RC22531 (W0207-124) July 25, 2002

IBM Research Report

A Publish-Subscribe Architecture and Methodology for Monitoring Distributed Systems

Karen Witting, James R. Challenger, Brian M. O'Connell
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

A Publish-Subscribe Architecture and Methodology for Monitoring Distributed Systems

Karen Witting
IBM T. J. Watson Research Center
P. O. Box 704
Yorktown Heights, NY 10598
witting@us.ibm.com

Jim Challenger
IBM T. J. Watson Research Center
P. O. Box 704
Yorktown Heights, NY 10598
challngr@us.ibm.com

Brian O'Connell
IBM Global Services Special Events Web Solutions
P.O. Box 12195
3039 Cornwallis Road, RTP NC 27709
boc@us.ibm.com

ABSTRACT
In order to support complex, rapidly changing, high-volume websites many components contribute to keeping the content current. Monitoring the workflow through all these components is a challenging task. This paper describes a system in which objects created by the various heterogeneous, distributed components can be distributed to any application choosing to present monitoring information. Creation, distribution, and presentation of the objects are all independent, leading to highly responsive and flexible monitoring of the complex system.

Keywords
Publish-Subscribe, Monitoring, Distributed Systems, Workflow Monitoring, Queue Monitoring, High Volume Web Serving, Content Management

INTRODUCTION
Systems comprised of a large number of interacting components require a monitoring system that is flexible and adjusts for its ever-evolving needs. Modern, high volume web sites and their supporting infrastructure are an example of this kind of large system. Components comprising systems such as this are often geographically dispersed. The now-common "24x7" availability requirements for such systems means that machines are dynamically added and removed to adjust for changing load and hardware failures. Any machine may have a range of components that must be monitored, and the collection of components on any particular machine may change over time. New types of components may be needed, and previously used components may be removed from the system. Any particular component may provide different types of monitoring data over time.

A flexible monitoring system must be able to collect monitoring data from disperse machines and diverse components and present different views of the same data as well as differing levels of detail. Monitoring data is used for a variety of purposes; detailed problem resolution, general flow analysis, assessment of system requirements, problem recreation, and for communication with management and peers. Views are required in real time to give confidence that the system is operating smoothly and to highlight problems as they appear. Views are also required for offline analysis. Since system structural characteristics such as hardware, operating systems and networking capabilities may need to be flexible over time, the monitoring system must be able to accommodate these kinds of changes.

In this paper we describe the system designed and implemented to monitor the publishing and content distribution systems for the Sydney 2000 Olympic Website [1] [6]. The system has been extended since then for use in monitoring the publishing infrastructure that delivers the IBM sponsored Special Events websites [2, 3, 4, 5]. We present the architecture of the monitoring system. We describe how monitoring data is collected, the method of distribution of the data, and the agents which receive and process the data. Specific case studies from the Sydney Olympics and the current Special Events sites are presented. Finally we present future enhancements for the continued support of IBM sponsored and hosted web sites.

RELATED WORK
Tivoli [7] software has been used to monitor availability and functionality of hardware components, complex software systems such as databases, and critical subsystems such as web servers. One powerful feature of Tivoli is the ability to trigger "repair" scripts to automatically correct well-defined problems such as server or database failure and log overflow. The problems detected by our system are generally too subtle for such an approach, requiring human judgment before action is taken. The ability to monitor the system remotely from laptops was also essential, but at the time Tivoli did not support remote access to the displays. Tivoli does not provide the customized views of subtle conditions that represent potential but not-yet realized failures. While some customized views can be built we required more complex hierarchical views of the information flow that could not be implemented in Tivoli. It was considered simpler to implement the views independently of Tivoli, also providing a more portable monitoring system, fully integrated with the publishing and content distribution system it supports.

Spong[8] was examined and extended as eSpong(not generally available) and is a key monitoring system used at the web serving facilities described in this paper to monitor conditions such as CPU load and server hits, Spong and eSpong present HTTP-based views of system loads, providing highly portable access to information in critical systems in the serving path. As well, HTTP based protocols provide a well-understood, secure solution to access through necessary obstacles such as firewalls.

BigBrother[9] is another system and network monitor. It lacks the hierarchical views we need for our system and would require some effort to customize it to present the data we produce effectively.

A number of systems have been built for monitoring of parallel and grid applications such as those by Vetter *et al* [10] and Miller *et al* [11], as well as for visualization of performance of parallel applications (Shaffer *et al*[12]). These systems are designed primarily for monitoring the performance of parallel applications, as opposed to monitoring workflow through multiple, heterogeneous systems.

SYSTEM ARCHITECTURE

The serving infrastructure is comprised of several serving complexes geographically distributed throughout the United States. Content for the serving complexes moves from its originator, through one or more steps, to its final destination. The number and configuration of the steps varies by event. At each step one or more components may provide monitoring data. An application specific probe gathers data from a component. Once gathered, this data is published to the distribution system which delivers it to subscribers. Figure 1 shows an abstract view of the systems, where M1 is delivering content to M2 and M3. On each machine producers gather and publish monitoring data into the distribution system. Consumers subscribe to selected monitoring data and format it for display.

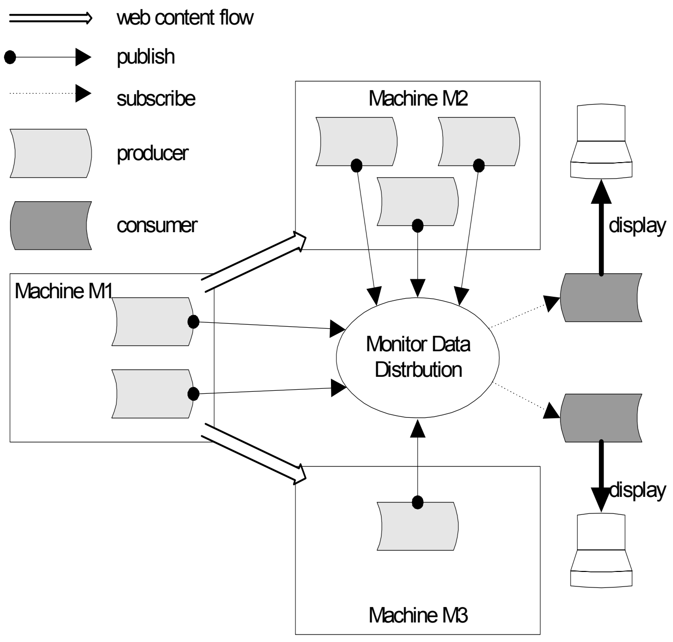


Figure 1: Monitoring System Architecture

GENERAL DESCRIPTION

The monitoring system consists of three independent elements: *producers* of data, *consumers* of data, and a *distribution* mechanism. Producers gather data, consumers receive data. The distribution mechanism coordinates the delivery of data from producers to the consumers. The data is encapsulated into a *monitor object.* The monitor object is designed to be independent of both the distribution mechanism and the consumer.

Monitor Objects

Monitoring data is wrapped into an opaque object called a *monitor object.* Monitor objects have properties associated with them that allow selection criteria to be applied by consumers. For our system we have three properties associated with any object: *event name*, (such as "www.wimbledon.org") *host name* (such as "server1.ibm.com") and *component name* (such as "Distributor" or "SaveFile"). This creates a selection space for use by consuming applications. Also provided by every monitor object are a *component type*, a *creation timestamp* and a function that can be used to convert the contents of the object into a key/value paired list. Beyond these base requirements, the component may add any data to the object that is relevant. The data in a monitor object is accessed by interrogating a self-describing object using a language specific mechanism. (Specifically, our implementation uses the Java™ reflection mechanism.) Thus, the data contained in the monitoring object can change independently of the distribution system and independently of the consumers.

Producers

Producers of monitoring data create and *publish* monitor objects at regular intervals and deliver them to the distribution system. Each producer extracts monitoring data that is specific to the component of the application that it is monitoring. Producers can be an integral part of the component. All producers use common facilities for creating and publishing monitoring objects. The process of publishing the monitoring data includes gathering the data, creating an appropriate object, setting properties on the object, and transforming the object to binary for transmission over the network. (Our implementation uses Java™ serialization to transform the object to binary.) The binary data and its properties are then published to the distribution system

Consumers

Consumers receive monitor data via subscription from the distribution system. After connecting to the distribution system, consumers specify selection criteria to control which monitoring objects they will receive. A consumer may choose to receive only data associated with a particular event, from a particular host, from a specific component, or any combination of the above. Consumers can also choose to receive all monitoring data.

Distribution

Monitoring data is distributed using a publish/subscribe model. Producers publish monitoring data and consumers subscribe to receive selected data. (Our mechanism is an implementation of Java™ Message Service (JMS)). From the perspective of the distribution system, monitoring data is simply opaque binary data. Producers and consumers interact only with the distribution system and thus are decoupled from each other. Producers can be added to the system, removed from the system and can change the type of object and data they are producing. Because consumers select objects to receive based on the properties associated with them, new producers in the system can be automatically detected by appropriate consumers. For instance, if a consumer is interested in all data associated with a particular event and a new producer is added which is generating data for that event, the consumer will automatically receive the new data. Consumers can also detect new types of objects, by using the language specific interrogation mechanism to identify that the object type has changed and that new data is available for display. Consumers are added and removed from the system whenever is appropriate.

QUEUES AND TASKS

The systems we monitor are composed of a series of cascading task/queue structures which form a hierarchy. Work flows through the system as tasks on queues. A task is represented by an object with associated methods. Tasks read and write data, do dependency analysis, web page assembly, or deliver completion notifications. A queue manages tasks which require execution. Each queue manages a particular kind of task. Tasks are created when external commands are received or as a result of executing other tasks. Tasks wait for available threads from a thread pool and execute. After the completion status of a task has been processed the task object is discarded. Queues manage waiting and executing tasks. See Figure 2 for a diagram showing the relationship of queues and tasks.

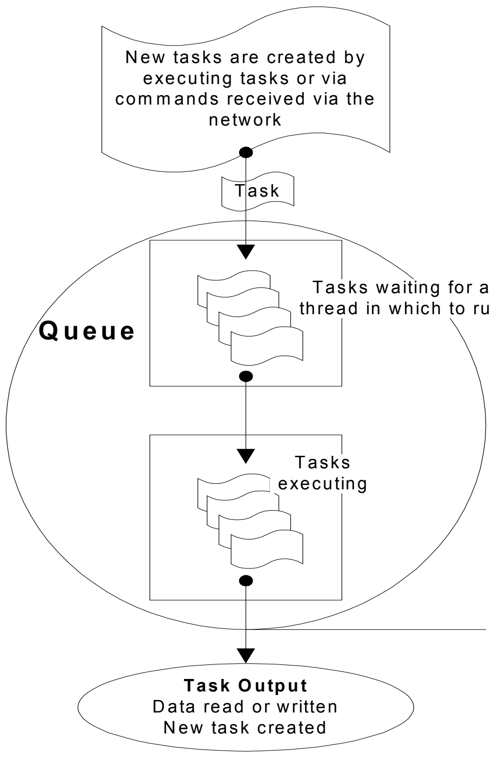

New tasks are created by executing tasks or via commands received via the network
Task
Queue
Tasks waiting for a thread in which to run
Tasks executing
Task Output
Data read or written
New task created

Figure 2: Queues and Tasks

Every queue generates monitoring data. Queues report on the number of tasks waiting and executing. For each interval queues present how many tasks enter the queue, complete successfully and fail. Each queue is a producer and creates a monitoring object containing queue specific data.

Queues form a workflow hierarchy, where the output of tasks on one queue is the addition of tasks onto queues below it in the hierarchy. Since each queue is a producer, monitoring data is generated from each node in the tree. Observing the monitoring data from each queue as a node within a branch presents an overall view of the health of the system. Figure 3 illustrates a tree of queues, where each queue in the tree generates a monitoring object.

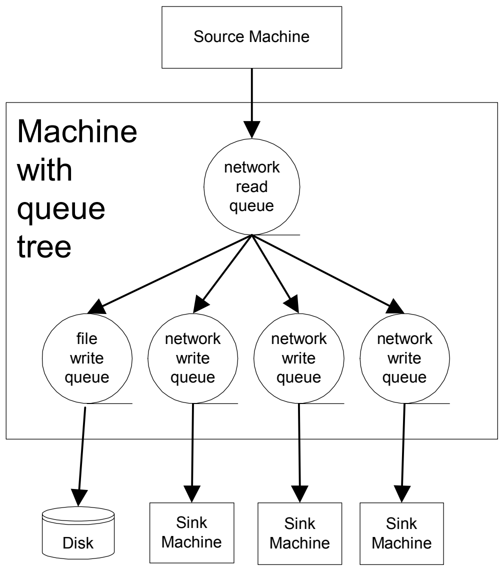


Figure 3: Queues on a machine

DATA PRESENTATION

Data presentation is constantly being revised and improved as experience dictates new approaches and as requirements change. The generation of data and the distribution of data has changed very little since its original design, but the mechanisms to present the data are under constant revision and modification. General types have emerged from direct experience.

Hierarchical Display of queues

System management staff requires assurance that work is properly flowing through the system and that one problem is not causing a ripple effect into other areas. The hierarchical display presents a high-level overview of the full system's status by showing the status for each queue. This view allows easy understanding of current workflow characteristics without other information that may be a distraction. This summary is also used to check for ripple effects when a problem is detected in other parts of the system. Figure 4 shows a workflow hierarchy where work is not significantly delayed at any point.

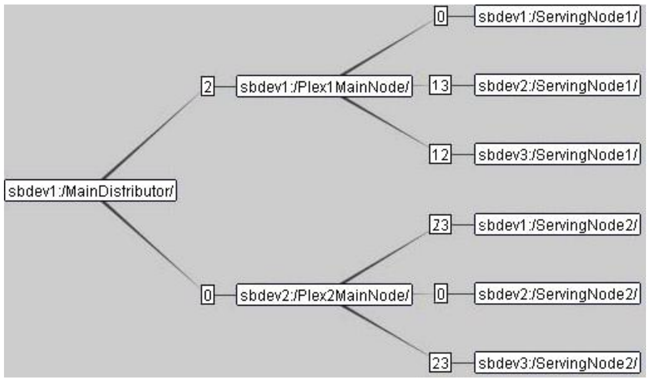
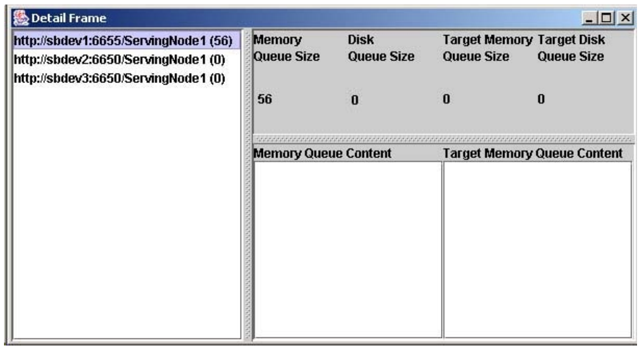


Figure 4: Hierarchical Queue Display

Custom Display

An enhancement to the original system allows monitoring objects to provide custom-built display mechanisms. This mechanism moves the presentation logic to the monitoring object and out of the consumer of the monitoring data. Consumers have a choice of whether to use this provided presentation logic. By providing a launch point for a monitoring object's custom-built presentation logic a consumer is able to present unbounded options. Figure 5 is an example of how a particular monitoring object chooses to display its data.



http://sbdev1:6655/ServingNode1 (56)
http://sbdev2:6650/ServingNode1 (0)
http://sbdev3:6650/ServingNode1 (0)

Memory Queue Size	Disk Queue Size	Target Memory Queue Size	Target Disk Queue Size
56	0	0	0

Memory Queue Content
Target Memory Queue Content

Figure 5: Custom Display of Monitoring Data

Table Display

Detailed information about queues in the system is displayed as a table, where each row in the table is a queue, and each column displays one aspect of the workflow through that queue. This detailed presentation allows more complete understanding of the workflow.

Figure 6 presents a segment of a table view for a set of queues. *"queued"* indicates the number of tasks waiting for execution and is the same value that can be found on the hierarchical view described earlier.

name	queued	active	bytes	objects
MainDistributor	15	20	16,956,142	501
Plex1Main	763	60	13,355,864	214
Plex2Main	225	25	15,600,900	438
Plex3Main	112	40	1,954,129	75
Plex4Main	21	40	445,932	68
Plex1Serve1	0	0	68,983	3
Plex1Serve2	3	10	3,947,485	160

Figure 6: Table view

HTTP Display

There are several situations where it is helpful to make monitoring data available through an HTTP server. We have created an HTTP interface which provides a subset of monitoring data to HTTP clients. This allows any kind of HTTP client to access and present monitoring data collected within our system. In particular, data can be sent through firewalls, to other forms of monitoring systems, and to non-Java based programs.

Statistical Logging

Monitoring data can be stored for offline analysis. To insure the accessibility of the data in the future it is converted to its key/value format and stored as an ascii log. Analysis tools read the log and do appropriate processing. Detailed charts and statistical analysis are created from the logged data which yields insights into the operation of the system, Figure 7 is an example of the kind of analysis which can be done.

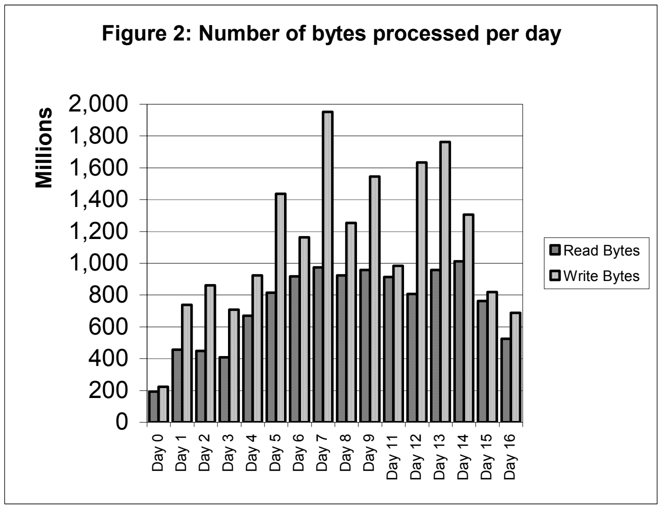


Figure 7: Number of bytes processed per day

EXPERIENCES

The original implementation and experience with the system occurred as part of hosting the Sydney 2000 Olympic Website [1]. The general design and flow of the system was re-used for monitoring the Events Infrastructure [2][3][4][5].

These two experiences are similar in that they both are primarily involved in distributing work through a complex, disperse system using queues. The quantity and variability of that work varied in the two systems. Monitoring of both systems is primarily concerned with ensuring that work is traveling through the system without significant delay.

Using the Hierarchical view

It is critical that the entire support staff be able to get a high-level view of the state of the system at any given moment. Refining the monitoring data into selected critical values and presenting that in a graphical way provided this. The hierarchical view is easy to comprehend so that everyone from content developers to webmasters and management, can quickly understand the health of the system.

One common problem during an event is work getting delayed within the system, forming high queue counts. Normally the hierarchical view of the system indicates that work is flowing efficiently through the system. This is indicated with the nominal, normally zero, numbers for each queue, which can be seen in Figure 8.

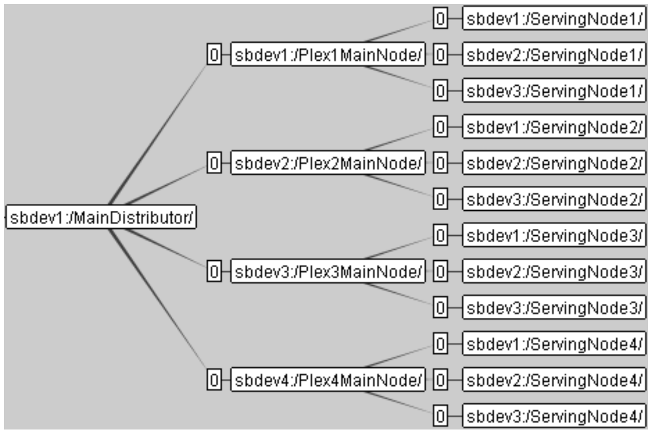


Figure 8: Error free view

When one of the numbers grows significantly, as seen in Figure 9, this is cause for concern. This view indicates that there are 56 tasks waiting. Technical webmasters understand that this number indicates the number of files that are waiting to be sent from sbdev1:/Plex1MainNode/ to sbdev1:/ServingNode1.

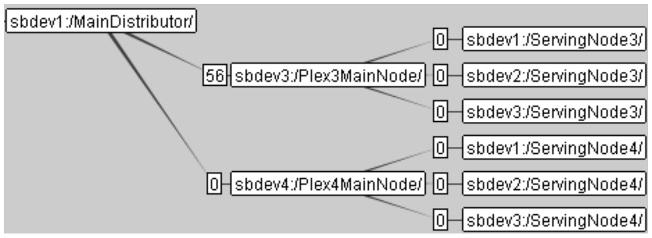


Figure 9: Possible problem

This high count may, or may not, indicate a problem in the system. Detailed views of the monitoring objects will provide the data that is needed to understand the cause of the delay. Here we present the different types of detailed views used.

Sydney 2000 Olympics

The Sydney 2000 Olympic Website [1] was hosted on a network of IBM RS/6000 SP2 complexes connected by a dedicated private network for inter-complex communication. All producers of monitoring data ran on AIX machines while consumers of monitoring data ran on a variety of platforms.

One important function of the monitoring systems was to provide data which could be used to generate daily update reports for management. Predicting the level of detail needed for these reports in advance is impossible. Because all the information was available through the system, modifying our first attempts at proper detail level was simply a matter of changing the analysis programs. The collection and distribution of data was not effected by changes in management's requirements regarding daily performance reporting.

Detailed understanding of the state of every queue in the system is a critical and challenging task. The table view allowed this to be accomplished with relative ease. It could be configured to present only the most relevant queues, or all queues.

During the event each queue tended to be in one of five states: quiet, active, slow, busy or down. Distinguishing these different states was necessary to determine when high queue counts were a concern, and what steps might be necessary to correct them.

Quiet Queues

Quiet queues have only a small quantity of work flowing through them. Figure 10 shows two queues, both of which have very few tasks waiting. The detailed information of "*bytes*" and "*objects*" indicate how many bytes and objects have been processed in the last minute. This rate of flow indication is very useful in understanding the current state of the queue. In this figure these numbers indicate that data is being processed by this queue, but the quantity of work is very small.

name	queued	active	bytes	objects
Plex1Serve1	0	0	68,983	3
Plex1Serve2	3	10	3,947,485	160

Figure 10: Quiet Queues

Active Queues

Active queues are receiving significant workload and are not overloaded. Active queues show large numbers for the bytes and objects per minute counts, and relatively low numbers for queued counts. Figure 11 shows several queues which are busy but not overloaded. They are each processing around 16 Megabytes of data in a minute, sending an average of 500 objects. We know, based on our configuration, that these values are near the theoretical maximums for the kinds of operations being performed by these queues, so we can feel confident that everything is working well. Although there are several large queued numbers, the largest is only about twice the number of objects which can be processed in a minute. This indicates that the queue is at most two minutes behind, which is acceptable in this case. At peak times a situation like this bears observing further, to ensure that the backlog is cleared out in a few minutes. It is always optimal to have all work processed in the same minute that it arrives.

Name	queued	active	Bytes	objects
MainDistributor	15	20	16,956,142	501
Plex1Main	1,284	60	16,898,348	685
Plex2Main	225	25	15,600,900	438

Figure 11: Active Queues

Slow Queues

Slow queues are not working at their expected capacity. Slowdowns generally indicate an undesirable system condition, like a networking problem. In Figure 12 we see that the system is busy, all queues are processing significant work each interval. The Plex1Main queue is significantly backed up. We observe that the rate of flow values for that queue are low, in particular, that it is only able to process 244 objects a minute. We expect to see this queue processing 500 or more objects in a minute. Unfortunately it is running at roughly half its expected rate and if that is not corrected we expect it will take well over an hour to work through the backlog. To calculate the expected time it takes to clear a backlog we divide the number of objects waiting, in this case 18,038, by the number of objects processed in a minute. In case the result is 74 minutes. Even at 500 objects per minute this delay is only improved to a half hour. This example is a severe case of the effect of combinations of network problems and heavy workload.

Name	queued	active	bytes	objects
MainDistributor	71	21	9,751,762	514
Plex1Main	18,038	60	6,094,036	244
Plex2Main	0	0	12,756,023	641

Figure 12: Slow Queue

Busy Queues

A queue that is receiving more work than it has workload capacity becomes backed up and busy. A busy queue is an example of a potentially subtle situation that may indicate failure of some component or simply indicate a spike in workload. The ability for all queues to keep up with the workload demand was a significant focus of the entire support team. As long as a large quantity of work continues to flow into the queue, the queue will not be able to recover. When a queue is falling behind and unable to process work in a timely manner, a great deal of focus and detailed understanding of the situation is required. Most of the time these situations are caused by a sudden heavy load of work being added to the system, or a networking problem. Once in the over loaded state a queue is monitored very closely until it is once again in a satisfactory state. During this time content developers and management required regular updates. In Figure 13 Plex1Main is processing work at a reasonable speed, but it is still almost 15 minutes behind its workload (8,375 waiting objects divided by 604 objects per minute). Further analysis indicates that no other queues are overloaded and the queue sending work to Plex1Main, MainDistributor, is experiencing a light load at the moment. Therefore, we expect that this problem will not last longer than the 15 minutes it will take to clear the current backlog. This queue will be closely watched until it has recovered.

name	queued	Active	bytes	objects
MainDistributor	0	0	2,052,747	60
Plex1Main	8,375	60	12,997,655	604
Plex2Main	0	0	2,069,651	61
Plex3Main	600	40	1,625,933	411
Plex4Main	2	40	3,547,756	457
Plex1Serve1	46	10	1,625,933	411
Plex1Serve2	29	10	17,865,748	527

Figure 13: Busy Queue

Down

Down queues are no longer able to process work. This is often because a machine is down so it is important that this fact be highlighted immediately. If work is flowing to that machine it will get delayed, resulting in large queued values. During quiet times a machine's down status may not result in high queued values immediately. We found that it was very helpful to highlight the fact that a machine was down. In Figure 14 Plex2Main is down because its active count and per minute byte and object counts are all zero. This means that no work is flowing through this queue. The queued number alone would not necessarily indicate that the queue is in trouble, but the detailed information indicates that there is a significant problem that must be corrected.

name	queued	active	Bytes	objects
MainDistributor	0	0	28	1
Plex1Main	0	0	28	1
Plex2Main	**670**	**0**	**0**	**0**

Figure 14: Down Queue

Events

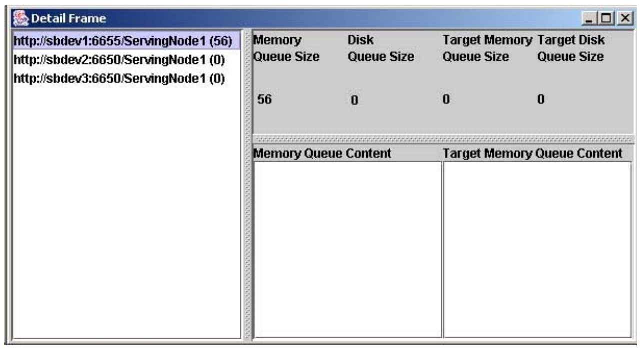
The events infrastructure is hosted on a network of Netfinity X86 machines connected by a virtual private network. All producers of monitoring data run Linux while consumers of monitoring data run on a variety of platforms.

The events infrastructure began using the monitoring system in January 2002 for the Australian Open Tennis tournament [2] and the system has been used for every event since [3, 4, 5]. Some details regarding the form of the monitoring object were improved, but the generation and distribution of the objects remain functionally similar to the original design. The events being monitored are high profile and problems must be resolved immediately. This kind of short duration, high intensity web event drove the design of the original monitoring system.

Queue backup resolution

The nature of the detailed data available about the queues in this system is different than the original table data. Referring to Figure 9 on page 5 we see that the hierarchical view shows a potential backup of 56 tasks on Plex3MainNode. When this kind of queue backup occurs the delayed tasks will be put into different states based on current system characteristics. The number of tasks in each state is presented in the detailed view for the queue.

Figure 15 illustrates the scenario where all the tasks are in memory. This indicates the network connection between the machines is functioning without error. The machine may be overloaded with other activity and currently not able to process the files in a timely manner or perhaps the files are being processed at the normal rate but an unusually large number were received. Distinguishing these two types of problems requires understanding how many files were sent in the last interval.


Detail Frame

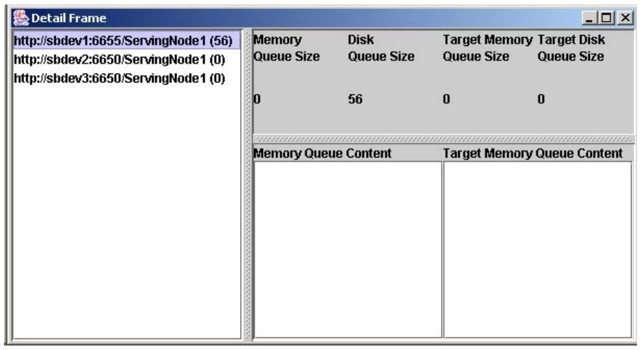
http://sbdev1:6655/ServingNode1 (56)
http://sbdev2:6650/ServingNode1 (0)
http://sbdev3:6650/ServingNode1 (0)

Memory Queue Size	Disk Queue Size	Target Memory Queue Size	Target Disk Queue Size
56	0	0	0

Memory Queue Content | Target Memory Queue Content

Figure 15: Backlog in Memory

In Figure 16 we see that the tasks are being saved on disk. This indicates that there is a communication error. The tasks are moved to disk so the backup resulting from the communication error does not exhaust memory.


Detail Frame

http://sbdev1:6655/ServingNode1 (56)
http://sbdev2:6650/ServingNode1 (0)
http://sbdev3:6650/ServingNode1 (0)

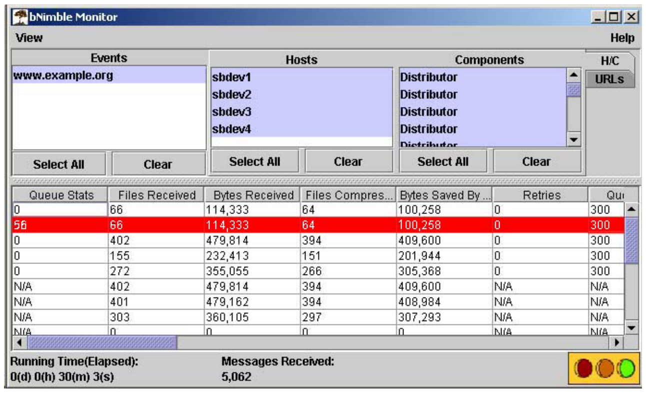
Memory Queue Size	Disk Queue Size	Target Memory Queue Size	Target Disk Queue Size
0	56	0	0

Memory Queue Content | Target Memory Queue Content

Figure 16: Backlog on Disk

Communication failure between two nodes can have many root causes, but most often indicates either a networking error or an entire machine failure. It is important to be able to isolate these very different kinds of problems quickly. Machine failures are detected when a machine fails to send monitoring data in the expected interval. In Figure 17 a row in the table is highlighted because the machine associated with that data has not sent data in the expected time period. This highlighting warns that the data for that machine is stale. Since this is the only machine highlighted, this indicates that the machine has failed and needs immediate attention. With this knowledge we can determine that the queue backup is being caused by the outage of machine Plex3MainNode.

This automatic detection of machine failure could also be used to automate actions such as rerouting traffic around the machine, or recycling the machine. In our system we have not yet implemented this level of problem detection and resolution.



Queue Stats	Files Received	Bytes Received	Files Compres...	Bytes Saved By ...	Retries	Qu
0	66	114,333	64	100,258	0	300
56	66	114,333	64	100,258	0	300
0	402	479,814	394	409,600	0	300
0	155	232,413	151	201,944	0	300
0	272	355,055	266	305,368	0	300
N/A	402	479,814	394	409,600	N/A	N/A
N/A	401	479,162	394	408,984	N/A	N/A
N/A	303	360,105	297	307,293	N/A	N/A
N/A	0	0	0	0	N/A	N/A

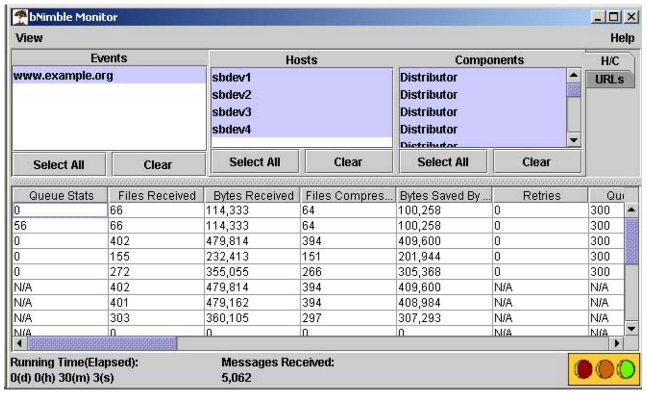
Running Time(Elapsed): 0(d) 0(h) 30(m) 3(s)

Messages Received: 5,062

Figure 17: Machine Failure

When multiple machines fail to report data the most likely cause problem such as a networking problem between the machines and a common point of distribution.

Figure 18 depicts the situation where there is a communication problem between the nodes. All data is being reported through the monitoring system, but data is not flowing between two particular machines. Investigation into the communication network between these nodes is required.



Queue Stats	Files Received	Bytes Received	Files Compres...	Bytes Saved By ...	Retries	Qu
0	66	114,333	64	100,258	0	300
56	66	114,333	64	100,258	0	300
0	402	479,814	394	409,600	0	300
0	155	232,413	151	201,944	0	300
0	272	355,055	266	305,368	0	300
N/A	402	479,814	394	409,600	N/A	N/A
N/A	401	479,162	394	408,984	N/A	N/A
N/A	303	360,105	297	307,293	N/A	N/A
N/A	0	0	0	0	N/A	N/A

Running Time(Elapsed): 0(d) 0(h) 30(m) 3(s)

Messages Received: 5,062

Figure 18: No Machine Failure

FUTURE ENHANCEMENTS

Improved error analysis mechanisms

A relational database is a powerful tool for detailed analysis of monitoring data. A consumer could be developed which would insert data into a database indexed by its timestamp. An analysis program could then do detailed problem analysis by running specially designed queries against the data.

Improved error reporting

Although the producers in the system currently publish one set of data each interval, there are many situations that would benefit from a less structured publishing approach. One such approach is to produce a specialized form of monitoring data for error situations. When an error occurs it is essential to get the information about that error into the monitoring system as quickly as possible in order to avoid unnecessary delay and reflect errors immediately.

SUMMARY

A system for monitoring complex queue-based systems using publish/subscribe for data distribution has been described. Workflow is controlled through a hierarchical organization of queues. Presentation of the data provides an immediate high-level hierarchical view of the queues and the work flowing through those queues. When problems are suspected, more detailed views are readily available. Data describing the workload of each queue takes advantage of language-specific features such as reflection to provide self-describing structures as well as encapsulation of data-specific views that do not have to be programmed into either the producer or the consumer of the data. Although our implementation is on a queue-based system, the monitoring system itself contains no dependencies on the structure of the underlying system and is readily adaptable to any underlying structure or topology. This system has proven to be extremely powerful and flexible.

ACKNOWLEDGMENTS

Several people have contributed to this work including Sandy Cash, Paul Dantzig, Cameron Ferstat, Ed Geraghty, Arun Iyengar, Herbie Pearthree, and Paul Reed.

REFERENCES

1. Sydney 2000 Olympic Website www.olympics.com from September 15 through October 1, 2000. Site no longer active.
2. Australian Open 2002 Website. www.ausopen.org from January 14, 2002 through Januray 27, 2002. Site no longer active.
3. Masters 2002 Website. www.masters.org
4. French Open 2002 Website. www.rolandgarros.org
5. Wimbledon 2002 Website. www.wimbledon.org
6. Jim Challenger, Arun Iyengar, Karen Witting, Cameron Ferstat, and Paul Reed. A Publishing System for Efficiently Creating Dynamic Web Content. In *Proceedings of IEEE INFOCOM 2000*, March 2000.
7. Tivoli Software. http://www.tivoli.com/
8. Spong. http://sourceforge.net/projects/spong
9. BigBrother. http://bb4.com/
10. Jeffery S Vetter and Daniel A Reed, "Real Time Performance Monitoring, Adaptive Control, and Interactive Steering of Computational Grids," *The International Journal of High Performance Computing*

Applications, Winter 2000, Volume 14, No 4, pp 357-366, April, 2002

11. Barton P. Miller, Mark D. Callaghan, Jonathan M. Cargille, Jeffrey K. Hollingsworth, R. Bruce Irvin, Karen L. Karavanic, Krishna Kunchithapadam and Tia Newhall, "The Paradyn Parallel Performance Measurement Tools," *IEEE Computer*, Volume 28 No 11, pp 37-46, November 1995
12. Eric Shaffer, Shannon Whitmore, Benjamin Schaeffer, and Daniel A Reed."Virtue: Immersive Performance Visualization of Parallel and Distributed Applications", *IEEE Computer*, December 1999