Improving forwarding logic in Slurm
Motivations

• Upcoming “multi-petaflop” systems will have thousand of nodes interconnected by large scale high-speed networks
• SLURM may have to pass by the same high-speed network used by applications, hence there are two issues:
  • Internal communications mechanisms in SLURM do not take into account the underlying network topology
  • Reverse communications (from slurmd to slurmctld) can be an important bottleneck with larger number of nodes

Reversed communications scalability bottleneck

- Performance Degradation visible in the increase of turnaround time and stretching of the diagram (detailed analysis in [1]).
- Every node involved in a job sends its own completion message directly to the controller:
  - too many EPILOG_complete RPCs
  - increase of processing time on the controller
Goals of the project

• Re-factoring of the communication logic of Slurm in order to provide partially deterministic direct and reverse tree communications.

  1) Increase performances by better handling the mapping between the trees of communication used by SLURM and the existing physical network connections.

  2) Provide the ability to aggregate messages directed to the controller to limit the amount of RPCs to be handled simultaneously when possible.
Overview

- Two new features have been implemented.
- Route Plugin (already in 14.11)
- Messages Aggregation (to appear in 14.11 or 15.xx)
Route Plugin

• Provides an opportunity to choose message forwarding nodes based on patterns other than the TreeWidth Parameter.
• Can off load some communication overhead from slurmctld.
  • A side effect may be more message hops and higher latency.
• Plugin Implementations
  • RoutePlugin=route/default
  • RoutePlugin=route/topology
Route Plugin -- default

• Splits a list of nodes to send a message into sublists based on tree width.
• Sent the message to at most treewidth nodes.
• If the number of nodes is > treewidth, include in the message header a list of node that the forwarding node with send the message.
• Receiving node may also split its list into sublists.
• Message forwarding is identical to current implementation
Route Plugin – default (2)

- Message Forwarding Between Nodes
- TreeWidth=3
Route Plugin -- topology

• Split message list into sublists based on switch topology in topology.conf (requires topology/tree)
• (message sent to first node on switch, and its message list is all nodes that are reached by that switch)
• Messages forwarded to nodes that are ‘close’
• Slurmctld will usually forward to fewer nodes so will have less overhead handling comm.
Route Plugin – topology (2)

- # topology.conf
- SwitchName=root Switches=is[0-1]
- SwitchName=is0 Switches=gw[0-1]
- SwitchName=is1 Switches=gw[2-3]
- SwitchName=gw0 Nodes=trek[0-1]
- SwitchName=gw1 Nodes=trek[2-3]
- SwitchName=gw2 Nodes=trek[4-6]
- SwitchName=gw3 Nodes=trek[7-9]
• Message Forwarding Between Nodes
Message Aggregation - Overview

• Resolves problem by aggregating epilog complete messages into a smaller number of composite messages, reducing the number of incoming TCP connections to serve.

• Essentially the reverse of the message forwarding/fanout mechanism used to reduce the load on the controller for broadcast messages.

• Will be enhanced in the future to support additional message types and destination nodes (not just the controller), and to support message responses.
Message Aggregation - How It Works

• If a compute node epilog script is configured (*Epilog* parameter in `slurm.conf`), or a job is killed (due to walltime or `scancel`) an **epilog complete message** is generated on each compute node for each job using that node.

• Without message aggregation:
  • Each epilog complete message is sent directly to `slurmctld` on the management node.

• With message aggregation:
  • Epilog complete messages are routed through a series of **message collector nodes**.
  • On each collector node, epilog complete messages received during a defined **message collection window** are collected and packaged inside a new **composite message** type.
  • When the window expires, the composite message is sent to the next collector node on the route to `slurmctld`. 
Message Aggregation - How It Works

Message Collection
Node Details

Incoming EPILOG Complete Messages and/or COMPOSITE Messages

_EPILOG Complete Messages generated on this node_

_Collect_Messages()_

_Collect_Messages()_

Messages Collection

Window Expires

Build New Composite Message

Send Composite Message to next collector node or slurmctld
Message Aggregation - How It Works

- The message collection window size = maximum number of messages plus maximum elapsed time.
  - A window expires when either the maximum messages or maximum time is reached, whichever occurs first.
  - A new window is started when the first epilog complete message or composite message is received following expiration of the previous window.

- The routing used for message aggregation is provided by the Route plugin.
  - This determines the number and identity of the message collector nodes and the number of message collection “hops” between each compute node and the management node.

- When slurmd receives a composite message, it extracts each epilog complete message and processes it as if it had been sent directly from the node that generated it.
Message Aggregation Topology: Example
Message Aggregation - Limitations

• The message aggregation feature involves a trade-off between the amount of aggregation and message delay.

• A larger message collection window size and/or a larger number of message collector nodes will increase the amount of aggregation (reducing the load on the controller) but will also increase message delay at collector nodes (delaying job completion).

• The feature is recommended mainly for systems subject to the job termination bottleneck problem. Experimentation may be required to determine the optimum message collection window size and number/layout of message collector nodes.
Message Aggregation - Configuration

- Disabled by default. Enabled with new slurm.conf parameter `MsgAggregationParams`.
- `MsgAggregationParams` defines the message collection window size as a maximum number of messages and maximum time (in milliseconds). Example:
  
  \[\text{MsgAggregationParams=WindowMsgs=10,WindowTime=100}\]

- The Route plugin and associated parameters determine the number and identity of the message collector nodes.
  - Either: `RoutePlugin=route/default with TreeWidth=n`
    
    With this option, the message aggregation route is the reverse of the message forwarding route used for broadcast messages (messages sent from controller to multiple compute nodes).
  
  - or: `RoutePlugin=route/topology with TopologyPlugin=topology/tree` and a topology.conf file.
    
    With this option, the message aggregation route is defined by the node topology configuration in topology.conf.
Performance evaluation experiments followed the same guidelines as in [1].
- Emulated environment: 5040 emulated nodes upon 16 physical nodes
- Light-ESP workload: 230 jobs with adaptable sizes according to size of the cluster
- Tested 3 different scenarios with the following configurations:
  1) No route plugin, no message aggregation
  2) Route plugin/default with message aggregation 100msgs in 10msec
  3) Route plugin/default with message aggregation 500msgs in 1sec
Experimentation Results - TCP connections

- Measured number of EPILOG_complete messages during the workload and number of COMPOSITE messages:

<table>
<thead>
<tr>
<th></th>
<th>Total number of EPILOG</th>
<th>Total number of Composites</th>
<th>Total nested Composites</th>
<th>Average EPILOG in Composites</th>
<th>Median EPILOG in Composites</th>
<th>Max EPILOG in Composites</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO MSG Aggregation</td>
<td>115358</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100MS in 10 msec</td>
<td>115486</td>
<td>35986</td>
<td>12570</td>
<td>3.2</td>
<td>4</td>
<td>29</td>
</tr>
<tr>
<td>500MS in 1s</td>
<td>115443</td>
<td>5162</td>
<td>1814</td>
<td>22.4</td>
<td>16</td>
<td>52</td>
</tr>
</tbody>
</table>

- A workload of 230 jobs that lasts about 1200sec produced about 115500 EPILOG complete messages
- The message aggregation technique diminished significantly the number of TCP connections that the controller has to serve
Experimentation Results - Processing time

NO Message Aggregation

System utilization for Light ESP synthetic workload of 230 jobs and SLURM upon 5040 nodes (16 cpu/node) cluster (emulation upon 16 physical nodes)

NO messages aggregation

[Diagram showing system utilization over time with annotations for system utilization and job start time impulse]
Experimentation Results - Processing time
With Message Aggregation 10ms window

System utilization for Light ESP synthetic workload of 230 jobs
and SLURM upon 5040 nodes (16cpu/node) cluster (emulation upon 16 physical nodes)
with messages aggregation: 100 msg in 10 msec window
Experimentation Results - Processing time
With Message Aggregation 1s window

System utilization for Light-ESP synthetic workload of 230 jobs
and SLURM upon 5040 nodes (16 cpu/node) cluster ( emulation upon 16 physical nodes)
with messages aggregation: 500 msg in 1 sec window

Time (sec)
• **Absolutely** no improvement at all in overall turnaround time of the whole workload!!
  • We can even observe a slight improvement in overall turnaround time with no messages aggregation which is due to the latency of composite messages creation
• But WHY no improvement with so much less messages to deal with on the controller side?
• The reason is because the actual processing of composite messages takes place by treating each EPILOG complete message one by one so _slurm_rpc_epiolog_complete is still executed n times._
• Hence we need to create a new function for the processing of composite messages in order to treat the epilog messages as an array and gain the time of locks, node_state_save, etc.
Conclusion and Future Works

- Re-factoring of forwarding logic in SLURM to consider the network topology design
  - Improves internal communications in general due to a better mapping of SLURM communication trees upon the underlying physical network
- Implementation of messages aggregation for reverse tree communications
  - Reduces the number of incoming TCP connections to serve on the controller
  - Still need to optimize the processing of those RPC messages in groups.
  - The logic of messages aggregation should be extended to be used for all types of messages (squeue, sinfo demands, node registration messages, etc)