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

In the field of particle physics, clouds are increasingly used to process and analyze data [1]. It is possible that commercial clouds might satisfy the computational requirements of the next generation of global research projects; however, it is likely that dedicated storage facilities will be required to store and preserve the research data. One scenario may be a hybrid solution of dedicated and opportunistic, private and commercial compute resources, linked by high-speed networks to a distributed set of dedicated storage repositories.

This paper describes such a hybrid 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 user. The distributed cloud can be integrated with existing storage systems or federations to provide full access to the research data. Although the focus of the distributed cloud is to deliver resources for particle physics application, it is also used for astronomy and can be used by researchers in other fields.

The distributed compute cloud uses cloudscheduler for VM provisioning and scheduling, and HTCondor for job scheduling. The two packages are designed for a dynamic environment where the resources on a cloud, or even the cloud itself, appear or disappear. 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 [2] and is based on ideas discussed in a paper describing "sky computing" [3]. The distributed compute cloud has run many millions of jobs on three continents for the ATLAS experiment at the CERN Laboratory in Geneva, Switzerland [4] and the Belle II experiment at the KEK Laboratory in Tsukuba, Japan [5].

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 (see section 2.3). 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.

There are other strategies and software for utilizing clouds in particle physics. These include Vcycle, VMDIRAC and an extension of the HTCondor job scheduler. In Vcycle, the resource provider creates VMs for the experiments that draw jobs from the experiments' central queue of tasks [6]. VMDIRAC is a module for the DIRAC workload management system [7] that can start VMs on clouds [8,9]. HTCondor can also be used to start VMs on Openstack clouds [10]. The distributed cloud computing system using cloudscheduler provides an infrastructure that can run workloads for any field or research without the need for application experts at each site. It is not dependent on project-specific workload management systems but can be integrated with them. Further, it can run workloads on all cloud types while the users or experiments only need to know about the familiar batch system interface to which they submit jobs.

In this paper, we give an overview of the original design of the distributed cloud system using Cloudscheduler Version 1 (CSV1) and motivation for a new version in Section 2. Section 3 describes Cloudscheduler Version 2 (CSV2). Throughout the paper, we highlight the external components used in both versions of the distributed compute cloud that are critical to the system.

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. 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. Multi-core VMs are booted on the clouds, whether a job requires one or

all of the cores in the VM, as there are benefits such as shared disk caches, fewer VM instances and it helps reduce the fragmentation of the resources on the clouds. The HTCondor client on the VM starts a partitionable slot during the contextualization process, which 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 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. CSV1 only requests one VM boot per cycle every minute (both configurable parameters) on one cloud in the list; on the subsequent cycle, it would request a VM on the next cloud in the list. The VM image and VM flavour are specified in the JDL file or in a configuration file. Once the VM is booted and contextualized, it joins the HTCondor pool and queued job(s) can start on this VM.

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 pickle 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 retire request issues the "condor_off" command to both the HTCondor startd and master daemons on the VM instance. The HTCondor slots on this machine are then listed as being in a "retired" state. 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). The Linux root user can configure the system, add or remove clouds and edit the files used for the contextualization of a VM. The normal Linux 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 there is a monitoring web page.

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 use the Openstack software, though Open Nebula, Amazon EC2, Google GCE and Microsoft Azure clouds have also been used.

We ran two instances of the CSV1 system in North America for ATLAS and Belle II, respectively, and another CSV1 in Europe for ATLAS (each with a separate HTCondor instance). The HTCondor instances dedicated to the ATLAS experiment are linked to the PanDA workload management system [15]. 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 "pilot jobs" that sets up the software environment and contacts the central system for payload jobs. A pilot job can run more than one payload job depending on its configuration.

The Belle II experiment uses the DIRAC workload management system [7]. There are three DIRAC Site Directors in Victoria that submit pilot jobs to the Belle II HTCondor instance if there are 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.

2.3 Motivation for cloudscheduler redevelopment

Cloudscheduler has evolved with the cloud technologies but it has become difficult to maintain. The CSV1 code is written in Python 2, which will no longer be supported after January 2020. CSV1 contains redundant code written to support operating system features and cloud software that no longer exist. CSV1 was designed as a multi-user system; however, it never functioned well as a mutli-user system as it would rapidly start and stop VMs in attempt to balance resources between competing projects.

CSV1 runs on a single node and uses in-memory data structures for the state information; making it prone to single errors and failure. An unscheduled outage or system crash would result in a loss of all running VMs

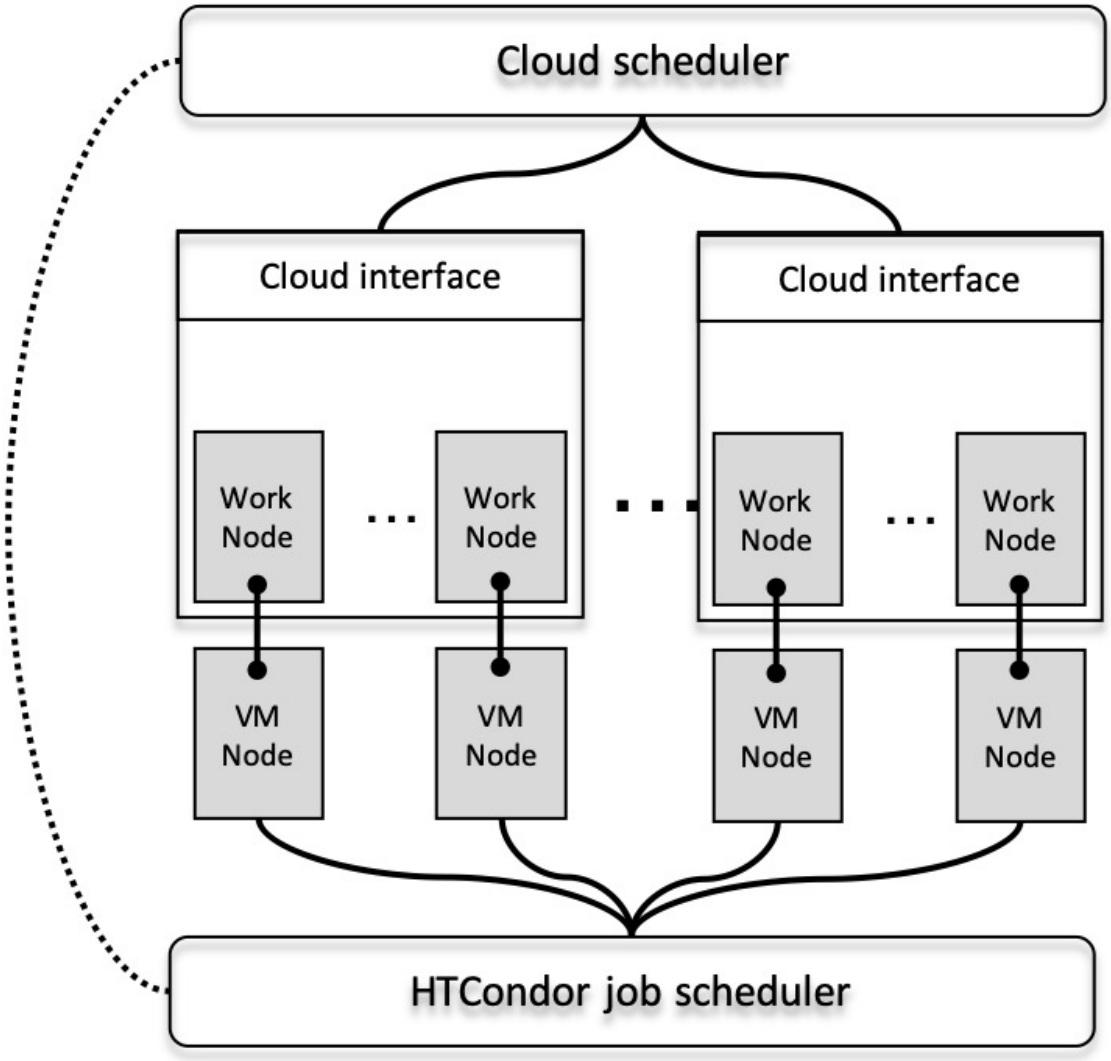

Fig. 1 Overview of the cloudscheduler Version 1 and HTCondor distributed compute cloud. A user or workload management system submits application jobs to the HTCondor job scheduler. Cloudscheduler reviews the job queue and the resources on the compute clouds. If there is a cloud with resources that meet 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.

and require a manual intervention on each cloud to remove the residual VMs (a time consuming and error prone task).

As mentioned in the previous section, there were separate instances of CSV1 for the experiments. While the separate CSV1 instances worked well, it meant one could not shift idle resources between projects. A single instance of cloudscheduler would also simplify the operations though it would need to be able to use one or more instance of HTCondor. The ATLAS and Belle II experiments use different authentication in HTCondor, and some sites do not allow their VMs to connect to a remote HTCondor pool.

It became apparent that a new version was necessary to update the software, simplify the operations, extend the current capabilities and add new functionality.

3 Cloudscheduler V2 distributed compute cloud

3.1 Overview

CSV2 is a framework of component processes written in Python 3 and built around the MariaDB (SQL) relational database [16]. There are processes for control, data gathering (pollers), user interfaces and for VM provisioning. The MariaDB ensures consistent state information, which is essential for the reliability of CSV2 (see Fig. 2).

The database allows data retrieval from multiple sources, and correlates information between different subsystems to obtain different views of the system data. The data is organized in tables, with rows representing objects and columns representing the attributes of an object. There are tables for the global and local configuration parameters. CSV2 also creates ephemeral data in tables for objects like jobs or virtual machines. The database also adds authorization features that protects the integrity of the data while allowing both local and remote access.

The database also ensures recoverability (assuming a regular database backup strategy). Should a catastrophic system loss occur (e.g. through hardware failure), restoration of the configuration data from the latest database backup will allow CSV2 to restart and return to full operation. Within one cycle, the data pollers (described below) will retrieve information from all configured subsystems (clouds and HTCondor job schedulers), and once the ephemeral data is updated, the database will accurately reflect the current state.

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 (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 contrast to CSV1, the CLI can be used from any computer and the user of the CLI does not need to have ssh access to or a Linux account on the machine where CSV2 runs. For all clients, 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. New entries are added to tables in the database for each new job and VM. The state information of existing entries are updated by the *pollers*, and obsolete information (for completed jobs/VMs) is removed.

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. The pollers consist of multiple tasks that run at different frequencies. For example, the *cloud poller* updates the MariaDB with information on the VMs every 60 seconds whereas it updates the list of VM images every 5 minutes. Typically, the dynamic information is updated every minute and the relatively static information on a longer timescale (between 5 minutes and one day)[2].

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).

CSV2 requires inter-process communication and signalling. For example, the User Interface (UI) signals the cloud poller when a cloud is added, so that cloud data is updated immediately. CSV2 uses AMQP, a reliable message delivery protocol, to signal processes to wake when there is important information that will impact the outcome. For example, the *cloud poller* looks for new clouds or VM images every 5 minutes. The AMQP message from the UI, signals the *cloud poller* to start a new cycle. This solution is better than relying on the pollers running on an overly frequent cycle. It is expected that AMQP will have an expanded role in further updates of the system.

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

[2] Note that the numbers given for cycle times, limits and thresholds are configurable parameters. Further, when we state that a poller or process is run every N seconds, this means that the process sleeps for N seconds after the last cycle before restarting its tasks.

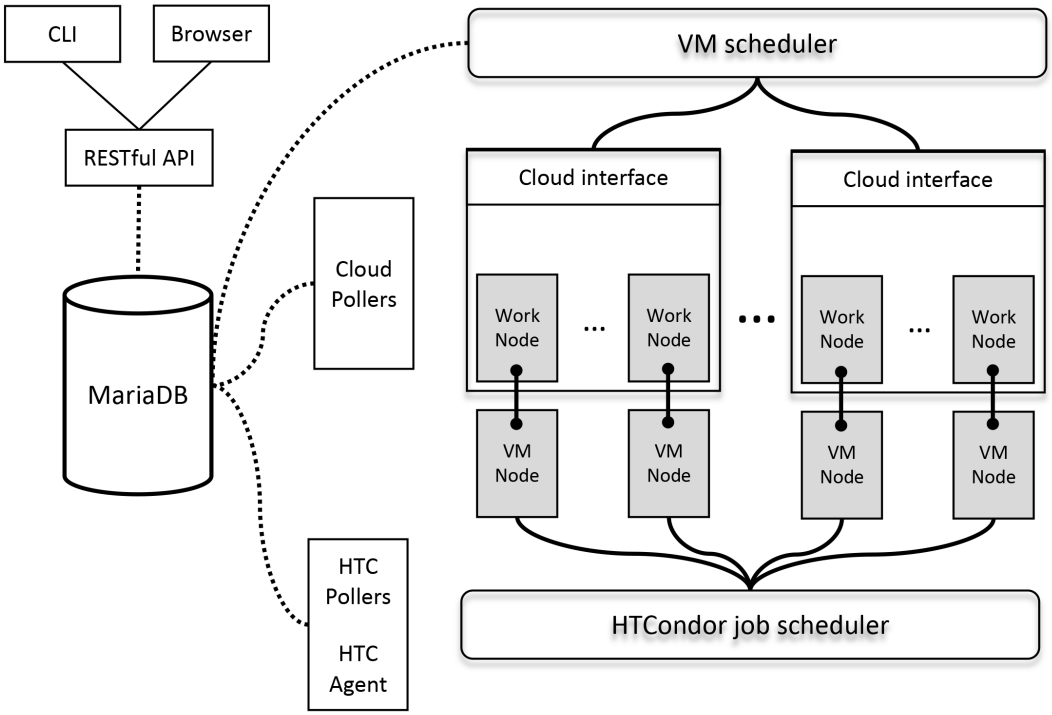

Fig. 2 Overview of the cloudscheduler Version 2 and HTCondor distributed compute cloud. The primary change is the introduction of the MariaDB for keeping track of the state of the system and "pollers" to gather information from the clouds and the job scheduler. The information in the database is used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

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

3.2 Workflow

The VM Scheduler runs every 10 seconds and retrieves virtual tables (views) from the MariaDB to determine its actions. A view can retrieve and relate data from multiple tables, perform calculations, sort, group, select, concatenate, resort and regroup to derive the information required. For example, one of views determines the state of the VMs (see Table 1). The primary tables used by the scheduler are the job queue and the compute resources.

We briefly describe the steps to create the job queue view. Jobs are grouped according to their resource requirements such as cores, disk, and memory. For example, ATLAS has jobs requiring either 1 or 8 cores, and Belle II has jobs that require clouds with a local Belle II storage repository. Jobs of similar requirements are grouped and counted (called a "job-group").

Next, the view finds the VM flavours for each cloud that can satisfy the job requirements of the job-group. For each cloud, the view assembles a list of (Cloud name:VM flavour) pairs with only one pair per cloud. The VM flavour is specified by the user or retrieved from the database. If the VM flavour is obtained from the database, then the one with the fewest cores and/or the least amount of memory is selected. If a cloud has no

VM flavour that meets the requirements, then that cloud is not included in the list of VM flavors (and will not be selected to boot VMs for this job-group).

The result of the view is the table shown below, listing the group name, the number of idle jobs, and a list of (Cloud name:VM flavour) for each unique job type (only a subset of the columns in the table is shown). Each row corresponds to a unique job type. The first row is for ATLAS 8-core jobs, the second row is for Belle II production (low I/O) jobs and the third row is for Belle II analysis (high I/O) jobs. The latter group requires a local Belle II storage element, which is set to run on different set of clouds.

Group	Idle jobs	Cloud name : VM flavour
atlas	86	arbutus-nf:c16-60gb-392, arbutus:c8-30gb-186, cc-east:c8-30gb-430, chameleon:m1.xxlarge, otter:b8
belle	1	arbutus:c8-30gb-186, cc-east:c4-15gb-205, desy:m1.xlarge, ecdf-b:m1.xlarge, otter:b8
belle	91	arbutus:c8-30gb-186, otter:b8

The list of (Cloud name:VM flavour) pairs in the above table is ordered by the priority of the cloud (called the cloud-priority). The cloud-priority can, for example, give lower priority to commercial clouds. Clouds that have the same priority are ordered alphabetically based on their cloud name.

Prior to reviewing whether the clouds in the job-group have free resources to boot new VMs, the system is checked for VMs in a *starting* or *unregistered* states (see Table 1), or if there are *running* VMs that are not fully utilized (e.g. empty or idle job slots). This indicates there is capacity in the system for the idle jobs and no new VMs will be booted.

The compute resources view is used to determine how many VMs with a specific flavour can be started on one cloud. The following table shows an example of a view that returns the group name, the (Cloud name:VM flavour) pair, and the number of Available VM slots that can be started on this cloud. In this example, the arbutus cloud has the capacity to start 109 VMs with the c8-30gb-186 flavour for ATLAS and 184 VMs with the same flavour for Belle II. The table was selected to show only the results of a query for a VM flavour of arbutus:c8-30gb-186, which is the optimal VM flavour for both projects (an unrestricted query returns over 100 rows).

Group	Cloud name : VM flavour	Available VM slots
atlas	arbutus:c8-30gb-186	109
belle	arbutus:c8-30gb-186	184

CSV2 makes it possible for one experiment to opportunistically utilize the resources of another experiment whose resources are idle. The number of available VM slots for each cloud and VM flavour (as shown in the above table) takes into account the opportunistic resources. In the simplest configuration, the primary group (experiment) has the priority use of a cloud, and the opportunistic group can also use that cloud if there are idle resources. The primary group has a quota of compute cores equal to the full allocation on the cloud, and the opportunistic group has an opportunistic quota of compute cores that is less than the full allocation. If the primary group does not use any resources, the opportunistic group can boot VMs up to their opportunistic quota. If the primary group begins submitting jobs, then the VMs of the opportunistic group will be retired and their jobs allowed to complete. Eventually there will be no jobs from the opportunistic group. The quotas of the primary and opportunistic groups are used when determining the compute resources view. This strategy has significantly improved the utilization of the clouds.

The compute resources view provides the *VM scheduler* with the information it requires to boot new VMs. Prior to issuing VM boot requests, a number of conditions are tested to avoid starting too many VMs. For example, if there are five or more VMs in a *starting* or *unregistered* state on a cloud, then no VMs will be started on that cloud for any group. In addition, if there are more that ten *idle* VMs within the clouds associated to a unique job type, then no VMs are started for that job type group.

If the above criteria are satisfiedc and the list of (Cloud name:VM flavour) pairs in the compute-view table shows available VM slots (see above table), then the *VM scheduler* will issue up to five VM boot requests on each cloud in the group.

At the end of the boot process, the VMs are contextualized with the configuration required by CSV2 (e.g. HTCondor), the particle physics software (e.g. CVMFS file system) and the experiment software. We discuss this in further detail in 3.4.

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 happens during

the start of the HTCondor clients on the VM after it is fully contextualized. A VM that is in an *unregistered* state that exceeds a "come alive" time (currently 2400 seconds) or fails to start any job within a "job alive" time (currently 300 seconds) 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, there are no more queued jobs that can run in the VM, or the VM developed problems communicating with the HTCondor server. 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 "keep alive" time) due to the sporadic nature of the workload management systems of the experiments. Both the ATLAS and Belle II workload management systems keep track of the idle or queued and running jobs, and use those numbers to submit additional jobs.

If the retire-flag of the VM in the MariaDB is set, then the *HTC machine poller* sends retire messages via AMQP to the *HTC Agent*. On receiving a retire message, the *HTC Agent* issues a "condor_off" command to both the HTCondor startd and master daemons on the VM instance, causing all the partitions to be marked "retiring" and preventing them from accepting new jobs. However, jobs currently running 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[3]. 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[4] can be uploaded by logging in directly to the cloud (e.g. via the Openstack or Amazon dashboards) 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. 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.

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. For most of 2019, ATLAS provided a sustained workload of longer running production jobs (typically 6-12 hours) while Belle II has periodic bursts of very short running analysis jobs (5-10 minutes). The time series plot highlights the difference in workload between the two projects for June-July 2019. Belle II is still at an early stage of its project lifetime and just starting to record particle collision data; it is expected that the number of Belle II jobs will increase in the future.

There are other configuration settings for the management of VMs with default parameters (for example, the cycle times of the pollers and VM scheduling algorithm parameters). We refer the reader to the documentation located at *cloudscheduler.readthedocs.io*[5].

3.4 VM Contextualization for particle physics applications

At the end of the boot process, the VMs are contextualized with the required software and the VM is configured for the application software [17]. The contextualization is controlled by metadata that is passed to the "cloud-init" process [18].

[3] 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.

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

[5] The documentation is work in progress

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.

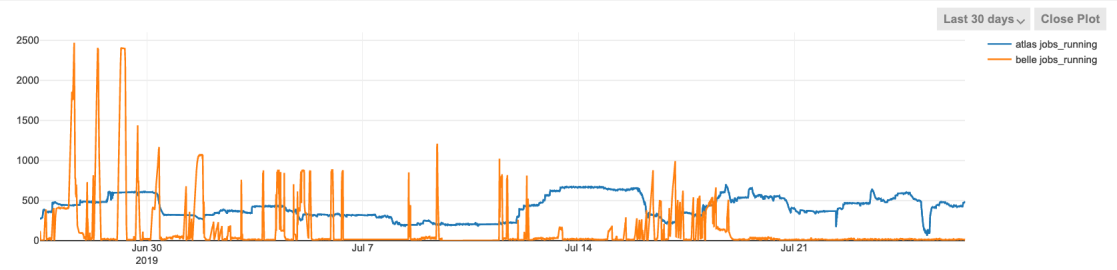

Fig. 4 Shows the number of running ATLAS jobs (blue) and Belle II jobs (orange). ATLAS was running production jobs requiring 4 or 8 cores whereas Belle II was periodically running very short analysis jobs during the 30-day period in 2019.

The contextualization of the VM instances for the ATLAS and Belle II experiments is very similar. Both use the micro-CernVM virtual machine image [19], the CernVM File System (CVMFS) [20], and optionally, *Shoal*, a dynamic web cache locator service [21].

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 files from CVMFS [22]. The CVMFS client is configured to use a specific squid cache; however, the CVMFS client can be configured to use the nearest squid cache to the VM by querying Shoal.

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). Both the HTCondor client and the CPU benchmark are retrieved from CVMFS. The necessary host certificates and SSH keys are added to the VM. All the VMs share the same host certificate; we currently use certificates from GridCanada.

It is important to keep track of the resources delivered by a cloud to an experiment. 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). We are currently working on reporting our accounting numbers back into APEL for ATLAS (the accounting information for Belle II will be added once the reporting system is operational) [23]. The fast benchmark is discussed in the following section.

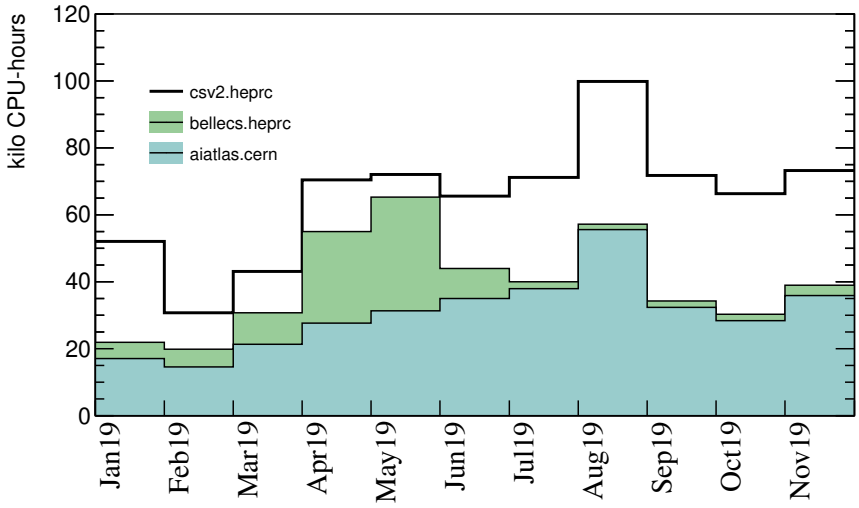

Fig. 5 The fast-benchmark hours (x1000) per month in 2019. The (processor) hours are extracted from the kernel activity in /proc/stat and is typically 80-90% of the wall clock time. The blue histogram (aiatlas.cern) is the ATLAS HTCondor instance cloud hosted at CERN. The white histogram (csv2.heprc) and the green histogram (bellecs.heprc) are the ATLAS and Belle II HTCondor instances hosted in Victoria, Canada.

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, 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 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 (see the recent presentation of the HEPiX Benchmark Working Group [26]).

In addition, we remark that the distributed compute cloud is using the Dynafed data federation service, developed by CERN IT, for locating input data [27,28]. 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 [29,30,31].

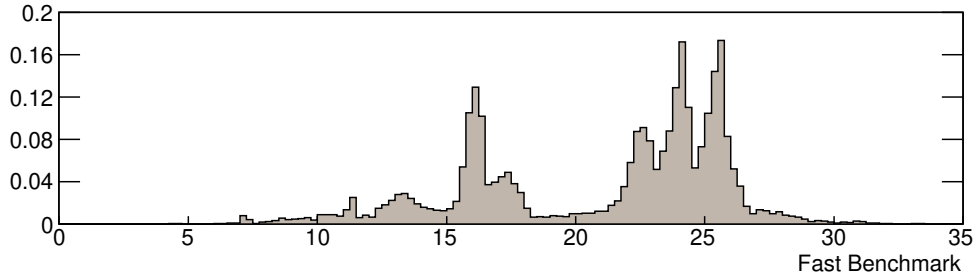

Fig. 6 Histogram of the fast benchmark measured by the VMs. Note that the histogram is normalized to unit area. The fast-benchmark is an estimate of the HEP-SPEC06 benchmark. There are entries from multiple clouds. The types of processors in each cloud are unknown. When running on whole real CPUs the 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.

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.

The CSV2 code repository is hosted at 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.

Conflict of interest

The authors declare that they have no conflict of interest.

References

1. E. Elsen, The end of computing's steam age, `https://home.cern/news/opinion/computing/end-computings-steam-age`
2. P. Armstrong *et. al.*, Cloud Scheduler: a resource manager for distributed compute clouds, arXiv preprint arXiv:1007.0050 (2010)
3. 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
4. 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
5. 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
6. 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
7. A Tsaregorodtsev *et. al.*, DIRAC Distributed Computing Services, J. Phys.: Conf. Ser. 513 032096 (2014), doi:10.1088/1742-6596/513/3/032096
8. 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
9. A. Amoroso *et. al.*, A modular (almost) automatic set-up for elastic multi-tenants cloud (micro)infrastructures, J. Phys.: Conf. Ser. 898 (2017) 082031, doi:10.1088/1742-6596/898/8/082031
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. MariaDB open-source relational database, https://mariadb.org
17. K. Keahey and T. Freeman, Contextualization: Providing One-Click Virtual Clusters, IEEE Xplore: 06 January 2009, DOI: 10.1109/eScience.2008.82
18. Cloud_init: the standard for customizing cloud instances. https://cloud-init.io
19. 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
20. 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
21. 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
22. Squid is a caching proxy for the Web. http://www.squid-cache.org.
23. APEL is an accounting tool that collects accounting data from sites participating in the EGI and WLCG infrastructures, https://wiki.egi.eu/wiki/APEL
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. A. Valassi *et. al.*, Benchmarking WLCG resources using HEP experiment workloads, to be published in the proceedings of the Computing in High Energy Physics Conference, Adelaide, 2019.
27. Dynafed - The Dynamic Federation project. http://lcgdm.web.cern.ch/dynafed-dynamic-federation-project
28. 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
29. 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
30. 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
31. F. Berghaus *et. al.*, The Dynafed data federator as grid site storage element, to be published in the proceedings of the Computing in High Energy Physics Conference, Adelaide, 2019.