Reference Metrics for Service-Oriented Architectures

Dr. Yun-Tung Lau
Science Applications International Corporation

This article presents a set of reference metrics for measuring the quality of services in Service-Oriented Architectures (SOAs). It introduces the metrics cube and scalability curve and applies them to the development of service-level agreements (SLAs) and capacity planning. This article also discusses the challenges and approaches for defining and allocating end-to-end metrics in a net-centric environment such as the Global Information Grid (GIG).

In an SOA, a set of loosely coupled services work together over a network to provide functionalities to end-users [1]. The service provider registers information about a service at a service registry. Service consumers can find the service from the registry and then invoke the service through the service interface.

For the Department of Defense (DoD), a set of GIG Enterprise Services will provide warfighting, business, and intelligence capabilities to support operational missions conducted by various communities of interest [2]. Examples include Net-Centric Enterprise Services (NCES) [3] and Net-Enabled Command Capability (NECC) [4].

Services in an SOA have well-defined service interfaces. They also have SLAs which are parts of the service contracts that specify the levels of service expected after deployment. A key aspect of an SLA is the set of metrics for measuring performance and quality of service. This article develops an overarching model of reference metrics relevant to end-user experience. It introduces the concept of a metrics cube that captures the relationship between the metrics which are then applied to the development of SLAs and capacity planning.

The reference metrics are important to the successful implementation, deployment and sustainment of SOA in the GIG because of the following:

- They form the basis for combining metrics across network and computing infrastructures for services in the GIG net-centric environment. Since those infrastructures typically fall under various responsible entities, having a basic reference set is critical to the development of end-to-end metrics relevant to an end-user's experience.
- They relate directly to consumer's (end-user) experience using a service. This includes timeliness, scalability, availability, and reliability, which are specified in SLAs.
- They are used throughout a system engineering life cycle, including requirement definition, SLA development, service design, performance testing, and SOA sustainment.

Reference Metrics

The reference metrics are collectively referred to by their symbols as the TSAR (service Time, Scalability, Availability, Reliability) metrics. They are defined in more detail in Table 1.

For synchronous services, such as a request/response Web service, T is simply the response time. It is measured from the time a consumer sends a request to when the consumer receives a response. Typically, T will have an average and a standard deviation. It is the sum of network latency (including transmission time, propagation time, Internet protocol delay, and congestion) and time spent at the service provider (including local processing time and back-end processing time).

“The TSAR metrics relate directly to consumer (end-user) experience with a service by answering the following questions: how fast (T), how much/many (S), how durable (A), and how reliable (R).”

For asynchronous services, such as a messaging service, T is the delivery time. It is measured from when a publisher sends a message to when subscribers receive it. T is typically a distribution with T_min, T_average, and T_max. A service-level agreement may guarantee delivery within a certain T_max.

Scalability (S) measures a service's ability to handle growing amounts of work within the desired time and reliability ranges. Examples are user load (number of users within a certain time span), number of requests per unit time, and size of requests or messages over a certain time.

Availability (A) is defined as one minus the percentage of planned and unplanned service down time. In other words, it is the combined probability that a service is up and running. It is often expressed as a number of nines, such as 99.9 percent (8.8 hours down/year). Contribution for A comes from planned hardware/software maintenance, hardware failure of networks and processors, and software failure due to fatal defects (e.g. memory leaks).

Finally, Reliability (R) is the percentage of service completion with anticipated results when the service is *available*. Hence if R = 95 percent, the error rate is 5 percent. Errors generally come from non-fatal software defects, requests rejected by load control mechanism (when availability is within the required range), or message loss during delivery (e.g., due to congestion or faulty network hardware). Unexpected results caused by problems in back-end processing (e.g., time out or failure of dependent services) are also considered errors. Note that reliability is defined at the application level. It measures how reliable a service performs its function when it is up and running.

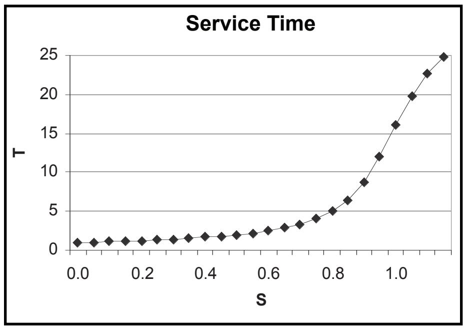
The TSAR metrics relate directly to consumer (end-user) experience with a service by answering the following questions: how fast (T), how much/many (S), how durable (A), and how reliable (R).

Metrics Cube

The TSAR metrics are not all independent. In general, as *S* increases, *T* goes up, and R goes down. The plot of T or R versus S is called a *scalability curve.*

Metrics	Symbols	Notes
Service time	T	Response time for synchronous services. Delivery time for asynchronous services.
Scalability	S	Examples are user load and number of requests per second.
Availability	A	It includes planned maintenance and unplanned down time.
Reliability	R	Due to defects, rejected requests, message loss, etc.

Table 1: *Reference Metrics for SOAs*



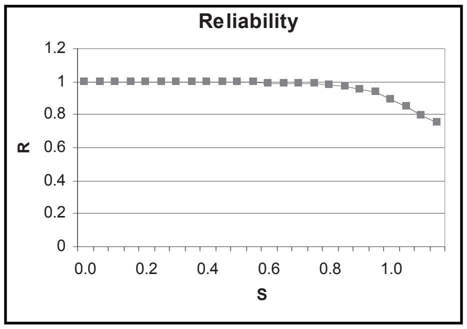


Figure 1: *Scalability Curves*

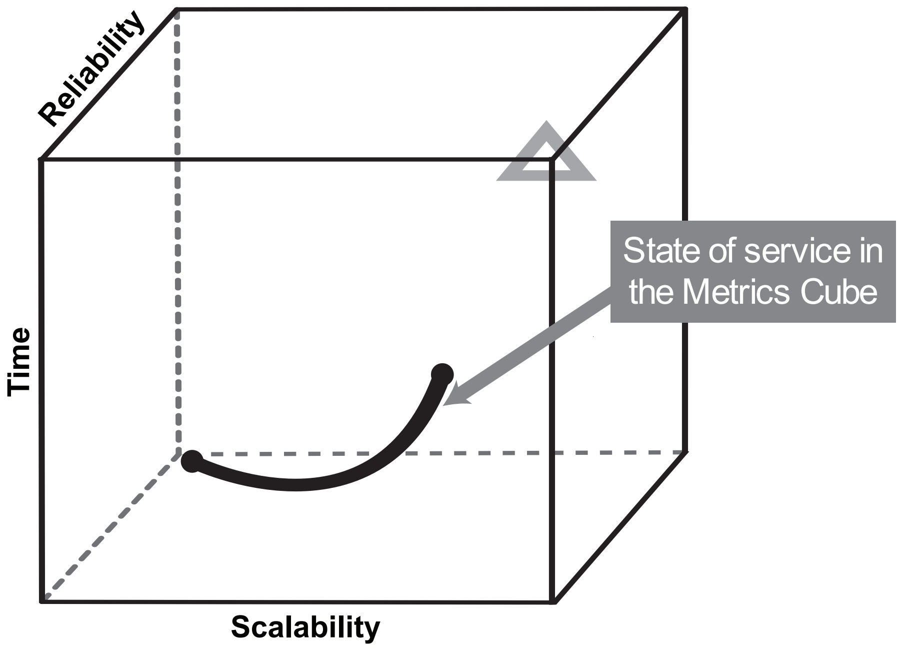
Figure 1 shows an example based on a model with a finite service queue [5]. As S increases, incoming requests/messages spend more time waiting in the queue, causing T to increase. Network latency is not included in this example. Also, in Figure 1, R includes the effect of both software defects and rejected requests (the latter happens when the number of requests in the queue reaches an upper limit). The exact values of the curves in Figure 1 are not important for the discussion here. An appendix in the online version of this article provides further details about the model.

In Figure 1, S is the rate of service requests relative to the maximal throughput μ. Here throughput is defined as the number of completed requests per unit time. As S increases from zero, the throughput increases linearly with S. However, as S approaches one (the rate of service requests approaching the maximal throughput), the throughput plateaus at μ and an increasing fraction of incoming requests are dropped. This is due to the limited computing or network infrastructure supporting the service. The finite queue model simulates this effect by limiting the number of requests/messages in the queue. Below the maximal throughput, one may use throughput as an approximate measure for S. This is convenient since commercial tools typically provide throughput as one of the measures.

The contributing factors to availability do not change as S increases. Thus, availability can generally be considered a constant. However, at very high levels of S, the extremely high demand exhausts the underlying computing and network infrastructure for the service, making it unable to perform any work (similar to a denial-of-service attack). This leads to an abrupt drop in availability. For all practical purposes, it should be determined experimentally that this situation does not occur within the expected range of S. Once this is done, availability can be considered independent of S. The following discussion assumes that this is true.

To help visualize the scalability behavior of a service, one may define a *Metrics Cube* using the minimum and maximum values of S, T, and R. The boundary S_min represents the threshold value and corresponds to the lower bound of normal operation. S_max, on the other hand, is the objective or peak operation value. As S increases, the state of a service can be traced along a scalability curve in the cube, as shown in Figure 2. The metrics cube is useful for specifying SLAs. This is discussed in the next section.

Figure 2: *Metrics Cube*



SLAs

The essence of an SLA is to specify a metrics cube and a required availability (A) range, as well as the statistical calculation of the metrics (e.g. average over an hour, one day, etc.). A nominal process of developing an SLA follows:

1. Service provider measures the scalability curve for a service.
2. Service consumers submit service-level requirements, which can be expressed as a metrics cube and availability range.
3. Service provider compares scalability curves with the requirements.
4. Service provider adds or subtracts computing resources to do the following:
   a. Optimize the scalability curve within the metrics cube.
   b. Meet the desired availability range.

5. Service provider refines and negotiates the SLA with the consumers (based on cost, schedule, and other factors).

In step 4, the scalability curve is shifted within the metrics cube when computing resources are changed. Figure 3 shows a cross-section of the T and S plane in a metrics cube. If the upper end of the scalability curve touches the T_max boundary, the SLA may potentially be violated because the service time will exceed the allowed maximum before S_max is reached. By adding computing resources (e.g. more servers), the curve is shifted downward.

However, if overdone, the curve comes well below T_max at the S_max boundary, indicating over-engineering and wasted computing resources; thus, the optimal configuration is to have the curve touch the upper corner (or somewhat below it as a reserve). Similarly, the optimal curve for reliability should touch the corner at (S_max, R_min). In terms of the metrics cube in Figure 2, the optimized scalability curve would touch the corner labeled with a triangle.

To be compliant with an SLA, the service provider needs to ensure that availability is within the required range. When the service is up and running, the service provider monitors the metrics and ensures that they stay within the required metrics cube.

Closing Remarks

This article defines a set of four reference metrics (collectively called TSAR metrics) for measuring the quality of sustained services in SOAs. It introduces the concept of a metrics cube, which is applied to the development of SLAs and capacity planning of computing resources.

For example, for NCES, a set of threshold and objective metrics have been defined. They are the equivalent of the minimal and maximal boundaries of the metrics cube. For NECC, the TSAR metrics are included in the developer's guide [6] and used in the system engineering process, most notably for SLA development.

An inherent challenge for defining end-to-end metrics (such as the TSAR metrics in Table 1) is that they typically have distributed contributions across network and computing infrastructures. Hence the responsibility for ensuring SLA compliance is shared by multiple entities in a net-centric environment such as the GIG. Nevertheless, the service provider is normally the primary sustainment interface to its service consumers. The provider allocates metrics to its dependent network and computing service providers, either through subordinate SLAs or by explicitly allocating portions of a metric to their responsible entities. The bases of such allocation are the formulas for combining metrics from multiple contributors (e.g. service providers, infrastructures). An appendix in the online version of this article provides such formulas.◆

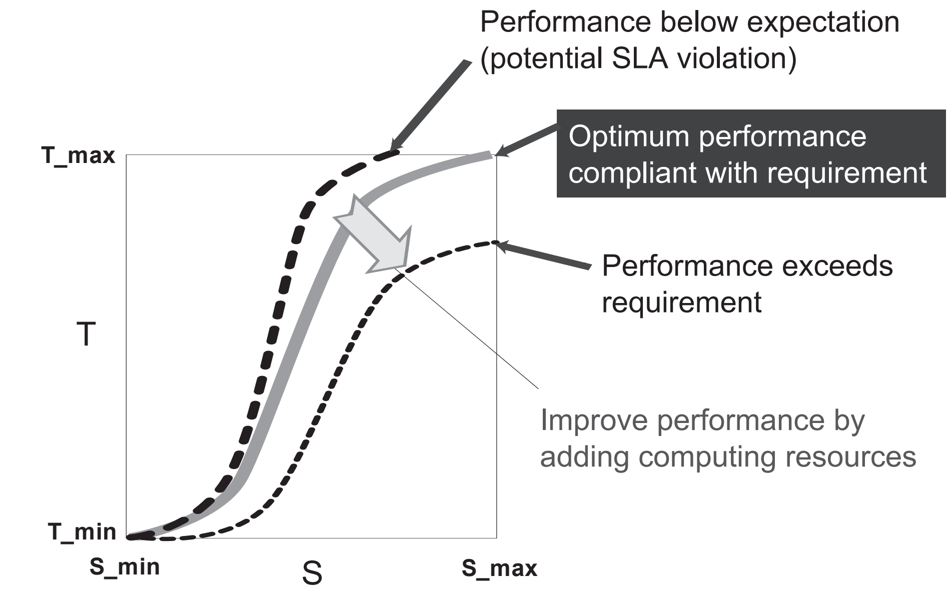


Figure 3: *Optimization of Scalability Curve*

Acknowledgment

The comments of John Schumacher (Principal Systems Engineer at SAIC) on an early draft of this article are greatly appreciated.

References[1]

1. Erl, Thomas. Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall, 2005.
2. United States. Joint Forces Command (JFCOM). "Global Information Grid Capstone Requirements Document." JFCOM 134-01. JFCOM, 2001.
3. United States. Defense Information Systems Agency (DISA). "Initial Capabilities Document for Global Information Grid Enterprise Services." DISA, 2004.
4. Assistant Secretary of Defense, Networks and Information Integration. "Net-Enabled Command Capability Milestone A Acquisition Decision Memorandum." Washington: DoD, 2006.
5. Menascé, Daniel A., and A.F. Virgilio. "Capacity Planning for Web Services: Metrics, Models, and Methods." Prentice Hall, 2001.
6. "Net-Enabled Command Capability Developer's Handbook." DISA, 2007.

Note

1. Some Web sites quoted here require a user account for access. Online application forms can be found on the sites. Some require government sponsorship.

About the Author

Yun-Tung Lau, Ph.D. is Vice President of Technology at SAIC. He has been involved in large-scale software architecture, design, and development for 18 years. Lau has served as chief architect for many software and enterprise architecture projects, from scientific computing and electronic commerce to command and control systems. He also holds a master's degree in technology management. Lau has published many articles in professional journals, and authored the book "The Art of Objects: Object-Oriented Design and Architecture."

SAIC
5113 Leesburg Pike
STE 200
McLean, VA 22041
Phone: (703) 824-5817
Fax: (703) 824-5836
E-mail: yun-tung.lau@saic.com