Overview

• Extremely large scale systems are here
• Effective, scalable programming is hard
• Start with a simple but powerful foundation: the Tree-based Overlay Networks (TBŌNs)
Overview

TBÔNs provide:

- An immediate path to scalable tools and infrastructure. Examples:
  - Paradyn Performance Tools
  - Vision algorithms
  - Stack trace analysis (new)

- A Research platform for new technologies:
  - New concepts in fault tolerance (no logs, no hot-backups).
  - As an framework for parallel applications
  - As a powerful alternative to the Map-Reduce idiom
  - As a generalized, scalable communication infrastructure
### HPC Trends from TOP500

<table>
<thead>
<tr>
<th>Date</th>
<th>Processor Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jun-99</td>
<td>13</td>
</tr>
<tr>
<td>Nov-99</td>
<td>16</td>
</tr>
<tr>
<td>Jun-00</td>
<td>18</td>
</tr>
<tr>
<td>Nov-00</td>
<td>19</td>
</tr>
<tr>
<td>Jun-01</td>
<td>0</td>
</tr>
<tr>
<td>Nov-01</td>
<td>28</td>
</tr>
<tr>
<td>Jun-02</td>
<td>54</td>
</tr>
<tr>
<td>Nov-02</td>
<td>58</td>
</tr>
<tr>
<td>Jun-03</td>
<td>58</td>
</tr>
<tr>
<td>Nov-03</td>
<td>58</td>
</tr>
<tr>
<td>Jun-04</td>
<td>75</td>
</tr>
<tr>
<td>Nov-04</td>
<td>No Data Available</td>
</tr>
<tr>
<td>Jun-05</td>
<td>92</td>
</tr>
<tr>
<td>Nov-05</td>
<td>129</td>
</tr>
<tr>
<td>Jun-06</td>
<td>140</td>
</tr>
<tr>
<td>Nov-06</td>
<td>228</td>
</tr>
<tr>
<td>Jun-07</td>
<td>302</td>
</tr>
<tr>
<td>Nov-07</td>
<td>340</td>
</tr>
</tbody>
</table>

### 06/2007 Processor Count Distribution

- 32: 3 systems
- 64: 2 systems
- 128: 3 systems
- 256: 0 systems
- 512: 152 systems
- 1024: 207 systems
- 2048: 76 systems
- 4096: 29 systems
- 8192: 28 systems
- No Data Available

- Systems Larger than 1024 Processors

© Barton P. Miller 2008

MRNet Overview
HPC Trends from

Average Processor Counts

<table>
<thead>
<tr>
<th>Date</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jun-07</td>
<td>2442</td>
</tr>
<tr>
<td>Nov-06</td>
<td>2042</td>
</tr>
<tr>
<td>Jun-06</td>
<td>1747</td>
</tr>
<tr>
<td>Nov-05</td>
<td>1463</td>
</tr>
<tr>
<td>Jun-05</td>
<td>1161</td>
</tr>
<tr>
<td>Nov-04</td>
<td>817</td>
</tr>
<tr>
<td>Jun-04</td>
<td>638</td>
</tr>
<tr>
<td>Nov-03</td>
<td>536</td>
</tr>
<tr>
<td>Jun-03</td>
<td>494</td>
</tr>
<tr>
<td>Nov-02</td>
<td>446</td>
</tr>
<tr>
<td>Jun-02</td>
<td>No Data Available</td>
</tr>
<tr>
<td>Nov-01</td>
<td>323</td>
</tr>
<tr>
<td>Jun-01</td>
<td>0</td>
</tr>
<tr>
<td>Nov-00</td>
<td>281</td>
</tr>
<tr>
<td>Jun-00</td>
<td>232</td>
</tr>
<tr>
<td>Nov-99</td>
<td>226</td>
</tr>
<tr>
<td>Jun-99</td>
<td>186</td>
</tr>
</tbody>
</table>
“I think that I shall never see
An algorithm lovely as a tree.”

Trees by Joyce Kilmer (1919)

If you can formulate the problem so that it is hierarchically decomposed, you can probably make it run fast.
**Distributed Control and Monitoring**

- **Hierarchical Topologies**
  - Application Control
  - Data collection
  - Data centralization/analysis

- **As scale increases, front-end becomes bottleneck**
TBÖNs for Scalable Systems

TBÖNs for scalability
- Scalable multicast
- Scalable gather
- Scalable data aggregation
TBŌN Model

Application Front-end

Tree of Communication Processes

Application Back-ends
MRNet: An Easy-to-Use TBŌN

Application-level packet

Packet filter

Filter state
**TBÖNs at Work**

- **Multicast**
  - **ALMI** [Pendarakis, Shi, Verma and Waldvogel ’01]
  - **End System Multicast** [Chu, Rao, Seshan and Zhang ’02]
  - **Overcast** [Jannotti, Gifford, Johnson, Kaashoek and O'Toole ’00]
  - **RMX** [Chawathe, McCanne and Brewer ’00]

- **Multicast/gather (reduction)**
  - **Bistro** (no reduction) [Bhattacharjee et al ’00]
  - **Gathercast** [Badrinath and Sudame ’00]
  - **Lilith** [Evensky, Gentile, Camp, and Armstrong ’97]
  - **MRNet** [Roth, Arnold and Miller ’03]
  - **Ygdrasil** [Balle, Brett, Chen, LaFrance-Linden ’02]

- **Distributed monitoring/sensing**
  - **Ganglia** [Sacerdoti, Katz, Massie, Culler ’03]
  - **Supermon** (reduction) [Sottile and Minnich ’02]
  - **TAG** (reduction) [Madden, Franklin, Hellerstein and Hong ’02]
Example TBÖN Reductions

• Simple
  - Min, max, sum, count, average
  - Concatenate

• Complex
  - Clock synchronization [Roth, Arnold, Miller ’03]
  - Time-aligned aggregation [Roth, Arnold, Miller ’03]
  - Graph merging [Roth, Miller ’05]
  - Equivalence relations [Roth, Arnold, Miller ’03]
  - Mean-shift image segmentation [Arnold, Pack, Miller ’06]
  - Stack Trace Analysis Tool [Wisconsin, LLNL]
Using MRNet for Tool Scalability

MRNet integrated into Paradyn
- Efficient tool startup
- Performance data analysis
- Scalable visualization
- Distributed Performance Consultant

Equivalence computations
- Graph merging
- Trace analysis
- Data clustering (image analysis)
- Scalable stack trace analysis
Problem of Tool Start-Up Latency

Tools often transfer a lot of data at start-up
- Debugger needs function names and addresses to set breakpoints by name
- Paradyn needs information about modules, functions, processes, threads, synchronization objects, call graph

Front-end:
- Just cannot keep up with data and control.

This is an example of an important scenario
Scalable Tool Start-up (Paradyn)

• Reduce redundant data transfer
  - Daemons deliver summary to front end using MRNet to find equivalence classes.
    • Functions, modules, control-flow graph, call graph, etc.
  - Front end asks equivalence class representatives for complete info.
  - Representative daemons send full info to front end.

• Reduce overhead of non-redundant data transfer
  - Machine resources, daemon info, process info
  - In-network concatenation of messages
    • Fewer send/recv operations
    • Front-end sees single message instead of many

• A log-time calculation: clock skew detection
Clock Skew Detection Algorithm

- **Phase 1:**
  - Repeated broadcast/reduce pairs to compute each process’ clock skew with directly connected children

- **Phase 2:**
  - Upward sweep to compute cumulative clock skew to all reachable daemons
Paradyn Start-up Latency Results
Paradyn with SMG2000 on ASCI Blue Pacific
Effect of Sub-Graph Folding Algorithm on Search History Graph Complexity

Number of Nodes vs. Processes

Unfolded SHG
Folded SHG
Background: Performance Consultant

- Paradyn’s automated performance diagnosis component: tells *why* there is a problem and points *where* to tune
- **Automated search using dynamic instrumentation**
  - Find performance problems with minimal user intervention
  - Insert and remove instrumentation code from processes as they run
    ⇒ Useful diagnosis results from a single run, with controlled overhead
- Tool daemons monitor and control application processes, tool front-end provides user interface
Background: Performance Consultant

• Search approach
  - Start with general, global experiments about application performance (e.g., CPU utilization is too high across all processes)
  - Collect performance data to test active experiments
  - Make decisions about experiments based on performance data
  - Refine search: if an experiment’s performance data is above user-configurable threshold, create new, more specific experiments and repeat

• Performance data streams from tool daemons for analysis (i.e., refinement decisions)
Three Approaches

• **Centralized Approach (CA):**
  - All performance data sent to the front end and all control from the front end.

• **Partially Distributed Approach (PDA):**
  - Global experiments (across all processes) monitored and controlled from the front-end using MRNet.
  - Local Performance Consultants on each node to look for local bottlenecks.

• **Truly Distributed Approach (TDA):**
  - Only local PC’s on each node to monitor and search.
  - Global results from merging local data, using MRNet.
Evaluation: Experimental Environment

• LLNL MCR cluster
  - 1152 nodes (1048 compute nodes)
  - Two 2.4 GHz Intel Xeons per node
  - 4 GB memory per node
  - Quadrics Elan3 interconnect (fat tree)
  - Lustre parallel file system

• su3_rmd
  - Quantum chromodynamics pure lattice gauge theory code from MILC collaboration
  - C, MPI
  - Weak scaling scalability study
Evaluation: DPC Front-End CPU Load

Front-End CPU Utilization

- CA
- PDA
- TDA

CPU Utilization (percent)

Processes

© Barton P. Miller 2008
Evaluation: DPC Daemon CPU Load

![Graph showing CPU utilization for different processes]

CPU Utilization (percent) vs Processes
Evaluation: DPC MRNet CPU Load

MRNet Internal Process CPU Utilization

CPU Utilization (percent) vs Processes

- CA
- PDA
- TDA
Evaluation: SGFA

Effect of Sub-Graph Folding Algorithm on Search History Graph Complexity

Unfolded SHG
Folded SHG

Number of Nodes

Processes

© Barton P. Miller 2008
TBÔNs for Scalable Aps: Mean-Shift Algorithm

- Cluster points in feature spaces
- Useful for image segmentation
- Prohibitively expensive as feature space complexity increases
TBŌNs for Scalable Aps: Mean-Shift Algorithm

~6x speedup with only 6% more nodes
Recent Project: Peta-scalable Tools

With: Dong Ahn, Greg Lee, Martin Schulz, Bronis de Supinski @ LLNL

Stack Trace Analysis Tool (STAT)
- Data representation
- Data analyses
- Visualization of results

Build a simple, useful tool that works at scale.
### TotalView on BG/L - 4096 Processes

<table>
<thead>
<tr>
<th>Operation</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single step</td>
<td>~15-20 secs.</td>
</tr>
<tr>
<td>Breakpoint Insertion</td>
<td>~30 secs.</td>
</tr>
<tr>
<td>Stack trace sampling</td>
<td>~120 secs.</td>
</tr>
</tbody>
</table>

Typical debug session includes many interactions

4096 is only 3% of BG/L!
Debugger Scalability Challenges

• Large volumes of debug data
• Many threads of control at front-end
• Vendor licensing limitations

• Approach: scalable, lightweight debugger
  - Reduce exploration space to small subset
  - Full-featured debugger for deeper digging
STAT Approach

• Sample application stack traces

• **Merge/analyze traces:**
  - Discover equivalent process behavior
  - Group similar processes
  - Facilitate scalable analysis/data presentation

• Leverage TBÖN model (MRNet)
Singleton Stack Trace

Appl.

```
_start
__libc_start_main
main
doSuffThenSleep
__sleep
__nanosleep_nocancel
__dl_sysino_int80
```
Merging Stack Traces

- Multiple traces over space or time

- Create call graph prefix tree
  - Compressed representation
  - Scalable visualization
  - Scalable analysis
Merging Stack Traces

 DIAGRAM: Diagram illustrating the merging of stack traces. The diagram shows three paths starting from `_start` through various functions such as `main`, `func1`, `func2`, `func3`, and `func4`.
2D-Trace/Space Analysis

Appl

... (Diagram of call graph and process hierarchy)

© Barton P. Miller 2008
2D-Trace/Space Analysis

STAT

TotalView
2D-Trace/Time Analysis
3D-Trace/Space/Time Analysis

- Multiple samples, multiple processes
  - Track global program behavior over time
  - Folds all processes together

- Challenges:
  - Scalable data representations
  - Scalable analyses
  - Scalable and useful visualizations/results
3D-Trace/Space/Time Analysis
Motivating Case Study: Pthread
Deadlock Exposed by CCSM

CCSM: Community Climate System Model
- **Multiple Program Multiple Data (MPMD)** model
- Comprises atmosphere, ocean, sea ice and land surface models
- Used to make climate predictions
- **MPI-based** application
Motivating Case Study: Pthread Deadlock Exposed in the CCSM

- Intermittently hangs:
  - Non-deterministic
  - Only at large scale
  - Appears at seemingly random code locations
  - Hard to reproduce
    - 2 hangs over 10 days (~50 runs)
- Stack traces can provide useful insight
- Many bugs are temporal in nature
  - Error not because behavior occurs, but because behavior persists!
- Need tools that run effectively at scale
STAT Performance on an IA64 Cluster

- 1024x4 Cluster
  1.4 GHz Itanium2
  Quadrics QsNetII

- 3844 processors, 0.741 seconds

Number of Application Tasks vs. Latency (secs)
STAT Performance on BlueGene/L

Number of Application Tasks vs. Latency (secs)

- 1-deep (VN Mode)
- 2-deep (VN Mode)
- 3-deep (VN Mode)

131,072 processes
A Platform for Research: Fault Tolerance

Two key observations:
• Leverage TBÕN properties
  - Inherent information redundancies
• Weak data consistency model: convergent recovery
  - Final output stream converges to non-failure case
  - Intermediate output packets may differ
  - Preserves all output information

Results in:
• No overhead during normal operation
• Rapid recovery
  - Limited process participation
• General recovery model
  - Applies to broad classes of computations
Current Reliability Approaches

• Fail-over (hot backup)
  - Replace failed primary with backup replica
  - Extremely high overhead: 100% minimum!

• Rollback recovery
  - Rollback to checkpoint after failure
  - May require dedicated resources and lead to overloaded network/storage resources
TBON Theory

TBÖN Output Theorem
Output depends only on channel states and root filter state

All-encompassing Leaf State Theorem
State at leaves subsume channel state (all state throughout TBÖN)

Result: only need leaf state to recover from root/internal failures

Filter requirements:
• Associative
• Communicative
• Idempotent
End Where We Started

TBōNs provide:

- An immediate path to scalable tools and infrastructure. Examples:
  - Paradyn Performance Tools
  - Vision algorithms
  - Stack trace analysis (new)

- A Research platform for new technologies:
  - New concepts in fault tolerance (no logs, no hot-backups).
  - As an framework for parallel applications
  - As a powerful alternative to the Map-Reduce idiom
  - As a generalized, scalable communication infrastructure
DILBERT, DO YOU HAVE THE BENCHMARK RESULTS?

DO YOU WANT THE TEN-MINUTE EXPLANATION OF WHY THE DATA ARE USELESS, OR A SIMPLE “HERE YOU GO”?

I’M IN SALES. HERE YOU GO.
MRNet References

• Roth and Miller, “The Distributed Performance Consultant and the Sub-Graph Folding Algorithm: On-line Automated Performance Diagnosis on Thousands of Processes”, PPoPP, March 2006.

www.cs.wisc.edu/paradyn
Extra Slides
TBÔNs in the Wild

Universitat Politècnica de Catalunya (Jesus Labarta):

Use MRNet to adaptively select trace granularity. Cluster analysis of traces to select representatives.

University of Oregon (Al Malony):

TauOverMRNet -- Collect and analyze TAU trace data using MRNet framework. Filters include random sampling, statistical analysis (mean, var., std. dev., etc.). Filter to throttle data rates based on feedback from nodes and trace data merging filter.
TBŌNs in the Wild

Krell Institute (formerly SGI):

Open|Speedshop: An open source performance tool suite.
Used to use IBM’s DPCL for distributed monitoring and control, but switching MRNet for scalability.

RENCI (Dan Reed, Todd Gamblin, Frank Mueller):

MPI tracing facility that includes local process-level performance statistics.
Use MRNet to control the granularity of the collection of performance data.
TBŌNs in the Wild

Paradyn Project (Mike Brim):

TBON-FS: Scalable file I/O for process control and monitoring.

Introduces the notion of a group file to operate on many instances of /proc.

Initial projects:
- Highly scalable Ganglia implementation (also simplifies the architecture).
- Group file shell.
- Highly scalable debugger - in collaboration with Totalview Tech.
Efficiency:
- Zero-copy paths
- Scatter-gather
- Binary data representation.
STAT Filter

sta_filter(vector<Packet> pkts_in,
    vector<Packet> pkts_out)
{
    for( i=0; i<pkts_in.size; i++){
        trace = deserialize( pkts_in[i] );
        ret_trace = merge( ret_trace,
            trace );
    }

    Packet p = serialize( ret_trace );

    pkts_out.pushback(p);
}
MRNet Front-end Interface

```c
front_end_main(){
    Network * net = new Network (topology);

    Communicator * comm = net->get_BroadcastCommunicator();

    Stream * stream =
        new Stream( comm, IMAX_FILT, WAITFORALL);

    stream->send("%s", "go");

    stream->recv("%d", &result);
}
```
MRNet Back-end Interface

```c
back_end_main(){
    Stream * stream;
    char *s;

    Network * net = new Network();

    net->recv("%s", &s, &stream);

    if(s == "go"){
        stream->send("%d", rand_int);
    }
}
```
MRNet Filter Interface

```cpp
imax_filter(vector<Packet> packets_in,
             vector<Packet> packets_out)
{
    for( i=0; i<packets_in.size; i++){
        result = max( result,
                       packets_in[i].get_int() );
    }

    Packet p("%d", result);

    packets_out.pushback(p);
}
```
Evaluation: Results Overview

• PDA and TDA: bottleneck searches with up to 1024 processes (limited by LLNL batch allocation size policy, not by our software or approach)

• CA: scalability limit at less than 64 processes

• Crucial: similar qualitative results using each approach
Performance Consultant: Example

- Part of a search history graph
SGFA: Example

CPUbound

c33.cs.wisc.edu
su3_rmd{1272}
main
setup
update
update_h
mult_sa_nn

su3_rmd{1273}
main
setup
update
update_h
mult_sa_nn

su3_rmd{7624}
main
setup
update
update_h
mult_sa_nn

c34.cs.wisc.edu
su3_rmd{7625}
main
Setup
update
update_h
mult_sa_nn

main

su3_rmd{7624}
main
setup
update
update_h
mult_sa_nn

su3_rmd{7625}
main
Setup
update
update_h
mult_sa_nn

© Barton P. Miller 2008
MRNet Overview
SGFA: Example

CPUbound

- c33.cs.wisc.edu
  - su3_rmd{1272}
    - main
      - setup
      - update
      - update_h
      - mult_sa_nn
  - su3_rmd{1273}
    - main
      - setup
      - update
      - update_h
      - mult_sa_nn
- c34.cs.wisc.edu
  - su3_rmd{"}
    - main
      - setup
      - update
      - update_h
      - mult_sa_nn
- main
  - Setup
  - update
  - update_h
  - mult_sa_nn

© Barton P. Miller 2008
SGFA: Example

- CPUbound
- c*.cs.wisc.edu
- su3_rmd{*}
- main
- setup
- update
- update_h
- mult_sa_nn
- mult_sa_nn
- mult_s
Evaluation: SGFA

Un-folded search history graph has over 30,000 nodes
Evaluation: SGFA
**TBÖNs for Scalable Applications**

- Many algorithms ⇒ equivalence computation
  - (Non-)equivalence to summarize/analyze input

<table>
<thead>
<tr>
<th>Application</th>
<th>Input</th>
<th>Filter</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trace Analysis</td>
<td>Trace file</td>
<td>Trace equivalence / Anomaly detector</td>
<td>Compressed traces, anomalous traces</td>
</tr>
<tr>
<td>Graph Merging</td>
<td>Sub-graphs</td>
<td>Sub-graph equivalence</td>
<td>Merged graphs</td>
</tr>
<tr>
<td>Data Clustering</td>
<td>Data Files</td>
<td>Object classifiers</td>
<td>Partitioned data</td>
</tr>
</tbody>
</table>
STAT Motivation

• Discover application behavior
  - Progressing or deadlock?
  - Infinite loop?
  - Starvation?
  - Load balanced?

• Tool goals:
  - Pin-point symptoms as much as possible
  - Direct user’s to root cause
**BG/L Scaling Test Setup**

- Run on a single rack BG/L system
  - 1024 compute nodes allows emulation of up to 1024 I/O node daemons (full BG/L)
- Emulate both coprocessor mode and virtual node mode of full BG/L (64 and 128 tasks per daemon, respectively)
- Ran 2-deep and 3-deep topologies
STATBench Revealed a STAT Scalability Issue

• Edge labels represented by task lists
  - Original implementation as strings
  - \([1,3,4,5,6,9,10,11,15]\) -> “\([1,3-6,9-11,15]\)”
  - Up to 75KB at 32,768 tasks

• Re-implemented edge label as a bit vector
  - 1 bit per task
    • Set to 1 if the task is in the list
    • Set to 0 otherwise
Large Scale System Reliability

- BlueGene/L
- Purple

MTTF (days)

Node Count

- 0.999 (~42 days)
- 0.9999 (~417 days)
- 0.99999 (~4,167 days)
- 0.999999 (~41,667 days)
- 0.9999999 (~416,667 days)
LLNL Parallel Debug Sessions
(03/01/2006 - 05/11/2006)

18,391 sessions!

Users need better tools
STAT prototype using MRNet

STAT Filter

FE

CP

CP

CP

BE

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI

MPI
STAT BG/L Experimental Setup

- STAT Front-end and communication processes on login nodes
- STAT Back-end on I/O nodes
- MPI task on compute nodes
BG/L Configuration

- Each node has 2 CPUs
- 14 login nodes
- 1,664 I/O nodes
  - Each I/O node connects to 64 compute nodes
- 106,496 compute nodes
  - Co-processor (CO) mode: 1 CPU for application process, 1 CPU for communication
  - Virtual node (VN) mode: 2 CPUs for application process
Future of STAT

• Future research detailed in paper

• Plans to make generally available
  - http://www.paradyn.org/STAT

• TBŌN computing papers & open-source prototype, MRNet, available at:
  - http://www.paradyn.org/mrnet
(More) HPC Trends from

- 60% are larger than $10^3$ processors
- 10 systems larger than $10^4$ processors

<table>
<thead>
<tr>
<th>System</th>
<th>Location</th>
<th>Size</th>
<th>Time Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>RoadRunner</td>
<td>LANL</td>
<td>$\sim 3.2 \times 10^4$</td>
<td>2008</td>
</tr>
<tr>
<td>Jaguar</td>
<td>ORNL</td>
<td>$\sim 4.2 \times 10^4$</td>
<td>2007</td>
</tr>
<tr>
<td>SunFire x64</td>
<td>TACC</td>
<td>$\sim 5.2 \times 10^4$</td>
<td>2007</td>
</tr>
<tr>
<td>Cray XT4</td>
<td>ORNL</td>
<td>$\sim 2 \times 10^5$</td>
<td>2008</td>
</tr>
<tr>
<td>BlueGene/P</td>
<td>ANL</td>
<td>$\sim 5 \times 10^5$</td>
<td>2008</td>
</tr>
<tr>
<td>BlueGene/Q</td>
<td>ANL/LLNL</td>
<td>$\sim 10^6$</td>
<td>2010-2012</td>
</tr>
</tbody>
</table>
Background: Data Aggregation

Filter function:

\[ f(\text{in}_n(\text{CP}_i); f_{s_n}(\text{CP}_i)) \rightarrow f_{\text{out}_n}(\text{CP}_i); f_{s_{n+1}}(\text{CP}_i)g \]

- Packets from input channels
- Current filter state
- Output packet
- Updated filter state
Background: Filter Function

• Built on state join and difference operators
• State join operator, $t$
  - Update current state by merging inputs

\[ i_{n_n}(CP_i) \cdot f \cdot s_n(CP_i) \cdot ! \cdot f \cdot s_{n+1}(CP_i) \]

- Commutative: $a \cdot t \cdot b = b \cdot t \cdot a$
- Associative: $(a \cdot t \cdot b) \cdot t \cdot c = a \cdot t \cdot (b \cdot t \cdot c)$
- Idempotent: $a \cdot t \cdot a = a$
Background: Descendant Notation

\[ \text{desc}^0(CP_i) \]
\[ \text{desc}^1(CP_i) \]
\[ \text{desc}^{k-1}(CP_i) \]
\[ \text{desc}^k(CP_i) \]

\[ \text{fs}(\text{desc}^k(CP_i)) : \text{join of filter states of specified processes} \]
\[ \text{cs}(\text{desc}^k(CP_i)) : \text{join of channel states of specified processes} \]
TBÖN Properties: Inherent Redundancy Theorem

The join of a CP’s filter state with its pending channel state equals the join of the CP’s children’s filter states.

\[ fs_n(CP_0) \sqcap cs_{n,p}(CP_0) \sqcap cs_{n,q}(CP_0) = fs_p(CP_1) \sqcap fs_q(CP_2) \]
The join of a CP’s filter state with its pending channel state equals the join of the CP’s children’s filter states.

\[ fs(\text{desc}^0(CP_0)) \sqcap cs(\text{desc}^0(CP_0)) = fs(\text{desc}^1(CP_0)) \]
TBŌN Properties: All-encompassing Leaf State Theorem

The join of the states from a sub-tree’s leaves equals the join of the states at the sub-tree’s root and all in-flight data.
TBÖN Properties:  
All-encompassing Leaf State Theorem

The join of the states from a sub-tree’s leaves equals the join of the states at the sub-tree’s root and all in-flight data

From Inherent Redundancy Theorem:

\[
\begin{align*}
fs(desc^1(CP_0)) &= fs(desc^0(CP_0)) \land cs(desc^0(CP_0)) \\
fs(desc^2(CP_0)) &= fs(desc^1(CP_0)) \land cs(desc^1(CP_0)) \\
&\quad \vdots \\
fs(desc^k(CP_0)) &= fs(desc^{k-1}(CP_0)) \land cs(desc^{k-1}(CP_0)) \\
fs(desc^k(CP_0)) &= fs(CP_0) \land cs(desc^0(CP_0)) \land \cdots \land cs(desc^{k-1}(CP_0))
\end{align*}
\]
TBON Theory

- TBŌN end-to-end argument: output only depends on state at the end-points

- Can recover from lost of any internal filter and channel states
State Composition

If CP$_j$ fails, all state associated with CP$_j$ is lost.

TBÔN Output Theorem:
Output depends only on channel states and root filter state.

All-encompassing Leaf State Theorem:
State at leaves subsume channel state (all state throughout TBÔN).

Therefore, leaf states can replace lost channel state without changing computation’s semantics.
State Composition Algorithm

if detect child failure
    remove failed child from input list
    resume filtering from non-failed children
endif

if detect parent failure
    do
        determine/connect to new parent
        while failure to connect
        propagate filter state to new parent
    endif