WOSP 2000 Panel on Architecture and Performance
Sept. 19, 2000

Are We Planning a Maginot Line to Face a Blitz Krieg?

Kevin Mills
National Institute of Standards and Technology
USA

Position: After a long and devastating war with Germany between 1914 and 1918, the French envisioned la Ligne Maginot, a technical marvel intended to fend off future invasions from their neighbor to the northeast. The dawn of 1940 introduced the French to Blitz Krieg, a new form of warfare that combined highly mobile mechanized land forces with close air support. Using these innovative tactics, the invading German forces simply bypassed the formidable Ligne Maginot and quickly took control of France. I suspect that a similar fate may befall current attempts to create an iterative design and performance analysis process intended to produce software architectures in which developers will have confidence. I fear we might attempt to win the previous war. Even if I am wrong, researchers owe it to themselves to consider from time to time whether or not they are asking the right questions. My fears are put forth in that spirit.

My fears are based upon some current facts regarding software development, as well as upon some future trends. Let's start with some current facts.

Some Current Facts

1. Most modern software systems are integrated atop component pieces of various sizes and types (e.g., operating system services provided by Windows, Linux, Solaris; middleware such as CORBA, COM, Java Server pages; databases such as Oracle, SQLserver, and Sybase; development frameworks such as Java, MS foundation classes, web browsers; and network protocols such as TCP/IP, HTTP, and Real Audio-Video).
2. Increasingly, software applications are moving to a web-based delivery paradigm in order to gain more platform independence. This current trend can be expected to accelerate.
3. Software platforms and their many components and frameworks are continuously and independently updated on a node-by-node basis using web-based software download mechanisms.

Implications

1. Software architectures will increasingly consist of software frameworks intended to enable or support components yet to be written.
2. The software frameworks and the components that use them will evolve independently through revisions and versions so that the number of combinations of deployed software will be too large to analyze rigorously for performance in advance of deployment.
3. Software will be deployed on a widely varying set of hardware nodes – so assumptions made about the capabilities of nodes might prove invalid in specific situations.

Some Future Trends

1. Current industry initiatives (e.g., Jini, Universal Plug and Play, Service Location Protocol, Salutation Consortium, Home Audio/Video interoperability specification) to develop adaptive and self-configuring software promise a future paradigm for software architectures where a software service can be developed, discovered by software clients, and configured so that the software client can use the service. In this model, the software service and client can be developed independently without ever necessarily being designed in an integrated manner.

2. Even embedded systems, such as cell phones and personal digital assistants, are being designed so that their software platforms can be extended and so that application components can be downloaded and varied over time.
3. Distributed systems will increasingly operate within mobile and wireless networks, where continuous connectivity can neither be assumed nor assured, and where the quality of communications bandwidth can vary greatly.

Implications

1. When services and clients are designed independently and then rendezvous with the exchange of an interface through which a client can invoke methods on a service, different implementations of the same type of service can be designed very differently – for example, a thin interface can be provided or a thick interface. Designs of this sort can lead to starkly differing performance for what appears to be the same client application.
2. When a software application is designed to operate over network connectivity that can vary in quality, even perhaps including periods of discontinuities, performance predictions that depend on the characteristics of the communication channel can prove highly inaccurate.
3. When services and clients are intended to discover each other and configure themselves for use, clients might wish to take into account the performance that a service can provide, and the performance that a service can provide will depend in part on the number of clients currently using the service, as well as on the number of clients who decide in the near future to use the service.

Questions for Software Performance Prediction in the Post-Maginot-Line Era

1. Will all software frameworks, components, and node platforms need to have a model of their own performance embedded in them?
2. Will node software need to maintain a current, executing simulation model of its own performance running in parallel with itself in order to predict the performance of the software frameworks and components that have been downloaded or that might be injected into a node?
3. Will software components need to inquire about the capabilities and expected performance among a set of nodes in order to select a node to execute on so that the performance requirements can be satisfied?
4. Can application software predict performance expectations while also exploiting the potential robustness in redundant nodes and services in a distributed network?
5. Will nodes and components benefit from built-in performance measurement gauges that can accumulate historical performance data in an effort to better inform predictions of future performance?
6. Can software architectures be designed to adapt their deployment of components in different ways in order to provide specified performance guarantees in the face of varying capabilities and utilizations among the underlying physical resources, such as processors, memory, communication channels, and input/output channels?

Bottom Line: Tomorrow's software environments will look more like today's trends and less like yesterday's failures. Is our current research aimed at solving yesterday's problems or at addressing tomorrow's software performance needs? The panel, specifically, and WOSP in general, should consider the future needs for software performance prediction, and should discuss research directions that can help to meet those needs. I would even recommend eschewing today's problems, because we can't solve today's problems with tomorrow's research – we can only hope to solve today's problems with yesterday's research. Similarly, we can only hope to solve tomorrow's problems with today's research. Is our research on target for the future?