Noname manuscript No.
(will be inserted by the editor)

High-throughput cloud computing with the Cloudscheduler VM provisioning service

F. Berghaus, K. Casteels, C. Driemel, M. Ebert, F. F. Galindo*, C. Leavett-Brown, D. MacDonell, M. Paterson, R. Seuster, R. J. Sobie, S. Tolkamp, J. Weldon

Received: date / Accepted: date

Abstract We describe a high-throughput computing system for running jobs on public and private computing clouds using the HTCondor job scheduler and the Cloudscheduler VM provisioning service. The distributed cloud computing system is designed to simultaneously use dedicated and opportunistic cloud resources at local and remote locations. It has been used for large scale production particle physics workloads for many years using thousands of cores on three continents. A decade after its initial design and implementation, Cloudscheduler has been modernized to take advantage of new software designs, improved operating system capabilities and support packages. The updated Cloudscheduler is more resilient and scalable, with expanded capabilities. We present an overview of the original design and then describe the new version of the distributed compute cloud system. We conclude with a review of the current status and future plans.

Keywords particle physics · cloud computing

Department of Physics and Astronomy, University of Victoria, Victoria, British Columbia, Canada
*TRIUMF, 4004 Wesbrook Mall, Vancouver, BC, Canada

1 Introduction

Clouds are increasingly being used for research computing (for example, see [1]). They provide a way to access significant computing resources without the need for investment in large, field-specific infrastructures. Commercial clouds offer a wide range of services and platforms, as well as large scale computational resources. Many research centres also operate compute clouds that may be dedicated to particle physics or shared with researchers in other fields. If a centre is committed to providing resources to a specific particle physics experiment (usually with a signed agreement), then it is considered a *pledged* resource; otherwise it is *opportunistic*. Compute clouds at research centres can provide both pledged and opportunistic resources.

Researchers may find themselves with a choice of many clouds: commercial and private, both pledged and opportunistic. Commercial clouds are often not an option as, in many countries, research grants cannot be used to lease computers. Our goal is not to discuss the merits of private versus commercial clouds; we refer the interested reader to two recent studies [2,3].

This paper describes a solution for running high-throughput computing workloads on compute clouds, irrespective of the underlying cloud software, location or its ownership. We call our design a *distributed compute cloud* where the resources are an aggregate of compute clouds hidden from the users.

The distributed compute cloud uses Cloudscheduler for VM provisioning and scheduling, and HTCondor for job scheduling. Briefly, Cloudscheduler reviews the HTCondor job queue and cloud resources to determine whether there are clouds that can start VMs that meet the job requirements. If a match is found, then Cloudscheduler requests that the appropriate VM image be booted on the cloud. Once the VM is instantiated, it joins the HTCondor pool and the user job is sent to the running VM instance.

The original design of Cloudscheduler was conceived in 2009 [4] and is based on ideas discussed in a paper describing "sky computing" [5]. The distributed compute cloud has run many millions of jobs on three continents for the ATLAS experiment at the CERN Laboratory in Geneva, Switzerland [6] and the Belle II experiment at the KEK Laboratory in Tsukuba, Japan [7].

Many of the custom and external software components have undergone significant change since the first version of Cloudscheduler. Further, the years of production operation have provided insight on how to simplify and improve the system for usability, reliability and maintenance. In 2018, Cloudscheduler was changed from a single Python code framework into a software platform with expanded functionality using current software technologies and practises. The new design is more robust, error tolerant and scalable, and is currently used for ATLAS and Belle II production workloads.

We remark that there are other strategies and software for utilizing clouds in particle physics, such as Vcycle [8], VMDIRAC [9], and HTCondor [10]. The design of these systems is different and complementary to our model. For further information on the other solutions, we refer the interested reader to the cited papers.

In this paper, we give an overview of the original design of the distributed cloud system using Cloudscheduler Version 1 (CSV1) in Section 2. Section 3 describes Cloudscheduler Version 2 (CSV2). Throughout the paper, we highlight the key external components used in both versions of the distributed compute cloud that have significantly improved its capability and reliability.

2 Cloudscheduler V1 distributed compute cloud

2.1 Overview

We briefly describe CSV1 to give some context and motivation for the new version. The architecture of the distributed compute cloud using CSV1 has been described in a number of papers [11,12] and is shown in Fig. 1. The key components are the HTCondor job scheduler, the Cloudscheduler VM provisioning service and the compute clouds. CSV1 provides API support for Openstack, Open Nebula, Amazon EC2, Microsoft Azure and Google GCE clouds.

HTCondor was selected as the job scheduler as it was designed to be a cycle scavenger [13], making it an ideal fit for a dynamic cloud environment where resources (VM instances) appear and disappear. The user or workload management system submits a job to the HTCondor job scheduler, specifying the job requirements (e.g. number of cores, memory and disk space) in the Job Description Language (JDL) file. It was observed that multi-core VMs make optimal use of the resources due to the sharing of disk caches, independent of whether a single job requires one or all of the cores in the VM. The HTCondor client on the VM starts a partitionable slot during the contextualization process that is subdivided depending on the resource requirements of the job.

CSV1 examines the list of pending jobs in the HTCondor queue and searches for a cloud with free resources that meets the job requirements as specified in the JDL file or as specified by the system-wide job defaults. If there is a cloud that meets those requirements, then CSV1 sends a request to the cloud to boot a VM. The VM image can be optionally specified in the JDL file. Once the VM is booted and contextualized, it joins the HTCondor pool and queued job(s) can start on this VM.

The CSV1 workflow is executed approximately once per minute. Decisions on VM provisioning are based on the information about the jobs and clouds. CSV1 retains the state information in memory and can write it to disk using the Python pickle module. The disk file makes it possible to restart CSV1 if there are minor issues or there is a simple code change. More significant changes or outages require the entire system to be shutdown.

CSV1 can request the start, retirement or immediate destruction of a VM. The start and destroy commands are directly issued via the cloud API. If there are no jobs in the HTCondor queue, then a running VM may be retired. A retire request is issued to the HTCondor client on the VM and the client deregisters from the HTCondor pool. The jobs running on the VM are allowed to finish and then the VM is destroyed.

HTCondor and CSV1 are managed via their respective command line interfaces (CLIs). Only an administrator can configure the system, add or remove clouds and edit the files used for the contextualization of a VM. Users can query the status of jobs and VMs using the HTCondor and CSV1 CLIs, respectively. CSV1 does not have a GUI for managing the system, although we have created a monitoring web page using the data retrieved with the CLI. In addition, we have developed customized scripts that populate an Elasticsearch database with information from CSV1, and also information about the application jobs (e.g. CPU time usage and job status indicators). Each VM also periodically writes to Elasticsearch with its current status.

2.2 Performance and status

The distributed compute cloud is integrated into the WLCG grid infrastructure [14] and has run production workloads for many years using clouds in North America, Europe and Australia. The majority of clouds used the Openstack software, though Open Nebula, Amazon EC2, Google GCE and Microsoft Azure clouds were also used.

We ran two instances of the CSV1 system in North America for ATLAS and Belle II as it is not designed for multiple HTCondor instances. We use separate HTCondor instances for ATLAS and Belle II for simplicity and robustness. We still operate a CSV1 system in Europe and plan to transition to CSV2 by the end of 2019; our goal is to manage all resources with a single instance of CSV2.

The HTCondor instance dedicated to the ATLAS experiment is linked to the PanDA workload management system [15]. The production team need not be aware that the resources associated with the HTCondor instance use multiple cloud resources. ATLAS typically uses a few hundred thousand cores at a given moment, and the distributed compute cloud contributes approximately 1% of the resources, comparable to the other Tier-2 compute centres operated in Canada. ATLAS (and Belle II) submit a *pilot job* that sets up the environment for the job and contacts the central system for a payload job. The pilot job can run more than one payload job depending on its configuration.

The Belle II experiment uses the DIRAC workload management system [16]. We operate three DIRAC Site Directors in Victoria that submit pilot jobs to the HTCondor instance if there are queued jobs in the central DIRAC server at KEK. The distributed compute cloud has provided 13% of the total workload of the experiment since January 2018. All of the Canadian resources for Belle II will continue to be provided by the distributed compute cloud.

3 Cloudscheduler V2 distributed compute cloud

3.1 Overview

The distributed compute cloud has been redesigned to improve reliability and scalability with expanded capabilities. CSV2 is a framework of components written in Python 3. The in-memory data of CSV1 are replaced with a MariaDB database [17]. CSV2 has independent scheduling, polling, and user interface processes that track the state of the clouds and the HTCondor pool (see Fig. 2). The database significantly improves the VM management and job scheduling, and makes it easier to respond to outages and software updates.

CSV2 introduces a RESTful web User Interface (UI) for administration, control, and detailed monitoring of the clouds and jobs. In contrast to CSV1, the CSV2 UI can be accessed through either a graphical user interface

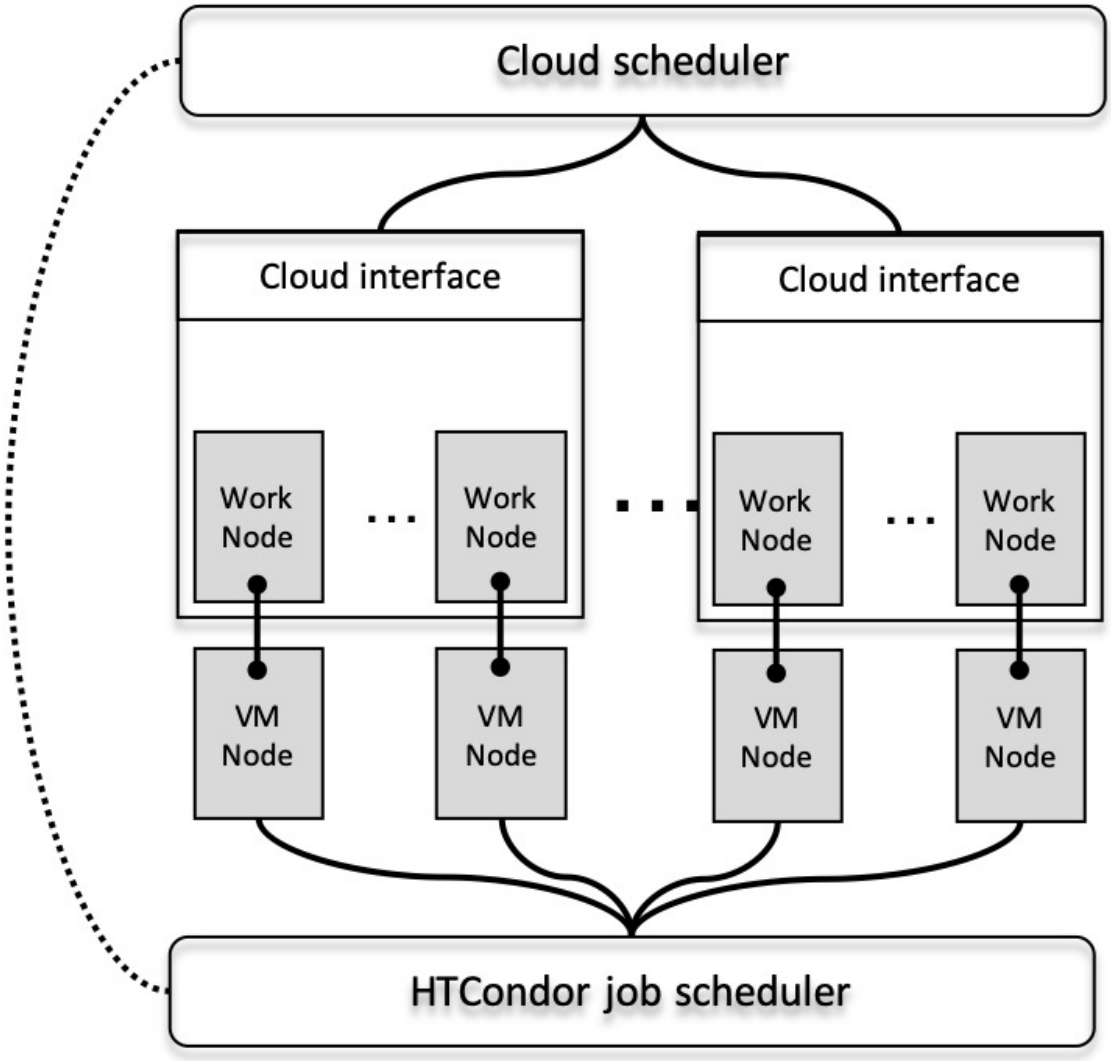

Fig. 1 Overview of the Cloudscheduler Version 1 and HTCondor distributed compute cloud. A user or workload management system submits application jobs to the HTCondor job scheduler. Cloudscheduler reviews the job queue and the resources on the compute clouds. If there is a cloud with resources that meet the requirements of the job, then Cloudscheduler issues a command to boot the user-specified VM. The instantiated VM registers with HTCondor and the job is submitted to the VM.

(GUI) using web browsers or a command line interface (CLI) client (with extensive man pages); the GUI and CLI have nearly the same functional capabilities. In both cases, the connections are authenticated either by x509 proxy certificates or by usernames and passwords. In Fig. 3, an example of the graphical status display is presented.

CSV2 has *pollers* that fill the MariaDB with information from the clouds and HTCondor. Fig. 2 shows a single *cloud poller*; however, there are separate *cloud pollers* for each cloud-type[1] (e.g. one poller can support multiple Openstack clouds). Fig. 2 also shows pollers for HTCondor (labelled as *HTC pollers*). There is a *HTC machine poller* that gathers information on the HTCondor machines and a *HTC job poller* that gathers information on the jobs in the HTCondor queue. Although Fig. 2 shows only one HTCondor instance, CSV2 can support multiple instances. For example, separate HTCondor instances are used by ATLAS and Belle II experiments, in part, due to differing security requirements. All HTCondor instances are supported by a single set of HTCondor pollers. However, each instance of HTCondor requires a *HTC Agent* to provide the correct security context when issuing requests to the HTCondor client on the VM (e.g. requesting a VM to de-register from the HTCondor pool).

[1] Currently there is API support for Openstack and Amazon EC2 clouds (support for other cloud APIs will be added on demand)

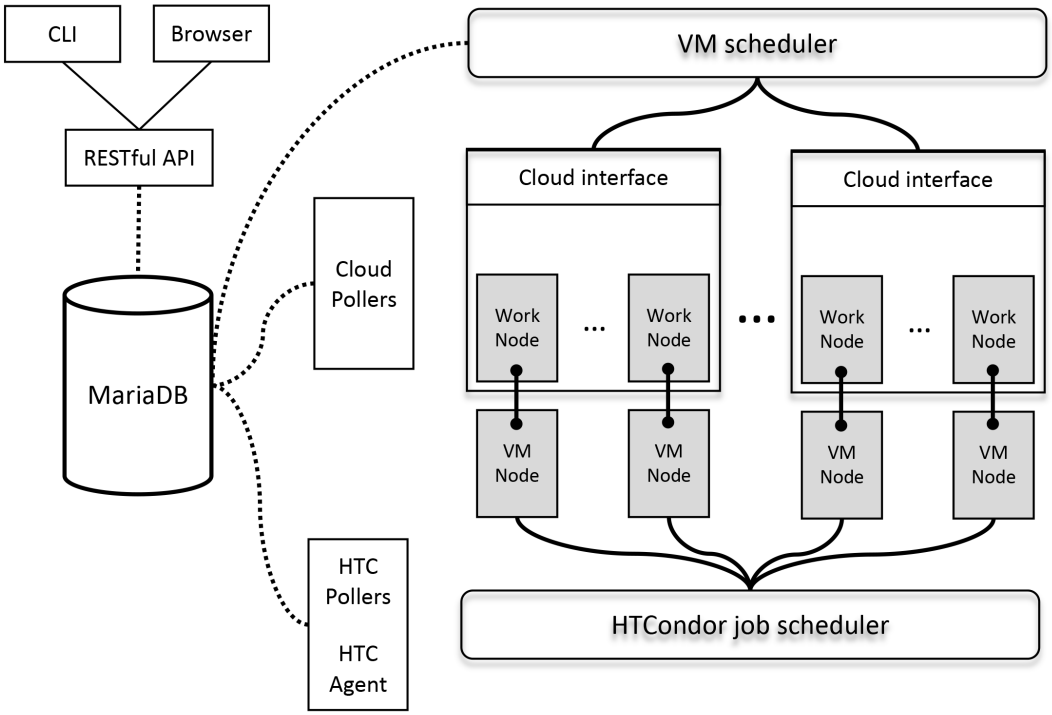

Fig. 2 Overview of the Cloudscheduler Version 2 and HTCondor distributed compute cloud. The primary change is the introduction of the MariaDB for keeping track of the state of the system and "pollers" to gather information from the clouds and the job scheduler. The information in the database is used for scheduling, management and monitoring.

3.2 Workflow

The workflow is determined by the configuration of the system, the job information and the state of the VMs. The state of the VMs (see Table 1) is dynamically determined by the information gathered by the *cloud poller*, *HTC job poller* and *HTC machine poller*, and stored in the MariaDB. We describe the typical workflow of an application job.

If there is a job in the HTCondor queue, then the *HTC job poller* creates a new entry in the MariaDB with the job information. The *VM scheduler* periodically queries the MariaDB for queued jobs and determines if there is a cloud with free resources that meet the requirements of the queued jobs. If the *VM scheduler* finds a cloud that is matched to the job(s), then it issues a request to start a VM (via the cloud API) to the cloud and creates an entry in the MariaDB for each new VM. A subsequent query to the MariaDB would show the VM to be in a *starting* state (boot process).

The contextualization of a VM at boot time is defined to be the process of configuring a VM at its deployment time [18] and is controlled by metadata that is passed to the "cloud-init" process [19]. There are a number of obligatory items to configure such as the HTCondor client. CSV2 provides default metadata files to manage all obligatory items (stored in the MariaDB). The ATLAS and Belle II software environments are configured during the contextualization. The user can add other files to set up the VM as required. For example, we run a benchmark suite to estimate the processing capability of the VM and configure the VM so that it periodically updates an Elasticsearch database. See Section 3.5 for more details on the contextualization of VMs for particle physics applications.

Once the VM has completed its contextualization, it is registered as a HTCondor machine. The *HTC machine poller* periodically retrieves the ClassAds of all the HTCondor machines via the HTCondor Python API and stores its ClassAd in the MariaDB. If there is no ClassAd for the VM, then a query of the MariaDB would show the VM to be in an *unregistered* state. The registration of the VM with the HTCondor pool is part of the contextualization process and normally occurs within a short time. A VM that is in an unregistered state for a

Table 1 The possible states of a VM in the CSV2 distributed compute cloud system

VM state	Description
starting	VM is booting/contextualizing
unregistered	VM is running and has not registered in HTCondor pool
idle	VM is running, registered in HTCondor pool and not running jobs
running	VM is running, registered in HTCondor pool and running jobs
retiring	VM is running, retired in HTCondor pool and will complete running jobs
manual	VM is flagged as being manually used and will be ignored by the *VM Scheduler*
error	VM in error state according to the cloud information

long period (defined by a configurable "come alive" time) or fails to start any job within a short period after registering with HTCondor (defined by a configurable "job alive" time) is usually problematic and is terminated.

If a VM remains idle for an extended period of time after completing one or more jobs, then either the job queue is empty or there are no queued jobs that can run in the VM. In this situation, the *VM scheduler* sets the retire-flag in the database entry of the VM. We have observed that keeping *idle* VMs alive for 30-60 minutes is optimal (defined by a configurable "keep alive" time) due to the sporadic nature of the workload management systems of the experiments.

If the *retire-flag* of the VM in the MariaDB is set, then the *HTC machine poller* sends retire messages using the AMQP protocol until it sees the partitions on the VM in the *retiring* state. These messages are retrieved by the *HTC Agent*, who then issues a "condor_off" command directly to the VM. Then the HTCondor master daemon on the VM marks all the partitions to be "retiring" preventing them from accepting new jobs. However, the remaining jobs on the VM are allowed to complete.

When a VM is in a *retiring* state and it is not running any jobs, then the *HTC machine poller* sends a request (via the cloud API) to the cloud to shut down that VM instance.

A VM can also be retired if there is a change in the quota of a cloud (e.g. cores, RAM) and the current usage now exceeds the new quota. Depending on the type of change in the quota, then either the user interface or the cloud poller will set the retire flag on a collection of VMs (based on information such as the age of the VM) needed to rebalance the system.

3.3 Configuration

CSV2 introduces users and groups to the distributed compute cloud (stored in the MariaDB). Users can be assigned different access privileges and can be members of multiple groups. Groups are defined, in our case, to be particle physics experiments or a development and testing area[2]. Each CSV2 group adds the compute clouds to its configuration and users in the group have the ability to add and manage the clouds.

The VM images and SSH keys[3] can be uploaded by logging in directly to the cloud or via the CSV2 GUI. One can define a default VM image and default SSH keys that will be automatically distributed to all Openstack clouds (this will be extended to other clouds). CSV2 keeps track of the VM images in the MariaDB. As the list of VM images can change, the *cloud poller* periodically queries the clouds for the list of available VM images as well as VM flavours and cloud quotas to keep the MariaDB up to date.

VM instances are sized and instantiated using flavours (also known as "instance types" in EC2 parlance). Users within a group may also define default flavours on a cloud by cloud basis and/or a group default flavour. Where both group and cloud default flavours are defined, the cloud default flavour will take precedence over the group default flavour for any particular cloud. If a job specifies no flavour, and no default flavours are found, then CSV2 will select one that uses the least resources and satisfies the specified job requirements (cores, disk, and RAM).

The GUI is accessed through a web browser and can also generate time series plots for each variable (stored in an InfluxDB). An example of a time series plot is shown in Fig. 4 where the number of running jobs for ATLAS and Belle II is plotted for the month of July 2019. The time series plot shows an unusual situation where ATLAS had longer running production jobs and Belle II had very short running analysis jobs.

[2] Openstack has projects and users. Previously, Openstack used tenants before changing to projects. As CSV2 uses other types of clouds, it was decided to identify ensembles of users as groups to distinguish it from Openstack projects.

[3] The SSH keys allow the user to log into a VM instance for debugging purposes.

Group	Jobs	Idle	Running	Completed	Held	Other	Foreign
atlas	875	372	483	20	0	0	0
belle	31	4	18	0	9	0	0

Group	Clouds	VMs	Starting	Unreg.	Idle	Running	Retiring	Manual	Error	Native Cores Used	Native Cores Limit	RAM
atlas	arbutus	309	0	0	0	309	0	0	0	2472	2500	
atlas	cc-east	20	0	0	0	20	0	0	0	160	160	
atlas	chameleon	12	0	0	0	12	0	0	0	96	96	
atlas	otter	50	0	0	0	47	0	0	3	400	400	
belle	arbutus	3	0	1	0	2	0	0	0	12	2500	
belle	cc-east	0	0	0	0	0	0	0	0	0	100	
belle	desy	2	0	0	0	2	0	0	0	16	70	
belle	ecdf-b	0	0	0	0	0	0	0	0	0	64	
	Totals	396	0	1	0	392	0	0	3	3156	5890	

Fig. 3 Snapshot of the CSV2 GUI showing the number of jobs for the ATLAS and Belle II experiments in the upper table. The second table shows the different clouds running for the two experiments and the states of the VM instances. At the time of screenshot, the Belle II experiment had few jobs running and the resources were mainly used by ATLAS. Only part of the GUI is shown.

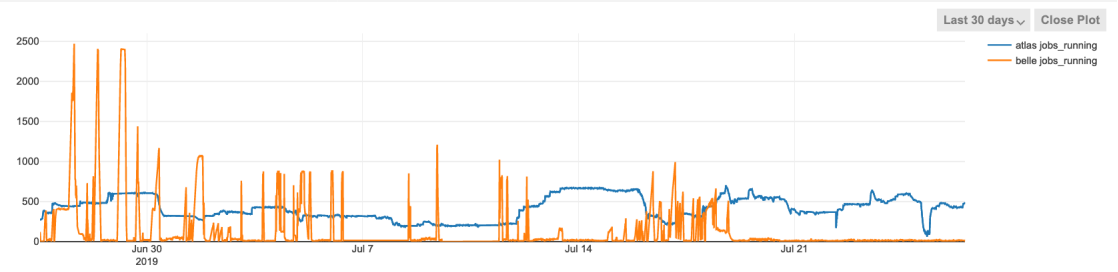

Fig. 4 Shows the number of running ATLAS jobs (blue) and Belle II jobs (orange). ATLAS was running production jobs requiring 4 or 8 cores whereas Belle II was periodically running very short analysis jobs during the 30-day period in 2019. The system responded to the varying workloads of the two projects.

There are other configuration settings for the management of VMs with default parameters (for example, the cycle times of the database pollers and VM scheduling algorithm parameters). We refer the reader to the documentation located at *cloudscheduler.readthedocs.io*[4].

3.4 VM Contextualization for particle physics applications

The contextualization of the VM instances for the ATLAS and Belle II experiments is very similar. Both use the micro-CernVM virtual machine image [20], the CernVM File System (CVMFS) [21], and optionally, *Shoal*, a dynamic web cache locator service [22].

The small micro-CernVM image (approximately 20 MB) saves storage space and can be started instantaneously. The image contains a read-only Linux kernel and a CernVM File System (CVMFS) client. An array of squid proxy caches around the world are used to host frequently used files from CVMFS [23]. The CVMFS client is configured to use a specific squid cache. We configure the CVMFS client to use the nearest squid cache to the VM by querying Shoal.

[4] The documentation is work in progress

The contextualization process then sets up the HTCondor client, and the environment for the experiment. The accounting and monitoring processes are activated, and the CPU benchmark suite is run on the VM (see the next section). The necessary host certificates and SSH keys are added to the VM.

It is important to keep track of the resources delivered by a cloud to an experiment. One option is to record the information as the VM is terminated, however, in some cases the termination is immediate without any opportunity to publish any information. We configure the VMs to write out their status every 15 minutes to Elasticsearch. The status includes configuration information, the fast benchmark and the timing information (obtained from /proc/stat). The fast benchmark is discussed in the following section.

3.5 Operations and status

The CSV2 distributed compute cloud started production use in early 2019. The ATLAS production using the clouds in North America was shifted to CSV2, and tested extensively for many months. The Belle II production system was transitioned to CSV2 in June 2019. ATLAS and Belle!II are now supported by a single instance of CSV2 but continue to use separate instances of HTCondor that are linked to their respective workload management systems.

In Fig. 5 we plot a histogram of the CPU hours weighted by a benchmark that is an estimate of the HEP-SPEC06 [24] benchmark. A centre usually runs the HEP-SPEC06 benchmark when their resources are being commissioned. As we are generally unaware of the underlying hardware as well as unaware of other processes running on the underlying hypervisors at the time jobs run, we can not use a static benchmark value. It is also impractical to run the HEP-SPEC06 on every VM, as it takes 2-3 hours (and there are also licensing issues). Instead, a fast-benchmark code (DB16) [25] is executed at the end of the contextualization of the VM instance to measure the performance of the dynamically provided resources. Fig. 6 shows a histogram (normalized to unit area) of the fast-benchmark measured for every VM booted in the first six months of 2019. The fast-benchmark gives a reproducible result if the VM is the only activity on the underlying node; however, the value is often underestimated if the node is busy or hyperthreading is used. The fast-benchmark is not ideal and there are ongoing efforts to find a more reliable estimate of the experiments' workload.

An important new feature of CSV2 is its ability to manage the jobs and clouds of multiple groups (experiments). This makes it possible for one experiment to opportunistically utilize the resources of another experiment whose resources are idle. CSV2 can be configured so that the opportunistic usage can be some fraction or all of the idle resources. If the resources are required, then the opportunistic usage is automatically scaled down by retiring those resources. The running jobs are allowed to complete so that the transition back to the fair share can take a number of hours. This feature has significantly improved the utilization of our resources.

In addition, we remark that the distributed compute cloud is using the Dynafed data federation service, developed by CERN IT, for locating input data [26,27]. Dynafed federates existing storage systems and provides a link to the closest copy of the data on the basis of GeoIP information. Dynafed is used to federate existing storage of the ATLAS and Belle II experiments [28,29].

4 Summary

We have presented the design of a distributed cloud computing system based on the Cloudscheduler VM provisioning platform. It remains a novel system for utilizing high-throughput workloads on multiple dedicated and opportunistic clouds located at remote locations and owned by other organizations. The system has provided significant resources to the ATLAS and Belle II particle physics experiments. The Cloudscheduler VM provisioning system has been updated to current software standards to make it more scalable and reliable, as well as adding new functionality. It is currently planned to use Cloudscheduler for the computing part of the proposed Belle II Canadian Raw Data Centre. Other plans include further integration and use of the Dynafed data federator to expand the running of data-intensive applications.

CSV2 is stored in GitHub (*https://github.com/hep-gc/cloudscheduler*) and documentation is found at *cloudscheduler.readthedocs.io*.

Acknowledgements We would like to thank the support of the Natural Sciences and Engineering Research Council of Canada and the Canadian Foundation of Innovation. In addition, the resources and generous support provided by Compute Canada, the Chameleon Project, the Edinburgh Compute and Data Facility (ECDF), Amazon and Microsoft are acknowledged.

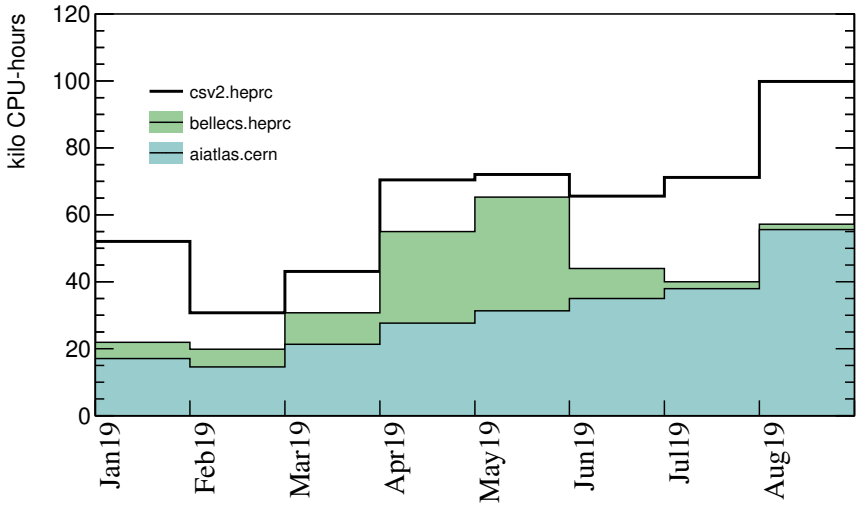

Fig. 5 The delivered fast-benchmark hours (x1000) per month in 2019. The blue histogram (aiatlas.cern) is the ATLAS HTCondor instance cloud hosted in CERN. The white histogram (csv2.heprc) and the green histogram (bellecs.heprc) are the ATLAS and Belle II HTCondor instances hosted in Victoria, Canada.

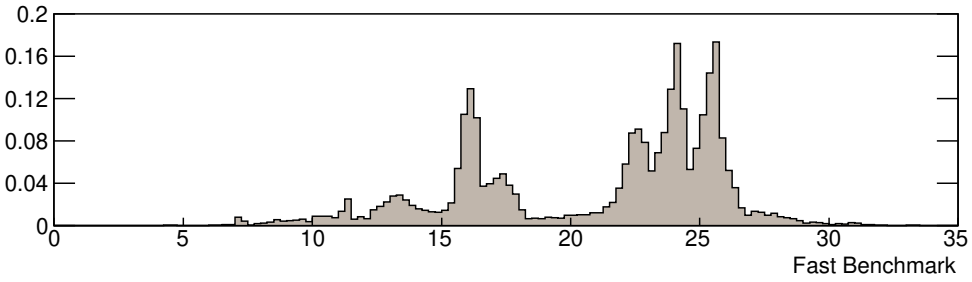

Fig. 6 Histogram of the fast benchmark measured by the VMs. Note that the histogram is normalized to unit area. The fast-benchmark is an estimate of the HEP-SPEC06 benchmark. There are entries from multiple clouds. The types of processors in each cloud are unknown. When running on whole real CPUs the benchmark values show a narrow peak, while when using hyperthreaded CPUs the peaks are wider and shifted to lower values. The continuum entries between the peaks represent measured benchmark entries when other (unknown) processes are running on the same hypervisor.

Conflict of interest

The authors declare that they have no conflict of interest.

References

1. M. Berman, Research Computing in the Cloud: Leveling the Playing Field, EDUCAUSE Review 54, no. 1, 2019
2. M. Gasthuber *et. al.*, HNSciCloud - Overview and technical Challenges, J. Phys.: Conf. Ser. 898 052040 (2017), doi:10.1088/1742-6596/898/5/052040
3. S. Fuess *et. al.*, The HEPCloud Facility: elastic computing for high energy physics – The NOvA Use Case, J. Phys.: Conf. Ser. 898 052014 (2017), doi:10.1088/1742-6596/898/5/052040

4. P. Armstrong *et. al.*, Cloud Scheduler: a resource manager for distributed compute clouds, arXiv preprint arXiv:1007.0050 (2010)
5. K. Keahey, M. Tsugawa, A. Matsunaga, and J. Fortes, Sky computing, Internet Computing, IEEE, 13(5):43–51, 2009, doi:10.1109/MIC.2009.94
6. R.P. Taylor *et. al.*, Consolidation of cloud computing in ATLAS, J. Phys.: Conf. Ser. 898 052008 (2017), doi:10.1088/1742-6596/898/5/052008
7. R.J. Sobie *et. al.*, Utilizing clouds for Belle II J. Phys.: Conf. Ser. 664 022037 (2015), doi:10.1088/1742-6596/664/2/022037
8. A. McNab *et. al.*, Managing virtual machines with Vac and Vcycle, J. Phys.: Conf. Ser. 664 022031 (2015), doi:10.1088/1742-6596/664/2/022031
9. R. Grzymkowski *et. al.*, Belle II public and private cloud management in VMDIRAC, J. Phys.: Conf. Ser. 664 022021 (2015), doi:10.1088/1742-6596/664/2/022021
10. B Bockelman *et. al.*, Interfacing HTCondor-CE with OpenStack, J. Phys.: Conf. Ser. 898 092021 (2017), doi:10.1088/1742-6596/898/9/092021
11. K. Fransham *et. al.*, Research computing in a distributed cloud environment, J. Phys.: Conf. Ser. 256 (2010) 012003, doi:10.1088/1742-6596/256/1/012003
12. I. Gable *et. al.*, A batch system for HEP applications in a distributed IaaS cloud, J. Phys.: Conf. Ser. 331 062110 (2011), doi:10.1088/1742-6596/331/6/062010
13. D. Thain, T. Tannenbaum and M. Livny, Distributed computing in practice: the Condor experience, Concurrency Computat.: Pract. Exper.2005;17:323 (2005), doi:10.1002/cpe.938
14. The Worldwide LHC Computing Grid, http://wlcg.web.cern.ch
15. F.H. Barreiro Megino *et. al.*, PanDA for ATLAS distributed computing in the next decade J. Phys.: Conf. Ser. 898 052002 (2017), doi:10.1088/1742-6596/898/5/052002
16. A Tsaregorodtsev *et. al.*, DIRAC Distributed Computing Services, J. Phys.: Conf. Ser. 513 032096 (2014), doi:10.1088/1742-6596/513/3/032096
17. MariaDB open-source relational database, https://mariadb.org
18. K. Keahey and T. Freeman, Contextualization: Providing One-Click Virtual Clusters, IEEE Xplore: 06 January 2009, DOI: 10.1109/eScience.2008.82
19. Cloud_init: the standard for customizing cloud instances. https://cloud-init.io
20. J. Blomer *et. al.*, Micro-CernVM: slashing the cost of building and deploying virtual machines, J. Phys.: Conf. Ser. 513 (2014) 032009, doi:10.1088/1742-6596/513/3/032009
21. J. Blomer *et. al.*, New directions in the CernVM file system, J. Phys.: Conf. Ser. 898 062031 (2017), doi:10.1088/1742-6596/898/6/062031
22. I. Gable *et. al.*, Dynamic web cache publishing for IaaS clouds using Shoal, J. Phys.: Conf. Ser. 513 032035 (2014), doi:10.1088/1742-6596/513/3/032035
23. Squid is a caching proxy for the Web. http://www.squid-cache.org.
24. M. Michelotto *et. al.*, A Comparison of HEP code with SPEC1 benchmarks on multi-core worker nodes, J. Phys.: Conf. Ser. 219 (2010) 052009, doi:10.1088/1742-6596/219/5/052009
25. P. Charpentier, Benchmarking worker nodes using LHCb productions and comparing with HEPSpec06, J. Phys.: Conf. Ser. 898 082011, doi:10.1088/1742-6596/898/8/082011
26. Dynafed - The Dynamic Federation project. http://lcgdm.web.cern.ch/dynafed-dynamic-federation-project
27. F. Furano, O. Keeble and L. Field, Dynamic federation of grid and cloud storage, Phys. Part. Nuclei Lett. (2016) 13: 629. https://doi.org/10.1134/S1547477116050186
28. F. Berghaus *et. al.*, Federating distributed storage for clouds in ATLAS, J. Phys.: Conf. Ser. 1085 032027 (2018). doi:10.1088/1742-6596/1085/3/032027
29. M. Ebert *et. al.*, Using a dynamic data federation for running Belle-II simulation applications in a distributed cloud environment, EPJ Web of Conferences 214, 04026 (2019). https://doi.org/10.1051/epjconf/201921404026