High performance is an extremely beneficial feature of a system in its own right. It improves the user experience in client applications and the scalability of the servers in the middle tier. In addition, it is a necessary prerequisite for the type of centralized hub-and-spoke architecture used by DGP.

Working to significantly improve the performance of a system forces the simplification of its architecture and code. Consolidation of the logic in a distributed system (centralization) is another technique that can dramatically improve performance.

The combination of high performance, high scalability, simplification and consolidation of the code is responsible for the majority of the reduction in both TCO and TTM for DGP systems. To support a given number of concurrent users, DGP systems require smaller infrastructures, less software licenses and fewer staff compared to other comparable systems.

Also, the end-to-end performance is one of the easiest measurements to collect during the normal use of a system.

Requirements

1. Every action a user takes in a UI should be completed and the UI returned to an interactive state (ready to respond to the next user action) in less than one second.

What: *Completing every action in a UI in less than one second (measured at the client application) defines a time budget for the API call, server processing, network latency and UI processing/display of the results. This performance standard includes full security authentication and authorization for each API call.*

Why: *The reason why every user interaction in an application should be completed in one second or less is explained very well in this video from a Google engineer responsible for web app performance:*
https://www.youtube.com/watch?v=Il4swGfTOSM
The previous video focused on the time to paint the screen. The following link is from another Google engineer explaining the Time-to-Interactive metric: https://medium.com/@addyosmani/the-cost-of-javascript-in-2018-7d8950fbb5d4

Testing/Proof: *Testing the performance of a system.*

2. The system must have high efficiency so that the performance of long-running processes (such as web services) does not degrade over time.

What: *Many web services will have good performance when they are first started, but will gradually slow down over time as the inefficient use of memory forces more and more garbage collections to occur.*

Why: *Software systems should start with good performance and maintain that same level of good performance over time, in spite of heavy use. Similarly, software systems should start with low resource use and maintain low resource use over time. Both of these requirements depend on the efficient use of managed memory.*

Testing: *API's should be built to collect CLR metrics from systems during their actual daily use. These metrics should be stored so they can be analyzed over time.*

3. System performance must be measured under real workloads.

What: *Systems must be designed and built to enable the collection of end-to-end performance measurements while a system is in use without adversely affecting the performance or scalability of the system.*

Why: *Systems must be able to meet the specified performance standards at scale, while under realistic workloads. Generating synthetic workloads for stress/load tests can be very difficult and expensive. Collecting performance metrics from the client applications of a subset of users during the normal use of a system is covered in more detail in the Monitoring section of the requirements and design sections of the documentation.*

Testing: *The developer tools of each web browser can be used to measure the performance of web pages. The monitoring functionality built into native DGP applications can collect performance metrics of a system during its normal use.*

