A (Brief) LIBI Update and Other MRNet-related Stuff

Joshua Goehner, Taylor Groves and Dorian Arnold
UNM

Dong Ahn and Greg Lee
LLNL
LIBI Approach

- LIBI: Lightweight infrastructure-bootstrapping infrastructure
  - Generic service for scalable distributed software infrastructure bootstrapping
    - Process launch
    - Scalable, low-level collectives

Diagram:
- Large Scale Distributed Software
  - Debuggers
  - System Monitors
  - Applications
  - Performance Analyzers
  - Overlay Networks

LIBI
- LaunchMON
- Job Launchers
  - SLURM
  - rsh/ssh
- Communication Services
  - OpenRTE
  - ALPS
  - MPI
  - COBO

Scalable Systems Lab
LIBI Architecture
LIBI Abstractions

- **host-distribution**: where to create processes
  - <hostname, num-processes>

- **process distribution**: how/where to create processes
  - <session-id, executable, arguments, host-distribution, environment>
**LIBI API**

`launch(process-distribution-array)`

- instantiate processes according to input distributions

`[send|recv]UsrData(session-id, msg)`

- communicate between front-end and session master

`broadcast(), scatter(), gather(), barrier()`

- communicate amongst session members
Example LIBI Front-end

```c
front-end( ){
    LIBI_fe_init();
    LIBI_fe_createSession(sess);

    proc_dist_req_t pd;
    pd.sessionHandle = sess;
    pd.proc_path = get_ExePath();
    pd.proc_argv = get_ProgArgs();
    pd.hd = get_HostDistribution();

    LIBI_fe_launch(pd);

    //test broadcast and barrier
    LIBI_fe_sendUsrData(sess1, msg, len );
    LIBI_fe_recvUsrData(sess1, msg, len);

    //test scatter and gather
    LIBI_fe_sendUsrData(sess1, msg, len);
    LIBI_fe_recvUsrData(sess1, msg, len);
}
```
Example LIBI-launched Application

session_member() {
    LIBI_init();

    // test broadcast and barrier
    LIBI_recvUsrData(msg, msg_length);
    LIBI_broadcast(msg, msg_length);
    LIBI_barrier();
    LIBI_sendUsrData(msg, msg_length)

    // test scatter and gather
    LIBI_recvUsrData(msg, msg_length);
    LIBI_scatter(msg, sizeof(rcvmsg), rcvmsg);
    LIBI_gather(sndmsg, sizeof(sndmsg), msg);
    LIBI_sendUsrData(msg, msg_length);

    LIBI_finalize();
}
LIBI Implementation Status

- LaunchMON-based runtime
  - SLURM or rsh launching
  - COBO PMGR service

- Rsh-based default
  - Pluggable launch topologies
  - Devised a provably optimal algorithm!
Optimal Launching Topology

- Assumptions
  - Homogenous computing environment
    - All nodes have the same computational power
  - Constant wait time between each local launch command
  - Constant remote launch time
    - physical network topology?
    - file system (and other resource) contention?

- Algorithm Overview
  - Pick first node as root
  - For every subsequent node, place at minimal launch point
MRNet/LIBI Integration

- MRNet uses LIBI to launch all MRNet processes
  - Parse topology file and setup/call LIBI_launch()

- Session front-end gathers/scatters startup information
  - Parent listening socket (IP/port)
LIBI v.s. MRNet default

The graph illustrates the comparison between LIBI and MRNet default in terms of MRNet bootstrap time (in seconds) against MRNet fanout. The x-axis represents the MRNet fanout, while the y-axis shows the MRNet bootstrap time (in seconds). Two lines are plotted: the red line represents the current MRNet, and the orange line represents MRNet over LIBI. The graph shows that as the MRNet fanout increases, the bootstrap time decreases for both versions, with the current MRNet achieving a lower bootstrap time compared to MRNet over LIBI.
LIBI Updates

- Back-porting MRNet Integration
  - As fastpath to MRNet on BG/Q
  - Original integration had some regression
    - Replaced “XT” and “rsh” network modes with “libi” network
    - No XT support
    - No lightweight back-end support
  - Reintegrate as additional instead of replacement mode
    - until full support for XT and other features via libi mode
LIBI Things to Talk About at CSCaDS

- Can LIBI help with OSS startup issues?

- What’s path forward for LIBI-based LaunchMON refactoring?

- (How) should LIBI interface w/ CDTI?

- Should LIBI be a CBTF component?
Other MRNet-related Happenings

- Desire an autonomous MRNet
  - Auto (re-)configuration for failure and performance
  - Need to understand how topology impacts performance
    - Performance models and validations
  - Online monitoring of relevant, dynamic parameters
  - Efficient heuristics for determining better configurations
  - Cost-benefit analyses to decide whether to change or not
Performance Model Assumptions

- Focus on reduction operations
- Single FE, IN or BE per physical node
- Filter latency is independent of in-degree
- Assume steady state workload
- Focus on both streaming and one-shot operations
  - Maximize throughput (for streaming operations)
  - Minimize latency (for one-shot interactive operations)
LogP-inspired Model Parameters

- Latency: $l$
- Gap: $g$ (packet size/bandwidth)
- Fan-out: $f$
- Message processing overhead: $o(f)$
- Filter latency: $c$

- Build a per-node model:
  - Recursion for depth
Single Wave Model

\[ T_i = l + g + o(f) + c + \max(T_c) \]

- \( T_i \) is the time to process the wave up to node \( i \)
- \( T_c \) is the time to process the wave for a node \( c \), where \( c \) is a child of \( i \).
Streaming Model

\[ T_i = l + g + \max(c_i + o_i(x)) \]

- Pipelining effectively removes recursion from single wave model
Parameter Value Generation

- Ping tests for platform dependent parameters
- Flat Topologies
  - To observe per-child costs
- Chain Topologies
  - To observe per-level costs
- Filer duration set to 0
Flat Topology (Fan-out) Tests

- Initial Measurement Observations:
  - quadratic (not linear) performance with fan-out

- Find the coefficients that correspond to $o(f)$: the message processing overhead as a function of fanout

- As fan-out increases, cost increases can be attributed to an overhead based on the breadth of the topology
  - assuming children are synchronized
Fan-out Tests
Chain Topology Tests

- As depth increases with other parameters accounted for, cost increases can be properly attributed to an overhead based on the height of the topology
  - assuming latency and bandwidth are constant at each level
Combining the results

- Combined flat and chain topology
  - Allows us to fill in message processing overhead coefficients

- Generated values for all the parameters

- Test these values with our model on some more complex topologies
Initial Results

- Generated several topologies with 128 BEs
  - Varying the internal structure

- OMNet++ simulation
  - Each simulated overlay node operates according to model

- MRNetBench
  - Allows full control over all parameters that influence performance
Initial Results

Comparison of real and simulated TBON performance

- Red bars: real
- Blue bars: simulation

Bar heights represent time to process a single wave (s) for different configurations:
- 11and12
- 8x16
- 2^7
- 128flat
- 64x2
Future Directions

- Larger scales

- Real tool/application (STAT)

- After model is validated, move on to next phases of autonomous operation
Autonomy Things to Talk about at CScADS

- Components of your computational model not captured or captured inaccurately?

- Overall relevance to your tool/analysis framework?

- Martin, why aren’t we working together on this? 😊