Adding Energy Efficiency To Openstack

Vojtech Cima, Bruno Grazioli, Seán Murphy, Thomas Michael Bohnert
ICCLab, Institute for Information Technnology (InIT),
Zurich Institute of Applied Sciences (ZHAW),
Winterthur, Switzerland.
Corresponding author (email): murp@zhaw.ch

***Abstract*— In this paper we consider how energy efficiency aspects can be added to Openstack. With the objective of devising an energy efficient resource manager for Openstack, we first analyze resource and energy utilization on our cloud resources. We observe that there is a large fixed cost associated with server usage patterns and powering down servers is necessary to achieve energy savings. Following this we consider migration mechanisms which are necessary to perform load consolidation: we find that hybrid live migration has the necessary robustness to be part of a sophisticated load management solution. Finally we discuss ongoing work on realizing a load manager for Openstack based on these observations.**

Keywords—cloud computing; energy efficiency; green data centres; smart cities.

I. INTRODUCTION

As the energy consumption of IT resources continues to increase, there is increasing pressure within the industry to ensure it maximizes its energy efficiency. While growth in energy consumption in the broader IT sector, continues to increase, driven by an increasing number of smart devices, the line between dedicated ICT devices and non-ICT devices will blur as we enter the IoT era.

One area within the IT sector which is very aware of energy issues is the cloud and data center segment. While there is some differentiation between these areas, there is also strong overlap and as more and more of organization's workloads move to the cloud, the overlap will increase.

Currently, the Data Centre segment accounts for 2.8% of energy consumption in Switzerland and slightly less – approximately 2% - in both the US and Western Europe [1][2]. As a major energy consumer, the segment has achieved much in terms of energy efficiency within the last decade, through a combination of improved cooling solutions and more efficient transformers on servers [3]. However, there is still more to be done.

From an energy perspective, Data Centres are an interesting use case, as they typically have substantial energy generation and storage capabilities; further, they often have some flexibility in how their workload is managed and how energy is consumed within their facilities. Although the very large DCs are often located in remote places, the vast majority of DCs are located within urban environments and consequently they form an important part of tomorrow's Smart Cities.

978-3-901882-70-8

Realizing energy efficiencies within Data Centres located in Smart Cities is a complex, multi-faceted problem involving understanding of municipal energy generation - including renewables - and consumption, municipal energy policies, energy distribution within cities, DC design and operations, cooling systems and IT systems.

One mechanism to realize energy efficiencies in the DC context is intelligent load management. Facebook has shown that it can deliver energy savings of up to 15% of their total energy bill using a mix of deferring work, backfilling servers with time-insensitive work, powering down unnecessary servers and shifting work to alternate DCs [4]. In this work, we discuss how this can be realized in the context of Openstack - a major open-source cloud computing stack targeted at both public and private clouds and thus the large portion of the DC segment which is less energy efficient.

The paper is structured as follows. In the next section, a brief description of the GEYSER project is given, highlighting the innovations within the project. Following this, there is a section detailing active energy management solutions for cloud contexts, with a particular emphasis on Openstack: this section is divided into subsections focusing on understanding server energy consumption in a cloud context, performance analysis of innovative live migration techniques and some work on developing an energy-based controller for Openstack. Finally, the work is concluded in section 4.

II. THE GEYSER PROJECT

GEYSER [5] is an EU FP7 project which is focused on increasing energy efficiency of DCs located in tomorrow's Smart Cities. A basic premise of the project is that the energy picture within a Smart City will be more complex than today, both in terms of generation and consumption. More specifically, Smart Cities will comprise of many energy generation options, a significant fraction of which will be based on renewables and will contain a local marketplace in which locally generated energy can be traded.

The Data Centre will be an important component in this context, being able to act as a prosumer: providing energy supply when necessary (with significant constraints) as well as having some ability to reduce its consumption when necessary.

Although Data Centres are very heterogeneous - different requirements, different locations, different sizes, different cooling solutions, different networking options, different security solutions etc - they can generally be classified into private or public and public can be further subdivided into co-

location or cloud. While much of the sector today is private, the move to cloud is clear and with increasing maturity of hybrid cloud solutions, more and more of the work will move to public cloud [6]. For this reason, a significant amount of energy within GEYSER is focused on the cloud use case.

The basic objective of GEYSER, then, is to realize a flexible, adaptive DC - or network of DCs - within a Smart City which can match the available energy supply with the available workload in an energy efficient fashion.

At the heart of GEYSER is an advanced Data Centre Management and Control system which can span multiple DCs. This can leverage a number of control actions within the DC such as activating, deactivating chillers, changing the operating temperature, increasing/decreasing the system workload or – in the case of networked cloud DCs – shifting some of the workload to another DC.

As the context for GEYSER is the Smart City context, an important aspect of GEYSER is how it integrates with the Smart City. Specifically, this means that the DC can share its energy generation options and indeed energy storage options with the Smart City but of course this must be done in a very controlled way such that it has no negative impact on the DC operator's business. Another aspect of the Smart City context that is assumed is a dynamic energy marketplace – specifically, in GEYSER it is envisaged that there are multiple markets operating over different timescales which offers the DC another dimension of flexibility in terms of its energy consumption. As renewable energy is an important consideration in GEYSER, forecasting is important. With much renewable energy depending on weather, weather forecasts can be used to infer availability of energy or almost equivalently, energy costs.

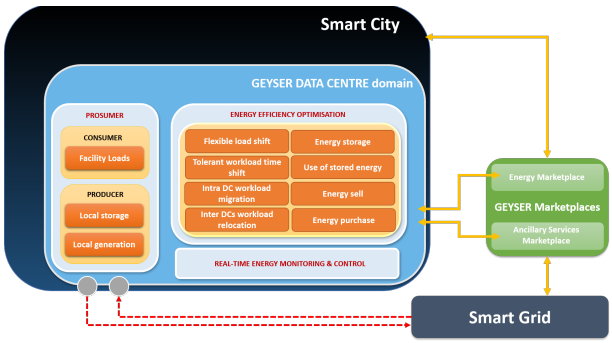

Figure 1: The GEYSER High Level Context [7]

The GEYSER architecture, then consists of many components ranging from data collection within the DC to control systems within the DC to interfaces with local energy marketplaces to interfaces to local weather forecasts. Clearly, then, it is a complex system and indeed one of the challenges within the project is striking the appropriate level of complexity – complex enough that it can incorporate all of this information in a useful manner, but not excessively complex that it will not be a realistic candidate for deployment in the DC context. (The GEYSER context is illustrated in Figure 1 – the system architecture is too detailed for inclusion here).

The cloud related aspects of GEYSER essentially focus on intelligent load management, generally in a networked DC cloud context. Specifically, this means that the cloud workload can be adapted depending on the prevailing energy conditions – if energy is available at a low price, then the cloud operator has an incentive to maximize resource usage; conversely, if energy is available at a high energy price, then the cloud operator has an incentive to reduce resource consumption by shifting workload in time or space. Workload can be increased by incentivizing customers of the service to increase their usage – generally, this can be done via price mechanisms. Time shifting workload typically involves deferring it, such that it can be executed at a time when there is lower utilization of the system and/or energy costs are lower; space shifting workload typically involves moving workload from one data centre to another. The latter is a tool which must be applied judiciously – in many cases, it is not straightforward to move work from one DC to another as, for example, data might be stored in a specific DC; also the process of moving from one DC to another is non-trivial and care must be taken.

In general, the above ideas are not new – they have been studied in an abstract context as well as in the context of more concrete technologies such as Eucalyptus and Open Nebula and indeed concrete supports for this have been developed in those contexts. However, support for Openstack has not been developed as yet: with Openstack being the cloud stack that is currently receiving the most attention in the marketplace, the focus here is on adding such capabilities to Openstack.

Given this context, then, we outline the mechanisms we are using within GEYSER to enable cloud based DCs to adapt in dynamic energy contexts.

III. Active Energy Management in Openstack

The objective of this work is to add sophisticated energy management supports to Openstack. In the context of GEYSER, this includes supports for increasing load and time and space-shifting load in response to the prevailing conditions and signals received from other components within GEYSER. These aspects, however, are still a work in progress and consequently the main focus here is on more rudimentary supports for energy management which focus on powering down servers when they are not required.

To develop an active energy management system, it was first necessary to have an understanding of load and energy consumption within cloud resources. Following this, we focused on a fundamental mechanism which is necessary to perform load consolidation – an important component of the energy management solution. Following that, we describe the work we have done on developing a basic energy management system for Openstack.

A. Understanding Server Energy Consumption

To understand the energy consumption of cloud resources, we added instrumentation to our own Openstack deployment. This is a modest deployment comprising of 13 servers – 10 Lynx Calleo 1240 servers with two Xeon processors and 3 IBM x3550 M4's, also with two Xeon processors. The systems are mid-range systems which have built in energy meters and dual power supplies for redundancy. The purpose of this work

was to combine information on usage of our cloud resources with energy consumption information to understand how usage impacts energy consumption.

1) Empirical Measurement of Server Energy Consumption

To understand the energy consumption of our servers, a basic tool was developed which incorporated information from the built-in energy meters and the Openstack resource usage.

The basic tool was built on the Kwapi [8] energy data collection components which are part of the larger Openstack ecosystem[1]. Kwapi provides support for collecting energy data from disparate energy meters – these could be meters built in to servers or standalone meters with IP addresses (which could be either wired or wireless). Kwapi provides meter registration and polling mechanisms: specific meters require drivers to be written which enable Kwapi to collect the data using the simple interface defined.

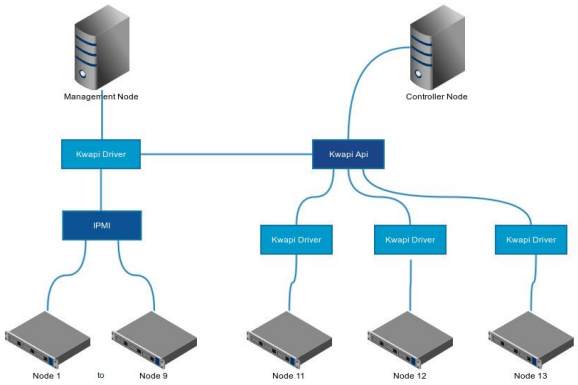

Figure 2: Architecture of Kwapi-based Energy Collection System

The servers in our systems had two different mechanisms to support energy collection. The Lynx Calleo servers provide energy information via the IPMI interface; the IBM servers facilitate access to the energy consumption information inside the host Linux OS. In the former case, Supermicro tools were used to obtain the information from the IPMI interface while in the latter case, it was necessary to install the libaem[2] module. The available energy collection mechanisms (IPMI and via host OS), necessitated functionality to be installed on two different networks – the IPMI control network and the network connecting the hosts. The jump host is connected to both the IPMI network and the rest of the network and hence this was used as the point where all the energy data for the Lynx servers was collected and published to Kwapi. In both cases, it was necessary to develop a simple driver which supports data collection in Kwapi.

Once the data is collected via Kwapi, the data is published to Ceilometer[3], the Openstack data store. There, it is stored alongside the other data that is commonly collected from an operating Openstack deployment, including significant events in the system such as when a new VM is launched, when a volume is created etc as well as regular updates on current consumption, eg how much load on a particular VM. Ceilometer also offers a straightforward programmable interface; it is essentially a database with a simple API wrapper. Consequently, it is relatively straightforward to write applications which leverage Ceilometer data to gain a better understanding of the usage of the resources.

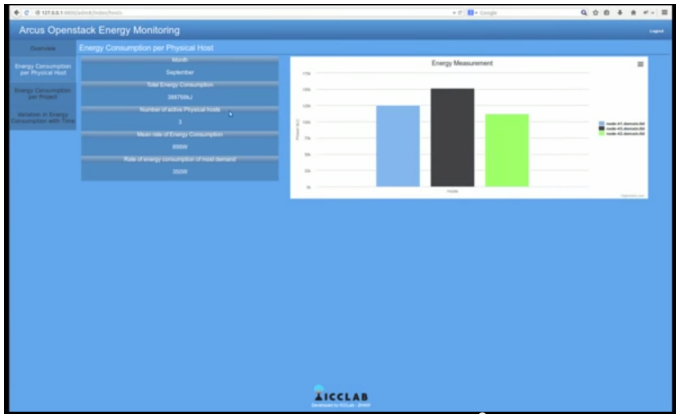

Figure 3: Screenshot from Energy Monitoring Tool

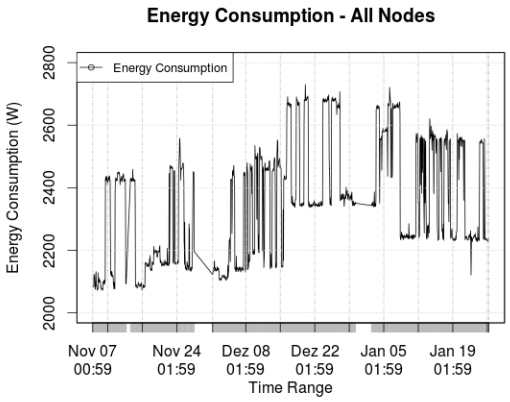

Figure 4: Variation of aggregate server energy consumption with time from Nov 2014-Jan 2015.

Given this, we developed an open-source tool[4] which can enable us to visualize energy consumption information and correlate it with resource usage in the system. A screenshot from this basic tools is shown in Figure 3 above. This enabled us to understand basic parameters relating to both the energy consumption of our servers as well as the usage of our servers. In particular, from this tool, we could see that the usage on our system was such that there was often low utilization followed by a significant spike as some larger data processing jobs were started on the systems (see Figure 4).

The operating range of the servers – from 150W-300W is an important point. Clearly, there are high fixed energy costs in

[1] Kwapi is currently hosted on launchpad which is a peripheral part of the openstack ecosystem.

[2] Libaem is part of the IBM Active Energy Manager suite.

[3] Ceilometer has been renamed Telemetry in later versions of Openstack, but the Ceilometer moniker is still in widespread use.

[4] https://github.com/icclab/arcus-energy-monitoring-tool

current server designs associated with having a server active and idle; naturally, this points at powering down servers when idle as a means to achieve power savings.

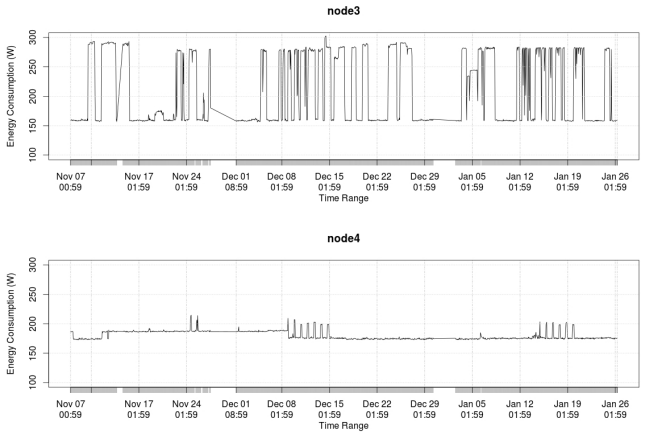

Figure 5: Variation of energy consumption for 2 servers in our system

Server energy consumption varies most with CPU utilization – sensitivity to disk or network usage is low. Consequently, we performed a basic experiment to understand how server energy consumption varies with CPU usage.

The experiment was performed on one of the IBM x3550 servers running a new version of Linux. The synthetic load was generated using the stress[5] tool. Specifically, the load on the processor was increased every interval, by increasing the amount of workers performing parallel computation tasks controlled by stress. (To stress CPU, stress performs repeated calculations of π). The energy consumption information was taken directly from the sensors on the server and CPU utilization was as taken from the server.

The results obtained (see Figure 6) show a somewhat non-linear relationship between total power consumption. Indeed the relationship is somewhat concave with the energy consumption saturating at approximately 50% CPU utilization.

The nature of the curve can be explained by understanding the different components in the system and how work is allocated to them. In general, energy consumption increases as more and more of the components of the system are activated and with the random allocation of work to CPUs performed by the OS, there is a quite even distribution across the available resources. In other experiments, we observed that there is approximately a 20W jump in energy consumption when a processor goes from idle to active (as measured by activity on one of its cores). Also, we noted that there is a minimal increase in energy consumption if the second thread in a hyperthreaded core is activated. These factors combined result in more significant increases in energy consumption as both the processors and cores are activated in the 0-50% aggregate CPU utilization range and a minimal increase in energy consumption above this, as all circuits on the are already active. (Note that we did not consider different CPU governors here – different CPU governors may exhibit different behaviour).

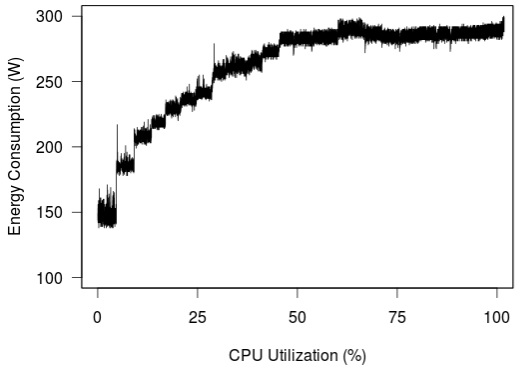

Figure 6: Variation of server energy consumption with CPU utilization

2) Modelling Server Energy Consumption

Although the results relating CPU utilization to energy consumption above exhibit somewhat concave characteristics, we wanted to understand how appropriate a linear model can be in this context, as it is easier to work with from an analytical point of view.

Consequently, we compared the CPU utilization for our servers with the energy consumption of the servers. Given the data that we had collected, it required a little effort to determine the server energy consumption: the data collection focused on the server utilization per VM and it was necessary to map this to the host server utilization. This was done by calculating the host server utilization as the weighted sum of the utilizations per VM, normalized by the number of vCPUs allocated to the VM. This total is then divided by the number of CPUs in the system. More specifically

$$u_{host} = (\sum_{vm} u_{vm} * vCPU_{vm})/CPU_{host}$$

where u_{host} is the total utilization of the hose, u_{vm} is the utilization of the VM, $vCPU_{vm}$ is the number of vCPUs allocated to the VM and CPU_{host} is the number of available CPUs on the host. The utilization parameters are measured in percentage terms.

The CPU utilization determined as above was then plotted with the energy consumption as shown in Figure 7 and Figure 8 above. The data was separated into the different server types as it was observed that the different server types had slightly different characteristics. Further, the data available for the different server types was quite different as can be seen above.

As can be seen from the results, the data set for the Lynx Calleo's is reasonably substantive: although the data set is clearly strongly weighted towards the low utilization (<20%) and high utilization (>80%) states, there are sufficient data points in the large 20-80% interval to conclude that the data set is meaningful. The measure of correlation is 0.89 for this set indicating that the linear model is quite accurate. It is worth

[5] http://linux.die.net/man/1/stress

noting, however that the concave behaviour is clearly visible in the data set with the quite steep rise in energy consumption for many of the data points in the 0-10% range.

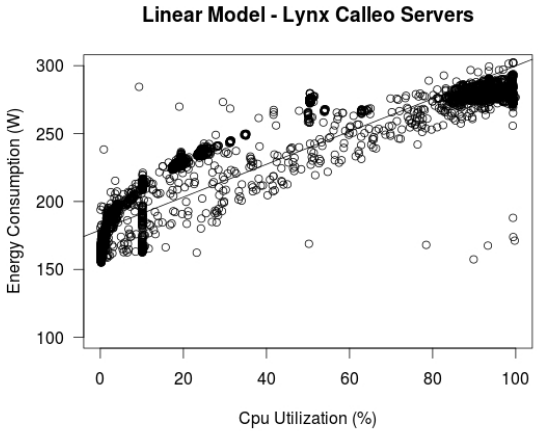

Figure 7: Curve fitting to construct linear relationship between CPU utilization and energy consumption for Lynx Calleo servers

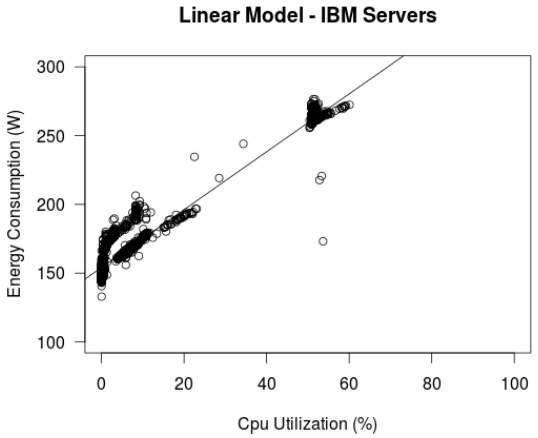

Figure 8: Curve fitting to construct linear relationship between CPU utilization and energy consumption for IBM x3550 servers

The data set obtained for the IBM servers is not as rich. Again, it has a small number of operating points, although this time, the higher operating point is not saturation. A linear model can be determined and indeed the correlation between the empirical data and the linear model is determined as 0.98. However, clearly the data is lacking significantly and the validity of this linear model is questionable.

B. Live Migration: a flexible Load Management Tool

Having gained an understanding of real cloud energy consumption, we then focused on one particular mechanism which is important in management of cloud systems generally and advanced resource management within cloud systems in particular. This is live migration.

Live migration is a basic mechanism that is well established in different hypervisors. Although it is established in different hypervisors, support for cross-hypervisor migration does not exist; consequently, when considering live migration, a specific hypervisor must be considered. In Openstack, the natural choice for hypervisor is Qemu/KVM as this is what most Openstack deployment use. Openstack communicates with the hypervisor through libvirt [9] which offers a uniform hypervisor interface. Consequently, to perform live migration on KVM hypervisors in Openstack, it must be done through libvirt.

Libvirt understands the capabilities of the hypervisor generally and, in particular, whether it supports live migration. Further, the Openstack configuration files can specify some specific flags that get passed to libvirt in certain circumstances if specific hypervisor capabilities should be leveraged.

So, although live migration has existed for some time within hypervisors, it is still relatively new in Openstack. The Icehouse [10] release of Openstack in April 2014 was the first release in which the Openstack community was confident that the live migration supports are reasonably robust.

There are however, some different variants of live migration – here, we describe the different forms of live migration and provide information on how they perform.

1) Pre-copy, Post-copy and Hybrid Live Migration

Here, we describe three variants of live migration. The main challenge in VM live migration lies in moving a potentially large amount of memory in a short amount of time. In general, the amount of memory to be moved relates to both the size of the VM and the activity taking place within the VM. Typically this can be some GB and it must be done in such a ways so as to realize almost imperceptible VM downtime. The problems when performing a live migration mostly relate to changes in memory that happen in the active VM during the migration process.

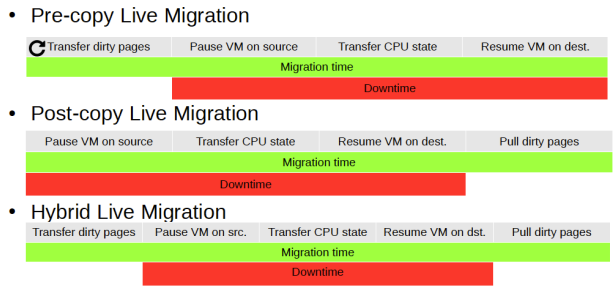

Figure 9: Basic mechanisms of pre-copy, post-copy and hybrid live migration

The standard form of live migration which has existed for some time now is so-called pre-copy live migration [11]. In the basic pre-copy migration process, the destination VM is created while the source VM continues to operate. The source memory is copied to the destination en masse. Next, there is a process in which any memory which has changed – so-called 'dirty pages' – since it was copied has to be sent to the destination again. This process continues iteratively until all the memory of the source has been copied to the destination and then control is passed from the source to the destination.

Post-copy migration is an alternative approach in which the control is moved from the source to the destination at the start

of the process. Then, the memory is moved from source to destination on-demand: whenever the destination needs to access memory that it does not have, it is requested from the source.

Both pre-copy and post-copy have advantages and disadvantages. In the pre-copy approach, the VM state remains in the source until the destination is ready to take over. In contrast, the post-copy approach is one in which VM state is split between the source and destination. Having VM state split between two nodes does introduce a risk that the VM state could be lost entirely if there are networking errors. The pre-copy mechanism repeatedly transfers changed memory from source to destination: if there is intense memory activity during the migration, then it is possible that the migration process could end up repeatedly sending the changed memory data and in the worst case fail to converge. In contrast, as the post-copy mechanism only transfers memory on-demand, the amount of memory that can be transferred is upper-bounded by the size of the VM and the process cannot get into a loop in which it does not converge.

As both approaches have advantages and disadvantages, a natural approach is to consider a hybrid. The hybrid approach is one in which there is a pre-copy phase which transfers most of the memory followed by a post-copy phase during which any modified memory is transferred.

Post-copy migration and hybrid migration have existed for some time as experimental approaches in a Linux context [12]. Because they require enhanced supports from the Linux kernel, they are still raw and have not been put into production. The specific kernel supports required enable a segmentation fault to be trapped and the associated memory requested from a remote machine.

Realizing hybrid live migration in Openstack is not trivial. It requires combining patches to the latest Linux kernel with patched versions of libvirt and then modifying some of the configuration settings in the Openstack installation to instruct libvirt to use hybrid live migration rather than the standard pre-copy approach.

Before integrating it into our energy efficient control solution, we needed to have some understanding of how it behaved. Consequently, we analyzed its performance.

2) Live Migration Performance

Our performance analysis of live migration in Openstack was carried out in on a very basic system – one comprising of three physical nodes: a controller and 2 compute nodes. The VMs were live migrated from one of the compute nodes to the other. The servers used for the work were mid-range IBM x3550's with Gigabit Ethernet interfaces and a Cisco Catalyst 2960G with GigE interfaces.

The objective of the analysis was to understand the capabilities of the different forms of live migration and how they could be used in practice.

The software installed on the system was as follows:

- Openstack Juno
- Libvirt 1.2.11 (with the wp3-postcopy patch by Cristian Klein[6])
- KVM/QEMU 2.1.50 (wp3-postcopy patch by David Gilbert[7])
- Ubuntu 14.04 with modified 3.18 rc3 kernel
    - The patch for handling memory page faults in user space by Andrea Arcangeli[8]

The testing focused on the downtime of the VM, the migration time of the VM and also the amount of data transferred where relevant.

The downtime of the VM was determined rather crudely – ping tests with 100ms intervals were used and the downtime was determined from the amount of lost ping packets: if 3 packets were lost, the downtime was considered 300ms.

VM Type	*vCPUs*	*Memory (MB)*	*Disk (GB)*
Tiny	1	512	0
Small	1	2048	20
Medium	2	4096	40
Large	4	8192	60
Extra large	8	16384	80

Table 1: Characteristics of the different VMs – these are defaults in Openstack

The migration time was determined as the time between the start and end of the migration process. The start of the migration process occurred when the VM was spawned on the destination, using the timestamp of the appropriate message in the log files; the end of the migration was defined as the time when the log files on the source contain an entry that the migration has completed. Naturally, the nodes were NTP synchronized which typically gives accuracy of some µs on a wired LAN; in any case, the measurements were of the order of seconds or 10s of seconds, so NTP accuracy is sufficient.

First, we performed basic experiments to understand the difference between pre-copy migration and hybrid migration on an idle system. (Note that we did not consider post-copy live migration here at all – as hybrid live migration combines the benefits of pre-copy and post-copy we just considered this). The results are shown in Figure 10 and Figure 11. There it can be seen that the migration time in all cases is modest, typically taking less than 10s and there is little difference between pre-copy and the hybrid approach. The migration time can be accounted for by the amount of memory allocated by the guest OS – as the VM is simply a Linux OS with no activity on it, the amount of memory that needs to be used is just the amount of memory required to run the OS. While this increases somewhat with larger VMs, the difference is not so significant, certainly in relation to the difference in VM size (see Table 1 for the characteristics of the VMs).

[6] https://git.cs.umu.se/cklein/libvirt/commits/wp3-postcopy

[7] https://github.com/orbitfp7/qemu/tree/wp3-postcopy

[8] http://lwn.net/Articles/604133/

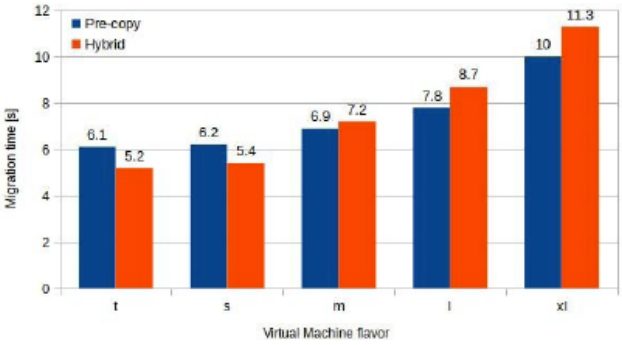

Figure 10: Migration time for unloaded VMs using both pre-copy and hybrid live migration

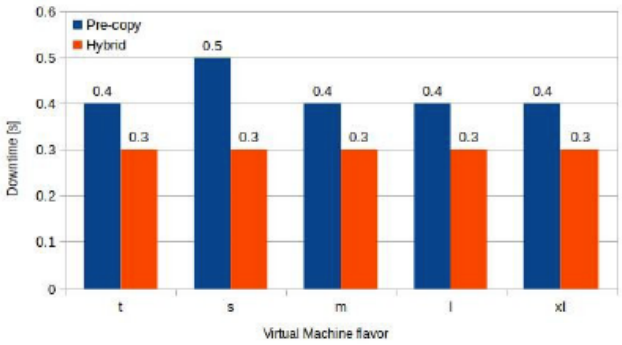

Figure 11: Downtime for unloaded VMs using both pre-copy and hybrid live migration

In the second set of tests, we performed, we considered VMs which were loaded. We used two memory load generation tools – the stress tool mentioned above and the appmembench tool as it offers more accurate control of the rate of change of memory.

In Figure 12 and Figure 13 the relationship between total data transferred and the amount of stressed memory is shown. The total data transfer increases linearly with the amount of stressed memory and clearly it can increase quite significantly (from 2-16GB), depending on the intensity of the memory activity. The migration time also increases, as would be expected and in these more extreme cases, it can take minutes for the migration to complete. It is worth noting that for these higher levels of memory intensity, the pre-copy mechanism failed to terminate – it is not suited to such workloads.

A similar set of tests were performed with appmembench. In this case, it was possible to more explicitly define the Memory Change Rate (MCR). As can be seen in Figure 14 convergence for pre-copy is again unreliable – it only converges in the case that the memory change rate is low. Hybrid live migration is much more robust even though it can take longer to complete the migration task.

Other tests were performed to understand the impact of network load (on the server network interfaces) and CPU load on the migration process. The results of these tests indicate that they do not have a great impact on the migration process – they do introduce 20-30% delays for heavy loads but have not been seen to impact the stability or convergence of the migration process.

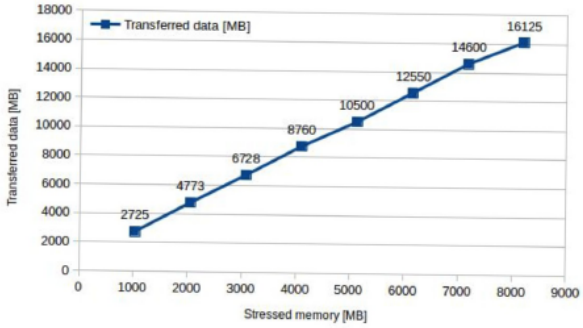

Figure 12: Data transfer for loaded VM – hybrid migration

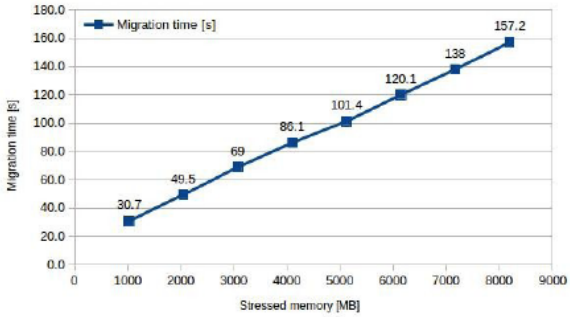

Figure 13: Migration time for loaded VM – hybrid migration

The key observations from this analysis are the following:

- Live migration is a robust mechanism on which a more advanced load management system can be built
- Migration time for large VMs takes some 10s of seconds on Gb/s hardware – we expect that for 10 Gb/s hardware this will drop by an order of magnitude resulting in migration times of seconds even for large VMs
- Downtime for VMs under live migration is typically less than 1s which is very management for most applications

C. Design of an Energy Management System for Openstack

Although all of the above work was performed in the context of Openstack, it alone will not deliver any energy efficiencies. The above work is an input to an energy management system for Openstack which is in it initial stages.

The key observations from the above work are that energy savings can be made via load consolidation and powering down servers; indeed for the usage data pertaining to our cloud, there are clear energy savings to be made using this approach.

In our initial work, we have developed a basic system which determines the load on servers in the cloud. If the utilization is below a threshold, then it migrates VMs off these servers. As live migration in Openstack accepts a host as the

destination, a host with a server utilization above the threshold but below an upper threshold is chosen. If it is possible to move all VMs off the lightly loaded server, then this server can be powered down to realize energy savings.

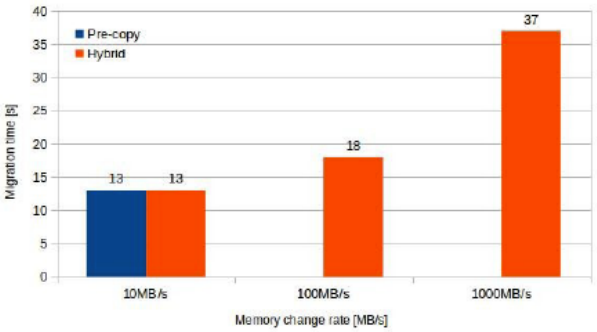

Figure 14: Migration time for pre-copy and hybrid migration with varying memory change rate

If the load on the system increases, then it may be necessary to activate some of the servers which are powered down. WakeOnLan is used to achieve this.

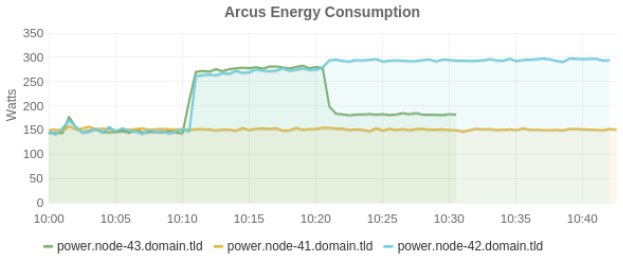

Figure 15: Variation of Energy Consumption in 3 node system

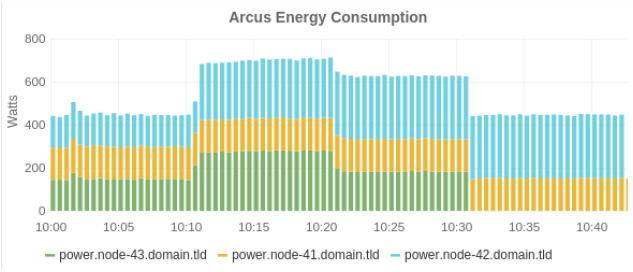

Figure 16: Variation in total system energy consumption in 3 node system

A basic variant of the system has been developed which works with Openstack. As yet, it has only been tested within the 3 node test environment described above. Initial results for the very basic 3 node system are shown in Figure 15 and Figure 16. In this simple example, there are 4 distinct phases: no load on the nodes, load introduced on the nodes and distributed evenly, load consolidation, and powering down unused resources – as can be seen, after load consolidation and powering down resources, the energy saving can be as high as 40%.

IV. Conclusions

In this work, we have considered how energy efficiencies can be added to Openstack. This work is part of a larger EU project focused on the much more complex problem of energy efficient DCs within Smart Cities: our specific part focuses on advanced Openstack cloud based load management to achieve energy efficiencies.

To realize energy efficiencies in Openstack, we developed an energy monitoring tool which enables a cloud operator to view energy consumption on her system. Using the data collected by this tool, we analyzed the energy consumption of our cloud resources where we found that the relationship between server utilization and energy consumption exhibits some concave properties, but can be approximated by a linear model.

Following this, we performed an analysis of different live migration mechanisms to understand their performance. We found that hybrid live migration is robust and can be used as part of an advanced load management solution. Finally, we identified how these findings can be used in an advanced Openstack load manager.

Acknowledgment

This work was partially supported by European Community under the SMARTCITIES-2013 call of the 7th FP for RTD - project GEYSER, contract 609211. The Authors are solely responsible for the content of this paper.

References

[1] "Rechenzentren in der Schweiz - Energieeffizienz: Stromverbrauch und Effizienzpotenzial" IWSB, August 2014 (in German).

[2] Bertoldi, Paolo. "A Market Transformation Programme for Improving Energy Efficiency in Data Centres." (2014).

[3] "Uptime Institute: Data Centre Industry Survey 2013", 2014.

[4] "Making Facebook's software infrastructure more energy efficient with Autoscale", August 2014 https://code.facebook.com/posts/816473015039157/making-facebook-s-software-infrastructure-more-energy-efficient-with-autoscale/

[5] I. Anghel, M. Bertoncini, T. Cioara, M. Cupelli, V. Georgiadou, P. Jahangiri, A. Monti, S. Murphy, A. Schoofs, T. Velivassaki. GEYSER: Enabling Green Data Centres in Smart Cities. In proceedings of 3rd International Workshop on Energy-Efficient Data Centres, June 2014

[6] RightScale 2014 State of the Cloud Report, Rightscale Corporation, April 2014.

[7] GEYSER framework and system specifications, GEYSER Deliverable 2.2, September 2014.

[8] Rossigneux, Francois, et al. "A Generic and Extensible Framework for Monitoring Energy Consumption of OpenStack Clouds." The 4th IEEE International Conference on Sustainable Computing and Communications (Sustaincom 2014).

[9] Libvirt – the virtualiation API, http://www.libvirt.org.

[10] Openstack Icehouse Release notes, https://wiki.openstack.org/wiki/ReleaseNotes/Icehouse, April 2014.

[11] Clark, Christopher, et al. "Live migration of virtual machines." Proceedings of the 2nd conf. on Symposium on Networked Systems Design & Implementation-Volume 2. USENIX Association, 2005.

[12] Hines, Michael R., Umesh Deshpande, and Kartik Gopalan. "Post-copy live migration of virtual machines." ACM SIGOPS operating systems review 43.3 (2009): 14-26.