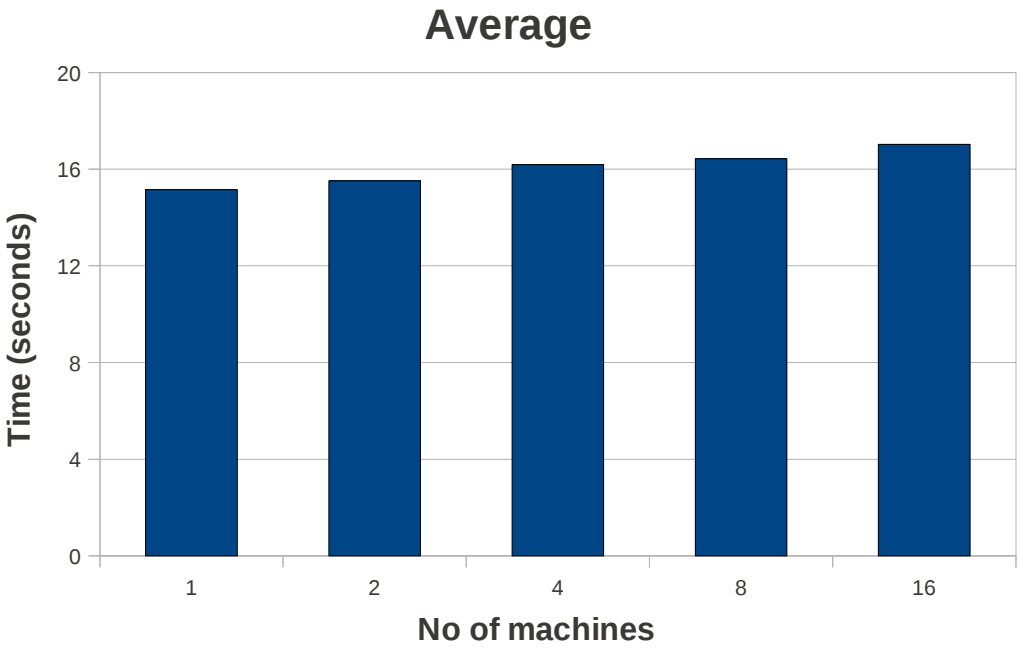
Query Execution

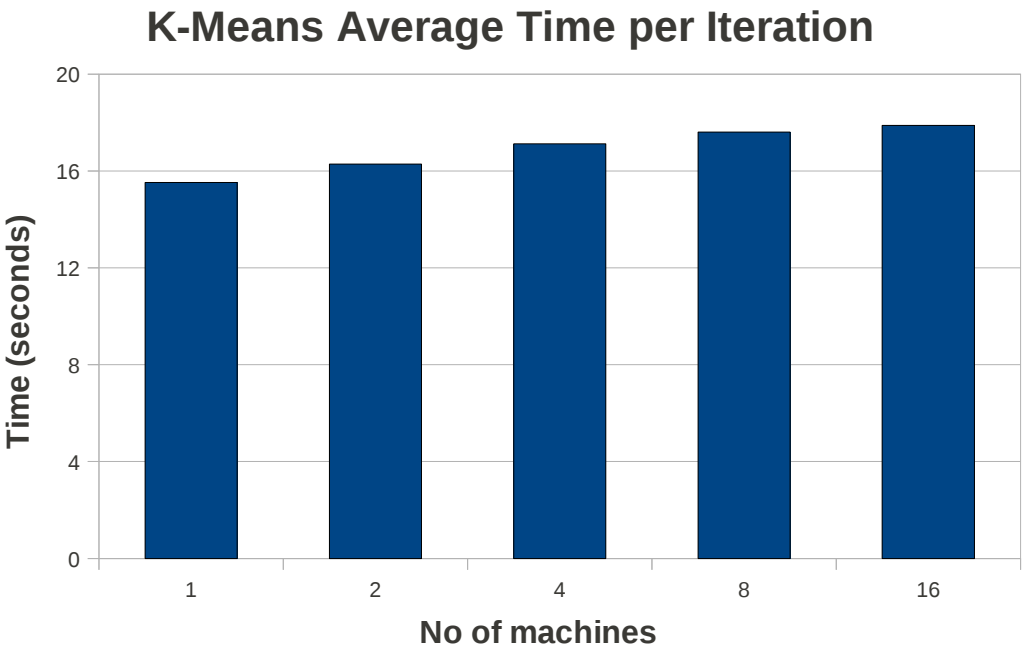
- The coordinator generates the code to be executed at each waypoint in the DataPath execution plan. A single execution plan is used for all the workers.
- The coordinator creates an aggregation tree connecting all the workers. The tree is used for in-network aggregation of the GLAs.
- The execution 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. This happens exactly in the same manner as for GLADE Shared-Memory.
- 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.

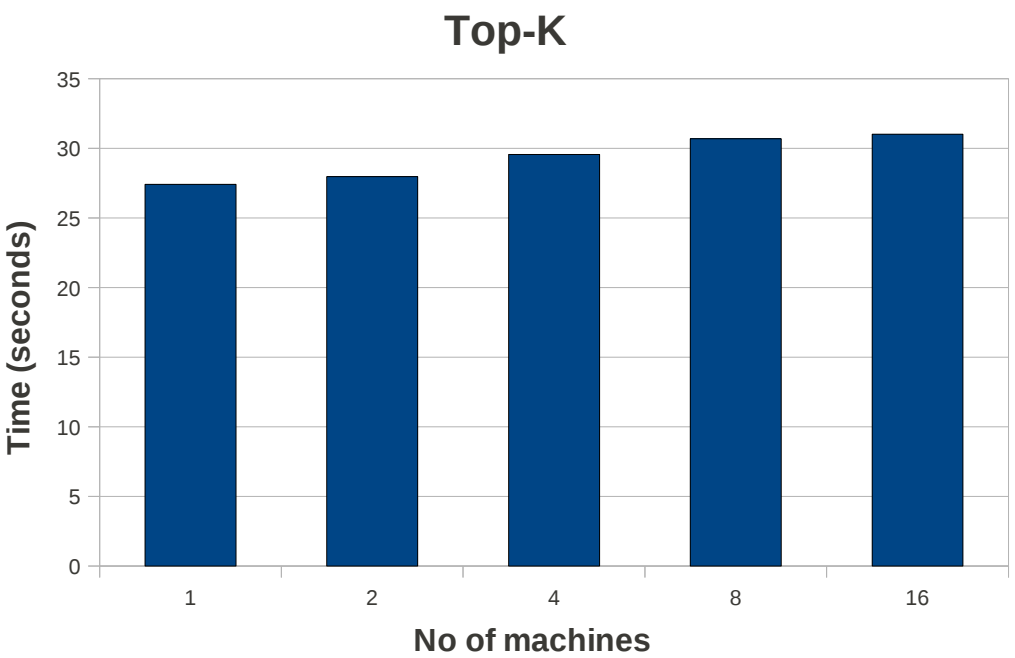
Experimental Setup

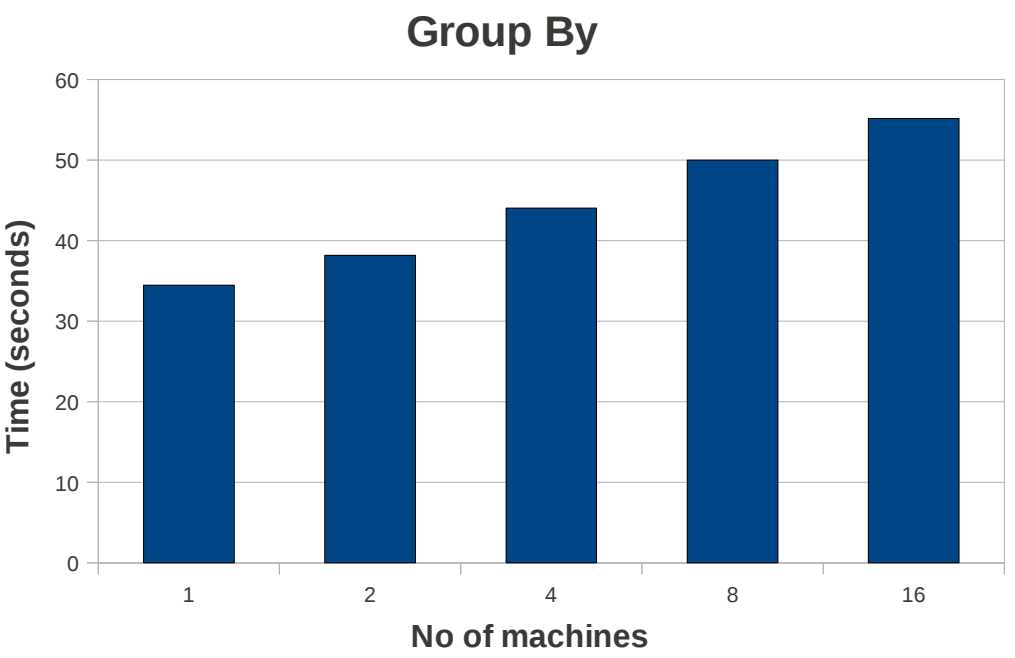
- 17 node cluster
  - 1 coordinator node, 16 worker nodes
  - CPU: 4 AMD Opteron cores @ 2.4GHz
  - Memory: 4GB
  - Disk: 1 hard-disk @ 50MB/s bandwidth
  - Network: 1Gb/s (125GB/s) switch
  - Ubuntu 7.4 Server, kernel 2.6.20-16 (32-bit)
- Data
  - One UserVisits instance (20GB) per node
  - Total maximum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

Experimental Results









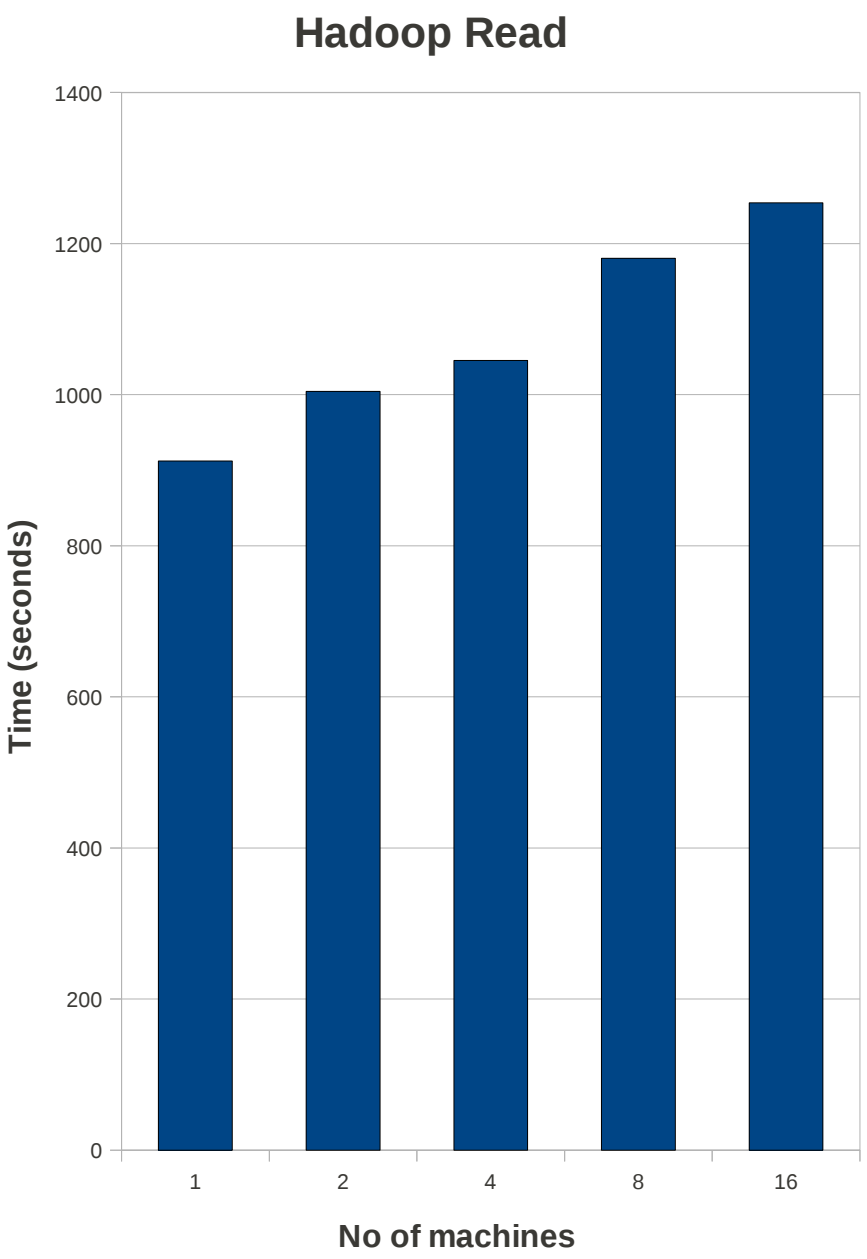
Outline

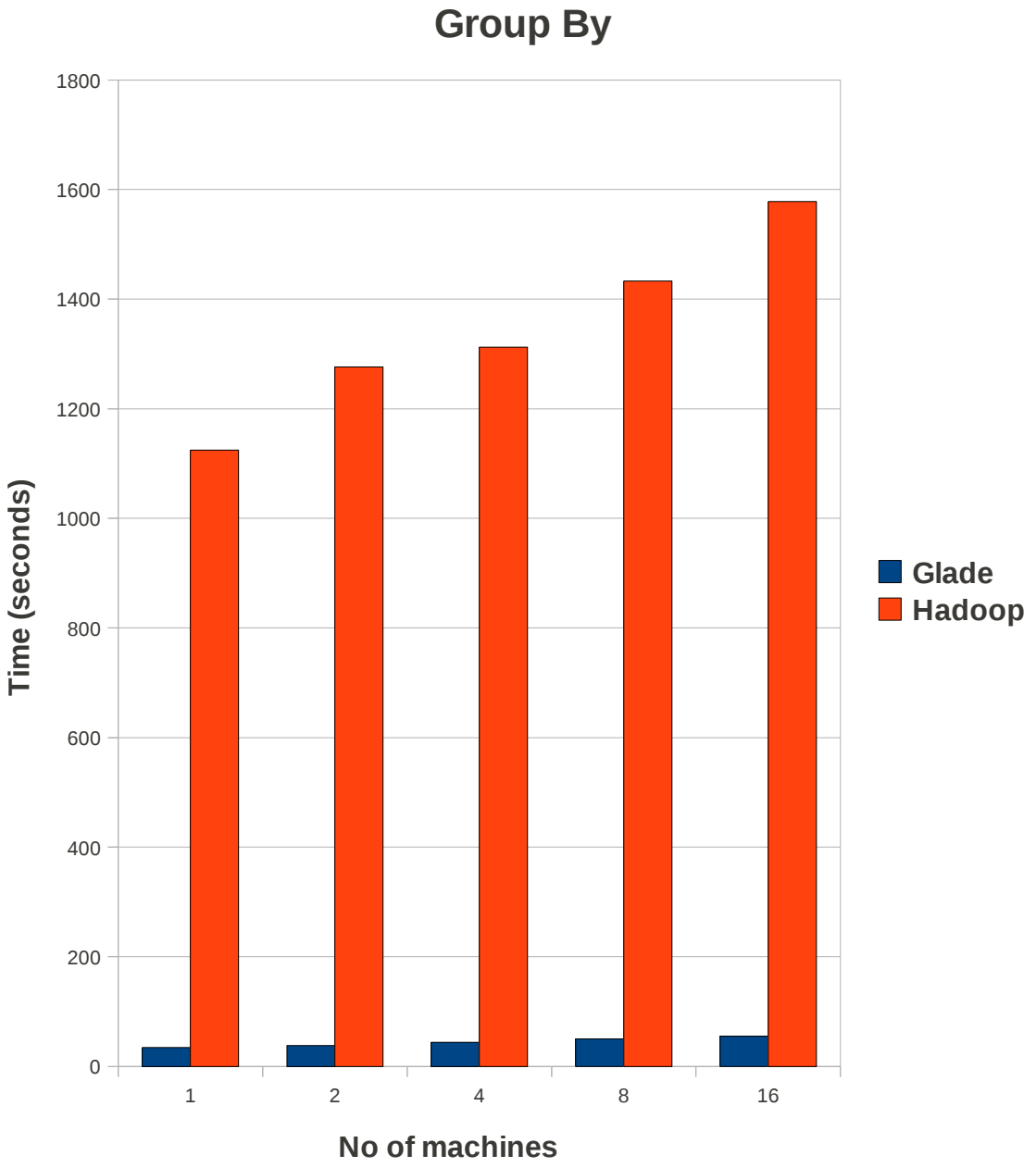
- DataPath Query Execution Engine
  - Storage Manager
  - Waypoints & Work-units
- Generalized Linear Aggregates (GLA)
- GLADE
  - Shared-Memory
  - Shared-Nothing
- GLA vs. Map-Reduce

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





Conclusions and Future Work

- GLA
  - Express complex aggregates through intuitive interface
  - User code
- GLADE
  - Architecture-independent
  - Efficient
    - I/O bound
  - Scalable
- Library of template aggregates
- GLA extension for approximate query processing
  - Online aggregation
- Fault-tolerance
- Bulk-loading

Collaborators:
Alin Dobra, University of Florida

Questions ???