Alternative Technologies

Enterprise Integrity: Proving Scalability

Vol. 2, No. 8

Need reassurance that your e-Business or EAI implementation will both scale and perform? Well, I've yet to encounter either a vendor or a consulting firm in this market that will not claim that their systems are scalable. And I've yet to encounter a clear and defensible statement of what such scalability means or how it was proven: a claim is not proof. True, successful implementations exist that meet complex requirements and make great reference accounts and case studies. Unfortunately, e-Business and EAI systems are increasingly less difficult to implement, but nonetheless depend on complex, multi-function, multi-layer, and multi-component architectures that are inherently difficult to benchmark.

Having worked with complex, mission critical systems for over twenty-five years and having focused on their performance and scalability (including designing, evaluating, and running benchmarks), I'm pretty skeptical of loose claims about scalability and performance. If those claims aren't well-defined, applicable to your requirements, quantifiable, and repeatable, then they have no value to you. Using the software stack of the last two issues, the following summarizes some guidelines on what you should look for in a benchmark or proof of scalability.

Architecture – A clean architecture facilitates your ability to understand which functions scale and which are likely to be bottlenecks, helps you evaluate a vendor's claims of scalability, and even helps you design your own benchmarks. The key is being able to divide and conquer, isolating and identifying problems. In general, more distinct components (to implement a function) means lower scalability and performance.

Integrated Design/Development/Deployment Environment – If you need to extend or enhance functionality, or develop either adapters or wrappers, benchmark the time and resources required to reach deployment and also to make further changes. Don't forget to test the scalability and performance of any components you develop. If libraries are provided, non-uniformity of the API can be a clue to potential bottlenecks.

System support facilities – These facilities determine how well simultaneous requests (or processes) are supported. Check for scalability in the simultaneous load such as numbers of threads, how quickly threads are initialized and terminated, how well threads are spread across resources (load balancing), and how well memory or disk is utilized.

Intelligent adapters (a.k.a. connectors) – Evaluate adapters limits on the number of simultaneous requests and how performance scales with load. Such tests are very sensitive to the particular business function being requested, so use a variety of request types. Consider scalability with size of the object or message, whether inbound to or outbound from the

application. If the adapter supports monitoring and tracking, at what overhead cost and how does this scale with tracking granularity? Measure operational overhead when adapters are added, removed, started, or stopped online.

Transport – What is the message throughput of the transport and how does this scale? If multiple methods of communicating messages are supported, test each. Bridges between synchronous and asynchronous protocols are often bottlenecks and so deserve special attention. Check for transactional integrity under various failure modes.

Routing / Message Broker – As noted last month, the throughput capabilities of your broker forms a floor on system performance. Tests should prove linear scalability within and beyond a single broker configuration. Message overhead can seriously degrade performance, so benchmarks should test throughput as a function of routing complexity.

Re-formatting – The amount of message re-formatting required depends on the message protocols and formats supported by the various components. A single new component that demands a new format can introduce serious overhead, depending on what must talk to what. Benchmarks should test performance against the number of message formats required.

Transformation – Data transformations per minute will vary with the number of transformation rules, their complexity, number of engines, and configuration (hub, multi-hub, or distributed). Check for rules cache overflows and the performance cost of refresh or modification. All should be independently tested. If the engine requires a specific data format such as XML, overhead needs to be varied.

Business process support – All the performance issues of transformation engines apply to process engines. Additionally, definition, redefinition, and process state update and interrogation need to be benchmarked. Process characteristics have a strong effect on throughput: number of branches and their type (simple condition, event, voting, priority, etc.), number of nodes in the overall process definition, node fan-in and fan-out, and so on. Benchmarks should identify performance for application to application, manual workflow with worklist management, and hybrid process definitions.

Obviously a general benchmark would be difficult to define. Testing scalability and performance of a prototype application may be your best bet, but don't get trapped by simplicity. Simple tests tell little and often give false comfort. True benchmarks should report statistical results (rather than best numbers) and record the effort involved in achieving those results. Obtain such information before you buy. Enterprise integrity, reflected in your EAI and e-Business systems, depends on predictable high performance and scalability.