# Panopticon: A Scalable Monitoring System

Duncan Clough
Department of Computer Science, University of Cape Town, Private Bag X3, Rondebosch, 7701, South Africa
dclough@cs.uct.ac.za

Stefano Rivera
Department of Computer Science, University of Cape Town, Private Bag X3, Rondebosch, 7701, South Africa
srivera@cs.uct.ac.za

Michelle Kuttel
Department of Computer Science, University of Cape Town, Private Bag X3, Rondebosch, 7701, South Africa
mkuttel@cs.uct.ac.za

Vincent Geddes*
Department of Computer Science, University of Cape Town, Private Bag X3, Rondebosch, 7701, South Africa
vgeddes@cs.uct.ac.za

Patrick Marais
Department of Computer Science, University of Cape Town, Private Bag X3, Rondebosch, 7701, South Africa
patrick@cs.uct.ac.za

## ABSTRACT

Monitoring systems are necessary for the management of anything beyond the smallest networks of computers. While specialised monitoring systems can be deployed to detect specific problems, more general systems are required to detect unexpected issues, and track performance trends.

While large fleets of computers are becoming more common, few existing, general monitoring systems have the capability to scale to monitor these very large networks. There is also an absence of systems in the literature that cater for visualisation of monitoring information on a large scale.

Scale is an issue in both the design and presentation of large-scale monitoring systems. We discuss Panopticon, a monitoring system that we have developed, which can scale to monitor tens of thousands of nodes, using only commodity equipment. In addition, we propose a novel method for visualising monitoring information on a large scale, based on general techniques for visualising massive multi-dimensional datasets.

The monitoring system is shown to be able to collect information from up to 100 000 nodes. The storage system is able to record and output information from up to 25 000 nodes, and the visualisation is able to simultaneously display all this information for up to 20 000 nodes. Optimisations to our storage system could allow it to scale a little further, but a distributed storage approach combined with intelligent filtering algorithms would be necessary for significant improvements in scalability.

*Now at Amazon.com Development Centre South Africa


*SAICSIT* '10, October 11-13, Bela Bela, South Africa


## Categories and Subject Descriptors

C.2.3 [**Computer**]: Communication Networks—*Network Operations: Network monitoring*; H.3.3 [**Informational Systems**]: Information Storage and Retrieval—*Information Search and Retrieval: Retrieval models*; H.5.2 [**Informational Systems**]: User Interfaces—*Graphical user interfaces (GUI)*

## General Terms

Design, Management

## Keywords

Monitoring System, Scalability, Visualisation

## 1. INTRODUCTION

The last decade has seen the emergence of massively large data centres composed of tens of thousands of nodes. The scale of data centres run by companies such as Amazon and Google has expanded rapidly as consumers demand ever more resources. As systems begin spanning data centres, large scale monitoring becomes essential.

With an ever increasing number of nodes, redundancy has moved outside the individual server, giving more importance to effective management and monitoring of an entire fleet of servers. However, few of the mature monitoring systems, that are constantly used in system administration, are designed for supervision of very large networks.

There are a variety of monitoring systems for identifying and detecting different classes of problem. While specialised monitoring systems can be deployed to detect most *specific* problems, more general systems are required to handle unexpected systemic issues [4]. General monitoring systems usually measure *metrics* such as CPU usage, free memory, network traffic, and availability [12]. Given such a set of metrics from a *fleet* of *nodes* (computers); system load, application performance, and outages can be readily interpreted. This has applications ranging from diagnosing performance bottlenecks in a single node to optimising cluster-usage in Capacity Computing.

Designing a scalable and efficient monitoring system is not a trivial task. A naïve implementation may impose

a heavy burden on a network, wasting a resource that the monitoring system is intended to preserve [14]. A large scale fleet will produce a massive amount of information that will need to be efficiently stored and quickly displayed. *Panopticon* is our experimental system that is able to non-intrusively monitor, efficiently store and quickly retrieve metrics from a very large fleet. As a prototype, it was not required to implement all the features of a general purpose monitoring system.

The monitoring and storing of metrics on a large scale is not the only difficulty; effectively visualising them is a problem that is often overlooked. If not displayed effectively, this information can easily overwhelm the user, hindering their ability to identify problems [4]. Traditionally, graphs are used for visualising metrics and are very effective on a small scale. However, on a large scale, graphs become visually limited by the number of curves that can be plotted simultaneously. Therefore, a new method of visualising monitoring information is required — one that can deal with massive sets of data, and takes into account the multi-dimensional nature of the information. A purely large-scale overview would not provide enough node-specific detail to be useful for identifying problems in individual nodes. Panopticon's Visualisation component presents a solution to these issues with our *Node-map* that provides information at multiple levels of detail, and our *Metric-map* that shows an overview of the fleet status. Thus, the visualisation provides a general overview of the status of all nodes, while also allowing access to the metrics of a specific node. Live changes in metric values are highlighted for easy identification and historic information on monitored nodes is also accessible.

We evaluated Panopticon by monitoring a live test clusters of 56 computers over a period of 4 months, as well as with simulated data for significantly larger fleets. Our analysis shows that the Panopticon approach can scale to very large fleets, with the potential to scale to millions of monitored nodes and be extended into a viable production monitoring system.

## 2. RELATED WORK

For small scale monitoring, *RRDTool* is widely used for storage of metrics and their visualisation [13]. Its approach of storing individual, time-granularity-specific metric values in their own separate databases has been very effective, but quickly becomes inefficient when dealing with thousands of computers [12]. As the scale increases, communicating information from nodes to the databases also becomes problematic.

*CARD* [1] is an early example of a monitoring system that employs a hierarchical approach specifically to improve scalability. This approach was modernised by *Ganglia* [12], with a multi-level tree of aggregation nodes pulling information up from monitoring *agents* on individual computers. Communication bottlenecks were avoided by only storing information in top-level RRDTool databases, thus eliminating network usage during information retrieval. Despite having a scalable architecture for collecting information, reliance on RRDTool storage limited the system to below 10 000 nodes. This highlights the need for scalability to be central in the design of all aspects of a large scale monitoring system.

Another approach is to distribute the storage of metrics across all nodes in the hierarchy, with queries being pushed down the hierarchy to be serviced by all interested nodes. *Astrolabe* [18] showed that this method is particularly scalable, with the system capable of handling a theoretical maximum of 300 000 nodes. Unfortunately, Astrolabe was designed for data mining and not visualising detailed information, so queries are limited to aggregates in order to reduce network usage. We improve on this limitation, making individual metric values from all nodes viewable.

A high level of fault tolerance is important to any massively scalable monitoring system. On a very large scale, problems become inevitable and agents have to gracefully recover from these problems [12]. Both Ganglia and Astrolabe use multiple levels of redundancy to increase robustness. Astrolabe goes further, using a randomised point-to-point message exchange to limit reliance on single channels.

The third aspect to a scalable monitoring system, visualising the monitoring information in a scalable manner, is often overlooked. Simultaneously visualising data from tens of thousands of computers leads to a user being presented with so much information that it is extremely difficult, if not impossible, to comprehend it effectively. A fleet of a million nodes would only compound this issue. Traditional graphs become too cluttered to effectively display information [11]. Tools such as OVIS-2 [3] suggest that statistical methods should be used to highlight important information. While this is effective at reducing information overload, our research specifically investigated techniques for visualising all the information rather than a subset. OVIS-2 also includes a physical 3D representation of a cluster, with components being coloured individually according to metric values. Although the physical representation is very useful in relation to actual clusters, it also limits the visual scalability.

Our monitoring information is multi-dimensional as there are multiple metrics to consider. This compounds the problem of large amounts of information to be expected from a large fleet. In order to build an effective, scalable visualisation for our monitoring system we looked at general visualisation techniques for massive, multi-dimensional datasets.

Pixel-oriented [9, 16, 15] aggregation solves the problem of loss of detail in multiple zoom levels by rearranging information into a more suitable format for the current zoom level. Hierarchical [7, 11] aggregation attempts to group related information to provide a meaningful overview of and context for substructures. These techniques allow a visualisation to provide a broader context to smaller subsets of data. Since nodes within a fleet have a logical and physical layout, aggregation is appropriate to monitoring systems.

Orthogonal projections from higher dimensions into understandable 3D or 2D coordinates help reduce the complexity of multi-dimensional data-sets, but have an associated loss of information [2]. *ShapeVis* [11] uses projections to group objects with similar multi-dimensional characteristics in similar 3D spacial locations — a concept that inspired our Metric-map. Parallel Coordinates [8, 10] are a popular method of rendering multi-dimensional data, but require aggregation techniques for better scaling [11]. We therefore decided not to include Parallel Coordinates in our visualisations.

Perspective manipulation and user-interaction can be used to enhance a visualisation. Filtering [2], linking and brushing[17] (propagating selection between different visualisations), zooming and distortion [11] are examples of methods that can improve a visualisation.

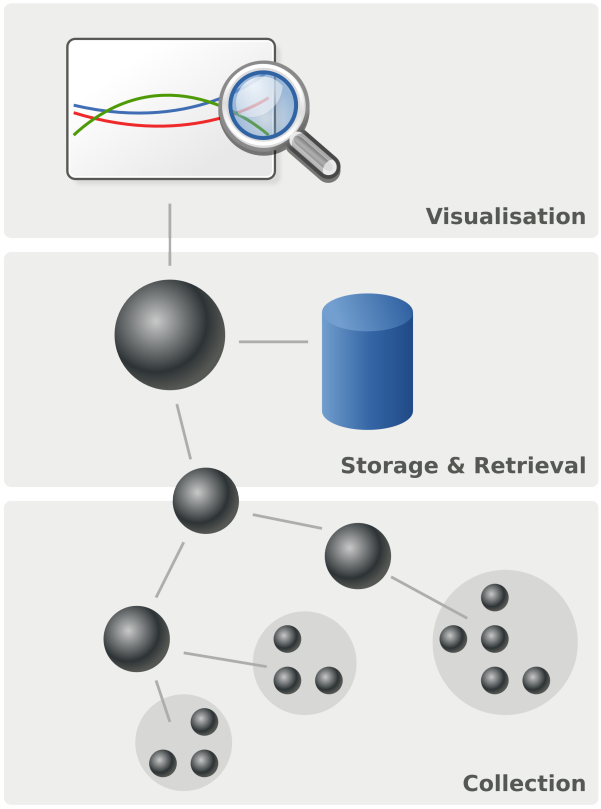


**Figure 1: Overview of Panopticon system architecture, showing the separate Visualisation, Storage & Retrieval, and Collection components**

Although each of the discussed techniques can be individually successful, it is a combination of techniques that often results in the most successful visualisations [10]. Our visualisations were, therefore, designed to use multiple techniques.

# 3. DESIGN & IMPLEMENTATION

Panopticon is the combination of three separate components, each focussing on one part of a much larger problem: Visualisation, Storage & Retrieval, and Monitoring & Collection (Figure 1). Each component runs as a separate entity, and communicates with other components over the network. This allowed each component to be developed and evaluated independently.

As a prototype system, the aim of Panopticon is to investigate scalability in the order of tens of thousands of nodes. Many features that would be desirable in production monitoring systems are considered non-vital to our research.

## 3.1 Monitoring Component

A monitoring system is responsible for measuring metrics of monitored nodes and collating the data without adversely affecting the performance of those nodes or their network [12]. It should be lightweight so that it does not interfere with activities on nodes and is able to continue reporting even when nodes are under intense load. It must be highly reliable and robust, as reporting failures will be perceived as node failures. It should also require as little maintenance as possible, so that it is not an administrative burden on a network. The importance of these characteristics only increases with scale [18].

In a similar approach to Astrolabe [18], the monitoring and aggregation system is a hierarchical network of agents (the *node-tree*). The agent is installed as a daemon on each node. It is responsible for observing and recording host system metrics, and providing them to other nodes on request. Internal nodes in the tree will poll their children for recent metrics. This information is propagated up through the tree, recording the route within the data. The root node receives metrics from every node in the system while only having to communicate with a few child nodes.

The system can handle unreliable hardware and networks by using multiple parent nodes for each child, all the way up to multiple roots. Duplicate data is automatically reconciled by ignoring the oldest data for a particular node.

For our purposes, we selected a similar small set of metrics as Ganglia [12]: uptime, usage, load (1, 5, and 15 minute averages), network usage, and free memory. These metrics are commonly used by monitoring systems, and easily available on POSIX. The exact metrics monitored would have negligible effect on our scalability evaluation, which is dependant on data quantity, rather than content.

The inter-agent communication uses a very simple client-server binary protocol. The only supported operations are requests for all available metrics on a particular node (both aggregated and locally observed) and a status report on the availability of aggregated data.

For communication with other systems, agents could run a Web Service Interface. Requests followed the REST architecture [6] and data was encoded in JSON [5]. This interface was used between the root node and the Storage and Retrieval system.

## 3.2 Storage & Retrieval Component

Determining the cause of a problem on a network often requires looking into the past and comparing against previous behaviour. Therefore, as well as providing up-to-date information on the current state of a fleet, a monitoring system should store any collected information for later use. We required Panopticon to store all live metrics at least once every 5 minutes, and to retrieve metrics as requested by the Visualisation component in real-time. The Storage & Retrieval component requires a high degree of availability, as it provides critical information without which system administrators are blind.

Our storage system was built around a high-performance SQL database, *MySQL*. Having a centralised MySQL server is an obvious limit to scalability and reliability: It is a single point of contention and failure. However, the simplicity of this approach in our proof-of-concept is considered to be enough of a gain to outweigh these issues.

A time granularity of 5 minutes is widely used by monitoring systems (it is MRTG's default [13]). It strikes a reasonable balance between information detail and network and storage overhead. In early tuning experiments, our RDBMS was shown to be capable of inserting 10 million rows in $170 \pm 32\,s$. This gave us an upper bound of being able to capture the information from 14 million nodes with a 5 minute resolution. Since this was well above our desired scale, a centralised RDBMS was considered sufficient for our system: it provided a good starting point for testing, while having reasonable scalability.

Storage and retrieval features are implemented separately. Storage is accomplished via a daemon polling multiple root aggregation nodes in parallel, and storing re-

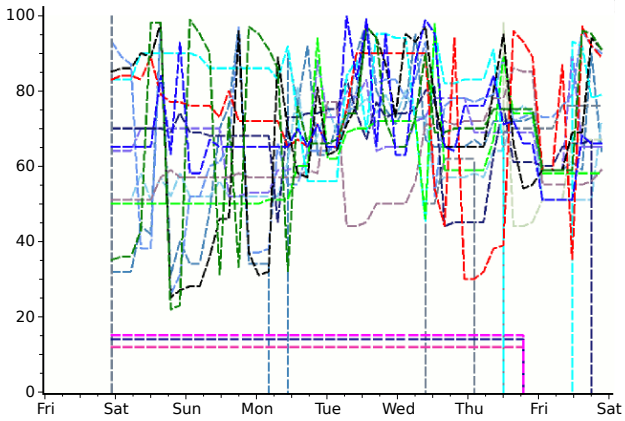


**Figure 2: A graph of memory usage in only 20 machines over a week, highlighting the problem of graph clutter.**

trieved metrics in the database. Incoming data is timestamped on the storage node (to remove the requirement that every participating node have accurate time) and quantised into 5-minute buckets. Only two database tables are used: a node list table (with static information and last-seen time stamp), and a full historical archive table. The archive table is indexed with a primary key on the `(address, time stamp)` tuple so that full table scans are never needed for the supported queries (below).

Retrieval is handled by a separate daemon, answering requests from a visualisation front-end and sending updates as they become available. The protocol is text-based, for ease of debugging. As the number of monitored nodes may be very large, the responses are delta-compressed where possible.

Commands supported are: selecting metrics and/or nodes of interest, and enabling or disabling live status updates. Historical queries can be performed for fleet status at a specific point in time, or for a set of consecutive metric values from a specific node that are suitable for a time-axis graph.

## 3.3 Visualisation Component

The central issue addressed in this work is the problem of monitoring and visualising a huge number of nodes simultaneously. So, above all, the visualisation must be scalable. We also required low-level information to still be easily accessible within our system. To meet these needs, we designed two different overview visualisations (Node-map and Metric-map) combining visualisation techniques discussed in Section 2. Additionally, we provided traditional per-node historical graphs. The key difference between the two visualisations is their approach to solving the problems of viewing low- and high-level information. The visualisations were also able to interact through two-way selection propagation.

The Visualisation component aims to present information about the state of the fleet to the user. This is done so that the user can identify problems with the fleet and their potential causes. New information needs to be presented to the user as soon as it is available and historic data also needs to be accessible.

Graphs can become too cluttered if too many computers are being monitored, and therefore are not applicable to Panopticon as the main visualisation (see Figure 2).

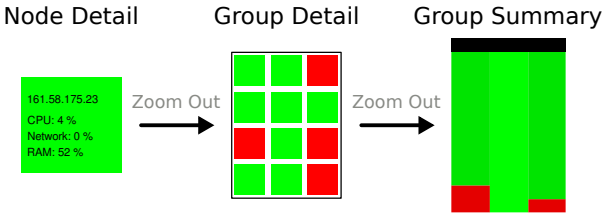


**Figure 3: The three different levels of detail of the Node-map.**

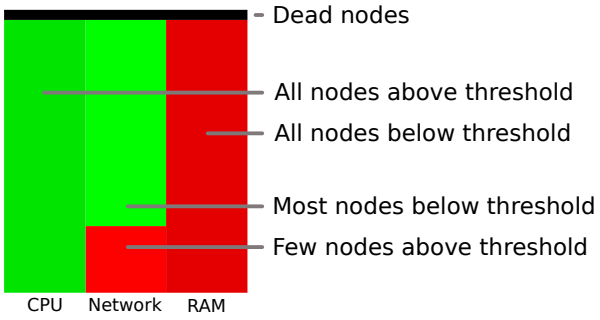


**Figure 4: Information in the Group Summary level of detail is summarised using an adjustable threshold value, and represents an overview of activity within a Group.**

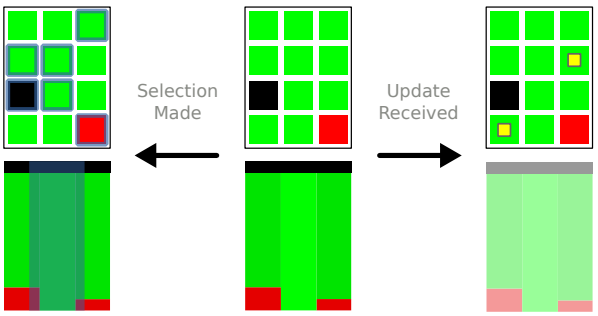


**Figure 5: An example of how selection and update highlighting are visualised on the Node-map.**

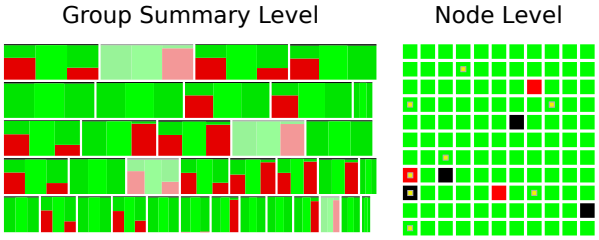


**Figure 6: Update highlighting shown in the context of a fleet, where areas of change are easily identifiable.**

However, as they are still useful when only looking at a single monitored machine, a historical graphing tool is included for that purpose. Nodes can be selected in both visualisations, and have their historic information plotted according to various time-granularities.

### 3.3.1 The Node-Map

The Node-map is designed to be the main source of low-level information, while also providing an aggregated view for a general summary of a fleet's status. Nodes are aggregated by logical layout and displayed in multiple levels of detail as shown in Figure 3. Metrics are grouped to be above or below an adjustable threshold. This reduces the amount of information presented by the Node-map, reducing the information needing to be processed by the user. The Node-map also briefly highlights any nodes that have had changes in their metric values, so that a user can easily notice any changes that occur (Figures 5 and 6).

When dealing with large numbers, viewing each node individually becomes infeasible because (*i*) too much information is then presented to the user, and (*ii*) most window tool-kits struggle to render tens of thousands of shapes in real time. Therefore, once the user has zoomed out, a group of nodes is replaced with a summary of the status of those nodes. This significantly reduces the amount of information that the user has to process, and reduces the complexity of the scene to be rendered.

The summary is designed to highlight the number of dead nodes, and the number of nodes on either side of the threshold for all three metrics being monitored. The dead nodes are represented by a black rectangle at the top of the top of the group. The width of the rectangle is set to be the group width, and the height is calculated by

$$\texttt{dead\_height}_g = \begin{cases} \max\left\{\frac{d_g}{n_g}, 5\%\right\} \times \texttt{height}_g & , \text{ if } d_g > 0 \\ 0 & , \text{ otherwise} \end{cases}$$

where a group $g$ has $n_g$ nodes, $d_g$ of which are dead. In the case where a tiny proportion of the total nodes are dead, we increase the proportion of the height to 5% of the group's height so that the black rectangle will at least be noticeable by the user.

The remaining space is then divided into three columns to represent information about each of the three metrics (in this case we are using CPU, Network and RAM usages, but these could be any three arbitrary metrics). An example is shown in Figure 4.

When at this level of detail, the user is now able to select entire groups of nodes. Selecting a group has the same effect as individually selecting every node within the group. Since individual nodes are not visible, the selection is aggregated as well. Instead of a border, a semi-transparent blue rectangle is drawn on top of the Group Summary, that represents the percentage of nodes within the group that are currently selected (Figure 5).

### 3.3.2 The Metric-Map

The Metric-map was designed to give a high-level overview of the status of the fleet and, as a result of this focus, specific details are not directly accessible. The base of the Metric-map is a hexagon, with coloured circles representing information about the fleet using their position, area, opacity and borders (Figures 7, 8 and 9).

When nodes are given to the Metric-map, their metrics are quantised (rounding down) to percentages in the set $Q = \{0\%, 20\%, 40\%, 60\%, 80\%, 100\%\}$. Then, nodes with the same values across all three metrics are put into

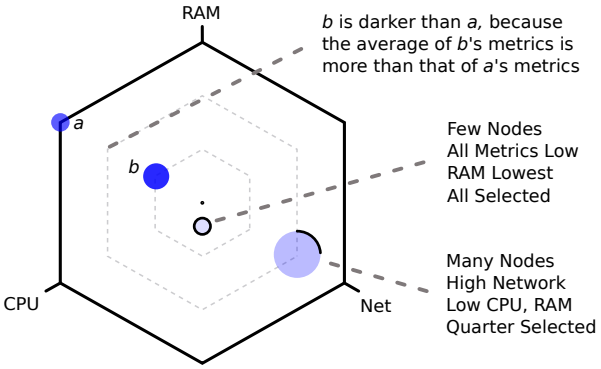


**Figure 7: The features of the metric-map.**

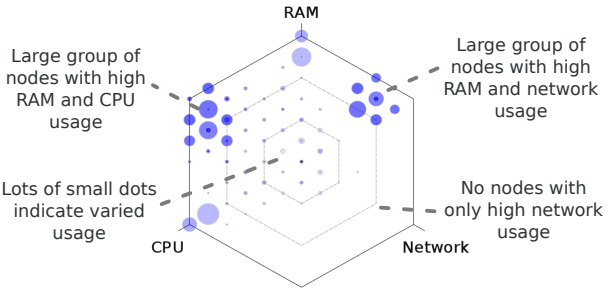


**Figure 8: An example scenario on the metric-map.**

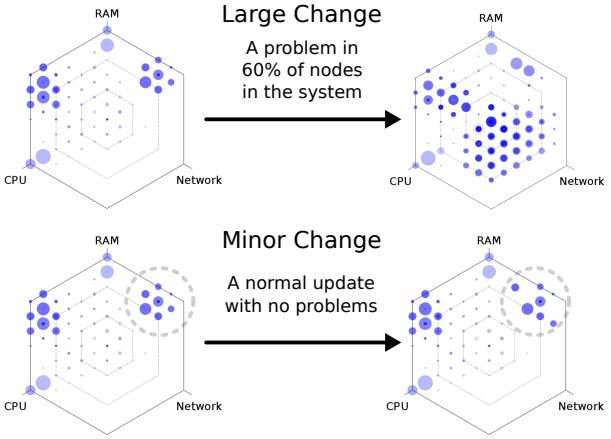


**Figure 9: Visualising differences in fleet status on the metric-map.**

*bins*, where each bin can be represented by a tuple $\mathbf{b} = (m_1, m_2, m_3) \in Q^3$.

For placement, the higher an $m_i$ is, the more it is "pulled" towards its corresponding corner, $\vec{\mu}_i$. The exact position $\mathbf{p} \in \mathbb{R}^2$ of a bin $\mathbf{b}$ is calculated as a sum of each $m_i$ multiplied with its corresponding $\vec{\mu}_i$:

$$\mathbf{p} = \sum_{i=1}^{3} m_i \vec{\mu}_i$$

This calculation does not result in a unique $\mathbf{p}$ for all $\mathbf{b}$. For example $(1\%, 1\%, 1\%)$ and $(40\%, 40\%, 40\%)$ both map to the point $(0, 0)$. Therefore the Metric-map also adjusts the alpha intensity based on the overall load for a given bin, with higher intensity corresponding to higher load. The alpha intensity $\alpha_\mathbf{b}$ of a given bin $\mathbf{b}$ is calculated as an average of each $m_i$ thresholded to a maximum of 1:

$$\alpha_\mathbf{b} = \min\left\{1,\ \frac{1}{3}\sum_{i=1}^{3} m_i\right\}$$

This method was used because it is intuitive to have richer colours linked to more activity, and fainter colours linked to less activity.

Bins are plotted as circles on the diagram. The radius of the circle is logarithmically proportional to the number of nodes represented by the bin. A logarithmic scale allows the Metric-map to scale well as the number of nodes increases.

Bins are selectable; clicking on a bin will select all nodes within that bin and highlight it with a thick black border. In the case that there is a partial selection within a bin, the bin is only partially bordered. The exact formula for the degrees of a bin $\mathbf{b}$ to be bordered is:

$$\theta_\mathbf{b} = \begin{cases} \max\left\{\frac{s_\mathbf{b}}{n_\mathbf{b}} \times 360,\ 1\right\} & ,\ \text{if } s_\mathbf{b} > 0 \\ 0 & ,\ \text{otherwise} \end{cases}$$

where $s_\mathbf{b}$ is the number of selected nodes in $\mathbf{b}$, and $n_\mathbf{b}$ is the total number of nodes in $\mathbf{b}$. In the case where $\frac{s_\mathbf{b}}{n_\mathbf{b}}$ is very small, we make sure that a selection is visible by making $\theta_\mathbf{b}$ at least 1.

## 3.4 System Design Methodology

An iterative methodology was followed for the development of our system. Firstly, the simplest working solution was put together, using scripting tools where possible. Additional functionality was developed as, and when, required by other components in the system.

Components were developed in different languages, as each component was developed using the most appropriate tools for its task. The Monitoring system was written in D, a low level systems programming language. The Database interface was implemented in Python, a scripting language. C++ and the Qt toolkit were used for the Visualisation front-end, using Qwt for historical graphing. OpenGL (via the Qt Graphics View Framework) was used for the main visualisations. Development and testing was done in Linux.

# 4. EVALUATION

Since access to upwards of 500 nodes was infeasible, simulated data was used at various stages to represent a very large fleet. Each component was subjected to an individual evaluation and tested with simulated data as well as data from our live test fleet over a period of four months.

**Table 1: Overview of our test fleets.**

| System | nodes | clusters | location |
|---|---|---|---|
| EC$^2$ Cluster | 20 | 1 | off-site |
| TSL | 36 | 1 | on-site |
| Everything | 56 | 2 | mixture |

Panopticon was tested on two separate clusters, as well as a single WAN cluster made by combining both the separate clusters (see Table 1). The TSL cluster was an active university computer laboratory being used by undergraduate students. Load was periodically generated on the EC$^2$ cluster by using a custom parallel $n$-body simulation.

## 4.1 Monitoring and Collection

In testing the monitoring system, we were faced with a practical problem: we had around 100 computers at our disposal, but wanted to see if our system could cope when 10 000 or more nodes were monitored. Therefore, we simulated fake zones which then aggregated their fake information to their parent aggregation nodes. Since the exact values of the information being monitored did not affect the behaviour of the collection system, the simulated results give a reliable indication of expected performance on real hardware. All reported results are based on average of 9 test-runs.

## 4.2 Storage and Retrieval

Over the four month test period the Storage and Retrieval system collected around one million rows of data (i.e. node-measurements) at a resolution of 5 minutes per node.

Raw storage component rates were determined by simulating a 10 000 node fake zone on a single root node and polling it continuously. Since the actual values of the data do not affect the database performance (within certain limits), pseudo-random data was used.

The performance of the data retrieval subsystem was evaluated for a fleet of a million nodes. Similarly, psuedo-random data was used, as the only effect the data has on performance is its delta-compressibility.

## 4.3 Visualisation

The Visualisation component was evaluated by two expert users: a departmental systems administrator from our university and a developer from Amazon EC$^2$ Web Services. While using our prototype system, these expert users were asked questions regarding the efficacy of the visualisations, with their responses being recorded. More detailed usability testing was not applicable, as Panopticon was developed to be a proof-of-concept, not a production system.

The expert users were shown the system in two different configurations: with live reported data to demonstrate the real-time aspects and with simulated data to demonstrate scalability. For the live data, 120 nodes were visualised due to intentional duplication in the aggregation component's configuration. Of these, around 55 nodes were active and reporting. Since there were at most only a few hundred computers at our disposal, a simulated monitoring system was required in order to test the Visualisation component with large fleets (we simulated 15 000 nodes).

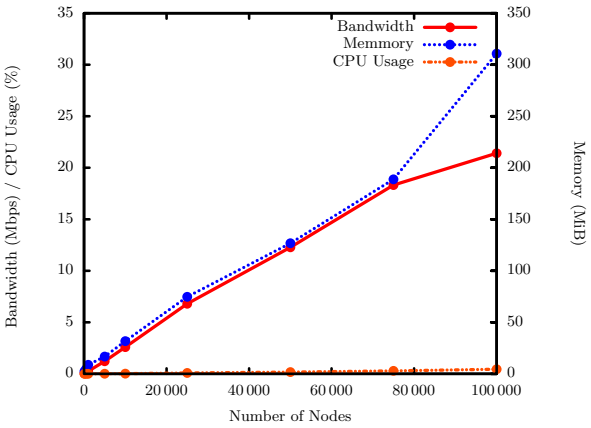


**Figure 10: The resources required of Panopticon's top aggregation node — they are well within the limits of current hardware.**

Unlike the other components which could be tested with random data, the Visualisation component requires realistic and sensible data for any meaningful evaluation to take place. However, implementing a back-end that can (*i*) realistically simulate a real-time monitoring system, (*ii*) supply realistic and consistent historic data, and (*iii*) have metrics that realistically relate to each other, is an extremely complex task requiring prolonged access to many more computers than we had access to. As a compromise, we generate pseudo-realistic monitoring information, with relative ease, by assigning simulated nodes to a set of characteristic usage classes (e.g. web servers, simulation clusters, general-use computer laboratories). The nodes are collected into groups according to usage classes, as could be expected in an orderly data centre. Each class was assigned common failure modes through consultation with expert users.

## 5. RESULTS AND CONCLUSIONS

In our four month real world monitoring exercise, our monitoring agents proved themselves to be robust and have a low system overhead. CPU and memory usage on monitored nodes was found to be negligible, even when polling once every 15 seconds. Aside from some minor MySQL dead-lock issues, which occurred even at the lowest transactional isolation level, everything ran well and was stable. During the last two months, the Visualisation component regularly connected to the retrieval system and was able to view the state of our test fleet.

In evaluations with simulated data, the performance requirements of aggregated nodes was shown to be well within the limits of consumer hardware; Table 2 and Figure 10 show our system was easily capable of aggregating information from 100 000 nodes to a single aggregation node on a LAN. Further protocol improvements, such as incorporating data compression, would be required for the system to be applicable to WANs or larger fleets.

The Storage component produced approximately 100 MiB of data and indices in the four month test, yielding an average of 30 KiB per node per day. Therefore, 1 TiB of storage would be able to hold a 90-day monitoring history for just under 400 000 computers — a trivial expense when considering the cost of maintaining a fleet of that size. When dealing with these quantities of information

**Table 2: Results for scalability evaluation. CPU usage was negligible and excluded.**

| Node Count | Bandwidth (Mbps) | Memory (MiB) |
|---|---|---|
| 100 | 0.03 | 2.44 |
| 500 | 0.13 | 4.72 |
| 1 000 | 0.25 | 8.72 |
| 5 000 | 1.22 | 16.76 |
| 10 000 | 2.59 | 31.76 |
| 25 000 | 6.20 | 74.76 |
| 50 000 | 12.28 | 126.76 |
| 75 000 | 18.32 | 188.76 |
| 100 000 | 21.41 | 310.76 |

**Table 3: Time taken for visualisation queries on 1 million nodes, averaged over 100 tests.**

| Query | Wall Time |
|---|---|
| Full State | 31.9 ± 0.3 s |
| Update | 19.9 ± 0.2 s |
| Rewind | 13.2 ± 0.1 s |
| Historic Data | < 0.002 s |

we found it important to ensure that database queries avoid full table scans and only return relevant information wherever possible.

The monitoring system communicated with the storage via a text-based JSON protocol. JSON was chosen for simplicity and readability, but parsing the information sent from the monitoring system became a significant bottleneck. Using Python's *simplejson* parser, our system was able to poll 10 000 nodes in 95 ± 5 s, leading to a limit of 25 000 nodes when monitoring at a 5-minute resolution.

The performance of the data retrieval subsystem was evaluated for a virtual fleet of a million nodes (see Table 3). It takes 32 s for a Full State query to complete, which is acceptable since it is a once off start-up cost. Update times of 20 s are also suitable, given the 5-minute monitoring resolution and the fact that this is a background operation that the user is not aware of. Historic Queries are instantaneous and therefore well within acceptable limits. Only the rewind time of 13 *s* is unacceptable, as near real-time response times are required for replay functionality to be immediately useful.

Despite these limitations, our system showed that an ordinary RDBMS could be used to store and retrieve information from a monitoring system with the order of 100 000 nodes in real-time. We suggest that in future work, a distributed storage system be investigated to achieve further scalability. While a centralised RDBMS is suitable in terms of storage requirements for a million-node fleet, other approaches would have to be followed to improve retrieval performance. We suggest either a decentralised, non-ACID-compliant storage system or a high degree of partitioning across multiple servers.

In the Visualisation component evaluations, overall, the expert users reported our visualisations to be useful when

**Table 4: Reported feedback from our two expert users on the Visualisation component, obtained during an interview while they experimented with the system.**

| | Positive Comments | Suggested Improvements |
|---|---|---|
| **Node-map** | Intuitive to use | Recent History view |
| | Layout is simple to understand | |
| | Effective group summary | |
| | Rewind ability is valuable | |
| **Metric-map** | Surprisingly intuitive to use | Metric-map should be the emphasised feature in the GUI |
| | Provides a useful overview | Update highlighting |
| **Historic Information** | Replay Tool very useful | More complex graphing tool |
| | | Recent History as part of main visualisation |
| **General** | Visualisations very effective for 100s of nodes | More than 3-level hierarchies for visualisation |
| | Effective up to 20 000 nodes | Update highlights should also appear on the Metric-map |
| | Selection propagation between visualisations is valuable | |

monitoring up to 20 000 computers. Although initially unfamiliar, the Metric-map was said to be "surprisingly intuitive" to use. One user stressed the importance of providing context to a current visualisation, through access to historic information. While the Replay Tool provided basic functionality in this regard, it was suggested the visualisations should incorporate a view of recent history. It was also noted that the expert users did not make use of the Node-detail level of detail. Table 4 lists all significant expert user feedback.

The expert users reported that the Visualisation component provided an effective visualisation of the overall fleet status. The Metric-map was seen to be better at representing overall information, with the Node-map being better suited to individual information. However, the interaction between the two visualisations, through selection propagation, was seen as the most important aspect of both.

Including easily accessible recent history provides a valuable perspective on current information. Users proposed that this would be convenient for identifying possible issues with the current state of a fleet. Making this recent-history visible alongside the real-time visualisation is a possible extension to enhance the Visualisation component, but would also require improvements to the data-retrieval process. While our system focussed on providing an alternative to graphing as a means of viewing monitoring information, graphs were still seen as necessary for viewing specific, long-term, historic information.

Features such as logical grouping, aggregation, multiple levels of detail and selection propagation between independent views enhanced our system's scalability. Information filtering techniques would be useful to further increase the scalability of the Visualisation component.

As a proof-of-concept system, Panopticon was successful. All components were shown to be able to scale effectively into the tens of thousands of monitored nodes, mainly limited by the Visualisation component and information retrieval speeds.

# 6. REFERENCES


[1] E. Anderson and D. Patterson. Extensible, scalable monitoring for clusters of computers. In *LISA '97: Proceedings of the 11th USENIX conference on System administration*, pages 9–16, 1997.

[2] D. Asimov. The grand tour: a tool for viewing multidimensional data. *SIAM Journal of Scientific and Statistical Computing*, 6(1):128–143, January 1985.

[3] J. M. Brandt, B. Debusschere, A. C. Gentile, J. R. Mayo, P. P. Pébay, D. Thompson, and M. H. Wong. OVIS-2: A robust distributed architecture for scalable RAS. In *Proceedings of 22nd IEEE International Parallel and Distributed Processing Symposium*, pages 1–8, April 2008.

[4] J. M. Brandt, A. C. Gentile, D. J. Hale, and P. Pébay, P. Ovis: A tool for intelligent, real-time monitoring of computational clusters. In *Proceedings of 20th IEEE International Parallel and Distributed Processing Symposium*, April 2006.

[5] D. Crockford. *RFC 4627: The application/json Media Type for JavaScript Object Notation (JSON).* IETF The Internet Society, July 2006.

[6] R. T. Fielding. *Architectural Styles and the Design of Network-Based Software Architectures.* PhD thesis, University of California, Irvine, 2000.

[7] J. Goldstein and S. F. Roth. Using aggregation and dynamic queries for exploring large data sets. In B. Adelson, S. Dumais, and J. Olson, editors, *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Celebrating interdependence*, pages 23–29. ACM, April 1994.

[8] A. Inselberg and B. Dimsdale. Parallel coordinates: A tool for visualizing multidimensional geometry. In A. Kaufman, editor, *Proceedings of the 1st Conference on Visualization '90*, pages 361–378. IEEE Computer Society Press, October 1990.

[9] D. A. Keim. Designing pixel-oriented visualization

techniques: Theory and applications. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):59–78, January 2000.

[10] D. A. Keim. Information visualization and visual data mining. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):1–8, January 2002.

[11] M. Kreuseler, N. Lopez, and H. Schumann. A scalable framework for information visualization. In *Proceedings of IEEE Symposium on Information Vizualization 2000*, page 27. IEEE Computer Society, 2000.

[12] M. L. Massie, B. N. Chun, and D. E. Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817–840, 2004.

[13] T. Oetiker. MRTG - the multi router traffic grapher. In *Proceedings of the 12th USENIX Conference on System Administration*, pages 141–148, December 1998.

[14] F. Sacerdoti, M. Katz, M. Massie, and D. Culler. Wide area cluster monitoring with ganglia. In *Proceedings of the IEEE International Conference on Cluster Computing*, pages 289–298. IEEE Press, December 2003.

[15] J. Schneidewind, M. Sips, and D. A. Kiem. An automated approach for the optimization of pixel-based visualizations. *Information Visualization*, 6(1):75–88, March 2007.

[16] M. Sips, J. Schneidewind, D. A. Keim, and H. Schumann. Scalable pixel-based visual interfaces: Challenges and solutions. In *Proceedings of Tenth International Conference on Information Visualization*. IEEE Press, July 2006.

[17] C. Stolte, D. Tang, and P. Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics*, 8(1), January 2002.

[18] R. Van Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems*, 21(2):164–206, May 2003.