Cosmos : A Unified Accounting System both for the HT-Condor and Slurm Clusters at IHEP

Ran Du[1,*], *Jingyan* Shi[1,**], *Xiaowei* Jiang[1,***], and *Jiaheng* Zou[1,****]

[1]Institute of High Energy Physics, Chinese Academy of Sciences, Beijing, CHINA, 100049

Abstract. HTCondor was adopted to manage the High Throughput Computing (HTC) cluster at IHEP in 2016. In 2017 a Slurm cluster was set up to run High Performance Computing (HPC) jobs. To provide accounting services for these two clusters, we implemented a unified accounting system named Cosmos.
Multiple workloads bring different accounting requirements. Briefly speaking, there are four types of jobs to account. First of all, 30 million single-core jobs run in the HTCondor cluster every year. Secondly, Virtual Machine (VM) jobs run in the legacy HTCondor VM cluster. Thirdly, parallel jobs run in the Slurm cluster, and some of these jobs are run on the GPU worker nodes to accelerate computing. Lastly, some selected HTC jobs are migrated from the HTCondor cluster to the Slurm cluster for research purposes.
To satisfy all the mentioned requirements, Cosmos is implemented with four layers: acquisition, integration, statistics and presentation. Details about the issues and solutions of each layer will be presented in the paper. Cosmos has run in production for two years, and the status shows that it is a well-functioning system, also meets the requirements of the HTCondor and Slurm clusters.

1 Introduction

HTCondor was adopted to manage the High Throughput Computing (HTC) cluster at IHEP in 2016. One year later, a Slurm cluster was established to manage High Performance Computing (HPC) jobs. Although both the HTCondor and Slurm clusters provide native job accounting services, it is still not enough to meet our demands.

There are four types of jobs to account at present. Firstly, 30 million single-core jobs per year run in the HTCondor cluster. Secondly, Virtual Machine (VM) jobs run in the legacy HTCondor VM cluster. Thirdly, parallel jobs run in the Slurm cluster, and some of these jobs are run on the GPU worker nodes to accelerate computing. Lastly, some selected HTC jobs are migrated from the HTCondor cluster to the Slurm cluster for research purposes. HT-Condor saves accounting data in its history files, while Slurm uses a relational database to account jobs. However, it is not fast enough to get job accounting information with history files, neither Slurm accounts jobs in our favored way. In addition, because both the HTCondor and Slurm clusters are managed by one administrator group, it would be convenient for managing clusters if a unified accounting system was provided. Based on this background,

[*]e-mail: duran@ihep.ac.cn
[**]e-mail: shijy@ihep.ac.cn
[***]e-mail: jiangxw@ihep.ac.cn
[****]e-mail: zoujh@ihep.ac.cn

© The Authors, published by EDP Sciences. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (http://creativecommons.org/licenses/by/4.0/).

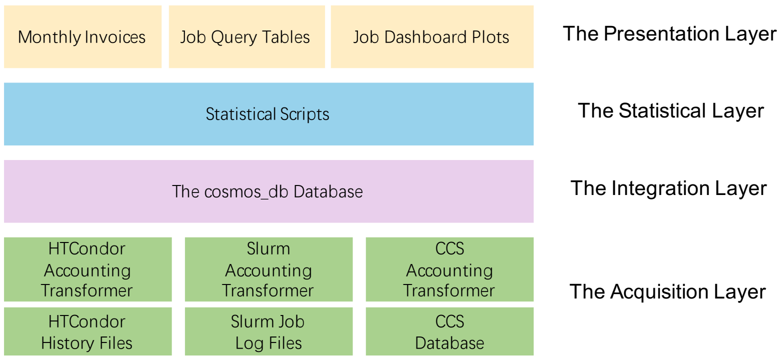


Figure 1: The four-layer system architecture of Cosmos.

Cosmos is implemented as a four layer system: acquisition, integration, statistics and presentation. These four layers work together to generate monthly accounting invoices for users and groups, and to check the status of clusters for administrators.

2 Related Works

Many systems[1–5] are developed to account resource usage. These systems have similar functional components but different implementations. This is because although they share the same target of accounting resource usage, but they differ in infrastructure scale, accounting metrics and the degree of system complexity.

Take CAOS[2] and GRACC[3] for instance. CAOS is developed to account cloud resource usage, and adopts layered architecture as well. However, different to four-layered Cosmos, CAOS consists of three layers: the collector, the backend and the dashboard. From the point of view of functionality, the collector layer of CAOS corresponds to the acquisition layer of Cosmos, the dashboard layer of CAOS can be mapped to the presentation layer of Cosmos, and the backend layer of CAOS is equivalent to the integration layer plus the statistics layer of Cosmos.

A similar situation is found with GRACC, which consists of five modules: probes, data collection, visualization and query tools, message broker, and data sinks. Again, the functionalities of these five parts are similar to the four layers of Cosmos, but with different implementations. Take the data sink as an example. GRACC adopts ElasticSearch (ES) as database, while Cosmos adopts MariaDB[6] to store accounting data. MariaDB is adopted because it is lightweight enough in our situations for now. But as is mentioned by GRACC, as the system scales up and becomes more complicated, new solutions like ES might be adopted.

3 Design and Implementation

Cosmos is implemented as a four-layer system: acquisition, integration, statistics and presentation. Accounting data are collected and transformed by the **acquisition layer**. Later the **integration layer** is responsible for transferring and storing the transformed data, which makes it easier for the **statistics layer** to generate job accounting statistics. Finally, users get detailed job information through the interfaces provided by the presentation layer. Figure 1 shows the four-layer system architecture of Cosmos.

3.1 The Acquisition Layer

The acquisition layer is responsible for collecting data for later accounting. Collected data are classified into three categories. The first category includes data about users, groups and experiments, the second category consists of hosts and hardware descriptions, and the third category are job metrics. Table 1 lists the data categories collected and transformed by the **acquisition layer**.

The first two categories are stored in a MySOL database called CCS that is easy to access, however, there are other issues to resolve. First of all, metrics stored in CCS have to get tailored for Cosmos usage. To solve this issue, we implemented several Python scripts to filter necessary metrics from CCS, and the filtered metrics are synchronized in intervals. Second, CCS is working as a data layer for other systems at the same time. In order not to interfere with other systems, the synchronization interval is set to 24 hours at present.Third, as administrators could update CCS database at anytime, with the low frequency of data synchronization, there could be data inconsistency. To make the accounting correct, the transformers in the integration layer (Sect. 3.2) treat the synchronized data as a cache: if data is missing, CCS would be queried instead to get the most up-to-date information.

Job metrics, the third category of data to collect, is more complicated to get. As mentioned in Sect. 1, there are four types of jobs, as a result, multiple workloads bring different accounting requirements.

Firstly, HTCondor and Slurm save job data in different ways. HTCondor saves detailed job information in history files[7], while Slurm adopts a database[8] to store jobs detailed information. However, after reviewing the Slurm job tables, it is found that job resource allocation is recorded in a node list way, but not by how-many-cores-per-node. Fortunately, Slurm provides a *scontrol show* command. With this command executed, detailed job information could be written into log files. As a result, both HTCondor and Slurm job information are collected from files. These two data transformers are implemented by the integration layer (Sect. 3.2) to extract job metrics from the files.

Secondly, it is normal that Slurm jobs may run more than one month since most Slurm jobs are multi-core parallel jobs. Because Cosmos is responsible to provide monthly accounting invoices for experiments, unfinished long jobs must be accounted as well. To solve this issue, Slurm prolog scripts are written to log unfinished jobs, and epilog scripts are used to log finished jobs. When the job transformers extract metrics from the log files, metrics of unfinished and finished jobs are saved in independent tables, and different accounting methods are imposed on these tables to generate monthly invoices.

Thirdly, HTCondor history files will get removed one week after file creation. Besides, history files are saved by the schedd server which is busy scheduling jobs. In order not to over-burden the schedd server, also not to miss any history files before file removal, finished job metrics are extracted every day.

Lastly, a Virtual Machine (VM) pool was added to the HTCondor cluster in 2019, which means the VM jobs have to be accounted as well. The accounting difference between the VM jobs and HTCondor jobs lies in the hostname mappings. To map the VM hostname and the physical hostname, a mapping interface is implemented. With the timestamp and the VM hostname, a physical hostname is returned if successfully matched.

3.2 The Integration Layer

Accounting related data are collected by the **accquisition layer** (Sec 3.1), later these data are transferred and stored in the **integration layer**. To make accounting easier, data are integrated and saved in a MariaDB database named *cosmos_db*. To allow Cosmos to scale-up

Table 1: Data categories collected by the acquisition layer.

Category	Data Description	Data Source
1	users, groups, experiments	the CCS database
2	hosts, hardware	the CCS database
3	job metrics	HTCondor and Slurm job log files

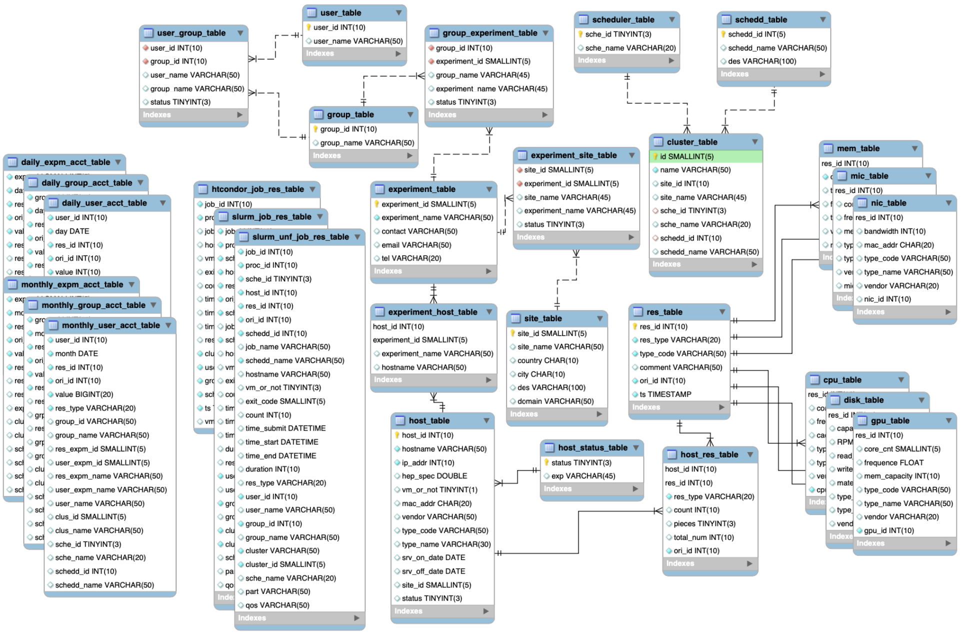


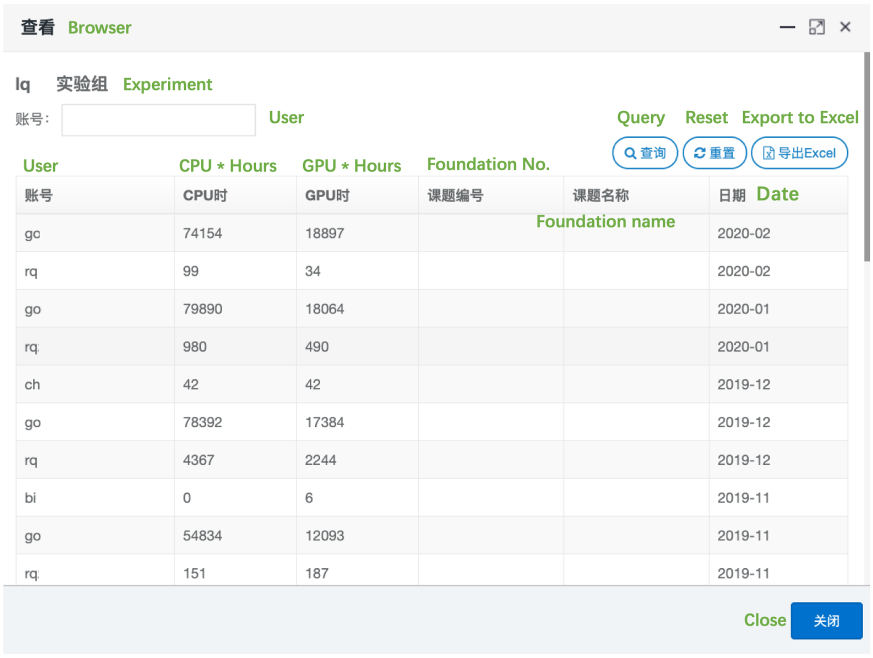
Figure 2: The schema design of the cosmos_db database.

and incorporate other clusters, tables of schedulers, clusters, and sites are established as well. Figure 2 shows the schema design of the cosmos_db database.

3.3 The Statistical Layer

The **statistical layer** has two functionalities. (1) To generate job monthly invoices for experiments and groups. (2) To implement statistical methods called by the job query interfaces in the **presentation layer** (Sect. 3.4).

To generate job monthly invoices, a Python package named *monthly_invoice* is implemented. The monthly_invoice package provides multiple command line options. With these options, administrators are able to generate monthly invoices of HTCondor jobs, VM jobs, and Slurm jobs respectively, or make a customized combination based on options. Besides, there are two ways to generate invoices: one is with REST APIs, the other is in csv files. Invoices generated with the REST APIs are uploaded to another support system named *service*, where experiment and group administrators browse monthly invoices. On the other hand, the csv output files are used to debug invoice generation or to import into another database for further development. Figure 3 shows the service system interface.



账号	CPU时	GPU时	课题编号	课题名称	日期
gc	74154	18897			2020-02
rq	99	34			2020-02
go	79890	18064			2020-01
rq	980	490			2020-01
ch	42	42			2019-12
go	78392	17384			2019-12
rq	4367	2244			2019-12
bi	0	6			2019-11
go	54834	12093			2019-11
rq	151	187			2019-11

Figure 3: Monthly invoices on the service system web page.

Job query and statistical requirements come from the HTCondor and Slurm administrators. To meet these requirements, the query and statistical methods are implemented in the **statistical layer** with the Python packages pandas[9] and plotly[10] . Pandas is used to format data stored in *cosmos_db* into dataframes, and dataframes can be turned into HTML tables directly in the **presentation layer** (Sect. 3.4). In addition, plotly takes dataframes as input to generate JavaScript statistical plots which are displayed by the **presentation layer** (Sect. 3.4).

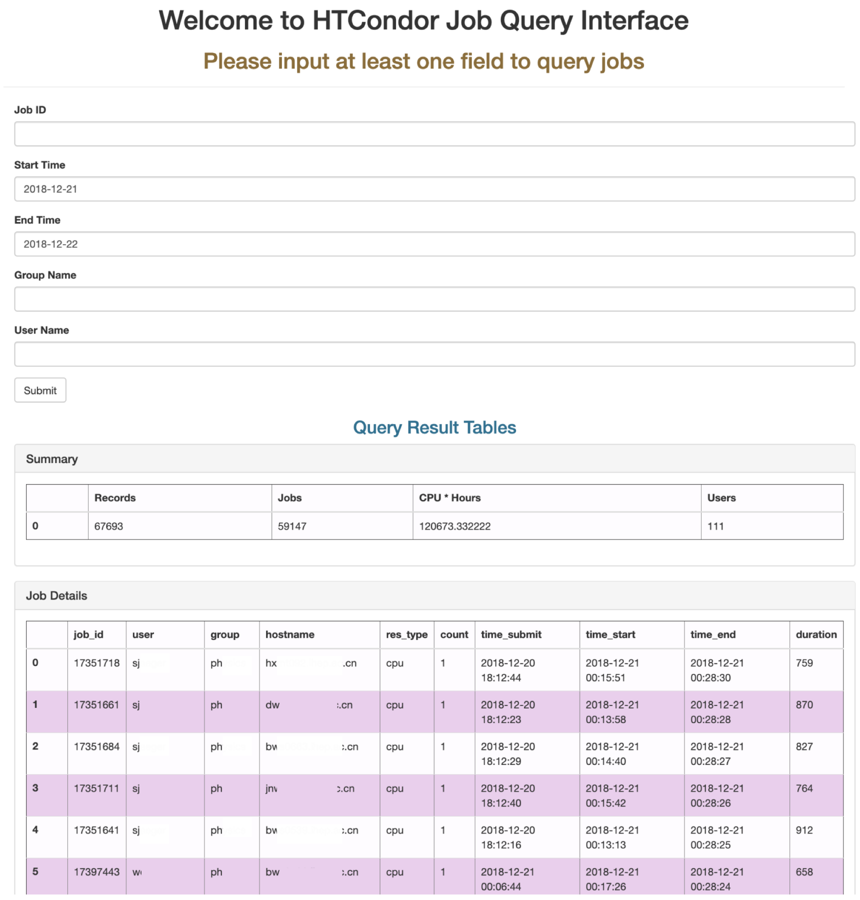
3.4 The Presentation Layer

As mentioned in Sect. 3.3, job query and statistical methods are implemented in the **statistical layer**, and the presentation layer is the very place to display the statistical results.

Two presentation types are supported, one is tables, the other is plots. Tables and plots are presented on web pages, and the Python package flask[11] is adopted to implement the web interface. With flask extensions, it is easy to present query and statistical results, also straight-forward to add more functionalities if additional requirements are proposed in future. Figure 4 and Figure 5 show the web interface.

4 System Status

Cosmos has been in production for two years. It is implemented to provide unified accounting services both for the HTCondor and Slurm clusters. As time goes by, more accounting requirements are proposed. As a result, Cosmos is always in the status of alpha-beta. To deal with additional requirements, git branches are used to modify Cosmos layers, and new


Welcome to HTCondor Job Query Interface

Please input at least one field to query jobs

Job ID	
Start Time	2018-12-21
End Time	2018-12-22
Group Name	
User Name	

Submit

Query Result Tables

Summary

	Records	Jobs	CPU * Hours	Users
0	67693	59147	120673.332222	111

Job Details

	job_id	user	group	hostname	res_type	count	time_submit	time_start	time_end	duration
0	17351718	sj	ph	hx .cn	cpu	1	2018-12-20 18:12:44	2018-12-21 00:15:51	2018-12-21 00:28:30	759
1	17351661	sj	ph	dw :.cn	cpu	1	2018-12-20 18:12:23	2018-12-21 00:13:58	2018-12-21 00:28:28	870
2	17351684	sj	ph	bw :.cn	cpu	1	2018-12-20 18:12:29	2018-12-21 00:14:40	2018-12-21 00:28:27	827
3	17351711	sj	ph	jnv :.cn	cpu	1	2018-12-20 18:12:40	2018-12-21 00:15:42	2018-12-21 00:28:26	764
4	17351641	sj	ph	bw :.cn	cpu	1	2018-12-20 18:12:16	2018-12-21 00:13:13	2018-12-21 00:28:25	912
5	17397443	w	ph	bw :.cn	cpu	1	2018-12-21 00:06:44	2018-12-21 00:17:26	2018-12-21 00:28:24	658

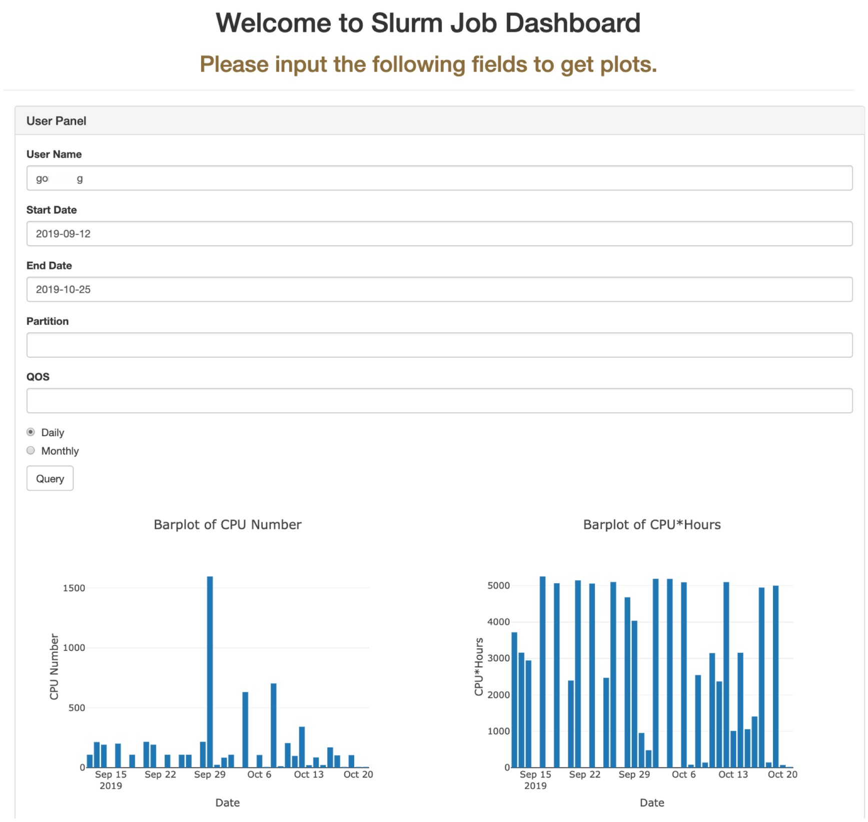
Figure 4: The job query table interface of Cosmos.

branches will be merged to the production version after careful tests. With the four-layer architecture, it is straight-forward to adapt to new requirements.

Cosmos is going to upgrade in the near future. Some remote sites have joined the HTCondor cluster at IHEP, so that accounting jobs run on these remote sites is becoming necessary. Slurm is going to upgrade to a new version in the summer of 2020, new metrics will be added to the accounting.

5 Conclusion

This paper presents a unified accounting system *Cosmos*. Cosmos is designed to account jobs both from the HTCondor and Slurm clusters at IHEP. To satisfy requirements of multiple workloads, Cosmos is implemented as a four-layer system: acquisition, integration, statistics and presentation. Each layer implements its core functionalities, also works together to generate monthly invoices and job statistics. Cosmos has been in production for two years, and provides solid accounting supports both for the HTCondor and Slurm clusters. And with the four-layer architecture, Cosmos is easy to adapt to new requirements.


Welcome to Slurm Job Dashboard

Please input the following fields to get plots.

User Panel

User Name: go g

Start Date: 2019-09-12

End Date: 2019-10-25

Partition:

QOS:

[✓ Daily] Monthly

Query

Figure 5: The job dashboard plots of Cosmos.

6 Acknowledgements

We wish to thank all the colleagues who contributed to this paper. The work presented is supported by the National Natural Science Foundation of China "The Two-staged Job Scheduling Algorithms and Resource Management Research about Workload Integration between HTC Cluster and HPC cluster" (No. 11805225), the National Natural Science Foundation of China "Container Virtualization Applied to High Energy Physics Computing" (No. 11775250), and the Youth Innovation Promotion Association of Chinese Academy of Sciences.

References

[1] K. Retzke, D. Weitzel, S. Bhat, T. Levshina, F. Wuerthwein, *GRACC: New generation of the OSG accounting*, in *Journal of Physics Conference Series* (2017), Vol. 898, p. 092044

[2] P. Andreetto, F. Chiarello, S. Traldi, *CAOS: a tool for OpenStack accounting management*, in *EPJ Web of Conferences* (EDP Sciences, 2019), Vol. 214, p. 07006

[3] D. Weitzel, B. Bockelman, M. Zvada, K. Retzke, S. Bhat, *GRACC: GRid Accounting Collector*, in *EPJ Web of Conferences* (EDP Sciences, 2019), Vol. 214, p. 03032

[4] A. Coveney, G. Corbett, *EGI Dataset Accounting and the WLCG*, in *EPJ Web of Conferences* (EDP Sciences, 2019), Vol. 214, p. 03028

[5] J. Andreeva, D. Christidis, A. Di Girolamo, O. Keeble, *WLCG space accounting in the SRM-less world*, in *EPJ Web of Conferences* (EDP Sciences, 2019), Vol. 214, p. 04021

[6] *Mairadb*, https://mariadb.org/, online, accessed on 10-Mar-2020

[7] *Htcondor history log files*, https://htcondor.readthedocs.io/en/stable/misc-concepts/logging.html, online, accessed on 10-Mar-2020

[8] *Slurmdbd service*, https://slurm.schedmd.com/slurmdbd.html, online, accessed 10-Mar-2020

[9] *Pandas*, https://pandas.pydata.org/, online, accessed on 10-Mar-2020

[10] *Plotly*, https://plot.ly/, online, accessed on 10-Mar-2020

[11] *Flask*, https://palletsprojects.com/p/flask/, online, accessed on 10-Mar-2020