What is MR+?

Port of Hadoop’s MR classes to the general computing environment

- Allow execution of MapReduce programs on any cluster, under any resource manager, without modification
- Utilize common HPC capabilities
  - MPI-based libraries
  - Fault recovery, messaging
- Co-exist with other uses
  - No dedicated Hadoop cluster required
What MR+ is NOT

• An entire rewrite of Hadoop
  – Great effort was made to minimize changes on the Hadoop side
  – No upper-level API changes were made
    • Pig, Hive, etc. do not see anything different

• An attempt to undermine the Hadoop community
  – We want to bring Hadoop to a broader community by expanding its usability and removing barriers to adoption
  – We hope to enrich the Hadoop experience by enabling use of a broader set of tools and systems
    • Increase Hadoop’s capabilities w/o reinventing the wheel

Why did we write it?

• Scalability issues with Hadoop/YARN
  – Launch and file positioning scales linearly
  – Wireup scales quadratically
  – No inherent MPI support

• Performance concerns
  – Data transfer done via http
  – Low performance (high latency, many small transfers)

• Barriers to adoption
  – Integrated RM, dictating use of dedicated system
  – Only supports Ethernet/http
Hadoop 1.0

- JobTracker receives client request
- Assigns tasks to nodes based on node resource availability data in heartbeat

No global state info!
Hadoop 1.0

- TaskTracker receives assignment
- JobTracker transfers all reqd files
- Execution managed by TaskTracker

• Task assignment done upon heartbeat
  - JobTracker uses synchronous processing of heartbeats
    • Max transaction rate 200 beats/sec
    • No global status info must wait for beat to assign tasks to any node
    • Linear launch scaling
  
• No internode communication
  - Hub-spoke topology
  - Precludes collective communication for wireup exchange
  - Wireup scales quadratically

• Simple fault recover model
Hadoop 2.0

- Cleaner separation of roles
  - Node manager: manages nodes, not tasks
  - Create new application master role
- Event-driven async processing of heartbeats
  - Improve throughput for better support of large clusters
Hadoop 2.0 (YARN)

- RM receives client request
- Assigns a container for Application Master to a node based on resource availability data in heartbeat

Hadoop 2.0 (YARN)

- Client launches AppMstr via corresponding NM
- AppMstr contacts RM with resource requirements, including preferred locations etc.
Hadoop 2.0 (YARN)

- RM returns node/container assignments to AppMstr
- AppMstr launches procs on allocated containers via corresponding NM

• Proc is launched and reports contact info to AppMstr
• AppMstr manages job, connections
Hadoop 2.0

- Two levels of task assignment done upon heartbeat
  - Faster, but now have to do it twice
  - No global status info must wait for beat to assign AM and tasks to any node
  - Linear launch scaling

- No internode communication
  - Hub-spoke topology with AM now at the hub
  - Precludes collective communication for wireup exchange

- Simple fault recover model

- Security concerns
  - Nodemanagers are heavyweight daemons operating at privileged level

Observations

- **MR+**
  - **SLURM**
    - 16,000 processes across 1000 nodes launched in ~20 milliseconds*
    - Wired and running in ~10 seconds
  - **Cray**
    - 139,000 processes across 8500 nodes launched in ~1 second
    - Wired and running in ~60 seconds*

- **Hadoop 2.0**
  - 2 processes on separate nodes
    - Launched in ~5-10 seconds
  - 12,768 processes on 3,192 nodes
    - Launched in ~10 min
    - Wired and running in ~45 minutes*

*prepositioned files
MR+ Approach

• Remove the Hadoop resource manager system

MR+ Approach

• Utilize the system resource manager, with ORTE as the abstraction layer
• Add a JNI-based extension to the existing JobClient class to interface to the RM
Differences

• RM s maintain system state
  – Don’t rely on heartbeats to avoid scalability issues
    • Look at connection state
    • Use multi-path connection topology
  – High availability based on redundant “masters”
  – Allocation can be performed immediately, regardless of scale

• Scalable launch
  – Internode communication allows collective launch and wireup
    (logN scaling)

• Reduced security concern
  – RM daemons very lightweight
    • Consist solely of fork/exec (no user-level comm or API)
    • Minimal risk for malware penetration
  – Orteds are heavier, but operate at user level

How does it work?

• “Overlay” JobClient class
  – JNI-based integration to Open MPI’s run-time (ORTE)
  – ORTE provides virtualized shim on top of native resource manager
    • Launch, monitoring, and wireup at logN scaling
    • Inherent MPI support, but can run non-MPI apps
    • “Staged” execution to replicate MR behavior
    • Preposition files using logN-scaled system

• Extend FileSystem class
  – Remote access to intermediate files
  – Open, close, read, write access
  – Pre-wired TCP-based interconnect, other interconnects (e.g., Infiniband, UDP) automatically utilized to maximize performance
What are the biggest differences?

*It's all in the daemons…*

* • Hadoop's node-level daemons do not communicate with each other
  – Only send “heartbeats” to the YARN resource manager
  – Have no knowledge of state of rest of nodes
  – Results in bottleneck at RM, linear launch scaling, quadratic wireup of application processes…but relatively easy fault tolerance

* • ORTE's daemons wireup into a communication fabric
  – Relay messages in a logN pattern across the system
  – Retain independent snapshot of state of system
  – Results in logN launch scaling, logN wireup, coordinated action to respond to faults…but more complex fault tolerance design

What are the biggest differences?

* …and in the RM*

* • Hadoop's RM retains no global state info
  – Allocation requests are queued and wait for heartbeats from nodes that indicate appropriate resources available
  – Results in delays until heartbeats arrive, suboptimal resource allocation unless wait to hear from all nodes (complication: nodes may have failed)...but easy to recover RM on failure

* • HPC RMs maintain global state
  – Can immediately allocate, optimize assignment
  – Results in very fast allocation times (>100K/sec)...but more difficult to recover RM on failure (methods have been field proven, but are non-trivial)
Three new pieces

• Jobclient.c
  – Contains JNI integration to ORTE
  – Serves as “HNP” in the ORTE system
    • Manages launch and sequencing of MR stages
    • Replaces Hadoop execution engine

• Filesystem.c
  – Support distributed file operations (open, close, read, write) using ORTE daemons for shuffle stage

• Mapred.c
  – Send and receive mapper output partition metadata

Overview of operation: defining the job

• jc = New jobClient
  – If OMPI libs not loaded, then load them and initialize ORTE system
  – Create a new map/reduce instance

• jc.addMapper/addReducer
  – Call as many times as you like, each with its own cmd line
  – Typically called once for each split
  – Includes param indicating relative expected run time

• jc.addFile, addJar
  – Indicate files to be transferred to remote nodes for use by mappers and reducers (archives automatically expanded on remote end)
  – Separately tracked for each map/reduce pair

• jc.runJob
  – Execute this map/reduce pair
  – Execution will commence as resources become available
  – Returns upon completion
Map/Reduce staging

• Current
  – Only one map/reduce pair can be executing at a time
  – Any number of pairs can be defined in parallel
  – Any sequencing of M/R pairs is allowed
    ▪ Results-based steering

• Future
  – Map/reduce pairs can operate in parallel
    ▪ Sequenced according to resource availability
  – runJob will queue job and immediately return
    ▪ isComplete() polled to determine completion

Resource definition

• Current
  – Allocation must be defined in advance
    ▪ Obtained from external RM
    ▪ Specified in hostfile – number of slots automatically set to number of cores on each node
    ▪ Java-layer determines what, if any, location preference
      ▪ Can use HDFS to determine locations
    ▪ Provided to jobClient as non-binding “hint” for each M/R split
      ▪ Highest priority given to placing procs there, but will use other nodes if not available

• Future option
  – ORTE can obtain allocation from external RM based on file specifications
    ▪ RM will treat file locations as non-binding “hint”, callback with allocation when number of desired slots is met (working on SLURM and Moab integration now)
    ▪ If you give allocation, we will use it
Some details/constraints

• Execute in ORTE session directory
  – Unique “scratch” directory tree on each node
  – Includes temporary directory for each process
  – All files preloaded to top-level location, and then linked to the individual process’ directory

• Jars automatically added to classpath

• Paths must be set*
  – “hadoop” must be in PATH on all nodes
  – OMPI must be installed and in PATH and LD_LIBRARY_PATH on all nodes

  *Typical HPC requirement

Overview of operation: execution

• For each pair, mappers go first
  – Longest expected running mappers have higher priority
    • Executed in priority order as resources permit, so lower priority could run first if resources for higher priority not available
  – Location “hint” used to prioritize available resources
    • If desired location available, it is used
    • Otherwise, alternative locations used
  – “strict” option
    • Limits execution strictly to desired locations

• When mappers fully completed, associated reducer is executed
  – Uses same “hint” rule as mappers
Resource competition

Variety of schemes by user option

- "eldest": priority to the longest waiting process across all executing M/R pairs
- "greedy": priority to the process expected to require longest running time in the same M/R pair*
- "sequential": priority to the next defined process in the same M/R pair, rotating to next M/R pair if all done
- "eager": priority to process expected to require shortest running time across all executing M/R pairs
- Many schemes can be supported by simply adding components

*current, default

Overview of operation: data transfer

- Reducers access mapper output via extensions to FileSystem class
  - Open, close, read, write APIs
  - Daemons on remote nodes transfer the data using ORTE/OMPI transports
    - Fastest method used, point-to-point
- Also support streaming mode
  - Requires mappers and reducers both execute at same time
    - Must have adequate resources to do so
    - Stdout of mappers connected to stdin of reducers
- Future
  - Look at MPI-I/O like solution
What about MPI?

- MPI permitted when all procs can be run in parallel
  - ORTE detects if MPI attempted and errors out if it cannot be supported
  - Mapper and reducer are treated separately
- MPI support always available
  - No special request required
  - Add flag at some point to indicate “all splits must be executed in parallel”?

What about faults?

- Processes automatically restarted
  - Time from failure to relocation and restart
    - Hadoop: ~5-10 seconds
    - MR+: ~5 milliseconds
  - Relocation based on fault probabilities
    - Avoid cascading failures
- Future state recovery based on HPC methods
  - Process periodically saves “bookmark”
  - Restart provided with bookmark so it knows where to start processing
  - Prior intermediate results are preserved, appended to new results during communication
Why would someone use it?

• Flexibility
  – Let customer select their preferred environment
    • Moab/Maui, SLURM, LSF, Gridengine, simple rsh, …
  – Share resources

• Scalability
  – Launch scaling: Hadoop (~N), MR+ (~logN)
  – Wireup: Hadoop (~N^2), MR+ (~logN)

• Performance
  – Launches ~1000x faster, potentially runs ~10x faster
  – Enables interactive use-case

• MPI library access
  – ScaLAPACK, CompLearn, PetSc, …

TPCH: 50G (Hive benchmark)
TPCH: 100G (Hive benchmark)

TPCH: 256G (Hive benchmark)
**Other Benchmarks**

- Movie's Histogram (30G)
- Rating's Histogram (30G)
- wordcount (wikipedia 150G)
- inverted-index (wikipedia 150G)

**Lessons Learned**

- Running MR using ORTE is feasible, provides benefits
  - Performance, security, execute anywhere
  - Access to MPI
  - Performance benefit drops as computation time increases

- Need improvement
  - Shuffle operation
    - Pre-position data for reducers that haven’t started yet
    - Requires pre-knowledge of where reducers are going to execute
    - More efficient, parallel file read access (perhaps MPI-IO)
  - Overlap mappers and reducers (resources permitting)
    - Don’t require all mappers to complete before starting corresponding reducers
Future Directions

• Complete the port
  – Extend range of validated Hadoop tools
  – Add support for HD2.0

• Continue testing and benchmarks
  – Demonstrate fault recovery
  – Large-scale demonstration

• “Alpha” release of code
  – Gain early-adopter feedback

• Pursue improvements
  – Shuffle, simultaneous operations

THANK YOU!