MOM vs. RPC:

Communication Models for Distributed Applications

Daniel A. Menascé • *George Mason University*

An important aspect of designing a distributed application is the communication model you use to connect its various components. Some middleware solutions,[1] such as Web services,[2] offer two of the most important communication paradigms: asynchronous messaging and remote procedure call (RPC).

Message-oriented middleware (MOM) lets a service's consumers physically and temporally decouple from the service providers (see www.webmethods.com and http://www-306.ibm.com/software/integration/wmq/). Communication between service providers and their consumers is asynchronous, and they don't need to be available at the same time because they communicate by sending and receiving messages from designated message queues. In contrast, RPC is a synchronous method of requesting remote service execution. Consumers must suspend service execution until they receive a reply from the provider.

MOM and RPC have advantages and disadvantages. MOM solutions tend to be more robust to failures than RPC, and they allow service requesters to continue to process while service providers work on their requests. However, programming MOM-based applications is more cumbersome because distribution isn't as transparent to the programmer as with RPCs. In this column, I provide a quantitative framework you can use to compare MOM- and RPC-based solutions.

Basic Model

Consider an *application program component* (APC) that requests a service from a service provider (SP). We assume that the APC could potentially execute some asynchronous code after sending a request for service but before receiving a reply from the SP; we call this time to execute the asynchronous code t_{async}. With RPCs, the APC must wait until the RPC returns a result before it can execute the asynchronous code.

With MOM, the APC and the SP exchange messages through a *message queue broker* (MQB), as Figure 1 shows. After sending a service request to the SP, the APC can execute its asynchronous code. The SP is constantly retrieving messages addressed to it from the MQB that contain requests for service execution. After completing this execution, the SP posts a message to the MQB addressed to the requesting APC, which retrieves the reply.

Let's look at how to compute a request's execution time, from the APC's viewpoint, for RPC- and MOM-based communication with an SP (see Figure 2).

We give an RPC-based request's execution time, T_{RPC}, as

$$\begin{aligned} T_{RPC} &= t_N + t_s + t_N + t_{async} \\ &= 2 \times t_N + t_s + t_{async}, \end{aligned} \qquad (1)$$

where t_N is the network time between two points and t_s is the response time of a request at the SP. As Figure 2a shows, the APC can execute the asynchronous code only after it receives the reply from the SP. The service response time t_s is a function of the average arrival rate λ_s of requests from all APCs to the SP and the average time x_s to process a service request. Thus, assuming a simple M/M/1 queuing model,[3] we calculate t_s as

$$t_s = x_s/(1 - \lambda_s x_s). \qquad (2)$$

I examine the MOM-based communication with the SP by considering two cases: the asynchronous code execution can complete either before the service execution (Figure 2b) or after it ends (Figure 2c). The red bars in Figure 2b and 2c indicate the average response time t_m of the MQB when it receives requests to execute a post-a-message or get-a-message operation. The MQB

1089-7801/05/$20.00 © 2005 IEEE

response time t_m is a function of the average arrival rate λ_m of message broker operations and the average time x_m to process a messaging operation. Thus, assuming an M/M/1 queuing model,[3] we calculate t_m as

$$t_m = x_m/(1 - \lambda_m x_m). \tag{3}$$

For the case of Figure 2b, a MOM-based request's execution time, T_{MOM}, is

$$\begin{aligned} T_{MOM} &= t_N + t_m + t_N + t_s + t_N \\ &\quad + t_N + t_m + t_N \\ &= 3 \times t_N + t_m + t_s + 2 \times t_N \\ &\quad + t_m. \end{aligned} \tag{4}$$

With Figure 2c, we can compute T_{MOM} as

$$\begin{aligned} T_{MOM} &= t_{async} + t_N + t_m + t_N \\ &= t_{async} + 2 \times t_N + t_m. \end{aligned} \tag{5}$$

We combine Equations 4 and 5 to obtain a single expression that accounts for the two cases in Figure 2b and 2c:

$$\begin{aligned} T_{MOM} &= \max \{t_{async}, 3 \times t_N + t_m \\ &\quad + t_s\} + 2 \times t_N + t_m. \end{aligned} \tag{6}$$

Numerical Example

Let's look at an example that considers the following parameters: x_m = 0.01 seconds, x_s = 0.1 seconds, t_N = 0.035 seconds. We consider that λ_m = 10 × λ_s and use the following values for λ_s: 2.5 requests per second, 5 requests per second, and 7.5 requests per second. We consider t_{async} equal to the service processing time x_s multiplied by a factor f, which varies from 0.1 to 10.

Figure 3 (next page) shows the variation of a request's execution time, T_{RPC} and T_{MOM}, as a function of the factor f for the three values of λ_s.

We can draw the following observations from the curves in Figure 3 and from Equations 1 through 5:

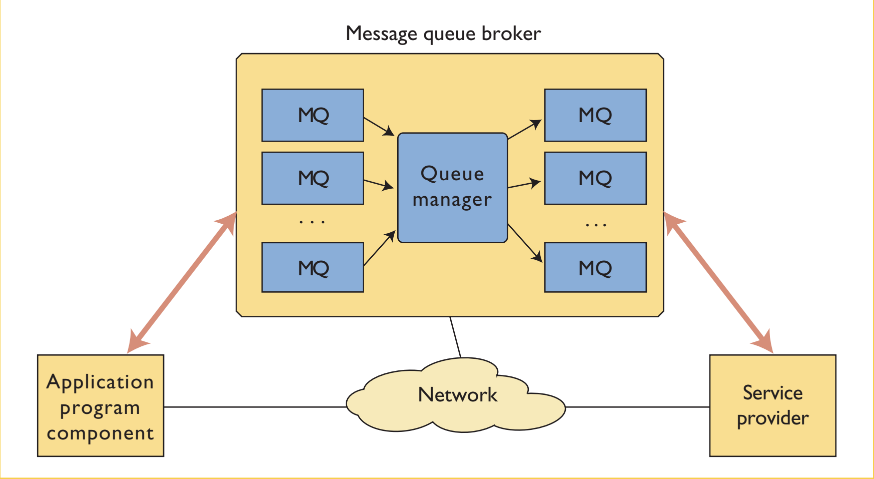

Figure 1. Message queue broker. A queue manager moves messages from incoming to outgoing message queues (MQs). An application program component communicates with a service provider through message exchanges.

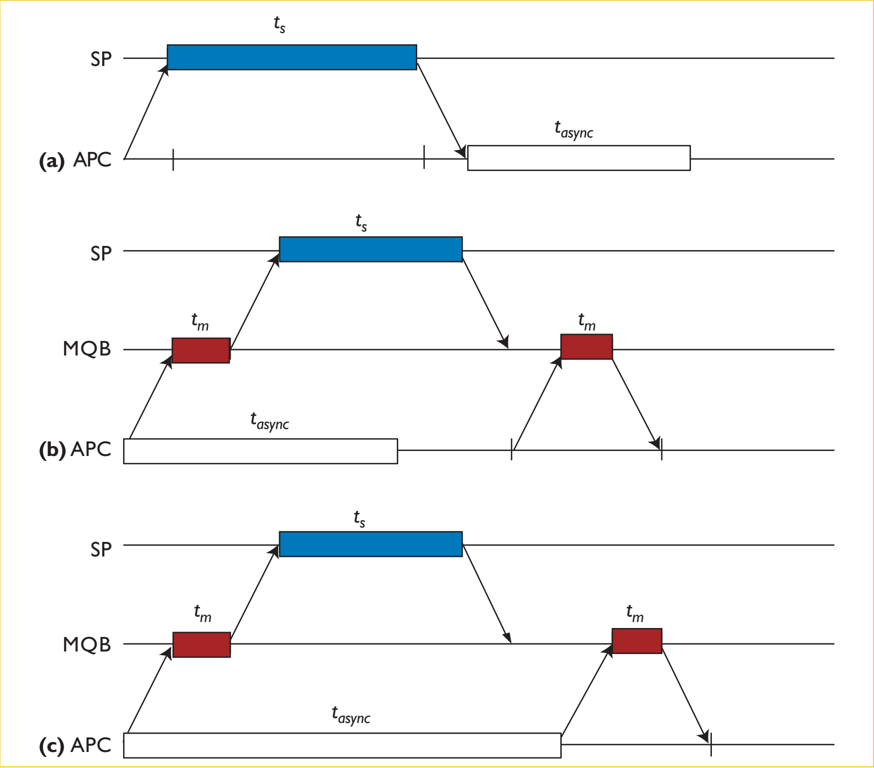

Figure 2. Time-axis analysis of RPC versus MOM. (a) In the RPC scenario, the application program component can execute asynchronous code only after receiving a reply from the service provider. (b) In this MOM scenario, the asynchronous code execution ends before the service execution is complete. (c) In this MOM scenario, the service execution ends after the asynchronous code execution is complete.

- RPC's execution time T_{RPC} increases linearly with the asynchronous code execution time. However, the intercept of these lines increases in a nonlinear fashion with the arrival rate of requests to the SP, due to Equation 2.

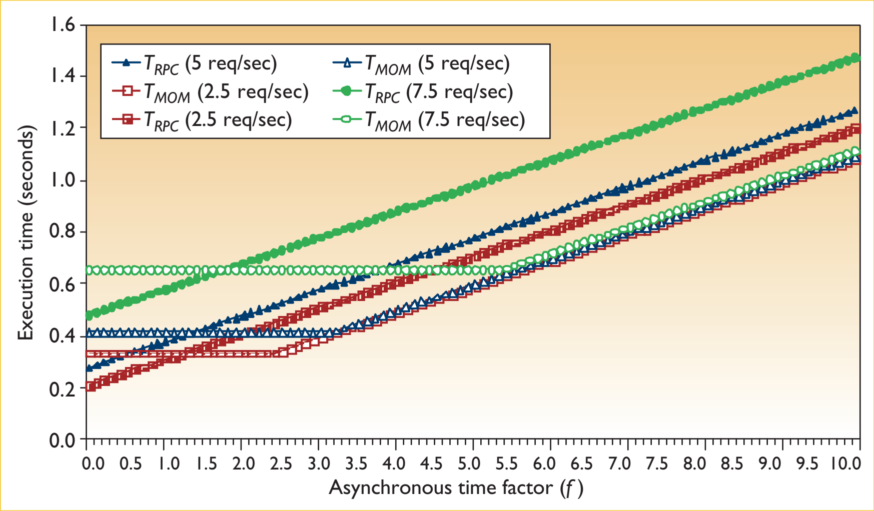

Figure 3. Request execution time for the RPC- and MOM-based communication models as a function of the asynchronous time factor f. The RPC time increases linearly with f. The MOM-based time is constant for low values of f and then increases linearly with f. For low values of f, RPC is better than MOM-based communication. The situation is reversed for larger f values.

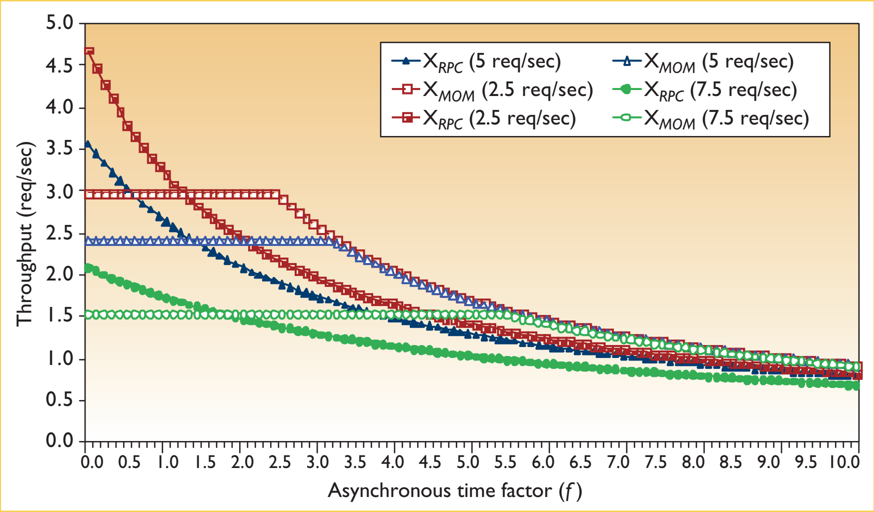

Figure 4. Request throughput for the RPC- and MOM-based cases as a function of the asynchronous time factor f. The throughput of RPC-based requests always decreases with f, whereas that of MOM-based requests is constant for small values of f and then decreases for larger values of f. For low values of f, the throughput of MOM-based communication is smaller than that of RPC. The situation is reversed for higher f values.

- MOM's execution time T_{MOM} doesn't vary with the asynchronous code execution time t_{async} as long as this time is smaller than the time needed to obtain a reply from the SP (see Equation 4). After that point, T_{MOM} increases linearly with t_{async} for a given value of λ_s. For λ_s = 5 requests per second, this happens when f = 3.2 — that is, $t_{async} = 3.2 \times x_s = 3.2 \times 0.1 = 0.32$ seconds.
- For small values of t_{async}, an RPC-based request has a smaller execution time than a MOM-based request. However, as t_{async} becomes significant, the parallelism that MOM-based communications' asynchronous behavior obtains provides better response time. For λ_s = 5 requests per second, this happens when f = 1.45 — that is, t_{async} = 0.145 seconds. We can combine Equations 1 and 5 to obtain a point at which T_{MOM} starts to outperform T_{RPC} — that is, $t_{async} = 3 \times t_N + 2 \times t_m$. Thus, as the load λ_m on the MQB increases, so does the point at which T_{MOM} starts to outperform T_{RPC}, as expected.

The throughput of RPC-based communication always decreases with f, whereas the throughput of the MOM-based case is flat for a given value of λ_s until t_{async} starts to dominate the time for the SP to reply. At this point, the throughput starts to decrease, and for very large values of t_{async} it tends toward that of the RPC case. Figure 4 shows the throughput of requests from the APC's viewpoint.

This simple framework indicates that both MOM- and RPC-based communications have their performance advantages. RPC is a better choice when the service provider's response is small, thus creating a negligible penalty for suspending execution while waiting for a reply. MOM-based communication works better for long-lived transactions that require long execution times from the SP. Most real applications are likely to combine situations in which some SPs reply very fast with situations in which other SPs take a long time before replying to the APC. As an example, consider a purchase order (PO) business process composed of the following steps:

1. check PO's correctness,
2. check item availability,
3. check buyer's credit limit,
4. request items from the warehouse,

5. ship items to buyer, and
6. generate invoice.

If the PO is correct (step 1), we can expect steps 2 and 3 to be executed in parallel. Only after steps 2 and 3 complete successfully — the items are available and the buyer has a sufficient credit limit — can an SP request items from the warehouse (step 4). Once the items arrive, a shipping order is started (step 5) and an SP generates the invoice (step 6). This business process could take several days to process. Thus, the communication between the SPs that execute the various steps should use asynchronous messaging. However, some of these steps might use other SPs that return fast responses. An example would be checking the credit limit in step 2. This could be accomplished via an RPC to a financial institution's SP. Thus, both types of communication models, RPC and asynchronous messaging, must be used where appropriate.

Acknowledgments

Grant number NMA501-03-1-2022 from the US National Geospatial-Intelligence Agency (NGA) partially supports this work.

References

1. S. Vinoski, "Where is Middleware," *IEEE Internet Computing*, vol. 6, no. 2, 2002, pp. 83–85.
2. F. Curbera et al., "Unraveling the Web Services," *IEEE Internet Computing*, vol. 6, no. 2, 2002, pp. 86–93.
3. D.A. Menascé, V.A.F. Almeida, and L.W. Dowdy, *Performance by Design: Computer Capacity Planning by Example*, Prentice-Hall, 2004.

Daniel A. Menascé is a professor of computer science, codirector of the E-Center for E-Business, and director of the MS in E-Commerce program at George Mason University. He received a PhD in computer science from UCLA. Menascé is a fellow of the ACM and a recipient of the 2001 A.A. Michelson Award from the Computer Measurement Group. Contact him at menasce@cs.gmu.edu.