Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks

Michael Isard
Microsoft Research, Silicon Valley

Mihai Budiu
Microsoft Research, Silicon Valley

Yuan Yu
Microsoft Research, Silicon Valley

Andrew Birrell
Microsoft Research, Silicon Valley

Dennis Fetterly
Microsoft Research, Silicon Valley


Peter Rutgers
February 22, 2012
Dryad

- "General-purpose distributed execution engine for coarse-grain data-parallel applications"

- Based on *dataflow graph*:
  - Vertices for (small) sequential computations
  - Edges for data channels between vertices
    - Implemented as files, TCP pipes, or shared memory
Motivation
Graph specification
Graph optimization
Using Dryad
Experiments
Conclusion
Motivation

Graph specification

Graph optimization

Using Dryad

Experiments

Conclusion
Motivation

- Recent trends:
  - Emerging large-scale internet services
  - Increasing number of cores, instead of faster cores
- Single administration
- Centralized management and control
- High-performance communication network
- Known (network) topology
Requirements

- Problems outside of scope:
  - unreliable/slow networks, distributed control of resources, authentication, access control
- Simple model to program
- Reliability of applications
- Efficiency
- Scalability
Design choices

- Scalability -> Data-parallelism
  - Specified by developer
  - Scheduling, distribution, fault-tolerance automatic

- Customizable dataflow graph
  - More general than related work (MapReduce, GPU shaders, parallel databases)
  - Low-level infrastructure to build simpler models on
Motivation

Graph specification

Graph optimization

Using Dryad

Experiments

Conclusion
System structure

- Vertices are mapped to physical resources
  - Binary + channel
  - URIs sent to daemon
- JM communicates with vertices via daemon
  - Checks status of execution, errors, amount of data in/out

Figure 1: The Dryad system organization. The job manager (JM) consults the name server (NS) to discover the list of available computers. It maintains the job graph and schedules running vertices (V) as computers become available using the daemon (D) as a proxy. Vertices exchange data through files, TCP pipes, or shared-memory channels. The shaded bar indicates the vertices in the job that are currently running.
Graph specification syntax

a. \( AS = A \uparrow n \)

b. \( BS = B \uparrow n \)

c. \( AS \geq BS \)

d. \( AS \gg BS \)

e. \( (B \geq C) \mathbin{\|} (B \geq D) \)

f. \( E = (AS \geq C \geq BS) \)

g. \( E \mathbin{\|} (AS \geq BS) \)

h. \( (A \geq C \geq D \geq B) \mathbin{\|} (A \geq F \geq B) \)
- Motivation
- Graph specification
- Graph optimization
- Using Dryad
- Experiments
- Conclusion
Optimizations

- Many vertices may fit in resources of 1 host
- Encapsulation command:
  - Replace subgraph (N vertices) with 1 new vertex
  - New vertex encapsulates subgraph
Optimizations

- Many vertices may fit in resources of 1 host
- Encapsulation command:
  - Replace subgraph (N vertices) with 1 new vertex
  - New vertex encapsulates subgraph
- Choose suitable channel type
  - Files - default
  - TCP: faster, but both jobs must be at same time
  - Memory: fastest, but both must be at same host
- Scheduler tries to satisfy requirements
Run-time refinement

- Graph divided in 'stages'
- Each stage has stage manager
  - Slow vertices re-scheduled
  - Can implement optimizations such as local aggregation steps

Figure 5: The stages of the Dryad computation from Figure 2.
- Motivation
- Graph specification
- Graph optimization
- Using Dryad
- Experiments
- Conclusion
Writing Dryad programs

- Dryad API in C++
  - API wrapper for other languages
- Can inherit from predefined vertex classes
  - map, reduce, distribute, join
- Wrapper for including existing programs
Higher-level languages

- Scripting language Nebula
  - Only specify stages
  - Each stage refers to executable
  - Combined with grep/perl for text processing

- Convert SQL query to Dryad graph
  - Future work
- Motivation
- Graph specification
- Graph optimization
- Using Dryad
- Experiments
- Conclusion
Experiments

- Performance evaluated on two applications
- SQL query
  - Database of telescope observations
  - Analyse 2.8 G of neighbouring objects
  - Small scale (10 computers)
- Data mining
  - Build histogram of search queries
  - Map-reduce pattern
  - Cluster of 1800 computers
Two graph structures tested

- Two-Pass stores intermediate results on disk
- In-Memory communicates via shared memory

Figure 8: The speedup of the SQL query computation is near-linear in the number of computers used. The baseline is relative to Dryad running on a single computer and times are given in Table 2.
Results - Data Mining

- To "verify that it works sufficiently well"
  - 10 TB of search query logs

- Original graph doesn't scale
  - Easy to try different optimizations

- Optimized setup takes 11:30 minutes
Conclusion

- Dryad: general data-parallel execution engine
- Efficient on both small and large clusters
- Easy to program, no concurrency issues
- Still adaptable for specific applications