STORM: Lightning-Fast Resource Management

Eitan Frachtenberg, Fabrizio Petrini, Juan Fernandez, Scott Pakin, and Salvador Coll

fabrizio@lanl.gov

http://www.c3.lanl.gov/~fabrizio

Performance and Architecture Laboratory
CCS-3 Modeling, Algorithms, and Informatics Group
Los Alamos National Laboratory
• More effective use of cluster resources
  • Lower response time
  • Higher throughput
• Transparent fault tolerance
  • No application modifications
Buffered Coscheduling (BCS) is a new methodology to:

- Improve system responsiveness and utilization,
- Tolerate inefficient programs (communication and load imbalance),
- Implement fault-tolerance
Buffered Coscheduling tries to achieve these goals by greatly simplifying the system software (resource management, communication libraries and fault-tolerance)
Buffered Coscheduling implements resource management, communication libraries and fault-tolerance on top of a common microkernel.
In this talk we will focus on STORM, a resource manager implemented on top of the Buffered Coscheduling microkernel.
STORM (Scalable TOol for Resource Management)

- **Goals**
  - Portability
  - High performance resource management
  - Research tool to investigate new job scheduling algorithms

- **Key innovation**: software architecture that enables resource management to exploit low-level network features
Outline

- Overview of resource management
- STORM architecture
- Implementation
- Performance evaluation
- Scalability analysis
Resource Management

- Resource allocation for parallel jobs
- Job launch and termination
- Cluster management
- Monitoring and debugging
## Characteristics of Desktops versus Clusters

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Desktop</th>
<th>Cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean time between user-visible failures</td>
<td>Years</td>
<td>Days down to hours</td>
</tr>
<tr>
<td>Scheduling</td>
<td>Timeshared</td>
<td>Batch queued or gang scheduled with large quanta</td>
</tr>
<tr>
<td>Job-launching speed</td>
<td>&lt; 1 second</td>
<td>Arbitrarily long (batch) or many seconds (gang scheduled)</td>
</tr>
</tbody>
</table>
State of the art in Resource Management

Resource Managers (e.g., PBS, LSF, RMS, LoadLeveler, Maui) are typically implemented using

- TCP/IP
  - Favors portability over performance
- Non-scalable algorithms for the distribution/collection of data and control messages
  - Favors development time over performance
- Performance not important for small clusters, but crucial for large clusters → need fast and scalable resource management
STORM implementation structure

- **STORM functions**
  - Heartbeat, file transfer, termination detection
  - Flow control, queue management

- **(STORM helper functions)**

- **STORM mechanisms**
  - XFER-AND-SIGNAL, TEST-EVENT, COMPARE-AND-WRITE

- **Network primitives**
  - Remote DMA, network conditionals, event signaling, ...
STORM mechanisms

STORM is based on only three mechanisms

**Xfer-and-Signal** Transfer (PUT) a block of data from local memory to the global memory of a set of nodes (possibly a single node).

**Test-Event** Local synchronization

**Compare-and-Write** Global query with boolean reduction

Efficient and scalable implementation of these mechanisms $\rightarrow$ STORM is scalable
Hardware support for XFER-AND-SIGNAL

- **XFER-AND-SIGNAL** transfers multicast a block of data to a group of nodes
- The multicast can be executed in HW
Hardware support for XFER-AND-SIGNAL

- The packet is routed through a root node during the ascending phase
- The flow-through latency of each switch is only a few tens of nanoseconds
The packet reaches the set of destinations during the descending phase.
Hardware support for XFER-AND-SIGNAL

- The packet reaches the set of destinations during the descending phase
Hardware support for XFER-AND-SIGNAL

- The packet reaches the set of destinations during the descending phase
Hardware support for COMPARE-AND-WRITE

- COMPARE-AND-WRITE executes a binary query on a set of nodes
Hardware support for COMPARE-AND-WRITE

- COMPARE-AND-WRITE executes a binary query on a set of nodes
Hardware support for 

COMPARE-AND-WRITE

• COMPARE-AND-WRITE executes a binary query on a set of nodes
Hardware support for

COMPARE-AND-WRITE

- **COMPARE-AND-WRITE** executes a binary query on a set of nodes
Hardware support for COMPARE-AND-WRITE

- COMPARE-AND-WRITE executes a binary query on a set of nodes
Hardware support for
COMPARE-AND-WRITE

- The results of the global query are combined on the way up
- The “worst” result wins: Yes if all the nodes send a positive ack, No otherwise
The results of the global query are combined on the way up.

The “worst” result wins: Yes if all the nodes send a positive ack, No otherwise.
The results of the global query are combined on the way up.

The “worst” result wins: Yes if all the nodes send a positive ack, No otherwise.
Hardware support for COMPARE-AND-WRITE

The STORM mechanisms XFER - AND - SIGNAL and COMPARE-AND-WRITE can be easily and efficiently implemented on top of the hardware broadcast.
The STORM mechanisms XFER-AND-SIGNAL and COMPARE-AND-WRITE can be easily and efficiently implemented on top of the hardware broadcast.
• COMPARE-AND-WRITE scales efficiently on Lemieux, Pittsburgh Supercomputing Center. Less than 10 µs on 768 nodes/3072 processors.
### Portability of the STORM mechanisms

<table>
<thead>
<tr>
<th>Network</th>
<th><strong>COMPARE-AND-WRITE (µs)</strong></th>
<th><strong>XFER-AND-SIGNAL (MB/s)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Gigabit Ethernet</td>
<td>$46 \log n$</td>
<td>Unknown</td>
</tr>
<tr>
<td>Myrinet</td>
<td>$20 \log n$</td>
<td>$\sim 15n$</td>
</tr>
<tr>
<td>Infiniband</td>
<td>$20 \log n$</td>
<td>Unknown</td>
</tr>
<tr>
<td>QsNET</td>
<td>$&lt; 10$</td>
<td>$&gt; 150n$</td>
</tr>
<tr>
<td>BlueGene/L</td>
<td>$&lt; 2$</td>
<td>$700n$</td>
</tr>
</tbody>
</table>
Experimental Results

- Setup
  - 64 nodes/256 processors ES40 Alphaserver cluster
  - 2 independent rails of Quadrics
  - Linux 2.4.3
  - Files are placed in a RAM disk, in order to avoid I/O bottlenecks

- Experiments
  - Job Launching
  - Job Scheduling
Launch times (unloaded system)

- The launch time is essentially constant when we increase the number of processors $\rightarrow$ STORM is highly scalable
Launch times (loaded system, 12MB executable)

- Launch time is more sensitive to network load rather than CPU load.
- In the worst-case scenario it still takes only 1.5 seconds to launch a 12 MB file on 256 processors.
The model shows that in an ES40-based Alphaserver a 12 MB binary can be launched in only 135 ms on 16,384 nodes.
Measured and predicted performance of existing job launchers

We compare the job launching performance of STORM with

- rsh
- RMS
- GLUnix
- Cplant
- Bproc
Measured and predicted performance of existing job launchers

![Graph showing measured and predicted performance of different job launchers.](image)

- rsh (measured)
- rsh \( t = 0.934n + 1.266 \)
- RMS (measured)
- RMS \( t = 0.077n + 1.092 \)
- GLUnix (measured)
- GLUnix \( t = 0.012n + 0.228 \)
- Cplant (measured)
- Cplant \( t = 1.379 \lg n + 6.177 \)
- BProc, measured
- BProc, \( t = 0.413 \lg n - 0.084 \)
- STORM (measured)
- STORM (modeled; see text)
Relative performance of Cplant, BProc, and STORM

- Cplant: $t = 1.379 \lg n + 6.177$
- BProc: $t = 0.413 \lg n - 0.084$
- STORM (modeled; see text)

Graph showing the factor of STORM time against the number of nodes, with lines indicating the performance of each system.
Effect of time quantum with an MPL of 2

- Cluster-wide jobs can be scheduled as fast a local process on a desktop OS.
Effect of node scalability

The scheduling algorithm is scalable with the number of nodes
A selection of scheduling quanta found in the literature

<table>
<thead>
<tr>
<th>Resource Manager</th>
<th>Minimal feasible scheduling quantum</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMS</td>
<td>30,000 milliseconds on 15 nodes (1.8% slowdown)</td>
</tr>
<tr>
<td>SCore-D</td>
<td>100 milliseconds on 64 nodes (2% slowdown)</td>
</tr>
<tr>
<td>STORM</td>
<td>2 milliseconds on 64 nodes (no observable slowdown)</td>
</tr>
</tbody>
</table>
Conclusions

- STORM uses an innovative design based on a small set of data-transfer and synchronization mechanisms:
  - XFER-AND-SIGNAL
  - TEST-EVENT
  - COMPARE-AND-WRITE
- STORM’s design makes it orders of magnitude faster than the best reported results in the literature for both job launching and process scheduling.
Conclusions (continued)

- STORM is a lightweight, flexible and scalable environment for performing resource management in large-scale clusters.
- It is indeed possible to scale up a cluster without sacrificing job-launching times, machine efficiency or interactive response time.
- HW support for collective communication can simplify system software and can help to achieve efficiency and scalability.
More information can be found at the following URLs:

Los Alamos Performance and Architecture Laboratory
http://www.c3.lanl.gov/par_arch

Resource management
http://www.c3.lanl.gov/~fabrizio

Quadrics network
http://www.quadrics.com and
http://www.c3.lanl.gov/~fabrizio/quadrics.html

DEMO in LANL booth (R3211)
Quadrics Network: Elan

- SDRAM I/F
- Thread Processor
- μ-code Processor
- DMA Buffers
- Inputter
- Link Mux
- FIFO 0
- FIFO 1
- MMU & TLB
- Table Walk Engine
- Clock & Statistics Registers
- PCI Interface
- 4 Way Set Associative Cache

- 72
- 64
- 64
- 64
- 28
- 28
- 28
- 28
- 66MHz
- 100 MHz
- 200MHz
- 10
- 10

TLB Synchronized with Host