Available online at www.sciencedirect.com
SCIENCE DIRECT®
Parallel Computing 30 (2004) 817–840
PARALLEL COMPUTING
www.elsevier.com/locate/parco

The ganglia distributed monitoring system: design, implementation, and experience

Matthew L. Massie [a,1], Brent N. Chun [b,*], David E. Culler [a,2]

[a] University of California, Berkeley, Computer Science Division, Berkerley, CA 94720-1776, USA
[b] Intel Research Berkeley, 2150 Shattuck Ave. Suite 1300, Berkerley, CA 94704, USA

Received 11 February 2003; received in revised form 21 April 2004; accepted 28 April 2004
Available online 15 June 2004

Abstract

Ganglia is a scalable distributed monitoring system for high performance computing systems such as clusters and Grids. It is based on a hierarchical design targeted at federations of clusters. It 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. It leverages widely used technologies such as XML for data representation, XDR for compact, portable data transport, and RRDtool for data storage and visualization. It uses carefully engineered data structures and algorithms to achieve very low per-node overheads and high concurrency. The implementation is robust, has been ported to an extensive set of operating systems and processor architectures, and is currently in use on over 500 clusters around the world. This paper presents the design, implementation, and evaluation of Ganglia along with experience gained through real world deployments on systems of widely varying scale, configurations, and target application domains over the last two and a half years.

Keywords: Monitoring; Clusters; Distributed systems

* Corresponding author. Tel.: +1-510-495-3075; fax: +1-510-495-3049.
E-mail addresses: massie@cs.berkeley.edu (M.L. Massie), bnc@intel-research.net (B.N. Chun), culler@cs.berkeley.edu (D.E. Culler).
[1] Tel.: +1-510-643-7450; fax: +1-510-643-7352.
[2] Tel.: +1-510-643-7572; fax: +1-510-643-7352.

0167-8191/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.parco.2004.04.001

1. Introduction

Over the last ten years, there has been an enormous shift in high performance computing from systems composed of small numbers of computationally massive devices [11,12,18,19] to systems composed of large numbers of commodity components [3,4,6,7,9]. This architectural shift from the few to the many is causing designers of high performance systems to revisit numerous design issues and assumptions pertaining to scale, reliability, heterogeneity, manageability, and system evolution over time. With clusters now the de facto building block for high performance systems, scale and reliability have become key issues as many independently failing and unreliable components need to be continuously accounted for and managed over time. Heterogeneity, previously a non-issue when running a single vector supercomputer or an MPP, must now be designed for from the beginning, since systems that grow over time are unlikely to scale with the same hardware and software base. Manageability also becomes of paramount importance, since clusters today commonly consist of hundreds or even thousands of nodes [6,7]. Finally, as systems evolve to accommodate growth, system configurations inevitably need to adapt. In summary, high performance systems today have sharply diverged from the monolithic machines of the past and now face the same set of challenges as that of large-scale distributed systems.

One of the key challenges faced by high performance distributed systems is scalable monitoring of system state. Given a large enough collection of nodes and the associated computational, I/O, and network demands placed on them by applications, failures in large-scale systems become commonplace. To deal with node attrition and to maintain the health of the system, monitoring software must be able to quickly identify failures so that they can be repaired either automatically or via out-of-band means (e.g. rebooting). In large-scale systems, interactions amongst the myriad computational nodes, network switches and links, and storage devices can be complex. A monitoring system that captures some subset of these interactions and visualizes them in interesting ways can often lead to an increased understanding of a system's macroscopic behavior. Finally, as systems scale up and become increasingly distributed, bottlenecks are likely to arise in various locations in the system. A good monitoring system can assist here as well by providing a global view of the system, which can be helpful in identifying performance problems and, ultimately, assisting in capacity planning.

Ganglia is a scalable distributed monitoring system that was built to address these challenges. It provides scalable monitoring of distributed systems at various points in the architectural design space including large-scale clusters in a machine room, computational Grids [14,15] consisting of federations of clusters, and, most recently, has even seen application on an open, shared planetary-scale application testbed called PlanetLab [21]. The system is based on a hierarchical design targeted at federations of clusters. It relies on a multicast-based listen/announce protocol [1,10,16,29] 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. It leverages widely used technologies such as XML for data representation, XDR for compact, portable data transport, and RRDtool for data storage and visualization. It uses carefully engineered data structures and algorithms to achieve very low per-node

overheads and high concurrency. The implementation is robust, has been ported to an extensive set of operating systems and processor architectures, and is currently in use on over 500 clusters around the world.

This paper presents the design, implementation, and evaluation of the Ganglia distributed monitoring system along with an account of experience gained through real world deployments on systems of widely varying scale, configurations, and target application domains. It is organized as follows. In Section 2, we describe the key challenges in building a distributed monitoring system and how they relate to different points in the system architecture space. In Section 3, we present the architecture of Ganglia, a scalable distributed monitoring system for high performance computing systems. In Section 4, we describe our current implementation of Ganglia which is currently deployed on over 500 clusters around the world. In Section 5, we present a performance analysis of our implementation along with an account of experience gained through real world deployments of Ganglia on several large-scale distributed systems. In Section 6, we present related work and in Section 7, we conclude the paper.

2. Distributed monitoring

In this section, we summarize the key design challenges faced in designing a distributed monitoring system. We then discuss key characteristics of three classes of distributed systems where Ganglia is currently in use: clusters, Grids, and planetary-scale systems. Each class of systems presents a different set of constraints and requires making different design decisions and trade-offs in addressing our key design challenges. While Ganglia's initial design focus was scalable monitoring on a single cluster, it has since naturally evolved to support other classes of distributed systems as well. Its use on computational Grids and its recent integration with the Globus metadirectory service (MDS) [13] is a good example of this. Its application on PlanetLab is another, one which has also resulted in a reexamination of some of Ganglia's original design decisions.

2.1. Design challenges

Traditionally, high performance computing has focused on scalability as the primary design challenge. The architectural shift towards increasingly distributed and loosely coupled systems, however, has raised an additional set of challenges. These new challenges arise as a result of several factors: increased physical distribution, long running distributed services, and scaling and evolution of systems over time. Increased physical distribution implies multiple, independently failing and unreliable components. This, in turn, requires designing applications whose management overheads scale slowly with the number of nodes. Long running distributed services imply the need to be highly available to clients of the service. This, in turn requires applications to be robust to a variety of different types of failures. Finally, the scaling and evolution of systems over time implies that hardware and software will change. This, in turn, requires addressing issues of extensibility and portability.

The key design challenges for distributed monitoring systems thus include:

- *Scalability*: The system should scale gracefully with the number of nodes in the system. Clusters today, for example, commonly consist of hundreds or even thousands of nodes. Grid computing efforts, such as TeraGrid [22], will eventually push these numbers out even further.
- *Robustness*: The system should be robust to node and network failures of various types. As systems scale in the number of nodes, failures become both inevitable and commonplace. The system should localize such failures so that the system continues to operate and delivers useful service in the presence of failures.
- *Extensibility*: The system should be extensible in the types of data that are monitored and the nature in which such data is collected. It is impossible to know a priori everything that ever might want to be monitored. The system should allow new data to be collected and monitored in a convenient fashion.
- *Manageability*: The system should incur management overheads that scale slowly with the number of nodes. For example, managing the system should not require a linear increase in system administrator time as the number of nodes in the system increases. Manual configuration should also be avoided as much as possible.
- *Portability*: The system should be portable to a variety of operating systems and CPU architectures. Despite the recent trend towards Linux on x86, there is still wide variation in hardware and software used for high performance computing. Systems such as Globus [14] further facilitate use of such heterogeneous systems.
- *Overhead*: The system should incur low per-node overheads for all scarce computational resources including CPU, memory, I/O, and network bandwidth. For high performance systems, this is particularly important since applications often have enormous resource demands.

2.2. Distributed systems

There are currently three classes of distributed systems where Ganglia is being used: clusters, Grids, and planetary-scale systems. Each class of systems presents a different set of constraints and requires making different design decisions and trade-offs in addressing our key design challenges. The constraints revolve primarily around how these systems are physically organized and distributed and what types of resources are scarce and/or expensive to use. Design decisions and trade-offs then involve how to address our key design challenges in light of these constraints.

As an example, Ganglia currently uses a multicast-based listen/announce protocol to monitor state within a single cluster. This approach offers several advantages including automatic discovery of nodes as they are added and removed, no manual configuration of cluster membership lists or topologies, and symmetry in that any node knows the entire state of the cluster. However, it also assumes the presence of a native multicast capability, an assumption which does not hold for the Internet

in general and thus cannot be relied on for distributed systems (e.g. Grids) that require wide-area communication.

The following summarizes the three classes of distributed systems Ganglia is currently deployed on:

- *Clusters*: Clusters are characterized by a set of nodes that communicate over a high bandwidth, low latency interconnect such as Myrinet [5] or Gigabit Ethernet. In these systems, nodes are frequently homogeneous in both hardware and operating system, the network rarely partitions, and, almost universally, the system is managed by a single administrative entity.
- *Grids*: Grids can be characterized as a set of heterogeneous systems federated over a wide-area network. In contrast to the general Internet, such systems are usually interconnected using special high speed, wide-area networks (e.g. Abilene, TeraGrid's DTF network) in order to get the bandwidth required for their applications. These systems also frequently involve distributed management by multiple administrative entities.
- *Planetary-scale systems*: Planetary-scale systems can be characterized as wide-area distributed systems whose geographical extent covers a good fraction of the planet. These systems are built as overlay networks on top of the existing Internet. A few implications of this are (i) wide-area bandwidth is not nearly as abundant compared to clusters or Grids (ii) network bandwidth is not cheap, and (iii) the network experiences congestion and partitions much more frequently than in either the cluster or Grid case.

3. Architecture

Ganglia is based on a hierarchical design targeted at federations of clusters (Fig. 1). It relies on a multicast-based listen/announce protocol [1,10,16,29] 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. Within each cluster, Ganglia uses heartbeat messages on a well-known multicast address as the basis for a membership protocol. Membership is maintained by using the reception of a heartbeat as a sign that a node is available and the non-reception of a heartbeat over a small multiple of a periodic announcement interval as a sign that a node is unavailable.

Each node monitors its local resources and sends multicast packets containing monitoring data on a well-known multicast address whenever significant updates occur. Applications may also send on the same multicast address in order to monitor their own application-specific metrics. Ganglia distinguishes between built-in metrics and application-specific metrics through a field in the multicast monitoring packets being sent. All nodes listen for both types of metrics on the well-known multicast address and collect and maintain monitoring data for all other nodes. Thus, all nodes always have an approximate view of the entire cluster's state and this state is easily reconstructed after a crash.

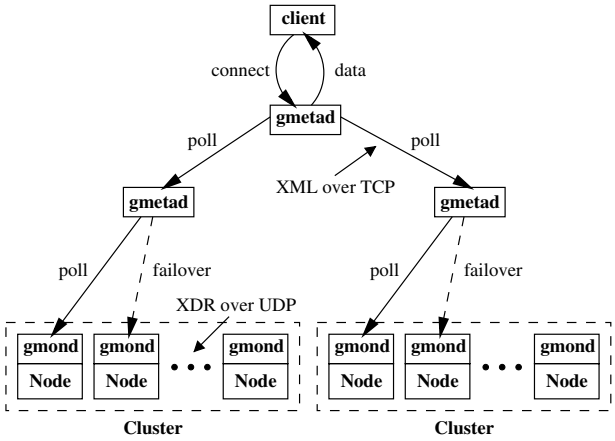

Fig. 1. Ganglia architecture.

Ganglia federates multiple clusters together using a tree of point-to-point connections. Each leaf node specifies a node in a specific cluster being federated, while nodes higher up in the tree specify aggregation points. Since each cluster node contains a complete copy of its cluster's monitoring data, each leaf node logically represents a distinct cluster while each non-leaf node logically represents a set of clusters. (We specify multiple cluster nodes for each leaf to handle failures.) 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 the same mechanism, namely a TCP connection to the node being polled followed by a read of all its monitoring data.

4. Implementation

The implementation consists of two daemons, `gmond` and `gmetad`, a command-line program `gmetric`, and a client side library. The 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. `gmond` runs on every node of a cluster. The Ganglia Meta Daemon (`gmetad`), on the other hand, provides federation of multiple clusters. A tree of TCP connections between multiple `gmetad` daemons allows monitoring information for multiple clusters to be aggregated. Finally, `gmetric` is a 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.

4.1. Monitoring on a single cluster

Monitoring on a single cluster is implemented by the Ganglia monitoring daemon (`gmond`). `gmond` is organized as a collection of threads, each assigned a specific task.

The *collect and publish thread* is responsible for collecting local node information, publishing it on a well-known multicast channel, and sending periodic heartbeats. The *listening threads* are responsible for listening on the multicast channel for monitoring data from other nodes and updating `gmond`'s in-memory storage, a hierarchical hash table of monitoring metrics. Finally, a thread pool of *XML export threads* are dedicated to accepting and processing client requests for monitoring data. All data stored by `gmond` is soft state and nothing is ever written to disk. This, combined with all nodes multicasting their state, means that a new `gmond` comes into existence simply by listening and announcing.

For speed and low overhead, `gmond` uses efficient data structures designed for speed and high concurrency. All monitoring data collected by `gmond` daemons is stored in a hierarchical hash table that uses reader-writer locking for fine-grained locking and high concurrency. This concurrency allows the listening threads to simultaneously store incoming data from multiple unique hosts. It also helps resolve competition between the *listening threads* and the *XML export threads* (see Fig. 2) for access to host metric records. Monitoring data is received in XDR format and saved in binary form to reduce physical memory usage. In a typical configuration, the number of incoming messages processed on the multicast channel far outweigh the number of requests from clients for XML. Storing the data in a form that is "closer" to the multicast XDR format allows for more rapid processing of the incoming data.

4.1.1. Multicast listen/announce protocol

`gmond` uses a multicast-based, listen/announce protocol to monitor state within a single cluster. This approach has been used with great success in previous cluster-based systems [1,10,16,29]. Its main advantages include: automatic discovery of nodes as they are added and removed, no manual configuration of cluster membership lists or topologies, amenability to building systems based entirely

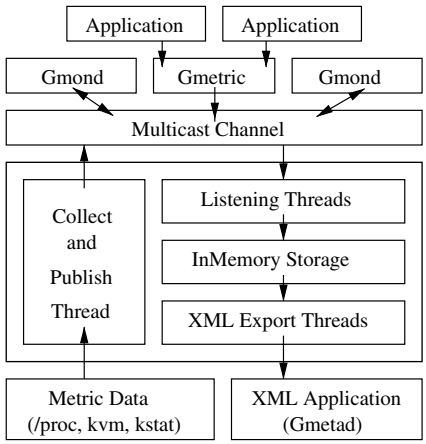

Fig. 2. Ganglia implementation.

on soft-state, and symmetry in that any node knows the entire state of the cluster. Automatic discovery of nodes and eliminating manual configuration is important because it allows gmond on all the nodes to be self-configuring, thereby reducing management overhead. Amenability to a soft-state based approach is important because this allows nodes to crash and restart without consequence to gmond. Finally, because all nodes contain the entire state of the cluster, any node can be polled to obtain the entire cluster's state. This is important as it provides redundancy, which is especially important given the frequency of failures in a large distributed system.

4.1.2. Publishing monitoring data

gmond publishes two types of metrics, built-in metrics which capture node state and user-defined metrics which capture arbitrary application-specific state, on a well-known multicast address. For built-in metrics, gmond currently collects and publishes between 28 and 37 different metrics depending on the operating system and CPU architecture it is running on. Some of the base metrics include the number of CPUs, CPU clock speed, %CPU (user, nice, system, idle), load (1, 5, and 15 min averages), memory (free, shared, buffered, cached, total), processes (running, total), swap (free, total), system boot time, system clock, operating system (name, version, architecture), and MTU. User-defined metrics, on the other hand, may represent arbitrary state. gmond distinguishes between built-in metrics and user-defined metrics based on a field in the multicast packets being sent.

All metrics published on the multicast channel are in XDR format for portability and efficiency. Built-in metrics are collected in a portable manner through well-defined interfaces (e.g. /proc, kvm, and kstat). Built-in metrics are sent on the multicast channel in an efficient manner by leveraging a static metric lookup table that contains all the static characteristics of each metric so that only a unique key and metric value needs to be sent per announcement. Built-in messages are either 8 or 12 bytes in length (4 bytes for the key and 4–8 bytes for the value). The metric key is always sent as an xdr_u_int while the metric value type depends on the specific metric being sent. User-defined metrics, on the other hand, have a less efficient XDR format because every metric characteristic must be explicitly defined. Such metrics can be published by arbitrary applications through use of the gmetric command-line program.

Tables 1 and 2 show subsets of the metric lookup table. In Table 1, we show representative metrics with their corresponding XDR key number, metric type, and value format while in Table 2, we show details of each built-in metric's collect/announce schedule, metric value thresholds, metric units and binary to text conversion details. These attributes for each metric determine how often a metric gets published on the multicast channel. The default values for the built-in metrics represent a trade-off between gmond resource use and metric time-series granularity for a 128-node cluster, our initial design point. These values can be modified at compile time to accommodate different environments.

The collection and value thresholds in the metric lookup table aim at reducing resource usage by collecting local node data and sending multicast traffic only when

Table 1
Example metrics defined in the gmond metric lookup table

Key (xdr_u_int)	Metric	Value format
0	User-defined	Explicit
1	cpu_num	xdr_u_short
2	cpu_speed	xdr_u_int
3	mem_total	xdr_u_int
4	swap_total	xdr_u_int
...	...	...
15	load_one	xdr_float
16	load_five	xdr_float
17	load_fifteen	xdr_float
...	...	...

Table 2
Example metric collection schedules with value and time thresholds defined in the internal gmond metric lookup table

Metric	Collected (s)	Val thresh	Time thresh (s)
User-defined	Explicit	Explicit	Explicit
cpu_num	Once	None	900–1200
cpu_speed	Once	None	900–1200
mem_total	Once	None	900–1200
swap_total	Once	None	900–1200
load_one	15–20	1	50–70
load_five	30–40	1	275–325
load_fifteen	60–80	1	850–950

significant updates occur. The collected attribute specifies how often a metric is collected. Larger values avoid collecting constant (e.g. number of CPUs) or slowly changing metrics. Value thresholds specify how much a metric needs to have changed from its value when it was last collected in order to be deemed significant. Only significant changes are sent on the multicast channel.

4.1.3. Timeouts and heartbeats

Time thresholds specify an upper bound on the interval when metrics are sent. Metrics are sent on the multicast channel over bounded, random intervals to reduce conflicts with competing applications and to avoid synchronization between gmond peers. Time thresholds allow applications to ascertain message loss on the multicast channel and determine the accuracy of metric values.

To reclaim storage for old metrics, gmond expires monitoring data using timeouts. For each monitoring metric, it uses two time limits, a soft limit ($T_{\max}$) and a hard limit ($D_{\max}$). Each incoming metric is timestamped at arrival with time T_0. The number of seconds elapsed since T_0 is denoted T_n. gmond performs no action when the soft limit is reached. gmond simply reports T_n and $T_{\max}$ to clients via

XML attributes. If $T_n > T_{max}$, then clients are immediately aware that a multicast message was not delivered and the value may be inaccurate. Exceeding a hard limit, on the other hand, results in the monitoring data being permanently removed from gmond's hierarchical hash table of metric data. While non-static, built-in metrics are constantly being sent on the multicast channel, application-specific metrics sent by applications using gmetric may become meaningless over time (e.g. an application simply exits). Timeouts are intended primarily to handle these types of cases.

To time out nodes that have died, gmond uses explicit heartbeat messages with time thresholds. Each heartbeat contains a timestamp representing the startup time of the gmond instance. Any gmond with an altered timestamp is immediately recognized by its peers as having been restarted. A gmond which has not responded over some number of time thresholds is assumed to be down. Empirically, we have determined that four thresholds works well as a practical balance between quick false positives and delayed determination of actual downtime. In response to new or restarted hosts, all local metric time thresholds are reset. This causes all metrics to be published the next time they are collected regardless of their value and ensures new and restarted hosts are quickly populated with the latest cluster state information. Without this reset mechanism, rarely published metrics would not be known to the new/restarted host for an unacceptably long period of time. It is important to note that the time-threshold reset mechanism only occurs if a gmond is more than 10 min old. This prevents huge multicast storms that could develop if every gmond on a cluster is restarted simultaneously. Future implementations will likely have new members directly bootstrap to the eldest gmond in a multicast group.

4.2. Federation

Federation in Ganglia is achieved using a tree of point-to-point connections amongst representative cluster nodes to aggregate the state of multiple clusters. At each node in the tree, a Ganglia Meta Daemon (gmetad) periodically polls a collection of child data sources, parses the collected XML, saves all numeric, volatile metrics to round-robin databases (Section 4.3) and exports the aggregated XML over a TCP sockets to clients (Fig. 1). Data sources may be either gmond daemons, representing specific clusters, or other gmetad daemons, representing sets of clusters. Data sources use source IP addresses for access control and can be specified using multiple IP addresses for failover. The latter capability is natural for aggregating data from clusters since each gmond daemon contains the entire state of its cluster.

Data collection in gmetad is done by periodically polling a collection of child data sources which are specified in a configuration file. Each data source is identified using a unique tag and has multiple IP address/TCP port pairs associated with it, each of which is equally capable of providing data for the given data source. We used configuration files for specifying the structure of the federation tree for simplicity and since computational Grids, while consisting of many nodes, typically consist of only a small number of distinct sites. To collect data from each child data source, Ganglia dedicates a unique data collection thread. Using a unique thread per data

source results in a clean implementation. For a small to moderate number of child nodes, the overheads of having a thread per data source are usually not significant.

Collected data is parsed in an efficient manner to reduce CPU overhead and stored in RRDtool for visualization of historical trends. XML data is parsed using an efficient combination of a SAX XML parser and a GNU gperf-generated perfect hash table. We use a SAX parser, as opposed to a DOM parser, to reduce CPU overhead and to reduce `gmetad`'s physical memory footprint. We use a hash table to avoid large numbers of string comparisons when handling XML parsing events. This hash table was generated by GNU gperf which, given a collection of keys, generates a hash table and a hash function such that there are no collisions. Every possible Ganglia XML element, attribute, and all built-in metric names comprised the set of keys for generating the hash table. The SAX XML callback function uses this perfect hash function instead of raw string comparisons for increased efficiency and speed. As the XML is processed, all numerical values that are volatile are also saved to RRDtool databases.

4.3. Visualization

Ganglia uses RRDtool (Round Robin Database) to store and visualize historical monitoring information for grid, cluster, host, and metric trends over different time granularities ranging from minutes to years (Fig. 3). RRDtool is a popular system for storing and graphing time series data. It uses compact, constant size databases specifically designed for storing and summarizing time series data. For data at different time granularities, RRDtool generates graphs which plot historical trends of

Fig. 3. Ganglia web front-end.

metrics versus time. These graphs are then used by Ganglia and exported to users using a PHP web front-end.

The web front-end uses TemplatePower (`templatepower.codocad.com`) to create a strict separation between content and presentation. This allows website developers to easily customize the look and feel of the website without damaging the underlying content engine. Custom templates can also be created to extend the functionality of the web front-end. For example, the NPACI Rocks Group has created a unique template which provides visualization of cluster PBS queues. Other groups such as WorldGrid have chosen to directly import the `gmetad` XML into their preexisting web infrastructure.

5. Evaluation and experience

In this section, we present a quantitative analysis of Ganglia along with an account of experience gained through real world deployments on production distributed systems. For the analysis, we measure scalability and performance overhead. We use data obtained from four example systems to make this concrete. For experience, we report on key observations and lessons learned while deploying and maintaining Ganglia on several production systems. Specifically, we describe what worked well, what did not work so well, and describe how our experiences have caused us to revisit certain design decisions in order to better support monitoring across a wide range of distributed systems.

5.1. Systems evaluated

We used four production distributed systems (Table 3) to evaluate Ganglia, each representing a different point in the architectural design space and used for different application purposes. The first system is Millennium, a system used for advanced applications in scientific computing, simulation, and modeling. Millennium [24] is a cluster in the UC Berkeley computer science department which consists of approximately 100 SMP nodes, each with either two or four CPUs. Each 2-way SMP consists of two 500 MHz Pentium III CPUs, 512 MB of RAM, two 9 GB disks, and both Gigabit Ethernet and Myrinet connections. Each 4-way SMP consists of four 700 MHz Pentium III CPUs, 2 GB of RAM, two 18 GB disks, and both Gigabit

Table 3
Systems evaluated

System	Number of nodes	Number of clusters
Millennium	100	1
SUNY	2000	1
UCB CS	150	4
PlanetLab	102	42

Ethernet and Myrinet connections. All nodes in Millennium are connected via both a Gigabit Ethernet network and a Myrinet network and run the Linux 2.4.18 SMP kernel.

The second system is SUNY Buffalo's HPC Linux cluster, currently the largest Linux cluster at an educational institution in the United States. This system is used primarily in the acceleration of cancer research, specifically investigation into the human genome, bioinformatics, protein structure prediction, and large-scale computer simulations. The system consists of approximately 2000 dual-processor SMP nodes. Each SMP is either a Dell PowerEdge 1650 or a Dell PowerEdge 2650 server. The majority of the nodes are PowerEdge 1650 servers, each of which contains dual 1.26 GHz Pentium III CPUs. The remaining PowerEdge 2650 nodes each contain higher speed, dual Xeon processors. The system also includes a 14 Terabyte EMC storage area network (SAN) and uses Extreme Networks BlackDiamond switches for Gigabit I/O connectivity between the nodes. All nodes in the SUNY cluster run the Linux 2.4.18 SMP kernel.

The third system is a federation of clusters in the UC Berkeley computer science department. These clusters are used for a variety of purposes including computational science and engineering, global distributed storage systems, and serving web content. The system consists of four clusters, each residing in the same building. The first cluster is the aforementioned 100-node Millennium cluster. The second cluster is a 45-node cluster of 2-way SMPs used by the Oceanstore [20] group. Each node in their cluster is an IBM xSeries 330 consisting of two 1 GHz Pentium III CPUs, 1.5 GB of RAM, and two 36 GB disks. Each of their nodes is connected to a Gigabit Ethernet network and runs the Linux 2.4.18 SMP kernel. The third cluster is an experimental 4-node cluster of 2-way SMPs used as part of the CITRUS [23] project. Each node in the CITRUS cluster consists of a 733 MHz Itanium CPU, 5 GB of RAM, and is connected to a Gigabit Ethernet network. The federation of CITRUS with the rest of the CS clusters using Ganglia is one concrete example of using Ganglia across heterogeneous CPU architectures. Finally, the fourth cluster is a 3-node web server cluster. Each node in that cluster is a 930 MHz Pentium II with 256 MB of RAM and an 8 GB disk.

The fourth system is PlanetLab, an open, shared planetary-scale application testbed [21]. PlanetLab currently consists of 102 nodes distributed across 42 sites spanning three continents: North America, Europe, and Australia. From Ganglia's point of view, each PlanetLab site can be viewed as essentially a small cluster consisting of 2–3 nodes. Each node at a site is either a Dell PowerEdge 1650 or a Dell Precision 340 MiniTower. Each Dell PowerEdge 1650 consists of a 1.26 GHz Pentium III CPU, 1 GB of RAM, two 36 GB Ultra 160 SCSI disks in a RAID 1 configuration, and dual on-board Gigabit Ethernet network interfaces. Each Dell Precision 340 consists of a 1.8 GHz Pentium 4 CPU, 2 GB of RAM, two 120 GB 72K disks, and a Gigabit Ethernet network interface. Local area connectivity within each site is fast (i.e. often Gigabit Ethernet). Wide-area network connectivity, on the other, can vary significantly both in terms of performance and financial costs incurred through bandwidth usage. All nodes in PlanetLab run a kernel based on Linux 2.4.19.

5.2. Overhead and scalability

In order for a distributed monitoring system to become widely used, it must first meet the prerequisites of having low performance overhead and being able to scale to production size systems. To quantify this, we performed a series of experiments on several production distributed systems running Ganglia. For performance overhead, we measured both local overhead incurred within the nodes (e.g. CPU overhead, memory footprint) as well as "global" overhead incurred between the nodes. (The latter is essentially network bandwidth, which we further decompose as being either local-area or wide-area.) For scalability, we measured overhead on individual nodes and quantified how overhead scales with the size of the system, both in terms of number of nodes within a cluster and the number of clusters being federated.

5.2.1. Local overhead

In Table 4, we show local per-node overheads for local monitoring for Millennium, SUNY, and PlanetLab. Data for this table was collected by running the `ps` command multiple times to obtain process information and averaging the results. For Millennium, these numbers represent the per-node overheads for a cluster of 94 SMP nodes. For SUNY, these numbers represent the per-node overheads for a cluster of 2000 SMP nodes. For PlanetLab, these numbers represent the per-node overheads incurred at a typical PlanetLab site. The measurements shown here were taken on a 3-node cluster of Dell PowerEdge 1650 nodes at Intel Research Berkeley. Because all PlanetLab sites currently consist of either two or three nodes and have essentially the same configuration, these numbers should be representative of all 42 PlanetLab sites.

We observe that local per-node overheads for local monitoring on Millennium, SUNY, and PlanetLab are small. Per-node overheads for nodes at a typical PlanetLab site account for less than 0.1% of the CPU, while on SUNY and Millennium, they account for just 0.3% and 0.4% of the CPU, respectively. Virtual memory usage is moderate, 15.6, 16.0, and 15.2 MB for Millennium, SUNY, and PlanetLab respectively. (Much of this VM is thread stack allocations, of which a small fraction is actually used.) Physical memory footprints, on the other hand, are small. On Millennium, `gmond` has a 1.3 MB physical memory footprint corresponding to 0.25% of a node's physical memory capacity. On PlanetLab, `gmond`'s physical memory footprint is even smaller, just 0.9 MB for 0.09% of a PlanetLab node's

Table 4
Local per-node monitoring overheads for `gmond`

System	CPU (%)	PhyMem (MB)	VirMem (MB)
Millennium	0.4	1.3	15.6
SUNY	0.3	16.0	16.7
PlanetLab	<0.1	0.9	15.2

total physical memory. Finally, for SUNY, physical memory usage is observed to be just 16.0 MB per node to store the entire global monitoring state of the 2000-node cluster on each node. Since `gmond` daemons only maintain soft state, no I/O overhead is incurred.

In Table 5, we show local per-node overheads for federation for Millennium, the UCB CS clusters, and PlanetLab. Data for this table was collected by running the `ps` and `vmstat` commands multiple times and averaging the results. For Millennium, these numbers represent the local node overhead incurred by `gmetad` to aggregate data from a single cluster with 94 nodes. For the UCB CS clusters, these numbers represent the local node overhead incurred by `gmetad` to aggregate data from four clusters with 94, 45, 4, and 3 nodes respectively. Each of these clusters was physically located in the same building. Finally, for PlanetLab, these numbers represent the local node overhead incurred by `gmetad` to aggregate data from 42 clusters spread around the world, each with 2–3 nodes each.

The data shows that local per-node overheads for federating data on Millennium, the UCB CS clusters, and PlanetLab have scaling effects mainly in the number of sites. We observe that for a system like PlanetLab with 42 sites, virtual memory usage is scaling with the number of sites. The primary reason for this is Ganglia's use of a thread per site, each of which uses the default 2 MB stack allocated by the Linux pthreads implementation. Physical memory footprints are small, ranging from 1.6 to 2.5 MB. CPU overhead is also relatively small, ranging from less than 0.1% for PlanetLab to about 1.1% for monitoring of four clusters in UC Berkeley computer science department.

I/O overhead and associated context switches and interrupts were observed to be significant in the current implementation of `gmetad`. The primary cause for this I/O activity is not `gmetad`'s aggregation of data per se, but rather its writing of RRD databases to disk to generate visualizations of the monitoring data. For all three systems, we measured average I/O activity ranging from 1.3 to 1.9 MB/s. For Millennium and the UCB CS clusters, this activity tended to be clustered over distinct intervals of time when `gmetad` polls each site (every 15 s by default). For PlanetLab, on the other hand, I/O activity was continuous since polling 42 sites over the wide-area takes varying amounts of time and RRD databases need to be written for 42 sites. On PlanetLab, we observed an increase in average context switches per second from 108 to 713 ctx/s and an increase in interrupts per second from 113 to 540 intr/s compared to not running `gmetad`. The resulting I/O, context switches, and interrupts have resulted in significant slowdowns on the node running `gmetad`, especially for interactive jobs.

Table 5
Local node overhead for aggregation with `gmetad`

System	CPU (%)	PhyMem (MB)	VirMem (MB)	I/O (MB/s)
Millennium	<0.1	1.6	8.8	1.3
UCB CS	1.1	2.5	15.8	1.3
PlanetLab	<0.1	2.4	96.2	1.9

5.2.2. Global overhead

In Table 6, we summarize the amount of network bandwidth consumed by Ganglia for both Millennium and PlanetLab. We decompose the network bandwidth into local-area monitoring bandwidth and wide-area federation bandwidth. The former accounts for the multicast packets sent within a cluster as part of the listen/announce protocol. The latter accounts for the TCP packets used to federate the data from multiple clusters and aggregate the results. Data for local-area, monitoring packets was collected by running `tcpdump` on a single node and by monitoring all multicast traffic sent on Ganglia's multicast group. Data for federated bandwidth was obtained by polling one node per cluster, measuring the number of bits of monitoring data, and dividing the number of bits by Ganglia's default polling interval (15 s) to compute a lower bound on the bandwidth. This number is necessarily a lower bound since it does not account for TCP headers, acknowledgments, and so on. The difference due to those extra packets, however, is not likely to be too significant given that the average cluster has a fair amount of data.

Our measurements show that for both Millennium and PlanetLab, the overall bit rates for monitoring and federating monitoring data are fairly small relative to the speed of modern local area networks. Millennium, for example, uses a Gigabit Ethernet network to connect cluster nodes together and also has at least 100 Mb/s of end-to-end bandwidth from cluster nodes to the node aggregating the cluster data and running `gmetad`. Within each PlanetLab site, nodes are also typically connected via a fast local-area network. On the other hand, sites also have widely varying network connectivity in terms of both network speed and underlying pricing structures based on agreements with their ISPs. Over a week's time, a 272 Kbit/s bit rate implies 19.15 GB of monitoring data is being sent. In a planetary-scale system like PlanetLab, the cost of sending such large amounts of data over the public Internet can be non-trivial, in particular for sites outside the US sending data over transcontinental links.

5.2.3. Scalability

In these experiments, we characterize the scalability of Ganglia as we scale both the number of nodes within a cluster and the number of sites being federated. For measuring scalability within a single cluster, we use the Berkeley Millennium. We selectively disable Ganglia `gmond` daemons to obtain cluster sizes ranging from 1 to 94 nodes and measure performance overheads. For measuring scalability across

Table 6
Network bandwidth consumed for local monitoring and federation

System	Monitoring BW/node (Kbits/s)	Federation BW (Kbits/s)
Millennium	28	210
PlanetLab	6	272

The monitoring bandwidth denotes average per node bandwidth for monitoring in a single cluster (i.e. bandwidth per `gmond`). The federation bandwidth denotes total bandwidth for aggregating data from a set of clusters (i.e. bandwidth for `gmetad`).

federated clusters, we use PlanetLab. We selectively configure `gmetad` to poll data from a subset of the 42 PlanetLab sites ranging from 1 site to all 42 sites and measure performance overheads. In both cases, we also use the size of the monitoring output from each cluster and the default polling rate of `gmetad` to provide a lower bound on the amount of bandwidth used for federation.

In Fig. 4a and b, we quantify the scalability of Ganglia on a single cluster by showing local-area bandwidth consumed as a function of cluster size. As a direct consequence of using native IP multicast, we observe a linear scaling in local-area bandwidth consumed as a function of cluster size. We also observe a linear scaling in packet rates, again due our use of native IP multicast as opposed to point-to-point connections. In both the bandwidth and packet rate cases, we observe small constant factors, which can be at least partially attributed to Ganglia's use of thresholds. At 90 nodes, for example, we measure local-area bandwidth consumed to be just 27 Kbits/s. On a Gigabit Ethernet network, 27 Kbits/s amounts to just 0.0027% of the total network's bandwidth. Packet rates were also observed to be reasonably small at this scale.

In Fig. 5a and b, we plot the performance overheads for federation as a function of number of clusters being federated. Data for Fig. 5a was collected by running the

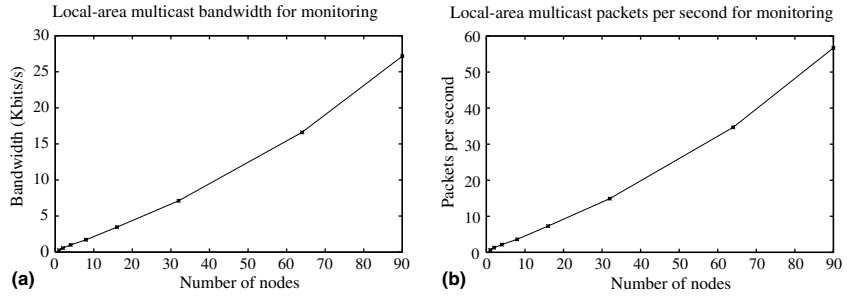

Fig. 4. Scalability as a function of cluster size. (a) Local-area multicast bandwidth; (b) local-area multicast packets per second.

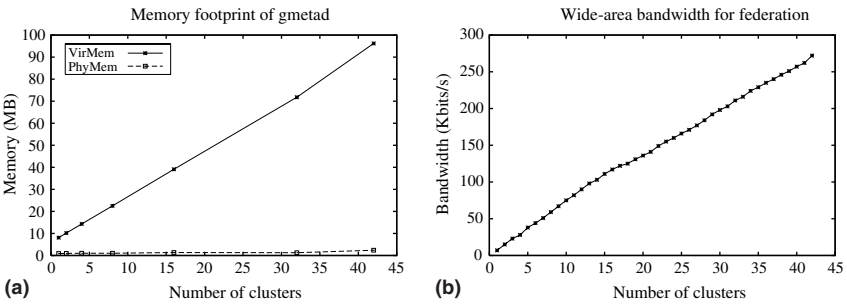

Fig. 5. Scalability as a function of number of clusters. (a) Local memory overhead for `gmetad`; (b) aggregate bandwidth for federation.

`ps` command multiple times and averaging the results. Data for Fig. 5b was collected by polling each of the 42 PlanetLab sites and estimating the federation bandwidth using the size of the monitoring output divided by the default polling rate as described in Section 5.2.2.

Our results show that virtual memory usage for federation is scaling linearly with the number of sites. As mentioned earlier, this linear scaling is a consequence of Ganglia's use of a thread per site, each of which gets a default 2 MB stack. This VM usage can be reduced in a number of straightforward ways. One possibility is to simply reduce the default thread stack size used in Ganglia. Since polling threads in Ganglia only a small fraction of their stack allocations, this should not cause any problems and would immediately result in substantially less VM usage. An alternative, and perhaps better, approach is to eliminate the thread per site approach entirely and to use an event-driven design using I/O multiplexing. Either of these approaches would result in significant reductions in VM scaling as a function of sites. Physical memory footprints and CPU overheads are already either small or negligible for all federation sizes measured.

We also observe that wide-area bandwidth consumed is scaling linearly with the number of sites. This is not surprising given that `gmetad` simply collects cluster data from all PlanetLab sites and does not perform any summarization (i.e. it simply collects the data). For 42 sites, we again observe that Ganglia is using 272 Kbits/s of wide-area network bandwidth. As mentioned, over a week's time, this works out to be 19.15 GB of data or an average of 456 MB of data per site. Over the wide-area, moving this amount of data around on a continuous basis can potentially result in non-trivial costs relative to the costs of the hardware at each site. This is clearly the case with a widely distributed system such as PlanetLab. It is less of an issue on Grid computing systems that link small numbers of large clusters together over research networks such as Abilene.

5.3. Experience on real systems

A key benefit of having a system that is widely deployed is that users figure out interesting ways to exercise its functionality and stress it in new and interesting ways. Original design decisions that seemed like good ideas at the time often need to be revisited in the face of new applications of the system and use in different regimes of the architectural design space. As already mentioned, Ganglia's original design point was scalable monitoring of a single cluster, in particular clusters running the Linux operating system. Since then, it has gone on to achieve great success. It currently runs on over 500 clusters around the world, has been ported to nine different operating systems and six CPU architectures, and has even seen use on classes of distributed systems that it was never intended for. Along the way, the architecture of the system has had to evolve, features needed to be introduced, and the implementation has been continually refined to keep it fast and robust. In this section, we present some of our experiences with real world deployments of Ganglia that try and capture some of this evolution.

5.3.1. Clusters

Clusters, not surprisingly, have been the dominant system architecture that Ganglia has been deployed on to date. In this domain, many of the original design decisions and assumptions made have actually proved to be quite reasonable in practice. The decision to start with a simple architecture which was amenable to a fast and robust implementation led to good scalability, robustness, and low per-node overheads. The decision to use a multicast listen/announce protocol for automatic discovery of nodes as they are added and removed was also key as it eliminated manual configuration and vastly reduced management overhead. This, combined with use of standard software configuration tools such as `automake` and `autoconf`, reduced the barrier to entry to a point where we conjecture people were inclined to simply try the system out and, in most cases, immediately obtained rich and useful functionality and became users.

The use of simple, widely used technologies such as XML for data representation and XDR for data transport was also a good one. These technologies are simple, self-contained, and offer a variety of existing tools which can be leveraged to extend Ganglia in interesting ways. For example, by exporting XML, integrating Ganglia into other information services which add query languages and indexing and building new front-ends to export Ganglia's monitoring information become straightforward exercises. Ganglia's recent integration with the Globus MDS is an example of the former, while WorldGrid's (`www.worldgrid.com`) custom front-ends to Ganglia are an example of the latter. Portability to different operating systems and CPU architectures has also been important. One example of a success here is Industrial Light and Magic, which currently uses Ganglia to monitor over 500 render nodes running a mix of Linux, Tru64, and Solaris.

Ganglia's evolving support for a broad range of clusters, both in terms of heterogeneity and scale, has also exposed issues which were not significant factors in its early deployments. For example, when Ganglia was initially released, clusters with 1000 or more nodes were fairly rare. However, in recent months, we have observed a number of deployments of Ganglia that have exceeded 500 nodes. SUNY Buffalo's HPC cluster, for example, is using Ganglia to monitor over 2000 Dell PowerEdge 1650 and PowerEdge 2650 SMP nodes. Extrapolating from the packets rates in Fig. 4, this 2000-node cluster would seem to imply a multicast packet rate of 1260 packets per second just for the monitoring data alone. Indeed, in practice, even with reductions in the periodic sending rate, we observe a packet rate of approximately 813 packets per second on the SUNY cluster. Clearly, a cluster of this scale challenges our design choice of wanting symmetry across all nodes using a multicast-based protocol. Our assumption of a functional native, local-area IP multicast has also proven to not hold in a number of cases.

A number of other issues have also arose as a result of early decision and implementation decisions. First, monitoring data in Ganglia is still published using a flat namespace of monitor metric names. As a result, monitoring of naturally hierarchically data becomes awkward. Second, Ganglia lacks access control mechanisms on the metric namespace. This makes straightforward abuse possible (e.g. publishing metrics until Ganglia runs out of virtual memory). Third, RRDtool has

often been pushed beyond its limits, resulting in huge amounts of I/O activity on nodes running `gmetad`. As a result, such nodes experience poor performance, especially for interactive jobs. Note, however, that such nodes are typically front-end nodes and not the cluster nodes that are used to run end-user applications. Finally, metrics published in Ganglia did not originally have timeouts associated with them. The result was that the size of Ganglia's monitoring data would simply grow over time. (This has since been partially addressed in the latest version of Ganglia which does feature coarse-grain timeouts.) The above issues are all currently being investigated and some subset of them will be addressed in the next version of Ganglia.

5.3.2. Planetlab

Despite not being designed for wide-area systems, Ganglia has been successfully monitoring PlanetLab for several months now. Following a series of feedback and modifications, it has since demonstrated exceptional stability and currently operates with essentially no management overhead. Besides its current degree of robustness, other properties which have proven valuable in its deployment include ease of installation, self-configuration on individual 2–3 clusters, and its ability to aggregate data from multiple sites and visualize it using RRDtool. On the other hand, it is important to note that PlanetLab really does represent a significantly different design point compared to Ganglia's original focus on clusters connected by fast local-area networks. As a result, we have encountered a number of issues with both its design and implementation. Some of these issues can be addressed within the current architecture with appropriate modifications, while others will likely require more substantial architectural changes.

Within Ganglia's current architecture, there are a number of issues that can be resolved with appropriate modifications. For example, one issue that has arisen lately is Ganglia's assumption that wide-area bandwidth is cheap when aggregating data. While this may be true on the Abilene network, on the public Internet this assumption simply does not hold. There are many, diversely connected sites all around the world, each with widely varying agreements with their ISPs on bandwidth and network pricing. [3] If Ganglia intends to support monitoring in the wide-area, it will need to make more judicious use of network bandwidth. A recent prototype using zlib compression has demonstrated reductions in bandwidth by approximately an order of magnitude for example. Other notable issues that have arisen in Ganglia's deployment on PlanetLab include its limitation on metrics having to fit within a single IP datagram, the lack of a hierarchical namespace, lack of timeouts on monitoring data, large I/O overheads incurred by `gmetad`'s use of RRDtool, and lack of access control mechanisms on the monitoring namespace. Some of these issues (e.g. lack of timeouts) have since been addressed.

[3] As an example, the PlanetLab site at the University of Canterbury in New Zealand currently pays $35 USD per GB of international data it sends as part of its contract with its ISP.

Longer term, there are a number of issues that will likely require more fundamental changes to Ganglia's architecture. Perhaps the biggest issue is scalability, both for monitoring within a single cluster and for federating multiple clusters over the wide-area. Within a single cluster, it is well-known that the quadratic message load incurred by a multicast-based listen/announce protocol is not going to scale well to thousands of nodes. As a result, supporting emerging clusters of this scale will likely require losing some amount of symmetry at the lowest level. For federation of multiple clusters, monitoring through straightforward aggregation of data also presents scaling problems. Ganglia's deployment on PlanetLab has already pushed it into regimes that have exposed this to some extent. Scalable monitoring across thousands of clusters in the wide-area will likely require use of some combination of summarization, locally scoped queries, and distributed query processing [17,30]. Self-configuration while federating at this scale will also require substantial changes to the original Ganglia architecture since manual specification of the federation graph will not scale. One promising direction here might be to leverage distributed hash tables such as CAN [25], Chord [28], Pastry [26], and Tapestry [31].

6. Related work

There are a number of research and commercial efforts centered on monitoring of clusters, but only a handful which have a focus on scale. Supermon [27] is a hierarchical cluster monitoring system that uses a statically configured hierarchy of point-to-point connections to gather and aggregate cluster data collected by custom kernel modules running on each cluster node. CARD [2] is a hierarchical cluster monitoring system that uses a statically configured hierarchy of relational databases to gather, aggregate, and index cluster data. PARMON [8] is a client/server cluster monitoring system that uses servers which export a fixed set of node information and clients which poll the servers and interpret the data. Finally, Big Brother (`http://www.b4.com`) is a popular commercial client/server system for distributed monitoring on heterogeneous systems.

Compared to these systems, Ganglia's differs in four key respects. First, Ganglia uses a hybrid approach to monitoring which inherits the desirable properties of listen/announce protocols including automatic discovery of cluster membership, no manual configuration, and symmetry, while at the same time still permitting federation in a hierarchical manner. Second, Ganglia makes extensive use of widely-used, self-contained technologies such as XML and XDR which facilitate reuse and have rich sets of tools that build on these technologies. Third, Ganglia makes use of simple design principles and sound engineering to achieve high levels of robustness, ease of management, and portability. Finally, Ganglia has demonstrated operation at scale, both in measurements and on production systems. We are not aware of any publications characterizing the scalability of any of these previous systems.

7. Conclusion

In this paper, we presented the design, implementation, and evaluation of Ganglia, a scalable distributed monitoring system for high performance computing systems. Ganglia is based on a hierarchical design which uses a multicast-based listen/announce protocol to monitor state within clusters and a tree of point-to-point connections amongst representative cluster nodes to federate clusters and aggregate their state. It uses a careful balance of simple design principles and sound engineering to achieve high levels of robustness and ease of management. The implementation has been ported to an extensive set of operating systems and processor architectures and is currently in use on over 500 clusters around the world.

Through measurements on production systems, we quantified Ganglia's scalability both as a function of cluster size and the number of clusters being federated. Our measurements demonstrate linear scaling effects across the board with constant factors of varying levels of importance. Measurements on four production systems show that Ganglia scales on clusters of up to 2000 nodes and federations of up to 42 sites. Simple extrapolation based on these numbers combined with local overhead data suggests that Ganglia is currently capable of comfortably scaling to clusters consisting of hundreds of nodes and federations comprised of up to 100 clusters in the wide-area. Additional optimizations (e.g. compression) within the existing architecture should help push these numbers out even further.

Acknowledgements

Special thanks are extended to the Ganglia Development Team for their hard work and insightful ideas. We would like to thank Bartosz Ilkowski for his performance measurements on SUNY Buffalo's 2000-node HPC cluster. Thanks also to Catalin Lucian Dumitrescu for providing useful feedback on this paper and to Steve Wagner for providing information on how Ganglia is being used at Industrial Light and Magic. This work is supported in part by National Science Foundation RI Award EIA-9802069 and NPACI.

References

[1] E. Amir, S. McCanne, R.H. Katz, An active service framework and its application to real-time multimedia transcoding, in: Proceedings of the ACM SIGCOMM '98 Conference on Communications Architectures and Protocols, 1998, pp. 178–189.

[2] E. Anderson, D. Patterson, Extensible, scalable monitoring for clusters of computers, in: Proceedings of the 11th Systems Administration Conference, October 1997.

[3] T.E. Anderson, D.E. Culler, D.A. Patterson, A case for now networks of workstations, IEEE Micro (February) (1995).

[4] D.J. Becker, T. Sterling, D. Savarese, J.E. Dorband, U.A. Ranawak, C.V. Packer, Beowulf: a parallel workstation for scientific computation, in: Proceedings of the 9th International Conference on Parallel Processing, April 1995.

[5] N. Boden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovic, W.Su. Myrinet, A gigabit per second local area network, IEEE Micro (February) (1995).
[6] E. Brewer, Lessons from giant-scale services, IEEE Internet Computing 5 (4) (2001).
[7] S. Brin, L. Page, The anatomy of a large-scale hypertextual web search engine, Computer Networks and ISDN Systems 30 (1–7) (1998) 107–117.
[8] R. Buyya, Parmon: a portable and scalable monitoring system for clusters, Software—Practice and Experience 30 (7) (2000) 723–739.
[9] A. Chien, S. Pakin, M. Lauria, M. Buchanon, K. Hane, L. Giannini, High performance virtual machines (hpvm): clusters with supercomputing apis and performance, in: Proceedings of the 8th SIAM Conference on Parallel Processing for Scientific Computing, March 1997.
[10] B.N. Chun, D.E. Culler, Rexec: a decentralized, secure remote execution environment for clusters, in: Proceedings of the 4th Workshop on Communication, Architecture and Applications for Network-based Parallel Computing, January 2000.
[11] Intel Corporation. Paragon xp/s product overview, 1991.
[12] Thinking Machines Corporation. Connection machine cm-5 technical summary, 1992.
[13] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman, Grid information services for distributed resource sharing, in: Proceedings of the 10th IEEE International Symposium on High-Performance Distributed Computing, August 2001.
[14] I. Foster, C. Kesselman, Globus: a metacomputing infrastructure toolkit, International Journal of Supercomputer Applications 11 (2) (1997) 115–128.
[15] I. Foster, C. Kesselman, S. Tuecke, The anatomy of the grid: enabling scalable virtual organizations, International Journal of Supercomputer Applications 15 (3) (2001).
[16] A. Fox, S. Gribble, Y. Chawathe, E. Brewer, P. Gauthier, Cluster-based scalable network services, in: Proceedings of the 16th ACM Symposium on Operating Systems Principles, October 1997.
[17] M. Harren, J.M. Hellerstein, R. Huebsch, B.T. Loo, S. Shenker, I. Stoica, Complex queries in dht-based peer-to-peer networks, in: Proceedings of the 1st International Workshop on Peer-to-peer Systems, March 2002.
[18] M. Homewood, M. McLaren, Meiko cs-2 interconnect elan-elite design, in: Proceedings of Hot Interconnects I, August 1993.
[19] R.E. Kessler, J.L. Scwarzmeier, Cray t3d: a new dimension in cray research, in: Proceedings of COMPCON, February 1993, pp. 176–182.
[20] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R.Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, B. Zhao, Oceanstore: an architecture for global-scale persistent storage, in: Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems, November 2000.
[21] L. Peterson, D. Culler, T. Anderson, T. Roscoe, A blueprint for introducing disruptive technology into the internet, in: Proceedings of the 1st Workshop on Hot Topics in Networks (HotNets-I), October 2002.
[22] The TeraGrid Project. Teragrid project web page (http://www.teragrid.org), 2001.
[23] UC Berkeley CITRUS Project. Citrus project web page (http://www.citrus.berkeley.edu), 2002.
[24] UC Berkeley Millennium Project. Millennium project web page (http://www.millennium.berkeley.edu), 1999.
[25] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, A scalable content-addressable network, in: Proceedings of the ACM SIGCOMM '01 Conference on Communications Architectures and Protocols, August 2001.
[26] A. Rowstron, P. Druschel, Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems, in: Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms, November 2001.
[27] M. Sottile, R. Minnich, Supermon: a high-speed cluster monitoring system, in: Proceedings of Cluster 2002, September 2002.
[28] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup service for internet applications, in Proceedings of the ACM SIGCOMM '01 Conference on Communications Architectures and Protocols, 1 September 2001.

[29] M. Stumm, The design and implementation of a decentralized scheduling facility for a workstation cluster, in: Proceedings of the 2nd IEEE Conference on Computer Workstations, March 1988, pp. 12–22.
[30] R. van Renesse, K.P. Birman, W. Vogels, Astrolabe: a robust and scalable technology for distributed system monitoring management and data mining, ACM Transactions on Computer Systems (2003).
[31] B.Y. Zhao, J.D. Kubiatowicz, A.D. Joseph, Tapestry: an infrastructure for fault-tolerant wide-area location and routing, Technical Report CSD-01-1141, University of California, Berkeley, Computer Science Division, 2000.