

1 **High-throughput cloud computing with the**  
2 **Cloudscheduler VM provisioning service**

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

6  
7 Received: date / Accepted: date

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

17 **Keywords** particle physics · cloud computing

## 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.

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 used whether a single 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

69 fragmentation of the resources on the clouds. The HTCondor client on the VM starts a partitionable slot during  
70 the contextualization process that is subdivided depending on the resource requirements of the job.

71 CSV1 examines the list of pending jobs in the HTCondor queue and searches for a cloud with free resources  
72 that meets the job requirements as specified in the JDL file or as specified by the system-wide job defaults. If  
73 there is a cloud that meets those requirements, then CSV1 sends a request to the cloud to boot a VM. CSV1  
74 only requests one VM boot per cycle (typically every minute) on one cloud in the list and on the subsequent  
75 cycle, it would request a VM on the next cloud in the list. The VM image can be optionally specified in the JDL  
76 file. Once the VM is booted and contextualized, it joins the HTCondor pool and queued job(s) can start on this  
77 VM.

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

82 CSV1 can request the start, retirement or immediate destruction of a VM. The start and destroy commands  
83 are directly issued via the cloud API. If there are no jobs in the HTCondor queue, then a running VM may  
84 be retired. A retire request is issued to the HTCondor client on the VM and the client deregisters from the  
85 HTCondor pool. The retire request issues two “condor\_off” commands to the HTCondor startd and master  
86 daemons on the VM instance. The HTCondor slots on this machine are then listed as being in a “retired” state.  
87 The jobs running on the VM are allowed to finish and then the VM is destroyed.

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

## 95 2.2 Performance and status

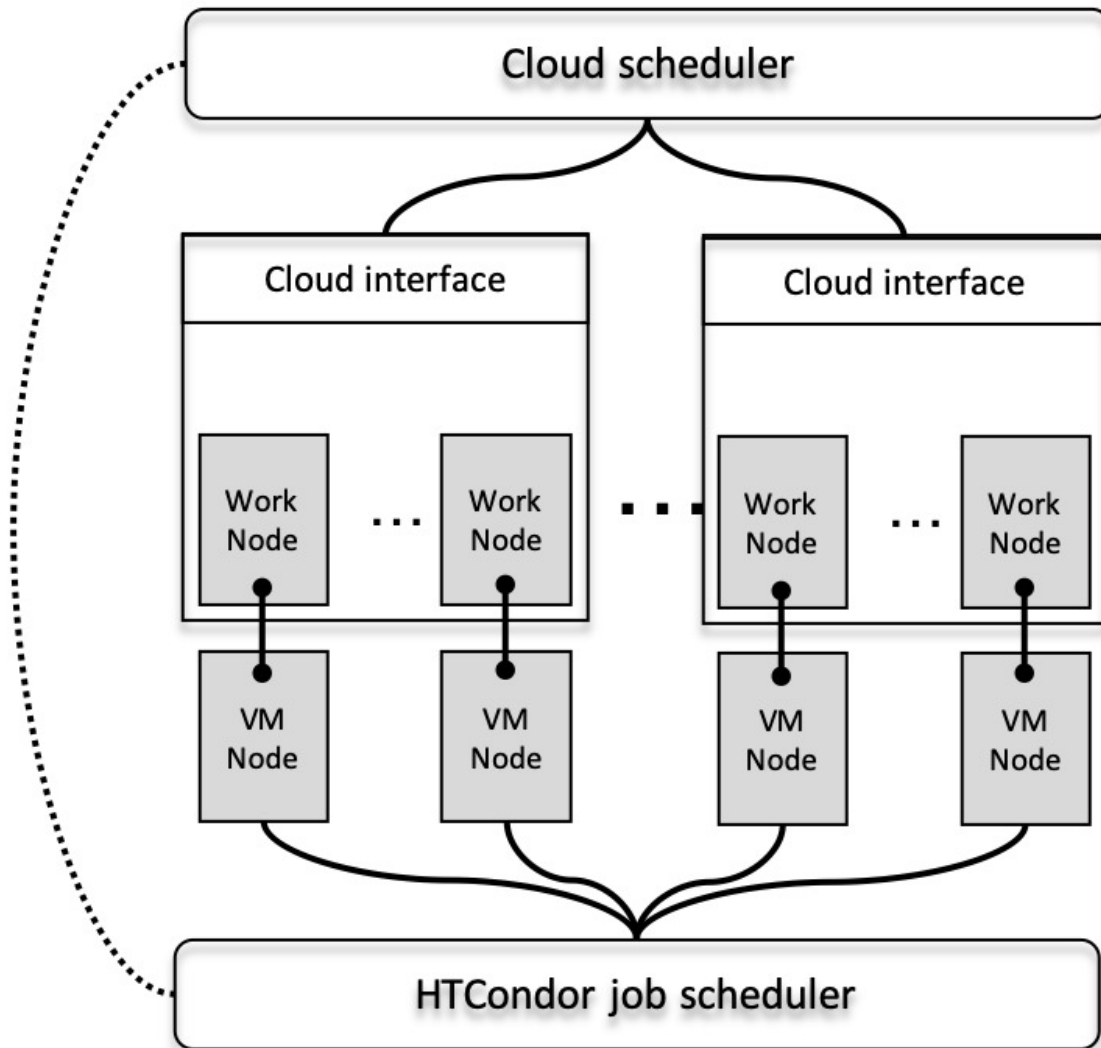
96 The distributed compute cloud is integrated into the WLCG grid infrastructure [14] and has run production  
97 workloads for many years using clouds in North America, Europe and Australia. The majority of clouds used  
98 the Openstack software, though Open Nebula, Amazon EC2, Google GCE and Microsoft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

100 We ran two instances of the CSV1 system in North America for ATLAS and Belle II, respectively, and  
101 another CSV1 in Europe for ATLAS (each with a separate HTCondor instance). The HTCondor instance ded-  
102 icated to the ATLAS experiment is linked to the PanDA workload management system [15]. The production  
103 team need not be aware that the resources associated with the HTCondor instance use multiple cloud resources.  
104 ATLAS typically uses a few hundred thousand cores at a given moment, and the distributed compute cloud  
105 contributes approximately 1% of the resources, comparable to the other Tier-2 compute centres operated in  
106 Canada. ATLAS (and Belle II) submit *pilot jobs* that sets up the software environment and contacts the central  
107 system for payload jobs. A pilot job can run more than one payload job depending on its configuration.

108 The Belle II experiment uses the DIRAC workload management system [7]. We operate three DIRAC  
109 Site Directors in Victoria that submit pilot jobs to the HTCondor instance if there are queued jobs in the central  
110 DIRAC server at KEK. The distributed compute cloud has provided 13% of the total workload of the experiment  
111 since January 2018. All of the Canadian resources for Belle II will continue to be provided by the distributed  
112 compute cloud.

## 113 2.3 Motivation for cloudscheduler redevelopment

114 Cloudscheduler was initially developed in 2009 and has evolved with the cloud technologies. The CSV1 code  
115 is written in Python 2, which will no longer be supported after January 2020. The multi-threaded module of  
116 CSV1 (a large and complex script) was becoming difficult to maintain. CSV1 also contained redundant code  
117 written to support operating system features and cloud software that no longer exist. CSV1 was designed as a  
118 multi-user system; however, it never functioned well as a mutli-user system. CSV1 would rapidly start and stop  
119 VMs in attempt to balance resources between competing projects.



**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.

120 CSV1 runs on a single node and uses in-memory data structures for the state information; making it prone  
 121 to single errors and failure. An unscheduled outage or system crash would result in a loss of all running VMs  
 122 and require a manual intervention on each cloud to remove the residual VMs (a time consuming and error prone  
 123 task).

124 CSV1 used separate instances for ATLAS and Belle II jobs in North America, and another for ATLAS jobs  
 125 in Europe. While separate CSV1 instances worked well, it meant one could not shift idle resources between  
 126 projects. A single instance of cloudscheduler would also simplify the operations.

127 Ideally, a single instance of HTCondor with CSV2 is the desired solution. However, the ATLAS and Belle II  
 128 experiments use different authentication in HTCondor, and some sites do not allow their VMs to connect to a  
 129 remote HTCondor pool. CSV2 is designed to use one or more instances of HTCondor with a single instance of  
 130 cloudscheduler.

131 The conclusion is that a new version was necessary to update the software, simplify the operations, extend  
 132 the current capabilities and add new functionality.

### 3 Cloudscheduler V2 distributed compute cloud

#### 3.1 Overview

CSV2 is a framework of components written in Python 3. The in-memory data of CSV1 is replaced with a MariaDB (SQL) database [16]. CSV2 has independent scheduling, polling, and user interface processes that track the state of the clouds and the HTCCondor pool (see Fig. 2).

The MariaDB ensures consistent state information, a vital component of CSV2 reliability. It allows one to optimize data retrieval from multiple sources, and correlate information between different subsystems to obtain different views of the system data. The database also adds authorization features that protects the integrity of the data while allowing both local and remote access.

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 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 HTCCondor. New entries are added to multiple 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<sup>1</sup> (e.g. one poller can support multiple Openstack clouds). Fig. 2 also shows pollers for HTCCondor (labelled as *HTC pollers*). There is a *HTC machine poller* that gathers information on the HTCCondor machines and a *HTC job poller* that gathers information on the jobs in the HTCCondor queue. The pollers themselves consist of multiple pollers 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 very minute and the relatively static information on a longer timescale (between 5 minutes and one day)<sup>2</sup>.

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 (it does not matter whether the experiments use the same instance of HTCCondor). 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.

Although Fig. 2 shows only one HTCCondor instance, CSV2 can support multiple instances. For example, separate HTCCondor instances are used by ATLAS and Belle II experiments, in part, due to differing security requirements. All HTCCondor instances are supported by a single set of HTCCondor pollers. However, each instance of HTCCondor requires a *HTC Agent* to provide the correct security context when issuing requests to the HTCCondor client on the VM (e.g. requesting a VM to de-register from the HTCCondor pool).

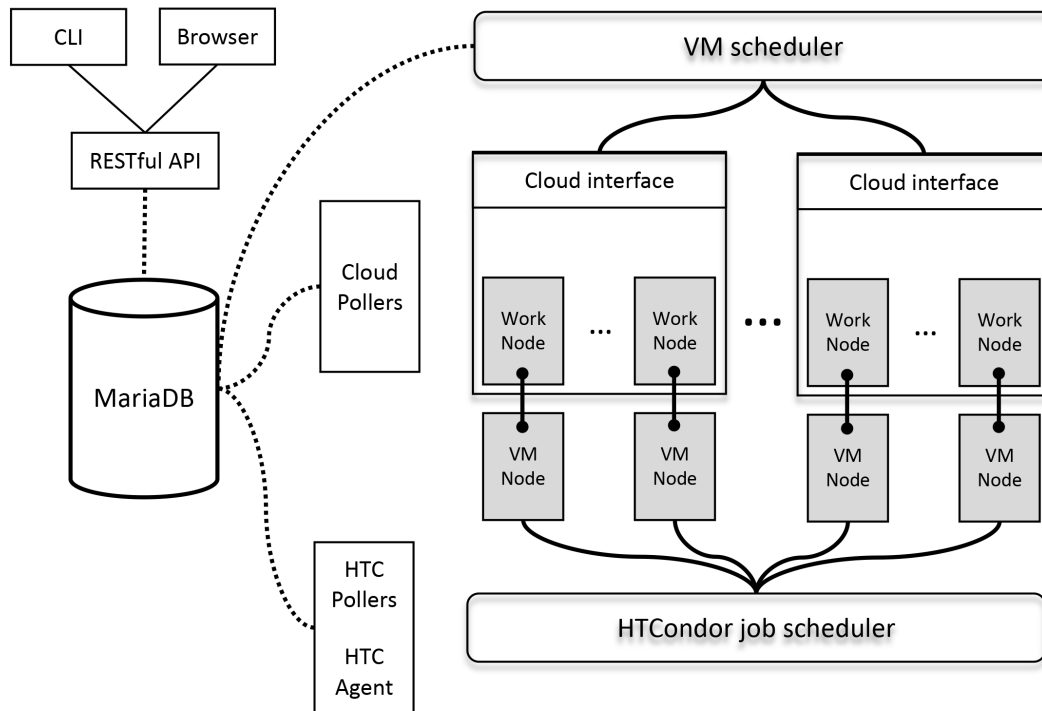
CSV2 requires inter-process communication and signaling. For example, the User Interface (UI) wakes the cloud poller when a cloud is added, so that cloud data is updated immediately or the machine poller signals the HTC Agent to retire a VM. CSV2 uses AMQP, a reliable message delivery protocol, to signal 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.

#### 3.2 Workflow

The VM Scheduler runs every 10 seconds and retrieves virtual tables (or views) from the MariaDB to determine its actions. A view can retrieve and relate data from multiple tables, perform calculations, sort, group, select,

<sup>1</sup> Currently there is API support for Openstack and Amazon EC2 clouds (support for other cloud APIs will be added on demand)

<sup>2</sup> 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
<i>starting</i>	VM is booting/contextualizing
<i>unregistered</i>	VM is running and has not registered in HTCondor pool
<i>idle</i>	VM is running, registered in HTCondor pool and not running jobs
<i>running</i>	VM is running, registered in HTCondor pool and running jobs
<i>retiring</i>	VM is running, retired in HTCondor pool and will complete running jobs
<i>manual</i>	VM is flagged as being manually used and will be ignored by the <i>VM Scheduler</i>
<i>error</i>	VM in error state according to the cloud information

181 concatenate, resort and regroup to derive the information required. One of views determines the state of the VMs  
 182 (see Table 3.1). The primary tables required by the scheduler are the job queue and the compute resources.

183 We briefly describe the steps to create the job queue view. Many of the idle jobs have identical requirements;  
 184 for example, the number of unique types of idle jobs submitted by ATLAS and Belle II is less than ten (e.g. 1  
 185 or 8 core; simulation, production or analysis jobs). The number of each type of idle job is counted and grouped  
 186 with its respective job requirements (called a job-group).

187 Next, the view finds the VM flavours for each cloud that can satisfy the job requirements of the job-group.  
 188 Often the VM flavour is specified for the cloud; otherwise a list of VM flavours available on this cloud is  
 189 retrieved from the MariaDB. Usually a cloud has multiple VM flavours, and the one with the minimal config-  
 190 uration that meets the job requirements (e.g. the fewest cores and/or the least amount of memory) is selected.  
 191 If the cloud has no VM flavours that meets the requirements, then that cloud is not included in the list of VM  
 192 flavours (and will not be selected to boot VMs for this job-group).

193 The result of the view is a table, as shown below, that lists the group name, the number of idle jobs, and a  
 194 list of (Cloud name:VM flavour) for each unique job type (only a subset of the columns in the table is shown).  
 195 Each row corresponds to a unique job type. The first row is for ATLAS 8-core jobs, the second row is for  
 196 Belle II simulation jobs and the third row is for Belle II production jobs. The difference between the two rows

for Belle II is that the first job-group is for simulation (low I/O) jobs and the second job-group is for production (high I/O) jobs. The latter group requires a local Belle II storage element, which gives a 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.xlarge, otter:b8
belle	0	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) 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. If no cloud-priority is set, then the list is based on the alphabetical order using the 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 3.1), 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), 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 c8-30gb-186 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 (an unrestricted query returns over 100 rows).

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

The *VM scheduler* has 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 than ten *idle* VMs within the the clouds associated to a unique job type, then no VMs are started for that job type group.

If the (Cloud name:VM flavour) 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 software 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 is part of the contextualization process and normally occurs within a short time. 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 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 “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 a retire message via AMQP to the *HTC Agent*. The *HTC machine poller* continues to send retire messages until it sees all the partitions on the VM in the retiring state. On receiving a retire message, the *HTC Agent* issues a “condor off” command directly to the HTCondor master daemon running on the VM, causing all the partitions to be marked “retiring”, 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.

### 249 3.3 Configuration

250 CSV2 introduces users and groups to the distributed compute cloud (stored in the MariaDB). Users can be  
251 assigned different access privileges and can be members of multiple groups. Groups are defined, in our case, to  
252 be particle physics experiments or a development and testing area<sup>3</sup>. Each CSV2 group adds the compute clouds  
253 to its configuration and users in the group have the ability to add and manage the clouds.

254 The VM images and SSH keys<sup>4</sup> can be uploaded by logging in directly to the cloud (e.g. the Openstack or  
255 Amazon dashboards) or via the CSV2 GUI. One can define a default VM image and default SSH keys that will  
256 be automatically distributed to all Openstack clouds (this will be extended to other clouds). CSV2 keeps track  
257 of the VM images in the MariaDB. As the list of VM images can change, the *cloud poller* periodically queries  
258 the clouds for the list of available VM images as well as VM flavours and cloud quotas to keep the MariaDB  
259 up to date.

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

266 The GUI is accessed through a web browser and can also generate time series plots for each variable  
267 (stored in an InfluxDB). An example of a time series plot is shown in Fig. 4 where the number of running jobs  
268 for ATLAS and Belle II is plotted for the month of July 2019. For most of 2019, ATLAS provided a sustained  
269 workload of longer running production jobs (typically 6-12 hours) while Belle II has periodic bursts of very  
270 short running analysis jobs (5-10 minutes). The time series plot highlights the difference in workload between  
271 the two projects for June-July 2019. Belle II is still at an early stage of its project lifetime and just starting  
272 to record particle collision data; we expect more production jobs and longer analysis jobs with the increasing  
273 volume of data.

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

### 277 3.4 VM Contextualization for particle physics applications

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

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

284 The small micro-CernVM image (approximately 20 MB) saves storage space and can be started instantane-  
285 ously. The image contains a read-only Linux kernel and a CernVM File System (CVMFS) client. An array  
286 of squid proxy caches around the world are used to host frequently used files from CVMFS [22]. The CVMFS  
287 client is configured to use a specific squid cache. We configure the CVMFS client to use the nearest squid cache  
288 to the VM by querying Shoal.

289 The contextualization process then sets up the HTCondor client, and the environment for the experiment.  
290 The accounting and monitoring processes are activated, and the CPU benchmark suite is run on the VM (see the  
291 next section). Both the HTCondor client and the CPU benchmark are retrieved from CVMFS. The necessary  
292 host certificates and SSH keys are added to the VM. All the VMs share the same host certificate, currently from  
293 GridCanada.

294 It is important to keep track of the resources delivered by a cloud to an experiment. One option is to record  
295 the information as the VM is terminated, however, in some cases the termination is immediate without any  
296 opportunity to publish any information. We configure the VMs to write out their status every 15 minutes to

<sup>3</sup> 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.

<sup>4</sup> The SSH keys allow the user to log into a VM instance for debugging purposes.

<sup>5</sup> The documentation is work in progress

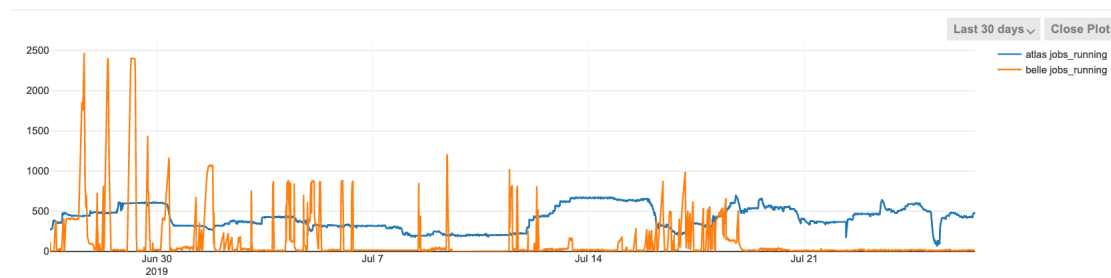


Group	Jobs	Idle	Running	Completed	Held	Other	Foreign
atlas	875	372	483	20	0	0	0
belle	31	4	18	0	9	0	0

Group	Clouds	VMs	Starting	Unreg.	Idle	Running	Retiring	Manual	Error	Native Cores Used	Cores Limit	RAM
atlas	arbutus	309	0	0	0	309	0	0	0	2472	2500	<div style="width: 98%;"></div>
atlas	cc-east	20	0	0	0	20	0	0	0	160	160	<div style="width: 100%;"></div>
atlas	chameleon	12	0	0	0	12	0	0	0	96	96	<div style="width: 100%;"></div>
atlas	otter	50	0	0	0	47	0	0	3	400	400	<div style="width: 100%;"></div>
belle	arbutus	3	0	1	0	2	0	0	0	12	2500	<div style="width: 0.5%;"></div>
belle	cc-east	0	0	0	0	0	0	0	0	0	100	<div style="width: 0%;"></div>
belle	desy	2	0	0	0	2	0	0	0	16	70	<div style="width: 23%;"></div>
belle	ecdf-b	0	0	0	0	0	0	0	0	0	64	<div style="width: 0%;"></div>
<b>Totals</b>		<b>396</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>392</b>	<b>0</b>	<b>0</b>	<b>3</b>	<b>3156</b>	<b>5890</b>	<div style="width: 53.6%;"></div>

**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.



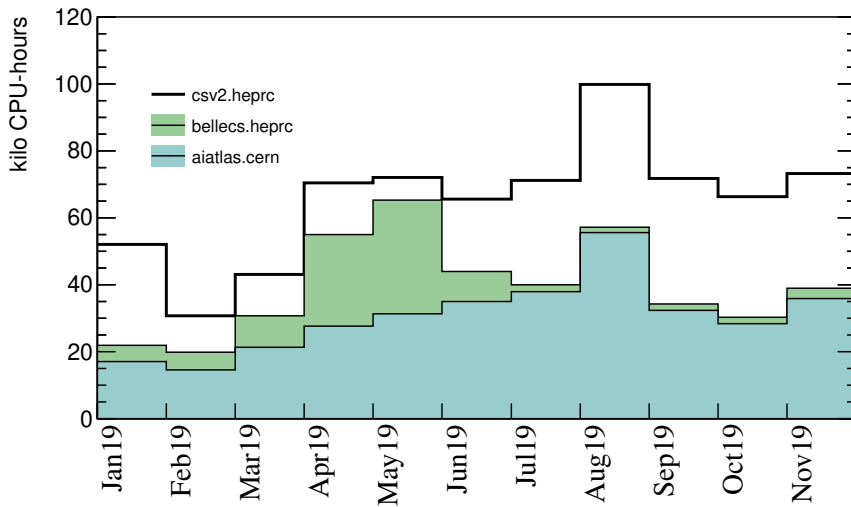
**Fig. 4** Shows the number of running ATLAS jobs (blue) and Belle II jobs (orange). ATLAS was running production jobs requiring 4 or 8 cores whereas Belle II was periodically running very short analysis jobs during the 30-day period in 2019.

297 Elasticsearch. The status includes configuration information, the fast benchmark and the timing information  
 298 (obtained from /proc/stat). We are currently working on reporting our accounting numbers back into APEL for  
 299 ATLAS [23]. The fast benchmark is discussed in the following section.

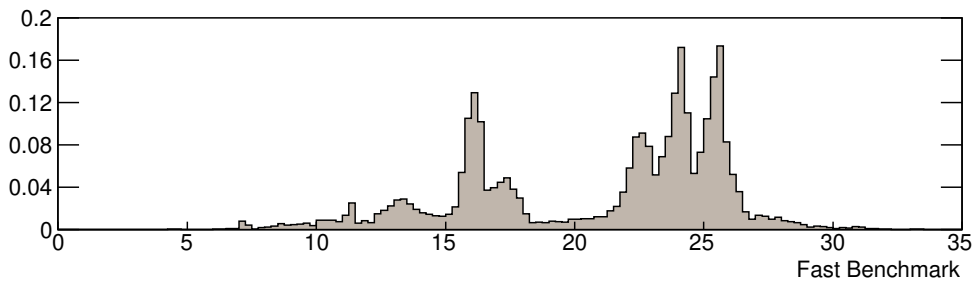
### 300 3.5 Operations and status

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

306 In Fig. 5, we plot a histogram of the CPU hours weighted by a benchmark that is an estimate of the HEP-  
 307 SPEC06 [24] benchmark. A centre usually runs the HEP-SPEC06 benchmark when their resources are being  
 308 commissioned. As we are generally unaware of the underlying hardware, we can not use a static benchmark  
 309 value. It is also impractical to run the HEP-SPEC06 on every VM, as it takes 2-3 hours (and there are also  
 310 licensing issues). Instead, a fast-benchmark code (DB16) [25] is executed at the end of the contextualization of  
 311 the VM instance to measure the performance of the dynamically provided resources.



**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.



**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.

312 Fig. 6 shows a histogram (normalized to unit area) of the fast-benchmark measured for every VM booted  
 313 in 2019. The fast-benchmark gives a reproducible result if the VM is the only activity on the underlying node;  
 314 however, the value is often underestimated if the node is busy or hyperthreading is used. The fast-benchmark  
 315 is not ideal and there are ongoing efforts to find a more reliable estimate of the experiments' workload (see the  
 316 presentation of the HEPiX Benchmark Working Group [26]).

317 In addition, we remark that the distributed compute cloud is using the Dynafed data federation service, de-  
 318 veloped by CERN IT, for locating input data [27, 28]. Dynafed federates existing storage systems and provides  
 319 a link to the closest copy of the data on the basis of GeoIP information. Dynafed is used to federate existing  
 320 storage of the ATLAS and Belle II experiments [29, 30, 31].

## 321 **4 Summary**

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

330 The CSV2 code repository is hosted at GitHub (<https://github.com/hep-gc/cloudscheduler>) and documen-  
331 tation is found at *cloudscheduler.readthedocs.io*.

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

## 335 **Conflict of interest**

336 The authors declare that they have no conflict of interest.

337 **References**

- 338 1. E. Elsen, The end of computing's steam age,  
339 <https://home.cern/news/opinion/computing/end-computings-steam-age>
- 340 2. P. Armstrong *et. al.*, Cloud Scheduler: a resource manager for distributed compute clouds, arXiv preprint arXiv:1007.0050 (2010)
- 341 3. K. Keahey, M. Tsugawa, A. Matsunaga, and J. Fortes, Sky computing, *Internet Computing, IEEE*, 13(5):43–51, 2009,  
342 doi:10.1109/MIC.2009.94
- 343 4. R.P. Taylor *et. al.*, Consolidation of cloud computing in ATLAS, *J. Phys.: Conf. Ser.* 898 052008 (2017), doi:10.1088/1742-  
344 6596/898/5/052008
- 345 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
- 346 6. A. McNab *et. al.*, Managing virtual machines with Vac and Vcycle, *J. Phys.: Conf. Ser.* 664 022031 (2015), doi:10.1088/1742-  
347 6596/664/2/022031
- 348 7. A. Tsaregorodtsev *et. al.*, DIRAC Distributed Computing Services, *J. Phys.: Conf. Ser.* 513 032096 (2014), doi:10.1088/1742-  
349 6596/513/3/032096
- 350 8. R. Grzymkowski *et. al.*, Belle II public and private cloud management in VMDIRAC, *J. Phys.: Conf. Ser.* 664 022021 (2015),  
351 doi:10.1088/1742-6596/664/2/022021
- 352 9. A. Amoroso *et. al.*, A modular (almost) automatic set-up for elastic multi-tenants cloud (micro)infrastructures, *J. Phys.: Conf.*  
353 *Ser.* 898 (2017) 082031, doi:10.1088/1742-6596/898/8/082031
- 354 10. B. Bockelman *et. al.*, Interfacing HTCondor-CE with OpenStack, *J. Phys.: Conf. Ser.* 898 092021 (2017), doi:10.1088/1742-  
355 6596/898/9/092021
- 356 11. K. Fransham *et. al.*, Research computing in a distributed cloud environment, *J. Phys.: Conf. Ser.* 256 (2010) 012003,  
357 doi:10.1088/1742-6596/256/1/012003
- 358 12. I. Gable *et. al.*, A batch system for HEP applications in a distributed IaaS cloud, *J. Phys.: Conf. Ser.* 331 062110 (2011),  
359 doi:10.1088/1742-6596/331/6/062110
- 360 13. D. Thain, T. Tannenbaum and M. Livny, Distributed computing in practice: the Condor experience, *Concurrency Computat.:*  
361 *Pract. Exper.*2005;17:323 (2005), doi:10.1002/cpe.938
- 362 14. The Worldwide LHC Computing Grid, <http://wlcg.web.cern.ch>
- 363 15. F.H. Barreiro Megino *et. al.*, PanDA for ATLAS distributed computing in the next decade *J. Phys.: Conf. Ser.* 898 052002  
364 (2017), doi:10.1088/1742-6596/898/5/052002
- 365 16. MariaDB open-source relational database, <https://mariadb.org>
- 366 17. K. Keahey and T. Freeman, Contextualization: Providing One-Click Virtual Clusters, *IEEE Xplore*: 06 January 2009, DOI:  
367 10.1109/eScience.2008.82
- 368 18. Cloud-init: the standard for customizing cloud instances. <https://cloud-init.io>
- 369 19. J. Blomer *et. al.*, Micro-CernVM: slashing the cost of building and deploying virtual machines, *J. Phys.: Conf. Ser.* 513 (2014)  
370 032009, doi:10.1088/1742-6596/513/3/032009
- 371 20. J. Blomer *et. al.*, New directions in the CernVM file system, *J. Phys.: Conf. Ser.* 898 062031 (2017), doi:10.1088/1742-  
372 6596/898/6/062031
- 373 21. I. Gable *et. al.*, Dynamic web cache publishing for IaaS clouds using Shoal, *J. Phys.: Conf. Ser.* 513 032035 (2014),  
374 doi:10.1088/1742-6596/513/3/032035
- 375 22. Squid is a caching proxy for the Web. <http://www.squid-cache.org>.
- 376 23. APEL is an accounting tool that collects accounting data from sites participating in the EGI and WLCG infrastructures,  
377 <https://wiki.egi.eu/wiki/APEL>
- 378 24. M. Michelotto *et. al.*, A Comparison of HEP code with SPEC1 benchmarks on multi-core worker nodes, *J. Phys.: Conf. Ser.*  
379 219 (2010) 052009, doi:10.1088/1742-6596/219/5/052009
- 380 25. P. Charpentier, Benchmarking worker nodes using LHCb productions and comparing with HEPspec06, *J. Phys.: Conf. Ser.* 898  
381 082011, doi:10.1088/1742-6596/898/8/082011
- 382 26. A. Valassi *et. al.*, Benchmarking WLCG resources using HEP experiment workloads, to be published in the proceedings of the  
383 Computing in High Energy Physics Conference, Adelaide, 2019.
- 384 27. Dynafed - The Dynamic Federation project. <http://lcgdm.web.cern.ch/dynafed-dynamic-federation-project>
- 385 28. F. Furano, O. Keeble and L. Field, Dynamic federation of grid and cloud storage, *Phys. Part. Nuclei Lett.* (2016) 13: 629.  
386 <https://doi.org/10.1134/S1547477116050186>
- 387 29. F. Berghaus *et. al.*, Federating distributed storage for clouds in ATLAS, *J. Phys.: Conf. Ser.* 1085 032027 (2018).  
388 doi:10.1088/1742-6596/1085/3/032027
- 389 30. M. Ebert *et. al.*, Using a dynamic data federation for running Belle-II simulation applications in a distributed cloud environment,  
390 EPJ Web of Conferences 214, 04026 (2019). <https://doi.org/10.1051/epjconf/201921404026>
- 391 31. F. Berghaus *et. al.*, The Dynafed data federator as grid site storage element, to be published in the proceedings of the Computing  
392 in High Energy Physics Conference, Adelaide, 2019.