Utilization is Virtually Useless as a Metric!

Adrian Cockcroft – Netflix

We have all been conditioned over the years to use utilization or %busy as the primary metric for capacity planning. Unfortunately, with increasing use of CPU virtualization and sophisticated CPU optimization techniques such as hyper-threading and power management the measurements we get from the systems are "virtually useless". This paper will explain many of the ways in which the data we depend upon is distorted, and proposes that we turn to direct measurement of the fundamental alternatives, and express capacity in terms of headroom, in units of throughput within a response time limit.

1. Background

The main new theme at the CMG 2005 conference seemed to be CPU virtualization. Many people using VMware, Xen, Solaris Zones, Hyper-Threading and other virtualization facilities are finding that their measurements of CPU utilization don't make sense any more. The primary performance management vendors were discussing how they are building some support for virtualization concepts into their tools. While the mainframe world has been dealing with virtualization for many years, the Unix/Linux and Windows world has now adopted a bewildering variety of virtualization techniques, with no standard metrics, and often without any indication that the CPU has been virtualized!

2. The Good Old Days

Lets start by examining the way we have traditionally collected and manipulated utilization, and highlight the underlying assumptions we have made in the past. These assumptions are not always stated, and in many cases they no longer hold.

Utilization is properly defined as busy time as a proportion of elapsed time. Utilization can also be obtained by multiplying the average throughput of the service by the average service time. If an operation takes 10ms to process and the service is performed 50 times a second then the utilization will be 50 * 0.010 = 0.5, usually written as 50%. When applied to a simple queue, with work arriving at random, we also know that the response time and the queue length increases at high utilization.

The most fundamental assumption here is that the average service time is constant, and does not depend upon the load level. This is one of the assumptions that has been broken by virtualization, Hyper-threading and variable speed power-saving CPUs. We will discuss these technologies later in this paper.

The simple queue has a single processing element. In reality, today's computer systems do not form simple queues and do not have a single processing element. Operating systems expose an abstraction that looks like a simple set of processing elements, and provide metrics to measure that abstraction, such as overall CPU utilization.

Common capacity planning techniques either assume that there is a single processing element, or a fixed number of identical processing elements, with characteristics that only change on relatively infrequent "upgrade" events. In fact <u>none of these assumptions can be relied upon for the most common systems in use today</u>. This is not an obscure feature of a specialized system design, it's baked into the low cost mainstream products that everyone uses!

Response time for a simple queue with a large number of users can be modeled as the service time divided by the unused capacity. For an idle system, utilization is near zero, so unused capacity (one minus the utilization) is near one, and the response time is similar to the service time. For a busy system, utilization is near one, and unused capacity is much less than one. So dividing into the service time, the response time is large.

For complex systems the utilization metric is not a reliable indicator of capacity. In some cases the reported utilization metric will reach 100% well before the system is out of capacity, so response time stays low until the load reaches a much higher throughput level. In other cases the utilization metric is not linearly related to throughput, so for example a 10% increase in throughput could cause the reported utilization metric to increase from 40% to 80%.

3. I/O Utilization

This paper is mostly concerned with CPU utilization, but there are lessons to be learned from I/O utilization.

In days of old, when tools like iostat were first written for Unix systems, a disk was a simple thing. It could seek, read and write, and while it was doing those things it was a busy disk and you had to wait for it to finish. Thus iostat reported disk utilization and when the disk got busy, the response time got bad. The advent of intelligent disk controllers changed all that. For example the SCSI protocol allows multiple commands to be sent to the disk, with completion in any order. Each disk consists of a pair of queues in tandem, one of requests that have not been issued, and one of requests that are currently inside the disk itself.

Storage virtualization has always been an issue. From the beginning, disks have been partitioned, and the utilization metrics of each partition have a complex relationship to the utilization of the disk as a whole. It is also very difficult to see the utilization of an individual file. Partitions can also be combined using a volume manager into stripes, concatenations, mirrors and RAID volumes.

Some versions of Unix still have a simplistic iostat that reports on a single simple logical queue, the more enlightened iostat commands have more useful options. For example on Solaris, you can see the iostat data on a per partition, per disk, per controller and per volume basis, and it does separate the wait queue metrics from the active queue metrics. To get per-volume stats you have to use the bundled Solaris volume manager, add-on products such as Vertitas Volume Manager don't work with iostat. Try iostat –xpnCez (think "expenses" to remember this combination) to get lots of detailed information.

Modern disk subsystems are built using storage area networks (SANs) and cached RAID disk controllers. The logical units (LUNs) that appear to the operating system to be single disks, are actually complex combinations of RAM cache and large numbers of disks.

In a high availability SAN configuration each LUN can be accessed via four paths. Dynamic multi-pathing spreads the load over all four paths and transparently hides failures. The same LUN appears as four separate LUNs to iostat, which has no way to know the actual configuration.

Multiple host systems share access to storage over a SAN. In many cases the same physical disk will be shared by several hosts. There is no way for each host to know how much I/O capacity is being used by other hosts, and poor or unpredictable response times can sometimes be traced to this effect.

When iostat thinks that a LUN is 100% utilized it makes the false assumption that a single service element is processing a single queue of requests. In fact multiple service elements are processing multiple queues of requests. A large number of concurrent requests can usually be processed, but there is no way for the iostat command to find out what the internal capability of a LUN is, so it cannot reliably report utilization.

Sophisticated I/O trace based capture and modeling tools such as Ortera Atlas (http://www.ortera.com) can determine what is happening at the filesystem and volume layers, and calculate a new metric called the "capability utilization" of the underlying LUN.

This problem is directly analogous to the situation we now find in the CPU realm.

4. Intel Hyper-Threading

Hyper-Threading is used by most recent Intel server processors. In summary, when a CPU executes code, there are times when the CPU is waiting for a long operation such as a memory read to complete. Since the pipeline of functional units inside the CPU is stalled for a very short period this is often called a "pipeline bubble". By adding a small amount of extra complexity to the CPU, it can be made to keep track of two separate execution states at the same time, and two completely isolated code sequences can share the same pipeline of execution units. In this way when one thread stalls, the other thread takes over and the pipeline bubbles are eliminated.

Intel describes this technology at http://www.intel.com/technology/hyperthread/ and Yaniv Pessach of Microsoft describes it at http://msdn.microsoft.com/msdnmag/issues/05/06/HyperThreading/default.aspx. I tried searching for information on the performance impact and performance monitoring impact and found several studies that describe improvements in terms of throughput that ranged from negative impact (a slowdown) to speedups of 15-20% per CPU. I didn't see much discussion of the effects on observability or of effects on response time. The Intel CPU pipeline is already quite efficient at keeping pipeline bubbles to a minimum, which is why the speedup isn't huge. An improvement of this magnitude does however represent a useful increase in capacity.

To summarize the information that I could find, the early measurements of performance included most of the negative impact cases. Software and hardware improvements mean that the latest operating systems on the most recent Intel Pentium 4 and Xeon chipsets have a larger benefit and minimize any negative effects.

From a software point of view, Intel advises that Hyper-Threading should be disabled for anything older than Windows XP, Windows Server 2003 and recent releases of RedHat and SuSe Linux. Older operating systems are naiive about Hyper-Threading and while they run, they don't schedule jobs optimally. In addition older Solaris x86 releases are also unaware of Hyper-Threading and Solaris 9 and 10 for x86 include optimized support. I have also been told that the Linux 2.6 kernel includes optimizations for Hyper-Threading that have been back-ported to recent patches for the Linux 2.4 kernel.

From a hardware point of view, the benefit of Hyper-Threading increases as CPU clock rates and pipeline lengths increase. Longer waits for pipeline stalls due to memory references and branch mispredictions create larger "pipeline bubbles" for the second Hyper-Thread to run in. It is possible that there may also be improvements in the implementation of Hyper-Threading at the silicon level as Intel learns from early experiences and improves its designs.

The fundamental problem with monitoring a Hyper-Threaded system is that it invalidates one of the most basic and common assumptions made by capacity planners. For many years the CPU utilization of a normal system has been used as a primary capacity metric since it is assumed to be linearly related to the capacity and throughput of a normal system that has not saturated. In other words, you expect that the throughput at 60% busy is about twice the throughput at 30% busy for a constant workload.

Hyper-Threaded CPUs are non-linear, in other words they do not "scale" properly. A typical two CPU Hyper-Threaded system behaves a bit like two normal fast CPUs with two very slow CPUs which are only active when the system is busy. The OS will always report it as a four CPU system.

Whereas the service time (CPU used) for an average transaction remains constant on a normal CPU as the load increases, the service time for a Hyper-Threaded CPU increases as the load increases.

If all you are looking at is the CPU %busy reported by the OS, you will see a Hyper-Threaded system perform reasonably well up to 50% busy then as a small amount of additional load is added the CPU usage shoots up and the machine saturates.

To be clear, the use of Hyper-Threads does normally increase the peak throughput of the system, they often provide a performance benefit, but they can make it difficult to manage the capacity of a system.

5. Sun CoolThreads “Niagara”

The latest member of the Sun SPARC processor family is the UltraSPARC-T1, also commonly known by its code-name “Niagara”. In some ways it is similar to Intel Hyper-Threading but it takes the concept to an extreme level. Sun provides some details of its features at http://www.sun.com/processors/UltraSPARC-T1/details.xml and there are many more links to detailed discussions from Richard McDougall’s blog entry at http://blogs.sun.com/roller/page/rmc?entry=welcome_to_the_cmt_era. The design goal is maximum throughput per watt for commercial workloads such as Java, database and web serving. This is a widely used low cost platform from Sun, so it is important to understand how to manage it from a capacity planning perspective.

The T1 contains eight CPU cores with four threads each that process integer calculations only, and a single shared floating point unit, all on the same chip. The CPU pipeline design is deliberately kept simple, and if only one thread is executing a large number of pipeline bubbles occur. The single thread performance is much slower than a typical Intel or AMD CPU, but since there are 32 threads on the T1, its overall throughput is comparable to several conventional CPUs for integer work. For floating point calculations, the single shared floating point unit gives poor performance that does not scale as more threads are added.

The Solaris operating system reports standard system monitoring metrics that show a T1 as a 32 processor system. One single CPU bound thread cannot make it more than about 3% busy. Unlike the Intel Hyper-Threaded systems, the first thread on each core does not come close to fully utilizing that core, so additional threads get quite good scalability. However, a core running four busy threads is unlikely to provide four times the throughput of a single thread, and the actual scalability ratio is going to be very application dependent.

The service time per transaction is going to increase at higher load levels, and the utilization characteristic is non-linear. The presence of any floating point code in your transaction is also going to have an impact on service times in a non-linear manner, as the T1 behaves like a single CPU system rather than a 32 CPU system when it is running floating point calculations.

6. CPU Power Management

AMD PowerNow! for the Opteron series of server CPUs dynamically manages the CPU clock speed based on

Utilization in order to reduce the overall power consumption of the system. The speed takes a few milliseconds to change, and it is not clear exactly what speeds are supported, but one report stated that the normal speed of 2.6GHz would reduce to as low as 1.2GHz under a light load.
This report also shows detailed CPU configuration. http://www.gamepc.com/labs/view_content.asp?id=opteron285&page=3

The problem with this for capacity management is that there is no indication of the average clock rate in the standard system metrics collected by capacity planning tools. PowerNow! is described by AMD at http://www.amd.com/us-en/0,,3715_12353,00.html and drivers for Linux and Windows are available from http://www.amd.com/us-en/Processors/TechnicalResources/0,,30_182_871_9033,00.html. In the future, operating systems may be able to take the current speed into account, and estimate the capability utilization, but the service time is higher at low clock rates, so we will always see some confusing metrics.

The current PowerNow! implementation works on a per-chip basis, and Opteron's have two complete CPU cores per chip that share a common clock rate. In a multiprocessor system made up of several chips, each pair of processing cores could be running at a different speed, and their speed can change several times a second.

Our basic assumption, that mean service time is a constant quantity, is invalidated in a very non-linear manner.

7. Virtualization as an Element of Architecture

The Enterprise Grid Alliance (http://www.gridalliance.org) has published a reference architecture that defines layers of virtualization. A diagram taken from the EGA Reference Architecture is shown below.

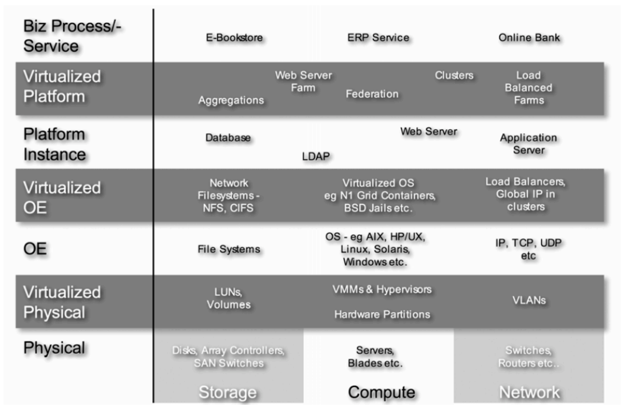


Figure viii - Some Grid Component Classes

For each physical layer in the areas of storage, compute and network, there is a corresponding virtualized layer. Most current metrics and management systems are built to address the physical layers only, and this represents a challenge both for tools vendors and for capacity planning professionals.

8. Virtualized Physical CPU Resources

As shown in the EGA reference architecture diagram, virtual machine monitors (VMMs), Hypervisors and Hardware Partitions provide a raw CPU resource that can host a complete operating system. IBM's VM on mainframes is an early example of a VMM, and Sun's Dynamic System Domains is an early example of hardware partitioning. They have been in use for many years on a small number of high end systems.

The major change in this space is that low cost commodity hardware now has the same features. VMware (http://www.vmware.com) and Xen (http://www.xensource.com) have become popular ways to host many instances of different operating systems on a single machine, and low cost hardware such as Sun's UltraSPARC-T1 and AMD's Opteron (for example see http://www.pantasys.com) have support for flexible hardware partitioning.

For hardware partitioning, the capacity management problem is that the number of CPUs in the machine can change, and most performance management tools do not collect and report this as a time varying metric. It usually reported as a system constant. In some cases a reboot is needed to effect a change, but the trend is to allow a dynamic change to be made to a running system. Such changes tend to happen relatively infrequently, perhaps twice a day at most.

For VMMs, the additional problem for performance tools is that the CPU allocation is non-integral, a fraction of a CPU can be provided and that fraction may be allowed to vary instantaneously. If an operating system running in a VMM is configured to provide all its idle time to other co-habiting operating systems, then it is by definition 100% busy all the time. However its ability to do more work is completely unrelated to the naive utilization metric. If all the hosted operating systems in a VMM get busy at the same time then response times will increase without any indication from the traditional system metrics.

In older tools, the number of CPUs reported by an operating system is not only static, it's an integer. For VMMs the average provisioned CPU must be reported in every time interval along with all the other metrics. Luckly, the wide use of VMware has already generated a good awareness for the issues, and there is quite wide support for VMware specific metrics in performance management tools from vendors such as BMC, Teamquest and Hyperic, however these vendors do not explicitly support Xen as far as I could tell.

9. Virtualized Operating System Resources

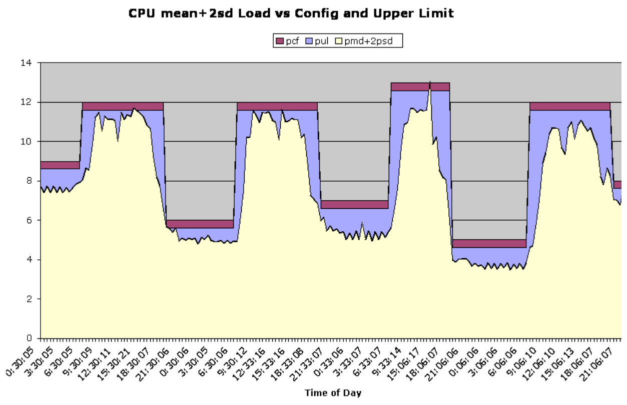
In some cases, the overhead of managing a completely separate operating instance is not needed. If many copies of the exact same version of the operating system are required then a resource container may be used to virtualize operating system resources. The traditional IBM Mainframe Logical Partition (LPAR) is an example. Solaris 10 provides zones (N1 Resource Containers) which can have their own root password, IP address and a defined share of the overall CPU resources. Solaris extended system accounting (http://perfcap.blogspot.com/2005/02/extended-accounting-in-solaris-8-10.html) collects data on a per zone basis, but few tools currently make use of this information. There is no current way to get CPU utilization on a per zone basis other than by using extended system accounting's interval accounting facility for all the processes in each zone.

An application that is subject to limitations on the share of the CPU that it can use, will have a response time characteristic that is dependent upon its share allocation. For this reason, it is important to collect and report the average CPU share allocation in an interval along with all other metrics.

10. Focus on Headroom

So if utilization is no longer useful, what should we replace it with in all those nice graphs we use for capacity planning? I propose that we focus on defining a metric called "headroom".

For simple systems, headroom reduces to "one minus margin minus utilization". So if your simple system is truly 65% busy and you never want it to be more than 80% busy, you are targeting a 20% margin and have 15% headroom. For simple systems where the amount of CPU power varies we can use a plot format that is described in a CMG03 Paper on "Capacity Planning for the Virtualized Datacenter". As shown below, the capacity and margin changes twice a day, to track the expected daytime utilization peak.

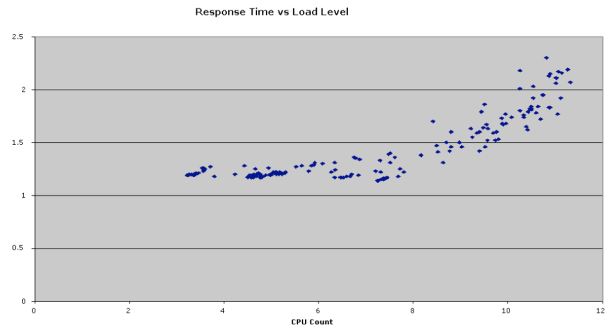


If the CPU clock rate is changing to save power, then the average CPU clock rate can be used to obtain an estimate of current capacity, and headroom can be calculated based upon the capacity at maximum clock rate. However, unlike a simple change in the number of CPUs, there is also an impact on service time, since the service time with reduced clock rate will be higher.

For complex systems, where utilization is not linearly related to capacity, the focus should shift from utilization and headroom based on busy time, to headroom based on throughput and response time.

If we know that a system can sustain a peak throughput of 5000 transactions per second with four Opteron CPUs at full clock rate, then if it is running at 4000 transactions per second it has 20% remaining headroom. We can replace our utilization graphs with graphs that have throughput on the Y-axis, and which show the peak throughput level as a separate line, which varies according to the average capacity of the system.

We may not know the peak throughput for a particular workload, but we can often measure response time for our transactions, either directly at the application level, or by inference from low level CPU statistics (such as microstate accounting in Solaris). Since the load varies over time, we can also make a plot of response time versus throughput, and fit a curve to it that we can use to determine the throughput at which response time will become unacceptable. An example plot, based on microstate measures of CPU usage is shown below.



11. Some Tools That Handle Virtualization

I have used Dave Fisk's Ortera Atlas storage analysis tool (http://www.ortera.com) to model virtualized I/O subsystems accurately. It uses custom kernel instrumentation on Solaris and Linux systems to obtain detailed trace data.
Hyperformix IPS (http://www.hyperformix.com) includes the ability to model Hyper-Threaded CPUs. I tried it out for a relatively simple case and the model made quite accurate predictions of throughput and response time.
BMC® Performance Manager for Virtual Servers (http://www.bmc.com) can manage large scale VMware installations.
Teamquest (http://www.teamquest.com) have support for VMware.
Hyperic have support for VMware. (http://www.hyperic.com/products/managed/vmware-management.htm)

12. Summary

Utilization is useless as a metric and should be abandoned. It has been useless in virtualized disk subsystems for some time, and is now useless for CPU measurement as well. There used to be a clear relationship between response time and utilization, but systems are now so complex that those relationships no longer hold. Instead, you need to directly measure raw available capacity and response time and relate it to throughput. Utilization is properly defined as busy time as a proportion of elapsed time. The replacement for utilization is headroom which is defined as the unused proportion of the maximum possible throughput.