MONITORING DISTRIBUTED SYSTEMS

A Publish/Subscribe Methodology and Architecture

Karen Witting, James Challenger, Brian O'Connell
IBM T. J. Watson Research Center and IBM Global Services Special Events

Abstract: 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 monitoring objects created by the various heterogeneous, distributed components are distributed to any application choosing to present monitoring information.

Key words: Publish-Subscribe, Monitoring, Distributed Systems, Workflow Monitoring, Queue Monitoring, High Volume Web Serving, Content Management

1. INTRODUCTION

Systems comprised of a large number of interacting components require a highly flexible monitoring system. Modern, high volume web sites and their supporting infrastructure are an example of this kind of large system. "24x7" availability requires extremely flexible monitoring to cope with ever-changing hardware and software components. New types of components may be needed, and previously active components may be removed from the system. Any particular component may provide different types of monitoring data over time.

In this paper we describe the system designed and implemented to monitor flows within the publishing and content distribution systems for the Sydney 2000 Olympic Website [1] [3] and the IBM sponsored Special Events websites [2].

2. SYSTEM DESCRIPTION AND ARCHITECTURE

The serving infrastructure is comprised of several geographically distributed complexes. Content for the serving complexes flows from its originator, through one

or more stages, to its final destination. The number and configuration of the stages varies by event. An application specific probe gathers monitoring data from the components at each stage. This data is published to the distribution system which delivers it to subscribers. Consumers subscribe to selected monitoring data and present it in various views for display. *Figure 1* shows an abstract view of the flow, where M1 is delivering content to M2 and M3.

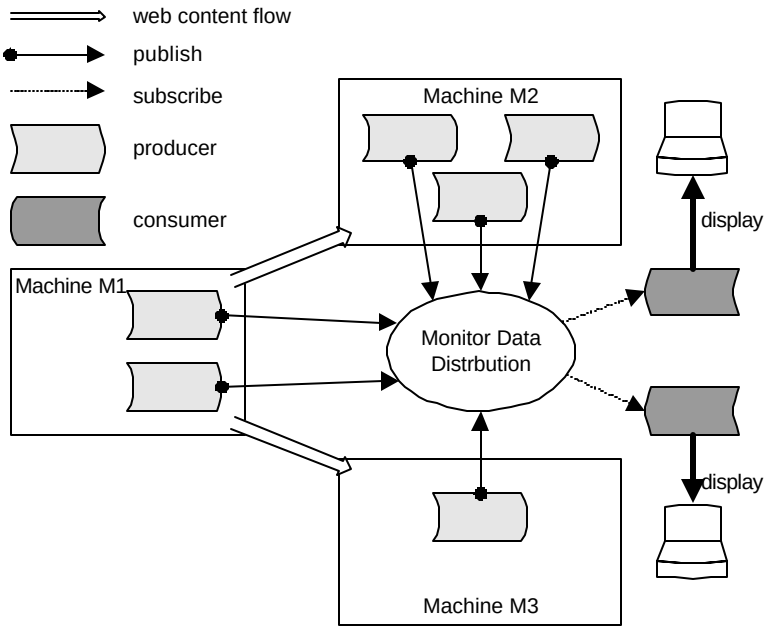

Figure 1: Monitoring System Architecture

The monitoring system consists of three main elements: *producers*, *consumers*, and a *distribution* mechanism. Producers gather and send out monitoring data, consumers receive data. The distribution mechanism coordinates the delivery of the data. The monitoring data itself is encapsulated into an opaque, self-describing *monitor object* which is designed to be independent of both the distribution mechanism and the consumer.

Monitor objects have properties that allow selection criteria to be applied by consumers. Three main properties are associated with every object: *event name*, ("www.wimbledon.org") *host name* ("server1.ibm.com") and *component name* ("SaveFile") to create a selection space for use by consuming applications. Beyond these base properties, a component may add any relevant data to the object. Data is accessed by interrogating the self-describing object allowing it to change independently of both distribution system and consumers.

Producers create and then *publish* monitor objects to the distribution system. Each producer extracts monitor data that is specific to the monitored component. All producers use common facilities for creating and publishing monitoring objects.

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

From the perspective of the *distribution system*, monitoring data is opaque. Producers and consumers interact only with the distribution system and thus are decoupled from each other. Because consumers are aware only of the self-describing *monitor objects* (and thus not explicitely aware of producers), producers can be added to or removed from the system and can change the type of object and data they are producing.

The systems we monitor are composed of a series of cascading hierarchically organized task/queue structures. Work flows through the system as tasks on queues. Every queue collects data about things like the number of tasks waiting and executing. Each queue is a producer and publishes a monitoring object containing the data collected about the queue.

Queues form a workflow hierarchy, where the output of tasks on one queue results in 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 hierarchy.

3. EXPERIENCES

The original implementation and experiences with the system occurred while hosting the Sydney 2000 Olympic Website [1]. The general design and flow of the system was re-used for monitoring the Events Infrastructure [2]. These two experiences are similar in that they both are primarily involved in distributing work via queues and consist primarily of ensuring that work travels through the system without significant delay. Our methodology for monitoring the systems is a product of our experiences running these sites.

The Sydney 2000 Olympic Website [1] was hosted on a network of IBM RS/6000 SP2 complexes interconnected by a high-speed dedicated private network. Producers ran on AIX machines; consumers ran on a variety of platforms. The events infrastructure is currently 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.

A key function of the monitoring systems is to provide data for management reports. Predicting the level of detail needed for these reports in advance is impossible. A novel hierarchical view of queues and tasks showing workflow enabled rapid identification of potential bottlenecks and provided a high level of flexibility in identifying and reporting problems. Detailed information about queues in the system was displayed in tabular views. When high queue counts are a concern the tabular views enable rapid diagnosis and correction.

Queues with work flowing through them were classified as *active*, *slow*, or *busy*. An *active* queue is receiving significant workload but is not overloaded. Active queues show large numbers of tasks flowing through them but relatively low

numbers for queued counts. That is, an *active* queue has high throughput but low queue lengths, which indicates that tasks in that queue have very little wait time.

A *slow* queue is not working at its expected capacity. Slowdowns generally indicate an undesirable system condition such as a networking problem. Throughput is lower than expected, generally resulting in excessive queue wait time. A *slow* queue has low throughput and may or may not have long queue lengths.

A *busy* queue is receiving more work than it has workload capacity for. A busy queue could be indicative of component failure or simply indicate a spike in workload. A *busy* queue has both high throughput and long queue lengths. It is usually acceptable for a queue to be *busy* for some period of time (for example, as a result of a load spike) but extended *busy* conditions could indicate subtle system failures.

The ability for all queues to keep up with the workload demand is a significant focus of the entire support team. 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, temporary, increase of work being added to the system, or a networking problem.

4. FUTURE ENHANCEMENTS

Enhancements include more sophisticated tools for log playback and database driven post-event analysis tools. Failure detection can be difficult; more specialized monitors to detect and rapidly report highly critical failures, increasing the granularity of reporting would be useful. Integration with SNMP and other standard protocols would allow other monitor clients to benefit from our queue based collection and reporting scheme.

5. ACKNOWLEDGMENTS

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

6. REFERENCES

1. Sydney 2000 Olympic Website www.olympics.com from September 15 through October 1, 2000.
2. Selected IBM Sponsored Web sites: www.ausopen.org, www.masters.org, www.rolandgaros.org, www.wimbledon.org.
3. Challenger, Jim, et al., A Publishing System for Efficiently Creating Dynamic Web Content. In *Proceedings of IEEE INFOCOM 2000*, March 2000.