A Multicast/Reduction Network
for Scalable Control and Data Collection

Dorian C. Arnold
(with Phil Roth and Bart Miller)
University of Wisconsin-Madison

Paradyn/Condor Week
March 4-6, 2002
Madison, WI
Introduction

- Scaling the Paradyn toolkit to a thousand nodes
- Paradyn has a Master/Slave architecture with a single level hierarchy
- Paradyn Front-End acts as manager for the remote Back-End daemons
Point-to-point communications schemes

- Poor Scalability
  - Front-End bottleneck
  - # of connections
  - Huge fan-out
  - “context” switching (threads/polling)
  - Large data volumes
Scalability Improvement Opportunities

Paradyn:

- **Downstream request messages** from Front-End to groups of Back-Ends.
  - Multicast instrumentation requests
  - Multicast performance data requests

- **Upstream data flows** from groups of Back-Ends to Front-End.
  - Collecting performance data
    - Aggregation of performance metrics
    - Performance consultant searches (whole-program)
Generally Applicable Solutions

- Software with one-to-many/many-to-one communication patterns
  - Distributed Computing
  - Master/Slave applications
  - Cluster administration tools
  - RPC systems (with integrated component management)
Goals

- Improve scalability of centralized, master/slave software architectures
  - Dissemination of control/request messages
  - Reduction of upstream data volume
Outline

- Multicast/Reduction Network
- Interface/Protocols
- Status & Future Directions
Multicast/Reduction Network

- Two-Fold Solution
  - Reduce fan-out (downstream -- from Front-End to Back-End)
Multicast/Reduction Network

- Two-Fold Solution
  - Reduce fan-out
    (downstream -- from Front-End to Back-End)
  - Reduce data volume
    (upstream -- from Back-End to Front-End)
Point-to-Point Network
Reducing Fan-out

Front-End

Internal Communication Infrastructure

BE BE BE BE BE BE BE BE BE BE BE BE BE BE BE BE

© 2002 Dorian C. Arnold
Multicast/Reduction Network
Reducing Fan-out

Front-End

Internal Communication Infrastructure

Data Filter
Infrastructure Details

- Network Topology
  - Configuration file
    - Currently manually generated
  - Research Questions:
    - Node partitioning between communication and application layer
    - Constructing logical topology
      - Physical topology, path characteristics, ...
    - Adaptation to node failure or load-imbalance
Infrastructure (cont’d)

Network Instantiation

- Staggered and Asynchronous:
  1. Internal nodes instantiate children
  2. Leaf nodes assigned internal ids
  3. Leaf nodes propagate “success” upstream
  4. Internal nodes record leaves reachable via downward links
  5. Internal nodes propagate reachability information toward root
Instantiating the Network
Instantiating the Network
Instantiating the Network
Instantiating the Network
Instantiating the Network
Instantiating the Network

Front-End

N={1,2}

N={3,4}

N={5,6}

N={7,8}

1 2 3 4 5 6 7 8
Instantiating the Network

Front-End

N={1,2,3,4,5,6,7,8}

N={1,2,3,4}

N={3,4}

N={5,6}

N={7,8}

N={1,2}

N={5,6,7,8}

N={1,2,3,4,5,6,7,8}
Instantiating the Network

N={1,2,3,4,5,6,7,8}

N={1,2,3,4}

N={5,6,7,8}

N={1,2}

N={3,4}

N={5,6}

N={7,8}

1  2  3  4  5  6  7  8

BE  BE  BE  BE  BE  BE  BE  BE
Protocols (cont’d)

☐ Interface

- **Communicators** name a Back-End node
- **Aggregate Communicators** name groups of Back-End nodes
- **Streams** name a flow of data over a communicator
  - Can have multiple streams to a given communicator
  - Aggregate communicators name a filter for data aggregation on upstream flows
- Users send and receive data across streams that connect the Front-End to one or more Back-Ends
Communication Nodes

- Batching/Unbatching data
  - Merge data elements into “batches” to reduce number of interactions

- Synchronization
  - Amongst streams of data from multiple incoming links

- Decoding/Encoding
  - Deciphering stream data to obtain typed objects
Synchronization

Data Aggregation Filter

Decoding

Unbatching

Encoding

Rebatching
Filters

- Multicast has been studied before!
- Filter design distinguishes this from previous work
  - Address issues of HPC applications with large data volumes
- Filter definition:
  - Input: one or more objects of a specified type.
  - Output: one object of same type as input
Filters (cont’d)

- Data Reduction
  • Aggregate multiple input data objects into a single object to propagate upward
- Example Filters (assuming numerical data stream):
  • Average
  • Sum
  • Min
  • Max
Filters (cont’d)

- Less Obvious Examples:
  - Data concatenation
  - Scalable data visualization

- Research Questions:
  - Synchronous vs. Asynchronous
  - Composite filters
  - Dynamic filter-set expansion
Current Status

- Implementing a first prototype
  - Experimenting with different partitioning strategies, topologies, etc.
  - Exploring designs that support user-extensible filtering capabilities
What’s Next

❑ Integrate with Paradyn 😊
❑ Network process organization
  • Partitioning nodes
❑ Network graph architecture
  • Balanced vs. Skewed Trees
  • Process Chains or Rings?
❑ Tool Daemon Protocol
  • Attaching to pre-running applications
❑ User-expandable filter sets
Other Opportunities

- Automated tree topology construction
  - based on physical network topology and possibly hints/directives from the user.

- Dynamic tree topologies
  - Reconfiguration in presence of failures, poor performance or load imbalances

- Multiple “Source” Trees
The End!