Maximizing hypervisor scalability using minimal virtual machines

Alfred Bratterud, Hårek Haugerud
Dept. of Computer Science
Oslo and Akershus University College of Applied Science
Oslo, Norway
alfred.bratterud@hioa.no

***Abstract*—The smallest instance offered by Amazon EC2 comes with 615MB memory and a 7.9GB disk image. While small by today's standards, embedded web servers[1] with memory footprints well under 100kB, indicate that there is much to be saved. In this work we investigate how large VM-populations the openStack hypervisor can be made to sustain, by tuning it for scalability and minimizing virtual machine images. Request-driven Qemu images of 512 byte are written in assembly, and more than 110 000 such instances are successfully booted on a 48 core host, before memory is exhausted. Other factors are shown to dramatically improve scalability, to the point where 10 000 virtual machines consume no more than 2.06% of the hypervisor CPU.**

***Index Terms*—hypervisor scalability, operating system bloat ,virtual machines, kvm**

I. Introduction

Our aim is to demonstrate the potential gain in resources and number of virtual machines per server, achievable on current cloud hypervisors by reducing CPU and memory footprints of virtual machines. The motivation for reducing resource consumption is obvious as it is positively correlated with power consumption, monetary cost and carbon dioxide emissions. The motivation for increasing the number of virtual machines per hypervisor is derived from the predominant business model of cloud vendors such as Amazon EC2 and Microsoft Azure, where the customer is charged a fixed price per running instance. Running more instances on the same hypervisor is then directly correlated to higher earnings per server.

We aim to provide answers to the following research questions:

1) What is the maximum potential gain, in terms of system resources, and virtual machine count, from reducing the resource usage of cloud guest operating systems
2) How can virtual machines and state of the art hypervisors be tuned in order to sustain the largest possible population of virtual machines.
3) How can minimal virtual machines be used as a reference component in a standardized scalability test which can easily be repeated on multiple hypervisors and with multiple guest architectures.

According to [1] the power consumption of cloud computing is on the level with India. Most of these resources are distributed by means of virtual machines running general purpose operating systems, which provide great flexibility but which also suffer from feature bloat. As an example, a standard Wordpress appliance was deployed on a minimal Amazon EC2 instance running Ubuntu[2]. 90% of the appliance consisted of the operating system while only 1.3% contained the wordpress source code. Taking into account all the supporting software and libraries required for wordpress to run it still amounted to only 10% of the total disk space, the remaining 1 GB belonging to Ubuntu. In contrast, Windows 3.11 required 10-15 MB of storage. While Windows 3.x is clearly obsolete several modern OS kernels are even smaller. Notably the L4 microkernel is only 10 000 lines of code, and is the only operating system kernel which is *proven correct* [2], making it the most

[1]One example includes yassl, of www.wolfssl.com

[2]64-bit Amazon Linux AMI 2013.03.1, T1 Micro, created June 2013.

predictable kernel to date. Also considering that modern embedded webservers require less than 100kB of memory it seems reasonable that usable VM's can be made orders of magnitude smaller than those offered by current public clouds.

II. Related work

While much has been done on the subject of scalability of services in cloud [3]–[5] , most of this work focuses on scaling services across hypervisors, or across virtual machines. Little work has been done on massive scalability testing of individual hypervisors. In 2006 IBM authors Theurer et.al [6] tested the scalability of Xen, up to 16 cores. Their emphasis was on scalability in terms of number of CPU cores, and they maintained a 1-1 relation between cores and guests, testing scalability up to a total of 16 virtual machines. Interestingly they do provide examples of situations where computational efficiency is improved by spreading the workload over several virtual machines, as opposed to performing the same calculation directly on the underlying system. In 2008 a comparative study was done between Xen, KVM and vmware [7]. Also here, an upper limit of 16 VM's was used. More recent studies [8], [9] have not been found to investigate any larger populations.

Also a lot has been done recently on VM consolidation. One of the larger such experiments from last year was done by [10] where a total of 6048 VM's are used. These instances are however only simulation machines, and distributed over 42 hypervisors, and 1008 CPU cores, where no core ever runs more than 6 emulated instances. Also VM packing is currently much studied, and while much of this work is theoretical and simulation based, several practical experiments are also conducted. Recent examples include [11] where a total of 56 virtual machines were deployed over 4 physical servers and a total of 64 cores, [12] where real trace data from a maximum of 648 virtual machines distributed according to various packing algorithms over 648 physical nodes were analyzed. As far as we have found, our results of 110 000 VM's per host with a modified hypervisor using both memory and swap, and 10 000 VM's per host on an unmodified hypervisor using system memory only, are unprecedented.

III. Minimal VM's from scratch

The smallest possible disk image bootable by Qemu is 512 bytes, as this is the size of a standard PC boot sector. In this work this space is not used for a boot loader, but rather to store all the machine code necessary to make what we call *Micro Virtual Machines* (MVMs), mimicking the behavior of a standard web server, using emulated CPU workloads and serial port I/O.

A. Emulating CPU profiles

Having access to VM's with various CPU profiles is in itself useful for conducting research on VM packing, in order to maximize its effects. Amazon, presumably for VM packing purposes, has recently placed larger emphasis on CPU behavior[3] especially for micro instances, which have a variable CPU SLA. As a first, preliminary experiment, different kinds of CPU profiles were successfully mimicked by a few lines of assembly, written directly to the boot sector. The result is 512-byte disk-images which can be booted directly with Qemu/kvm from the command line.

The disk images are also successfully tested on openStack. Uploading an image through the horizon interface, will enable the creation of a bootable openStack VM, with console access. Future work includes running minimal VM experiments on openStack compute nodes.

B. True sleep for MicroMachines

To determine the upper bounds of hypervisor scalability, it is necessary to reduce the intrinsic activity level of VM's to a minimum. The only way to make an x86 CPU do nothing, without turning off its power, is to have it wait for an interrupt. The "hlt"-instruction effectively puts the CPU to sleep until the next interrupt is fired, and is the means by which operating system kernels such as linux implement idling. When no I/O devices are enabled, the next interrupt will come from the Programmable Interval Timer (PIT). For a physical computer this means that the CPU is completely idle, but for Qemu-instances this is not the case, as the Qemu process also has to emulate the inner workings of the PIT-chip. For this reason, a virtual machine in a "hlt"-state will

[3]An extensive introduction is given in the Amazon micro instance documentation: http://bit.ly/aKp2Io

always consume some amount of CPU. With few VM's this will cause no than an occasional flicker between 0% and 1% CPU usage on the host. When scaling up in numbers however, it will cause host CPU to become the primary limiting factor.

C. Disabling the PIT

The x86 PIT can be set to either interval mode (default), or one-shot mode. In interval mode it will interrupt the CPU based on a preset frequency of between ca. 18.2 Hz and 1.1 Mhz. The default frequency on boot is already the lowest, so the only way to decrease the load it incurs on the host is to set it to one-shot mode, in which case a single interrupt is fired after a given 16 bit register countdown. Once fired, the PIT remains inactive until another one-shot is ordered, or the mode of operation is changed.

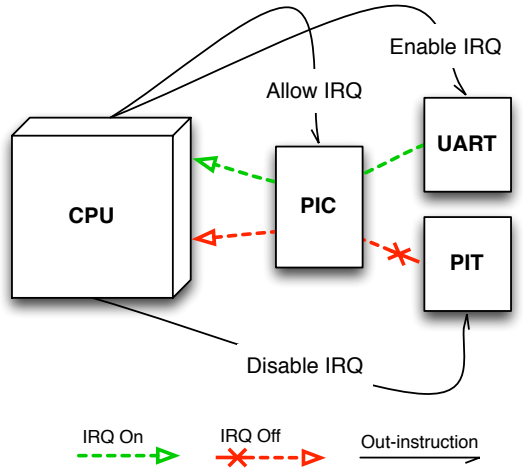

Fig. 1. *Showing the relationships between the CPU, the Programmable Interrupt Controller (PIC), the Serial controller (UART) and the Programmable interval timer (PIT). The serial interrupt is enabled on the UART, the bitmask in the PIC changed to allow the interrupt to pass through to the CPU, while the interrupt from the PIT is being disabled.*

Once the PIT was disabled the serial port interrupt was enabled, making data input on the serial port the only way to wake the VM. A serial interrupt handler was written, with the ability to serve different kinds of text content based on the request string. Enabling the serial interrupts involve changing the bit-mask in the Programmable Interrupt Timer (PIC) to allow the interrupt to pass through to the CPU, and then enabling firing of the interrupt on data input, in the UART. Once this is done, the system will be in a completely I/O-dependent event-based mode, which is good for emulating systems such as web servers, but which incurs but an absolute minimum of load on the underlying host. We found that the effect of disabling the PIT had a surprisingly big impact on virtual machine scaling, as discussed in the next chapters.

IV. Hypervisor optimization

In order to maximize the number of simultaneous virtual machines running on a single server, several versions of the KVM kernel modules and the QEMU emulator was tested as well as various combinations of compile time and run time options.

A. Optimizing the VM framework

The current LTS version of Ubuntu, 12.04, includes Linux kernel version 3.2.0 and QEMU emulator version 1.0, qemu-kvm-1.0. When running the MVMs these and several other important parameters were varied in order to obtain an optimal framework for running MVMs:

- Linux kernel version (3.2.0 and 3.8.0)
- QEMU version (1.0 and 1.4.0)
- MVM architecture (i386 and X86_64)
- KVM modules (enabled and disabled)
- Compiling QEMU (options disabling extra and unnecessary hardware features)
- Running QEMU (options reducing emulator features at runtime)
- Server power saving (enabled and disabled)

B. Initial experiments

All the experiments in this study was performed on a Dell PowerEdge R815 server with four 12-core AMD Opteron 6234 CPUs and 128 GBytes of RAM. In the experiments the MVMs were started gradually, making it possible to detect how the resources were consumed with an increasing number of MVMs. System data like memory usage and CPU usage was collected from the proc filesystem. The memory consumed was calculated from the difference in the free memory of the system with and without virtual machines running.

Some initial experiments were performed using an unmodified Ubuntu 12.04 LTS installation.

Slowly booting MVMs, all the 48 CPUs were exhausted running in kernel mode and the server eventually halted before the MVM count reached 2000. Disabling the kernel KVM modules, the system was able to boot 20 000 virtual machines. By updating the Linux kernel from the version 3.2.0 to 3.8.0 the system was able to boot 20 000 MVMs also with the KVM kernel modules present.

C. Exploring the limits of the server

In order to optimise the server to support as many virtual machines as possible, the experiments were run using the latest 3.8.0 kernel and the newest version of QEMU, 1.4.0. Both compile and runtime options were systematically tweaked to reduce the run time size of the MVMs.

Each line in Table I corresponds to an experiment performed by starting new MVMs every one-tenth second, continuing until the system is exhausted and halts or becomes unresponsive due to lack of memory or CPU capacity or both.

The first three columns show the parameters which were varied and the next column shows the number of MVMs the server was able to boot before halting or becoming unresponsive. The last columns show the resource consumption of the experiments.

The difference between the two architectures is not substantial, but the i386 MVMs are in most cases slightly smaller and in true sleep mode and with the KVM kernel disabled, it enables the system to run a total of 110.239 virtual machines.

The importance of turning off the PIT in the virtual machines when idle is apparent. It makes the system able to sustain more than twice as many virtual machines as seen from the last four lines of Table I. The CPU usage of the server is drastically reduced and lack of memory is the main feature limiting the number of MVMs.

When using the KVM kernel modules, the memory footprint of each virtual machine is smaller, less than three KBytes. But when all RAM is consumed, the server is not able to utilise the swap memory and start more MVMs. Still, when using the KVM modules, the system is able to run more than 50 000 virtual machines even when their internal clocks are consuming CPU cycles.

In all the experiments of Table I, the CPUs were running at their highest frequency, 2.4 GHz. The corresponding results when running the experiments on the same server with the default power saving scheme is shown in Table II. When the sleep feature is off, the MVMs consume a lot of CPU and in this case the system is not able to sustain as many MVMs as when the CPUs are persistently running at maximum frequency. However, when the internal MVM clocks are turned off the results are quite similar, since the CPU usage in any case is low and the CPU power scheme thus is less relevant.

V. A STANDARDIZED SCALABILITY TEST

In order to predictably test the impact of vm count on system[4] resources, a standardized test was developed. The following features were found to give good results:

- y axis shows normalized values for resource consumption, 0-100%.
- x axis represents VM count, in intervals
- Low interval count (around 20), with error bars, to reduce noise in the diagram. The size of the interval will be determined by the expected maximum dividedvided by 20.
- y should be expected to increase with x.
- x is naturally limited by the first resource to reach a given threshold, here set to 90%.
- Time to boot is disregarded, although this was severely affected by several variables. The underlying assumption is that booting is a fairly rare activity and thus not an essential factor in determining scalability.

A. The effect of Power saving features

One factor immediately uncovered by running the scalability test was the effect of the power saving features on the AMD CPU discussed above. Fig.2 shows the effect as a steep incline, following a bump, after which the incline is gentler. This results from a changing of scale; as the CPU frequency increases so does the number of CPU cycles per jiffy, making the same work doable with fewer jiffies.

B. The effect of the PIT

The scalability test is also used to show the dramatic effect of disabling the PIT inside the

[4]In the following, vanilla Ubuntu 12.04 LTS with KVM

Sleep	KVM	Arch	MVMs	%CPU	%RAM	%swap	size/kB	Minutes	Boot/sec
0	0	64	15070	100.0	42.2	0	3692	40	0.16
0	0	32	15054	99.9	41.1	0	3603	34	0.14
0	1	64	51912	66.0	98.6	5.7	2805	121	0.14
0	1	32	50224	65.3	98.4	3.6	2782	115	0.14
1	1	64	63832	3.8	98.6	19.2	2847	135	0.13
1	1	32	87202	3.8	98.3	42.3	2790	200	0.14
1	0	64	106771	7.1	99.0	100.0	3737	414	0.23
1	0	32	110239	8.9	99.0	100.0	3620	305	0.17

TABLE I

Exploring the limits of a servers ability to support virtual machines by gradually booting MVMs until the server becomes unresponsive. Each line corresponds to a complete experiment.

Sleep	KVM	Arch	MVMs	%CPU	%RAM	%swap	size/kB	Minutes	Boot/sec
0	0	64	9268	82.8	26.5	0.0	3775	18	0.12
0	0	32	9500	85.0	99.2	100.0	42042	19	0.12
0	1	64	27747	55.3	59.8	0.0	2844	60	0.13
0	1	32	31259	60.2	66.2	0.0	2795	69	0.13
1	1	64	87157	2.1	98.8	42.4	2801	277	0.19
1	1	32	87206	2.1	98.7	41.5	2771	281	0.19
1	0	64	97226	11.1	98.7	88.4	3780	316	0.20
1	0	32	111179	5.8	99.0	100.0	3589	529	0.29

TABLE II

Repeating the same experiments with the default power saving feature enabled. Compared to the experiments of Table I where all CPUs were running at maximum frequency, the results of the lower half of the table is quite similar.

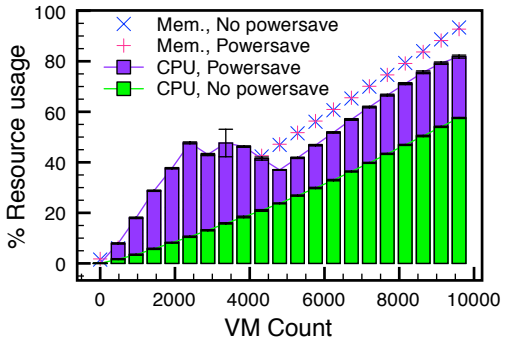

Fig. 2. *Scalability test showing the effect of the CPU power saving features. Disabling them yields CPU scaling close to linear, as expected. The VM-count reaches 9600 VM's before memory consumption reaches 90% and the experiment stops. Maximum CPU utilization was 81.84% with powersaving enabled, and 57.55% without.*

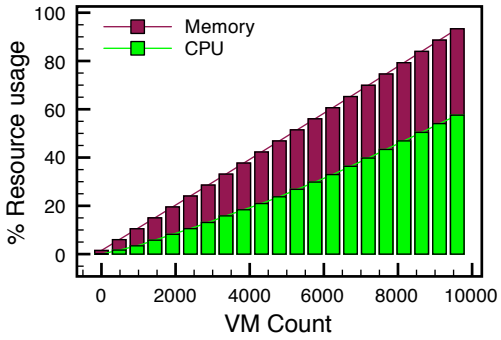

Fig. 3. *Scalability test showing the impact of idling MVM's with the PIT enabled (default), on the system resources. The VM-count reaches 9600 MVM's before memory consumption reaches 90% and the experiment stops. The maximum reached CPU utilization was 57.55%*

guests. Fig.3 shows a CPU usage of 57.55% with the PIT turned on, as opposed to 2.06% with the PIT turned off, shown inn Fig.4. Although this CPU usage is accumulated by a large number of machines, removing it will enable a significantly higher level of scalability; all this CPU capacity is now available to service requests inside the guests.

VI. Conclusion and Future Work

Disabling the interval timer (PIT) inside virtual machines showed a 55% reduction in overall CPU consumption on the hypervisor, when the system

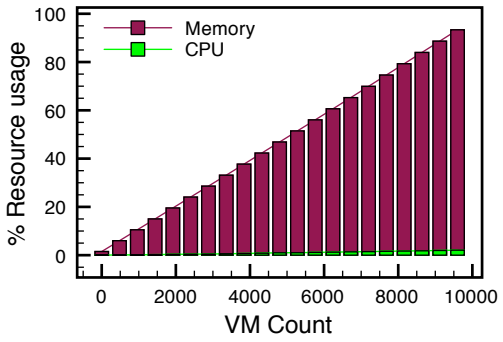

Fig. 4. *Scalability test showing the impact of idling MVM's with the PIT disabled, on system resources, in contrast to Fig.3. The VM's will now only wake up only on serial data. The VM-count reaches 9600 VM's before memory consumption reaches 90% and the experiment stops. The maximum reached CPU utilization was 2.06%*

memory was saturated (at 93%) with MVM's. In this case, 98% of the hypervisor CPU capacity is available to service the virtual machines.

Our results show that reduction in operating system resource usage has great potential for reducing memory and CPU consumption on cloud hypervisors. This can dramatically increase the amount of virtual machines a cloud provider can host on a single hypervisor, and further increase the potential effects of server consolidation using VM packing algorithms.

Future work includes

- Running scalability experiments with minimal VM's on openStack
- Developing prototypes of minimal vm's, custom tailored for the services they provide.
- Determining the effect of upscaling on QoS
- Including more resources in the scalability test, such as network traffic and disk IO

Lastly, we intend to test the minimal VM's ability to serve as simulations of full-sized clouds. Our current results in the 10k - 100k range, makes this seem like a promising route to a closer-to-life simulation environment.

The microMachine source code is published under GPL and available on github [13].

REFERENCES

[1] (2012, Aug.) How clean is your cloud. Greenpeace Climate Reports. [Online]. Available: http://www.greenpeace.org/international/en/publications/Campaign-reports/Climate-Reports/How-Clean-is-Your-Cloud/

[2] (2013, Jun.) A formally correct operating system kernel. The L4.verified project website. [Online]. Available: http://www.ertos.nicta.com.au/research/l4.verified/

[3] J. Yang, J. Qiu, and Y. Li, "A profile-based approach to just-in-time scalability for cloud applications," in *Proceedings of the 2009 IEEE International Conference on Cloud Computing*, ser. CLOUD '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 9–16. [Online]. Available: http://dx.doi.org/10.1109/CLOUD.2009.87

[4] J. Wu, Q. Liang, and E. Bertino, "Improving scalability of software cloud for composite web services," in *Cloud Computing, 2009. CLOUD '09. IEEE International Conference on*, 2009, pp. 143–146.

[5] J. Gao, P. Pattabhiraman, X. Bai, and W. Tsai, "Saas performance and scalability evaluation in clouds," in *Service Oriented System Engineering (SOSE), 2011 IEEE 6th International Symposium on*, 2011, pp. 61–71.

[6] A. Theurer, K. Rister, O. Krieger, R. Harper, and S. Dobbelstein, "Virtual scalability: Charting the performance of linux in a virtual world," in *2006 Linux Symposium*, vol. 2, pp. 393–402.

[7] X. Xu, F. Zhou, J. Wan, and Y. Jiang, "Quantifying performance properties of virtual machine," in *Information Science and Engineering, 2008. ISISE '08. International Symposium on*, vol. 1, 2008, pp. 24–28.

[8] D. Leite, M. Peixoto, M. Santana, and R. Santana, "Performance evaluation of virtual machine monitors for cloud computing," in *Computer Systems (WSCAD-SSC), 2012 13th Symposium on*, 2012, pp. 65–71.

[9] D. Armstrong and K. Djemame, "Performance issues in clouds: An evaluation of virtual image propagation and i/o paravirtualization," *The Computer Journal*, vol. 54, no. 6, pp. 836–849, 2011. [Online]. Available: http://comjnl.oxfordjournals.org/content/54/6/836.abstract

[10] E. Feller, C. Morin, and A. Esnault, "A case for fully decentralized dynamic vm consolidation in clouds," in *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, 2012, pp. 26–33.

[11] J. Chen, K. Chiew, D. Ye, L. Zhu, and W. Chen, "Aaga: Affinity-aware grouping for allocation of virtual machines," in *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*, 2013, pp. 235–242.

[12] S. Takahashi, A. Takefusa, M. Shigeno, H. Nakada, T. Kudoh, and A. Yoshise, "Virtual machine packing algorithms for lower power consumption," in *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, 2012, pp. 161–168.

[13] (2013, Jun.) Virtual Micro Machines toolkit. github source repository. [Online]. Available: https://github.com/alfred-bratterud/MicroMachines