Today…

- Last session
  - Pregel

- Today’s session
  - Dryad and GraphLab

- Announcement:
  - Project Phases I-A and I-B are due today
Objectives

Discussion on Programming Models

Why parallelism?

Parallel computer architectures

Traditional models of parallel programming

Examples of parallel processing

Message Passing Interface (MPI)

MapReduce

Pregel, Dryad and GraphLab

Last 3 Sessions
Dryad
Dryad

- In this part, the following concepts of Dryad will be described:
  - Dryad Model
  - Dryad Organization
  - Dryad Description Language and An Example Program
  - Fault Tolerance in Dryad
Dryad

- In this part, the following concepts of Dryad will be described:
  - Dryad Model
  - Dryad Organization
  - Dryad Description Language and An Example Program
  - Fault Tolerance in Dryad
Dryad

- Dryad is a general purpose, high-performance, distributed computation engine

- Dryad is designed for:
  - High-throughput
  - Data-parallel computation
  - Use in a private datacenter

- Computation is expressed as a directed-acyclic-graph (DAG)
  - Vertices represent programs
  - Edges represent data channels between vertices
Unix Pipes vs. Dryad DAG

- **Unix Pipes: 1-D**
  
grep | sed | sort | awk | perl

- **Dryad: 2-D**
  
grep^{1000} | sed^{500} | sort^{1000} | awk^{500} | perl^{50}
Dryad

In this part, the following concepts of Dryad will be described:

- Dryad Model
- Dryad Organization
- Dryad Description Language and An Example Program
- Fault Tolerance in Dryad
Dryad System Organization

- There are 3 roles for machines in Dryad
  - Job Manager (JM)
  - Name Server (NS)
  - Daemon (D)
Program Execution (1)

- **The Job Manager (JM):**
  - Creates the job communication graph (job schedule)
  - Contacts the **NS** to determine the number of **Ds** and the topology
  - Assigns **Vs** to each **D** (using a simple task scheduler—not described) for execution
  - Coordinates data flow through the data plane

- Data is distributed using a distributed storage system that shares with the Google File System some properties (e.g., data are split into chunks and replicated across machines)

- Dryad also supports the use of NTFS for accessing files locally
Program Execution (2)

1. Build
2. Send .exe
3. Start JM
4. Query cluster resources
5. Generate graph
6. Initialize vertices
7. Serialize vertices
8. Monitor Vertex execution

© Carnegie Mellon University in Qatar
Data Channels in Dryad

- Data items can be *shuffled* between vertices through *data channels*

- Data channels can be:
  - Shared Memory FIFOs (intra-machine)
  - TCP Streams (inter-machine)
  - SMB/NTFS Local Files (temporary)
  - Distributed File System (persistent)

- The performance and fault tolerance of these mechanisms vary

- Data channels are abstracted for maximum flexibility
In this part, the following concepts of Dryad will be described:

- Dryad Model
- Dryad Organization
- Dryad Description Language and An Example Program
- Fault Tolerance in Dryad
Dryad Graph Description Language

- Here are some operators in the Dryad graph description language:

\[ A^n = A \times A \times \ldots \times A \]

\[ A \leq B \]

\[ A \gg B \]

- (Cloning)
- (Pointwise Composition)
- (Bipartite Composition)
- (Merge)

© Carnegie Mellon University in Qatar
Skyserver SQL Query (Q18):
- Find all the objects in the database that have neighboring objects within 30 arc seconds such that at least one of the neighbors has a color similar to the primary object’s color.

There are two tables involved:
- photoObjAll and it has 354,254,163 records
- Neighbors and it has 2,803,165,372 records

For the equivalent Dryad computation, they extracted the columns of interest into two binary files, “ugriz.bin” and “neighbors.bin”
Example Program in Dryad (2)

- [distinct]
- [merge outputs]

```sql
select
table u.objid
from u join <temp>
where
  u.objid = <temp>.neighborobjid and
  |u.color - <temp>.color| < d
```
Example Program in Dryad (3)

- Here is the corresponding Dryad code:

```dryad
GraphBuilder XSet = moduleX^N;
GraphBuilder DSet = moduleD^N;
GraphBuilder MSet = moduleM^(N*4);
GraphBuilder SSet = moduleS^(N*4);
GraphBuilder YSet = moduleY^N;
GraphBuilder HSet = moduleH^1;

GraphBuilder XInputs = (ugriz1 >= XSet) || (neighbor >= XSet);
GraphBuilder YInputs = ugriz2 >= YSet;
GraphBuilder XToY = XSet >= DSet >> MSet >= SSet;

for (i = 0; i < N*4; ++i)
{ XToY = XToY || (SSet.GetVertex(i) >= YSet.GetVertex(i/4)); }

GraphBuilder YToH = YSet >= HSet;
GraphBuilder HOutputs = HSet >= output;

GraphBuilder final = XInputs || YInputs || XToY || YToH || HOutputs;
```

© Carnegie Mellon University in Qatar
Dryad

In this part, the following concepts of Dryad will be described:

- Dryad Model
- Dryad Organization
- Dryad Description Language and An Example Program
- Fault Tolerance in Dryad
Fault Tolerance in Dryad (1)

- Dryad is designed to handle two types of failures:
  - Vertex failures
  - Channel failures

- Vertex failures are handled by the JM and the failed vertex is re-executed on another machine

- Channel failures cause the preceding vertex to be re-executed
Fault Tolerance in Dryad (2)

Completed vertices

X[0]  X[1]  X[3]

Slow vertex

X[2]

Duplicate vertex

X'[2]

Duplication Policy = f(running times, data volumes)
GraphLab

- In this part, the following concepts of GraphLab will be described:
  - Motivation for GraphLab
  - GraphLab Data Model and Update Mechanisms
  - Scheduling in GraphLab
  - Consistency Models in GraphLab
  - PageRank in GraphLab
GraphLab

In this part, the following concepts of GraphLab will be described:

- Motivation for GraphLab
- GraphLab Data Model and Update Mechanisms
- Scheduling in GraphLab
- Consistency Models in GraphLab
- PageRank in GraphLab
Motivation for GraphLab

- Shortcomings of MapReduce
  - Interdependent data computation difficult to perform
  - Overheads of running jobs iteratively – disk access and startup overhead
  - Communication pattern is not user definable/flexible

- Shortcomings of Pregel
  - BSP model requires synchronous computation
  - One slow machine can slow down the entire computation considerably

- Shortcomings of Dryad
  - Very flexible but steep learning curve for the programming model
GraphLab

- GraphLab is a framework for parallel machine learning
GraphLab

- In this part, the following concepts of GraphLab will be described:
  - Motivation for GraphLab
  - GraphLab Data Model and Update Mechanisms
  - Scheduling in GraphLab
  - Consistency Models in GraphLab
  - PageRank in GraphLab
Data Graph

- A graph in GraphLab is associated with data at every vertex and edge.

- Arbitrary blocks of data can be assigned to vertices and edges.
Update Functions

- The data graph is modified using update functions
  - The update function can modify a vertex $v$ and its neighborhood, defined as the scope of $v$ ($S_v$)
Shared Data Table

- Certain algorithms require global information that is shared among all vertices (Algorithm Parameters, Statistics, etc.)
  - GraphLab exposes a **Shared Data Table (SDT)**

- SDT is an associative map between keys and arbitrary blocks of data
  - \( T[\text{Key}] \rightarrow \text{Value} \)

- The shared data table is updated using the *sync mechanism*
Sync Mechanism

- Similar to Reduce in MapReduce
  - User can define fold, merge and apply functions that are triggered during the global sync mechanism

- Fold function allows the user to sequentially aggregate information across all vertices

- Merge optionally allows user to perform a parallel tree reduction on the aggregated data collected during the fold operation

- Apply function allows the user to finalize the resulting value from the fold/merge operations (such as normalization etc.)
In this part, the following concepts of GraphLab will be described:

- Motivation for GraphLab
- GraphLab Data Model and Update Mechanisms
- Scheduling in GraphLab
- Consistency Models in GraphLab
- PageRank in GraphLab
Scheduling in GraphLab (1)

- The scheduler determines the order that vertices are updated.

The process repeats until the scheduler is empty.
Scheduling in GraphLab (2)

- An **update schedule** defines the order in which update functions are applied to vertices
  - A parallel data-structure called the **scheduler** represents an abstract list of tasks to be executed in Graphlab
- **Base (Vertex) schedulers in GraphLab**
  - Synchronous scheduler
  - Round-robin scheduler
- **Job Schedulers in GraphLab**
  - FIFO scheduler
  - Priority scheduler
- Custom schedulers can be defined by the **set scheduler**

**Termination Assessment**
- If the scheduler has no remaining tasks
- Or, a termination function can be defined to check for convergence in the data
GraphLab

- In this part, the following concepts of GraphLab will be described:
  - Motivation for GraphLab
  - GraphLab Data Model and Update Mechanisms
  - Scheduling in GraphLab
  - Consistency Models in GraphLab
  - PageRank in GraphLab
Need for Consistency Models

- How much can computation overlap?
Consistency Models in GraphLab

- GraphLab guarantees **sequential consistency**
  - Guaranteed to give the same result as a sequential execution of the computational steps

- User-defined consistency models
  - Full Consistency
  - Vertex Consistency
  - Edge Consistency
GraphLab

- In this part, the following concepts of GraphLab will be described:
  - Motivation for GraphLab
  - GraphLab Data Model and Update Mechanisms
  - Scheduling in GraphLab
  - Consistency Models in GraphLab
  - PageRank in GraphLab
PageRank (1)

- **PageRank** is a link analysis algorithm.

- The rank value indicates an importance of a particular web page.

- A hyperlink to a page counts as a vote of support.

- A page that is linked to by many pages with high PageRank receives a high rank itself.

- A PageRank of 0.5 means there is a 50% chance that a person clicking on a random link will be directed to the document with the 0.5 PageRank.
PageRank (2)

- Iterate:

\[ R[i] = \alpha + (1 - \alpha) \sum_{(j,i) \in E} \frac{1}{L[j]} R[j] \]

- Where:
  - \( \alpha \) is the random reset probability
  - \( L[j] \) is the number of links on page \( j \)

\[ R[5] = \alpha + (1 - \alpha) \left( \frac{1}{3} R[1] + \frac{1}{1} R[4] \right) \]
PageRank Example in GraphLab

- PageRank algorithm is defined as a per-vertex operation working on the scope of the vertex

```plaintext
pagerank(i, scope){
  // Get Neighborhood data
  (R[i], W_{ij}, R[j]) \leftarrow \text{scope};
  // Update the vertex data
  R[i] \leftarrow \alpha + (1-\alpha) \sum_{j \in N[i]} W_{ij} \times R[j];
  // Reschedule Neighbors if needed
  if R[i] changes then
    \text{reschedule_neighbors_of}(i);
}
```

© Carnegie Mellon University in Qatar
How MapReduce, Pregel, Dryad and GraphLab Compare Against Each Other?
## Comparison of the Programming Models

<table>
<thead>
<tr>
<th>Programming Model</th>
<th>MapReduce</th>
<th>Pregel</th>
<th>Dryad</th>
<th>GraphLab</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fixed Functions</strong> – Map and Reduce</td>
<td>Supersteps over a data graph with messages passed</td>
<td>DAG with program vertices and data edges</td>
<td>Data graph with shared data table and update functions</td>
<td></td>
</tr>
<tr>
<td><strong>Concurrent execution of tasks within map and reduce phases</strong></td>
<td>Concurrent execution of user functions over vertices within a superstep</td>
<td>Concurrent execution of vertices during a stage</td>
<td>Concurrent execution of non-overlapping scopes, defined by consistency model</td>
<td></td>
</tr>
<tr>
<td><strong>Distributed file system</strong></td>
<td>Distributed file system</td>
<td>Flexible data channels: Memory, Files, DFS etc.</td>
<td>Undefined – Graphs can be in memory or on disk</td>
<td></td>
</tr>
<tr>
<td><strong>Fixed Phases</strong> – HDFS Locality based map task assignment</td>
<td>Partitioned Graph and Inputs assigned by assignment functions</td>
<td>Job and Stage Managers assign vertices to available daemons</td>
<td>Pluggable schedulers to schedule update functions</td>
<td></td>
</tr>
<tr>
<td><strong>DFS replication + Task reassignment / Speculative execution of Tasks</strong></td>
<td>Checkpointing and superstep re-execution</td>
<td>Vertex and Edge failure recovery</td>
<td>Synchronous and asychronous snapshots</td>
<td></td>
</tr>
<tr>
<td><strong>Developed by</strong></td>
<td>Google</td>
<td>Google</td>
<td>Microsoft</td>
<td>Carnegie Mellon</td>
</tr>
</tbody>
</table>
References

- This presentation has elements borrowed from various papers and presentations:

- Papers:

- Presentations:
  - Dryad Presentation at Berkeley by M. Budiu: [http://budiu.info/work/dryad-talk-berkeley09.pptx](http://budiu.info/work/dryad-talk-berkeley09.pptx)
  - GraphLab1 Presentation: [http://graphlab.org/uai2010_graphlab.pptx](http://graphlab.org/uai2010_graphlab.pptx)
  - GraphLab2 Presentation: [http://graphlab.org/presentations/nips-biglearn-2011.pptx](http://graphlab.org/presentations/nips-biglearn-2011.pptx)
Next Class

Distributed File Systems