http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 10 (1): 61-70, 2011
ISSN 1812-5638

Modeling and Performance Evaluation of Message-oriented Middleware with Priority Queuing

S.S. Alwakeel and H.M. Almansour
College of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia

Abstract: The main component in distributed computing is the communication middleware that is used to connect the system's heterogeneous components and to manage the interactions between these components. In Remote Procedure Call (RPC) middleware, the client uses synchronous methods to communicate with the service provider. Message-Oriented Middleware (MOM) uses messaging to communicate asynchronously between the client and the service provider. In this research, we designed and developed an analytical M/M/1 model with first in first out policy and a priority queuing model for evaluating the performance of MOM and RPC middleware. The models compare the performance of RPC to MOM with priority queuing and analyze the throughput of their communication paradigms. Various input parameters are used to determine the optimal environment setting of the paradigms that achieve the maximum throughput performance. The research results prove that by using MOM with priority queuing, system throughput can be improved. This, in turn, will enhance the performance of many middleware practical applications such as middleware in wireless sensor networks and in applications of middleware for pervasive computing.

Key words: MOM performance, RPC middleware performance, MOM with priority queuing, middleware performance modeling, MOM throughput analysis

INTRODUCTION

A middleware system is a middle layer between the network operating system and the distributed application. It implements the session and presentation layer of the OSI reference model (Verissimo and Rodrigues, 2001; Myerson, 2002). Its main goal is to enable communication between distributed components. Middleware makes programming for distributed computing much easier. It acts like an Operating System (OS) for distributed computing architecture. It also provides transparency for the application that uses it and those applications do not need to know which hardware or operating system is being used on the other side.

Usually, the middleware has a common interface that can be used by the distributed application. From the OS perspective, it can communicate with various operating systems using those OS's own interfaces. This helps the distributed application to communicate with a wide variety of applications regardless of OS type.

Based on this, five different categories of middleware now exist (Verissimo and Rodrigues, 2001; Myerson, 2002): Remote Procedure Call (RPC), Message-Oriented Middleware (MOM), object-oriented middleware, transactional middleware and application servers.

Remote procedure call is similar to normal procedure with the exception that the call procedure is executed remotely and the result is then returned to the caller. The main disadvantage in RPC is the lack of asynchronous operation support.

Message-oriented middleware supports the communication between the distributed components via message-passing client components, which send messages containing requests for service execution, together with parameters, to a server component across the network. The server responds with a reply message containing the result of the service execution. Examples of MOM implementation are: IBM Web Sphere MQ and Microsoft Message Queuing (MSMQ). The main advantages of MOM include asynchronous communication, which obviates the need for the application to wait for the result; it just sends the message and performs its asynchronous operations. Another advantage of MOM is its Offline Mode, where the client and service provider can go offline at any time without losing any messages sent. Once they are up again they can retrieve all pending messages. In addition, MOM allows for an event-driven system; the administrator can also rate the clients based on their Service Level Agreement (SLA) and can request various levels of

Corresponding Author: Dr. Sami Saleh Alwakeel, College of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia Tel: +96614677101

Quality of Service (QoS). However, due to the above, MOM systems require resource-rich devices (Verissimo and Rodrigues, 2001; Mascolo *et al*., 2002), especially in terms of memory in which to store persistent queues of messages received but not already processed. A detailed discussion of middleware related technologies and history can be found in (Verissimo and Rodrigues, 2001; Myerson , 2002).

Literatures on middleware paradigms usually belong to the following major categories:

- Distributed computing
- Performance study and QoS of middleware systems
- Security of middleware systems
- Mobile computing middleware
- Middleware in Ad-Hoc and Sensor networking

In a research article by Krafzig *et al*. (2004) a review is given for existing concepts of middleware implementation in distributed computing. It describes different communication models used in distributed computing. Vinoski (2002) describes different classifications of middleware and the challenges they face in implementation.

One of the main challenges in middleware is performance. Many researches focus on communication middleware modeling and its performance, but different applications have different requirements and these affect performance. Menascwe (2005) discusses the performance of two communication models for distributed applications, RPC and MOM. It provides a quantitative approach for comparing RPC- and MOM-based solutions. It also analyzed an MOM solution that uses an M/M/1 queuing model. Studies by Tran *et al*. (2002) and by Maheshwari and Pang (2005) test the performance of commercial MOM through measuring various parameters such as Quality of Service (QoS), effectiveness (message per second) and time of delivery for batch of messages. In a study by Sachsa *et al*. (2009) a methodology is presented for the performance evaluation of MOM platforms using the SPECjms2007 standard benchmark. In addition to providing a standard workload and metrics for MOM performance, the benchmark in that study provides a flexible performance analysis framework that allows users to tailor the workload to their requirements.

Banavar *et al*. (1999) discussed MOM applications, implementation and open research areas such as flow-graph model, scalability, distributed implementation, message reliability, message ordering and security. Saiedian *et al*. (2002) proposed a framework for evaluating the most widely used Distributed Object Model (DOM) middleware. A number of important managerial items such as cost, personnel, technology resources, training, enterprise changes and time constraints have been identified, explained and justified as the evaluation criteria. A case study of a production web-based system is also presented to demonstrate the use of the framework. Yang *et al*. (2009) present the design and implementation of an overlay-based messaging system that can manage the end-to-end QoS in wide-area publish/subscribe communications, based on the application requirements. This system actively exploits the diversity in the network paths and redirects the traffic over links with good quality.

MOM security acknowledges the fact that there is no application that controls it. Protecting physical resources and controlling message flow security and privacy are the key areas in MOM security. These cases have been overviewed in Banavar *et al*. (1999). On the other hand, Lingel (2001) identifies and examines three main solutions with respect to security: authentication, authorization and encapsulation. The techniques discussed are simple, such as username and password, or advanced, such as Public Key Infrastructure (PKI).

The implementation of middleware in mobile and pervasive computing is also a current research interest. Mascolo *et al*. (2002) review various challenges in using middleware in mobile computing. Low power, slow CPU and little memory are examples of such challenges. It classifies middleware solutions according to the computational load they require in execution, the communication paradigms they support and the kind of context representation they provide to the applications.

Blumenthal *et al*. (2003) described the different challenges faced in using middleware in Wireless Sensor Networks (WSN). The study defines the characteristics needed in middleware for use in sensor networks. In a study by Schiele *et al*. (2008) the design of an energy-aware middleware for pervasive computing is described.

The objective of this research is to develop and study MOM with priority queuing and to determine its optimal usage environment. An approximate queuing model is developed in order to study the MOM paradigm. System throughput is used as the key parameter in evaluating the performance of the target model (MOM with priority queuing). Another objective is to compare the research model with other models, specifically RPC and MOM with M/M/1 queuing.

RPC AND MOM ARCHITECTURES AND QUEUING MODELS

Here, we describe the architectures of Remote Procedure Call (RPC) and Message-Oriented Middleware and present the queuing models developed to evaluate

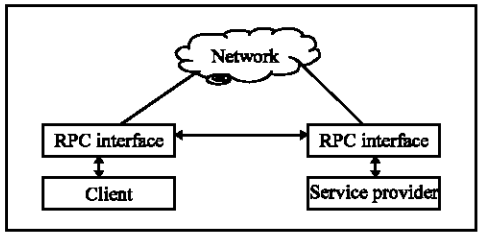

Fig. 1: RPC architecture

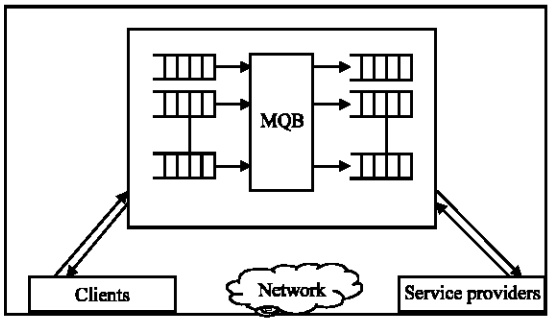

Fig. 2: MOM architecture

their performance. RPC uses synchronous operations to accomplish its tasks. A server with RPC uses the RPC interface to define its call procedures. The call is routed through the network to another application where it is executed and the result is then returned to the caller (Fig. 1). Thus, remote procedures can be called as if they where local ones.

MOM operates on the principles of message passing and queuing. Its architecture is composed of three main components: Client, service provider and the Message Queue Broker (MQB) (Fig. 2).

The Client is an application that needs to conduct some processing in a remote site (or to obtain results from one). After sending the message, the application can perform any asynchronous code until it receives the results. The service provider is a server that accepts messages and processes them according to the required service. The server sends the results to the MQB, to be returned to the client. The MQB is at the heart of MOM; it does the routing and filtering according to a user-defined access-list and it takes messages sent from the client and routes them to their destinations. This is why many researchers have tried to find the best implementation and optimal configuration for maximizing the performance of the MQB. Communication between the client and the service provider is asynchronous. Also, they do not need to be available at the same time to communicate with each other; the MQB will send the messages to each one when it is available.

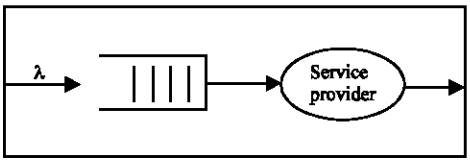

Fig. 3: RPC queue model

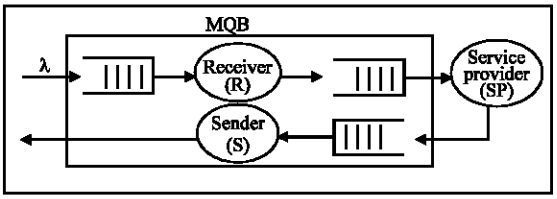

Fig. 4: MOM queuing model

To study the performance of RPC and MOM and to evaluate their throughput, two queuing models were developed and these are shown in Fig. 3 and 4. Remote Procedure Call (RPC) uses M/M/1 FIFO queuing. All clients that call the service provider are on hold until it processes their call and returns the result (Fig. 3, RPC Queue Model).

In Fig. 4, we show the MOM queuing model, which serves many applications and service providers. For the purpose of this research, we made two assumptions. First, we assume that there is only one receiving queue for all applications. Second, there is only one service provider. The queue can be either FIFO or a priority queue (Fig. 4 MOM Queuing Model).

As shown in Fig. 4, the MQB structure has two main servers: the Receiver and the Sender. The Sender in MQB has two scenarios: in the blocking scenario, the client application has to make a request for the Result. In the non-blocking scenario, the MQB sends the response to the client using a predefined method, such as the Call-Back function.

In our design, we assume that all the clients use the non-blocking scenario. Two serving models based on the M/M/1 queuing discipline will be analyzed; the first is FIFO, where all messages arriving at the system (from the clients) are processed based on the First In First Out policy for all the queues. The second is Priority, where messages are served based on their priority parameter. This parameter is the expected processing time in seconds for the client asynchronous code; the client has to put the estimated processing time of the asynchronous code into the message. This estimation can be based on application expectation, where the application estimates the time using various software and hardware parameters. Alternatively, this time can be estimated based on

historical data, where the application uses previous time records of asynchronous operations to estimate the current one.

To lower the waiting time for the clients after they have processed their asynchronous operation, we assigned a higher priority to clients with a lower asynchronous time. Our aim in using priority queuing is to minimize the waiting time as much as possible for all clients but the priority model has to work on trust between clients and MQB to be effective.

The variables used in our model for MOM with M/M/1 for both FIFO and Priority queue are: Arrival rate λ service rate μ, Network Delay (ND) and average waiting time for clients (W). Both the arrival rate and the service rate are assumed to have a poisson distribution, while the network delay is deterministic. The main performance measures of the study models are:

- **System throughput:** The percentage of time spent by the client processing its asynchronous operation to the execution time of the system. A throughput of 100% means that the clients had no waiting time. RPC has a throughput of 0% because it does not perform any asynchronous operations
- **Client waiting time:** The waiting time of the client after finishing its asynchronous operation and before receiving the execution result
- **Execution time:** The time spent in the system for a single call message
- **System time:** The total time from sending the message until all operations is finished (i.e., until both execution time and asynchronous operation time are completed)

Models' time flow: For the RPC model, the message is sent to the service provider and the client suspends operations until it receives the result. The RPC time flow is represented in Fig. 5. For MOM, there are two scenarios for the client message flow (Fig. 6): MOM with

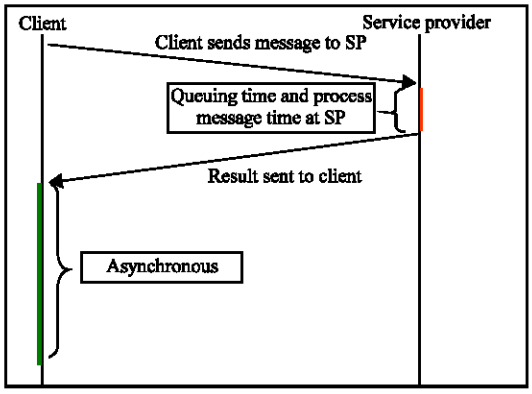

Fig. 5: RPC time flow

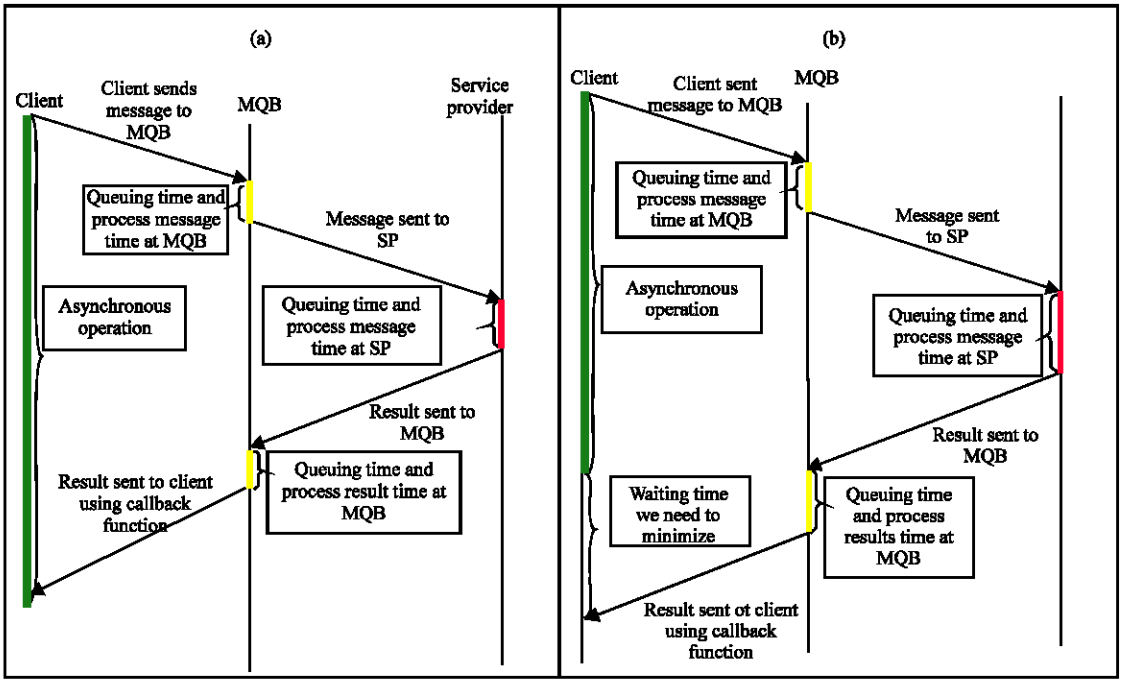

Fig. 6: MOM time flow. (a) MOM with asynchronous tiem> service execution time and (b) MOM with asynchronous time< service execution

asynchronous time greater than service execution time. This is the optimal case where the waiting time for the client will be zero seconds. With the second scenario, MOM has asynchronous time less than service execution time. The waiting time in this case will be greater than zero. We need to minimize it as much as we can by using priority queues to serve clients that have low asynchronous time first.

ANALYTICAL MODELING OF RPC AND MOM

RPC model: The RPC analytical model is a single server with a queue, so T_{RPC} is calculated as follows, using M/M/1 FIFO analysis (Bertsekas and Gallager, 1992):

$$T_{RPC} = \frac{1}{(\mu sp - \lambda sp)} + 2ND \quad (1)$$

MOM M/M/1 FIFO queue: To determine the throughput of an MOM system and to calculate the waiting time for the client (W), we use the queuing model in Fig. 4. We first find the average execution time spent in the MOM system with FIFO queue (T_{FMOM}). For this, we proceed as follows:

For MQB Receiver (T_R), the average message processing time is:

$$T_R = \frac{1}{\mu R - \lambda R} \quad (2)$$

MQB Sender processing time (T_S) is defined as follows:

$$T_S = \frac{1}{\mu s - \lambda s} \quad (3)$$

And finally, for the service provider (T_{SP}), we have:

$$T_{SP} = \frac{1}{\mu sp - \lambda sp} \quad (4)$$

To calculate the average execution time spent in the MOM system with FIFO queue (T_{FMOM}), we combine the above equations and we add the Network Delay (ND) to find the total time. We add 4*ND to the total time of the servers (T_{FMOM}), since ND is not included in any of these three processing times equations, Therefore we have:

$$T_{FMOM} = T_R + T_S + T_{SP} + 4ND \quad (5)$$

To find the average waiting time for a client after finishing its asynchronous operation (tasync) we proceed as follow: From Fig. 6 we can see that there are two cases for the waiting time (w_1): zero waiting time (Fig. 6a) and waiting time (w_2)>0 (Fig. 6b). These are the only cases for the MOM system and therefore for the first case (Fig. 6a) we have:

$$W_1 = \max\left(T_{async}, T_{FMOM}\right) - t_{async} \quad (6)$$

$$W_1 = \max\left(t_{async} - t_{async}\right) = 0$$

i.e., there is no waiting time in case t_{async} is greater than T_{FMOM} and therefore w_1 is zero, while for Fig. 6b, we have:

$$W_2 = T_{FMOM} - t_{anyc} \quad (7)$$

Combining Eq. 6 and 7, we derive the following general equation for client waiting time after sending message I.

$$W_i = \max\left(t^i_{async} - T_{FMOM}\right) - t^i_{async} \quad (8)$$

where, t^i_{async} is the asynchronous operation time for a client who sends a message i. From Eq. 8, we can calculate the average waiting time for clients in an MOM system, using the following equation:

$$W = \max\left(T_{async}, T_{FMOM}\right) - T_{async} \quad (9)$$

where, T_{async} is the average asynchronous time for all clients and can be calculated as follows:

$$T_{async} = \sum_{i=1}^{n} t^i_{async} \quad (10)$$

where, n is the number of messages sent to the MOM system.

The Throughput of the system can now be defined as follows:

$$\text{Throughput} = \frac{T_{async}}{T_{async} + W} \quad (11)$$

MOM M/M/1 priority queue: Here, we compare MOM with priority queuing with FIFO and determine their average waiting times. The priority values depend on the asynchronous operation processing time (t_{async}), where a high priority is associated with messages sent by clients with low t_{async}. All servers in the MOM system will implement priority queues.

To find the average message time spent in the MOM system with priority queuing (T_{PMOM}), we need to calculate

the average message execution time spent in the MOM system for priority class k (T^k_{PMOM}). We also calculate the average waiting time for priority class k (W_k). Assuming that the service rate is identical for all classes in each server, we can compute the utilization factor for each server as follows (Bertsekas and Gallager, 1992):

$$\rho^K_y = \frac{\lambda^K_Y}{\mu y} \tag{12}$$

where, Y in (ρ^K_y) refers to either MQB Receiver (ρ^K_R), MQB sender (ρ^K_R), or service provider (ρ^K_{SP}) and k is the priority class number.

To calculate T^K_{FMOM}, we first calculate the message waiting time in a queue for each class k in server Y(Q^K_Y), which is calculated as follows:

$$Q^K_Y = \frac{\sum_{i=1}^{n} \lambda^i_Y X^2_y}{2\left(1-\rho^1_Y - \cdots - \rho^{k-1}_Y\right)\left(1-\rho^1_Y - \cdots - \rho^k_Y\right)} \tag{13}$$

where, $\overline{X^2_Y}$ is the second moment of the average service time of server Y and n is the number of priority classes.

With equal arrival rate we can simplify the Eq. 13 to be:

$$Q = \frac{n\lambda_y X^2_y}{2\left(1-\rho_y(K-1)\left(1-k\rho_y\right)\right)} \tag{14}$$

We can calculate the average time in the MOM system for priority class k by calculating it for each server as follows: For MQB receiver:

$$T^K_R = \frac{1}{\mu_R} + Q^K_R \tag{15}$$

For MQB Sender:

$$T^K_S = \frac{1}{\mu_S} + Q^K_S \tag{16}$$

For service provider:

$$T^k_{SP} = \frac{1}{\mu_{sp}} + Q^K_{sp} \tag{17}$$

So, the total average time in the MOM system for priority class k is defined by:

$$T^K_{FMOM} = T^K_R + T^K_S + T^k_{sp} + 4ND \tag{18}$$

The network delay is identical to FIFO queue.

To calculate the total average time in the system for all classes, we define T_{MOM} as:

$$T_{FMOM} = \sum_{K=1}^{n} T^k_{FMOM} \tag{19}$$

where, n is the total number of priorities in the MOM system.

From Eq. 19, we can compute the average waiting time for messages in priority class k as follows:

$$W_k = \max\left(T^k_{async}, T^K_{FMOM}\right) - T^K_{async} \tag{20}$$

where, n is the average asynchronous time for messages in class k and can be calculated as follows:

$$T^k_{async} = \frac{\sum_{i=1}^{n} t^k_{async}}{n} \tag{21}$$

where, n is the number of messages sent to priority class k in the MOM system and t^k_{async} is the asynchronous operation time for the client that sent message i in class k.

Finally, from Eq. 20, we can calculate the total average waiting time for an MOM system with priority queuing as follows:

$$W = \frac{\sum_{k=1}^{n} W_k}{n} \tag{22}$$

where, n is the total number of priorities in the MOM system.

For the throughput of an MOM system using priority queuing, we first define it for each class k as follows:

$$\text{Throughput}^k = \frac{T^k_{async}}{T^k_{async} + W_k} \tag{23}$$

Using Eq. 23, the throughput of the system is them given by:

$$\text{Throughput} = \frac{\sum_{k=1}^{n} \text{throughput}^k}{n} \tag{24}$$

Performance results analysis: Here, we compare the middleware call procedures for RPC and MOM (with FIFO queuing) with MOM middleware with Priority queue, for the total average waiting time (W). The same parameter values are assumed for all procedures.

To compute the analytical model results, we will assume the following values: $\lambda_R = \lambda_{SP} = \lambda_S = \lambda = 10$, $\mu_R = \mu_S = 20$, $\mu_{SP} = 11$, N = 0.035 and the number of priority

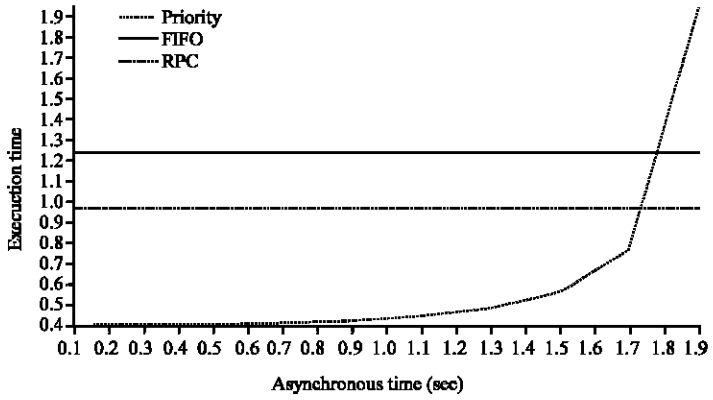

Fig. 7: Analytical execution time for high SP utilization

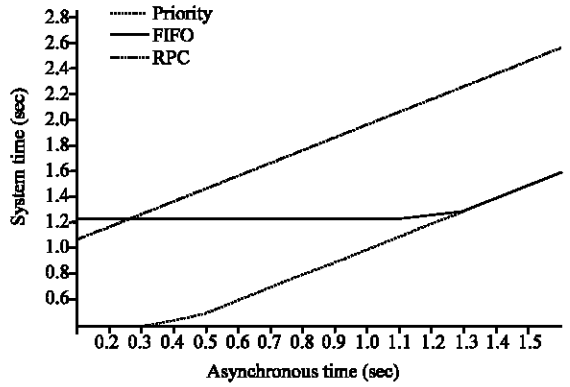

Fig. 8: Analytical system time for high SP utilization

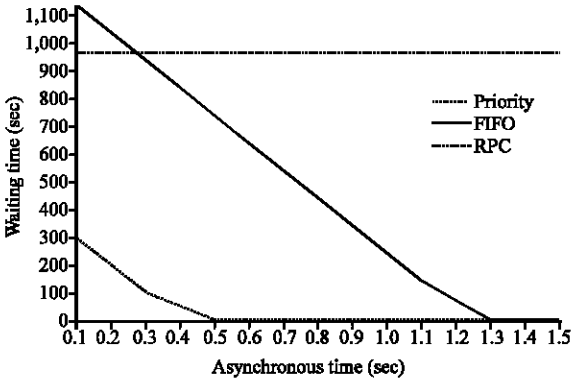

Fig. 9: Analytical waiting time for high SP utilization

classes = 10. For Priority queue, the total arrival rate will be divided among various classes. The arrival rate values will be ascending, where the highest priority class (λ_1) will have a lower arrival rate (= 0.18) with rate increments equal to 0 .36 and the lowest priority class (λ_{10}) will have a higher arrival rate (=1.82).

Performance with high service provider utilization: For a service provider utilization equal to 90%, we derive the following results from our analytical model:

Execution time (T_{MOM}): As shown in Fig. 7, FIFO Model has a constant average execution time, regardless of the asynchronous time of the client. For priority queue, the execution time is indirectly proportional to asynchronous time. After 1.8 sec, the execution time for priority queue begins to go higher than FIFO. However, this does not mean that FIFO is better as our main performance measure is the waiting time for the client, not how long his message stays in the system.

The system time: Here, we compare the system time of the MOM with the RPC to find when it is better to switch from an RPC system to an MOM system. The RPC total system time is the execution time plus the asynchronous time. We include here the asynchronous operation time in the system time in order to enable the comparison of MOM with RPC. The execution times for both RPC and MOM are given by the following:

$$S_{MOM} = \max\{T_{async}, T_{MOM}\} \quad (25)$$

$$S_{RPC} = T_{RPC} + T_{async} \quad (26)$$

Figure 8a and b show the system time as a function of asynchronous time. As shown, RPC is better than MOM with FIFO for small asynchronous operation time values. MOM with Priority is the best for all cases. However, as the asynchronous operation time increases, the two MOM models have almost the same system time.

The waiting time (W): As shown in Fig. 9, RPC is better (in terms of waiting time) than FIFO for small values of asynchronous time. Priority queue shows a significant improvement compared to FIFO and RPC. By using

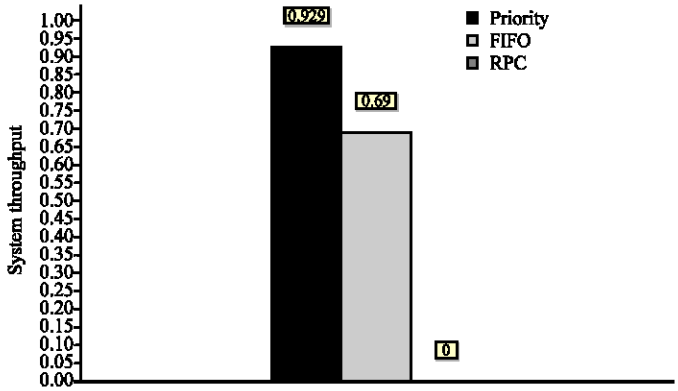

Fig. 10: Analytical throughput result for high SP utilization

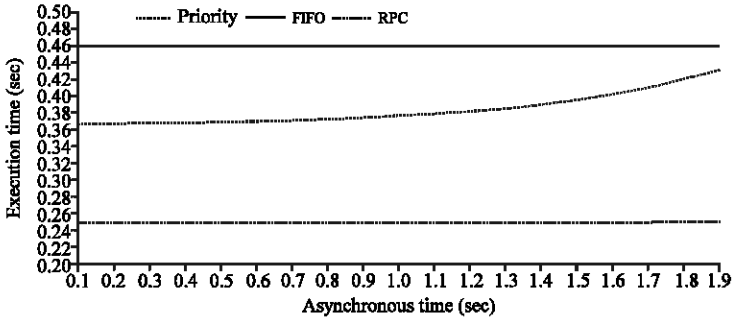

Fig. 11: Analytical execution time for low SP utilization

priority queues, we can improve the overall client experience, since the clients will have lower waiting times.

The throughput of the system: The throughput for our systems is shown in Fig. 10.

As shown in Fig. 10, the MOM with Priority queue has increased the throughput of the system to 93%, compared to RPC. MOM with FIFO also increases the throughput, but only to 69%.

Performance with low service provider utilization: For this case, we investigate a system where the Service Provider is only 50% utilized. This gives us an indication of how each model behaves when it is not fully utilized. The arrival rate values will again be ascending where the highest priority class (λ_1) will have a lower arrival rate (= 0.1), with rate increments equal to 0.1 and the lowest priority class (λ_{10}) will have a higher arrival rate (= 1.0). We derive the following results from our analytical model: The Execution Time (T_{MOM}):

For this case, Fig. 11 shows that priority queue is better than FIFO for all values. However, as shown, the difference is not as large as in the previous case. For RPC, the execution time is now less than Priority queue waiting time.

The system time: The system time, which includes the asynchronous time, is shown in Fig. 12 we can see that the RPC model is better than MOM with FIFO when the asynchronous time is 0.2 sec or less. It is also better than MOM with Priority when the asynchronous time is 0.1 sec or less. MOM with Priority queue is however better in waiting time than FIFO until the asynchronous time equals 0.5 sec, after which they are the same.

The waiting time (W): Figure 13 shows that the RPC model for clients that have asynchronous time of 0.1 sec or less is better than both MOM models. This is because the overheads of the MOM system make each client's waiting time longer than his asynchronous operation. MOM with Priority queue is better in than FIFO queue up to asynchronous time equal to 0.5 sec, after which they are the same.

System throughput: As shown in Fig. 14, both MOM models have almost the same throughput; Priority queue is slightly higher, by only 2%. The throughput of FIFO is higher when compared to a system with high SP Utilization (e.g., the throughput is equal 69% while here it is 89%). For MOM with Priority queue it is almost the same.

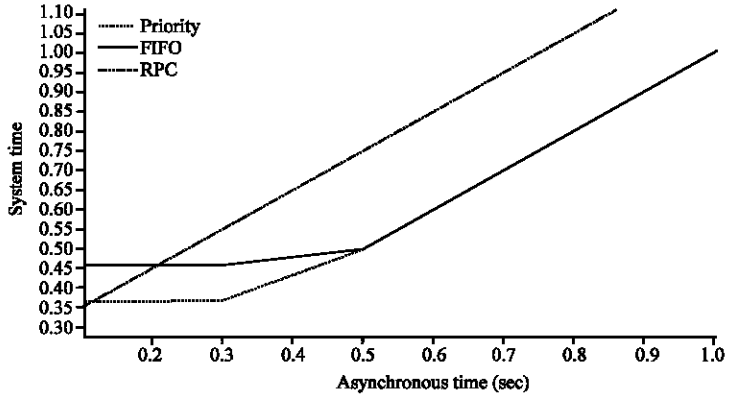

Fig. 12: Analytical system time for low SP utilization

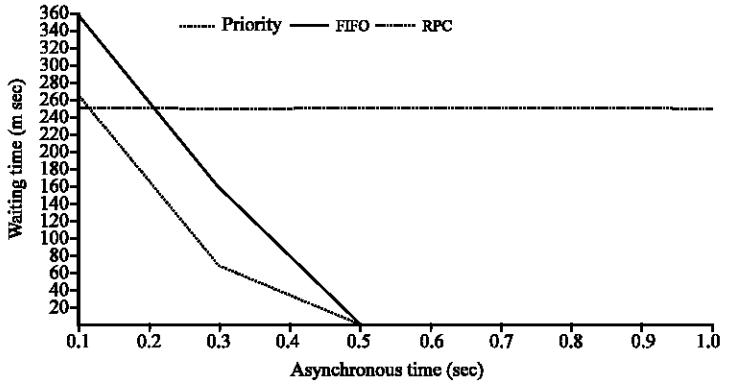

Fig. 13: Analytical waiting time for low SP utilization

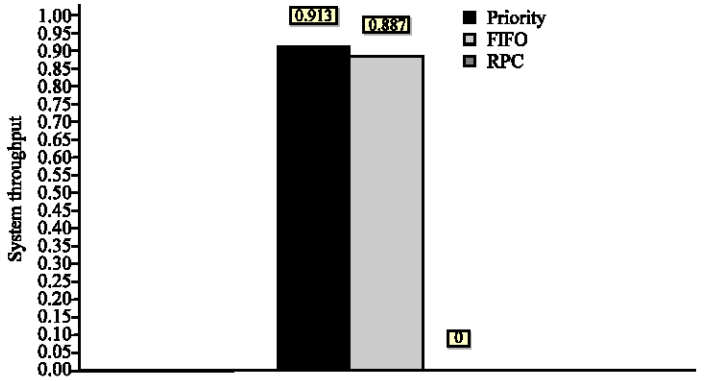

Fig. 14: Analytical throughput result for low SP utilization

CONCLUSIONS AND FUTURE RESEARCH

In this study, Message-Oriented Middleware (MOM) is analyzed using two models: FIFO and Priority queue. We also compared it to middleware that uses RPC. We developed an analytical model for MOM with FIFO and Priority queue and compared their results. The study has been carried out using different arrival rate patterns, service provider utilization and asynchronous operation process time. Throughout the study, we tested the main two parameters that affect the decision over which model to use and these are: service provider utilization and the asynchronous operation process time. We have found that using priority queuing has improved the overall

throughput of the system. This, in turn, will improve the performance of many middleware practical applications such as middleware in wireless sensor networks and in applications of middleware for Pervasive and Ad-Hoc Computing.

We have seen that increasing the average asynchronous time increases the overall system throughput dramatically for the Priority queue model. Therefore, for all MOM applications, it is recommended to use asynchronous operations as much as possible. To improve further the priority class model performance we suggest the implementation of Priority model with Aging, where the message's priority increases dynamically with queuing time and to use Pre-emptive message queuing.

REFERENCES

Banavar, G., T. Chandra, R. Strom and D. Sturman, 1999. A case for message oriented middleware. Proceedings of 13th International Symposium on Distributed Computing, Sept. 27-29, Bratislava, Slovak Republic, pp: 1-18.

Bertsekas, D. and R. Gallager, 1992. Data Networks. 2nd Edn., Prentice Hall, Englewood Cliffs, NJ.

Blumenthal, J., M. Handy, F. Golatowski, M. Haase and D. Timmermann, 2003. Wireless sensor networks-new challenges in software engineering. Proceedings of 9th IEEE International Conference on Emerging Technologies and Factory Automation, September 2003, Lisbon, Portugal, pp: 1-6.

Krafzig, D., K. Banke and D. Slama, 2004. Enterprise SOA: Service-Oriented Architecture Best Practices. Prentice Hall, New Jersey, USA.

Lingel, K., 2001. Security requirements for message-oriented middleware. EDP. Audit Control Security, 29: 1-2.

Maheshwari, P. and M. Pang, 2005. Benchmarking message-oriented middleware: TIB/RV versus SonicMQ. Concurrency Computation Practice Experience, 17: 1507-1526.

Mascolo, C., L. Capra and W. Emmerich, 2002. Mobile computing middleware. Lecture Notes Comput. Sci. Ser., 2497: 20-58.

Menascwe, D.A., 2005. MOM vs. RPC: Communication models for distributed applications. IEEE Internet Comput., 9: 90-93.

Myerson, J.M., 2002. The Complete Book of Middleware. Auerbach Publications, Boca Raton.

Sachsa, K., S. Kouneva, J. Baconb and A. Buchmann, 2009. Performance evaluation of message-oriented middleware using the SPECjms 2007 benchmark. Performance Evaluation, 66: 410-434.

Saiedian, H., N. Ghanem and J. Natarajan, 2002. Framework for evaluating distributed object models and its application to web engineering. Ann. Software Eng., 13: 71-96.

Schiele, G., M. Handte and C. Becker, 2008. Experiences in designing an energy-aware middleware for pervasive computing. Proceedings of 2008 6th IEEE International Conference on Pervasive Computing and Communications, March 17-21, Hong Kong, China, pp: 504-508.

Tran, P., P. Greenfield and I. Gorton, 2002. Behavior and performance of message-oriented middleware systems. Proceedings of 22nd IEEE International Conference on Distributed Computing Systems Workshops, July 2-5, Vienna, pp: 645-654.

Verissimo, P. and L. Rodrigues, 2001. Distributed Systems for System Architects. Kluwer Academic Press, New York.

Vinoski, S., 2002. Where is middleware. IEEE Internet Comput., 6: 83-85.

Yang, H., M. Kim, K. Karenos, F. Ye and H. Lei, 2009. Message-oriented middleware with QoS awareness. Proceedings of 17th International Joint Conference on Services-Oriented Computing, November 2009, Stockholm, Sweden, pp:1-15.