EPJ Web of Conferences **214**, 07035 (2019) https://doi.org/10.1051/epjconf/201921407035
CHEP 2018

Quasi-online accounting and monitoring system for distributed clouds

Rolf Seuster[1,*], *Frank* Berghaus[1], *Kevin* Casteels[1], *Colson* Driemel[1], *Marcus* Ebert[1], *Colin* Leavett-Brown[1], *Michael* Paterson[1], and *Randall* Sobie[1]

[1]University of Victoria, 3800 Finnerty Road, Victoria BC V8P 5C2, CANADA

Abstract. The HEP group at the University of Victoria operates a distributed cloud computing system for the ATLAS and Belle II experiments. The system uses private and commercial clouds in North America and Europe that run OpenStack, Open Nebula or commercial cloud software. It is critical that we record accounting information to give credit to cloud owners and to verify our use of commercial resources. We want to record the number of CPU-hours of the virtual machine. We continuously collect the CPU usage and an estimate of the HEPSpec06 units of the VM obtained during the boot of the VM and uploads it into an Elastic Search database. The information is processed and published as soon as it is available. The data is published in tables and plots in Kibana and as a cross check in ROOT. We have found the system to be useful beyond gathering accounting information and can be used for monitoring and diagnostic purposes. For example, we can use it to detect if the payload jobs are stuck in a waiting state for external information. We will report on the design and performance of the system, and show how it provides important accounting and monitoring information on a large distributed system.

1 Introduction

The research group for computing in high-energy physics ath the University of Victoria runs workloads for two experiments, for the ATLAS experiment at the Large Hadron Collider at CERN in Geneva, Switzerland and for the Belle II experiment at the SuperKEKB accelerator at KEK in Tsukuba, Japan. These workloads are run on distributed clouds all over the world. Currently we utilize 15 different clouds. Most of these are OpenStack based clouds in North America and Europe. We also use commercial clouds based on Azure, Amazon and Google.

Most of these clouds belong to other institutes, therefore, for proper accounting of the delivered CPU time to institutes, we need to accurately separate the CPU resources used on these clouds. Up to now, there was no need to also separate the type of workloads we run on these clouds apart from keeping the two experiments ATLAS and Belle II separate. We do however have the means to also steer workloads to specific clouds.

First we briefly describe how we run distributed workloads on the clouds. The second part of this paper describes the framework we have setup to accurately and promptly collect accounting information about the resources we utilize on all clouds. The third part of this paper describes how we re-use large parts of the framework for quasi-online monitoring. The

*e-mail: seuster@uvic.ca

last part describes an idea to transfer secrets onto fresh VMs in a secure and reliable way, where we ensure only VMs booted through our system can retrieve these secrets.

2 CloudScheduler

To run distributed workloads on many clouds we use the cloudscheduler framework [1] developed by our group. Figure 1 describes how cloudscheduler operates. Initially, no VMs are running and no jobs are scheduled in the queuing system. We utilize the HTCondor queuing system [2], but other queuing systems could be used as well. Next, a user submits jobs.

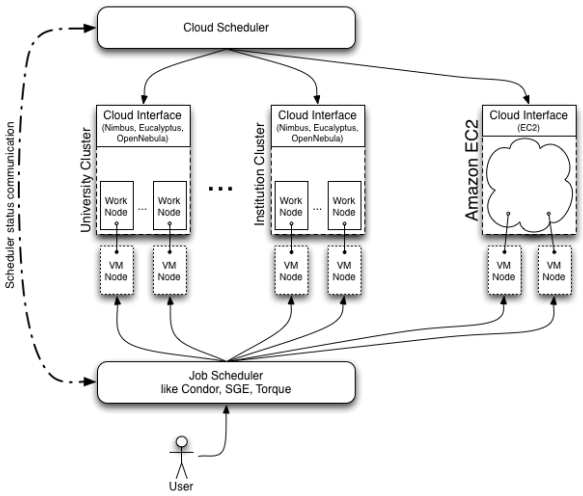

Figure 1. cloudscheduler

Periodically, cloudscheduler enquires HTCondor about idle jobs and free resources. If there are enough resources for the idle jobs, cloudscheduler will wait until HTCondor assigns these jobs to the free resources. If there are not sufficient free resources available, cloudscheduler will boot new VMs on clouds with free VM slots. These new VMs will then run the idle jobs after booting. An important part of the contextualization is to register the new resources with HTCondor. If there are free resources for longer periods of time, cloudscheduler will shutdown these VMs.

In total, we run three instances of the cloudscheduler. Two instances are located at University of Victoria, where we run workloads for the ATLAS and the Belle II experiments. A third instance is located at CERN where we run ATLAS workloads on the European clouds we utilize. This redundancy allows us to increase the efficiency of our system when instances of cloudscheduler become unavailable due to maintenance or other general failures.

3 Accounting Framework

The accounting framework relies on several building blocks. For estimating the computing performance of the VMs we use the fast benchmark of the HEPiX benchmarking group, known as DB12 [3]. For storing the accounting information we rely on the stable operation and excellent performance of the ElasticSearch(ES)/Kibana instance hosted by the IT group at CERN. For collecting and uploading the data into ElasticSearch, we rely on python scripts

and pycurl, because that didn't require additional installation of any software packages on the VM images we utilize.

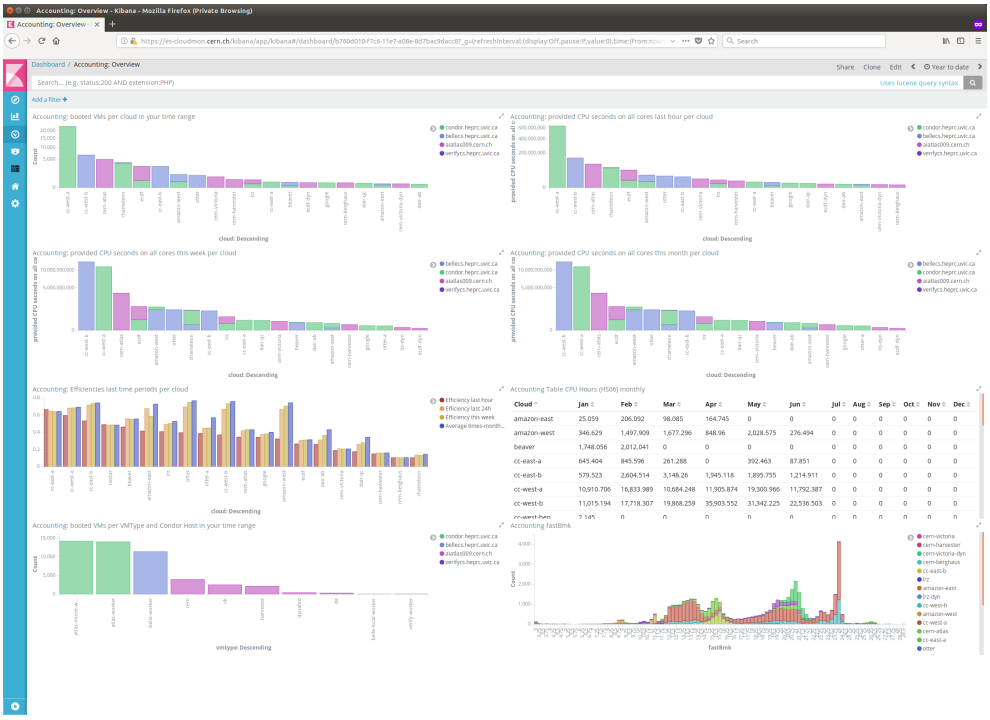

Figure 2. Accounting: The time range of all distributions on this page is shown on the top left, "Year to date". The Screenshot was taken at the end of June. The two top distributions show the number of VMs (left) and the delivered CPU time per cloud (right). The next two plots show the provided CPU time per week (left) and per month (right). The third row shows on the left the efficiency in time windows of last hour, last 24h, last week and last month. The table on the right is the most important entry; it shows the accumulated CPU hours per cloud multiplied by our estimate of the HEPSpec06. The distributions at the bottom of the screenshot show the number of booted VMs per VMtype, reflecting VMs with different configurations. This plot enables the user to quickly apply filters by clicking on a certain VMtype to show only these VMs. The right distributions show the results of the DB12 benchmark, "fastBmk", which we use to estimate the HEPSpec06 for a VM.

When a VM boots, we run the fast benchmark DB12 of the HEPiX benchmarking working group to estimate the performance of the CPU. The DB12 benchmark gives an approximation of the HEPSpec06 units and is typically accurate to within 25 % [4]. These numbers are then parsed for every update from the resulting json file the benchmark produced.

Together with the benchmark score, we upload the uptime of the VM as well as all times given by /proc/stat summed up for all CPUs on this host. These unit of these times are 'ticks', [1] which we also store in ElasticSearch to be able to correct to seconds for all VMs.

We utilize several time windows to these numbers. For detecting trends quickly, we store these numbers for the last hour, last day and last week. For the accounting we store these numbers for the last month. The collector script is run hourly, and also daily, weekly and monthly to be able to compute the difference between these points in time and the current

[1] On most Linux systems correspond this unit corresponds to 1/100ths of a second

numbers for the upload. The difference is calculated in the upload script in python on the VM immediately before the upload into the ElasticSearch instance.

In order to avoid heavy computations and heavy post-processings on the ElasticSearch instance, we do most computations on the VMs itself. Mostly, we were interested in monthly resource utilizations. Therefore, we create on document per VM per month, which we keep updating with current numbers for the various time windows. The monthly CPU numbers are uploaded to ElasticSearch in variables which names include the month, so that we can calculate the CPU utilization for a given month just by summing up all CPU numbers in the variables that contain the name of this month.

If a VM is up during two or more calendar months, it will create two or more documents in ElasticSearch. We upload the documents with a specific name, that contains the name and boot time of the VM, plus the current month and year. In effect, we are overwriting documents for all running VMs, which also means that all information must be present in all the uploaded documents. This differs from updating an existing document, where only the updated information is needed.

With these preparations, only light calculations are needed on the ElasticSearch instance. We need to correct for the unit used in the CPU numbers as well as sum up monthly numbers in the table used for the accounting. For calculating the provided CPU Hours, we multiply the CPU times by our estimate of HEPSpec06 based of the DB12 benchmark.

The results for the provided CPU Hours are shown in the screen shot in Figure 2, separated into each month and into each cloud we use. The meaning of the other distributions in this screenshot can be found in the caption of this figure.

The stored numbers in ElasticSearch allows us to addionally calculate efficiencies, which are defined as the CPU times spent in user mode divided by the uptime. This we do for the various time intervals we use to store these numbers. Here, we can clearly distinguish different behaviours of the different clouds. Typically, commercial clouds perform very well, and opportunistic resources have a much wider spread in performance. Typically the efficiency also depends on the current workload that is run on a particular cloud as well as the general configuration of the hardware the cloud runs on. For example, IO intensive jobs don't perform well on clouds that don't use SSDs for the local storage.

Updating existing documents comes with the additional advantage that failed updates due to temporarily bad connections between the VM and the ElasticSearch instance will be corrected with the next upload an hour later. So, the accounting information would be off by at most one hour, meaning one missed update.

Figure 2 shows a screen shot of the ElasticSearch instance as seen in June. A fourth cloudscheduler instance is visible, verifycs.heprc.uvic.ca, which is used for testing, but the CPU resources it managed are negligible.

4 Quasi-online Monitoring

With a small extension of the accounting framework, in a second step we also implemented a Quasi-online monitoring for checking the success of the payload, the experiments' workloads. The source of information is experiment dependent, as well as the procedures to retrieve information about job successes and failures. Common to both experiments is the location of the ElasticSearch instance. Since the success rate of jobs is mostly interesting only for a short periods of time after the job finishes, there's no strong requirement of keeping old data permanently. Therefore, we store this information on an ElasticSearch instance running on a VM at the University of Victoria, without backing up the data.

The huge advantage our monitoring provides over the experiments' own monitoring is that we can easily separate jobs depending upon which cloud they ran. Typically, this in-

formation is lost in the experiments' databases, because they are not aware that we run on distributed clouds. One could identify different clouds from the hostname, which both experiments store, but this is tedious, and newly added clouds would likely need code changes to properly identify them. Our approach will automatically add new entries for new clouds and also allows users to apply filters easily, e.g. only select a certain cloud.

4.1 Quasi-online Job Monitoring in ATLAS

For the ATLAS experiment, all information required for the monitoring of the job successes can be collected outside of the worker-nodes. One piece of information must be collected on the machine that runs cloudscheduler and is collected once an hour. This information connects the host-name of the VM to the cloud on which they ran. It also stores which jobs were running on which VM, so even single faulty VMs can later be identified. Further information comes from HTCondor, which is retrieved in the same script.

The second source of information is the panda job database [5]. Again, once an hour, a script enquires about all jobs that ran on our Panda site(s) and retrieves it's successes or failures or marks if a job is still running. Here, the document name is the GlobalJobID, which is used to match the information from the two sources. Short running jobs might only be registered on the panda database, and could be missing from information retrieved from cloudscheduler and HTCondor, so the cloud would be marked as "Unknown".

Figure 3 shows the screenshot of the ATLAS Job monitoring. The most relevant plots are in the second row, which shows the Job Status (left) as well as the Error Diagnosis in case of failures (right). The other plots are described in the caption.

4.2 Quasi-online Job Monitoring in Belle II

Also for the Belle II experiment, two sources of information are used. Here, however, we need to run scripts on the VMs to collect information. Every 15 minutes, a hook in HTCondor runs a script, which will use HTCondor to transfer its output back to the condor server at our home institute. Amoung other information we transfer back the job ID. This ID allows us to later enquire the DIRAC job database about the job status.

Additional information is collected on the server running HTCondor and cloudscheduler. A script extracts from HTCondor all job IDs of the jobs that ran on all clouds we use. Each job will create a new document. The status of these jobs will initially be marked as "Running". A bulk upload into ElasticSearch is done at the end of this script.

A second script then enquires ElasticSearch for all jobs that are still marked as "Running". It will create a bulk request into the DIRAC job database to update their job statuses and send the updates of their statuses into the ElasticSearch instance.

The left plot in the second row in Figure 4 shows the most important distribution for the Belle II Job Monioring, the Status Codes. In the third row, the distribution of number of jobs at the 10 largest sites is plotted. This is just to illustrate how our site compares to other sites, because we typically provide between a fourth up to a third of the overall computing resources to the Belle II experiment. All other plots are discribed in the caption.

5 How to transfer Secrets onto VMs

The ElasticSearch instances we utilize are all secured with a username and password for uploading data. Therefore, the question arises, how we can transfer this username password combination securely onto our VMs.

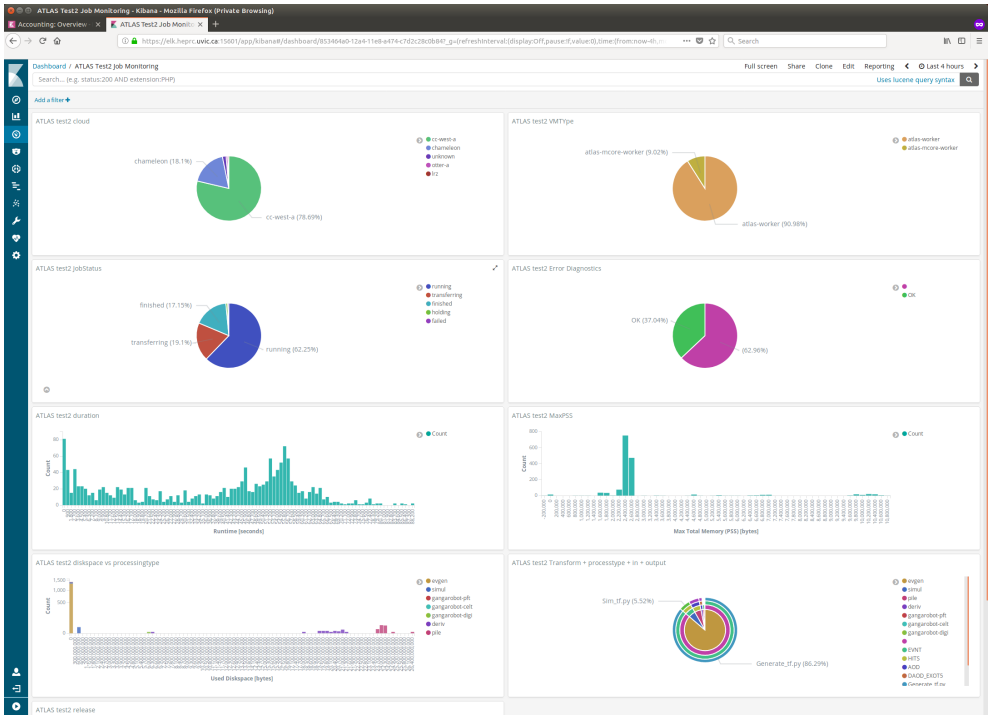

Figure 3. ATLAS Job Monitoring: The time window for all plots in this screenshot is shown in the top right: "Last 4 hours". The two top plots show the cloud and the VMType of the VMs we are running and allow the user to quickly apply filters. The next row of plots shows the status of the jobs as retrieved from the Panda job database. Most jobs in this time window are still running or transfering their output. The Error diagnostics gives further details in case jobs failed. The 4 plots in the lower half of the screen shot show further information about the jobs from the Panda database like the runtime, the memory consumption and the disk space usage of the jobs. Also shown is the job type, denoted as "Transform + processtype + in + out", which combines 4 highly correlated variables.

In the following, we discuss an idea we implemented on how to securely transfer secrets onto newly booted VMs where we ensure that these secrets are available only to VMs we booted. These secrets could be GSI keys, or the username and password used for the Elastic-Search instances we use. Once HTCondor is configured properly to use GSI, one can use the HTCondor to transfer additional secrets, but the question remains how to transfer these GSI keys securely on the the VMs. There are basically two vectors of attack, one where someone from outside listens to our traffic and sniffs secrets or an outsider joins our HTCondor pool and listens this way to exchanged secrets. The OpenStack traffic goes typically through HTTPs connections, so has some security already build-in. But how can we further improve the security ?

On the cloudscheduler server (CS), the procedure when CS initiates to boot a VM is now as follows, which is also described in Figure 5

1. CS initiates to boot a new VM on a cloud. At the same time it will generate a binary file 'RandomFile' of N random bits which is specific to this VM. The secret will be encrypted with these random bits to yield the 'encrypted payload'.

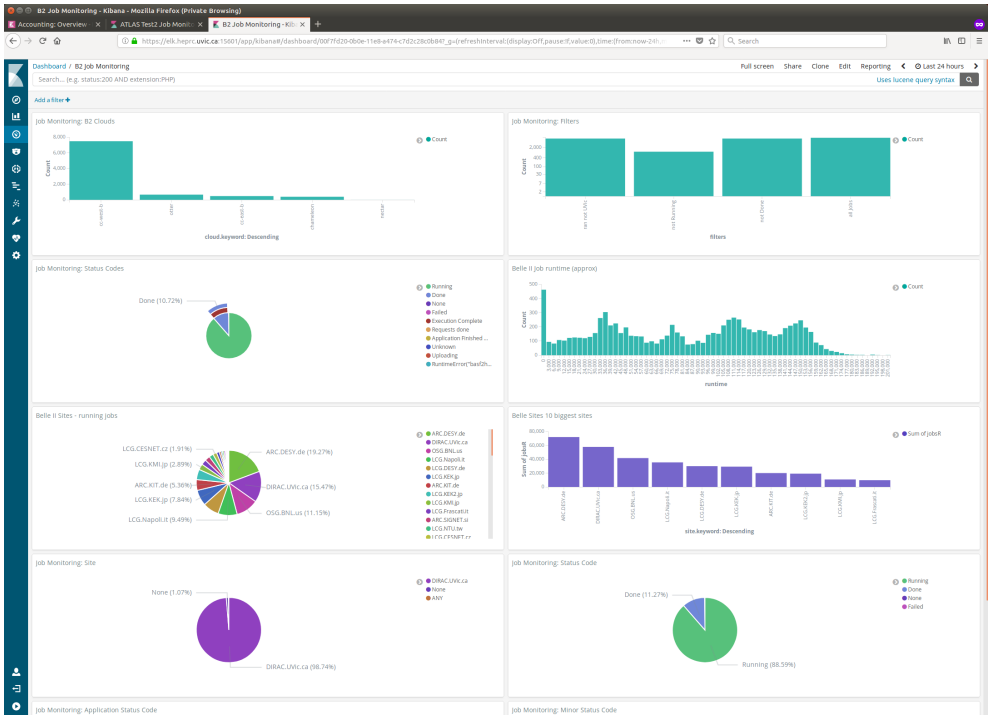

Figure 4. Belle 2 Job Monitoring: Here the timewindow is last 24h. The plots at the top show the clouds and left other variables commonly used for filtering. THe second row of plots show the interesting quantity on the left, the Status Code. Most of the jobs are still running. On the left, the approximate runtime of the job, basically it shows how long this job was seen on our VMs. The third row is a convenience for us, it shows the top 10 sites conntributing to Belle II computing, left as a pie-chart, right as a bar graph. The bottow row shows where jobs eventually ran - failures are typically retried at a different site up to three times. Right is again the job status.

2. On the VM a normal ssh public and private key pair are generated during the boot process. The VM then requests the encrypted secret via a HTTP(s) request to CS. The HTTP request contains the public part of the ssh key pair, in practice it is gzipped and base64 encoded:

   ```
   curl http://cloudscheduler/secret.tar?pubkey
   ```

3. CS then decodes and un-gzips the public ssh-key and encrypts the 'RandomFile' with it. CS also tars the encrypted 'RandomFile' as well as the 'encrypted payload' into a file secret.tar, which it sends back in response to the HTTP request from the VM.

4. The VM then decrypts the encrypted 'RandomFile' with the private ssh-key and can then decrypt the secret with the original 'RandomFile'.

With this procedure, the transfer of the secret is secured, only CS and the VM have unencrypted copies of it. This does not yet protect against an imposer sending an properly constructed HTTP request. To further strengthen the security, a shared secret between the CS and the VMs can be applied to add additional layers of encryption of the 'RandomFile'. This could be for example the requestID of the boot request of OpenStack.

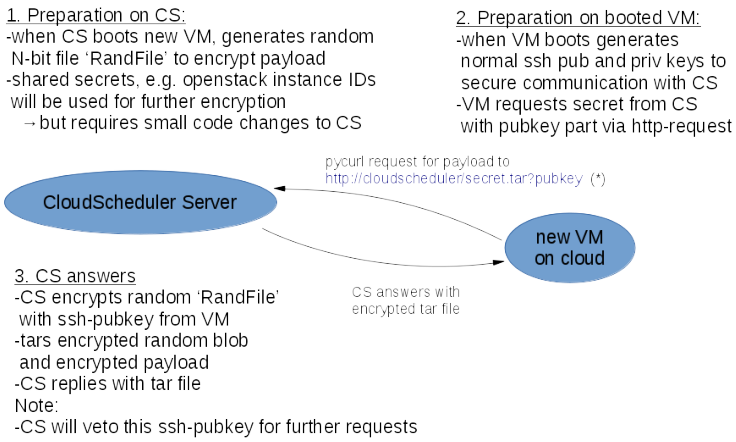

Figure 5. Suggested procedure transfer secrets onto a freshly booted VM. To further improve security, a shared secret between cloudscheduer and the VM should be used to add additional layers of encryption.

6 Conclusions

Accounting in our distributed cloud computing is done by populating an ElasticSearch instance and displaying results in a table format in Kibana. The framework gives other useful information as well, like efficiencies calculated by user times over uptimes of VMs. This has been proven to be stable and reliable, as well as accurate and with a very small delay of one hour at most. The sytem has been proven to be extremely useful for our purposes.

An extension of the accounting system is the job monitoring framework, which helps us identifying mis-behaving clouds. It provides a wealth of information and can help understanding cloud specific problems.

The presented transfer of secrets into VMs is a novel approach which will be fully integrated into the cloudscheduler and provides additional security.

References

[1] R Sobie, A Agarwal, I Gable, C Leavett-Brown, M Paterson, R Taylor, A Charbonneau, R Impey and W Podiama *Science Cloud '13 Proceedings of the 4th ACM workshop on Scientific cloud computing*

[2] HTCondor: High Throughput Computing: http://htcondor.org

[3] HEPiX Benchmarking working group: http://w3.hepix.org/benchmarking.html

[4] D. Giordano, M. Michelotto, M. Alef, "Next Generation of HEP CPU Benchmarks", Proceedings of this Conference

[5] T. Maeno, et al. "PanDa for ATLAS distributed computing in the next decade" J. Phys.: Conf. Ser. **898**, 052002 (2017)