



# Predicting performance

Sarah Saltzman of Compuware looks in her crystal ball

*Many developers use the waterfall methodology when building and deploying distributed applications.* They follow the sequential process of define, design, build, test, deploy. The problem with this approach is that it can result in a lengthy trial-and-error testing phase or the deployment of an application with inherent performance risks due to problems being discovered too late in the testing cycle. Fixing problems at this late stage of the project can pose a significant risk to the overall project costs and possibly an impact on the release date. The tendency is for testing to be completed in the development labs and for the application to then be 'thrown over the fence' for IT staff to deploy in the production environment. Even if load testing has been completed, performance is not guaranteed in the live production network and inevitably performance problems are found after deployment. This simply isn't good enough in today's competitive business landscape - test teams need to be able to guarantee performance in the live production environment prior to deployment.

This is an age old problem and QA teams have been taking steps to address it. To answer the question of why an application might perform well in a QA lab and then not so in the production environment test teams have looked to two variables; the number of users and the network itself. As both variables are perceived to link to capacity (server or network) test teams have looked to capacity planning to solve the problem of guaranteeing performance of a new application in the production environment. QA teams currently use load testing and network modelling tools in isolation for capacity planning. Although these tools provide valuable insights, they do not enable you to guarantee performance. They fall short on two accounts; the metrics used and integration of the system components. Many capacity planning tools use metrics such as utilisation and uptime to gauge projected performance. However, the most meaningful measure of production readiness is the end user experience; for performance testing this is the end user transaction response time, as this actually drills down to look at the performance level the end user is actual getting. There is also a tendency to test client/server components separately and in isolation - however this

provides no guarantees on how they will perform as part of an integrated system, which contains not only servers and clients, but also the service qualities of the production network itself. That's not to say you should not undertake capacity planning or load testing: in server load testing, in particular, is a great way to find out how the application performs from the server perspective.

The wide area network (WAN) is also an important area to cover, especially when you consider that the application is likely to be used by many concurrent remote users. There are many ways to predict the WAN capacity an application is likely to require. Modelling and simulation tools can be used to look at the sequence of messages and traffic through a network model of the WAN. However, these are often complex to use and most significantly ignore the factor of human behaviour, with the tools often ignoring how and what functionality within the application will be used over the WAN. This has led to people using application pilots. Although this does let you understand how an application performs in a real production environment, it does not give you an analysis or insight into the root cause of a problem and leaves QA and development teams with very little time to correct it.

In order to combat this, some organisations have set up network integration labs to test for application performance. Although these labs can provide in depth information and metrics on how an application will perform in the production environment, they are still unable to provide analysis into the cause of performance problems. The analysis does not tell the QA team whether the problem was caused by the network, the database, the application architecture or low level code. Essentially, QA teams remain in the dark as there is no indication of how the problem can be rectified.

## The proactive approach

Now that we have identified the problems and issues that development teams encounter when testing for performance, let's look at how they can be addressed by testing for performance much earlier in the development cycle. The first thing to do is to profile code as soon as it is created. This can be done using tools

that actually drill down and look at how each piece of code is performing. QA managers should look to examine the code at a very detailed level, looking at things like how much CPU time is allocated to each line of code. This provides test teams with the ability to assess whether an application is too chatty and issuing too many SQL calls for example. By profiling the code and analysing it, developers will be able to see what code is most inefficient and therefore requires tuning. In addition, the analysis provided by the profiling will also allow the team to consider whether changes to the project at a higher level need to be made. For example, the team may decide that there is a need for greater efficiency that can't be solved by tweaking the code and therefore decide to change the application's architecture.

Profiling not only requires you to look at the code, but also the range of technical environments in which it is likely to be running. QA teams should be looking for profiling tools that allow them to see how changes in network bandwidth, latency, load and TCP window size affect each end user's response time. Within the profiling process QA teams should look to undertake thread analysis where analysis on node processing delays, data transmission characteristics and application turns is provided. This allows you to identify application dependencies and therefore see potential performance problems resulting from these dependencies at an early stage.

By undertaking these measures and combining them with load testing and network modelling capabilities QA teams should find themselves in a position where they have ironed out performance problems well before traditional testing stage even begins. In fact when it comes to the load testing stage, those QA teams taking a proactive approach will only be checking to ensure the application performs as predicted, rather than trying to uncover and diagnose performance problems. As we all know the earlier you iron out problems, the less expensive they are, so by taking this approach QA teams should not only save budget but also improve the quality, reliability and performance of applications. **PT**