Web Portals for High-performance Computing: A Survey

PATRICE CALEGARI and MARC LEVRIER, Atos, France
PAWEŁ BALCZYŃSKI, Atos, Poland

This article addresses web interfaces for High-performance Computing (HPC) simulation software. First, it presents a brief history, starting in the 1990s with Java applets, of web interfaces used for accessing and making best possible use of remote HPC resources. It introduces HPC web-based portal use cases. Then it identifies and discusses the key features, among functional and non-functional requirements, that characterize such portals. A brief state of the art is then presented. The design and development of Bull extreme factory Computing Studio v3 (XCS3) is chosen as a common thread for showing how the identified key features can all be implemented in one software: multi-tenancy, multi-scheduler compatibility, complete control through an HTTP RESTful API, customizable user interface with Responsive Web Design, HPC application template framework, remote visualization, and access through the Authentication, Authorization, and Accounting security framework with the Role-Based Access Control permission model. Non-functional requirements (security, usability, performance, reliability) are discussed, and the article concludes by giving perspective for future work.

CCS Concepts: • **Human-centered computing** → **Graphical user interfaces**; • **Information systems** → *RESTful web services*; • **Computing methodologies** → *Massively parallel and high-performance simulations*; • **Software and its engineering** → Cloud computing;

Additional Key Words and Phrases: High-performance computing, HPC, HPC-as-a-service, HPCaaS, software-as-a-service, SaaS, web portal, science gateway, dashboards, RESTful API, user interface, customizable GUI, application templates, job management, cloud computing, service-oriented architectures

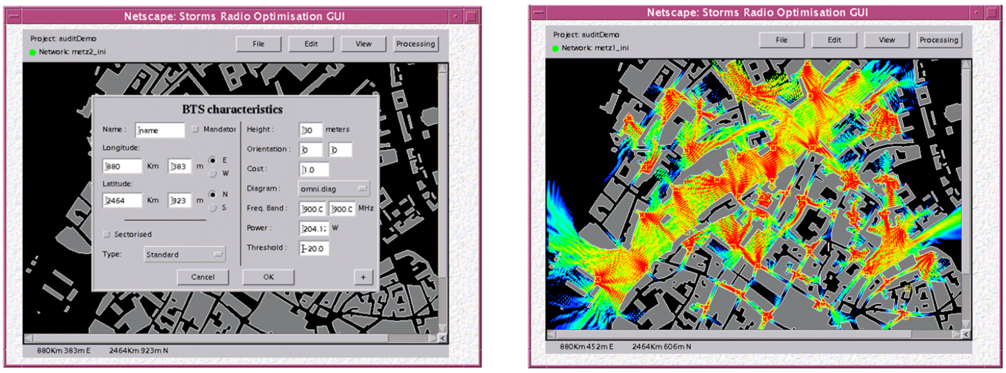
ACM Reference format:
Patrice Calegari, Marc Levrier, and Paweł Balczyński. 2019. Web Portals for High-performance Computing: A Survey. *ACM Trans. Web* 13, 1, Article 5 (February 2019), 36 pages.
https://doi.org/10.1145/3197385

1 INTRODUCTION

The computing power of supercomputers is continuously increasing. In the November 2018 Top500 [68] ranking, the Department of Energy's Summit supercomputer, installed at Oak Ridge National Laboratory (ORNL), reached a Linpack performance of 143.5 petaflops (i.e., 143.5×10^{15} floating point operations per second) on 2,397,824 cores (i.e., processing units), improving its own performance by 17.3% in 6 months. This is a 54.3% performance increase compared to the former most powerful computer in the world that reached 93 petaflops one year earlier: Sunway TaihuLight, a system developed by China's National Research Center of Parallel Computer Engineering

Authors' addresses: P. Calegari and M. Levrier, Atos, Big Data and Security, HPC & Quantum division, extreme factory, 2 rue de la Piquetterie, Teratec Campus, Bruyères-le-Châtel, 91680, France; emails: {patrice.calegari, marc.levrier}@atos.net; P. Balczyński, Atos, Business & Platform Solutions, Łąkowa 11, Łódź, 90-554, Poland.

1559-1131/2019/02-ART5
https://doi.org/10.1145/3197385



Project: auditDemo
Network: metz2_ini

Name : name
Mandator
Longitude: 880 Km 383 m [✓ E] W
Latitude: 2464 Km 923 m [✓ N] S
Sectorised
Type: Standard
Height : 30 meters
Orientation : 0 0
Cost : 1.0
Diagram : omni.diag
Freq. Band : 900.C 900.C MHz
Power : 204.1: W
Threshold : -20.0

880Km 383m E 2464Km 923m N

Project: auditDemo
Network: metz1_ini

880Km 452m E 2464Km 606m N

Fig. 1. STORMS radio optimization web GUI with remote processing access to a supercomputer (1998).

& Technology (NRCPC). There is a growing number of computing centers (local, regional, national, and international) with petaflopic class systems that are increasingly used every day in many domains: biology, engineering, finance, imaging, chemistry, oil and gas, entertainment, medicine, cosmetics, manufacturing, astronomy, food industry, and so on. High-performance Computing (HPC) [116] is adopted by more and more scientists to help them solve their complex problems. All these end-users have a deep scientific knowledge in their domains but not necessarily in HPC. Moreover, even for those who have HPC knowledge, their time is better spent on their main work than on HPC questions, hence the need of tools, like web portals, to facilitate the access and use of HPC resources in an efficient way.

One of the first references to a web-based portal used for running a numerical simulation on a supercomputer can be found in the proceedings of a conference held in June 1998 [102]. During this conference, a Java applet-based Graphical User Interface (GUI) with a client–server architecture was shown for running radio wave simulations remotely on a supercomputer in Lausanne (Switzerland) from a Netscape web browser in Rhodes (Greece), and the graphical results could be visualized on a simple laptop (snapshots of this web interface "ancestor" is shown in Figure 1). This work started in 1997, within the European ACTS STORMS project [1, 80, 102] (Software Tool for the Optimization of Resources in mobile Systems, 1995–1998).

The same year, the UNiform Interface to COmputing and data REsources (UNICORE) project [71] started with the goal to develop an interface to manage batch jobs running at remote sites. The first implementation was also based on Java and Java applets.

At the same time, the web application WebSubmit [2, 100, 101], based on Common Gateway Interface (CGI) and Tcl, was developed by researchers at the U.S. Department of Commerce's National Institute of Standards and Technology (NIST). The project, introduced in 1996 [97], was intended to simplify access to HPC resources at a single site. The first public release of the WebSubmit portal was made available at the end of 1998.

These three projects had no relationship to each other, but they started the HPC portal era. Since then, the need for accessing, using, monitoring, managing, and controlling HPC resources through a simple web interface has been constantly growing. These web interfaces can be found under several names: HPC web interfaces, HPC web portals, HPC gateways, science gateways, HPC boards, and so on. For the sake of clarity and simplicity, we will only use the term "HPC portal" in the remainder of this article.

Specialized HPC portals have been developed for specific domains (e.g., e-HTPX [76] for structural biology in 2004, ENES [7] for climate modelling in 2009, CIPRES [103] for phylogenetics science in 2010, NSG [48, 81, 114] for neuroscience in 2013) or for the specific needs of large organizations (e.g., The DoD HPCMP HPC Portal [79] of the U.S. Department of Defense High Performance Computing Modernization Program). As noted in Reference [74] that presents a Java portlet approach for accessing distributed environments, many HPC-like web portals have also been set up for the Grid project among the following: The National Partnership for Advanced Computational Infrastructure (NPACI) HotPage [118], the Telescience Portal [107], the Cambridge CFD Grid Portal [120], the CCLRC e-Science Centre (now STFC) HPC Portal [119], and the UK National Grid Service (NGS) Portal [74].

In 2007, the Distributed Interactive Engineering Toolbox (DIET) [3] project implemented a Grid-aware web interface [78, 83]. It was used as a basis for the DIET WebBoard in 2009: a Java/Jsp web interface used to manage the Grid for the French medical research Décrypthon [105] project. Several HPC portals based on the DIET middleware have been (and continue to be) developed in projects for dedicated applications. It was also used by the Sysfera-DS portal [11] listed in Section 5.1.

Many reviews about Grid and HPC services provided through the web have been written: e.g., References [86], [95], and [84]. HPC portals are also addressed in more general HPC "best practices" books like [106]. A large-scale survey [93] was conducted in the USA in 2015 to *"measure the extent and characteristics of the gateway community,"* and its authors concluded that: *"[...] gateways are an active part of the science and engineering research and education landscape. Many scientists and educators depend on web-based applications to access specialized resources, particularly for computational tools and data access and analysis."* In 2018, a survey of efforts in HPC Cloud [104] reported that *"There is a considerable engineering effort to improve usability of HPC cloud, such as the creation of Web portals or simplification of procedures to allocate and access remote resources."* With the continuously growing usage of "as-a-Service/Cloud/On-Demand/Grid" services, and with the increasing need of HPC power for simulating and analyzing always bigger data, HPC portals are now used daily.

While we use terms like "HPC-as-a-Service" (HPCaaS), "Cloud," or "service provider," the HPC portals we primarily address here are web interfaces focusing on HPC end-users like scientists who submit daily computation jobs and visualize data remotely. HPC portals are different from Cloud management portals such as those proposed by Infrastructure-as-a-Service (IaaS) and Platform-as-a-Service (PaaS) providers for allocating HPC resources on a Cloud. The Cloud management feature will, however, also be addressed in this article, because it is complementary to HPC portal core features and tends to gain importance these days.

We begin this article with a use-case overview in Section 2, first discussing how HPC portals are special compared to other portals. Then we define functional and non-functional requirements that characterize HPC web-based portals in Sections 3 and 4, and we identify their mandatory, key, and nice-to-have features among these requirements. In Section 5, a brief state-of-the-art of HPC portals is presented, comparing the features of the best-known ones. One of the goals of this article is to study what HPC portals need to address in the Cloud era. In Section 6, we introduce the Bull extreme factory computing studio (XCS) project: It aims at developing an HPC portal that supports all the key features identified and discussed in the previous Sections. Then, in Section 7, XCS portal version 3 (XCS3) is taken as an example of a solution or a common thread to describe possible architecture and implementation choices that answer the HPCaaS needs. Section 8 discusses how XCS3 addresses the non-functional requirements introduced in Section 4. The purpose is to share our software engineering vision, and this could serve as an example for those who want to build their own HPC portal. Finally, Section 9 concludes this article and gives perspectives for future work.

2 HPC PORTAL USE CASES

This section presents what makes HPC portals special, what typical use cases and user categories are concerned, and which contexts HPC portals are not relevant.

2.1 What Makes HPC Portals Special?

HPC deals with both high-performance computing demands and huge computing input/output data. The relationships between the portal and these computing and storage back-ends are key and rather complex to implement in a secure and efficient way. Data processed in HPC environments can be very confidential (especially for industries or financial organizations). HPC portals have to deal with these constraints in the way they execute the services on the back-ends (through impersonation, i.e., run the service "as" an authenticated user), as well as in the way they show or filter information in the portal views (role-based displays).

Domain knowledge in HPC and in web application development is very different. The design and integration of an enterprise-grade HPC portal has to be achieved by a team with both a good understanding of HPC culture and a deep expertise in web development. The first challenge is to create such a team with both skills. Indeed, to scale and cover all the needs of a large organization or HPC service provider, several system interfaces (on both web front-end and HPC back-end) must be understood and implemented together: abstraction layers for job schedulers, HPC software and middleware interactions, web frameworks and their fast evolving libraries, authentication services, impersonation tools, remote visualization technologies, and so on. And this should be done by taking into account HPC use cases and end-users' needs.

2.2 Typical Use Cases and User Categories

When designing an HPC portal, as for any other software, use cases have to be thoroughly studied and specified. All HPC portal functions and features are not always used or even used at all, depending on the use case. Not all HPC portal users have the same objectives or use the same workflows. The same applies to organizations delivering the HPC service.

The functions and features described in this document match most of the requirements of user communities (engineers, computer scientists, researchers, scientific software developers, etc.) and their typical use cases:

- single domain computation (computational fluid dynamics, finite element analysis, molecular dynamics, financial risk analysis, animation movie rendering, etc.) pre-processing, solving, post-processing on specific applications,
- multiple domain workflows (code coupling, multi-disciplinary physics),
- parallel code development and testing,
- benchmarking (software and/or hardware).

Most typical categories of organizations delivering HPC services are

- IT departments in an industrial company,
- academic HPC solutions (schools, universities),
- regional or national computing centers,
- Cloud service providers (general purpose hyperscalers, HPC on-demand companies),
- HPC market places (e.g., Ubercloud [70] and Fortissimo [32]).

2.3 Irrelevant Use Cases and Limitations

At the time of writing, it seems that some organization types, such as some Cloud service providers, as well as HPC research and academic environments, consider that their user communities are familiar with HPC concepts and Unix/Linux Command Line Interface (CLI). Therefore, their solutions do not usually include an HPC portal. As opposed to marketplaces and companies in the industry segment, for whom user-friendly, platform-agnostic, and web interfaces are musts.

Like any other technologies and products, HPC portals cannot solve every problem and come with some drawbacks. The major reason for not using an HPC portal is the need for some users to make a significant use of a CLI, especially Unix/Linux shell terminals opened through SSH when the networking policy rules of the service provider allow it. Advanced HPC experts are often used to managing their work through CLI, and they may be reluctant to use a GUI or call web services. Typically, some of them are used to write shell scripts directly in the HPC cluster environment to partially or fully automate their workflows.

HPC portals help to unify working methods in contexts where many applications and solvers are used. This naturally implies a lack of flexibility when it comes to reproducing rich application-specific user interfaces. In a similar way, scheduler-specific and/or advanced options are not always available through portal abstraction layers.

IT and HPC administrators do not always have the web service system and security-related culture or the administration skills required to administrate web servers. Moreover, there is a risk, in terms of software durability, when customers invest for centralizing all HPC services in one single tool. Both might be issues for the acceptance of HPC portals.

3 HPC PORTAL FUNCTIONAL REQUIREMENTS

This section presents the functional requirements of HPC portals. It lists features and splits them into three categories. Throughout the article, we will use the following definitions to characterize the level of importance of the presented HPC portal features:

(1) **mandatory features:** These features are those that cannot be missing (i.e., if one of them is not supported then the software cannot be considered to be an HPC portal),
(2) **key features:** These features are those that are usually expected by users but that are not always supported. If one or several of them are missing, then certain users might consider that the HPC portal is not able to handle their application workflows, but the HPC portal can still be suitable for some use cases,
(3) **nice-to-have features:** These features do not have to be available. They are not expected nor requested, except by a minority of users, and they are not often supported by HPC portals. A nice-to-have feature is often either new or supported by another non-HPC software or web service.

However, as we will see in Section 5.2, this feature categorization has been evolving during the past 20 years and continues to evolve. So it should only be seen as a momentary snapshot of the HPC portal landscape and not as a permanent classification.

3.1 Mandatory Functional Features

The mandatory features are the minimum set of services that an HPC portal must provide. They define what a minimal viable HPC portal is.

3.1.1 Job Management. This is the mainspring of an HPC portal: managing batch scheduler jobs and the applications that these jobs execute. This feature should provide services to execute the following functions (listed by order of priority):

(1) submit (and resubmit) jobs with parameters that can be of three types:
- scheduler parameters: queue, walltime, number of cores/nodes, priority, dependency, and so on.

    - application parameters: input file, application specific tuning, version, and so on.
    - environment parameters: work directory, paths, libraries, and so on.
(2) monitor jobs: job status, application logs, result files, HPC resource usage, accounting credit consumption, and so on.
(3) terminate and suspend jobs,
(4) resume suspended jobs,
(5) modify job parameters (mainly batch scheduler parameters).

3.1.2 *Data Management.* This feature strongly relates to Job Management and can be seen as its complement. While the HPC community traditionally focuses on computing and performance aspects, producing and interpreting scientific data is the ultimate goal of HPC solutions and clusters. Data management is a keystone of the users' workflows and implies design and implementation efforts that match job-related features. Indeed, most compute jobs do process input data, write results in many different formats and generate log files. All these data need to be remotely managed by end users. So one of the main challenges for HPCaaS is to make remote data management as flexible as if it were local. The data management feature usually provides services to execute the following functions:

(1) upload and download files,
(2) preview file content,
(3) copy, move, rename, delete files,
(4) compress and uncompress archive files,
(5) browse user data spaces (restricted to users' privileges),
(6) manage file ACLs (Access Control List),
(7) monitor quota(s).

A minimal set of these services was considered sufficient when the first HPC portals showed up in the HPC community (see Section 5), but nowadays users expect to have all of them.

3.2 Key Functional Features

In addition to the mandatory features listed above in Sections 3.1, the functional features listed below turn out to be key. The key features are linear satisfiers that customers and users expect to be supported by an HPC portal even if they are not mandatory (i.e., a minimal viable HPC portal does not need to support them). The list of key features has naturally grown from our users' and customers' requests since 2004 and also from the observation of the features supported by all the HPC portals we have seen since we work in this field. This list will continue to evolve with the customer needs and expectations, and some of the nice-to-have features listed in Section 3.3 might be added to it in the future.

3.2.1 *Multi-tenancy.* When an HPC solution is meant to be used and shared by several customers while preserving maximum security (i.e., groups of concurrent users from different companies, possibly competitors), or by researchers of different entities, it has to come with means to

- strictly isolate these customers in separate environments (named tenants),
- manage clusters, queues, applications, users, projects, directory services, and so on, per tenant,
- aggregate, account, and report each entity resource usage.

The multi-tenancy feature is a must have for any Cloud (or Cloud-like) solution and is especially complex for HPC services when interconnects, containers, and virtualization come into play.

3.2.2 Multi-cluster. The Multi-cluster feature allows users to manage several HPC clusters simultaneously from a unique portal. It is key for compute centers that operate several clusters in a unique location.

3.2.3 Multi-scheduler. All HPC clusters rely on a job scheduler, a.k.a. batch system, that is responsible for controlling (queuing, scheduling, etc.) unattended execution of jobs. An HPC portal should be compatible with most of them (e.g., Slurm, Altair PBS Pro, IBM LSF, SGE/OGE Sun/Oracle Grid Engine, Torque, OAR). The multi-scheduler compatibility feature makes it possible for an HPC solution to seamlessly integrate existing and new environments so that a single portal instance can be used no matter what the cluster management systems look like. Moreover, it permits the transparent replacement of a scheduler by another: As the scheduler type is hidden by the portal, users do not have to care about its specificities.

Together, the Multi-cluster and Multi-scheduler features allow the administrators to manage simultaneously several HPC clusters running different schedulers (e.g., an old system, a test/validation cell, and a production system). Both features are extremely important to ensure that an HPC portal can become widely used in the long term.

3.2.4 Multi-directory Service. This is the ability for a portal to identify users against one or more directory services. For instance, when multi-tenancy is required, these tenants are configured to identify against distinct directory servers. Typical directory services used in HPC environment are: Lightweight Directory Access Protocol (LDAP), Network Information Service (NIS), and Active Directory (AD).

3.2.5 Remote Visualization. The remote visualization feature makes it possible for users to execute their interactive two-dimensional (2D)/3D graphical applications on remote servers instead of their local host and to open visualization sessions to remotely control them. All the graphical application computing and rendering, including 3D rendering, takes place server-side on GPU-enabled nodes. The user's mouse and keyboard inputs are transferred to the server node that, in return, encodes the graphical scene into pixel frames and sends them as video stream back to the client's host. The client is then only responsible for the rendering of the 2D video stream.

The first motive is to make it possible for users to only work on remote data without having to transfer them back and forth between local host and servers. Typically, computation result files can be very large (gigabytes to terabytes) and cannot be efficiently transferred through regular internet or enterprise private networks.

The second motive is to share graphical sessions with users who collaborate from different locations, each of them being able to interactively work on the same display.

Another motive is intellectual property protection. Remote visualization can be used to increase data security and confidentiality in some cases. Indeed, HTTPS and related protocols are well accepted by IT security officers, and easy to secure in firewalls and monitor as opposed to SSH that many organizations do not accept to provide end-users with. Together with role-based approach (see Section 4.1.3), remote visualization can prevent certain categories of users from physically accessing and transferring data while still being allowed to remotely pre- and post-process it. Such users can interact with the data on their screens, but they cannot download or copy the huge files that include intellectual property, because only video content (pixels) are actually sent to the client host, not original data.

3.2.6 HTTP RESTful API. The possibility to fully control web services through an Application Programming Interface (API) without requiring a browser is a plus to integrate a web portal approach into existing workflows or software with non-web interfaces (graphical or not). REpresentational State Transfer (REST) is an architectural style often used for web development of

APIs [112]. RESTful (or REST) web services communicate with standard HyperText Transfer Protocol (HTTP) methods such as `GET, POST, PUT, DELETE`. They involve collections of resources (e.g., jobs, users, applications, cluster, etc.) with identifiers, for example, `/jobs/123` for job with ID number 123. Resources can be operated upon using standard verbs, for example, "`GET /jobs/123`" to get information about job 123 or "`DELETE /jobs/123`" to terminate this job. The important point is that RESTful requests are stateless by nature: No state is saved, and each transaction is independent. This ensures robustness and scalability. RESTful APIs are now the standard for service oriented architectures (SOA), business-to-business (B2B) interfaces, and online marketplaces. A W3C HPC Community Group was created in 2012 to work on APIs that expose HPC resources via the web [12], but their activity seems stopped (their website has not been updated since 2014). Several RESTful APIs were developed for HPC (almost one for each HPC portal listed in Section 5.1): NEWT API (NERSC) [49], Lorenz API (LLNL) [98], CyVerse science APIs [26], Nimbix API [40], ActiveEon ProActive API [14], XCS API [31] (see Section 7.3), and so on. To keep up with the growing demand for service-oriented architectures and micro-services, this feature is key.

3.3 Nice-to-have Functional Features

The nice-to-have features are delighters that customers and users do not expect but that could be good differentiators for the success of an HPC portal. Here is a non-exhaustive list of such features:

3.3.1 Workflow Engine. A Workflow Engine is used to control the execution of multiple jobs with dependencies and conditions. To let the users define their complex job workflows, it should handle the following:

- any required data movement to ensure that the output of a job can be used as input by any subsequent jobs,
- control constructs such as loops and conditional tests on job variables,
- job dependencies.

3.3.2 User Data Archiving. Once input data have been frozen and computation results have been produced and validated, they are no longer modified and their size is generally large (gigabytes to terabytes per job). Keeping such data volume for a long time on an HPC cluster parallel file system or file storage is expensive, and actually not manageable, because its size keeps increasing over time. Therefore, the best solution for long-term retention is archiving, which comes with extra features such as legal archiving regulation support. Demand for HPC user data archiving is growing, especially in HPCaaS contexts where data are not managed locally anymore. The archiving functions can be made available directly from HPC portals (e.g., through external API calls).

User data archiving should not be mistaken with "user data backup": most organizations have general-purpose disk-based, tape-based, or even Cloud storage-based backup systems and related software to backup user and system data. These use classical backup policies such as daily increments, weekly full backups, and so on. However, in most use cases, this is not necessarily relevant for HPC data being processed. For instance, for a given HPC computation, input data come from storage systems that are usually backed up using backup-dedicated solutions, and relevant result files may be archived on safe storage systems for tracking or legal purposes (which topics are out of the scope of our study). It is commonly accepted in HPC that backup of large result files is not considered critical, since the consequence of a hypothetical data loss would be negligible: Indeed, input data can be retrieved from their original storage, and result data can be recomputed if necessary, the cost of a rare data re-computation being much lower than the cost of a systematic data backup.

3.3.3 Cloud Management. Cloud bursting or hyperscaling (i.e., short-term allocation, provisioning and release of external HPC resources) is more and more seen as an interesting opportunity to extend on-premises HPC solutions without further hardware investments [104]. It makes this extension possible through an OPerational EXpenditure (OPEX) expense model, which is often considered more flexible by financial authorities as opposed to CAPital EXpenditure (CAPEX). Cloud management tools and web portals do exist and are used to orchestrate nodes (usually Virtual Machines), deploy operating systems, install applications (usually packaged in containers), and administrate HPC clusters on Cloud platforms. These functions may be made available from an HPC portal directly. One reason for this is to present a unique point of access for all HPC related services. The HPC portal should then securely handle a large number of concepts related to the use of resources in the Cloud such as the user or/and the customer credentials to access the Cloud provider platform(s), credits, VPN/security groups, cluster provisioning playbooks/recipes, and so on.

4 NON-FUNCTIONAL REQUIREMENTS

The non-functional requirements are the technical characteristics that make an HPC portal suitable for sustained production. They should be optimized as much as possible for the global quality of the product. This section presents such non-functional requirements and their relevance to HPC portals.

4.1 Security

Security is a major concern for all types of web interfaces, especially when it grants access to a significant amount of confidential resources and R&D data. It should not be seen as a "feature" but rather as a mandatory quality that is part of a whole policy, including software design and implementation as well as installation and setup rules, system configuration guidelines, good user practices, and network security monitoring. One key aspect, among many others, is that a web server runs as a single system user with little or no privilege (such as *nobody*, *portal*, etc.) and needs a safe impersonation mechanism to trigger the execution of a task on behalf of a real user account. Figure 2 shows the security layers that content data (files, requests, video streams, etc.) and credentials have to go through between an HPC portal user and a remote HPC cluster. Each layer must be addressed without inducing any breach in the security chain. In this context, an HPC portal is mainly responsible for authentication (see Section 4.1.2) and authorization (see Section 4.1.3) controls. External accesses need to be secured by using HTTPS certificates and cryptographic protocols such as Transport Layer Security (TLS) defined in RFC2246 [77] in 1999 and updated in RFC6176 [109] in 2011.

4.1.1 Physical Security. From a general point of view, user data and thus user activity is often more secured in data centers than on laptops or on local workstations (e.g., in SMEs or home offices that are not always properly secured). The security/confidentiality is ensured by the physical protections of the data centers together with the web services and HPC portal software intrinsic security properties.

4.1.2 Authentication. Authentication is the mechanism that allows a system to securely ascertain that a user really is who he/she claims to be. In other words, the authentication process checks credentials (e.g., password, token, etc.) to prove the identity (e.g., login name) of a user.

The importance of authentication for HPC portals depends on their organizational contexts and technical environments. Typically, a self-made, in-house developed solution for a single unit on a single location inside one organization may not require enterprise-grade authentication.

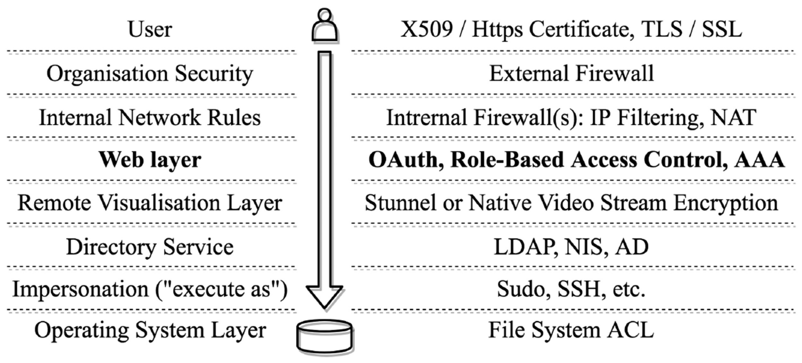

User
X509 / Https Certificate, TLS / SSL
Organisation Security
External Firewall
Internal Network Rules
Intrernal Firewall(s): IP Filtering, NAT
Web layer
OAuth, Role-Based Access Control, AAA
Remote Visualisation Layer
Stunnel or Native Video Stream Encryption
Directory Service
LDAP, NIS, AD
Impersonation ("execute as")
Sudo, SSH, etc.
Operating System Layer
File System ACL

Fig. 2. Security layers for HPC portals.

Conversely, HPCaaS solutions provided by major compute centers need to implement sophisticated and reliable authentication systems.

In case an HPC portal solution has to deal with several flavors of authentication systems, two angles must be considered:

- the list of authentication technologies to be supported to cover target use cases,
- the degree to which an HPC portal needs to take part in the authentication process.

Possibilities range from the portal authenticating users with its own user and password database to entirely delegating authentication to operating system-level or external services such as directory services and/or Single-Sign-On (SSO) services.

The ability to support a large range of external authentication services is a logical goal to be compatible with as many compute center security policies as possible. But having this configurable, flexible, and maintainable turns out to be complex at many levels: a large range of technologies, operating system specific implementations and behaviors, support of organization-specific credential management policies, compatibility with Java-based portals, and so on.

Due to the complex nature of HPC workflows and execution chain, there is an increasing focus on Kerberos network authentication protocol [43, 115]. In computing centers where the environment is fully controlled by Kerberos, users are granted “tickets” by the Kerberos server, allowing them to access the HPC resources without typing their login/password multiple times or having services passing passwords to each other over the network. They are automatically authenticated all along the execution chain as long as the ticket has not expired.

Kerberos is a key feature for an HPC portal product, because it is used by most of the largest HPC computing centers and large companies. For these organizations, the HPC portal Kerberos support is absolutely mandatory, since in a Kerberized environment each and every service including the portal must be protected with Kerberos tickets.

Another aspect is that Microsoft AD, which is a standard in large industrial companies, relies on Kerberos as well. From a Linux-based HPC portal solution, a possible gateway between both worlds is to go through

- Linux SSSD (System Security Services Daemon) [67] based on RFC2307 [90], and
- GSS-API (Generic Security Service Application Program Interface), the Kerberos wrapper published as RFC1508 [96].

4.1.3 Authorization. Authorization is the mechanism that allows a system to determine what an authenticated user is allowed to do. It checks the level of user permissions to access (e.g., read, write, create, delete) secured resources (e.g., files, jobs, clusters, applications, etc.).

To enable, disable, or adapt content to be managed and displayed in accordance with security policies and authorization mechanisms, roles must be assigned to all users. Role-Based Access Control (RBAC) is a permission model that allows fine-grained access control management: It grants user privileges that exactly match the set of features users are allowed to use and information they are allowed to see. How roles are configured to grant access to content and services is controlled by the Authentication, Authorization, and Accounting (AAA) security framework.

AAA is a security framework that aims at controlling access and usage of computer resources with policies. Each and every access to a feature, function, resource, and/or view is configurable, checked against a role, and traceable in AAA. As a consequence, this allows administrators to easily modify user authorization scopes without requiring any modification in the portal source code.

4.1.4 Accounting. Accounting is the mechanism that allows a system to track, record, and store the activities of users once they are authenticated and authorized. The recorded information can be used for

- accountability to identify users and determine their responsibility for past activities, and/or
- billing to charge users based in the resources used (data accessed, data transferred, compute, storage, applications, portal features, etc.) in terms of duration (e.g., computation elapsed time, pure CPU time, access time, etc.), quality (e.g., computation precision level, solution accuracy, confidentiality level, SLA, etc.), and size (e.g., number of nodes, memory size, file size, CPU power, etc.).

4.2 Usability

An HPC portal, like any software with a user interface, should be easy to use, simple, clear, intuitive, and responsive. The usability of HPC portals mainly depends on the following properties:

- **Operability.** An HPC portal should be easy to operate and control. In other words, its functions should be clearly defined as well as easy to find, execute, and monitor. Its interface should be as simple as possible while offering all the expected functions. For example, it should ensure that the number of necessary interactions (e.g., the number of necessary "clicks") to achieve a desired effect (e.g., a job submission) is reduced to a minimum.
- **Learnability.** HPC portal users should be able to learn how to use it with effectiveness, efficiency, freedom from risk, and satisfaction in the context of their work. In other words, the user interface should be intuitive and flexible enough to adapt to all business workflows.
- **User interface aesthetics.** HPC portal GUI aesthetics depends on the graphical design of the web pages (colors, element shapes, logos, fonts, etc.) and the layout of their functional web elements (frames, buttons, menus, etc.). Aesthetics is important for IT administrators and Cloud providers, since the portal web pages present the brand image of their data center, university, or company to their internal users and/or external clients. Aesthetics is also important for users who should feel pleased and satisfied when interacting with the GUI. For example, users should not be disturbed or distracted by any visual element.

To fulfill these requirements, the following features are key.

4.2.1 HPC Application Template Framework. This is a simple and documented tool-set aiming at providing administrators with the ability to create/update job submission web forms (application "publication") and job submission scripts (that are run once the form is submitted). Ideally, a single template model should cover all possible cases. In this case, job submission methods are unified and homogeneous, no matter what the applications do and look like.

4.2.2 Customizable GUI. Each user is unique and has unique requirements and constraints in his work. The GUI should be highly customizable to be able to easily present any user workflow and allow its execution. The portal administrators and the end-users should be able to create/modify/save/share the user interface layout and look&feel according to their needs. Together with RBAC, a customizable GUI makes it possible to fulfill many divers requirements from different user communities with a unique portal instance.

4.2.3 Responsive Web Design (RWD). A user interface with RWD makes it possible to optimize the display of web GUI elements according to the display geometry and resolution of the end-user's terminal. Typically, it automatically re-dispatches elements of dashboards depending on which device the web application is accessed from: workstation with large screen, laptop with small screen, tablet, smartphone, and so on.

4.3 Performance

HPC portals have to satisfy HPC specific constraints that do not exist for other web portals. An HPC portal is intended to hide and replace the user interaction with job schedulers, taking into account that job schedulers were designed to be driven through the command line by humans and not to receive a large amount of requests from a web application. On the front-end, hundreds (or even thousands) of users want to follow the status of their jobs and they might refresh their web page very often. On the back-end, an HPC batch scheduler handles thousands (or tens of thousands) of jobs and returns information without any time constraint. This implies that users want to have an up-to-date status instantly while on the back-end a call to the scheduler server can take several seconds (or minutes when hundred of thousands of jobs are monitored) before returning a complete status. This is different from a standard website where millions of users send requests about data that are immediately available whether it changes at a high or low frequency rate.

Performance requirements for an HPC portal are not more challenging than for a social media website, but it has to deal with a major bottleneck: the job scheduler (see Section 3.2.3). Since the primary interface of HPC portals with the cluster is the batch system service, and since some batch systems are synchronous and/or weakly threaded, they can be counter-responsive when loaded by a large number of requests. The main consequence is a risk for the web portal and the service to get overloaded and/or stalled.

Because of these near-real-time constraints, an HPC portal web server should behave like a cache: It should gather information from the HPC cluster(s) as quickly as possible (knowing that the scheduler response time is a limit) and it should broadcast the latest known information to the users independently from the information gathering frequency. If user update requests were directly sent to the HPC cluster, then the scheduler server would quickly be overloaded, and hence there would be a serious degradation of the service for all users (even those working locally on the HPC cluster without using the portal).

4.4 Reliability

Since the goal of an HPC portal is to provide a service to a potentially large number of high-profile users, such service requires protection against data loss, data corruption, as well as downtime and system failures. However, this protection does not rely on the portal implementation but rather on adequate IT practices, processes, and external tooling. This includes backup, High Availability (HA)/Fault tolerance, and Disaster recovery.

4.4.1 Portal Data Backup. As already touched upon in Section 3.3.2, most organizations have general-purpose backup systems and related software. They can be used to back up data and systems involved by HPC portals. This can include user profiles, HPC cluster setup configurations (servers, services, and database), history, accounting data, and so on.

4.4.2 HA/Fault Tolerance. HPC web portals are more and more considered as important services, since downtime often impacts a large number of high-profile users. This becomes critical in commercial HPC-as-a-Service offers where Service Level Agreement (SLA) may be governed through contracts between service providers and customers.

4.4.3 Recoverability. Disaster recovery is the ability for services and related user data to survive a major accident, such as the physical destruction of a data center and IT infrastructure on a given site (resulting from any kind of major hazard). Most of disaster recovery plans (DRP) involve elaborate and expensive data center-to-data center high-performance networking, clustered service mechanisms, and data replication mechanisms. From our experience, disaster recover plans covering the HPC scope are quite rare, since HPC is in general not considered a critical activity in most organizations (as opposed to ERP, CRM, commercial services, and the like). Large organizations owning more than one data center sometimes split their HPC infrastructure in one or more parts for this reason. The HPC portal can then just be considered as any other service/software in the DRP.

5 STATE-OF-THE-ART OF HPC PORTALS

This section reviews the list of the most significant multi-domain (as opposed to domain-specific) HPC web portals. It presents a comparison matrix of their features and deduces some trends from it.

5.1 HPC Portal List

Numerous HPC portals have been developed in the past years. The most significant multi-domain ones are listed in Table 1. This list was built by exchanging with HPC portal developers and distributors by gathering information from our HPC clients and, finally, by asking to all of these contacts if they knew about any other "unlisted" HPC portal to be sure that none of the significant ones was forgotten. All of the 24 listed HPC portal products are (or have been before being discontinued) available for internal on-premises usage.

Many HPC projects are developing their own community-specific web portal from scratch or based on open source resources. An example is given by the EasyGateway team in Reference [87] who developed a science gateway for meteorological forecast based on #2 Apache Airavata. Listing all those project-specific or community-specific portals is, however, not the purpose of this review.

Some HPC Cloud providers have developed web portals for their own usage (e.g., Penguin Computing Scyld Cloud [62], Sabalcore [36], and Portal). With these web portals, job and file management is done through a Command Line Interface (CLI), unlike the listed HPC portals that use a GUI. Moreover, these web portals are not sold, rented, or open sourced, so they cannot be installed on-premises. That is why we did not list them in Table 1.

A new type of HPC portal exists for web-based HPC applications. For example, the MIT SuperCloud portal workspace [110] provides authentication, encryption, and access control to enable the secure exposure of web services running on HPC systems. Such a portal acts as a reverse proxy for accessing web applications ran as HPC jobs by end-users. It is worth citing this new use case

Table 1. List of the Major HPC Portals Available for Internal On-premises Usage

	HPC portal name (with year of first release)	Comments
#1	Agave ToGo [16] with CyVerse Science APIs [26] (2013)	Open source
#2	Apache Airavata [18, 99, 108] Django Portal (2003)	Open source
#3	Compute Manager [23] by Altair (2009)	Replaced by #14 in 2018
#4	eCompute [4] by Altair (2003)	Replaced by #3 in 2009
#5	eBatch by Serviware/Bull (2004)	Replaced by XCS1 in 2011
#6	EnginFrame [28] by NICE/Amazon (1999)	
#7	HPCDrive [10] by Oxalya/OVH (2007)	Discontinued in 2015
#8	HPC Gateway Appli. Desktop [13, 37] by Fujitsu (2015)	
#9	HPC Pack Web Components [47] by Microsoft (2008)	On Microsoft Windows only
#10	JARVICE Portal [40] by Nimbix (2012)	
#11	MOAB Viewpoint [15] by Adaptive Computing (2006)	
#12	Open OnDemand [33, 91] (2017)	Open source
#13	Orchestrate [54] by RStor (2018)	
#14	PBS Access [56] by Altair (2018)	
#15	Platform Application Center (PAC) [8, 111] by IBM (2009)	
#16	ProActive Parallel Suite [14] by ActiveEon (2014)	Open source
#17	Sandstone HPC [113] (2016)	Open source
#18	ScaleX Pro [61] by Rescale (2012)	
#19	SynfiniWay [9, 89, 117] by Fujitsu (2010)	Replaced by #8 in 2015
#20	Sysfera-DS [11] by Sysfera (2011)	Discontinued in 2015
#21	UNICORE Portal [71, 85] (project:1997, v7:2014)	Open source since 2004
#22	WebSubmit Portal [2, 97, 100, 101] by NIST (1998)	Last update in 1999
#23	XCS1/XCS2 [30] by Bull/Atos (2011/2014)	Replaced by #24 in 2018
#24	XCS3 (described in Sections 6 and 7) by Atos (2017)	

that is the result of the convergence of traditional HPC, Big Data, and Cloud. However, portals of this type were not included in the list, because they are too different from those we discuss in this article.

The HPC portal landscape evolves extremely quickly. Hence, 4 of the listed portals have had a short life and are already discontinued: #4 eCompute and #19 SynfiniWay were replaced by new products while #7 HPCDrive and #20 Sysfera-DS have undergone company closing. And 2 other portals are being replaced in 2018: #3 Compute Manager, and #23 XCS2.

Major Cloud players have HPC offerings (e.g., Microsoft Azure Big Compute [46], Google Cloud Platform (GCP) HPC [35], and Amazon Web Services (AWS) HPC Portal [17]) that should not be mistaken for "HPC Portals" as addressed in this article. However, these Cloud offerings are compatible with most of the HPC portal products listed in Table 1, and some of them can propose their own HPC portal solutions. Microsoft has a native HPC portal (#9 HPC Pack Web Components) for Windows clusters and, in August 2017, it acquired Cycle Computing with its CycleCloud web GUI [25] aimed at managing HPC cluster in the Cloud. In February 2016, Amazon acquired NICE with its #6 EnginFrame HPC portal, which powers CloudFrame, a Self-Service Provisioning portal. In spite of these acquisitions and product discontinuations, in the end, at least 16 different actors are active in the HPC portal domain, which is a lot for such a niche market. In Table 1, we see that from 2003 to 2018 there has been at least one new actor distributing a new HPC portal every year (except in 2005 and 2015). This proves that this domain is very dynamic and important at the moment.

It can be noticed in Table 1 that 6 HPC portals are open sourced:

- #1 Agave ToGo that started in 2011 as a pilot project for the iPlant Collaborative.
- #2 Apache Airavata that had a long history within academia at the Extreme Computing Lab at Indiana University since 2003, before its new home in Apache software foundation where its architecture was revamped.
- #12 Open OnDemand that is based on the Ohio Supercomputer Center's OSC OnDemand platform [52],
- #16 ProActive Parallel Suite whose core technology [82] was initially developed by a team of 45 developers at INRIA (French institute for Computer Science), and that has been extended by ActiveEon, a spin-off from INRIA created in 2007.
- #17 Sandstone HPC that was developed at University of Colorado Boulder in 2017.
- #21 UNICORE Portal that started in 1997 within the framework of the UNICORE project, a research project funded by the German Ministry of Education and Research, and that was open sourced in 2004.

Moreover, two other portals, from the early years, had their sources available to customers/users:

- #5 From 2004 to 2011, eBatch portal source code was provided to customers who were free to modify it, provided that they maintained their modified version on their own.
- #22 WebSubmit was mainly used within NIST organization for internal needs, but the source code could (and still can) be freely downloaded at Reference [2]. The code was updated for the last time in July 1999.

Most of the widely used HPC portals are commercial products for which it is difficult to get accurate information about number of instances, users, applications, and implementation details. Customers are anyway not likely to publish such information. We gathered all the public information we could find to write this short review, but we could not get all the detailed information we would have liked to present. However, this review is broad enough to give a rough idea of the main trends in the domain and to serve as a basis for the study presented in this article.

5.2 HPC Portal Review

Table 2 shows a comparison matrix of HPC portal features. It compares the features of the latest release of each HPC portal listed in Table 1. By definition, mandatory features introduced in 3.1 are supported by all HPC portals so they are not shown in the comparison matrix. Additionally, the non-functional requirements (security, usability, performance, and reliability) discussed in Section 4 would need extensive qualitative benchmarks to be completely and fairly compared: Such significant work is out of the scope of this article. We chose to compare the key and nice-to-have functional features listed in Sections 3.2 and 3.3, as well as some features identified among the non-functional requirements and whose support is easy to check: three for security authorizations (introduced in Section 4.1.3), and three for usability (introduced in Section 4.2).

The matrix was filled in with information coming from publicly available documents (scientific articles, brochures, product websites, etc.) and from HPC portal users, developers, and distributors.

In Table 2, we can see that only 2 HPC portals offer all of the 15 features (#8, #16), and that 2 others offer all of the 12 key features (#6 and #24) selected for this study. When we only look at the 16 currently available HPC portals (i.e., not being replaced or discontinued), we observe that

Table 2. Comparison Matrix of HPC Portals Key and Nice-to-have Features

	Key functional features						Nice to have functional features			Key non-Functional features					
	Multi-tenancy	Multi-cluster	Multi-scheduler	Multi DS	Remote viz	HTTP RESTful API	Workflow engine	User data archiving	Cloud management	RBAC	AAA	Kerberos	HPC application templates	Customizable GUI	RWD
#1 Agave ToGo	✓	✓	✓	✓	✗	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓
#2 Apache Airavata™	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
#3 Compute Manager[14]	✓	✓	✗[1]	✓	✓	✓	✓	✗	✗	✓	✓	✓	✓	✓	✗
#5 eBatch[14]	✗	✓	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✓	✗	✗
#4 eCompute[14]	✓	✓	✗[1]	✓	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✗
#6 EnginFrame	✓	✓	✓	✓	✓	✓	✗	✗	✚[2]	✓	✓	✓	✓	✓	✗
#7 HPCDrive[14]	✓	✗	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓	✗	✗	✗
#8 HPC Gateway	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
#9 HPC Pack Web Com.	✓	✗	✗[3]	✗	✗	✓	✓	✗	✓[4]	✓[5]	✓[5]	✓[5]	✗	✗	✗
#10 JARVICE	✓	✗	✓	✗	✓	✓	✓	✓	✓[6]	✓	✓	✗	✓	✗	✓
#11 MOAB Viewpoint	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓
#12 Open OnDemand	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✗	✓	✗	✓
#13 Orchestrate	✓	✓	✓	✓	✓	✓	✗	✗	✓	✓	✓	✗	✓	✓	✓
#14 PBS Access	✓	✓	✗[1]	✓	✓	✓	✓	✗	✚[7]	✓	✓	✓	✓	✓	✓
#15 PAC	✓	✓	✗[8]	✓	✓	✓	✓	✚[9]	✓	✓	✓	✓	✓	✓	✓
#16 ProActive Parallel S.	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
#17 Sandstone HPC	✓	✓	✗	✓	✗	✓	✗	✗	✗	✗	✓	✓	✗	✗	✓
#18 ScaleX™ Pro	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✗[10]
#19 SynfiniWay[14]	✗	✓	✓	✓	✗	✗	✓	✓	✗	✓	✓	✓	✓	✗	✗
#20 Sysfera-DS[14]	✓	✓	✓	✗	✓	✗	✗[11]	✗	✗	✓	✓	✗	✓	✗	✓
#21 UNICORE 7 Portal	✓	✓	✓	✓	✗	✓	✓	✗	✗	✓	✗	✗	✓	✓	✗
#22 WebSubmit[14]	✗	✗	✓[12]	✓	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗
#23 XCS2 [14]	✓	✓	✓	✓	✓	✗	✗	✓	✗	✓	✓	✓	✓	✓	✗
#24 XCS3	✓	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓	✓[13]	✓	✓	✓

Rows of HPC portals being replaced or discontinued are grayed. (✓ = Supported ✗ = Not supported ✚ = Indirectly supported via an external software/portal or through compatible web services)
[1]Altair PBS Professional (commercial or open source) scheduler only. [2]Supported by CloudFrame and MyHPC solution on AWS only. [3]Microsoft HPC Pack scheduler only. [4]On Microsoft Azure only. [5]Through Active Directory services. [6]For Nimbix Cloud management. [7]Supported by PBS Control. [8]IBM Spectrum LSF scheduler only. [9]Supported by IBM Spectrum Protect. [10]Under development. [11]Supported by most releases but not by the last one. [12]LSF scheduler and NQS (Network Queuing System) only. [13]With XCS v3.7 and higher. [14]Replaced or discontinued.

- all of the 15 features are supported by at least 50% of them,
- the most implemented features (i.e., supported at least by 15 portals) are Multi-tenancy and HTTP RESTful API, as well as the AAA and RBAC security features,
- the least implemented features (i.e., supported by less then 10 portals) are Multi-scheduler, Remote visualization, Kerberos, and, not surprisingly, all the nice-to-have features.

When we compare the features supported by the currently available HPC portals with those of the replaced or discontinued ones, we observe that three of them were almost not (or not at all) supported in the past: HTTP RESTful API, Cloud Management, and RWD. HTTP RESTful API is now supported by almost every portal. Nowadays, RWD is supported by every modern GUI software. The Cloud Management feature is now available with most of the new portals (10 directly and two through an external software/portal), which shows a growing interest for HPC usage in the Cloud. These three points prove that the needs for the features evolve quickly over time.

6 THE EXTREME FACTORY COMPUTING STUDIO PROJECT

This section presents the eXtreme factory Computing Studio (XCS) portal. It first gives an overview of its history and then lists the functional features of its current version (v3).

6.1 Brief History

We had a first experience in HPC portal development around 2004. We had to face an increasing demand for web front-ends to HPC solutions so we built eBatch, a simple Tomcat/JSP (Java Server Pages)/Servlet based portal aiming mostly at hiding job scheduler and job submission complexity from users. The other main goal was to provide HPC services across an enterprise WAN where SSH connections were not always allowed. It was flexible enough to rapidly adapt to many site-specific HPC implementations, job schedulers, and job submission scripts. However, many enterprise-grade features were missing: state-of-the-art authentication mechanism, remote visualization, accounting, multi-tenancy, and so on.

In 2008, we decided to leverage this experience and to develop an enterprise-grade HPC portal including remote visualization capabilities, which turned out to be the missing piece enabling 100% remote HPC user experience. The new project was named eXtreme factory Computing Studio (XCS).

6.1.1 XCS1. The XCS project started in May 2009. For the first portal implementation, we selected the portal framework Liferay Portal [44] (against eXo [53]), because it included some ready-to-use features that we needed (users/groups management, content management with look customization, LDAP integration). Moreover, Liferay had been declared “Best open source Enterprise Portal” by InfoWorld [5] in 2008 and was very well ranked when compared to other portal frameworks (see Reference [75]). For the core engine that handles the interactions between the web environment and the HPC cluster (i.e., scheduler), we chose Globus [58] for the same reasons: It had ready-to-use features that we needed (multi-scheduler support, etc.) and was popular. Our main goal was to accelerate the Time To Market (TTM) for our portal and this goal was successfully reached in May 2011, when XCS release 1.0 was announced and started to be used at our facilities.

At the beginning, the XCS portal has been developed for our own HPC on-demand offering exclusively. But we quickly understood that other computing centers wanted to have their own portal on-premises, so we packaged it as a product.

Then we started to learn from our customer experiences and user feedback, and we improved this first release: We replaced Globus by our own “cluster integration layer,” because Globus was too complex for the simple tasks we needed and too difficult to maintain. We drastically reduced the number of steps required to execute actions from the portal (submit jobs, download files, etc.), and we added support for remote visualization tools (e.g., XRV [30] or TurboVNC [69]). The result was XCS2.

6.1.2 XCS2. The XCS2 portal has been in production on a few tens of sites, with a few hundred HPC applications integrated and used by a few thousand users. Over the years, its code grew, and most of the Liferay features were replaced with custom implementations (e.g., to support NIS in addition to LDAP). This required the creation of a duplicated user database that had to be maintained and synchronized. That also made development team responsible for third-party libraries and it slowed down development of new features. Finally, it turned out that Liferay content management feature was almost not used and lacked flexibility for XCS2 user pages customization. So, in 2015, even though XCS2 was satisfactory, we decided to refactor it thoroughly with modern concepts in mind.

XCS2 user interface was lacking support for many new web trends (e.g., GUI edition and customization) desired by customers and it was difficult to maintain it as a monolithic (i.e., not modular) software. The RESTful API added in latest versions of XCS2, even though simple and only end-user oriented, proved its usefulness. These facts motivated us to propose a new modular approach.

6.1.3 XCS3. The XCS2 RESTful API was extended with all missing endpoints, including all administrative ones, and it became an independent application: the "XCS3 Java Core RESTful API web application," upon which we started to build the new web GUI. XCS3 GUI is considered as an application GUI rather than as an enterprise portal. We established the following high-level specifications:

- **Architecture:**
  - a modular architecture,
  - a modern design pattern appropriate to the MVW (Model-View-Whatever) technology,
  - an independent user account management (i.e., not imposed by the framework).
- **Implementation:**
  - a sustainable web framework,
  - an easy to maintain and upgrade software with smooth maintenance effort,
  - an extensive list of supported platforms for the server side,
  - an extensive list of supported browsers for the client side (i.e., the latest versions of Chrome, Firefox, Safari, Internet Explorer, and Edge).
- **Features:**
  - support of all the mandatory and key functional features listed in Section 3.1 and 3.2,
  - fulfill the non-functional requirements listed in Section 4.

We made a comparative case study of the latest web development frameworks (e.g., AngularJS [6], React [57], extJS [29], Ember [27], etc.), involving leaders of several similar projects. AngularJS was finally selected as main framework for a new XCS3 user interface web application, because it was one of the framework providing all the necessary features and it had a large backing community: It had a high chance to be a sustainable choice, because many companies were investing in its development, a lot of web applications were based on it, and its popularity was making it widely known among new developers.

XCS3 has been designed with all the functional features listed in Section 3 in mind. This induced the need to support two user experiences: handling batch jobs and handling interactive jobs (typically remote visualization sessions). XCS3 design was also driven by security, efficiency, and usability to best answer the non-functional requirements listed in Section 4. It directly impacted design and technology choices as described in Section 7.

All the knowledge gathered during XCS2 software life was extremely valuable for the new developments but less than 15% of its original code was kept, so XCS3 is really a brand new software.

XCS3 has been released mid-2017 (at the time of writing, the latest release is v3.7). It is used for production on several sites and has already been tested with more than 60 HPC simulation applications (ANSYS Fluent and CFX, CD-Adapco StarCCM+, NUMECA Fine/Marine, Principia Deeplines, OpenFoam, LSTC LS-DYNA, ESI VPS / PAM-CRASH, Gromacs, Namd, Blender, Dassault Systèmes SIMULIA CST Studio Suite, Exa PowerFLOW, etc.), as well as visualization software (Paraview, Ensight, native GUIs of the applications listed above, etc.). So far, no limitations were found with the XCS 3 application integration framework: All applications requested by users could be integrated and exposed in the portal. In a general manner, any new batch or interactive application can be easily exposed and executed (as explained in Section 7.5).

6.2 XCS3 Functional Features

XCS3 aims to be used both for customers on-premises and for HPC on-demand Cloud services delivered by our data center, third-party providers, or organization computing centers. It supports all the mandatory and key functional features introduced in Sections 3.1 and 3.2. Here are some details about the way these features are provided:

- **Job management** (see Section 3.1.1)
  XCS3 comes with the ability to create and edit web forms that present all the necessary job submission parameters for a given HPC application. Jobs can be submitted through these web forms. They can also be monitored (status, resources used, etc.), detailed (parameters, etc.), resubmitted, and terminated.
- **File management** (see Section 3.1.2)
  XCS3 file browser presents to users the mount points (user-browseable data spaces) selected by the Portal or Customer Administrator. Files can be uploaded, downloaded, copied, moved, deleted, compressed, and uncompressed. They can also been previewed with contextual handling (through file extension) of most major media formats such as texts, formatted documents, images, videos, and audios.
- **Multi-tenancy** (see Section 3.2.1)
  A single XCS3 portal instance supports the management of several distinct organizations (companies, entities, etc.) making it suitable for HPC service/Cloud providers.
- **Multi-cluster** (see Section 3.2.2)
  A single XCS3 portal instance can address several clusters, potentially using distinct job schedulers.
- **Multi-scheduler** (see Section 3.2.3)
  XCS3 is compatible with all major HPC schedulers (Slurm, Altair PBS Pro, IBM LSF, SGE/OGE Sun/Oracle Grid Engine) as well as with OAR, a more confidential one used in a few Universities (e.g., Nice, Luxembourg).
- **Multi Directory Service Compatibility** (see Section 3.2.4)
  XCS3 OAuth-based RESTful API implements a standard and uniform front-end layer to major directory service back-ends. Note that OAuth does not federate identities, that is, if several customers (declared as independent tenants in XCS3) have their own distinct directory services then each login name must be unique (i.e., the same login name cannot be declared in different directory services configured behind the portal). XCS3 supports LDAP, NIS, and AD with LDAP attributes. There are two possible settings:
  - Users and Linux groups (i.e., Projects) information can be imported into XCS3 in Read-Only mode from the directory service(s). This is the only way to do when directory services are configured in Read-Only mode.
  - Users and Linux groups (i.e., Projects) can be created by XCS3 administrator if the directory services are configured in Read-Write mode. Note that only LDAP can be configured this way.

- **Remote visualization** (see Section 3.2.5)
  Most remote visualization technologies can be integrated into XCS3 but at the moment only two are actually used: TurboVNC [69] and XRV [30] (eXtreme factory Remote Visualizer):
  — TurboVNC is a widely used remote desktop software. It was first released in 2004 and is a derivative of Virtual Network Computing (VNC).
  — XRV is an enterprise distribution of the open source Xpra [72] software. Xpra was created in 2008 and is partially funded by the XRV/XCS project since 2012. It uses heuristics to automatically adjust image quality and speed based on measured network latency, network bandwidth, image size, and image movements. Based on the measured data, it selects dynamically the most appropriate picture/video compression codec and optimizes its parameters. XRV bandwidth consumption is lowered thanks to its seamless integration of individual remote windows directly into the client's desktop session (i.e., by default, only the application windows are transferred, not the whole desktop).

  XCS3 supports remote visualization session sharing between users, provided that the remote visualization technology supports it too, which is the case for XRV and TurboVNC. A user can attach a list of coworkers to each of his/her remote visualization session. The coworkers are then authorized to visualize and interactively work on these shared sessions together until the session owner disables the sharing authorization or stops the session.

  If needed, remote visualization sessions can be scheduled for future access via a shared calendar. As sessions are processed as interactive jobs, this induces the use of advanced reservation mechanisms in the job schedulers (for those that support this feature). These are especially useful for planning shared sessions involving users from several locations, for instance.

  The live video stream of remote sessions can be visualized using a software installed on the client host (e.g., an XRV or TurboVNC client software), or using the HTML5 in-browser remote visualization client that does not involve any specific software on the client side. In the latter case, the user can visualize the graphical sessions in a modal window (i.e., a child window of the web application in front of the XCS3 dashboard) or in separate browser tabs.
- **HTTP RESTful API** (see Section 3.2.6)
  All of the XCS3 core features are accessible through its HTTP RESTful API. An overview of its endpoints is given in Section 7.3. The XCS3 HTTP RESTful API is a Software-as-a-Service– (SaaS) oriented API primarily dedicated to HPC, although it could also easily be used for SaaS solutions in other domains. It was patented [94] by its creators.

6.3 Functional Features Not Supported in XCS3

The following nice-to-have features (introduced in Section 3.3) are not implemented in XCS3. Some explanations are given below.

- **Workflow Engine** (see Section 3.3.1)
  In XCS3, job dependency support and application templates with conditional fields (see Section 8.2) are usually powerful enough to cover most of the simple workflow requirements (i.e., no conditional loops, only simple decision and termination tests, no interactivity). This satisfies most use cases of our users. Complex workflows are often included in business/domain specific web portals but are not used in general-purpose HPC portals most of the time (this observation is based on our experience working with HPC users for 20 years). A graphical workflow editor is, however, a nice feature even for simple workflows. So we are considering implementing one in the future.

- **User Data Archiving** (see Section 3.3.2)
  In the first version of XCS (v1.0 in 2010), an archiving feature was implemented together with the file management feature. It was using a third-party long-term keyword-based archiving API. But since no user actually ever used it, we disabled it in XCS2. As a consequence, we did not re-implement it in XCS3. As far as we know, customers have clearly not expressed any urgent need of data archiving or backup related tools in HPC portals so far, probably because these are handled elsewhere in the organization, and because HPC data have a basic lifecycle and become obsolete soon after computing is over. Long term archiving out of the HPC solutions such as object storage, potentially on the cloud, looks more promising.
- **Cloud Management** (see Section 3.3.3)
  Cloud management and HPC services are historically placed under the responsibility of different IT managers and system administrators. This is why most Cloud management tools are independent from HPC middleware and portals. Merging both worlds was not our priority, but we are already working on their emerging convergence, using XCS3 GUI versatility to connect Cloud manager APIs.

7 ARCHITECTURE AND IMPLEMENTATION

This section describes XCS3 architecture, implementation choices, and RESTful API. Then it explains how XCS3 integrates with HPC environments and applications, and it ends with information about XCS3 setup.

7.1 XCS3 Architecture

Figure 3 shows the global architecture of XCS3. The web software consists of two main components. The first one, the Java Core RESTful API web application, controls all core HPC objects (e.g., clusters, users, jobs, applications, etc.). The second one, the GUI web application, controls all web interface aspects (e.g., look&feel, dashboard components, user preferences, etc.). Both components act as separate web servers, which can be accessed through HTTPS/HTTP. Both validate users' authorization through the OAuth service that runs within the XCS3 Java Core (as shown in Figure 5).

In the HPC Platform middleware layer, one or more HPC middlewares (e.g., job scheduler, parallel file system, etc.) rely on Pluggable Authentication Modules (PAM) and/or Name Service Switch (NSS) mechanisms to validate users' authentication and/or authorization.

The HPC cluster middleware and HPC applications layers are independent pieces of software. XCS3 architecture preserves this independence with non-intrusive and independent modules: the HPC cluster integration module and the HPC application abstraction framework module.

This architecture allows independent changes, replacements, and updates in XCS3 code at any level: the RESTful API Java core, the presentation layer (i.e., GUI) libraries, the HPC cluster integration module, the HPC application integration code, and so on. Moreover, this architecture respects the separate security layers scheme as seen in Figure 2.

The GUI web application consists of the following:

- application sErver For angular user InteRface (kEFIR): a web application built with HTML5, AngularJS, and RWD approach. It includes layout templates, look&feel themes, and dashboard components.
- Web Application Server And Broker Interface (wasabi): a web application responsible for the following:

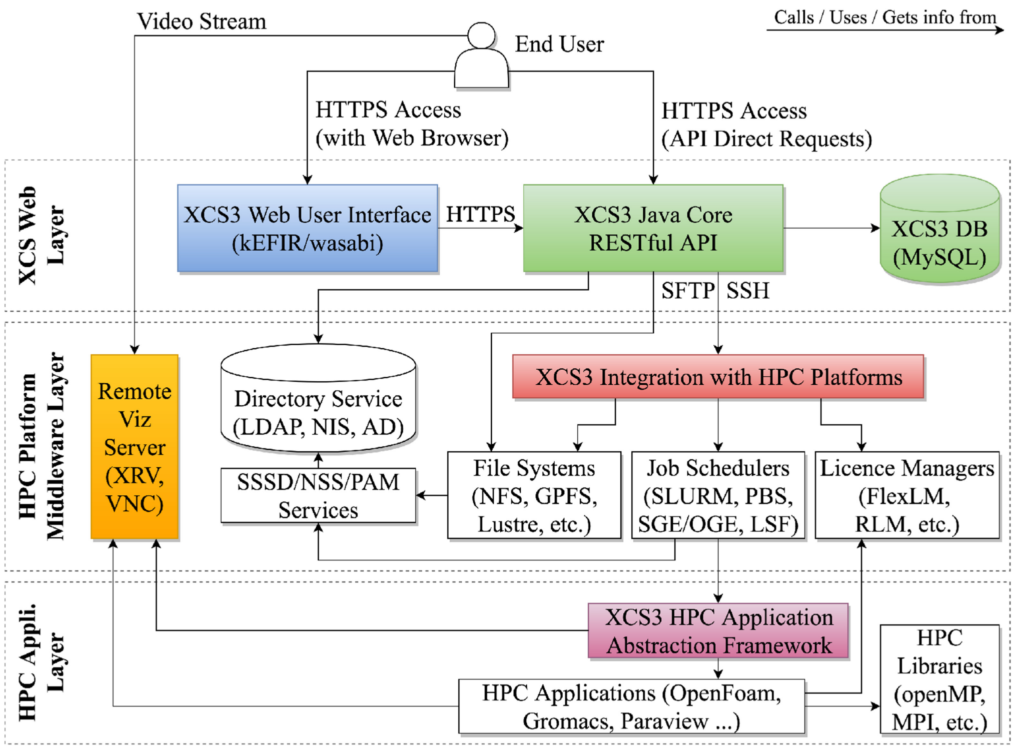

Calls / Uses / Gets info from
Video Stream
End User
HTTPS Access (with Web Browser)
HTTPS Access (API Direct Requests)
XCS Web Layer
XCS3 Web User Interface (kEFIR/wasabi)
HTTPS
XCS3 Java Core RESTful API
XCS3 DB (MySQL)
SFTP
SSH
HPC Platform Middleware Layer
Remote Viz Server (XRV, VNC)
Directory Service (LDAP, NIS, AD)
XCS3 Integration with HPC Platforms
SSSD/NSS/PAM Services
File Systems (NFS, GPFS, Lustre, etc.)
Job Schedulers (SLURM, PBS, SGE/OGE, LSF)
Licence Managers (FlexLM, RLM, etc.)
HPC Appli. Layer
XCS3 HPC Application Abstraction Framework
HPC Libraries (openMP, MPI, etc.)
HPC Applications (OpenFoam, Gromacs, Paraview ...)

Fig. 3. XCS3 high-level architecture.

- serving kEFIR web application,
- proxying RESTful requests between kEFIR and the XCS3 API application,
- providing additional custom endpoints used internally by kEFIR application.

7.2 XCS3 Web Layer Implementation

As shown in Figure 4, the XCS3 web user interface is built on HTML5 [38] and AngularJS [6] (v1.6) and its supported libraries and technologies such as Bootstrap [22] (one of the most popular HTML, CSS, and JS framework for developing RWD projects), sass [60] (CSS extension language), and NVD3 [50] (for charts and graphs). Web frameworks are evolving very quickly, and we are already starting to port our code from AngularJS (written in JavaScript) to Angular [73] (written in TypeScript). This decision is driven by the fact that AngularJS entered a three-year-long Term Support (LTS) period in July 2018 before its end of support, while Angular (v.6) is stable and has become widely used.

As shown in Figure 5, the Java Core RESTful API application uses well-proven libraries such as Hibernate [34] (to handle/serialize Java objects), Jackson [39] (multi-purpose Java library for processing JSON formatted data), Orika [55] (light Java bean object mapper), SSHJ [66] (for SSH2 connections), and Spring Boot [65] (set of components based on Spring).

Spring [64] allows the simple creation of standalone Java applications: It was built as a dependency-injection container (one of the principles of software craftsmanship widely used in Java ecosystem), and it evolved to become a whole framework that handles the infrastructure and that should be considered more as a platform of components than just as a container.

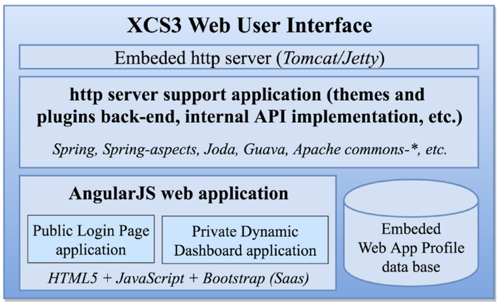

XCS3 Web User Interface
Embeded http server (Tomcat/Jetty)
http server support application (themes and plugins back-end, internal API implementation, etc.)
Spring, Spring-aspects, Joda, Guava, Apache commons-*, etc.
AngularJS web application
Public Login Page application
Private Dynamic Dashboard application
HTML5 + JavaScript + Bootstrap (Saas)
Embeded Web App Profile data base

Fig. 4. XCS3 User interface architecture.

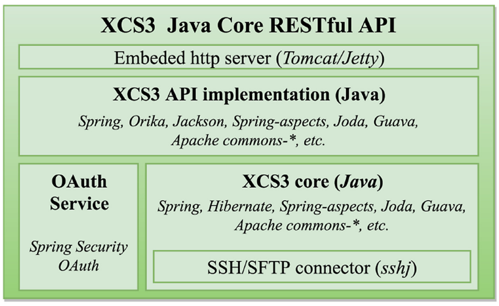

XCS3 Java Core RESTful API
Embeded http server (Tomcat/Jetty)
XCS3 API implementation (Java)
Spring, Orika, Jackson, Spring-aspects, Joda, Guava, Apache commons-*, etc.
OAuth Service
Spring Security OAuth
XCS3 core (Java)
Spring, Hibernate, Spring-aspects, Joda, Guava, Apache commons-*, etc.
SSH/SFTP connector (sshj)

Fig. 5. XCS3 Java core RESTful API architecture.

And just to give a complete view of the development environment we can add that software built is managed under Maven [20] with the Jenkins [41] automation server. All project and code management is done with Atlassian's enterprise collaboration applications: documentation (specifications, technical discussions, and reviews) with Confluence [24], project and issue tracking with JIRA [42], and GIT code management with Bitbucket [21].

The security of a software is tightly linked to its quality. It is thus crucial to ensure that an HPC portal be as robust and bug-free as possible. The Quality Assurance (QA) of XCS3 is controlled by regular manual testing (essential for usability feedback) and by intensive automated tests based on standard software: Serenity [63] for graphical interface tests and REST Assured [59] for the RESTful API functional and conformity tests. Security tests (e.g., intrusion, code injection tentative) are regularly done.

7.3 XCS3 RESTful API Overview

As written in Section 3.2, several RESTful APIs dedicated to HPC resource usage already exist, but no standard has emerged so far, and none of them were flexible enough for our needs. That is why we decided to create our own. The exhaustive list of XCS3 RESTful API endpoints and HTTP methods is available at Reference [31].

The usual primary HTTP methods (also called "verbs") are used to manage objects (also called "entities") exposed by endpoints: `POST` is used to create new objects, `GET` is used to retrieve information, `HEAD` is used to obtain meta-information about an object (e.g., check its existence), `PUT` is used to update existing objects, and `DELETE` is used to delete objects.

Here are some examples of XCS3 RESTful API usage:

- the RESTful endpoint "`/`" together with the `GET` method is used to retrieve information about endpoints availability,
- the RESTful endpoint "`/jobs`" together with the `POST` method is used to submit a job (an example of a complete request command can be found in Section 7.5),
- the RESTful endpoint "`/jobs/{jobId}`" together with the `GET` method is used to get the status of job `jobId`,

7.4 XCS3 HPC Environment Integration

To make XCS3 easy to integrate in any kind of environment (industry customer, public compute center, Cloud HPC service provider, etc.), three main design choices were made and implemented:

- System abstraction layers at several level (as explained in Section 6.3): for standard HPC directory services, for all major HPC batch schedulers, and for all major impersonation mechanisms.

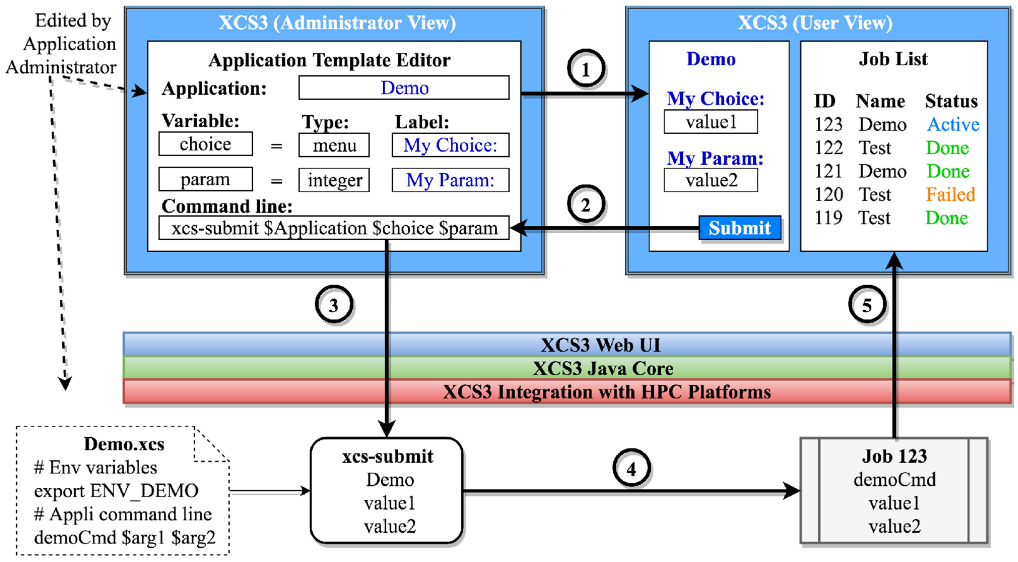


Fig. 6. XCS3 job submission workflow.

- Non-intrusive HPC application integration through two possible approaches:
  - Integration of HPC applications via our default XCS3 job submission templates (described in Section 7.5).
  - Integration of HPC applications using customer's existing scripts for experienced organizations that want to use their own self-maintained job submission methods.
- Non-adherent to any HPC hardware or environment

The HPC cluster integration layer is mainly written with Bash (and some Python) scripts. First, Bash scripts are best suited for working with HPC schedulers; second, they are easy to maintain and adapt for customer specific needs; and, third, they are very common in HPC environments, which makes the integration within customer cluster smoother.

7.5 XCS3 HPC Application Integration

The Application Administrator can create application web forms with the "Application Template Editor." This tool allows the user to place some objects (e.g., fields, menus, check-boxes, radio buttons, file selectors) on a page and to associate them with labels and variables. The template needs then to be associated with a script that includes the HPC application command line and exports its required environment variables.

Once created, an application submission form can be assigned to one (or several) project(s). Users assigned to this project are then allowed to submit jobs through the application form. Figure 6 shows the full lifecycle of an application template for a sample *Demo* application, from its creation by the administrator to its usage by an end-user in a job submission workflow. We suppose that XCS3 Application Administrator has created the *Demo* web form and written the associated `Demo.xcs` script in advance. The job submission workflow showed in Figure 6 starts when the user opens the *Demo* submission form:

(1) When the user opens the Job Submission dashboard, XCS3 renders the *Demo* web form with the information previously edited by the administrator in the Application Template Editor.

(2) User fills the fields of the *Demo* form with `value1` and `value2` and clicks on "Submit."
(3) `xcs-submit` is executed by XCS3 portal user under the end-user identity through an impersonation ("execute as") security mechanism: `xcs-submit` creates a job script with all the information it gathers (scheduler and HPC cluster characteristics from XCS3 settings, application instructions from `Demo.xcs`, argument values from the web form: `value1` and `value2`) and it submits the job script to the scheduler. `xcs-submit` software acts like a meta-scheduler: It takes "standardized" arguments (input, output, stage in/out mode, queue, job name, number of nodes/processes/cores-per-processes, job dependence, job arrays, priority, walltime, etc.) and translates them to submit jobs on any scheduler.
(4) The job gets an ID (e.g., 123), it is queued until enough resources are available on the HPC cluster, and then it is executed.
(5) Job status is monitored by the HPC cluster integration mechanisms. The Java Core web application updates its data base with this information, and the job status is then made available to the web GUI through the data source mechanism. The user can then monitor his/her jobs and get his/her results on XCS3 dashboards.

If a user wants to submit a job without using the web GUI, then the same workflow can be executed by sending requests directly to the HTTP RESTful API web application. Here is a simple example of how to submit a job through the XCS3 HTTP RESTful API with the `curl` command that can be used from a terminal prompt to send HTTP requests[1]:

```
user:~$ curl -X POST -H "Authorization: Bearer 2YotnFZFEjr1zCsicMWpAA"
-d '{application:"Demo",project:"TWEB",params:{choice:"value1",
param:"value2",jobType:"computeHPC",input:"/home/user/data.in",
output:"/home/user/out/"}}' https://www.extremefactory.com/api/v3/jobs
```

The job status list can be obtained (in JSON format) by typing:

```
user:~$ curl -X GET -H "Authorization: Bearer 2YotnFZFEjr1zCsicMWpAA"
https://www.extremefactory.com/api/v3/jobs
```

7.6 XCS3 Setup

This section explains how the XCS3 software is packaged and how its logical architecture maps to physical servers.

XCS3 setup consists in installing two web servers and their associated data bases on a single (or on several) Linux server(s) with JRE 8 (Java Runtime Environment) installed. By default, RPM files are provided for servers with RedHat Linux but packages can be built for any other standard Linux distribution. Once the packages are installed, the configuration step is done by editing default parameters and initial settings in two or three configuration files. It is completed by filling web forms in the XCS3 GUI itself to declare the two main integration bindings:

(1) a directory service: IP address, credentials, Base DN (Distinguished Name), attributes, and so on.
(2) a scheduler client node that will allow communicating with the HPC cluster: IP address, credentials, and so on. The package "XCS3 Integration with HPC Platforms" needs to be installed on this node (e.g., with its RPM file).

[1]The `2YotnFZFEjr1zCsicMWpAA` string is the OAuth 2.0 token obtained from an initial request sent to the API with the user login/password information.

The handling of additional clusters just requires declaring an additional scheduler client node for each of them in the XCS3 GUI.

8 XCS3 NON-FUNCTIONAL FEATURES

This section discusses how XCS3 addresses the non-functional requirements introduced in Section 4: security, usability, performance, and reliability.

8.1 XCS3 Security

Most of the topics listed below relate to the Java Core RESTful API web application, because the GUI web application cannot execute any action nor access any data without calling the HTTP RESTful API service. So most of the security effort is done in the core application. The following security features (introduced in Section 4.1) are implemented in XCS3:

- **Authentication** (see Section 4.1.2)
  XCS3 supports two authentication methods that are configurable:
  - delegation to directory services like NIS and LDAP in the case they are configured to authenticate users: In this case, identification data and user attributes are collected from the directory service tables, and the directory service is used to check user passwords,
  - delegation to an authentication system that can be local or external (e.g., a Kerberos service as described in Section 4.1.2): In this case, only identification data and user attributes are managed at directory service level. Authentication is actually delegated to the authentication system that generates a ticket. This ticket is then handled by XCS3, since it must be used and checked by all the services along the execution chain.

  Other modes, which were supported in early XCS releases, are now deprecated, because they were too basic in terms of security level and ability to be maintained in production (e.g., local host users and dedicated user database).

  Kerberos was supported by XCS2 but the implementation was too restrictive and did not take advantage of standard security layers used by modern operating systems. Kerberos support was not implemented in the first releases of XCS3 (due to lack of resources) but it was added in the last release by relying on GSS-API [96]. For future releases, we are investigating other implementation choices that should rely more on operating-system-level mechanisms such as SSSD [67]. For instance, this will ease integration with Microsoft Active Directory technology.
- **Authorization** (see Section 4.1.3)
  Authorization is managed by the open industry-standard protocol OAuth 2.0 [51] (published as RFC6749 [88] and RFC6750 [92]). OAuth tokens are generated once users are authenticated. Users (or even programs and services) are then authorized to access XCS3 OAuth protected API and GUI by using his/her token.

  The access to functionalities and resources at a very fine-grained level is controlled by roles (as defined in Section 4.1.3). Seven roles are pre-defined: overall administrator, customer administrator, project administrator, application administrator, license manager, standard user, and restricted user. New roles can be created with the Role Editor tool directly available from the administrator dashboard: The scope of about 60 privileges (i.e., actions on an object) can be defined to create new roles. Objects includes applications, user data space, compute power, remote visualization, information about licenses, job status, accounting, and so on. Examples of privileges are the following: CAN_SUBMIT_JOB, CAN_LIST_JOBS, and CAN_UPLOAD_FILES. The possible scopes are (by hierarchical importance):
  - **Global:** the whole portal instance (i.e., all users and resources behind the portal).

- **Customer:** the scope of the Directory Service defined for the selected customer (i.e., all users and resources in the same tenant).
- **Project:** the scope of the project(s)/Linux group(s) the user belongs to (i.e., all users and resources in the same Linux group(s) as the user).
- **User:** only the user scope (i.e., user own resources).
- No scope (i.e., the privilege is not granted).

Each privilege consists of an action and only one scope. Scopes are not disjunctive, so scopes with higher priority extend the range of lower scopes.

- **Accounting** (see Section 4.1.4)
XCS3 gathers a lot of data during job executions. These data can be used for informational, statistical, and/or accounting purposes. Some information is regularly updated and available during job lifecycle (e.g., elapsed time or job status), while some other information is only available after a job is finished (e.g., consumed resources or execution time). The XCS3 accounting mechanism is similar to that of batch schedulers: The accounting data of a job are generated/updated only once (just after the job end). All the information is stored in an XCS3 internal database from which it can be retrieved by API calls. Accounting data can be viewed, sorted, and filtered in XCS3 web GUI. It can also be exported (in CSV format files) for ease of post-processing like usage statistics, inputs for fair-share configurations, or billing.
In addition to this automatic accounting history recording, XCS3 can manage accounting credits at project level (scope). Credits can be added/removed by administrators, and they are dynamically decremented according to a customizable credit formula when jobs are executed. Credit formulas are defined by the administrators at customer and/or project scope. They use standard arithmetic operators and take any XCS3 accounting variables (e.g., number of cores/nodes/GPUs, elapsed time, etc.) as their operands. Once all the credits of a project are spent, users of this project cannot submit jobs anymore, but their active jobs can continue to run until their end. The number of used credits then becomes negative during this phase. The administrator can set a minimum overused credit limit, and when this limit is reached, all the jobs of the project can then be killed.

8.2 XCS3 Usability

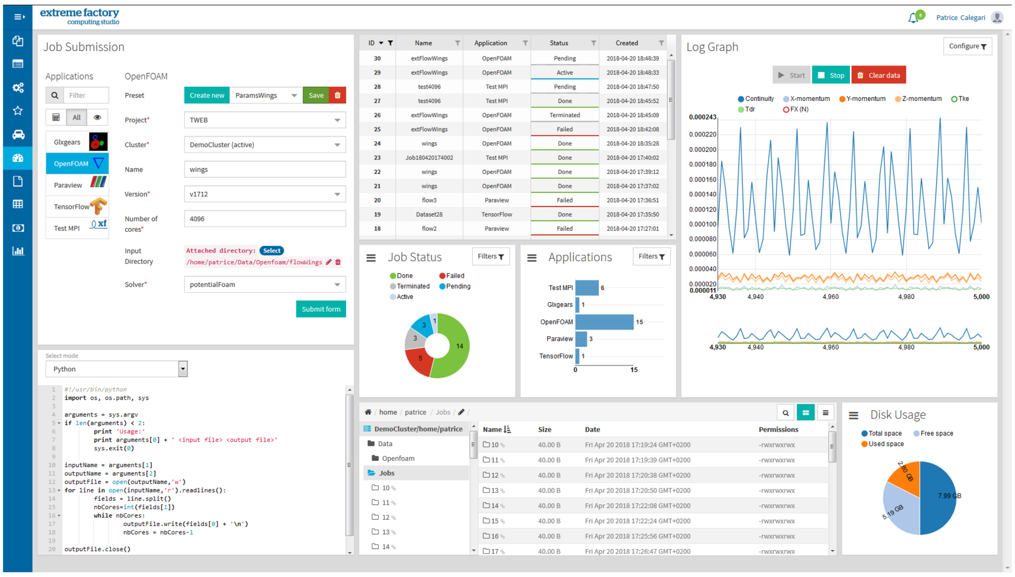
The GUI is the most visible part of the portal. It is built on top of the XCS3 HTTP RESTful API core web services, and it is responsible to handle most of the usability requirements. Here are the main XCS3 web GUI features that were implemented to fulfill usability requirements discussed in Sections 4.2:

- **Operability: double authorization checking** (see Section 4.2)
To fulfill security requirements, such as authorization (introduced in Section 4.1.3), without impacting XCS operability, authorization granting is checked both at the GUI and at the API levels. For example, the views are controlled and filtered based on user roles at the GUI level. Therefore, requests that would anyway be rejected by the RESTful API because of a lack of privilege are never sent by the GUI to the XCS3 API core web application. User authorizations for using any view or dashboard component in the XCS3 GUI are OAuth-controlled 2.0 [51] (as is done for using the XCS3 API core web services).
- **Learnability: tooltips** (see Section 4.2)
When a user selects a web form field, a description of the field and an example of its usage are popped up. This tooltip feature helps the users to quickly learn how to use the GUI for

managing their jobs and files. This feature also helps the administrators to configure the XCS3 web applications.

- **Aesthetics: customizable GUI themes** (see 4.2)
  The colors (i.e., CSS) and images/logos used in the GUI are packaged in theme files that users and administrators can import/export dynamically. Users can select their favorite theme from the list of themes made available by the administrator. Users can also create and share their own custom theme by using an integrated theme editor. Note that only the portal administartor can change the theme of the login page.
- **HPC Application templates** (see Section 4.2.1)
  The XCS3 application template behaves as an abstraction layer that hides operating system, scheduler and shell environment complexity. It presents a unified way to submit HPC/batch computation and remote viewing/interactive jobs (see Section 7.5 for details about HPC application integration). HPC application web form can be created with an interactive "Application Editor": alpha-numerical fields (with type checking: integer, float, text, etc.), checkboxes, file/directory selectors, radio buttons, and pull-down menus can be added with a few clicks. Fields can be conditional, that is, they can be made visible only under some conditions based on the values entered in other fields of the same form. Field values can then be associated to variables used in the job submission script (see Section 7.5 for details about this mechanism).

  In addition to the standard HPC application templates, software license templates are also supported. A software license management abstraction layer operates through the impersonation layer to allow the administrators to control several flavors of license files, license keys, and license servers: FlexLM, RLM, key-based public licensing services, and so on.
- **Fully customizable GUI** (see Section 4.2.2)
  Users can define their workspace look&feel in their XCS3 GUI. They can manage the following elements:
  - **GUI languages:** The GUI is internationalized and can potentially support any language (as long as the translation files are written). At the moment, four languages are already available: English, French, Polish, and German. Users can select and save their default language in their profile.
  - **Dashboards:** The concept is to have a dashboard, that is, a single-page application (SPA), to minimize the number of "clicks" needed to perform all actions of any user workflow. Users can create several dashboards for different usages, switch from one to another in one click, and save them for future use. Users can create/edit/delete their dashboards by placing, configuring, and resizing Dashboard Components (DC). They can share their dashboards with other users by importing/exporting them (in JSON format files). An example of a typical end-user XCS3 GUI dashboard is shown in Figure 7.
  - **Dashboard Components (DC):** Dashboards are built from a collection of building blocks that we call Dashboard Components (DC). Each DC might be considered as a small "App." Typical end-user DCs are as follows: Job Submission, File Management, License Management, Data Table, Data Graph, Scratchpad, Data Plotter, Image viewer, and Text Editor. Administrators granted with the appropriate role and privileges can use DCs dedicated to the administration of Applications, Users, Customers, Projects, Clusters, Directory Services, Credits, Credit Formulas, Roles, and Licenses.
  - **Data tables/Data graphs:** tables and graphs can be defined in DCs to list or show graphical statistics about HPC objects (e.g., clusters, users, jobs, applications, etc.). Users can define the type of object attributes (e.g., job status) they want to show as well as how they



OpenFOAM

Preset	Create new ParamsWings Save
Project*	TWEB
Cluster*	DemoCluster (active)
Name	wings
Version*	v1712
Number of cores*	4096
Input Directory	Attached directory: Select /home/patrice/Data/Openfoam/flowWings
Solver*	potentialFoam

ID	Name	Application	Status	Created
30	extFlowWings	OpenFOAM	Pending	2018-04-20 18:48:39
29	extFlowWings	OpenFOAM	Active	2018-04-20 18:48:33
28	test4096	Test MPI	Pending	2018-04-20 18:47:50
27	test4096	Test MPI	Done	2018-04-20 18:45:52
26	extFlowWings	OpenFOAM	Terminated	2018-04-20 18:45:09
25	extFlowWings	OpenFOAM	Failed	2018-04-20 18:42:08
24	wings	OpenFOAM	Done	2018-04-20 18:35:28
23	Job180420174002	Test MPI	Done	2018-04-20 17:40:02
22	wings	OpenFOAM	Done	2018-04-20 17:39:12
21	wings	OpenFOAM	Done	2018-04-20 17:37:02
20	flow3	Paraview	Failed	2018-04-20 17:36:51
19	Dataset28	TensorFlow	Done	2018-04-20 17:35:50
18	flow2	Paraview	Failed	2018-04-20 17:27:01

Select mode: Python

```
#!/usr/bin/python
import os, os.path, sys

arguments = sys.argv
if len(arguments) < 2:
        print 'Usage:'
        print arguments[0] + ' <input file> <output file>'
        sys.exit(0)

inputName = arguments[1]
outputName = arguments[2]
outputFile = open(outputName,'w')
for line in open(inputName,'r').readlines():
        fields = line.split()
        nbCores=int(fields[1])
        while nbCores:
                outputFile.write(fields[0] + '\n')
                nbCores = nbCores-1

outputFile.close()
```

home / patrice / Jobs /

DemoCluster/home/patrice
Data
Openfoam
Jobs
10
11
12
13
14

Name	Size	Date	Permissions
10	40.00 B	Fri Apr 20 2018 17:19:24 GMT+0200	-rwxrwxrwx
11	40.00 B	Fri Apr 20 2018 17:19:39 GMT+0200	-rwxrwxrwx
12	40.00 B	Fri Apr 20 2018 17:20:38 GMT+0200	-rwxrwxrwx
13	40.00 B	Fri Apr 20 2018 17:20:50 GMT+0200	-rwxrwxrwx
14	40.00 B	Fri Apr 20 2018 17:22:08 GMT+0200	-rwxrwxrwx
15	40.00 B	Fri Apr 20 2018 17:22:24 GMT+0200	-rwxrwxrwx
16	40.00 B	Fri Apr 20 2018 17:25:56 GMT+0200	-rwxrwxrwx
17	40.00 B	Fri Apr 20 2018 17:26:47 GMT+0200	-rwxrwxrwx

Fig. 7. Typical XCS3 user dashboard with eight dashboard components: Job Submission, Data Table (with job status), Data Chart (with job status), Data Chart (with application statistics), Data Plotter, Text Editor, File Manager, and Data Chart (with disk usage).

want to sort and filter them. They are periodically updated at a configurable frequency with values read from data sources. All these preferences are saved in the user profile.

- **RWD** (see Section 4.2.3)
  All XCS3 web elements are RWD enabled on the login page and the dashboards: dashboard components, notification windows, buttons, fonts, graphs, and so on. Their position is optimized and/or their size is adapted based on client display characteristics.

8.3 XCS3 Performance

XCS2 was a single monolithic web application responsible for both back-end and front-end interactions (i.e., to communicate with HPC clusters and to serve web pages to users). With this architecture, the portal response time was highly impacted by scheduler performance and HPC cluster size. Moreover, it made performance analysis difficult to perform.

XCS3 is split into two independent web applications: the "Java Core RESTful API" to deal only with the HPC clusters and the "GUI" to handle user interactions (as described in Section 7.1). Moreover, these two servers are communicating through RESTful HTTP requests (see Section 7.3), whose asynchronous and stateless nature mitigates bottlenecks and/or deadlock situations: If for any reason, one application is slowed down or stalled, then the other one is not impacted. It continues to run without using the latest information of the failing server. Everything gets back to normal as soon as the failing service is up and running again.

Such separation minimizes the performance issues observed with XCS2. This architecture also simplifies the way we can analyze and monitor performance measures of the whole solution.

8.3.1 Java Core RESTful API Performance. The raw performance of the Java Core RESTful API web application can be measured by controlling the number of concurrent requests handled in a specified amount of time. Such performance tests are run daily with JMeter [19] to qualify new implemented functions. To reach our goal (hundreds of API requests per second), the following choices have had the most impact:

- caching of cluster data in XCS3 database,
- selective mechanism for updating cache information in the database: The data update rate in the cache depends on both the entry type and status. For instance, resource usage accounting data and job states are not refreshed at the same rates, and entries that are more likely to change fast are updated more often than others,
- auto-adaptive mechanism for updating cache information in the database: Each datum in the cache is associated to a "validity period" (i.e., an estimate of the best moment to update this data). The value of this validity period depends on some monitored factors like the HPC scheduler response time: The quicker the response time, the smaller the invalidation time,
- specific optimizations aimed at reducing both response time and size of messages for the most often called API endpoints (e.g., "`/jobs/{jobId}/status`") were implemented.

8.3.2 GUI Performance. The GUI application was itself split into two parts: the login part designed and optimized (as a static page) to reduce first access loading time and the Single Page Application (SPA) part optimized for user workflows. An SPA is an application running on the client side (web browser), that is, when web page content needs to be updated, web pages are not reloaded, and only new content is downloaded asynchronously. The use of such a technology makes it possible to update information faster in the GUI, since only the necessary data of the page (i.e., a small part of it) is loaded when needed. The GUI response time is boosted by the stateless nature of the MVW design pattern as well as the JavaScript client-side rendering and content caching, implemented with AngularJS.

Note that this makes performance measurement of the GUI server complex, because simple criteria, like the number of concurrent pages loaded in a given time, which is a good information for API and static web pages, no longer make sense in this context. GUI response time might depend on many factors, such as network performance and good/bad design of application but also, as most of the computation for SPA is done on client side, by the device used to interact with application.

The SPA response time was optimized by using a "request queue" mechanism: If several Dashboard Components (DC) process data from a same API endpoint (e.g., the same data can be shown in a tabular of a Data Table DC and in a pie chart of a Data Graph DC), then only one API request is sent, and the response is propagated to all querying DC. This approach decreases overall GUI response time and also prevents API overloading when using complex dashboards with multiple components.

Other performance optimization choices that have been made are as follows:

- optimized server-side filtering of database entries (only the relevant data to be displayed in tables is actually transferred),
- JavaScript source-code modularization,
- minification and compression of the JavaScript code that is loaded on-demand on the web page (the reduced size of the loaded code decreases its load time and makes its parsing faster),
- use of AngularJS specific optimization techniques (e.g., watchers usage minimization),
- use of performance-proven libraries (e.g., lodash [45]).

8.4 XCS3 Reliability

Here is how the three reliability concepts introduced in Section 4.4 apply to XCS3.

- **Portal Data Backup** (see Section 4.4.1)
  XCS3 has no adherence with any backup technology: Any regular disk- or tape-based backup systems and software can be used. XCS3 contents that require proper backup are mostly:
  - XCS3 databases
  - XCS3 configuration files
  - XCS3 log files
  - Job submission scripts (only in case of customer-specific requests)

  These XCS3 contents should be added to any existing standard backup policies such as daily increments, weekly full backups, and a few weeks retention. Both XCS3 cold and hot database backups are supported. For cold backups, `cron` tabs may be used to stop and restart the XCS3 services out of working hours.

  Another way to easily implement XCS3 backups is to use snapshots and/or clones of the VM that runs XCS3 services when a virtualized approach is used (as presented in the next paragraph).
- **HA/Fault tolerance** (see Section 4.4.2)
  The XCS3 web portal is based on standard underlying technologies and services: mostly Apache front-end HTTP server, Tomcat module for Java/dynamic content, AngularJS framework, and MySQL database server. As such, XCS3 does not onboard its own fault-tolerance mechanism and is in general integrated with third-party fault-tolerance systems, which is a common and easy practice as far as web portal services are concerned. XCS3 has no adherence to such systems.

  The easiest way to implement HA is to virtualize XCS3 (and its underlying services) using state-of-the-art hypervisors for their ability to rapidly and transparently switch an active VM from a failing physical host to another one without service interruption. In the worst case, some users may be logged out from their session but without any loss of their work, since it is constantly serialized/committed to the database. When XCS3 is installed on customer premises, it easily integrates with any virtualization technology that matches customer IT standards. In the case of commercial software (like Citrix XenServer and VMware vSphere) customers can then tap into an existing pool of licenses. Such solutions can also be based on open source software like Proxmox VE, KVM, and VirtualBox.

  Another approach is to use OS-level clustered HA mechanisms such as Red Hat Cluster, Heartbeat, and Pacemaker. These methods implement active-passive models and require extra custom scripting to match the customer's environment. They are rarely used by our customers, probably because they are not as flexible or efficient as VM live migration and active-active HA model.
- **Recoverability** (see Section 4.4.3)
  From an HPC middleware development standpoint, disaster recovery is not a concern, since it is up to the customer's IT managers to design the disaster recovery principles and scope as well as data replication, in coherence with their own business constraints. From our experience, the complete HPC infrastructure is rarely covered by the disaster recovery plan (DRP).

  However, a single XCS3 instance supports the use of multiple clusters, and it makes no assumptions about where they are located. XCS3 can continue to handle jobs as long as not all the HPC premises are affected by a disaster, provided that external network bandwidth and latency are good enough. If a DRP exists and integrates HPC services, then XCS3 will be perfectly fine without inducing any special configuration effort. In this, XCS3 contributes to ease the integration of HPC workloads in a DRP. XCS3 data backup and HA are definitely more sensitive.

9 CONCLUSION AND FUTURE WORK

HPC portals are way more than just a light web presentation layer to address a bunch of job submission scripts. They must involve elaborate system interfaces with the organization's computing, visualization, reliability, and security services. These imply a deep knowledge of HPC specific usages, applications, and system architectures. Moreover, HPC portals must now be "HPC service provider ready" to properly serve and isolate tenants from a single solution and configuration point.

In this article, we listed functional and non-functional requirements that HPC portals should address to answer the current and future challenges of HPC-as-a-Service usage. Of these requirements, we categorized features as "mandatory," "key," and "nice-to-have." The design and development of XCS3 was chosen as a common thread to show how the mandatory and key features can be implemented in one software, and we discussed how the non-functional requirements could be addressed.

The HPC portal domain evolves extremely quickly, so we will need to adapt our XCS3 software regularly. In the near future, we plan to develop new micro services with associated dashboard components. The goal is to support more monitoring features (system performance, power consumption, temperature, HPC application profile, etc.). In the longer term, we plan to investigate the opportunity of integrating a workflow engine, capacity planning functions, and Cloud management capabilities (orchestration, provisioning, cloud bursting, etc.).

On supercomputers as well as on the Cloud, Data Analytics (DA), Deep Learning (DL), Machine Learning (ML), and HPC requirements tend to converge. We anticipate that current HPC portals will soon evolve to address all of these domains, so adding DA, DL, and ML support to XCS3 is one of our priorities.

ACKNOWLEDGMENTS

The authors thank Adam Wieczorek, Michał Bystry, Paweł Chrząszczewski, Zbigniew Rosiak, Grzegorz Kosiacki, Piotr Bubel, Kamil Zaharan, Jakub Muras, and Krzysztof Chołody for their active work on this project, as well as Sébastien Lacour, Olivier David, Dominique Le Corre, Emilien Remy, Aline Ficet, Romuald Sifflet, Laurent Le Grandois, Solofo Ramangalahy, Hugues Bonhomme, Olivier Jean, Marek Walczak, Michał Kaczanowski, and Eric Eppe for their constructive ideas and their support, and those who kindly answered our questions about their HPC portals. The authors also thank Pascale Bernier-Bruna and the anonymous reviewers for their valuable comments and helpful suggestions to improve the quality of the article.

REFERENCES

[1] STORMS: Software Tool for the Optimization of Resources in mobile Systems–FP4-ACTS–AC016–European Commission. Retrieved from http://cordis.europa.eu/project/rcn/30460_en.html.
[2] WebSubmit: A Web-based Interface to High-Performance Computing Resources. Retrieved from https://math.nist.gov/mcsd/savg/websubmit.
[3] DIET: The Grid and Cloud middleware. Retrieved from https://graal.ens-lyon.fr/~diet.
[4] Altair Engineering Introduces New Web Portal Technology for Computational Grids and Distributed Computing. Retrieved from http://www.altair.com/NewsDetail.aspx?news_id=69.
[5] InfoWorld announces our 2008 Best of Open Source Awards. Retrieved from http://www.infoworld.com/article/2637858/open-source-software/infoworld-announces-our-2008-best-of-open-source-awards.html.
[6] AngularJS. Retrieved from https://angularjs.org.
[7] ENES Portal and IS-ENES2 project - vERC. Retrieved from https://verc.enes.org/.

[8] Platform Application Center introduced by Platform Computing. Retrieved from https://www.thefreelibrary.com/Platform+Application+Center+introduced+by+Platform+Computing.-a0212119400.
[9] Fujitsu SynfiniWay V4 enables industrial-strength Enterprise Clouds. Retrieved from https://www.fujitsu.com/uk/news/pr/fs-20100907.html.
[10] Oxalya - HPC by OVH.COM. Retrieved from http://www.oxalya.com.
[11] SysFera-DS Version 5.0 Introduced. Retrieved from https://www.hpcwire.com/off-the-wire/sysfera-ds-version-5-0-introduced/.
[12] W3C High Performance Computing Community Group. Retrieved from https://www.w3.org/community/hpcweb.
[13] Fujitsu Launches HPC Gateway Web Software. Retrieved from https://insidehpc.com/2015/07/fujitsu-launches-hpc-gateway-web-software.
[14] ActiveEon ProActive Parallel Suite. Retrieved from http://proactive.activeeon.com.
[15] Adaptive Computing Viewpoint. Retrieved from http://www.adaptivecomputing.com/products/hpc-products/viewpoint.
[16] Agave Platform Tooling Overview—Agave ToGo. Retrieved from https://agaveapi.co/tooling.
[17] Amazon Web Services AWS High Performance. Retrieved from https://aws.amazon.com/hpc.
[18] Apache Airavata. Retrieved from http://airavata.apache.org.
[19] Apache JMeter Official Site. Retrieved from http://jmeter.apache.org.
[20] Apache Maven Official Site. Retrieved from https://maven.apache.org.
[21] Bitbucket. Retrieved from https://www.atlassian.com/software/bitbucket.
[22] Bootstrap—The most popular HTML, CSS, and JS library in the world. Retrieved from http://getbootstrap.com/.
[23] Compute Manager: Job Submission and Management Portal—PBS Works. Retrieved from http://www.pbsworks.com/PBSProduct.aspx?n=Compute-Manager&c=Overview-and-Capabilities.
[24] Confluence—Team collaboration software. Retrieved from https://www.atlassian.com/software/confluence.
[25] CycleCloud—Cycle Computing. Retrieved from https://cyclecomputing.com/products-solutions/cyclecloud/.
[26] 2017. CyVerse—Science APIs. Retrieved from http://www.cyverse.org/science-apis.
[27] Ember.js. Retrieved from https://emberjs.com.
[28] EnginFrame Cloud Portal—NICE. Retrieved from https://www.nice-software.com/products/enginframe.
[29] Ext JS JavaScript framework for web apps—Sencha. Retrieved from https://www.sencha.com/products/extjs.
[30] Extreme factory. Retrieved from https://atos.net/en/products/high-performance-computing-hpc/bull-extreme-factory.
[31] Extreme factory Computing Studio REST API documentation. Retrieved from https://public.extremefactory.com/docs/xcs/rest-api.
[32] Fortissimo Marketplace—HPC Solutions and services. Retrieved from https://www.fortissimo-project.eu/.
[33] GitHub—OSC/Open-OnDemand: Open-source project based on the Ohio Supercomputer Center's OnDemand platform. Retrieved from https://github.com/OSC/Open-OnDemand.
[34] Hibernate. Everything data. Retrieved from http://hibernate.org.
[35] High Performance Computing (HPC) Solutions Google Cloud. Retrieved from https://cloud.google.com/solutions/hpc.
[36] HPC Services—Sabalcore. Retrieved from http://www.sabalcore.com/services.
[37] HPC Workload-optimized Solutions: FTS—Fujitsu Global, HPC Gateway Application Desktop. Retrieved from http://www.fujitsu.com/global/microsites/hpc/products-services/index.html.
[38] HTML5 W3C. Retrieved from https://www.w3.org/TR/html5.
[39] JacksonHome FasterXML Wiki. Retrieved from http://wiki.fasterxml.com/JacksonHome.
[40] JARVICE is the Cloud Platform for Big Compute—Nimbix. Retrieved from https://www.nimbix.net/jarvice.
[41] Jenkins. Retrieved from https://jenkins.io.
[42] JIRA—Issue & project tracking software. Retrieved from https://www.atlassian.com/software/jira.
[43] Kerberos: The Network Authentication Protocol. Retrieved from http://web.mit.edu/kerberos/www/.
[44] Liferay. Retrieved from https://www.liferay.com.
[45] Lodash. Retrieved from https://lodash.com.
[46] Microsoft Azure Big Compute : HPC & Batch. Retrieved from https://azure.microsoft.com/fr-fr/solutions/big-compute.
[47] Microsoft HPC Pack R2 Web Components 4.5. Retrieved from http://microsoft-hpc-pack-r2-web-components.software.informer.com/4.5.
[48] Neuroscience Gateway Portal. Retrieved from http://www.nsgportal.org/.
[49] NEWT NERSC Web Toolkit. Retrieved from https://newt.nersc.gov.
[50] NVD3 Re-usable charts for d3.js. Retrieved from http://nvd3.org.
[51] OAuth 2.0 - Community Site. Retrieved from https://oauth.net/2.

[52] OnDemand - Ohio Supercomputer Center. Retrieved from https://www.osc.edu/resources/online_portals/ondemand.
[53] Open Source Collaboration Software Platform—eXo. Retrieved from https://www.exoplatform.com.
[54] Orchestrate—RStor. Retrieved from https://rstor.io/products/orchestrate.
[55] Orika reference guide. Retrieved from https://orika-mapper.github.io/orika-docs.
[56] PBS Access & PBS Control, Altair Technology Conference (ATC'2017). Retrieved from http://blog.altair.co.kr/wp-content/uploads/2017/09/Altair_ATC_jhpark_20170915.pdf.
[57] React— JavaScript library for building user interfaces. Retrieved from https://facebook.github.io/react.
[58] Research data management simplified—globus. Retrieved from https://www.globus.org.
[59] REST Assured. Retrieved from http://rest-assured.io.
[60] Sass: Syntactically Awesome Style Sheets. Retrieved from http://sass-lang.com.
[61] ScaleX, Rescale–Platform. Retrieved from http://www.rescale.com/products.
[62] Scyld HPC Cloud Appliance, Penguin Computing. Retrieved from http://www.penguincomputing.com/solutions/scyld-hpc-cloud-appliance.
[63] Serenity BDD—Automated Acceptance Testing with Style. Retrieved from http://www.thucydides.info.
[64] Spring. Retrieved from https://spring.io.
[65] Spring Boot—Projects. Retrieved from https://projects.spring.io/spring-boot.
[66] SSHJ - SSHv2 library for Java. Retrieved from https://github.com/hierynomus/sshj.
[67] SSSD. Retrieved from https://github.com/SSSD/sssd.
[68] TOP500 Supercomputer Sites. Retrieved from https://www.top500.org.
[69] TurboVNC. Retrieved from http://www.turbovnc.org.
[70] UberCloud Marketplace. Retrieved from https://community.theubercloud.com/store/.
[71] UNICORE - Distributed computing and data resources. Retrieved from https://www.unicore.eu.
[72] Xpra home page. Retrieved from https://xpra.org.
[73] Angular. Retrieved from https://angular.io.
[74] Asif Akram, Dharmesh Chohan, David Meredith, and Rob Allan. 2007. CCLRC portal infrastructure to support research facilities: Research articles. *Concurr. Comput.: Pract. Exper.* 19, 6 (Apr. 2007), 751–766. DOI: https://doi.org/10.1002/cpe.v19:6
[75] Asif Akram, Dharmesh Chohan, Xiao Dong Wang, Xiaobo Yang, and Rob Allan. 2005. A service oriented architecture for portals using portlets. In *Proceedings of the UK e-Science All Hands Meeting*, Simon J. Cox and David W. Walker (Eds.). 192–199.
[76] R. Allan, R. Keegan, D. Meredith, M. Winn, and G. Winter. 2004. e-HTPX - HPC, Grid and Web-Portal Technologies in High Throughput Protein Crystallography. In *Proceedings of the UK e-science All Hands Meeting*, Simon J. Cox (Ed.). 187–193.
[77] Christopher Allen and Tim Dierks. 1999. The TLS Protocol V.Version 1.0. *RFC 2246*. DOI: https://doi.org/10.17487/RFC2246
[78] Hrachya Astsatryan, Vladimir Sahakyan, Yuri Shoukouryan, Michel Daydé, Aurélie Hurault, Marc Pantel, and Eddy Caron. 2008. A grid-aware web interface with advanced service trading for linear algebra calculations. In *Proceedings of the 8th International Meeting High Performance Computing for Computational Science (VECPAR'08)*. 106–113.
[79] C. A. Atwood, R. C. Goebbert, J. A. Calahan, T. V. Hromadka III, T. M. Proue, W. Monceaux, and J. Hirata. 2016. Secure web-based access for productive supercomputing. *Comput. Sci. Eng.* 18, 1 (Jan. 2016), 63–72. DOI: https://doi.org/10.1109/MCSE.2015.134
[80] Patrice Calegari, Frédéric Guidec, Pierre Kuonen, Blaise Chamaret, Stéphane Ubéda, Sophie Josselin, Daniel Wagner, and Mario Pizarosso. 1996. Radio network planning with combinatorial optimization algorithms. In *Proceedings of the 1st ACTS Mobile Telecommunications Summit 96*, Chr. Christensen (Ed.), Vol. 2. 707–713.
[81] Ted Carnevale, Amit Majumdar, Subha Sivagnanam, Kenneth Yoshimoto, Vadim Astakhov, Anita Bandrowski, and Maryann Martone. 2014. The neuroscience gateway portal: High performance computing made easy. *BMC Neurosci.* 15, 1 (2014), P101. DOI: https://doi.org/10.1186/1471-2202-15-S1-P101
[82] Denis Caromel, Ludovic Henrio, and Bernard Paul Serpette. 2004. Asynchronous and deterministic objects. In *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'04)*. ACM, New York, NY, 123–134. DOI: https://doi.org/10.1145/964001.964012
[83] Eddy Caron, Frédéric Desprez, and David Loureiro. 2008. All-in-one graphical tool for the management of DIET a GridRPC middleware. In *Grid and Services Evolution*, Norbert Meyer, Domenico Talia, and Ramin Yahyapour (Eds.). CoreGRID Workshop on Grid Middleware (in conjunction with OGF'23), Springer, Berlin, 169–187.
[84] B. Cramariuc and O. Cramariuc. 2010. A brief review of HPC provided as a WEB service by SMEs - Present situation and future trends. *J. Appl. Comput. Sci. Math.* 4 (Mar. 30 2010), 20–24.

[85] Dietmar W. Erwin and David F. Snelling. 2001. UNICORE (Uniform Interface to Computing REsources): A grid computing environment. In *Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing (Euro-Par'01)*. Springer-Verlag, London, Berlin, 825–834.

[86] Geoffrey Fox and David Walker. 2003. e-Science gap analysis. *National e-Science Centre, UK e-Science Technical Report UKeS-2003-01* (2003).

[87] Antonella Galizia, Luca Roverelli, Gabriele Zereik, Emanuele Danovaro, Andrea Clematis, and Daniele D'Agostino. 2017. Using Apache Airavata and EasyGateway for the creation of complex science gateway front-end. *Fut. Gener. Comput. Syst.* (2017). DOI: https://doi.org/10.1016/j.future.2017.11.033

[88] Dick Hardt. 2012. The OAuth 2.0 Authorization Framework. *RFC 6749*. DOI: https://doi.org/10.17487/rfc6749

[89] R. Henry, P. Lagier, and D. Plaindoux. 2004. FSE grid middleware: Collaborative grid environment for distributed computing. *Fujitsu Sci. Techn. J.*, 40, 2 (2004), 269–281.

[90] Luke Howard. 1998. An Approach for Using LDAP as a Network Information Service. *RFC 2307*. DOI: https://doi.org/10.17487/RFC2307

[91] David E. Hudak, Douglas Johnson, Jeremy Nicklas, Eric Franz, Brian McMichael, and Basil Gohar. 2016. Open OnDemand: Transforming computational science through omnidisciplinary software cyberinfrastructure. In *Proceedings of the XSEDE16 Conference on Diversity, Big Data, and Science at Scale (XSEDE'16)*. ACM, New York, NY. DOI: https://doi.org/10.1145/2949550.2949644

[92] Michael Jones and Dick Hardt. 2012. The OAuth 2.0 Authorization Framework: Bearer Token Usage. *RFC 6750*. DOI: https://doi.org/10.17487/rfc6750

[93] Katherine A. Lawrence, Michael Zentner, Nancy Wilkins-Diehr, Julie A. Wernert, Marlon Pierce, Suresh Marru, and Scott Michael. 2015. Science gateways today and tomorrow: Positive perspectives of nearly 5000 members of the research community. *Concurr. Comput.: Pract. Exp.* 27, 16 (05 2015), 4252–4268. DOI: https://doi.org/10.1002/cpe.3526

[94] Marc Levrier, Patrice Calegari, Sébastien Lacour, and Paweł Balczyński. 2016. Programming interface device for generating dedicated computer service programmes for using shared computer resources.

[95] Maozhen Li and Mark Baker. 2006. *A Review of Grid Portal Technology*. Springer, Berlin, 126–156. DOI: https://doi.org/10.1007/1-84628-339-6_6

[96] John Linn. 1993. Generic security service application program interface. *RFC* 1538. https://doi.org/10.17487/RFC1508

[97] Robert R. Lipman and Judith E. Devaney. 1996. WebSubmit - Running supercomputer applications via the web. *Poster*. In *Proceedings of the Annual Conference on SuperComputing (SuperComputing'96)*.

[98] J. W. Long. 2013. Lorenz APIs and REST Services. DOI: https://doi.org/10.2172/1078546

[99] Suresh Marru, Lahiru Gunathilake, Chathura Herath, Patanachai Tangchaisin, Marlon Pierce, Chris Mattmann, Raminder Singh, Thilina Gunarathne, Eran Chinthaka, Ross Gardler, Aleksander Slominski, Ate Douma, Srinath Perera, and Sanjiva Weerawarana. 2011. Apache Airavata: A framework for distributed applications and computational workflows. In *Proceedings of the 2011 ACM Workshop on Gateway Computing Environments (GCE'11)*. ACM, New York, NY, 21–28. DOI: https://doi.org/10.1145/2110486.2110490

[100] Ryan P. McCormack, John E. Koontz, and Judith. Devaney. 1998. *WebSubmit: Web-based Applications with Tcl.* Technical Report 6165. U.S. Dept. of Commerce, Technology Administration, National Institute of Standards and Technology (NIST).

[101] Ryan P. McCormack, John E. Koontz, and Judith Devaney. 1999. Seamless computing with WebSubmit. *Concurr.: Pract. Exper.* 11, 15 (1999), 949–963. DOI: https://doi.org/10.1002/(SICI)1096-9128(19991225)11:15⟨949::AID-CPE462⟩3.0.CO;2-Y

[102] R. Menolascino, P. Cullen, P. Demestichas, S. Josselin, P. Kuonen, Y. Markoulidakis, M. Pizzaroso, and D. Zeghlache. 1998. A realistic UMTS planning exercise. In *Proceeding of the 3rd ACTS Mobile Communication Conference (SUMMIT'98)*, Vol. 1. 157–162.

[103] M. A. Miller, W. Pfeiffer, and T. Schwartz. 2010. Creating the CIPRES science gateway for inference of large phylogenetic trees. In *Proceedings of the Gateway Computing Environments Workshop (GCE'10)*. 1–8.

[104] Marco A. S. Netto, Rodrigo N. Calheiros, Eduardo R. Rodrigues, Renato L. F. Cunha, and Rajkumar Buyya. 2018. HPC cloud for scientific and business applications: Taxonomy, vision, and research challenges. *ACM Comput. Surv.* 51, 1, Article 8 (Jan. 2018), 29 pages. DOI: https://doi.org/10.1145/3150224

[105] Bard Nicolas, Bolze Raphael, Caron Eddy, Desprez Frédéric, Heymann Michaël, Friedrich Anne, Moulinier Luc, Nguyen Ngoc-Hoan, Poch Olivier, and Toursel Thierry. 2010. Décrypthon grid - Grid resources dedicated to neuromuscular disorders. In *Proceedings of the Healthgrid Applications and Core Technologies (HealthGrid'10)*, Vol. 159. 124–133. DOI: https://doi.org/10.3233/978-1-60750-583-9-124

[106] Anwar Osseyran and Merle Giles. 2015. *Industrial Applications of High-Performance Computing: Best Global Practices*. Chapman & Hall/CRC.

[107] Steven T. Peltier, Abel W. Lin, David Lee, Stephen Mock, Stephan Lamont, Tomas Molina, Mona Wong, Lu Dai, Maryann E. Martone, and Mark H. Ellisman. 2003. The telescience portal for advanced tomography applications. *J. Parallel Distrib. Comput.* 63, 5 (2003), 539–550. DOI: https://doi.org/10.1016/S0743-7315(03)00061-3

[108] M. Pierce, S. Marru, L. Gunathilake, T. A. Kanewala, R. Singh, S. Wijeratne, C. Wimalasena, C. Herath, E. Chinthaka, C. Mattmann, A. Slominski, and P. Tangchaisin. 2014. Apache Airavata: Design and directions of a science gateway framework. In *Proceedings of the 2014 6th International Workshop on Science Gateways*. 48–54. DOI:https://doi.org/10.1109/IWSG.2014.15

[109] Tim Polk and Sean Turner. 2011. Prohibiting Secure Sockets Layer (SSL) Version 2.0. *RFC 6176*. DOI:https://doi.org/10.17487/RFC6176

[110] Andrew Prout, William Arcand, David Bestor, Bill Bergeron, Chansup Byun, Vijay Gadepally, Matthew Hubbell, Michael Houle, Michael Jones, Peter Michaleas, Lauren Milechin, Julie Mullen, Antonio Rosa, Siddharth Samsi, Albert Reuther, and Jeremy Kepner. 2017. MIT SuperCloud portal workspace: Enabling HPC web application deployment. In *Proceedings of the 2017 IEEE High Performance Extreme Computing Conference (HPEC'17)*. 1–6. DOI:https://doi.org/10.1109/HPEC.2017.8091097

[111] Dino Quintero, Scott Denham, Rodrigo Garcia da Silva, Alberto Ortiz, Aline Guedes Pinto, Atsumori Sasaki, Roger Tucker, Joanna Wong, and Elsie Ramos. 2012. *IBM Platform Computing Solutions*. IBM Redbooks. 370 pages.

[112] Leonard Richardson, Mike Amundsen, and Sam Ruby. 2013. *RESTful Web APIs*. O'Reilly Media, Inc.

[113] Zebula Sampedro, Thomas Hauser, and Saurabh Sood. 2017. Sandstone HPC: A domain-general gateway for new HPC users. In *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact (PEARC'17)*. ACM, New York, NY. DOI:https://doi.org/10.1145/3093338.3093360

[114] S. Sivagnanam, A. Majumdar, K. Yoshimoto, V. Astakhov, A. Bandrowski, M. E. Martone, and N. T. Carnevale. 2013. Introducing the neuroscience gateway. In *Proceedings of the 5th International Workshop on Science Gateways (IWSG'13)*, Vol. 993. CEUR-WS.org.

[115] Jennifer G. Steiner, B. Clifford Neuman, and Jeffrey I. Schiller. 1988. Kerberos: An authentication service for open network systems. In *Proceedings of the USENIX Winter Conference*. 191–202.

[116] Thomas Sterling, Matthew Anderson, and Maciej Brodowicz. 2017. *High Performance Computing: Modern Systems and Practices* (1st ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA.

[117] K. Suzuki, N. Uchida, H. Kuraishi, and J. Wagner. 2008. HPC solutions for the manufacturing industry. *Fujitsu Sci. Techn. J.* 44, 4 (2008), 458–466.

[118] M. Thomas, S. Mock, and J. Boisseau. 2000. Development of web toolkits for computational science portals: The NPACI HotPage. In *Proceedings the 9th International Symposium on High-Performance Distributed Computing*. 308–309. DOI:https://doi.org/10.1109/HPDC.2000.868671

[119] Xiao Dong Wang and Rob Allan. 2007. HPC Portal: A Repository of portal resources. In *Proceedings of the UK e-Science All Hands Meeting*, Simon J. Cox (Ed.). National e-Science Centre, 629–635.

[120] Xiaobo Yang, Mark Hayes, Karl Jenkins, and Stewart Cant. 2004. *The Cambridge CFD Grid Portal for Large-Scale Distributed CFD Applications*. Springer, Berlin, 478–481. DOI:https://doi.org/10.1007/978-3-540-24685-5_70

Received October 2017; revised September 2018; accepted November 2018