Wide Area Cluster Monitoring with Ganglia

Federico D. Sacerdoti, Mason J. Katz
San Diego Supercomputing Center
{fds, mjk}@sdsc.edu

Matthew L. Massie, David E. Culler
Univ. of California, Berkeley
{massie, culler}@cs.berkeley.edu

Abstract

In this paper, we present a structure for monitoring a large set of computational clusters. We illustrate methods for scaling a monitor network comprised of many clusters while keeping processing requirements low. A design for presenting high-level web-based summaries of the monitor network is provided, along with a generalization to a distributed, multiple-resolution monitoring tree. Emphasis is placed on scalability, fast query response, fault tolerance, and grid compatibility. Experimental evidence is presented that demonstrates the performance of our design.

1. Introduction

Computational clusters made from commodity components have gained an established seat in production environments. [5,17] Like other production-class resources, we desire to monitor clusters for auditing, accounting, performance assessment, and design feedback purposes. Monitoring clusters comprised of distributed components also creates unique challenges. First, we need a "nervous system" in the cluster that alerts us to damaged components. Second, detailed monitoring enables users to effectively tune the performance of parallel applications.

Wide-area monitoring is necessary because we often wish to process and display monitored information at a remote location. It is also helpful to provide statistics over multiple clusters, to give a larger yet less detailed view of computing resources. This concept generalizes to a multiple-resolution view of a cluster set, where the system can provide an overall summary and other views with progressively more detail about fewer clusters. In this paper we focus on novel techniques to efficiently support a multiple-resolution view of large cluster sets.

The strategy uses a distributed tree structure that enables organizations to monitor an arbitrarily large number of clusters while placing bounds on the required processing load. To enable efficient multiple-resolution views, we organize monitored data in structures that support high-speed queries with different levels of detail.

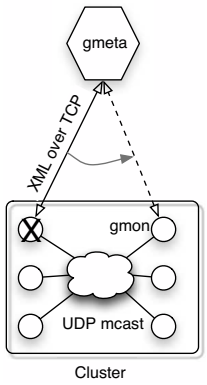

Figure 1: Ganglia local and wide area monitor interaction. Gmon runs on each cluster node; gmeta can fail over between nodes.

Our work extends the wide-area scalability and performance of the Ganglia monitoring system [1]. Ganglia is comprised of two components, the *Gmon* local-area monitoring system, and the *Gmeta* wide-area system, the emphasis of this paper. `Gmeta` processes and presents monitoring information gathered from one or more clusters running the `Gmon` local-area monitor.

The Gmon system operates at the cluster level and gathers metrics such as heartbeats, hardware/operating system parameters, and user-defined key-value pairs from every node. Gmon uses *UDP multicast* to exchange these metrics within the cluster. The local-area multicast backbone enables gmon agents to organize into a redundant, leaderless network where nodes listen to their neighbors rather than polling them [1]. With this method, the monitor does not need *a priori* knowledge of cluster nodes. Gmon can adapt to a dynamically changing cluster, using soft-state techniques [15] to incorporate newly arrived and departed nodes automatically. Practical experience shows that such a model is beneficial for the dynamic environment inside commodity clusters.

Gmon communicates with its Gmeta counterpart using XML streams sent over TCP connections (fig 1). These reports are suitable for travel over wide area networks, where multicast support is unreliable. However TCP is not enough. Any monitoring system that operates over the wide-area must handle remote failures.

Both stop and intermittent failure masking is facilitated by Gmon's design. All Gmon agents have redundant global knowledge of the cluster, so that any node can supply a complete report containing the state of itself and all its neighbors. The wide-area Gmeta uses this ability to automatically fail-over when a cluster node malfunctions (fig 1), preventing a node stop failure from disrupting its monitoring activities. To handle intermittent failures, Gmeta retries the failed node periodically. If multiple failures render the monitored cluster unreachable, Gmeta keeps a set of metric histories that aid in forensic analysis.

This work concentrates on how monitoring information is aggregated, archived, and delegated. The wide-area Gmeta concerns itself only with a metric's type and context: which host, and in which cluster it originated from. We show how additive reductions on monitoring data (called summaries) reduce the amount of network and processing resources required by the monitoring system, and how a simple query language leads to dramatic increases in the efficiency of the presentation layer.

The rest of this paper is organized as follows. In Section 2 we present related work concerning cluster monitoring. Section 3 outlines the principle requirements and goals of a monitoring system, follwed by the characterization of several approaches designed to meet them. In Section 4 we present quantitative evidence of the performance of our solution, and in Section 5 we discuss limitations inherent in our strategy and paths for future work. Section 6 finally concludes our findings.

1. Related Work

While Ganglia specifies a strong delineation between cluster-level monitoring and wide-area monitoring, many similar systems do not. Accordingly we concentrate on monitors that provide wide-area capabilities. Related systems also vary in terms of their monitoring strategy (polling or event-driven) and their information hierarchy (host-based, user-based, or job-based).

The Supermon system [2] employs a wide-area monitoring strategy similar to our own. A *mon* server

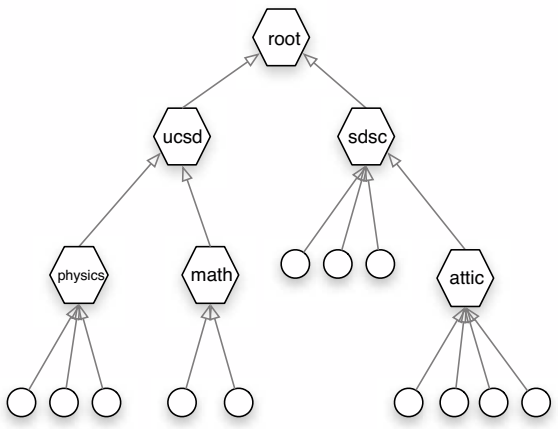

Figure 2: A hierarchical monitoring tree. Hexagons are wide-area monitors, circles represent cluster-area gmons. Grey lines denote trust edges. This configuration is used in the experimental section as well.

on every node serves monitoring data on a TCP port. A *supermon* server collects this data by serially connecting to each mon server. Supermon must have a priori knowledge of each cluster node; the system cannot incorporate new nodes without an explicit registration step. The system keeps no record of metric history, making time-series measurements such as CPU% difficult.

Supermon requires O(CH) network connections to obtain cluster state, where CH is the number of hosts in all clusters. Ganglia requires just one (to its multicast channel) and by gathering knowledge gradually over time, can satisfy queries using only its local state, without the need for any communication. However, its mon agents are quite lightweight and support rapid polling rates. Its ability to organize into a tree hierarchy is also similar to our own design.

Both Supermon and Ganglia use recursive languages to represent monitored data, S-expressions and XML respectively. This choice allows the desirable characteristic of hierarchical composability. A Supermon provides output in the same format as mon, enabling traditional hierarchies for the distribution of monitoring tasks. Although information flow from all clusters to the root can become overwhelming, this tree structure is the key to efficiently manage an arbitrarily large number of clusters (fig 2).

Clumon [3] is a monitoring system developed at NCSA. It relies on a subset of SGI's Performance Co-Pilot monitoring system [4] to collect host data, and additionally tracks jobs submitted to PBS queues. Clumon uses a job-based information hierarchy,

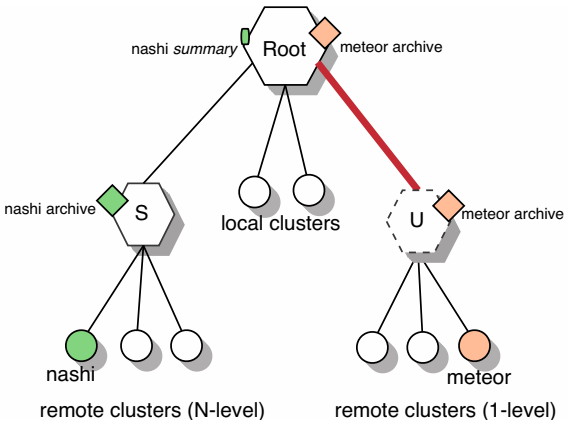

Figure 3: Two wide-area monitor designs. The 1-level gmeta on the right reports the union of its children's data to root. The more scalable *N-level* monitor reports a summary of its children, which lightens the root's archiving load.

organized around the PBS batch queues. The system has no allowance for monitoring multiple clusters as one cohesive unit, nor does its native data format (SQL) allow a hierarchical structure similar to Ganglia and Supermon.

Ganglia VO (Virtual Organization) [9] is a specialized version of Ganglia developed at the University of Chicago. Their local-area monitor is unaltered, while the wide-area system extends Ganglia to allow a 2-level monitoring tree, and can report summary data at each level. Ganglia VO explores fractional access policies on a grid of clusters, and has a user/group-centric information hierarchy based on virtual organizations. The authors emphasize policy semantics and enforcement over wide-area scaling and performance.

The Metacomputing Directory Service (MDS) [18] is the wide area information service used by the Globus project [19]. Although similar to a wide-area cluster monitoring system, MDS is designed to characterize a resource rather than profile it. As such, its storage system is biased towards relatively static cluster attributes such as their network interface types, CPU speeds, and memory capacities. MDS does not keep metric histories, nor does its TTL-based caching method lend itself to tracking quickly changing metrics. However, MDS employs a more sophisticated certificate-based trust model than Ganglia, and its monitoring hierarchy has a more dynamic structure. Ganglia and MDS have a symbiotic relationship via interfaces that allow Ganglia's gmon to function as a leaf node in an MDS tree.

Dproc [20] uses kernel-level communication channels to efficiently distribute cluster monitoring information via the /proc virtual filesystem interface. Dproc currently does not support wide area monitoring.

Much work in the literature has explored strategies for network monitoring. The Network Weather Service [10] and work done at Bell Labs [7,8] fall into this category. Ganglia does not share this goal. We are biased towards monitoring physical hosts at the endpoints of a network, rather than the network itself. Therefore our design concentrates on minimizing the monitor's overall resource use rather than perturbing and measuring the network state.

2. Design

We begin by enumerating the significant requirements of an effective wide-area monitor.

- Light weight. This requirement is critical for accurate measurements, and because computing cycles are valuable.
- Low impact on monitors. The monitoring systems themselves have a finite computing capacity, which must suffice for gathering, archiving, and querying activities on monitored data.
- Robust in face of cluster failures. Any monitor operating over the wide area must deal gracefully with remote failures.
- Scalability. The monitoring system must scale to handle an arbitrarily large number of clusters. At the limit we can imagine every cluster in the world connected together in a single massive Grid.
- Historical archives. Monitors by nature handle streams of time series data. The system should efficiently archive these streams and support basic queries against them.

These requirements are desirable in an effective wide-area monitoring system. In following sections we present a set of designs that address them to varying degrees.

The monitoring tree model is common to all designs presented here. The nodes of the tree include all clusters in the set to be monitored, and wide-area gmeta agents. Non-leaf nodes are gmeta monitors, which are symmetrical to each other with no functionally distinct

```
<GRID NAME="SDSC" AUTHORITY="my URL">
 <CLUSTER NAME="Meteor">
  <HOST NAME="compute-0-0">
   <METRIC NAME="cpu_num" VAL="2" TYPE="int"/>
   <METRIC NAME="load_one" VAL=".89" TYPE="float"/>
    ...
  </HOST>
  <HOST NAME="compute-0-1">
   <METRIC NAME="cpu_num" VAL="2" TYPE="int"/>
   <METRIC NAME="load_one" VAL=".89" TYPE="float"/>
    ...
  </HOST>
 </CLUSTER>
 <GRID NAME="ATTIC" AUTHORITY="my URL">
   <HOSTS UP="10" DOWN="1"/>
   <METRICS NAME="cpu_num" SUM="20" NUM="10" />
   <METRICS NAME="load_one" SUM="17.56" NUM="10" />
   ...
 </GRID>
</GRID>
```

Figure 3: The Ganglia XML language with GRID tags. Each node in the cluster has about 30 monitoring metrics, which can also be user-defined. The nested *Attic* grid is in summary form.

leaders or slaves. Edges are trusts that allow TCP connections carrying XML monitoring data to occur. We manually configure the unidirectional trust edges such that a child must explicitly trust its parent. Leaf nodes are clusters running local-area gmon monitors, and may be attached anywhere in the tree (fig 2). Monitoring information originates at leaf nodes and is passed upwards towards the root along the trust edges.

In the design analysis the following symbols are used. C is given as the number of clusters in the cluster-set to be monitored. H is the maximum number of hosts in a cluster, and m represents the number of metrics monitored per host.

2.1. One-level monitor tree

The simplest organization for a wide-area monitor is to aggregate all clusters into a one level tree. In this design the tree root must perform monitoring activities for all clusters in the set. Early versions of Ganglia have this design despite their ability to form deeper trees.

Although we may organize a Gmeta monitor into a hierarchy, the system is not scalable. A node in the monitoring tree reports the union of its children's data to its parent, and will process and archive data for all clusters in its subtree. Nodes perform no reduction of monitoring data, forcing the root to bear the brunt of the data from the entire cluster set. When many clusters are present, the root may be quickly overwhelmed. In addition, every monitor between a cluster and the root will keep identical metric archives for that cluster (fig 3). As metric archiving is a processor-intensive task, this redundancy is unwanted. As a result this design does not satisfy the scalability requirement.

However, this design does address the requirements of remote cluster failures, low impact on monitored systems, and historical archiving. Remote failures are handled identically to link failures, and are detected with TCP timeouts. Since all gmon agents have redundant global state, stop and intermittent failures of cluster nodes are handled easily (fig 1). Even in cases of a complete partition with a cluster, the monitor will attempt to re-establish contact at a steady frequency, ensuring that failures do not cause permanent fissures in the monitoring tree.

Ganglia keeps historical records of data in specialized time-series databases [11], whose stream-based design supports a wide range of time scale queries employing lossy compression with a bias towards recent data. Therefore we can see a metric's history over the past year but with less resolution than if we ask about more recent behavior. The databases are highly optimized for this type of data and do not grow in size over time. With this facility Ganglia records the history of each numeric metric. If a monitored node has failed, it keeps a "zero" record during the downtime, aiding time-of-death forensic analysis.

A previous paper [1] has shown the impact of gmon on the clusters themselves is negligible even for large systems. As an example, the monitor on a 128-node cluster uses less than 56Kbps of network bandwith, roughly the capacity of a dialup modem.

While we assume that the wide area monitors are run on nodes that are not immediately in the "compute bound" pool of a cluster (like gmon), the resource requirements of gmeta are still of interest. As this design is susceptible to an overloaded root node, it does not satisfy the low monitor impact requirement.

The one-level monitoring tree as implemented by this design addresses three of the five requirements, leaving out scalability and low monitor impact. Although we have given only a brief analysis of the scalability shortcomings, the experimental section provides more detailed evidence.

2.2. N-level monitoring tree

A straightforward solution to the scalability problem is to leverage the depth of the monitoring tree. It is desirable to make each node's processing load constant,

independent of the total number of clusters being monitored. We begin to address this goal by extending the monitor's data language to make the tree structure explicit.

A GRID tag is added to the Ganglia XML language, where a grid is defined as a collection of clusters and other grids. When queried, a gmeta defines its subtree with grid tags for later use by its upstream parents. Nodes use this new information to determine which clusters are *local* and which are *remote*. We define local clusters as leaf nodes attached directly to the node; their output comes directly from gmon. Remote clusters are instead local to another gmeta. The key observation is that another monitor is the *authority* for the source, and therefore we only need to give it secondary interest.

Gmeta only keeps numerical summaries of data from clusters it is not an authority on. This simple hint conserves resources and reduces response latency within the monitoring tree.

A cluster or grid summary looks exactly like the data for a single host except each metric value represents an additive reduction. This reduction is performed across a known set of nodes, and the summary explicitly records the set size. In this way a summary contains enough information to determine a metric's sum and mean. This definition has shown to work well in practice, although statistics such as standard deviation and median are not supported.

While nodes report local cluster data at full detail, they provide only lower-resolution summaries for grids of remote clusters. By summarizing remote cluster data, we dramatically reduce the amount of information sent along edges of the monitoring tree, as well as the local state required for archiving.

If we let *m* be the amount of monitoring data for a single host, the upper bound on the amount of information any node sends upstream in the tree is O(m). The savings afforded by the new method are additive; in the previous design the root node received *all* monitoring data, O(CHm), where CHm is the total number of metrics monitored. A drawback of both designs is that only numeric metrics can be reliably summarized. Non-numeric metrics are only visible in the highest-resolution cluster views.

Each gmeta includes a URL pointer to itself when queried. Upstream nodes incorporate these authority pointers with their summary state. Each coarse summary report includes the URL that hosts a higher resolution view. By following these pointers, we can locate the leaf node that possesses a cluster's data at its highest resolution. This pointer-based distributed tree forms the heart of our design. In the next section we present a strategy to reduce the processing load of the viewer.

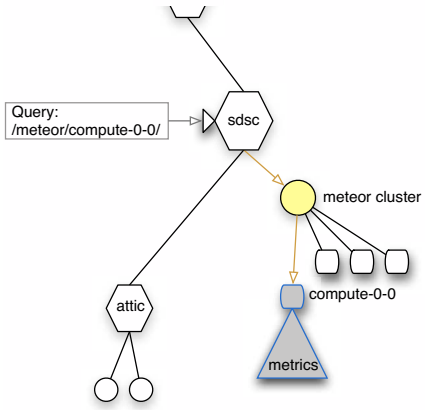

Figure 4: Query processing. The SDSC gmeta is queried for the only the metrics of node compute-0-0 in the meteor cluster.

2.3. N-level monitoring tree with efficient queries

The most common method of viewing the monitor tree is with Ganglia's web frontend[1]. This and other viewers request raw XML from a gmeta agent and parse it for display. The processing required to view the tree is therefore proportional to the size of the XML returned by the monitor. A problem with past gmeta designs was overwhelming the web frontend with very large XML reports. Moreover, when viewing a specific cluster or host, much of the report is unnecessary. While the N-level tree design ameliorates the first problem with compact cluster summaries, the viewer program often still has to parse unused portions of the XML. The heart of this inefficiency lies in the reporting tactic of the monitors: either the entire tree rooted at a monitoring node is reported, or nothing at all.

We improve the efficiency of viewing applications by introducing a simple query engine to our monitoring system. Instead of returning the entire tree rooted at a node, monitors accept a small path-like query that specifies a single local subtree to report (fig 4). Low-latency query response is a primary goal of our design.

[1] See http://ganglia.sourceforge.net/ demo section for an example.

The query engine is designed to keep pace with Ganglia's data-hungry web frontend. This presentation application performs a Ganglia query in the critical path of each generated web page, making anything but a low-latency query engine inappropriate.

While we considered several XPath implementations for this task [12], they proved to be too heavyweight and inefficient for our purposes. Our intuition was a simpler query facility could achieve the efficiency gains we sought.

2.3.1. Parsing and Summarization

To summarize metrics and support queries on the monitoring data, incoming XML must be parsed. However to give the fastest query response, we do not perform this potentially long task at query time. To insure the most immediate query response in all situations the N-level gmetad summarizes data "in the background", on a separate time scale from query processing.

The summarization time scale involves downloading XML data from clusters and computing the summaries. Gmeta system gathers data from sources at a low frequency polling interval, generally every 15 seconds, independent of any query processing. All failure detection is done at this time scale as well. Gmeta then parses the new data, and organizes both the contents and its summary using in-memory structures.

Queries results are based only on the latest fully-parsed data, making long parsing times relatively insignificant. If a query arrives during parsing, the previous summary will be returned. Gmeta is not fully double-buffered, but has fine-grained locks on its data structures that enable the parser and query engine threads to operating at once. This strategy essentially trades some data freshness for query performance.

2.3.2. Serving Queries

By organizing the parsed monitoring data in a series of hash tables, we can support very low-latency queries. Our approach approximates a DOM design [14] where each XML tag name keys into a hash table, which contains all tags at a given level in the XML tree. At any given node this tree is relatively shallow since only summaries of remote grids are kept. A node must search at most three hash table levels to find the desired subtree: data sources, summaries and cluster nodes, and node metrics (fig 4).

These hash lookups complete in O(1) time, however the time to dump the actual data takes longer. Serving a grid or cluster summary takes O(m) time to complete since summaries are the size of data from a single host. The time to complete a full-resolution cluster query is proportional to the cluster size, and takes O(H) operations. As full-resolution cluster queries are expensive, we also support a cluster-summary query for large clusters.

Local cluster summaries are an optimization for the benefit of the viewing applications. This filter returns a summary report for a single cluster. While not strictly necessary, this filter enables the presenting application to easily switch between a high-level overview and the full resolution view of a cluster. We have found this ability useful when examining very large clusters [13] where the full view can overwhelm the browser's processing capacity and must be used sparingly.

Even this simple query support affords much improved performance for the Ganglia web viewing application, and we believe this result will generalize to other clients such as an alarm-detection and notification application.

3. Experience

In this section we present the experimental setup and results that quantify the performance of our wide-area monitor.

Three experiments show the advantages of our N-level monitoring technique. The first illustrates the computational scalability and efficiency of the N-level monitoring tree versus the older 1-level version with a fixed set of clusters. The second set of results explores monitoring performance for various cluster sizes. The final experiment demonstrates the advantages that XML query support gives to the presenter application.

All experiments employ gmon emulators called *pseudo-gmond* to generate controlled Ganglia XML datasets for the monitoring tree. These agents behave identically to a cluster's gmon daemons, except their metric values are chosen randomly. Their XML output conforms to the Ganglia DTD, and therefore requires the same processing effort by the gmeta system under study. From the gmeta perspective, the only difference between the real and simulated clusters is the time to download the XML tree.

By design, the pseudo-gmon agents have more predictable performance than their real counterparts, allowing the experiments to discount gmon processing and XML transit latencies from clusters. Since we are interested in the performance difference between two

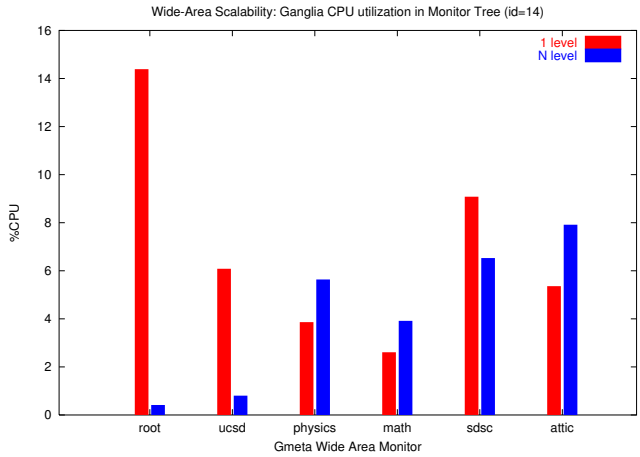

Figure 5: Wide area scalability results. The x-axis shows CPU% used by the each gmeta monitor in the monitoring tree from figure 2. *N-level* is the new gmeta design that reports cluster summaries to its parent.

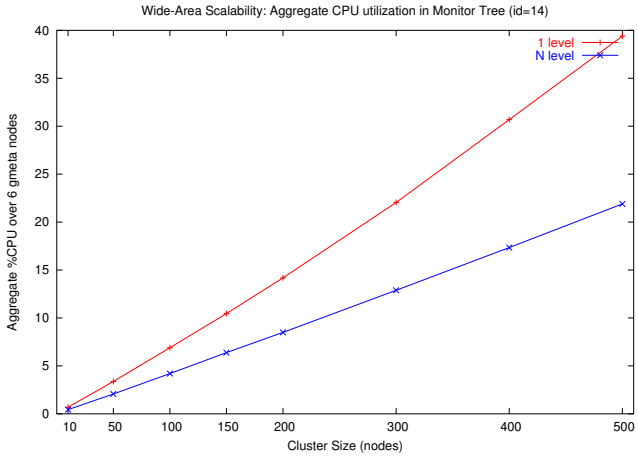

Figure 6: Changing cluster sizes. The monitoring tree is kept unchanged, while the size of the 12 monitored clusters increases. The y-axis is the sum of the CPU utilization across all gmeta nodes.

gmetad designs, pseudo-gmonds allow us to remove unnecessary experimental variables without affecting the validity of the results.

3.1. Experimental Setup

We conduct all experiments on a 10-node Linux cluster named *Alpha*. The cluster has ten homogenous nodes, each with dual 2.2Ghz Pentium 4 processors, 1GB RAM, and Gigabit Ethernet networking. Alpha runs the Rocks clustering software version 2.3.2, which is based on RedHat Linux 7.3 with kernel version 2.4.18-27.7.xsmp. The monitoring daemons were compiled with gcc 2.96, and glibc 2.2.5.

The experiments compare the *1-level* gmeta with the *N-level* version. The 1-level gmeta is defined as Ganglia monitor-core version 2.5.1, while the N-level code is based version 2.5.4, currently in beta testing phase.

Both versions operate in the six-node monitoring tree as shown in figure 2. The twelve clusters in the tree are simulated with pseudo-gmons. Percentage CPU measurements are taken for each gmeta machine, which are otherwise unloaded. The standard `ps` command with an extended time format is used to obtain CPU timings[2]. Gmeta nodes store metric archive databases on a RAM-backed `tmpfs` file system to eliminate their disk I/O requirements.

[2] The `getrusage()` system call has trouble taking accurate measurements of the multi-threaded gmetad daemons.

In order to obtain reliable measurements in the face of system interrupts and other disruptions, we calculate CPU usage percentages over a 60-minute timing window. By employing a large measurement period, small system interruptions have only a negligible effect on data points. However, since we emphasize relative timings rather than absolute ones, a consistent measurement strategy is more critical than the specific collection method used.

The third experiment measures the performance of Ganglia's web frontend. Mod_PHP version 4.1.2 with its native SAX XML parser runs this PHP application, whose pages are served by the Apache webserver version 1.3.27. The web frontend is run on an unloaded machine, separate from the gmeta node. Timings in the are taken with `gettimeofday()` calls inserted just before the socket connection to the gmeta agent and after the completion of the XML parsing.

3.2. Results

To determine scaling benefits of the N-level monitor over the 1-level design, we measure the CPU utilization of every gmeta node in the monitoring tree from figure 2. In this experiment, each of the twelve monitored clusters has 100 hosts. Figure 5 shows the results of this experiment. Bars in the graph are grouped by gmeta monitor, and bar heights represent the percentage of wall-clock CPU time used by the gmeta daemons over a one-hour period.

In the second experiment we vary the size of the twelve monitored clusters. Figure 6 shows the aggregate

percentage CPU utilization over all gmeta nodes in the tree. To find each point, we collect the percent CPU used by each gmeta as in the previous experiment, sum them together and divide by 6, the number of gmeta nodes in the tree. The data point at cluster size 100 represents the sum of all bars in the first plot (fig 5). All monitored clusters have the same number of hosts, and care was taken to ensure the gmon cluster simulators had similar query latencies for all sizes.

The final experiment measures the benefit of the subtree query engine in the N-level design. Timings are taken from the perspective of Ganglia's web viewing application and appear in table 1. Each value represents the time needed by the viewer to download and parse the XML from a gmeta agent in the monitoring tree. The viewer presents the tree in three central ways. The *meta* view summarizes all monitored clusters. The *cluster* view describes one cluster at full-resolution, and the *host* view shows all information known about a single host. We calculate the speedup row in table 1 as the *1-level time* / *N-level time* for each view.

We point the viewer at the *sdsc* gmeta node for this test where the clusters have 100 hosts each, similar to the first experiment. All pages are generated from a single gmeta query, and each value in table 1 is the average of five samples.

3.3. Discussion

The first experiment examines the scalability of the N-level design relative to the 1-level monitor. In a scalable design, we expect to see load transferred from the root node in the monitoring tree downwards towards the leafs, and a general evenness of work throughout the tree. Such behavior would indicate the hierarchy is effectively distributing load, and the system is scalable.

The results show the 1-level design concentrates load at the root of the monitoring tree (the *root* and *ucsd* monitors) as expected. The N-level monitor pushes this computation towards the leaf nodes. Leaf gmeta daemons pay a summarization penalty in the new design, as seen by the higher bars in figure 5. We argue that the work required to process raw cluster data is acceptable when done at the lowest tree level. Non-leaf monitors clearly benefit from this summary processing; their load is drastically reduced compared to their 1-level counterparts. The transfer of processing load suggests the N-level monitor is indeed more scalable than the 1-level design.

The second experiment illuminates how a monitoring tree responds to different cluster loads. The N-level design scales linearly with a low slope when we increase the size of the monitored clusters. The 1-level version exhibits a higher-sloped scaling behavior that appears linear, but actually has a slight upward curve. This non-linear behavior is caused by overloading the root node and duplication of metric archives.

	Meta	*Cluster*	*Host*
1-level	2.091	2.093	2.096
N-level	.0092	.198	.003
Speedup	227	10.5	698

Table 1: Time (in sec) for the web frontend to query and parse Ganglia XML from the *sdsc* gmeta node. The columns represent various web views. The rows are experimental runs.

The 1-level design is limited by the processing ability of the root node. As it gets saturated with load, the root gmeta begins to slow down. Threads must wait in run queues as spare cycles become scarce, and the percent CPU utilization becomes non-linear with respect to smaller runs. Moreover, adding nodes to the monitoring tree will not alleviate this problem, as shown in the first experiment. The N-level design distributes load more evenly across the tree as shown in the first experiment. By not saturating its root node, the N-level monitor maintains a linear scaling in this experiment.

The results of this experiment show effects other than the load transfer between nodes in the tree. In all data points the aggregate CPU usage is less for the N-level monitor. This result is due to redundancy in the system, specifically superfluous metric archives (fig. 3). Nodes in the N-level monitoring tree keep only summary archives of descendants rather than full duplicates, yielding a near-linear increase in archive state, and lowering the total amount of work performed by the system.

The third experiment focuses on the new query support in our design. While a 1-level viewing application must receive a full tree from its gmeta agent, the N-level viewer can request a particular XML sub-tree. The host view in particular benefits from this facility. The 1-level viewer must parse and discard much of the data it receives. For example if we choose to view a single host of cluster A, the viewer must parse and discard data about all other hosts in the cluster. As a result we

see that the N-level delivers a large performance gain in this view by reducing the parsing load of the presenter.

The web frontend used in the 1-level design generates its own summaries for the meta view, while the N-level viewer obtains its summaries directly from the gmeta daemon. Correspondingly, table 1 shows a large gain for the N-level meta view. The parsing load of the full-resolution cluster view is similar for the two monitor designs. We note that no distinction is made between the time to download the XML and the time needed to parse it. However the relatively small XML trees involved (<1MB in all cases) and the dedicated high speed Gigabit Ethernet connections used suggest that the downloading time is dominated by TCP startup and does not vary significantly with the size of the XML.

All code used in these experiments is available upon request.

4. Limitations and Future Work

A major limitation to our wide-area monitor design is the difficulty of constructing the tree hierarchy. While the local gmon system needs no a priori knowledge of cluster nodes, the gmeta design lacks this advantage. We would like to incorporate a wide-area trust model similar to MDS, where parents have no explicit knowledge of their children. Children in an MDS tree periodically send *join* messages to their parents, who verify trust via a cryptographic certificate sent with the message. Nodes are automatically pruned from the tree if their join messages cease.

The MDS design has a self-organizing structure that makes it easier to deploy and maintain, and its soft-state techniques mirror Ganglia's local-area gmon monitor. This extension of gmeta remains as future work.

While our wide-area monitor provides efficient access to monitored data, it has no mechanism to process it at a pragmatic level. We would like to implement a general alarm mechanism that tracks the data and automatically identify situations that should be relayed to a human observer. This feature will become increasingly important as the size of the monitor tree grows. Such an alarm system may require a more detailed query mechanism than we currently provide. A richer query language based on regular expressions is planned for next version of Ganglia.

The way we currently employ the metric archiving tools is not scalable with the number of numeric metrics gathered per host. Although the RRD time-series database system is efficient, Gmeta's use of them leads to a performance bottleneck. Specifically, our archiving technique makes too many updates to the file-based databases, causing unnecessary disk I/O. We believe in future designs gmeta can manipulate its RRD databases in a more efficient manner.

5. Conclusion

In this paper we presented techniques for monitoring multiple clusters over wide-area networks, and have conducted experiments to quantify our performance claims. We have introduced three straightforward strategies that lead to significant increases in the performance of our wide-area monitor. Our N-level design organizes clusters and monitors them in a distributed hierarchical tree structure. This hierarchy provides a scalable backbone for the system. Each node in the tree shares the processing load by providing distilled summaries of monitoring data to its parent. The experimental results show these summaries not only reduce load on the system, but also enable a useful multiple-resolution view of the tree. Finally, we have demonstrated that by supporting a simple query language, the efficiency of presentation applications can be much improved.

Experience with computational clusters shows that effective monitoring is an essential utility for day-to-day operation. The summarization and query techniques of Ganglia's wide-area gmeta monitor open the door for large monitoring grids that can hopefully enable a more cooperative and reliable computing environment in our field.

Acknowledgements

This paper has benefited from conversations, critiques, and code from many people. In particular we would like to thank Eric Fraiser, Phil Papadopoulos, and Greg Bruno for their insight and support, and Jason Smith, Steven Wagner, Martin Knobloch, and Phillip Radden for their code contributions and comments.

References

1. M. Massie, B. Chun, D. Culler. *The Ganglia Distributed Monitoring System: Design, Implementation, and Experience*. Pending publication, 2003.
2. M. Sottile, R. Minnich. Supermon: *A high-speed cluster monitoring system*. In Proceedings of IEEE Cluster Computing, Chicago IL, Sept 2002.
3. NCSA Cluemon. http://clumon.ncsa.uiuc.edu/doc-info.html

4. SGI Performance Co-Pilot product information. http://www.sgi.com/software/co-pilot/

5. The TeraGrid Project. http://www.teragrid.org/, 2001.

6. J. Halpern and Y. Moses, *Knowledge and common knowledge in a distributed environment*. Proceedings of the 3rd Annual Symposium on Principles of Distributed Computing, pp. 50#61, 1984.

7. J. Jiao, S. Naqvi, D. Raz, and B. Sugla. *Towards Efficient Monitoring*. IEEE Journal on Selected Areas in Communications, 18(5), April 2000.

8. M. Dilman and D. Raz. *Efficient reactive monitoring*. In proceedings of IEEE INFOCOM, Alaska, April 2001.

9. Ganglia VO / WorldGrid Installation Guide. http://people.cs.uchicago.edu/~cldumitr/soft/moni/docs/

10. R. Wolski, N. Spring, and J. Hayes. *The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing*. Journal of Future Generation Computing Systems, 15(5-6):757--768, October 1999.

11. The RRDtool time-series data archiving system. http://people.ee.ethz.ch/~oetiker/webtools/rrdtool/

12. World Wide Web Consortium. *XML Path Language (XPath). Version 1.0*. W3C Recommendation. November 16, 1999. http://www.w3.org/TR/xpath

13. The 2000-Node Dell Cluster at the Center for Computational Research, University of Buffalo. http://www.ccr.buffalo.edu

14. A. Le Hors and P. Le Hegaret and G. Nicol and J. Robie and M. Champion and S. Byrne. *Document Object Model (DOM) Level 2 Core Specification Version 1.0*. W3C Recommendation, 2000.

15. S. Raman, S. McCanne. *A Model, Analysis, and Protocol Framework for Soft State-based Communication*. Proceedings of ACM SIGCOMM, 1999.

16. M. Katz, P. Papadopoulos, G. Bruno. *Leveraging Standard Core Technologies to Programmatically Build Linux Cluster Appliances*. CLUSTER 2002: IEEE International Conference on Cluster Computing. April 2002.

17. The Rocks Cluster Register. http://www.rocksclusters.org/rocks-register/

18. S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke. *A Directory Service for Configuring High-Performance Distributed Computations*. volume Proc. 6th IEEE Symp. on High-Performance Distributed Computing, pages 365--375, 1997.

19. The Globus Project. http://www.globus.org/

20. S. Agarwala, C. Poellabauer, J. Kong, K. Schwan, and M. Wolf. *Resource-Aware Stream Management with the Customizable dproc Distributed Monitoring Mechanisms*. Proc 12th IEEE International Symp. on High Performance Distributed Computing (HPDC-12), Seattle, Washington, June 2003.