iTKO LISA™

The Next Frontier for Virtualization: Over-Utilized Systems

Identifying and eliminating the most critical and costly constraints from the enterprise software development and testing lifecycle.

Whitepaper: August 2009

By Ken Ahrens, Sr. Solution Architect,
and Jason English, VP Communications
iTKO LISA

iTKO LISA
1505 LBJ Freeway
Suite 250
Dallas, TX 75234
USA

www: http://www.itko.com
email: info@itko.com
tel: 877-BUY-ITKO (289-4856)

©2009, Interactive TKO, Inc. All rights reserved.

Introduction

Hardware virtualization technology is now par for the course in enterprise IT infrastructures. Enterprises can realize rapid cost savings through the use of hardware virtualization to reduce server proliferation, consolidate desktop environments, and run numerous apps with a smaller combined hardware footprint. The key value espoused by leading virtualization vendors (such as VMware, Microsoft, Citrix, etc.) is making better use of "under-utilized" servers that do not require dedicated system resources.

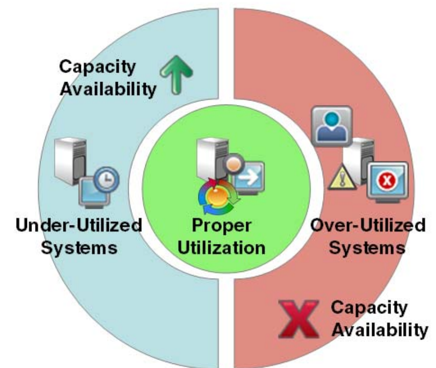
But what are we supposed to do about the kinds of systems that are not good candidates for hardware virtualization? What about the critical **over-utilized** applications that create bottlenecks among teams that need access throughout the software development and testing lifecycle? Hardware virtualization has a much less compelling value proposition in this scenario.

A strategy to apply in this case is the use of Service Virtualization (SV) to reduce multiple team dependencies on over-utilized systems, allowing them to work in parallel. This article will explain the utilization patterns encountered in enterprise IT environments, and which type of Virtualization (Hardware virtualization, Service virtualization, or a combination of both) should be applied to receive the greatest value.

Defining Utilization Patterns

For simplicity, we'll just categorize enterprise systems into one of 3 categories of utilization and figure out what this means from the standpoint of virtualization:

- **Under-Utilized** systems have low CPU/memory/disk usage, or otherwise there is significant capacity on the system to run additional applications.
- **Over-Utilized** systems may have high CPU/memory/disk usage (for instance a transaction mainframe or multi-terabyte system of record), or perhaps they are services outside of the control of the Enterprise, with usage limitations or per-use costs (such as third-party services, cloud computing, SAAS, etc.).
- **Properly Utilized** systems, which are working optimally, and can be left alone.



Causes of Under-Utilized Systems:

- ***Infrequently Used*** – Low usage does not necessarily equate with low business value (as a rarely used service could process very expensive orders), but it does usually mean that the system could get by with less processing power during those hours when it is not used.
- ***Excess Capacity*** – For teams that didn't have the time or resources to do significant capacity planning tests, many architects hedge on the safe side of providing too much capacity rather than too little. At some enterprises this creates a capacity glut of servers and bandwidth.
- ***More Efficient Software*** – Developers and testers can of course build very robust software, or perhaps optimize the software performance and use of system resources over time, to get their services to run very efficiently.

In many of the above cases of under-utilization, conventional Hardware Virtualization approaches have proven successful in eliminating excess capacity by consolidating servers and applications. Performance testing and benchmarking can help identify good targets for hardware virtualization, but do these under-utilized systems represent the greatest cost and risk to IT?

Causes of Over-Utilized Systems:

Why would systems end up over-utilized and unable to handle ongoing transactions, particularly in supporting software test and delivery teams?

- ***Common Bottlenecks*** – Literally this means that many teams depend upon access to the same constrained system or service. Usually this is due to a design decision early on, but in shared environments, the usage pattern is hard to predict when a service is called upon by many customers, and other teams' services in unplanned ways.
- ***Limited Capacity*** – More likely for services that have been in production for some time, it's not hard to imagine that new technology teams get hardware upgrades, while older technology gets left in the dust. Or the production environment may be re-built, but pre-production servers needed by development and testing may not get the same upgrade due to budget constraints.
- ***Inefficient or Unscalable*** – Unfortunately, technologies and platforms may not provide the scalability initially advertised.

While we hope that responsible architecture, integration and deployment strategies will result in proper system utilization, it typically doesn't work out that way in today's heterogeneous, distributed application environments. There are simply too many variations in how the systems are connected and used, and too many components in the environment that aren't under one team's control.

The Challenges of Over-Utilization

Since under-utilization is a problem that can often be solved with conventional forms of virtualization, let's take a look at why over-utilization of key systems can create significant problems for the enterprise.

Over-Utilization Due to Common Bottlenecks

Well-respected enterprise application design patterns are actually causing software development lifecycle bottlenecks where numerous teams rely on a key system or technology. These systems become new constraints to development and testing, and push the original goal of time and cost savings further out of reach.

Design of Core Services – In some Enterprise IT organizations, a few key services are so fundamental that they are used in a majority of projects. From the standpoint of re-use this is exactly what we were looking for, but while designing, testing, and running our applications, these core services create dependencies which slow teams down, because unlimited access for software teams is often considered too risky.

Examples of core services include:

- ***Customer/Account*** – An example of one of the most commonly created core services, so there is a single place to get customer records.
- ***Billing*** – Customers don't want to receive multiple bills from the same vendor, so this is also a common core service to create.
- ***ERP System*** – Especially for internal purposes (HR, Financials, etc.) companies adopt a single platform and require everyone to integrate.

Customer Example:

A large North American Telco had issues in replicating their core SAP system across a myriad of environments. Only 3 of 7 non-production environments have an SAP instance they can use. Also the training system does not have a copy of SAP, so they cannot train their Customer Service Reps on applications that integrate with SAP until they are rolled into production. The result is that bugs are not found until it is too late to repair them and still deliver projects on time.

- **Data Warehouse** – Numerous services depend upon the same set of data, especially for account information. In order to make data available for development purposes, this data usually comes from a production "refresh" as needed, and is put into pre-production.
  - ***Data setup & teardown*** – Managing relevant, stable and secure data in the environment can become a huge drain on team productivity, sometimes consuming 60% or more of testing time.
  - ***Data volatility*** - When data is treated as community resource for multiple development and test teams, other teams can pollute the test data (for instance deleting a key user required for your regression test suite).
  - ***Secure data*** - In addition, live data must be suitably scrubbed or "desensitized" to protect unauthorized tester access to account information, which is a time-consuming process.

- **Cloud Services** – Saving money by using external service vendors on a pay-per-use basis is great, but not when they get in the way of being able to do testing and development. This means we have another team we need to coordinate with from a data standpoint (test accounts, etc.) and potentially we are charged a usage fee when calling on these services.

  Incremental fees are fine if they are a part of a revenue-generating customer activity in the live application, but they are a problem when they become a development cost to the business. For instance, how can we avoid skyrocketing fees for load tests that include a cloud service, or analyze negative ramifications if the cloud service doesn't meet their SLA?

Customer Example:
A regional insurance company needed to be cautious about how they run their performance tests for adding new auto insurance customers. Among other things, the new client application performs a credit check and runs an automobile history report. During one of their routine load tests against the new app, they sent actual requests for automobile history reports to a third party, resulted in a $15,000 access charge. After this unexpected charge they simply chose not to test any third-party service calls until they reach production.

Over-Utilization Due to Limited Capacity

With the low cost of hardware, you would assume that having environments that were just too small would be a thing of the past. But there are reasons why some key systems may continue to be under-sized:

- ***Unexpected Growth*** – It's great when a new core service gains popularity and adoption across the enterprise. Perhaps it is an indication that well thought-out designs are being adopted. But the downside is that too much interest can overload a new service under a deluge of transactions. Or many parties may need to jockey for position to get their test data loaded into the new service.
- ***IT Re-alignment*** – In some cases, companies (and their entire IT infrastructures) have been acquired, or hardware was only partially upgraded. For instance, the production environment is fully built, but pre-production is scaled down. This can be a problem when multiple teams are trying to integrate their systems to a common environment that is a moving target. Common strategies of dealing with this problem are to schedule the execution of tests across all teams, so they are not all hitting the constrained resource at the same time, which limits the new company's ability to take new functionality to market.
- ***Expensive to Duplicate*** –Today's composite apps can involve a dozen components with the overall application no longer in one team's full control, so virtualizing one system doesn't make much of a difference, because of the interdependent nature of the application. Who cares about saving a few thousand dollars in hardware costs, when the software and maintenance effort of a complex system can cost millions?

Over-Utilization Due to Inefficient Applications

As much as we'd like enterprise applications to run smoothly, it's common for systems to run slowly. This problem is exacerbated when your application might be performing poorly, and to improve this metric you depend on testing against other services/systems that are also running slowly.

- ***COTS Products*** – Vendors have to write "one size fits all" types of applications. The result is that without investment in customization, these apps will run more slowly than if your exact use case was coded by hand. As the shift goes from developers writing nearly all the code in the past (e..g legacy COBOL) to focusing more on just writing the business logic (e.g. BPM), the amount of third-party code is skyrocketing. Unfortunately there isn't much that can be done to optimize third-party code in the short run, which may drastically affect testing and analysis of the code that your team *does* write.
- **Older Technology** – Despite earlier predictions, the mainframe does not seem to be going away anytime soon. If your mainframe has a large partition for development and test then consider yourself lucky. Most IT shops have to deal with very small LPARs resulting in scheduling conflicts or adjustments that must be made to test strategy.

Customer Example:
A large financial services company was unable to properly load test applications going to production. While they had dedicated environments and test cases, their small test mainframe was at CPU capacity while the non-mainframe apps were only at 30%. They never learned the breaking points of their system until they actually pushed code to production, a risky proposition. Eventually they found out the hard way when they had an outage that took out all transactions for their largest customer.

- **Popularity Reduces Flexibility** – Being popular can be a curse. If you change something fundamental about your service, lots of other teams may need to be involved. Those other teams may have more clout, and could try to block your code changes from being made. This can result in possible efficiencies never making it into your core services, so they grow slower over time.
- **Compliance** – Implementing features such as robust logging, audit trails, governance, monitoring, security, etc. can significantly slow your app. We typically think on the positive side that these features bring to visibility, but early in your development cycle you may not care too much about these benefits.

Hardware Virtualization Can't Solve Bottlenecks of Over-Utilized Systems

We've reviewed why systems are becoming over-utilized, as compared to under-utilized. The problem is that the hardware and desktop/OS virtualization tools we have at our disposal are great for optimizing under-utilized resources, but they are often incapable of virtualizing the behavior of over-utilized systems.

Even the largest virtual server deployment and centralized management of VMs may not allow for the virtualization of 3rd party cloud services or mainframe assets. However, these same concepts of virtualization can be applied in new ways to eliminate bottlenecks in our overtaxed core services.

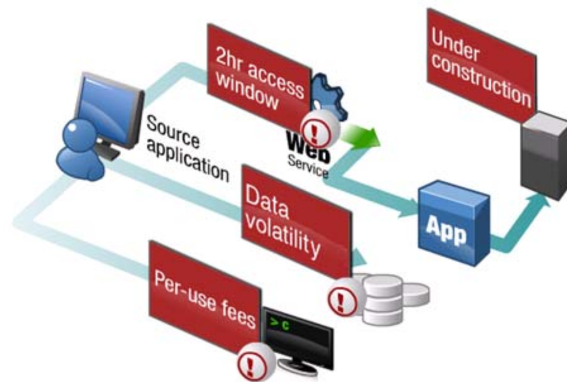
Solution: Service Virtualization (SV)

Service virtualization involves the simulation of software service behavior and the modeling of a virtual service to stand in for the actual system during development and testing. Think of it like a "stunt double" for your most constrained, critical applications. Service virtualization addresses each of the aforementioned hardware virtualization limitations such as:

- Providing 24/7, on-demand access to ready test environments managed on your terms
- Removing capacity constraints of over-utilized systems
- Addressing test data volatility across distributed systems and conflicts among teams
- Reducing or eliminating the cost of invoking 3rd party systems for non-production use

While Hardware Virtualization focuses improvements on the systems an IT team can control, such as servers in the data center and desktop, the complementary practice of Service Virtualization is geared toward optimizing the distributed and shared types of applications that are resistant to being imaged as a hard drive in a hypervisor: the over-utilized systems such as mainframes, partner-managed services and components under development by other teams.

Let's look at a sample enterprise with 3 downstream dependencies after applying SV techniques (from the bottom up):



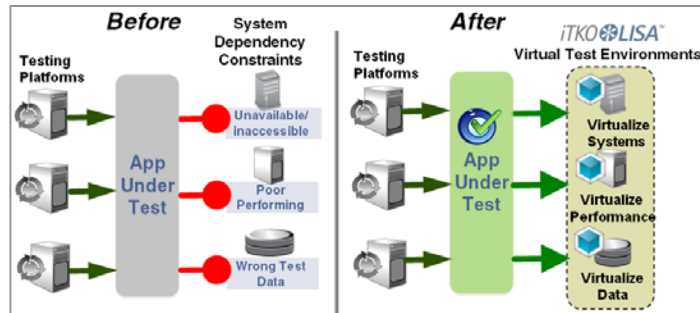
- ***Mainframe access problem eliminated:*** During an instance when the key transaction system is available, it is captured and reproduced as a sophisticated model of that behavior, which is then hosted as a virtual service and accessible 24/7 in a virtual service environment.
- ***Database access and data volatility eliminated:*** Rather than having to negotiate set up and removal of test data sets from multiple sources and partner systems, the team can get a stable set of data to validate scenarios without impacting the live systems or other teams' test data.
- ***Mitigated an unavailable or incomplete system exposed through a Web Service:*** Construct a Virtual Service from a WSDL, SOAP documents, XML samples, and other artifacts, whether or not the service, or the back-end system is available for testing. If desired, the Virtual Service could be accessed as a "pass thru" to intercept test transactions, while allowing live transactions to continue through to the real service and legacy application.

Examples of how SV Answers the Challenges of Over-Utilized Systems:

Solving a Common Bottleneck – When each team can get their own replicated test environment and data provided as a Virtual Service at minimal cost, there is no longer a need to compete for testing time during a small availability window, much less secure a capital expenditure just to get access to a hosted service-based system.

Customer Example: *A leading US airline replicated much of the functionality of a partner's reservation system that charged high per-use fees when used for testing purposes, as well as restrictions on usage due to the fact that millions of transactions needed to be handled on the system daily. Switching to a virtual service model of the system allowed multiple teams to have their own stable environment and data for test scenarios, as well as saving millions of dollars annually (up to $12M) in non-revenue-generating access fees for the live service.*

Solving Improper Capacity – Trying to pinpoint the reason for functional errors and performance problems in a modern distributed software architecture is becoming increasingly difficult. IT departments can seek to throw more hardware and software at the problem, but that cannot solve the costly and time consuming software setup and configuration effort of enabling test and pre-production environments for multiple distributed teams to use.



Customer Example: *A leading bank which has grown by acquisition found that they could not effectively load test middleware which tied more than 70 acquired systems together using their existing tools. They had several developers attempting to hand-code "stubs" and "responders" to replicate the environment for 2 years, with little success. Using Service Virtualization techniques, two test engineers configured realistic models of the current bank middleware systems. They replaced the multi-year "stub" project in a few months and grew the practice to include another half-dozen teams across the bank. They have avoided up to $30M in costs for provisioning new environments over the course of a year.*

Solving Inefficiency – Every business faces increased pressure to meet customer and regulatory demands with greater agility, and leveraging distributed software components and integration frameworks can add agility to our ability to deliver expected business outcomes. At the same time, the service-oriented approach to delivering software makes it harder to ensure service level performance and quality, due to an increased rate of change in the environment.

Customer Example: *A US Federal agency needed to prove that their technology selections and architecture elements were performance ready, before they were put in place in the live environment. By virtualizing the rest of the environment and its expected response times and load variability, they were able to compare benchmarks and select the appropriate technology for their architecture, without needing to "bang" on critical live systems used for weather, air traffic and defense purposes.*

Summary

Hardware virtualization is becoming ubiquitous in large enterprises, offering immediate cost savings and efficiency. While the concept of desktop and server virtualization has great value for optimizing less utilized systems, it is challenged in replicating heavily utilized or constrained systems such as mainframes, incomplete components, or third-party services.

Consider how much you invest in "under-utilized" hardware and desktop OS infrastructure, versus the high cost of integrating and maintaining traditional software test environments. There's a high probability that the development and integration cost of those traditional test environments exceeds your hardware costs by several orders of magnitude. The prohibitive expense of software test environments means applications are not tested early or often enough, which leads to even costlier failures in production. Virtualization of the behaviors of over-utilized systems can address this serious shortcoming.

Instead of copying the contents of a hard drive or replicating a piece of software running in an OS, Service Virtualization focuses on modeling the communication paths "between the boxes" –

components such as web services, databases, RESTful services and asynchronous messaging. This lets teams break their dependencies on the bottlenecks which are preventing them from getting their jobs accomplished.

Development and testing teams need access to an ever-increasing number of services and systems of record that are not readily available. Service Virtualization is a strategy for letting teams take the principles of virtualization beyond the data center, simulating more distributed, complex environments, where significant value remains to be realized.

Request a Business Case Assessment from iTKO

Visit us today at ***http://www.itko.com*** and ask iTKO for a Business Case Assessment. We will demonstrate how our LISA Test, Validate and Virtualize solutions can help your teams deliver software with quality and agility, with less cost, risk and effort.

For more insights on Virtualization:

- iTKO Blog on Virtualization topics: http://blog.itko.com/virtualization
- iTKO LISA Virtualize software and whitepapers: http://www.itko.com/products/virtualize.jsp

About iTKO

iTKO helps customers transform the software development and testing lifecycle for greater quality and agility in an environment of constant change. iTKO's award-winning LISA™ product suite can dramatically lower quality assurance costs, shorten release cycles, reduce risks, and eliminate critical development and testing constraints by virtualizing IT resources to provide accessibility, capacity and security as needed across interdependent teams.

LISA test, validation, and virtualization solutions are optimized for distributed, multi-tier applications that leverage SOA, BPM, cloud computing, integration suites, and ESBs. iTKO customers include industry leaders such as eBay, American Airlines, Citigroup, Time Warner, SwissRe, Bank of America and government agencies including the U.S. Department of Defense.