A Measurement Study of Server Utilization in Public Clouds

Huan Liu
Accenture Technology Labs
San Jose, California
huan.liu@accenture.com

***Abstract*—Due to a server's non-proportional energy consumption, it is highly desirable to increase server utilization in order to lower energy consumption and minimize environmental impact. To increase the utilization level, we must first understand the current utilization level in the state-of-the-art cloud environment. Towards that goal, we devise a new technique which can be used to measure the CPU utilization in a cloud. We sampled two different infrastructure cloud providers (IaaS or Infrastructure as a Service) on their server utilization. Surprisingly, our results show that there is a significant room for improvement.**

I. Introduction

One potential benefit of using a public cloud, such as Amazon EC2 or Microsoft Azure, is that a cloud could be more efficient. This is because a cloud supports many users, and it can potentially achieve a much high server utilization through aggregating large demands.

In order to improve utilizations in a cloud further, we must first understand the level of utilization achievable in a state-of-the-art cloud environment. In this paper, we report an extensive study of server CPU utilization across two different public cloud providers. To measure a cloud physical server's utilization, we launch a small probing Virtual Machine (VM) (often the smallest VM offered) in a cloud provider, and then from within the VM, we monitor the CPU utilization of the underlying hardware machine. Since a cloud is built around multi-tenancy, there are several other VMs running on the same physical hardware. By measuring the underlying hardware's CPU utilization, we measure the collective CPU utilization of other VMs sitting on the same hardware. We use a small VM as the probing VM to make sure that we take the least amount of capacity away from the target physical machine, such that it can host as many other VMs as possible. Our study shows that there is a lot of room for public clouds to become more efficient.

The contributions of this paper are:

1) Propose a new technique to measure a cloud physical server's CPU utilization.
2) Conduct a CPU utilization study across two different public cloud providers.

In Sec. II, we first briefly describe the two cloud providers we investigated. Then, in Sec. III, we describe the measurement technique in details and also the measurement results we have collected. We then conclude in Sec. IV.

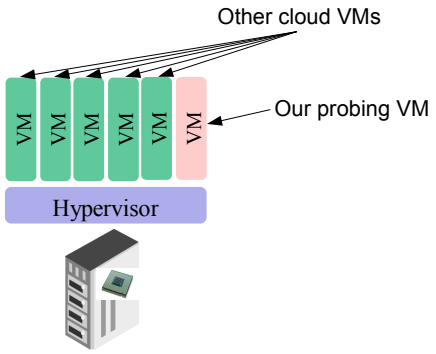

Figure 1. Using a probing Virtual Machine (VM) to measure the underlying hardware's (hence, other VMs') CPU utilization.

II. The two public cloud providers

We have measured two cloud providers. In this section, we give a brief description of them, including their offerings, the underlying hardware, and the technology used.

A. Amazon EC2

Amazon EC2 offers five classes of instances (Amazon's terminology for VM): standard and micro instances, high-memory instances, high-CPU instances, cluster compute instances and cluster GPU instances. Each class of instances runs in a separate cluster on its own hardware. We only examine two classes of instances; hence we will only briefly describe these two classes.

We can determine the underlying hardware through a combination of techniques [1]. The type of physical server running the standard instances is the same as that running the micro instances. Each physical server has a single socket Intel Xeon E5430 4-core 2.66GHz processor. Although we have seen a few physical servers based on AMD Opteron CPUs, they are rare and we believe they are being phased out. The high-CPU instances run on a different set of hardware. The hardware server consists of dual socket Intel Xeon E5410 4-core 2.33GHz processors; thus, there are 8 physical cores in total.

The smallest standard instance is the *m1.small* instance. It has one virtual core and it has 1 EC2 Compute Unit (ECU), which is equivalent to roughly 40% of a single physical core's capacity. Thus, on the physical hardware, it is possible to host up to 10 *m1.small* instances. We use

m1.small instances as our probing VMs. Even though EC2's new Micro instance is smaller, its CPU allocation fluctuates greatly, which is not suitable for our probing purpose. The largest standard instance is the *m1.xlarge* instance, which has 4 virtual cores with 2 ECU each. The total CPU allocation is limited to roughly 75% of the physical CPU's capacity across four cores.

The smallest high-CPU instance is the *c1.medium* instance, which has 2 virtual cores each with 2.5 ECU. We use *c1.medium* as the probing VM. It takes up 1/4 of the underlying physical CPU's capacity. The largest high-CPU instance is the *c1.xlarge* instance, which has 8 virtual cores each with 2.5 ECU. A *c1.xlarge* instance takes up the whole physical CPU capacity.

Amazon EC2 caps the maximum CPU to the ECU specified. An instance cannot use more than its allocated ECU even if the other instances on the same hardware are idle. We believe that Amazon does not oversubscribe the hardware, but we could not determine that experimentally. Since our probing VM takes away capacity that could be allocated to other VMs, one may argue that we need to scale up the measured CPU utilization. For *m1.small*, it should be scaled up by a factor of 10/9, and for *c1.medium*, it should be scaled up by a factor of 4/3. For simplicity, we report the raw measured result without scaling up.

B. GoGrid

GoGrid offers several different VMs with varying sizes, but all of them run on the same type of hardware. The physical server consists of dual socket Intel quad-core processors. The smallest VM has 512MB of memory and 1 virtual core. It is guaranteed 0.5 of a physical core's CPU capacity; however, it can burst up to 1 full physical core if other VMs are idle. The largest VM has 16GB of memory and 8 virtual cores, and it can take up the full physical CPU. GoGrid employs the Xen hypervisor, and a virtual core could be scheduled to run on any given physical core.

III. Measuring by CPU temperature

Our measurement technique is based on the observation that when a CPU chip is used, the chip gets warmer. Conversely, when the CPU is idle (i.e., in the halt state), little heat is generated and, as a result, the chip is cooler. From our probing VM, we periodically read the CPU temperature sensors; then, based on a thermal model presented in Sec. III-A, we can deduce the average utilization.

Modern CPUs from Intel all have an embedded thermal diode at the hottest spot of each core. The temperature diode could be read through an MSR. The temperature reported by an Intel CPU is a relative temperature expressed as the difference between the current chip temperature and the critical shutdown temperature. Thus the temperature reading goes down as the chip gets warmer. At a reading of 0, the chip would automatically shutdown to protect from a meltdown. The temperature diode on each core has a reading resolution of one Celsius degree.

To read the thermal diode, we design a kernel driver which not only reads the temperature, but also reads the APIC ID by running the `cpuid` instruction. The APIC ID uniquely identifies a particular core; thus, we can attribute the temperature reading to the correct core. From the user space, we periodically call the kernel driver to sample the temperature reading. In the two cloud providers where we are able to read the temperature sensor (Amazon EC2 and GoGrid), both use the Xen hypervisor. Under the Xen hypervisor, a virtual core is not fixed to a physical core, and it could be scheduled to run on any physical core. By repeatedly sampling, we are able to read the temperature diode on all physical cores, and we take an average across all cores and treat it as the die temperature.

In our sampling study, we want to get a sample every second. Since the small VM we use in a cloud has a smaller number of virtual cores (one or two) than the number of physical cores, we sample at a higher frequency – every 100ms – in the hope that we will be able to read from each physical core at least once in a second. If we are able to get at least one sample from a core during the second, we take the latest sample. If not, we use a stale sample from the most recent past as an approximation. Since the user program and the kernel driver are all light weight, and since we sample at a low frequency (100ms), the load we introduce to the host is negligible.

A. Thermal model

Once we have a sample of the chip temperature, we need to translate it into the actual CPU utilization. In order to translate, we have to build a correlation between the two. We borrow the CPU thermal model developed in [2]. The model was developed to translate from the CPU utilization to the CPU temperature. However, as we will show, it is easy to extend the model so that we can reverse the direction and translate from the temperature to the CPU utilization instead. Although more accurate models–such as Hotspot [3] [4] [5]–exist, those models are difficult to apply when translating from temperature to utilization.

We assume the energy generated from a CPU is proportional to the utilization level, and that it causes a linear increase in the temperature. This can be expressed as:

$$dT = c_1 U dt \tag{1}$$

where c_1 is a constant.

The linear relationship is a reasonable assumption. However, there is often a constant term to the equation. When a CPU is idle, it typically consumes a non-negligible amount of power. But, to simplify the presentation, we assume the constant is zero, and we note that a non-zero constant would not affect the derivation.

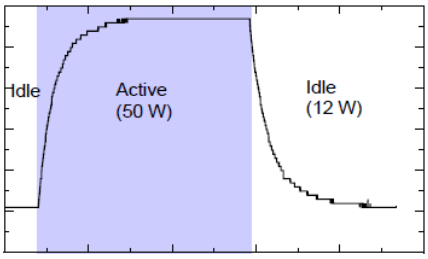

Figure 2. CPU thermal dynamics as predicted by equation (4).

The differential between the CPU temperature and the ambient temperature causes the chip to cool down. Again, we assume the decrease in temperature is linear to the temperature differential.

$$dT = -c_2(T - T_0)dt \tag{2}$$

where c_2 is a constant.

Combining the two equations, the net temperature fluctuation is

$$dT = [c_1 U - c_2(T - T_0)]dt \tag{3}$$

Solving the differential equation, we can get the chip temperature as a function of time as follows:

$$T(t) = -\frac{c_0}{c_2}e^{-c_2 t} + \frac{c_1}{c_2}U + T_0 \tag{4}$$

The equation above has a dynamic and a static component. The static component predicts the steady-state temperature if the CPU is at a certain utilization U for a long period of time, i.e.,

$$T(\infty) = \frac{c_1}{c_2}U + T_0 \tag{5}$$

The dynamic component ($-\frac{c_0}{c_2}e^{-c_2 t}$) predicts that the temperature increases or decreases (when going from a high utilization to a low utilization) gradually until it reaches the steady-state temperature. The increase or decrease is an exponential function of time.

Fig. 2 shows what equation (4) predicts when we first add a constant load to the CPU, and then subsequently remove it after the CPU has reached the steady state.

If equation (4) accurately describes the CPU's time response, we can reverse it to translate from a temperature measurement back to the instantaneous CPU utilization. Since the dynamic response is an exponential function of time, we can easily estimate the parameter c_2. We just need to measure a CPU's transient behavior, put it on a log scale, then perform a least-square linear fit. The slope of the curve is the estimated c_2.

Unfortunately, there are a couple of challenges in using equation (4). First, we need to have a much finer granularity of sampling. Since the thermal time constant is in the milliseconds range [6], we need to sample at the same time scale to capture the dynamics. But, sampling at a higher rate could present too high a load to the host which will affect the measurement.

Second, we observe that a real CPU behaves slightly differently from what equation (4) predicts. Fig. 3 shows the measured average temperature reading when a constant load is first added to a CPU at time 0, then removed after two minutes. The CPU used is an Intel Core2 6300 1.86GHz dual-core CPU in an HP XW4400 workstation. We have also tested on several other CPUs in cloud servers, and we notice that the thermal response is almost identical.

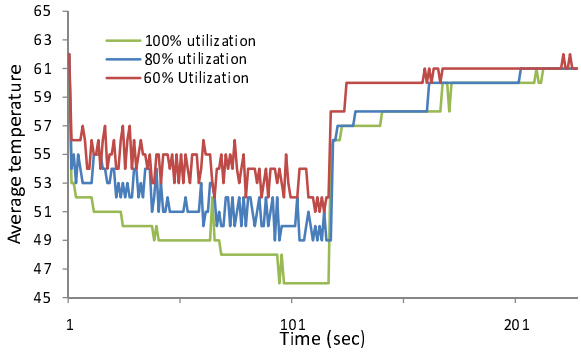

Figure 3. CPU's dynamic thermal response to utilization change.

There are a couple of differences from the theoretical prediction. First, the dynamic component does not follow an exponential curve. The temperature jumps to roughly 60% of the steady-state temperature almost immediately after the load is applied. Then it takes a long time for the temperature to slowly ramp up to the steady-state temperature. This is similarly true when the load is removed at the end of two minutes. The temperature drops quickly, and then it takes a long time for the temperature to reach its steady-state. Second, equation (4) predicts that the dynamic component is independent of the utilization U. Unfortunately, we observe that the time response differs when different loads are put on the CPU. For example, during the slow ramp up phase, the temperature increases more if a higher load is applied.

Even though we cannot use equation (4) to predict the instantaneous CPU utilization from the CPU temperature, we can still measure the longer term average utilization. Due to thermal conservation, the heat generated must be the same as the heat dissipated. The conservation may not hold over a shorter period since the heat sink can trap heat; however, over a longer period of time, or for a period where the heat sink's temperature remains the same from the beginning to the end, thermal conservation holds. Combining equation (1) and equation (2), we have

$$\int c_1 U dt = \int c_2(T - T_0)dt \tag{6}$$

Dividing both sides by the time interval and the constant c_1, we have

$$\bar{U} = \frac{1}{c_1/c_2}(\bar{T} - T_0) \quad (7)$$

The equation states that the average utilization $\bar{U}$ is a linear function of the average temperature $\bar{T}$. The two constants c_1/c_2 and T_0 can be readily estimated from equation (5). Even though the dynamic component of equation (4) is not accurate compared to what we observe experimentally, the static component does match well. Specifically, equation (5) predicts that the steady-state temperature is linear to the load (i.e., the utilization U).

Fig. 4 shows the steady-state temperature reported by the CPU as a function of the CPU load we put on it. We use the *lookbusy* workload generator [7] to generate the desired CPU utilization. The data is collected on an *m1.xlarge* Amazon EC2 instance. Amazon limits an *m1.xlarge* instance to consume at most around 75% of the CPU. To measure the steady-state, we put a constant load on the VM, then wait few minutes before we read the temperature sensor.

As shown in the figure, the steady-state temperature increases almost linearly as predicted by equation (5). Using least-square line fitting, we can easily estimate the constant $\frac{c_1}{c_2}$ (from the slope) and the constant T_0 (from the intersection). Combining with the temperature reading, we can then use equation (7) to estimate the average utilization $\bar{U}$.

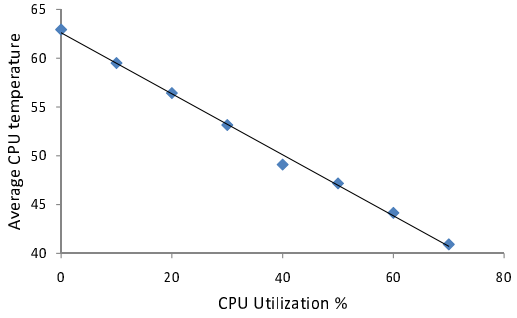

Figure 4. CPU's steady-state temperature as a function of the CPU utilization.

The constants c_1/c_2 and T_0 are a function of a number of independent variables, including the heat sink (material and placement), the fans (placement and speed), the ambient temperature etc. Because these variables diff from one hardware box to another and from one rack location to another, the constants vary from hardware to hardware too. Thus, it is not possible to determine them ahead of time. Instead, after we launch a VM and after we finish monitoring the underlying hardware's CPU temperature, we start a characterization phase. In this phase, we drive a number of characterization workloads each with a different CPU utilization, wait for the temperature to stabilize, then read the CPU temperature sensors. After we have collected a number of utilization and steady-state temperature pairs, we use least-square line fitting to estimate the two constants – c_1/c_2 and T_0, then we use them to infer the CPU utilization.

One potential problem of our approach is that, during the characterization phase, the underlying CPU may not be idle, since other VMs may be actively using the CPU. Fortunately, it is easy to tell when other VMs are active. Except when idle or when fully loaded, a CPU has to alternate between fully loaded and fully idle at any instant of time. The average of the busy and idle cycles determines the average utilization. Correspondingly, the CPU temperature fluctuates up and down, in tune with the CPU's busy and idle cycles. From the CPU's temperature fluctuation, we can easily tell whether the CPU is idle or not. For example, in Fig. 3, the curves for the 80% and 60% load fluctuate much more than the curve for the 100% utilization.

In addition, we also check the lowest temperature recorded during the temperature sampling, and we use that as a validation to verify when we are in an idle period. We can check the CPU is idle in between driving two characterization workloads, but it is also possible that other VMs are active while we are driving a characterization load. In that case, the interference from other VMs will cause the data point to be an outlier. When we notice outliers or non-idle period in between driving two characterization workloads, we restart the characterization phase. In Amazon EC2, which is quite busy, we have to wait for a few days in a couple of occasions before we can finish the characterization phase.

B. Technique validation

We validate the accuracy of our inference technique and the thermal model in a cloud environment in our lab. The cloud environment consists of two HP XW4400 workstations, each contains an Intel Core2 6300 1.86GHz dual-core CPU. They run the Xen hypervisor and they are managed by Eucalyptus [8]. We purposely launch two VMs on the same physical hardware. One VM generates the workload, while the other VM quietly monitors the thermal sensor. We have conducted an extensive study to evaluate the accuracy of equation (7). We generate a number of varying workloads of different characteristics and we compare the actual utilization with the utilization as predicted by equation (7).

The utilization as predicted by equation (7) is very accurate when the average utilization changes infrequently, for example, when the CPU is running a batch workload. This is because a steady workload would allow the CPU temperature to reach the steady state, and the temperature reading would accurately reflect the CPU utilization. In addition, the under-estimation during ramp-up and the over-estimation during ramp-down perfectly compensate for each other. Fig. 5 shows the actual utilization and the inferred utilization for a workload where we first run the *lookbusy*

load generator for two minutes and then idle for two minutes. While we consistently underestimate during the first two minutes when the chip is warming up, we also consistently overestimate during the last two minutes when the chip is cooling off. They compensate for each other such that the estimated average utilization is 39.1%, almost identical to the actual average utilization of 40.5%.

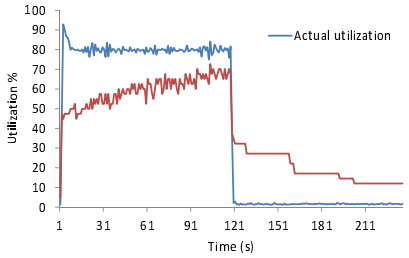

Figure 5. Evaluating the accuracy of equation (7). An example of a more steady workload. The lookbusy workload for 2 minutes, then idle for 2 minutes.

The estimation from equation (7) may be somewhat off when the CPU load fluctuates rapidly. This is mainly due to the fact that the temperature does not have enough time to reach a steady state before the load changes again. Among the workload that we have evaluated, the largest estimation error we have found is 15.1% off from the actual utilization.

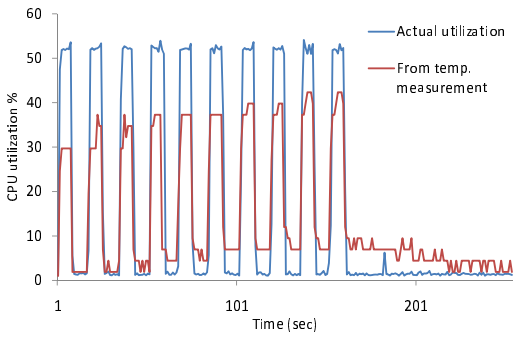

Figure 6. Evaluating the accuracy of equation (7). An example of a more dynamic application. 10 times gzip/gunzip a video file with a 10 seconds pause in between.

Fig. 6 shows the workload that has the largest deviation. In this workload, we gzip/gunzip a 100MB video file for 10 times and wait for 10 seconds in between. As shown in the picture, the workload changes too quickly before the CPU temperature has a chance to catch up. We under-estimate the first few active cycles by quite a bit, but it improves over time as the heat trapped causes the estimate to go up. When the load is removed, the CPU spends a long time dissipating the remaining heat. Overall, the estimated average utilization is 14.16%, which is slightly lower than the actual utilization of 16.13%.

We also examined many mixed workload composing of many different applications, and they tend to have a deviation that is smaller than that from a dynamic workload. Fig. 7 shows a workload composing of many applications from the SPEC CPU 2006 suite [9]. Some are run as a batch for a longer period of time, while others run for a short amount of time, and then they are killed. This workload has an average utilization of 39.4%; whereas, the estimated utilization is 42.1%.

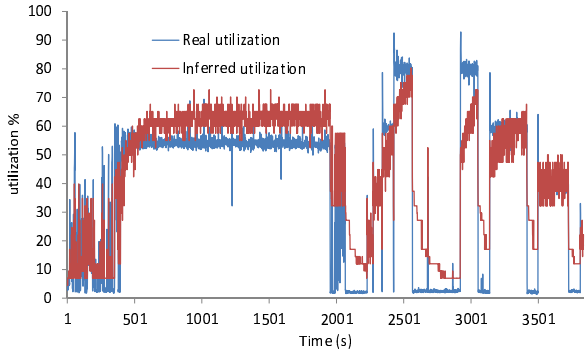

Figure 7. Evaluating the accuracy of equation (7). An example of a mixed application profile. A big batch job at the beginning followed by a few smaller jobs.

C. Sampling errors

We are inferring the utilization from the temperature reading. Besides the errors introduced by parameter estimation (i.e., for c_1/c_2 and T_0), there are also several errors introduced by the cloud environment.

1) Long sampling interval: In order to sample each physical core, we rely on the Xen hypervisor to schedule our virtual core to each of the physical cores. Unfortunately, we have no control over how Xen assigns our virtual core and we may not be able to get on a physical core for a long time. In such a case, we use the last sample we were able to get from that core. Since the temperature may have changed since we last sampled the core, the stale sample reading would introduce sampling errors.

Fig. 8 shows the distribution of inter-sample distance for one of the servers for a total of 24 hours. The majority of samples have a distance of zero from the previous sample, which means that, in most cases, we can sample the core again in the next second. Proportionally, there are much fewer samples that have a distance of more than 0. For example, there are 1144 samples that are 1 second away from the previous sample, compared to 73,762 samples that are 0 second away from the previous sample. Although rare, the tail of the distribution is quite long. There are few cases where we have to wait for more than 4 or 5 minutes before we can sample a core again. Due to the proportionally much smaller number of large sampling intervals, we do not expect it to contribute significantly to the error.

Not being able to get on a core impacts our technique validation in Sec. III-B as well, although we observe a

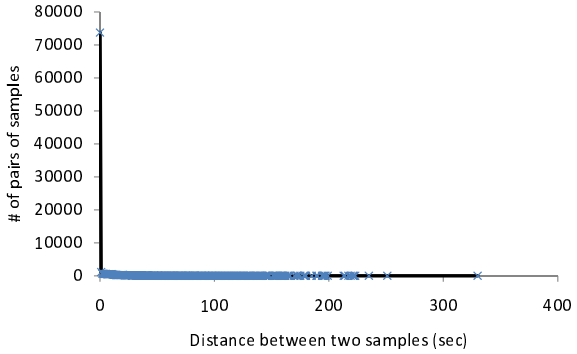

Figure 8. Distribution of inter-sample distance in seconds.

Floating point benchmark		Integer benchmark	
410.Bwaves	44.65	400.Perlbench	42.94
416.Gamess	44.73	401.bzip2	45.35
433.Milc	49.42	403.Gcc	47.81
434.Zeusmp	45.63	429.Mcf	50.12
435.Gromacs	49.47	445.Gobmk	44
436.CactusADM	48.99	456.Hmmer	43.61
437.Leslie3d	48.07	458.Sjeng	44.01
454.Calculix	42.67	462.Libquantum	48.85
459.GemsFDTD	47.63	464.h264ref	45
465.Tonto	45.47		
470.Lbm	50		
482.sphinx3	45.74		

Table I
SPECCPU 2006 BENCHMARK STEADY-STATE TEMPERATURE WHEN THE CPU IS LOADED AT 100%.

smaller inter-sample distance because our physical servers only have 2 cores. The longest we have to wait to get on a core in our lab environment is 163 seconds. Since the inter-sample distance is not much longer in the cloud, we do not expect it to introduce significantly more error than what we observe in our lab environment.

2) Workload imbalance: How workload is assigned to different cores affects the temperature distribution. For the same amount of workload, if all of it concentrates on a single core, that single core would become much hotter. Even though other cores would warm up as well due to the heat spreader and the heat sink spreading the heat, they are typically a few degrees cooler. However, if all cores share the same workload, the temperature across the chip would be more even.

In our lab environment, we test two cases. In one case, we force 100% load onto one core, and in the other case, we even out the load across the two cores (50% on each). We observe that not only the temperature distribution is different, the average temperature differs as well between the two cases, typically by 1 or 2 degrees. Such a difference would affect our measurement accuracy. Fortunately, the Xen hypervisor does a good job evening out the workload across the different cores. We observe that, in the two cloud providers we sampled, the temperature of all cores increase and decrease evenly, both during the chip characterization phase and the measurement phase.

3) Incomplete characterization: In both Amazon EC2 and GoGrid, the small probing VM we use cannot burst beyond a certain limit set by the cloud provider. For example, for standard instances in EC2, we could only drive a *m1.small* instance to use up to 10% of the physical machine's capacity. This would cause errors in estimating the constants using equation (5) as we could not obtain $T(\infty)$ and U pairs for U that is larger than the limit. To see how much off we are in estimating the constants, we characterize 10 *m1.xlarge* instances in EC2. We collect $T(\infty)$ for U from 0% to 75% in 2.5% increments. We then estimate the constants in equation (5) using all data pairs and estimate the constants using only data pairs where $U \leq 10\%$. On average, the constants estimated using partial data is 5.6% off from the estimate given by the full data pairs.

Even though the estimated constants are a little off, we believe their impact on our measurement result is minimal. The reason is because, as we will see in the results section, our observed utilization is often within the range that we are able to drive the CPU utilization. Focusing on data points within the observed utilization range makes sure that we focus on the more relevant characterization data.

4) Workload variations: Different workloads exercise different parts of the chip, and thus, they all generate different amount of heat [10] [11]. Therefore, for the same CPU utilization, two different workloads may cause a different temperature increase.

To illustrate the difference in temperature, we run a subset of the benchmarks in the SPEC CPU 2006 suite [9] on an HP XW4400 workstation with an Intel Core2 6300 1.86GHz dual-core CPU. We run each benchmark for a long period of time for the temperature to reach its steady state, we then take an average of the temperature sample for 60 seconds. Each of the benchmarks we run consumes 100% of the CPU.

Table III-C4 shows the average steady-state temperature for the set of benchmarks we run. As a reference, the temperature reading with zero CPU utilization is 63 degree. The temperature varies widely from 42.67 to 50.12, with an average of 46.39. In comparison, the *lookbusy* workload generator we use to characterize a chip results in a temperature of 45. This means that, for the hottest application, we may be over-estimating the CPU utilization by up to 12.9% ($\frac{63-42.67}{63-45} - 1$). But for the coolest application, we may be under-estimating the CPU utilization by up to 28% ($1 - \frac{63-45}{63-50.12}$).

Even though our estimate could be off, we are close enough to measuring the average of a collection of applications. Our workload generator reaches a steady-state temperature of 45, compared to the average of 46.39 of the benchmark applications. Furthermore, as we will show in the next section, the average utilization in cloud is low, and even

at 28% off, the absolute percentage number would be small. For example, if we infer the average utilization to be 10%, the actual utilization would be between 8.8% and 13.8%, taking into account the overestimation and underestimation factor we see from the SpecCPU2006 benchmarks. Lastly, we note that temperature is a more direct measure of the server's energy consumption and CPU utilization is just a proxy.

D. Measurement results

We launch 20 *m1.small* probing VMs in Amazon EC2 in its Virginia data center. By checking the gateway address, we can easily determine if the VMs are located on the same physical host [12]. Using the technique, we verify that all 20 probing VMs are all running on their own hardware.

Compared to GoGrid, EC2 shows a much greater amount of activities. This is not surprising as EC2 is one of the most popular cloud today. Fig. 9 shows the CPU utilization for one of the measured host. The time shown on the X-axis is in UTC time. The starting time – Sunday 12:00am – is Saturday 5pm in the Pacific time zone. As shown in the figure, there are sustained long periods of time when the CPU is active. But when only one VM is active, its utilization is capped at 10%; thus, the physical CPU utilization remains low. The only time when the CPU utilization is high is when a number of VMs are active at the same time. Unfortunately, this is rare, and we only observe few VMs are active at the same time.

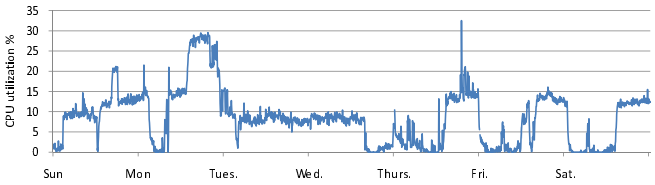

Figure 9. CPU utilization as collected by an *m1.small* probing VM.

The overall average utilization across 20 servers over the whole week is 7.3%. Each server has a different utilization. Averaged over the whole week, the individual server's utilization ranges from 3.6% to 16.9%.

Cloud server utilization exhibits a diurnal pattern. Fig 10 shows the average hourly utilization across the 20 servers for the whole week. It is shown for a whole week based on the UTC timezone. Since the Virginia data center is the first EC2 data center, many people outside of the east coast still use it extensively. To cover the whole usage in the US, we consider 8am to 8pm EST to be the "day" time, which corresponds to 12pm to 12am in UTC time. The average utilization in the "day" is 8.09%, whereas the average utilization in the "night" is 6.76%. From the graph, it is clear that the peak often happens in the day. The highest peak in the "day" can be twice as much as the trough in the "night", e.g., on the Monday. Although we observe the diurnal pattern between day and night, we do not observe much difference between weekdays and weekends. The diurnal pattern suggests that, if we can move computation demands, for example, by using a spot market mechanism, we could even out the usage pattern, and be able to support a higher level of utilization.

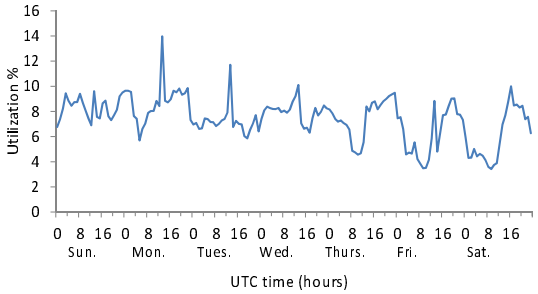

Figure 10. The average server utilization across 20 hosts hosting EC2 standard instances. Shown in UTC time for a whole week.

m1.small is the most popular type of instances at Amazon. To see how popularity affects the utilization, we also look at another type of instances – the high-CPU instances, which run on a separate type of hardware. We launch 10 *c1.medium* instances, and again, we measure for a whole week. The overall average utilization across 10 servers over the whole week is 7.8%. Each server has a different utilization. Averaged over the whole week, individual server's utilization ranges from 4.15% to 16.6%.

Fig. 11 shows the CPU utilization for one of the 10 servers. The average utilization for this server is low for two reasons. First, there is only one VM active. There are either no other VMs assigned to this server, or all other VMs are idle the whole time. Second, the VM is only active during the day; thus, when averaged over the day, the average utilization is quite low.

More interestingly, for five days of the week, the CPU utilization always goes up at around 6:30am UTC (11:30pm PST) and it stays high until roughly 1pm UTC (6am PST). We suspect this is caused by the same VM as the start time and end time on each day are the same. Since it is very difficult to relaunch an instance and have it running on the same physical host, especially for five straight days, it must be that the VM remains on for the whole time. It is often believed that the cloud's pay-per-use model would discourage idle servers. Unfortunately, what we observed indicates that this is not always true. Apparently, the cost of a cloud VM is so low that some users choose to keep the VM on rather than having to worry about saving/restoring the state.

The diurnal pattern is evident in these 10 servers as well. Fig 12 shows the average hourly utilization across the 10 servers for the whole week. The average utilization in the "day" is 8.35%, whereas the average utilization in the "night" is 6.6%. The peak in the "day" is more pronounced on these 10 servers. The highest peak in the "day" can be

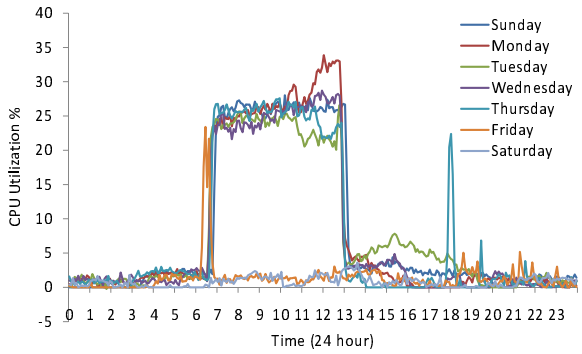

Figure 11. CPU utilization as collected by a *c1.medium* probing VM.

as much as four times the trough in the "night", e.g., on the Tuesday. In addition to the diurnal pattern between day and night, there is also a slight difference between weekdays and weekends. The usage on the Tuesday and the Thursday is notably higher.

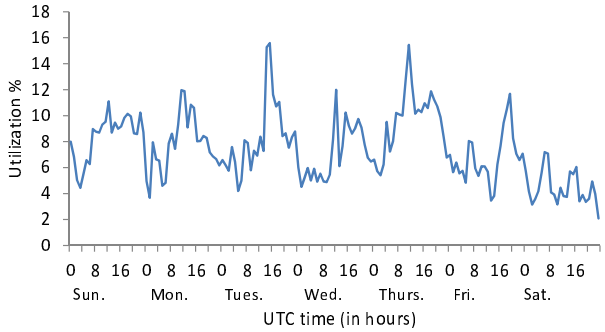

Figure 12. The average server utilization across 10 hosts hosting EC2 high-CPU instances. Shown in UTC time for a whole week.

Lastly, we also use the same technique to measure server utilization in GoGrid. We launch 10 smallest VMs in GoGrid and we measure for a whole week. The overall average utilization across 10 servers over the whole week in GoGrid is 3.2%. Each server has a different utilization. Averaged over the whole week, individual server's utilization ranges from 2.2% to 4.5%. Clearly, its utilization is significantly below that of Amazon's.

IV. Conclusion

Efficiency improvements directly translate into cost savings and a reduction in the environmental impacts. The first step towards improving a cloud's efficiency is to understand what is not yet efficient and to identify areas for improvement. This paper gives an glimpse of cloud efficiency in two state-of-the-art clouds.

We devise a new technique to measure a cloud physical server's CPU utilization. Using the technique, we launch a number of small probing VMs across two different cloud providers, and measure the utilization by other VMs on the same physical host. Our sampling study finds that servers in both cloud providers that we examined are under-utilized. Even though Amazon EC2 is popular and it has many customers using its infrastructure, its server utilization still has a lot of room to improve. Our observation points out that further research is needed on how to improve cloud utilization.

References

[1] H. Liu, "Amazons physical hardware and ec2 compute unit," http://huanliu.wordpress.com/2010/06/14/amazons-physical-hardware-and-ec2-compute-unit/.

[2] A. Weissel and F. Bellosa, "Dynamic thermal management for distributed systems," in *Proc. Workshop on Temperature-Aware Computer Systems(TACS)*, June 2004.

[3] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware microarchitecture," in *Proc. of the 30th International Symposium on Computer Architecture*, June 2003, pp. 2–13.

[4] W. Huang, S. Ghosh, K. Sankaranarayanan, K. Skadron, and M. R. Stan, "Hotspot: Thermal modeling for CMOS VLSI systems," *IEEE Transactions on Component Packaging and Manufacturing Technology*, 2005.

[5] K. Skadron, K. Sankaranarayanan, S. Velusamy, D. Tarjan, M. Stan, and W. Huang, "Temperature-aware microarchitecture: Modeling and implementation," *ACM Transactions on Architecture and Code Optimization*, vol. 1, no. 1, pp. 94–125, Mar. 2004.

[6] F. J. Mesa-Martinez, E. K. Ardestani, and J. Renau, "Characterizing processor thermal behavior," in *Proc. Workshop on Temperature-Aware Computer Systems(TACS)*, June 2004.

[7] D. Carraway, "Lookbusy – a synthetic load generator," http://www.devin.com/lookbusy/.

[8] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, ser. CCGRID '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 124–131. [Online]. Available: http://dx.doi.org/10.1109/CCGRID.2009.93

[9] SPEC, "SPEC CPU2006 benchmark," http://www.spec.org/cpu2006/.

[10] F. Bellosa, A. Weissel, M. Waitz, and S. Kellner, "Event-driven energy accounting for dynamic thermal management," in *Proc. Workshop on Compilers and Operating Systems for Low Power (COLP'03)*, Sep. 2003.

[11] C. Isci and M. Martonosi, "Runtime power monitoring in high-end processors: Methodology and empirical data," in *Proc. 36th Annual ACM/IEEE International Symposium on Microarchitecture*, Dec. 2003.

[12] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *Proc. 16th ACM conference on computer and communications security*, 2009.