The Ganglia Distributed Monitoring System
Scalable distributed monitoring system for high-performance computing systems such as clusters and Grids.

Relies on a multicast-based listen/announce protocol to monitor state within clusters and uses a tree of point-to-point connections amongst representative cluster nodes to federate clusters and aggregate their state.

XML for data representation, XDR for compact, portable data transport, and RRDTool for data storage and visualization.

Ported to extensive set of operating systems and processor architectures.
Motivation for development

- Clusters are now the de-facto building block for high performance systems – the need for scale and reliability has become key issues.

- Heterogeneity was previously non-issue when running a single vector supercomputer or an MPP, but now must be designed for from the beginning, since systems that grow over time are unlikely to scale with the same hardware and software base.

- Since clusters today consist of hundreds or thousands of nodes, manageability has become a key issue also.
Using Ganglia for Wikipedia

- http://ganglia.wikimedia.org/latest/

- Paper: http://www.ece.cmu.edu/~ece845/docs/massie04.pdf
Key Design Challenges

- **Scalability** – scale gracefully with the increase in no. of nodes in the systems.

- **Robustness** – robust to node and network failures of various types.

- **Extensibility** – extensible in the types of data that are monitored and the nature in which such data is collected.

- **Manageability** – time and effort required to administer the system does not grow linearly with increase in the number of nodes in the cluster.

- **Portability** – portable to a variety of operating systems and CPU architectures.

- **Overhead** – incur low per-node overheads for all scarce resources including CPU, memory, I/O, and network bandwidth.
Clusters – set of nodes that communicate over a high bandwidth, low latency interconnect such as Myrinet or Gigabit Ethernet. Nodes are frequently homogeneous in both hardware and OS, the network rarely partitions, and the system is managed by a single administrative entity.

Grids – set of heterogeneous systems federated over a wide-area network. Usually interconnected using special high speed, wide-area networks (e.g.: Abilene, TeraGrid’s DTF network) in order to get the bandwidth required for applications. It frequently involves distributed management by multiple administrative entities.

Planetary-scale systems – wide area distributed systems whose geographical extent covers a good fraction of the planet. Built as overlay networks on top of the existing Internet. Bandwidth is not nearly as abundant compared to clusters or Grids, network bandwidth is no cheap, and network experiences congestion and partitions frequently.
Architecture
How does it function?

- To monitor state within clusters
  - Heartbeat messages on a well-known multicast address enables automatic discovery of nodes. No manual configuration of cluster membership lists.
  - Each node monitors its local resources and lets others know of its state.
  - Each node listens to monitoring data from other nodes.
  - Therefore, any node knows the entire state of the cluster.

**Assumption:** presence of a native multicast capability, that does not hold for the Internet in general and thus cannot be relied on for distributed systems such as Grid that require wide-area communication.
To monitor aggregate states of each cluster

- Each leaf node specifies a node in a specific cluster being federated, while nodes higher up in the tree specify aggregation points.

- Since each node contains a complete copy of its cluster’s monitoring data, each leaf node logically represents a distinct cluster.

- Aggregation at each point in the tree is done by polling child nodes at periodic intervals.

- Monitoring data from both leaf nodes and aggregation points is then exported using a TCP connection to the node being polled followed by a read of all its monitoring data.
**Implementation**

- **Ganglia Monitoring Daemon (Gmond)**
  - Provides monitoring on a single cluster by implementing the listen/announce protocol and responding to client requests by returning an XML representation of its monitoring data.
  - Runs on every node of the cluster

- **Ganglia Meta Daemon (Gmetad)**
  - Provides federation of multiple clusters.
  - TCP connections between multiple Gmetad daemons allows monitoring information for multiple clusters to be aggregated.

- **Gmetric and client side library**
  - Command-line program that applications can use to publish application-specific metrics, while the client side library provides programmatic access to a subset of Ganglia’s features.
Implementation – Ganglia Monitoring Daemon (GMOND)

Responsible for local node info, publishing it and sending heartbeats.

Listens on multicast channel for monitoring data from other nodes, and updates in-memory.

Hash table of monitoring metrics.

Dedicated pool of threads to process client requests for monitoring data.
Implementation – Gmond continued ...

- All data stored is soft state & never written to disk.
- Data stored in hierarchical hash table that uses reader–writer locks for concurrency. Concurrency allows
  - *listening* threads to store incoming data from multiple unique hosts simultaneously.
  - competition resolution between *listening* and *XML export* threads for access to host metric records.
- Monitoring data is received in XDR format and stored in binary form to reduce physical memory usage. Allows for more rapid processing of the incoming data.
- Built-in vs. User-defined metrics (gmond distinguishes the two based on a field in the multicast packet)
- For portability, all metrics published in XDR format and collected from well-defined interfaces e.g. /proc, kvm
A static metric lookup table used to report built-in metrics. Only unique key + metric value needs to be sent per announcement.

Not possible with user-defined metrics.

Default values can be changed at compile time.

The collection and value thresholds reduce resource usage by collecting local node data and sending multicast traffic only when significant updates occur.

### Key (xdr_u_int) Metrics Value Format

<table>
<thead>
<tr>
<th>Key</th>
<th>Metric</th>
<th>Value Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>User-defined</td>
<td>Explicit</td>
</tr>
<tr>
<td>1</td>
<td>cpu_num</td>
<td>xdr_u_short</td>
</tr>
<tr>
<td>2</td>
<td>cpu_speed</td>
<td>xdr_u_int</td>
</tr>
<tr>
<td>3</td>
<td>mem_total</td>
<td>xdr_u_int</td>
</tr>
</tbody>
</table>

### Metric Collected (s) Value Thres. Time Thresh.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Collected (s)</th>
<th>Value Thres.</th>
<th>Time Thresh.</th>
</tr>
</thead>
<tbody>
<tr>
<td>User-defined</td>
<td>explicit</td>
<td>explicit</td>
<td>explicit</td>
</tr>
<tr>
<td>cpu_num</td>
<td>once</td>
<td>none</td>
<td>900-1200</td>
</tr>
<tr>
<td>cpu-speed</td>
<td>once</td>
<td>none</td>
<td>900-1200</td>
</tr>
<tr>
<td>mem_total</td>
<td>once</td>
<td>none</td>
<td>900-1200</td>
</tr>
<tr>
<td>load_one</td>
<td>15-20</td>
<td>1</td>
<td>50-70</td>
</tr>
</tbody>
</table>
Two time limits – a soft limit ($T_{\text{max}}$) and a hard limit ($D_{\text{max}}$).

Reports $T_n$ (time elapsed since collection) and $T_{\text{max}}$ to clients. If $T_n > T_{\text{max}}$, clients know that message was not delivered and the value may be incorrect.

Exceeding hard limit causes the monitoring data being permanently removed from Gmond’s hierarchical hash table of metric data.

Each heartbeat contains a start timestamp. Altered timestamp lets peers know the instance has been restarted.

Gmond that has not responded some number of time thresholds is down.

When new or restarted, all local metric time thresholds are reset. Makes sure that rarely published metrics are known to the restarted or new host immediately.

To avoid huge multicast storms if every gmond in a cluster is restarted simultaneously, the time-threshold reset mechanism only occurs if gmond is more than 10 minutes old.
Implementation – Ganglia Meta Daemon (Gmetad)

- At each node in the tree, Gmetad periodically polls a collection of child data sources, parses the collected XML, and saves the metric values to round-robin database.

- Stored data exported in XML format over TCP socket to clients.

- IP address and port pair to identify data source. Multiple IP addresses for failover.

- A dedicated data collecting thread for each data source.

- Use of SAX Parser to parse collected XML data (lower CPU overhead).

- SAX callback function uses the perfect HASH function generated by GNU gperf instead of raw string comparisons.
RRDtool Round Robin Database (compact, constant size) to store and visualize historical monitoring information.

For data at different time granularities, RRDtool generates graphs which plot historical trends of metrics vs. time.

The generated graphs are displayed to the clients using a PHP web front-end.

TemplatePower is used for the web front-end to create a strict separation between content and presentation.
Evaluation

Four production distributed systems for evaluation

- **Millennium (UC Berkeley)** – cluster of approximately 100 SMP nodes each with either 2 or 4 CPUs. All nodes connected via Gigabit Ethernet and Myrinet.

- **HPC Linux cluster (SUNY Buffalo)** – approximately 2000 dual-processor SMP nodes connected using Gigabit Ethernet and running Linux 2.4.18 SMP kernel.

- **Federation of clusters (UC Berkeley)** – Four clusters in same location. 100-node Millennium cluster, 45-node cluster, 4-node cluster, and 3-node cluster.

- **PlanetLab** – consists of 102 nodes distributed across 42 sites spanning three continents (America, Europe, and Australia). Each site is a small cluster of 2–3 nodes.
Evaluation – Overhead

- Local overhead incurred within the node

- Local node overhead for aggregation with gmetad

- Global overhead (network bandwidth)
Evaluation – Scalability

(a) Local-area multicast bandwidth

(b) Local-area multicast packets per second

Figure 3: Scalability as a function of cluster size.
Evaluation – Scalability

Figure 4: Scalability as a function of number of clusters.
Real systems experience

- Helped figure out ways to exercise its functionality.

- Changes had to be made to seemingly good original design decisions. Architecture has evolved, features added, and implementation refined.

- To gain popularity, design choice was
  - multicast to avoid manual configuration
  - standard configuration tool such as automake and autoconf

- Use of widely used, simple self-contained technologies such as XML for data representation and XDR for data transport was a good design choice
Real systems experience

- Support for broad range of clusters (heterogeneity and scale) exposed issues that were not significant factors in its early deployment.

- Found assumption of a functional native, local-area IP multicast not to hold true in number of cases.

- Step forward by its implementation on a planetary-scale system. Assumption that wide-area bandwidth is cheap not true on the public Internet.

- Multicast may not be a good choice as nodes reach thousands.

- For federation of multiple clusters, straightforward aggregation will present problems.
Discussion of related work

- Cluster monitoring efforts focusing on scale
  - Supermon
  - CARD
  - PARMON
  - Big Brother

Ganglia differs
- Hybrid approach to monitoring.
- Use of widely available technologies (XML, XDR).
- Simple design and sound engineering to achieve high levels of robustness, ease of management and portability.
- Demonstrated operation at scale.
Theius: A Streaming Visualization Suite for Hadoop Clusters

Jon Tedesco

IC2E 2013, San Francisco, CA, USA
Jon Tedesco, Roman Dudko, Abhishek Sharma, Reza Farivar, Roy Campbell
Theius: A Streaming Visualization Suite for Hadoop Clusters

Jon Tedesco
Motivation

- Monitoring
- Prediction

Problem
- System administrators
  - Bottleneck for detecting & responding to failures
  - Communicate state of system quickly
- What good is all this data if we can’t understand it?
Related Work

- Cloud monitoring
  - Streaming, real-time data
  - What type of data can we collect?

- Failure prediction
  - Online prediction algorithms (real-time)
  - What type of predictions can we make?

- Visualization
  - What are the state-of-the-art visualization techniques for this data?
Related Work – Visualization

- Existing systems
  - Hadoop task, log, analysis, performance analysis
  - Visual presentation specifically?

- Ganglia
  - Complete monitoring solution
  - Accessible & widely used
  - State-of-the-art visualizations
Related Work – Ganglia

Overview of Triton Cluster

CPUs Total: 2992
Hosts up: 287
Hosts down: 0

Avg Load (15, 5, 1m):
46%, 46%, 46%

Localtime:
2012-04-30 22:58

Rocks Tools:
Job Queue | Cluster Top | Gmetrics

Cluster Load Percentages

- 100+ (21.60%)
- 75-100 (1.085%)
- 50-75 (1.392%)
- 25-50 (30.21%)
- 6-25 (55.77%)

Triton Cluster CPU last hour

Triton Cluster Memory last hour

Triton Cluster Network last hour

Triton Cluster load_one ( ) last hour sorted descending | Columns 4

tcc-5-48.local

tcc-5-32.local

tcc-5-23.local

tcc-5-28.local
Related Work – Ganglia

```
This host is up and running.

Time and String Metrics

bootstrap          Mon, 12 Dec 2011 08:58:09 -0800
machine_type       x86_64
ps                  
ps-0                pid=17666, cmd=gnomed, user=nobody, %cpu=5.63, %mem=0.02, size=64, data=2536, shared=1964, vm=114028
ps-1                pid=1, cmd=init, user=root, %cpu=0.00, %mem=0.00, size=36, data=32, shared=460, vm=10372
ps-2                pid=2, cmd=migration/0, user=root, %cpu=0.00, %mem=0.00, size=0, data=0, shared=0, vm=0
ps-3                pid=3, cmd=ksofirqd0, user=root, %cpu=0.00, %mem=0.00, size=0, data=0, shared=0, vm=0
ps-4                pid=4, cmd=watchdog/0, user=root, %cpu=0.00, %mem=0.00, size=0, data=0, shared=0, vm=0
ps-5                pid=5, cmd=migration/1, user=root, %cpu=0.00, %mem=0.00, size=0, data=0, shared=0, vm=0
ps-6                pid=6, cmd=ksofirqd1, user=root, %cpu=0.00, %mem=0.00, size=0, data=0, shared=0, vm=0
ps-7                pid=7, cmd=watchdog/1, user=root, %cpu=0.00, %mem=0.00, size=0, data=0, shared=0, vm=0
sys_clock           Mon, 30 Apr 2012 23:00:28 -0700
uptime              140 days, 13:2

Constant Metrics

cpu_num             8 CPUs
cpu_speed           2400 MHz
mem_total           24676844 KB
swap_total          4096552 KB
```
Design Principles

- Interactive
  - Responsive and controllable
- Real-time
  - Streaming, real-time, automatic
- Informative
  - Direct attention to potential problems and artifacts
- Intuitive
  - Demand skill, not experience
- Scalable
  - Visualize large clusters without sacrificing usability
Design Principles

- Objectives
  - Streaming data
  - Configurable and interactive
  - Informative

- Use cases
  - Heterogeneous cluster
  - Rack failure
  - Node failure
  - Uneven load distribution
Visualization – Architecture

- **Architecture**
  - Simulator
    - Generates simulated cluster data
    - Streams data to clients
  - Webpage
    - Asynchronous & interactive

- **Implementation**
  - JavaScript
    - d3.js
    - jQuery
  - Python
  - AJAX
Visualization – Architecture

Data
- Methodology
  - Data types from previous work
  - Heuristic values
- Examples
  - CPU, memory, context switch rate
  - Log events
  - MapReduce tasks and jobs
  - Failure or event prediction
Visualization – Overview

The diagram shows a topology graph indicating node health. The graph features nodes with varying colors and sizes, representing different states and conditions.

Controls:
- Resume View

Visualization:
- Color
- Node Memory Usage (%)
- Size
- Node Health (%)

Data Characteristics:
- One rack prone to failure

Legend:
- Node Memory Usage (%) of 1.0
- Node Memory Usage (%) of 0.0

Information:
<table>
<thead>
<tr>
<th>Events</th>
<th>Nodes</th>
<th>Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>08/05/12 22:20</td>
<td>ERROR</td>
<td></td>
</tr>
<tr>
<td>08/05/12 22:20</td>
<td>ERROR</td>
<td></td>
</tr>
<tr>
<td>08/05/12 22:20</td>
<td>FATAL</td>
<td></td>
</tr>
<tr>
<td>08/05/12 22:20</td>
<td>INFO</td>
<td></td>
</tr>
<tr>
<td>08/05/12 22:20</td>
<td>FATAL</td>
<td></td>
</tr>
<tr>
<td>08/05/12 22:20</td>
<td>INFO</td>
<td></td>
</tr>
<tr>
<td>08/05/12 22:20</td>
<td>FATAL</td>
<td></td>
</tr>
<tr>
<td>08/05/12 22:20</td>
<td>WARN</td>
<td></td>
</tr>
<tr>
<td>08/06/12 22:20</td>
<td>FATAL</td>
<td></td>
</tr>
<tr>
<td>08/05/12 22:20</td>
<td>WARN</td>
<td></td>
</tr>
<tr>
<td>08/05/12 22:20</td>
<td>ERROR</td>
<td></td>
</tr>
<tr>
<td>08/05/12 22:20</td>
<td>WARN</td>
<td></td>
</tr>
<tr>
<td>08/05/12 22:20</td>
<td>FATAL</td>
<td></td>
</tr>
<tr>
<td>08/05/12 22:19</td>
<td>WARN</td>
<td></td>
</tr>
<tr>
<td>08/05/13 22:19</td>
<td>WARN</td>
<td></td>
</tr>
<tr>
<td>08/05/12 22:19</td>
<td>FATAL</td>
<td></td>
</tr>
<tr>
<td>08/05/12 22:19</td>
<td>WARN</td>
<td></td>
</tr>
<tr>
<td>08/05/12 22:19</td>
<td>WARN</td>
<td></td>
</tr>
</tbody>
</table>
Visualization – Main Visualization
Visualization – Navigation
Visualization – Tree
Visualization – MapReduce
Visualization – TreeMap
Visualization – Circle Packing
Visualization – Scatterplot
Visualization – Individual Node

Node Information

- CPU Usage: 0.76
- Memory Usage: 0.93
- Context Switch Rate: 0.69
- Last Failure Time: 08/05/12 22:33
- Predicted Failure Time: 08/05/12 22:33
- Prob of FATAL Event: 0.24
- Prob of ERROR Event: 0.14
- Prob of WARN Event: 0.02
- Prob of INFO Event: 0.46
- Estimated Health: 0.30

Node Visualization Controls

- Break Down By:
  - Event severity
  - Event facility
Visualization – Controls
Visualization – Aggregate Data
Visualization – Timeline

History Controls

Topology Graph Showing Node Health

Information

<table>
<thead>
<tr>
<th>Events</th>
<th>Nodes</th>
<th>General</th>
</tr>
</thead>
<tbody>
<tr>
<td>06/05/12 22:20 ERROR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>06/05/12 22:20 ERROR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>06/05/12 22:20 FATAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>06/05/12 22:20 INFO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>06/05/12 22:20 FATAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>06/05/12 22:20 FATAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>06/05/12 22:20 FATAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>06/05/12 22:20 WARN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>06/05/12 22:20 FATAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>06/05/12 22:20 FATAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>06/05/12 22:20 FATAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>06/05/12 22:19 WARN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>06/05/12 22:19 WARN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>06/05/12 22:19 WARN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>06/05/12 22:19 WARN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>06/05/12 22:19 WARN</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Experimental Setup

- 5 graduate students at UIUC
  - No prior experience with Ganglia or Theius
- 4 comparative tasks
  - Both Ganglia & Theius
- 6 scenarios for trends and correlations
  - Theius only
- Timings & subjective feedback
User Study – Comparative Tasks

- Tasks
  - Scenario 1
    - CPU usage in single node
  - Scenario 2
    - Node with highest CPU
  - Scenario 3
    - High memory usage nodes
  - Scenario 4
    - Aggregate cluster use
User Study – Theius Tasks

- Task 1
  - Identify abnormal rack in heterogeneous cluster 2.2 s
- Task 2
  - Identify rack with abnormal CPU usage 6.2 s
- Task 3
  - Identify machine that logged the last fatal error 10.0 s
- Task 4
  - Identify machine with high CPU, memory usage, or context switch rate 67.4 s
- Task 5
  - Identify rack with high CPU, memory usage, or context switch rate 1.2 s
- Task 6
  - Identify correlation between context switch rate and CPU usage 7.8 s
User Study – Subjective Feedback

- Intuitive
  - Intuitive & interactive interface

- Observed trends without prior experience
  - All users new to both Theius & Ganglia

- Scatterplot matrices
  - Particularly interesting for correlations