Self-optimization of Clustered Message-Oriented Middleware

Christophe Taton[1] Noël De Palma[1] Jérémy Philippe[1] Sara Bouchenak[2]

[1]Institut National Polytechnique de Grenoble, Grenoble, France
[2]Université Grenoble I, Grenoble, France

E-mail: {Christophe.Taton,Noel.Depalma,Jeremy.Philippe,Sara.Bouchenak}@inrialpes.fr

Abstract

Today's entreprise-level applications are often built as an assembly of distributed components that provide the basic services required by the application logic. As the scale of these applications increases, coarse-grained components will need be decoupled and will use message-based communication, often helped by Message-Oriented Middleware or MOMs.

In the Java world, a standardized interface exists for MOMs: Java Messaging Service or JMS. And like other middleware, some JMS implementations use clustering techniques to provide some level of performance and fault-tolerance. One such implementation is JORAM, which is open-source and hosted by the ObjectWeb consortium.

The full version of this paper intends to describe performance modeling of various clustering configurations and validate our model with performance evaluation in a real-life cluster. In doing that, we observed that the resource-efficiency of the clustering methods can be very poor due to local instabilities and/or global load variations.

Keywords: *MOM, JMS, Autonomic management, Self-optimization.*

1 Background: Java Message Service (JMS)

JMS is part of Sun's J2EE platform. It provide a programming interface (API) to interconnect different applications through a messaging middleware. The JMS architecture identifies the following elements:

- **JMS provider**: an implementation of the JMS interface for a Message Oriented Middleware (MOM). Providers are implemented as either a Java JMS implementation or an adapter to a non-Java MOM.
- **JMS client**: a Java-based application or object that produces and/or consumes messages.
- **JMS producer**: a JMS client that creates and sends messages.
- **JMS consumer**: a JMS client that receives messages.
- **JMS message**: an object that contains the data being transferred between JMS clients.
- **JMS queue**: a staging area that contains messages that have been sent and are waiting to be read. As the name queue suggests, the messages are delivered in the order they are sent. A message is removed from the queue once it has been read.
- **JMS topic**: a distribution mechanism for publishing messages that are delivered to multiple subscribers.
- **JMS connection**: A connection represents a communication link between the application and the messaging server. Depending on the connection type, connections allow users to create sessions for sending and receiving messages from a queue or topic.
- **JMS session**: Represents a single-threaded context for sending and receiving messages. A session is single-threaded so that messages are serialized, meaning that messages are received one-by-one in the order sent.

For our experiments we chose JORAM (Java Open Reliable Asynchronous Messaging). It is open source software released under the LGPL license which incorporates a 100% pure Java implementation of JMS. JORAM adds interesting extra features to the JMS API such as the clustered queue mechanisms. The following section describes the mechanism of queue clustering.

2 Clustered Queues

The clustered queue feature provides a load balancing mechanism. A clustered queue is a cluster of queues (a given number of queue destinations knowing each other) that are able to exchange messages depending on their load.

Each queue of a cluster periodically reevaluates its load factor and sends the result to the other queues of the cluster. When a queue hosts more messages than it is authorized to

Fourth International Conference on Autonomic Computing (ICAC'07)
0-7695-2779-5/07 $20.00

do, and according to the load factors of the cluster, it distributes the extra messages to the other queues. When a queue is requested to deliver messages but is empty, it requests messages from the other queues of the cluster. This mechanism guarantees that no queue is hyper-active while some others are lazy, and tends to distribute the work load among the servers involved in the cluster. The figure above shows an example of a cluster made of two queues. An heavy producer accesses its local queue (queue 0) and sends messages. The queue is also accessed by a consumer but requesting few messages. It quickly becomes loaded and decides to forward messages to the other queue (queue 1) of its cluster, which is not under heavy load. Thus, the consumer on queue 1 also gets messages, and messages on queue 0 are consumed in a quicker way.

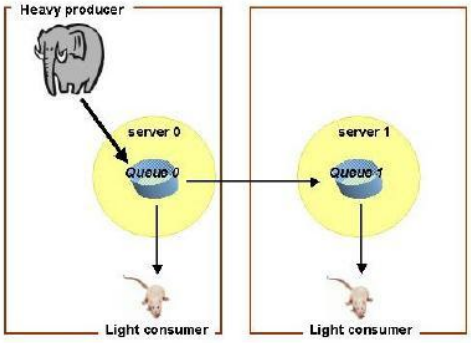

Figure 1. A queue cluster

3 Related work

We describe a self-optimization mechanism in the case of a queue clustering technique. Some projects only analyse JMS performance whereas others target the self-optimization of J2EE infrastructure but do not focus on MOM self-optimization.

Regarding JMS performance, [3] provides an analysis of the throughput performance of JMS Using Websphere-MQ. [4] analyses a specific performance problem: The Message Waiting Time for the Fiorano-MQ Server. [2] describes a QoS Evaluation of JMS, it examines the impact of JMS attributes on performance.

About self-optimization, several projects which have addressed the issue of element management in a cluster of machines. In these projects, the software components required by any application are all installed and accessible on any machine in the cluster. Therefore, allocating additional resources to an application can be implemented at the level of the protocol that routes requests to the machines (Neptune [5] and DDSD [7]). Some of them (e.g. Cluster Reserves [1] or Sharc [6]) assume control over the CPU allocation on each machine, in order to provide strong guarantees on resource allocation.

4 Conclusion and future work

Providing a scalable and efficient Message Oriented Middleware is an important topic for today's computing environments. This paper analyses the performance of a Message Oriented Middleware and proposes a self-optimization algorithm to improve the efficiency of the MOM infrastructure. We describe (i) the key parameters impacting the performance of the MOM and (ii) the rules that control these parameters for optimal prformances. This paper also presents an evaluation that shows the impact of these parameters on the MOM. Currently, the control loop has a very basic actuator to lead a client connection to a specific queue. The advantage of this actuator is its simplicity. However, the control loops cannot reconfigure the client connection during a session. Part of our future work is about providing a more powerful actuator. This actuator will provide the control loop with the ability to migrate a client connection when necessary. This requires a mechanim to move session data on other queue.

References

[1] M. Aron, P. Druschel, , and W. Zwaenepoel. Cluster Reserves: a mechanism for resource management in cluster-based network servers. In *International Conference on Measurement and Modeling of Computer Systems (ACM SIGMETRICS-2000)*, Sant Clara, CA, June 2000.

[2] S. Chen and P. Greenfield. Qos evaluation of jms: An empirical approach. In *HICSS '04: Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 9*, page 90276.2, Washington, DC, USA, 2004. IEEE Computer Society.

[3] R. Henjes, M. Menth, , and C. Zepfel. Throughput performance of java messaging services using websphereMQ. In *5th International Workshop on Distributed Event-Based Systems (DEBS)*, Lisboa, Portugal, 7 2006.

[4] M. Menth and R. Henjes. Analysis of the message waiting time for the fioranoMQ JMS server. In *26th International Conference on Distributed Computing Systems (ICDCS)*, Lisboa, Portugal, 7 2006.

[5] K. Shen, H. Tang, T. Yang, and L. Chu. Integrated resource management for cluster-based internet services. In *5th USENIX Symposium on Operating System Design and Implementation (OSDI-2002)*, Dec. 2002.

[6] B. Urgaonkar and P. Shenoy. Sharc: Managing CPU and network bandwidth in shared clusters. *IEEE Transactions on Parallel and Distributed Systems*, 15(1), 2004.

[7] H. Zhu, H. Ti, and Y. Yang. Demand-driven service differentiation in cluster-based network servers. In *20th Annual Joint Conference of the IEEE Computer and Communication Societies (INFOCOM-2001)*, Anchorage, AL, Apr. 2001.