BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **17**, No 5
Special issue with selected papers from the workshop
"Two Years Avitohol: Advanced High Performance Computing Applications 2017"

Sofia • 2017 Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.1515/cait-2017-0057

Tools and Services for High Performance Computing

Accounting Services for Heterogeneous Computing Resources

Dimitar Dimitrov, Emanouil Atanassov

Institute of Information and Communication Technologies, Bulgarian Academy of Sciences, 1113 Sofia, Bulgaria
E-mails: *d.slavov@bas.bg* *emanouil@parallel.bas.bg*

Abstract: *The accounting platform is a web-service based system for collection and analysis of accounting data from different infrastructure resources like High Performance Computing* (*HPC*)*, Cloud and storage systems. The platform has two major components – backend API services along with different data publishers and a client web UI module for visualization and operations. The backend API is designed to gather information from different job management systems, cloud vendors, and storage providers and use micro-service architecture. The web UI module is written in Python, JavaScript and has integrated SAML login module for user authentication and authorization. It is capable of visualizing the gathered data in dynamic OLAP style and supports standard export formats like CSV and Excel. Through the accounting platform, it is possible to obtain a full view of the usage patterns of an integrated electronic infrastructure and to see from one point all information about the different resources comprising the hybrid computing and data infrastructure.*

Keywords: *Distributed systems, containers, scalable development, analytics.*

1. Introduction

In the last decade, there was a drastic expansion of the computing resource types available for the scientific communities to operate with. Projects like EGEE

(European e-Infrastructure for research and e-Science) [1] and regional ones like SEE-GRID-1/2, SEE-GRID-SCI, BalticGrid, etc., had established a stable Grid computing infrastructure [2, 3] across whole Europe with significant interest and usage from different research communities like natural sciences [4], meteorology, environmental protection, e-Health, e-Government, and etc. The infrastructure in these projects was mainly low to mid-range standard server-side hardware. After that project like PRACE (Partnership for Advanced Computing in Europe) [5] and regional ones like HP-SEE [6] had put the foundation for inter-Europe collaboration on High Performance Computing (HPC) level where the project partners had offered advanced High-Performance Computing hardware. In the last few years, Cloud computing came into focus and EGI Cloud Federation [7] was established offering IaaS-type cloud services made of academic private clouds and virtualized resources. All these different resources provide different computational capabilities for the researchers but they have different operational processes, usage records, and monitoring requirements.

Having such a different landscape of resources, we decided to create an accounting system to operate with infrastructures of various types and collect data for heterogeneous computing resources. The platform is used for the resources in the data center of the Institute of Information and Communication Technologies and the VI-SEEM [8] project partners' data centers.

2. Infrastructure and resources

The accounting system was designed to operate and collect data from the infrastructure resources on a national level and the regional ones that we collaborate with. On a national level, we have Avitohol supercomputer system and HPCG system (**http://www.hpc.acad.bg**).

The supercomputer Avitohol is a heterogeneous system , which consists of 150 HP Gen8 servers and 300 Intel Xeon Phi 7120P coprocessors. Each of the servers has two CPUs – Intel Xeon E5-2620 V2 2.6 GHz and two Xeon Phi coprocessors. There are four management nodes to operate the storage system – an HP MSA 2040 SAN with around 100 TB capacity. All the nodes are connected in a fat-free topology via non-blocking FDR InfiniBand connection. To manage the system resource a torque/Moab resource manager is installed on the gateway node. There are a lot of libraries and frameworks available for the user but in order to use it effectively one must port or use a code that is effectively written for such heterogeneous system with coprocessors. In such architectures, the accounting is very tricky, in our case on the main node there are 32 cores on 2.6 GHz but on the coprocessor nodes there 480 cores on 1 Ghz. Also, an OpenStack environment is installed that offers infrastructure-as-a-service (IaaS), whereby scientists have the option to use virtual servers and other resources as a private cloud.

HPCG system is an HP-based cluster, on the previous generation HP Cluster Platform Express 7000. It is operational since 2010 with hardware upgrades in 2014 and 2015. It is a very heterogeneous system because it has standard CPU nodes, nodes with Xeon Phi coprocessors and nodes with NVidia Tesla M2090 cards as described in Fig. 1.

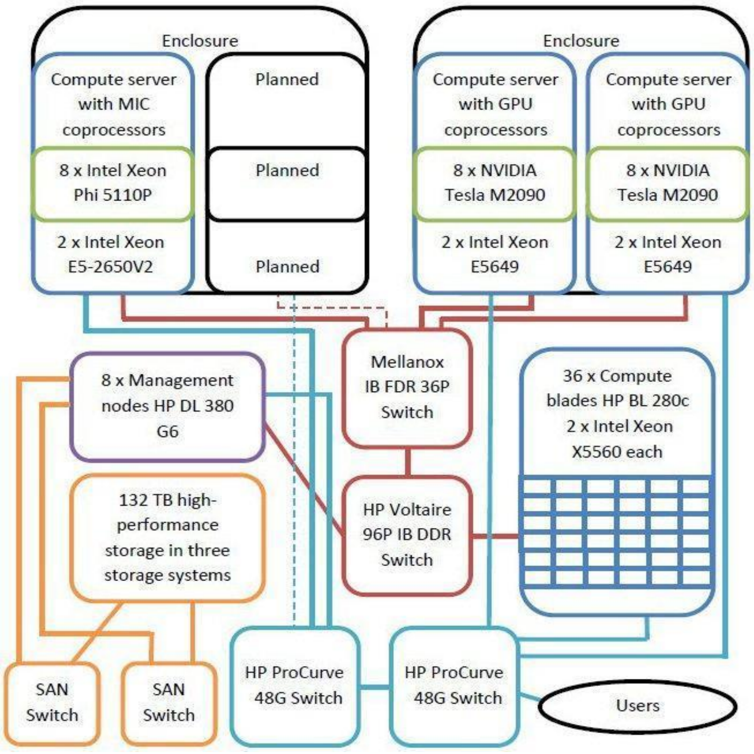
Enclosure
Compute server with MIC coprocessors
8 x Intel Xeon Phi 5110P
2 x Intel Xeon E5-2650V2
Planned
Planned
Planned
Enclosure
Compute server with GPU coprocessors
8 x NVIDIA Tesla M2090
2 x Intel Xeon E5649
Compute server with GPU coprocessors
8 x NVIDIA Tesla M2090
2 x Intel Xeon E5649
8 x Management nodes HP DL 380 G6
Mellanox IB FDR 36P Switch
36 x Compute blades HP BL 280c 2 x Intel Xeon X5560 each
132 TB high-performance storage in three storage systems
HP Voltaire 96P IB DDR Switch
SAN Switch
SAN Switch
HP ProCurve 48G Switch
HP ProCurve 48G Switch
Users
Fig. 1. HPCG system architecture

Table 1. VI-SEEM resources by project collaborate countries

Resource	Country	Total		
		CPU-cores	GPU-cores	Phi-cores
ARIS	Greece	8.520	–	–
Cy-Tera	Cyprus	1.392	16.128	–
Avitohol	Bulgaria	2.400	–	18.300
PARADOX	Serbia	1.696	108.544	–
NIIFI SC	Hungary	768	–	–
Leo	Hungary	1.344	628.992	–
InfraGRID	Romania	456	3.136	
ICAM	Romania	4.096	–	–
UPT-HPC	Albania	144	–	–
FINKI	FYROM	768	–	–
Armcluster	Armenia	128	–	–
BA-HPC	Egypt	1.040	–	–
Gamma	Jordan	8	2.496	–
Zaina	Jordan	56	–	–
Total		22.816	759.296	18.300

On a regional level, we have established collaboration for e-Infrastructure with most of the countries in the Southeast Europe Region and operate the accounting services in the common projects we have (Table 1). On the table below there is a list of the resources that accounting system currently works with, in the scope of the VI-SEEM project – which aim is to create a Virtual Research Environment (VRE) in Southeast Europe and Eastern Mediterranean. Data in accounting is collected on a daily basis and supports HPC, Cloud, Grid and Storage records.

3. Platform architecture and operations

The accounting system accumulates and reports utilization of the different types of resources in the infrastructure using standard metrics. The information that is relevant to the VI-SEEM project or other project resources usage is gathered and organized in different databases.

The starting point of the information is the resource managers that hold usage records for the resource they operate on. Due to the heterogeneity of the resources, an individual approach is applied based on the management software. For example, most of the HPC resources in the VI-SEEM project use Torque/Maui job schedulers that store the data in log files but other use SLURM manager that store the usage data in a database. That is why based on the resource manager different publisher agents are developed. These are lightweight clients for the Accounting backed REST API that the only purpose is to collect, pack and send the data. For most of the systems, there is an agent developed and there is a basic one that can be easily extended if the resource manager is not in the supported list or there are some specific requirements or policy that needs to be taken into account. The implemented agents are written in Python and for ease of deployment and simplicity, the library dependencies are kept to a minimum. Upon installation of the agent the installation script creates an appropriate cron job (a system task to run periodically on daily basis) to call the backend. When triggered the agent asks the backend for the latest known information on its side by timestamp and sends only the missing part. This assures a lightweight procedure for an update and an easy resubmit for all the information if needed.

The server-side (Fig. 2) is implemented in again in Python using Flask web framework [9]. The server-side offers different approaches to send information – it can either receive log files and parse them based on the resource origin or the resource itself can consume the REST API for the resource type via custom logic. At this moment there are four basic resource types that are supported – HPC, Grid, Cloud, and Storage but for the future, they can be extended. The architecture of the backend is done in a way that allows dynamic specification and API endpoint definition. All modules are put in containers and can be easily shipped or scaled. At the current stage, the system works normally with one Nginx server [10] as load-balancer and reverse-proxy, three Gunicorn [11] servers for the Flask API and one MySQL database.

The access to the accounting reports is available via the web interface that supports the VI-SEEM Single Sign-on for authentication and authorization and a local user management. The user can access the information grouped in a table by date, year, country, resource name, virtual research community and application in rows and columns. For each of the resource types there is separate view – compute data for HPC and GRID computing data, cloud data for cloud accounting and storage data for storage accounting. Each generated report is presented in a table and simple charts grouped by the user choice.

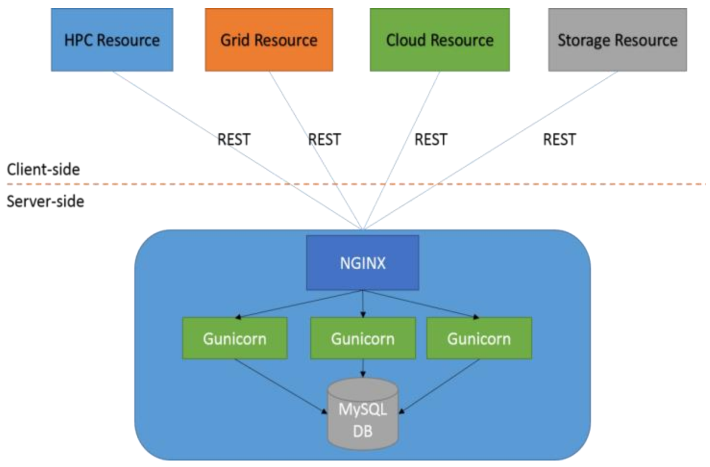

Fig. 2. Accounting backend architecture

The user roles in the accounting system have three levels of hierarchy – regular user, site manager, and administrator. The regular user role allows the user to browse and extract reports about the usage and can be limited to the information he can see. The site manager role is for the local admins where they have extended rights to configure the resources they manager – to describe an application in their resource, to map HPC queue directly to application to retrieve an API key the REST API and so on. The administrator of the accounting system has all the privileges above and one extra tab view where a summary of the published log records is shown and a resource management option to update user privileges.

The UI allows dynamic OLAP style data visualization where the user defines the parameters how data to be grouped by within a time range. For each of the resources, there are defined parameters that can be chosen. The front-end interface is written in a combination of Flask Jinja2 templates and different JavaScript libraries for datatype management, better user experience, charting and excel export, and etc.

4. Integration, deployment and processes automation

In order to deliver the platform described above, we needed to solve several challenges that anybody faces during building and maintaining a multi-module distributed system. The solution that solved most of them was a combination of Docker [12] and Rancher [13] (as container manager for Docker environments).

An agile platform that is composed of various services as the accounting one should be portable and easily scalable. Here the term "scalability" is used from a backend operational perspective. For each of the resources that are monitored, the system manages and collects various data in terms of format and size. That is why each of the collector services for resource or group of resources are built and shipped in a separate Docker container. The containers give this easy isolation and portability between machines and environments.

We can generalize the development process in four stages (Fig. 3) – feature/module development, integration phase, testing, and production deployment.

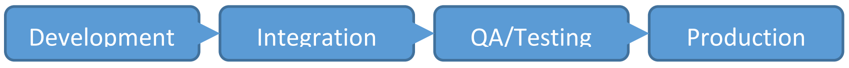

Fig. 3. Process flow

The initial point of the process is the version control system (in our case git [14]). When a developer wants to create a new feature or update some module it must use the git-flow model for branching, versioning and release it. The model is based on the idea to support two branches of the application – development and master branches. Upon start of a new feature/work the developer must create a temporary branch for his work and when it is finished and passes the tests, the new code is merged into the development branch. After a successful merge with the development branch, the continuous integration server is responsible to compile, run the automation tests and provide a working version of the application for quality assurance and review. Based on the outcome of the quality review the feature/update is either send back to be reworked or a new release is created by merging the development branch in the master one. The process is visualized in Fig. 4.

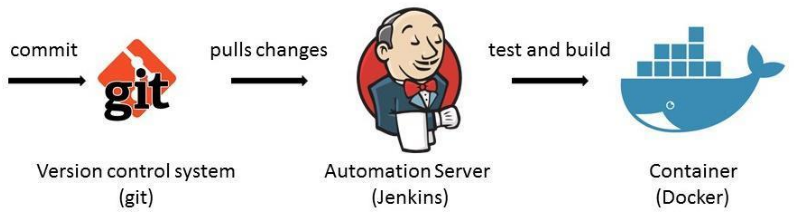

Fig. 4. Integration flow

When the master branch is stable with the new features or updates a Docker image is created as stable release ready version. This is the point where the Rancher comes in use. Rancher is a container management platform that manages the containers using multi-layer architecture. There is Rancher server(s) and Rancher hosts. Rancher server is the management node; it orchestrates the whole infrastructure and provides the user interface and API for it. The rancher hosts are the basic units of resource in the Rancher environment and can be represented as any Linux machine, physical or virtual with specific minimum requirements for CPU, RAM, and OS version. This approach enables the deployment of containers on different infrastructures and even different resource providers as Amazon Cloud, Google Cloud or private solutions like VMWare vSphere. The hosts have rancher agent running on them that provides the necessary information about the host health state and the running containers. Rancher provides the option to define different container environments on a different host in this way one can have QA environment on a private local machine and production and early customer release environment on public infrastructure. The deployment and update of containers are done via the Rancher server, once a Docker image(s) is ready and put in some container repository, the deployment can be done via automation solution like Jenkins [15] or the manual via Rancher CLI or UI interface. For deployment, Rancher supports application stack templates, where the admin can define the architecture of the containers for example – load-balancer with two application server containers and one database for them (see Fig. 5).

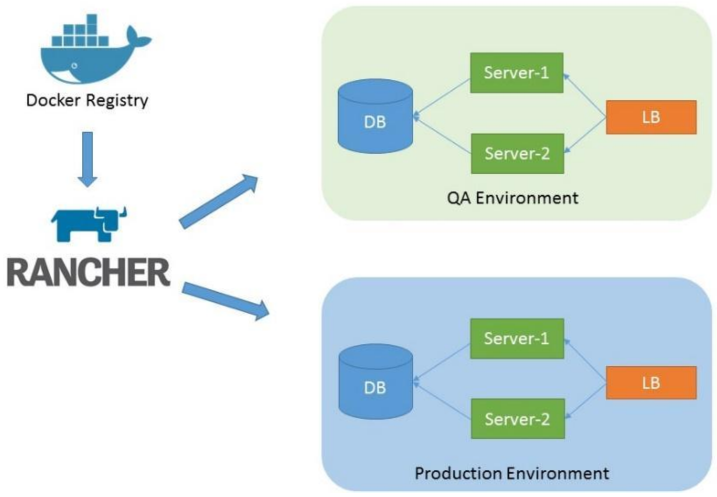

Fig. 5. Deployment flow

5. Future work and conclusion

The growth and expansion of the variety of computational resources, services, and heterogeneous systems require more agile systems for monitoring, accounting, and operations. In the presentwork we demonstrate an accounting system that can run in a container which offers rapid development, deployment, and portability across different infrastructure. From a software architecture point of view, the system is designed to be modular and easy extendable if new data needs to be collected and processed.

In the future, we plan to extend the system to support fully customized accounting data types. The site manager can specify dynamically what will be the data format that he needs to collect and the system will create a dynamic REST API endpoint for this data type. Another feature that will be very helpful is a usage pattern and prediction functionality [16] to provide insights about the usage of the resources and provide help for better utilization and efficiency. A module like this will be very useful to show a correlation between high-level services and hardware utilization because at this moment it is very hard, almost impossible to account this.

***Acknowledgements*:** This work is supported by EC programme H2020, Grant Agreement No 675121, VI-SEEM Project and by the National Science Fund of Bulgaria under Grant DFNI-I02/8.

References

1. The EGEE project website.
   http://www.eu-egee.org
2. N e w h o u s e, S., J. M. S c h o p f, A. R i c h a r d s, M. A t k i n s o n. Study of User Priorities for e-Infrastructure for e-Research (SUPER). – In: Proc. of UK All Hands Meeting, September 2007.
3. M i s e v, A., E. A t a n a s s o v. User Level Grid Quality of Service. – In: I. Lirkov, S. Margenov, J. Waśniewski, Eds. Large-Scale Scientific Computing. LSSC 2009. Lecture Notes in Computer Science. Vol. **5910**. Berlin, Heidelberg, Springer, 2010.
4. A n d r e e v a, J. Monitoring of the Computing Activities of the LHC Experiments for the CCRC08 and BEYOND. – In: Proc. of 3rd International Conference Distributed Computing and Grid-Technologies in Science and Education, 30 June-4 July 2008, Dubna, Russia.
5. Partnership for Advanced Computing in Europe.
   http://www.prace-ri.eu/
6. High-Performance Computing Infrastructure for South East Europe's Research Communities.
   https://www.hp-see.eu/
7. EGI Federated Cloud.
   https://www.egi.eu/federation/egi-federated-cloud/
8. Virtual Research Environment in Southeast Europe and the Eastern Mediterranean.
   https://vi-seem.eu/
9. Flask Web Framework.
   http://flask.pocoo.org/
10. Nginx Server.
    https://nginx.org
11. Gunicorn Python WSGI HTTP Server for UNIX.
    http://gunicorn.org/
12. Docker Container.
    https://www.docker.com
13. Rancher – Container Management Platform.
    https://rancher.com
14. Git Version Control System.
    https://git-scm.com/
15. Jenkins Automation Server.
    https://jenkins.io /
16. ProM tool for Process Mining,
    http://www.processmining.org/
17. V e n k a t r a m, K., M. G e e t h a. Review on Big Data and Analytics – Concepts, Philosophy, Process and Applications. – Cybernetics and Information Technologies, Vol. **17**, 2017, No 2, pp. 3-27.