An Extensible, Portable, Scalable Cluster Management Software Architecture

IEEE International Conference on Cluster Computing

James H. Laros III

Sandia National Labs
jhlaros@sandia.gov
A Software Architecture

• Paper describes a software architecture
  – Not an Implementation
• Architecture has been implemented!
  – Cluster Infrastructure Toolkit (CIT)
• Numerous cluster installs at Sandia
  – Largest 1861 nodes
• Recently installed at sites outside Sandia
Requirements

• What made our requirements different?
  – 1000’s instead of 10’s or 100’s of nodes
  – Diskless
    • Due to classified switching requirement
      – Our largest cluster “multi-headed”
    • Practical reasons also…
      – Disks expensive in large numbers
      – Prone to failure
  – Extensibility
    • Legacy, current and future hardware and topologies
      – We don’t want to write new s/w for every cluster
  – “Production” level
    • People managing cluster don’t have to be cluster-savvy
Introduction

• Commodity Clusters = commodity devices
  – Nodes (of course)

• Also:
  – Terminal Servers
  – Power Controllers
  – Network Devices
  – Who knows what’s next?

• Topologies
  – Diagnostic
  – User networks
Class Hierarchy

- Hierarchical, object-oriented class structure
- Describes:
  - Device capabilities
  - Device characteristics (catch-all)
- General at top (Device class)
- Specific at bottom
- Leverages commonality in devices
  - inheritance
- Allows for device specific characteristics and capabilities
  - Override inherited attribute methods
  - New attribute methods unique to device
- Describes legacy and current device capabilities (now)
- Extensible for future devices!
  - Just add appropriate class(s)
What's in a class?

- **Device class**
  - Very generic, things all devices potentially share
    - Interface(s)
    - Serial number
    - Things we want defined in sub-classes

- **Device sub-class**
  - 1st separation into categories.
    - Most or all devices fall into one of these
      - Node, Power, TermSrver, Network, (Equipment)

- **Further sub-classing**
  - Less regular
  - Governed by type and specific device.
Persistent Object Store

- Generically “database”
- The objects stored are instantiated from the Class Hierarchy
  - Device::Node::Alpha::DS10
- The objects also define a linkage which describes the Topology
  - Attributes may be populated with names of other objects
  - Recursion establishes path to device
- A software representation of the physical cluster
  - It’s device makeup
  - How it is wired
  - Everything considered important about the cluster
Persistent Object Store (cont)

Type: Device::Power::RPC3
Name: Power-1
Console:Power-1, 23

Type: Device::Node::Alpha::DS10
Name: Node-1
Console:ETS-TS-1, 5
Power: RPC3-1, 2

Type: Device::TermSrvr::ETS
Name: ETS-TS-1
Console:ETS-TS-1, 23
Power: RPC3-1, 3
Layered Utilities (Cluster Management Tools)

- Describes wide range of utilities that exploit the underlying architecture
- Tools don’t change when used on a different cluster!

- Database Utilities
  - Get/set/etc.
- Foundational capabilities
  - Power
  - Boot
  - Status
- High level tools
  - Tools for the sys admin
  - Provides user with same interface, no matter what cluster looks like
Layered Utilities

Layered Utilities

Layered Utilities

Class Hierarchy

Database Interface Layer

Database (Persistent Object Store)
Scalability

• Collections
  – Enable grouping for organization and/or scalability
  – Contain devices and/or collections
  – Tools can act on a collection if appropriate
    • Powerful parallelism tool

• Dynamic groups
  – Groups can be formed “on the fly”
  – Based on attributes like “leader”
    • Nodes can be organized by a common leader designation
    • Another target for parallelism!

• Hardware hierarchy
  – Important in large clusters
  – Offload of work to lower layer in hierarchy
  – “database” understands this hierarchy
Conclusion

• Targeted to support Cplant™
  – Broadened scope from the beginning to support wide variety of clusters
  – Make as few assumptions as possible
  – Keep things as generic as possible
  – Isolate anything that can be considered site-specific
    • Easy to find and modify if necessary
• Open source
  – released under LGPL
• We use open source wherever we can
  – Written in Perl
    • Use many Perl modules (CPAN)
  – Get rid of “not created here” attitude
    • Satisfies requirements + works + free = use it
Future Work

• Continue with aspects of implementation
  – Expand device support
    • Class hierarchy covers more variety of devices over time
  – Addition of new tools
    • Both core and upper level
  – Documentation!!!!!!!!!!!!

• New methods to enhance scalability
  – Diskless without drawbacks
Contacts

- jhlaros@sandia.gov