Dryad, Map Reduce and Data-Parallel Programming

What is the problem

- QUESTION: What is the problem Map/Reduce and Dryadsolve?
  - You have a large cluster of computers
  - You have a large set of data distributed over computers
  - You have a computation over the data set you would like to do
- How can you:
  - Make it easy to write the computation
  - Make it easy to get performance from the cluster?

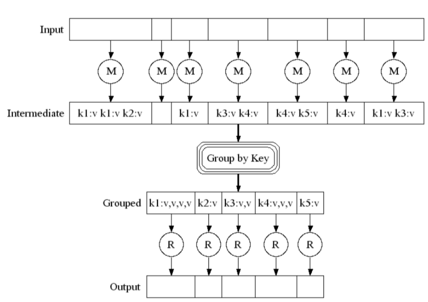
Basic idea

- Let programmers specify the computation parts
  - E.g. counting words, traversing graphs
- Let the framework handle:
  - Communication
  - Scheduling
  - Data partitioning/replication

Map/Reduce: basic model

- Input: data distributed across a set of nodes
- Map: process/select input values to intermediate values
  - map(string key, string value) → list {string key2, string value2}
- Reduce: combine all values with shared intermediate key
  - reduce(string key2, list {string values}) --> list {output}

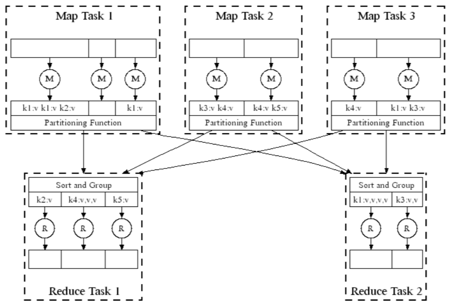
Logical Model



Example Uses

- Note: user just writes:
  - input data specification
  - Map/reduce tasks
  - MR library does the rest! no worries about distribution, FT, parallelism
- Grep
  - Map Input: files:
  - Map output: matching lines/filenames
  - Reduce input/output: matching lines/filenames
- URL access frequency from logs
  - Map Input: web server logs
  - Map output: key = web page, value = 1 (or more)
  - reduce input: web page, list of hits
  - Reduce output: web page, sum of hits

Execution In parallel



Map Reduce under the covers

- Simple idea: what makes it work?
  – Scalability via partitioning + locality
  – Fault tolerance via failure detection/retry
- Key ideas:
  – partitioning input
  – Grouping output locally to disk
  – Reduce pulls results
  – Single manager coordinates things

What Map/Reduce does

- Start a master – one copy of the program to direct everything else
  – QUESTION: is this a single point of failure? Does it matter?
- Master splits input data into 16-64 MB chunks
  – How does it know the input data, size?
    - files must be specified some how
- Master picks idle workers for map & reduce tasks

Map task

- Map library: Assign chunks to workers
  – QUESTION: How?
    - Anyway reasonable; want locality within chunk if possible
  – Library reads in data in some granularity, parses key/value pairs, invokes map
- User map code:
  – Execute map task, write output
- Map library:
  – buffer outputs into R (# reduce tasks) local files
  – Notify master when done, locations on disk (file names?) of intermediate files

Reduce task

- Reduce worker told which map nodes to pull from
  – Groups intermediate data by intermediate key
  – Process key + list of intermediate values
  – Write back to a single file per task (could be many intermediate keys in a reduce task)
- QUESTION: how are reduce tasks assigned to nodes?
  – Could be separate nodes
  – Could be on nodes with lots of intermediate results for the key range assigned to the reduce task

Why does this work?

- Partitioning of input allows easy scalability
- Mixing between map and reduce (O(M nodes x R node)) not too bad…

- Note: saving state to intermediate storage takes time (slow I/O) …
- QUESTION: is it important?
- NOTE: not streaming/pipelined

Fault Tolerance

- What can fail?
  - Master: retry whole operation
  - Mapper: re-execute map on original data
  - Reduce: refetch data from mapper (mapper need not re-execute)
    - Why is this possible? Mapper writes output to disk, not pushed to reducer in memory
    - Atomically commit data via rename (write temporary, rename to final version at output) to prevent duplicates in output

More optimizations

- Combiners
  - What if you are emitting "1" for lots of words in a document and reducing produces the count; creates lots of intermediate data
  - SOLUTION: **combiners** to locally combine/aggregate at mapper before reduce
    - Is a version of reduce function that writes intermediate values not final outputs
- Sequencing
  - Can connect a set of M/R tasks together for richer analysis
  - Output of one reduce phase is input to next map phase (e.g. 5-10 for web indexing)

What's wrong with MapReduce?

- Literally Map then Reduce and that's it…
  - Reducers write to replicated storage
- Complex jobs pipeline multiple stages
  - No fault tolerance between stages
    - Map assumes its data is always available: simple!
- Output of Reduce: 2 network copies, 3 disks
  - In Dryad this collapses inside a single process
  - Big jobs can be more efficient with Dryad

Complaints about Map/Reduce

- Parallel databases do it already and better
  - Map/reduce easy to represent as a query (select, apply function, group by:
    - SELECT custID, sum(amount)
      FROM Sales
      WHERE date BETWEEN
      "12/1/2009" AND "12/25/2009"
      GROUP BY custID
  - like map(cust ID) if date in range emit (cust ID, amount); reduce(cust id, list amounts) emit sum (amounts)

More specifics

- Databases store data in more efficient formats:
  - row vs column
  - indexes
  - compressed
- RESPONSE: can M/R do this?
  - Can integrate into input/output format
  - Can use M/E to preprocess data into efficient formats & compress

More complaints

- Cannot really do join:
  - select data from table 1 and matching data from table 2 (two different inputs to map task) and output matches ONLY equijoin; and must scan both inputs completely
  - Reduce needs to do cross product of inputs from two tables
    - Big blow up; cannot start until all inputs available
  - Need to scan both completely

Dryad

- Map/Reduce has a single flow of data:
  - partition data to mappers, then mix to reducers, then output
  - Can sequence multiple jobs in a row
- Dryad goal: more flexible data flow with more operators
  - Can do more traditional database queries

Motivation

- Complex queries in SQL hard to express as map/reduce:
  - The task is to identify a "gravitational lens" effect: it finds 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.
- In SQL: select a star, then select all neighbors of the star, then find ones with similar color + coordinates close enough
- General dryad goal: support execution of dataflow graphs

Advantages of DAG over MapReduce

- Big jobs more efficient with Dryad
  - MapReduce: big job runs >=1 MR stages
    - reducers of each stage write to replicated storage
    - Output of reduce: 2 network copies, 3 disks
  - Dryad: each job is represented with a DAG
    - intermediate vertices write to local file

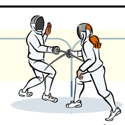
Dryad Properties

- Provides a general, clean execution layer
  - Dataflow graph as the computation model
  - Higher language layer supplies graph, vertex code, channel types, hints for data locality, …
- Automatically handles execution
  - Distributes code, routes data
  - Schedules processes on machines near data
  - Masks failures in cluster and network

But programming Dryad is not easy

22

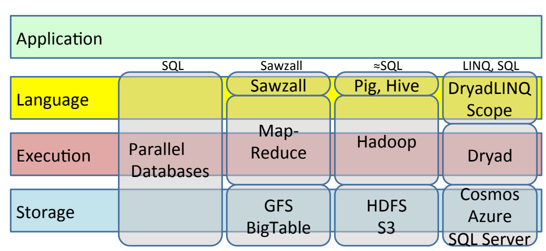
Dryad vs. Map-Reduce



Dryad	Map-Reduce
• Many similarities	
Execution layer	Exe + app. model
Job = arbitrary DAG	Map+sort+reduce
Plug-in policies	Few policies
Program=graph gen.	Program=map+reduce
Complex (↑features)	Simple
New (< 2 years)	Mature (> 4 years)
Still growing	Widely deployed
Internal	Hadoop

23

Data-Parallel Computation

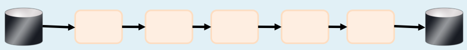


24

2-D Piping

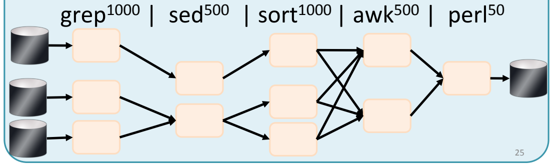
- Unix Pipes: 1-D

  grep | sed | sort | awk | perl



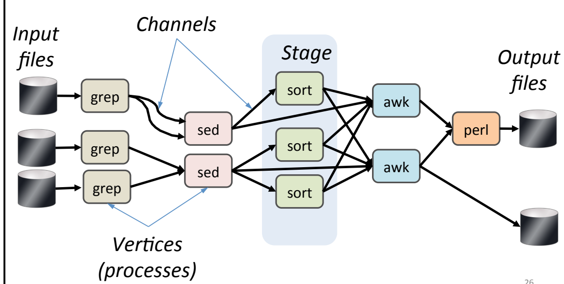
- Dryad: 2-D

  grep^{1000} | sed^{500} | sort^{1000} | awk^{500} | perl^{50}



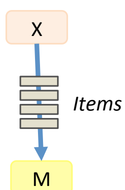
25

Dryad Job Structure



26

Channels

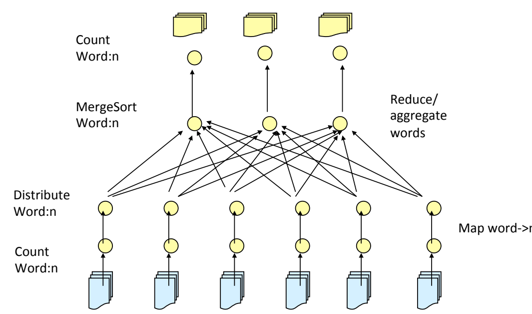


Finite streams of items

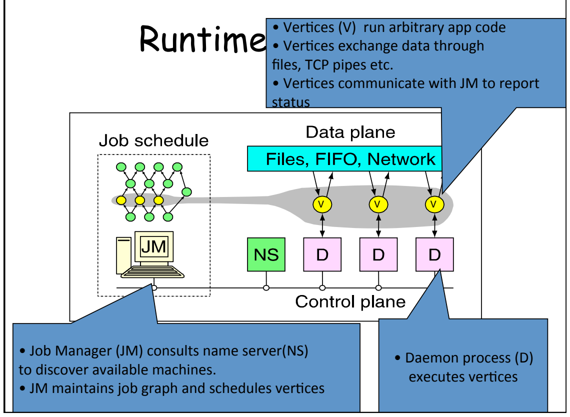
- distributed filesystem files (persistent)
- SMB/NTFS files (temporary)
- TCP pipes (inter-machine)
- memory FIFOs (intra-machine)
- **QUESTION: Why?**

27

WordCount in Dryad



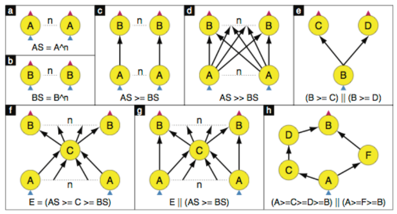
Runtime



Dryad DataFlow

- M/R does two things: partition + mix
- Dryad has more flexibility:
  - pointwise connection (send all data from node a to node b)
    - Can send from Ai to Bj, or from Ai to B (combine at one node) or from A to Bi (distribute/fan out)
  - bipartite (like reduce; send results from all As to all Bs
  - Can mix: send some data to different places
    - e.g send summary to a node that then propagates to all workers
  - Dryad has lots of standard operators: hash, sort, merge

Graph Types



How to think about Dryad

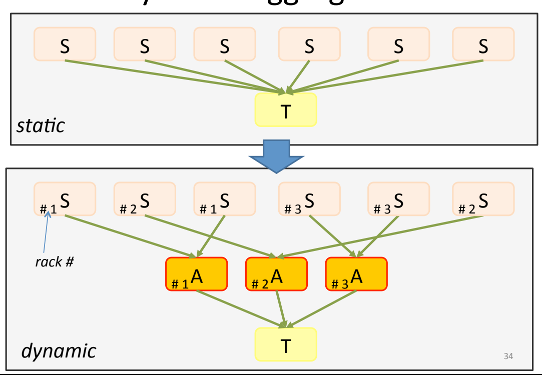
- Map-Reduce: shell scripts with pipes
- Dryad: python programs

- Decompose map-reduce into component parts, so can be re-used
  - Input parsing
  - Data distribution
  - Reduction/aggregation
  - Sorting
  - Merging
  - Communication channels
  - Counting
  - Hash table

Scheduling at JM

- General scheduling rules:
  - Vertex can run anywhere once all its inputs are ready
    - Prefer executing a vertex near its inputs
  - Fault tolerance
    - If A fails, run it again
    - If A's inputs are gone, run upstream vertices again (recursively)
    - If A is slow, run another copy elsewhere and use output from whichever finishes first

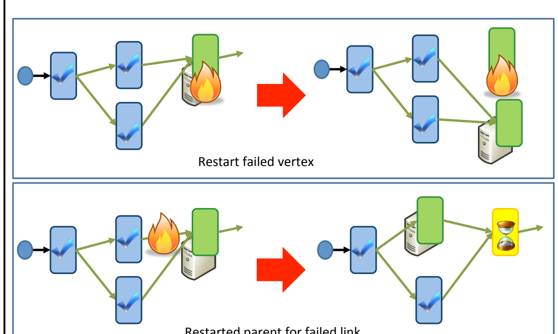
Dynamic Aggregation



Optimizing Dryad applications

- General-purpose refinement rules
- Processes formed from subgraphs
  - Re-arrange computations, change I/O type
- ***Application code not modified***
  - System at liberty to make optimization choices
- High-level front ends hide this from user
  - SQL query planner, etc.

Fault Tolerance



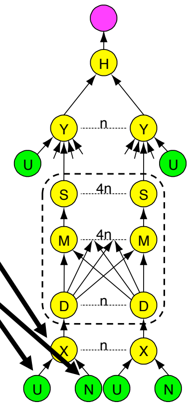
Dryad example 1: SkyServer Query

- 3-way join to find gravitational lens effect
- Table U: (objId, color) 11.8GB
- Table N: (objId, neighborId) 41.8GB
- Find neighboring stars with similar colors:
  - Join U+N to find
    T = N.neighborID where U.objID = N.objID, U.color
  - Join U+T to find
    U.objID where U.objID = T.neighborID
    and U.color ≈ T.color

SkyServer query

```
select
  u.color,n.neighborobjid
from u join n
where
  u.objid = n.objid
```

```
u: objid, color
n: objid, neighborobjid
[partition by objid]
```




```
[distinct]
[merge outputs]
```

```
(u.color,n.neighborobjid)
[re-partition by
n.neighborobjid]
[order by n.neighborobjid]
```

```
select
  u.objid
from u join <temp>
where
  u.objid = <temp>.neighborobjid
and
  |u.color - <temp>.color| < d
```