LIBI: The Lightweight Infrastructure-Bootstrapping Infrastructure

Joshua Goehner & Dorian Arnold
University of New Mexico
Infrastructure-bootstrapping

Given an infrastructure’s binary image(s) and a process/host distribution, start all relevant processes on their respective hosts and propagate necessary startup information.

- Before bootstrapping:
  - Program image on storage device
  - Set of (allocated) computer nodes

- After bootstrapping:
  - Application processes started on computer nodes
  - Application’s configuration complete
    - ready for primary operation
What Infrastructures Need Bootstrapping?

- They all do! Every piece of distributed software needs to be instantiated at some point
  - Applications
  - Tools
  - System services
“I gotta get my application up and running!”
(1) “First, I start all the processes on the appropriate nodes”
(2) “Next, I must disseminate some initialization information”
Bootstrapping is complete when the infrastructure is ready for steady-state usage.
Bootstrapping Operations

- Process Launching
- Information Dissemination
Bulk Launching Alternatives

- All Strategies Employ Daemons
  - Service daemons or node level agents to start application processes

- Strategies vary along two dimensions
  - Degree of daemon persistence
    - service infrastructure → more persistent
    - application specific → less persistent
  - Daemon interconnection topology
    - simple → less scalability
    - hierarchical → more scalability
Degree of (daemon) Persistence

- Persistent daemons, persistent connections
  - MPD

- Persistent daemons, transient connections
  - SLURM, ALPS

- Transient daemons, transient connections
  - Scela, MRNet default, MRNet on XT
(Daemon) Interconnection Hierarchy

- Centralized
- Rings
- Trees
Scalable Bootstrapping Alternatives

- Infrastructure-specific, scalable mechanisms
  - Still limited by sequential operations
  - Not portable to other infrastructures

- Using high-performance resource managers
  - Myriad interfaces
  - No communication facilities

- Generic bootstrapping infrastructures
  - LaunchMON: targets tools with wrapper for existing RMs
LIBI Approach

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

- **session**: set of processes (to be) deployed
  - *session master* manages other members
  - *session front-end* interacts with session master
  - LIBI currently supports only master/member communication

- **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 (cont’d)

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);

    return 0;
}
```
Example LIBI-launched Application

```c
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
  - Tested 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
  - Inspired by Park et al’s optimal multicast tree [ICPP ‘96]
  - Pick first node as root
  - For every subsequent node, place at minimal launch point
Algorithm for Optimal Launch Topology

```plaintext
def find_optimal_topology( node_list, model_params ){
    dequeue list head, set as root of tree
    compute root’s “ready time”

    while( node_list not empty ) {
        dequeue list head

        add node to tree at smallest “ready time” node
        compute node’s “ready time”
        recompute parent’s “ready time”
    }
}
```
Performance Results

- Focus on task launching
  - Lots of data available for communication topologies

- MRNet Start-up improvements
LIBI Microbenchmark Results

![Diagram showing the relationship between Launch Time (sec) and Number of Processes for LIBI and Sequential methods. The graph illustrates a linear increase in launch time as the number of processes increases.]
<table>
<thead>
<tr>
<th>Communication Time (sec)</th>
<th>Number of Processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>0</td>
</tr>
<tr>
<td>0.02</td>
<td>500</td>
</tr>
<tr>
<td>0.03</td>
<td>1000</td>
</tr>
<tr>
<td>0.04</td>
<td>1500</td>
</tr>
<tr>
<td>0.05</td>
<td>2000</td>
</tr>
</tbody>
</table>

**LIBI Microbenchmark Results**

![Graph showing communication time vs. number of processes for different operations.](image-url)
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)
MRNet Sequential-ish Bootstrapping

- Parent creates children
  - Local $\rightarrow$ `fork()/exec()`
  - Remote $\rightarrow$ `rsh`-based mechanism
- Integrated instantiation and information propagation
- MRNet’s “standard”
LIBI v.s. MRNet default

![Graph showing LIBI vs. MRNet default](image-url)
LIBI v.s. MRNet default (broken down)

- MRNet TBON Formation
- LIBI Communication
- LIBI Launch
- Preparation for LIBI
Future Research and Development

- Optimal Launch Topology
  - Performance analysis
  - Scalability bounds
  - Optimization heuristics
  - Impact of resource contention and other simplifications

- Mechanisms to alleviate file system contention
  - Like our scalable binary relocation service

- More flexible process and host distributions
  - Instantiating different images in same session
  - Integrating allocating and launching

- Integrated scalable communication infrastructure

- Refactoring LaunchMON