Response-Time Analysis of Composite Web Services

Daniel A. Menascé • *George Mason University* • *menasce@cs.gmu.edu*

Web services let programs submit requests to other programs over the Internet via open protocols and standards.[1] Many traditional Web sites, including popular search engines like Google and large online bookstores such as Amazon.com, are boosting their traffic through Web service APIs.

A single Internet application can invoke many different Web services — for example, the metasearch engine WebSifter uses several online ontologies to refine a user's request into a more meaningful query and then submits that query to various search engines in parallel.[2] We call such applications *composite Web services*.[3] As I discussed in a previous column, many important challenges stem from the quality-of-service issues in composite Web services.[4] In this column, I address the impact of slow services on the overall response time of a transaction that uses several Web services in parallel.

The Computing Paradigm

Consider the distributed application in Figure 1. An initialization step S_0 is followed by a parallel invocation of N Web services running on different servers. We assume the average time to obtain a response from any Web service 1 through $N-1$ to be S time units. Web service N is slower to respond; we assume its average response time to be $g \times S$, where $g \geq 1$. (The factor g is the slowdown factor of Web service N.) The application's final step S_f can only be executed after all N Web services have responded. The time spent waiting for all N Web services to respond is the synchronization component of the application's overall response time.

This type of computing paradigm, called *fork and join*, is typical of many parallel and distributed applications. I use this simple model here to explore the impact of g on the average time T required to execute all N services — the average time taken inside the dashed rectangle of Figure 1.

A Performance Model

Let $T(g)$ be the average time to execute Figure 1's fork and join as a function of g. I am interested in exploring the impact of g on the application's overall slowdown factor G, which is

$$G = \frac{T(g)}{T(1)}. \qquad (1)$$

The time it takes to execute N Web services that must synchronize after they've all completed is $\max_{i=1}^{N}\{S_i\}$, where S_i is the time it takes to execute Web service i ($i = 1, ..., N$). It's easy to show that if all service times S_i are exponentially distributed with the same mean S, then

$$T(1) = H_N \times S, \qquad (2)$$

where H_N is the Nth harmonic number defined as

$$\sum_{i=1}^{N} 1/i.$$

To find a general expression for $T(g)$, I adapted the Markov chain model I derived in a previous article that studied the scheduling of parallel tasks running on heterogeneous multiprocessors.[5] A general state of this Markov chain is (i, j, k), where i ($i = 0, ..., N-1$) indicates the number of Web services still running on the fast Web services, j ($j = 0, 1$) is the number of Web services running on the slow Web service, and k ($k = 1, ..., N$) is the number of Web services yet to complete. I found a closed-form solution for this Markov chain in a previous study.[5] We can easily obtain $T(g)$ from that solution:

$$T(g) = \frac{S}{\left(N - 1 + \frac{1}{g}\right)P(N)} \qquad (3)$$

where

$$P(N) = \left[1 + \sum_{i=1}^{N-2} F(i) + gF(1) + V\right]^{-1},$$

$$V = \frac{1}{g}\sum_{j=1}^{N-1}\frac{1}{j}\sum_{i=j}^{N-1}F(i),$$

$$F(i) = \prod_{j=1}^{N-i-1}\frac{N-j}{N-j-1+\frac{1}{g}}.$$

The expression in Equation 3 reduces, as expected, after some algebraic manipulation to Equation 2 when $g = 1$.

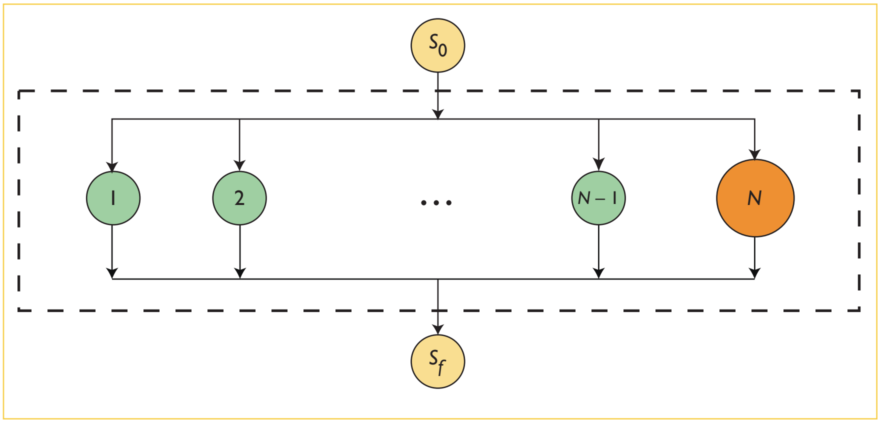

Figure 1. A composite Web service. After an initialization step S_0, N Web services are invoked in parallel. Service N takes longer than the others, and the final step S_f can only be carried out after all N services have completed.

Analysis of the Results

Clearly, when $g = 1$, the overall slowdown factor G is equal to 1. As g increases, more time is spent at the slow Web service relative to the others. Figure 2 shows the variation of G as a function of g for four different values of the number of Web services (N = 5, 10, 15, and 20) the application uses. For example, when g is equal to 5, the application's overall slowdown factor is 2.4 when five Web services are involved. In other words, if an application uses five Web services, one of which is five times slower than the other four, the overall average response time will be 2.4 larger than what would be achieved if all five Web services had the same response time as the four fast services.

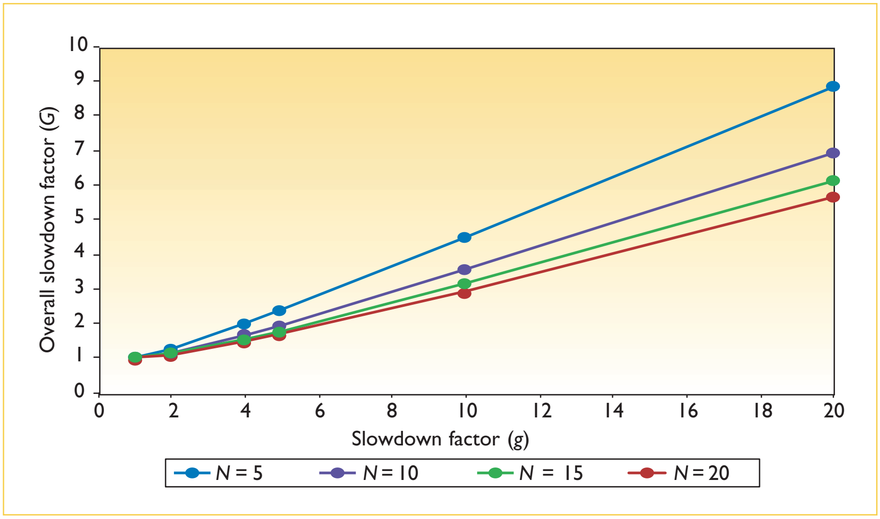

Figure 2. Application slowdown factor G versus Web service slowdown factor g. We consider four different values of the number of Web services N.

Figure 2's curves show two things. One, for any value of N, G increases with g. For small values of g (below four), G's increase is nonlinear. This nonlinearity is because the time spent waiting for the synchronization of all Web services is an important component of overall response time. As g increases, it becomes more likely that all $N - 1$ fast Web services will complete before the slow one. $T(g)$ is thus dominated by the time it takes to execute the slow Web service; at this point, the variation of G versus g becomes almost linear. Two, for the same value of g, G decreases with the number of Web services the application invokes in parallel. (This effect is more pronounced for larger values of g.) The explanation is that as g increases, the slowest Web service dominates overall response time, decreasing the synchronization effect. But $T(1)$ increases with N, thus G decreases with N.

Figure 3 (next page) shows the variation of the average response time $T(g)$ normalized with respect to the average response time S of the fast Web services versus g. For example, as the curves indicate, for $g = 2$ and $N = 20$, the application's average response time is approximately equal to four times the average response time of the fastest Web service.

Figure 3's curves indicate two things. One, the average normalized response time increases with g in a nonlinear fashion for low values of g. For higher values of g, the variation is almost linear. This is because for low values of g, synchronization time dominates the time spent at the slow service. Two, the average normalized response time increases with N, especially for low values of g, due to the relatively high synchronization time. As g increases, the impact of N decreases because the time spent at the slow Web service dominates overall execution time.

Concluding Remarks

We can generalize the fork-and-join model discussed here in the following cases:

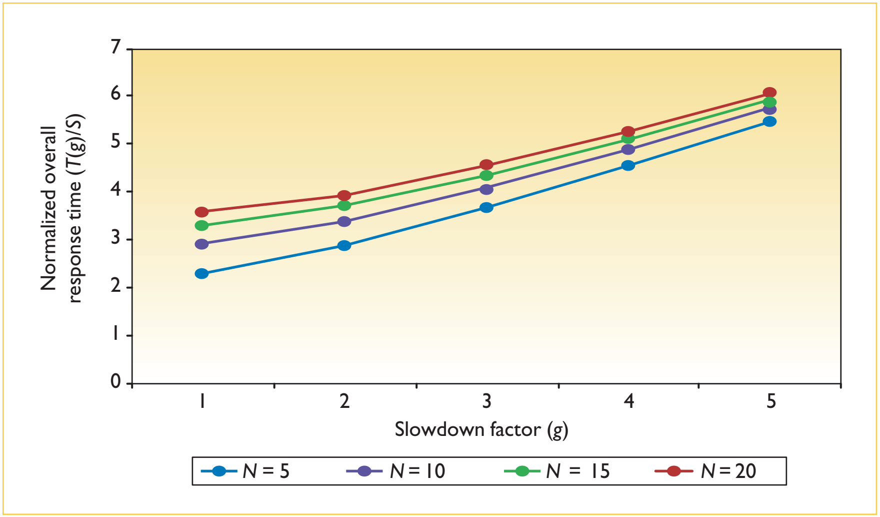

Figure 3. Normalized response time of the application versus the Web service slowdown factor g. The variation with g is nonlinear for low values of g and becomes linear for higher values of g.

- The application's structure consists of a sequence of fork and joins connected by one or more steps. In this case, the application's total average response time is the sum of the times spent at each fork-and-join phase plus the time spent at each serial step.
- The application's structure includes fork-and-join components executed in a mutually exclusive fashion with given probabilities. In this case, the application's average response time is obtained as the weighted sum of the solutions; the weights are the probabilities of executing each fork and join.
- One of the Web services is faster, as opposed to slower, than all others (for example, $g < 1$). In this case, the derivation of Equation 3 works for any value of $g > 0$.

Although simple, the model discussed in this article provides good insights on the performance impact of a slower service that participates in an application using several Web services. We can improve the application's scalability as a whole by reducing the time spent at the slow server. Caching of the results that the server provides can significantly improve performance.

References

1. F. Curbera et al., "Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI," *IEEE Internet Computing*, vol. 6, no. 2, 2002, pp. 86–93.
2. W. Kim, L. Kerschberg, and A. Scime, "Learning for Automatic Personalization in a Semantic Taxonomy-Based Meta-Search Agent," *J. Electronic Commerce Research and Applications (ECRA)*, vol. 1, no. 2, 2002, pp. 150–173.
3. B. Benatallah, Q.Z. Sheng, and M. Dumas, "The Self-Serve Environment for Web Services Composition," *IEEE Internet Computing*, vol. 7, no. 1, 2003, pp. 40–48.
4. D.A. Menascé, "QoS Issues in Web Services," *IEEE Internet Computing*, vol. 6, no. 6, 2002, pp. 72–74.
5. D.A. Menascé et al., "Static and Dynamic Processor Scheduling Disciplines in Heterogeneous Parallel Architectures," *J. Parallel and Distributed Computing*, vol. 28, no. 1, 1995, pp. 1–18.

Acknowledgments

The author's work is partially supported by grant number NMA501-03-1-2022 from the National Imagery and Mapping Agency (NIMA).

Daniel A. Menascé is a professor of computer science, the co-director of the E-Center for E-Business, and the director of the MS in E-Commerce program at George Mason University. He received a PhD in computer science from the University of California, Los Angeles, and published the books *Performance by Design*, *Capacity Planning for Web Services,* and *Scaling for E-Business* (Prentice Hall, 2004, 2002, and 2000). He is a fellow of the ACM and a recipient of the A.A. Michelson Award from the Computer Measurement Group.