DOI 10.12694/scpe.v17i2.1160
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 17, Number 2, pp. 115–128. http://www.scpe.org

ENABLING SCALABLE DATA PROCESSING AND MANAGEMENT THROUGH STANDARDS-BASED JOB EXECUTION AND THE GLOBAL FEDERATED FILE SYSTEM

SHAHBAZ MEMON[†§], MORRIS RIEDEL[†§], SHIRAZ MEMON[†§], CHRIS KOERITZ[‡], ANDREW GRIMSHAW[‡], AND HELMUT NEUKIRCHEN[§]

Abstract. Emerging challenges for scientific communities are to efficiently process big data obtained by experimentation and computational simulations. Supercomputing architectures are available to support scalable and high performant processing environment, but many of the existing algorithm implementations are still unable to cope with its architectural complexity. One approach is to have innovative technologies that effectively use these resources and also deal with geographically dispersed large datasets. Those technologies should be accessible in a way that data scientists who are running data intensive computations do not have to deal with technical intricacies of the underling execution system. Our work primarily focuses on providing data scientists with transparent access to these resources in order to easily analyze data. Impact of our work is given by describing how we enabled access to multiple high performance computing resources through an open standards-based middleware that takes advantage of a unified data management provided by the the Global Federated File System. Our architectural design and its associated implementation is validated by a usecase that requires massivley parallel DBSCAN outlier detection on a 3D point clouds dataset.

Key words: UNICORE, Genesis II, statistical data mining, data processing, distributed file system, security, standards, parallel processing

AMS subject classifications. 68M14

1. Introduction.

An ever increasing number of datasets from scientific experimentation such as earth observatories or computational simulations generate an enormous amount of information for discovering useful knowledge. In order to analyze data, the area of statistical data mining provides useful methods and tools to extract and explore useful patterns or prediction models. The field of statistical data mining comes with intuitive methods to learn from data, using a wide variety of algorithms for clustering, classification and regression. Several implementations are available, for example, Matlab, R, Octave [3], or scikit-learn. Mostly, these tools offer serial implementation of the algorithms, which is quite challenging (i.e. insufficient memory, extremely long running times, etc.) for processing the volume of data having terabytes or petabytes of magnitude. Considering that amount, the resources running the data processing tools require large number of processors, as well as much more primary and secondary storage. Therefore, parallel tools and platforms such as Hadoop [15] implementing the map reduce paradigm [18] and selected massively parallel algorithm developments based on the MPI and OpenMP environments are commonly used.

We observe mainly tools for (High Performance Computing) HPC and High Throuput Computing (HTC) paradigms evolving concurrently, but each supporting their own set of requirements. Scientific communities, either from biology, physics and medicine adopt more conservative approaches in order to retain their focus on scientific findings and as such traditional HPC environment still play a major role in the relatively new realm of 'big data'. Given the stability of HPC environments and its benefits using locally parallel filesystems with parallel I/O techniques motivates our work to enable straightforward job executions managed by HPC sites that seamlessly access data from a distributed file system service which has not been traditionally supported in HPC-based execution services.

We validate our approach with a use case from earth science using 3D points cloud obtained from devices that measure a large number of points of an object surface (i.e. in our case the inner city of Bremen). The data analysis of this dataset has the goal to cluster special data points and identify any noise elements. In this paper we describe the architectural design and implementation necessary to run data analysis jobs on HPC resources through standards-based UNICORE middleware [10] and uses data from the Global Federated File System [28] that we derive from the architecture of the Extreme Science and Engineering Discovery Environment (XSEDE) [24]. UNICORE is a HPC middleware and deployed on production on XSEDE supercomputing sites, whereas

[†]Juelich Supercomputing Centre, Forschungszentrum Juelich GmbH Juelich, Germany
[‡]Department of Computer Science,University of Virginia Charlottesville, USA
[§]School of Engineering and Natural Sciences, University of Iceland, Reykjavik, Iceland

the GFFS is a distributed network file system which is an integral component of the Genesis II platform, that is also consider to be a middleware element in XSEDE. Our architectural design overcomes the limit that the data is hosted by the GFFS cannot be easily made available to the job executions that perform data clustering over the points cloud data set.

The paper is structured as follows. Section 2 describe the basic background of the technologies and standards used as part of our research. Section 3 lists a detailed requirement analysis we obtained during the course of the integration effort. Section 4 describes the security model and the implementation we derived for supporting the requirements from Section 3. Section 5 offers detailed insights on our architectural design and its realization that enable the UNICORE and GFFS integration while addressing the selected requirements. Section 6 takes a massively parallel data analysis application in order to validate our work based on a real world use case. Section 7 provides a brief overview of the related work, and the paper concludes in Section 8.

This article is a joint and extended version of [34] and [40].

2. Background. This section gives a brief background of the technologies, algorithms and standards that supported our work.

2.1. UNICORE. UNICORE is an HPC middleware which is built upon the priniciples of Service Oriented Architecture (SOA). It realizes compute, information and data functions through a set of stateful web services [42]. These services are designed in such a way that they enable seamless access to heterogeneous high performance computing resources. In this sense, the middleware layer to these clusters provides access and location transparency to compute and and thus offers scientists a unique environment hiding low level technical complexities (i.e. avoiding writing and submitting error-prone scheduler dependent job scripts). The compute access transparency enables an abstraction of different flavors of resource management systems (sometimes also referred to as schedulers), such as SLURM [45] or Torque [6], and more notably through a unified and standard interface. Figure 2.1 depicts the basic UNICORE architecture that is composed of layers with distinct functionality, including Client, Services and Target System Interface.

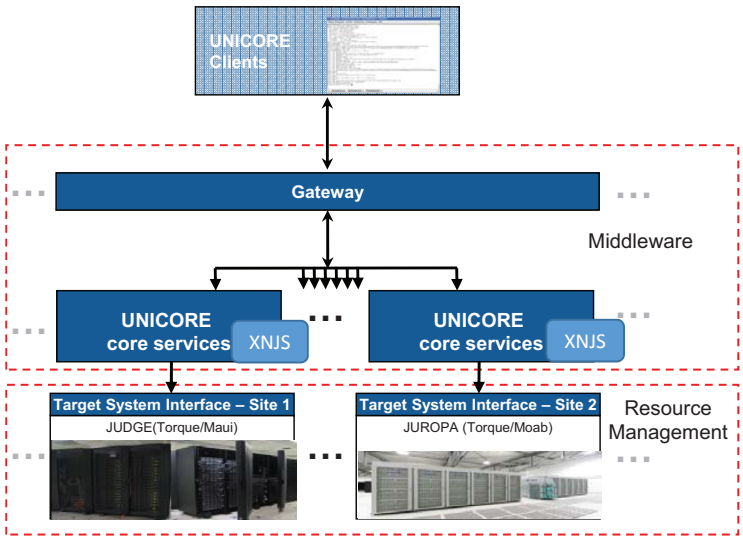


FIG. 2.1. *Basic UNICORE architecture with example deployments of two HPC systems in Juelich (JUDGE and JUROPA).*

The client layer provides API and end user interface, which include interfaces for constructing and sending client requests to remotely deployed services. The client side API is useful for scientific communities which are not necessarily using the UNICORE's povided interface, but instead their own clients such as their application specific science portals or gateways (e.g. UltraScan Scientific Gateway [33]. Hence, all important functionality of the middleware services can be invoked through the direct client API interaction. The end user interface

offers a rich client interface called UNICORE Rich Client (URC) [19], with advanced user controls to compose and orchestrate scientific workflows. The second variant is a command line client called UNICORE Command Line (UCC), which provides an interface for advanced users who know the low level details of writing job request scripts to be executed jointly on the batch system. For a more detailed architecture explanation we refer to [10].

The Services layer plays a vital role in enabling job execution and data management by means of SOAP [16] over XML based web services. Not only the Services layer implement the core functionalities, but also the hosting environment which can host and deploy stateless and stateful web services, for instance, WS-I (Web Services Interoperability) [5] and WS-RF (web Services Resources Framework) [42]. Job management functionality implements a complete life cycle through which job passes, and that includes submission, monitoring and data staging. The job management functionality is supported by UNICORE's embedded scheduling and execution framework, called XNJS (Extended Network Job Supervisor) [43]. It manages the incoming middleware requests against the hosted application and resource capabilities (for example number of available nodes, processors per node etc.). The Services layer gives a configuration based interface to expose underlying cluster resource and environment, so that XNJS can perform resource match making upon the client initiated job requests. After validating the job request, the XNJS component formats the job to the generic UNICORE protocol, and then sends it to the resource manager specific implementation of Target System Interface. This is the layer where the generic UNICORE script gets translated to the request formatted according to the batch system.

As a summary, a simple job execution sequence comprises of, client job submission to the Services layer, then the request is forwarded to XNJS, and then it is communicated to the Target System tier. This tier in turn directly interacts with the low level batch system, and fetch job statuses, and manage underlying running file transfers during the job's execution life cycle.

2.2. The Global Federated File System. The Genesis II Global Federated File System (GFFS) [28] is a distributed file system that provides researchers with tools for securely managing and sharing their scientific data. The GFFS offers a set of interfaces that manage jobs and provide access to the required scientific data. This is achieved through the GFFS-Queue component, which is also based on SOA wherein standards-based interfaces are adopted for storing and accessing remote compute and data resources. In order to support federation across different organizational entities, the GFFS provides a hierarchical file system structure with standard namespace locations for storing user profiles, groups, directories, and service elements such as meta-scheduling queues and Basic Execution Service endpoints (BES) [29] for processing jobs. Figure 2.2 provides an overview of the integrated architecture with Genesis II and UNICORE Basic Execution Service (BES) endpoints interacting with the GFFSs root container.

The GFFS implements many of the standard Unix commands (such as `cp` and `mv`) in a console mode through the so called grid shell. There is also a GUI view of the GFFS, which supports rich drag and drop file management. The GFFS also provides a FUSE file system interface [2] that allow users to mount the GFFS on a Unix directory and operate on files in the GFFS as if a user is interacting with her local file system. The GFFS has an export feature like NFSv4 that allows users to share part of their own file system visible within the GFFS, and to other users part of the broader federated infrastructure. The GFFS Queue is a metascheduler that supports submission of multiple jobs for subsequent distribution to the execution service endpoints connected to the queue. The GFFS Queue provides researchers with a mechanism for managing and controlling their computations via a GUI as well as with familiar command line tools such as "qstat" and "qkill". Jobs will be distributed to BES resources automatically by the GFFS Queue, but can also be rescheduled as needed. The XSEDE project benefits from the GFFS by giving researchers a way to securely share data with their colleagues and by providing a high level view of the computational resources available at the XSEDE infrastructure through the GFFS Queue.

2.3. Standards. RNS (Resource Namespace Service) [36] is an Open Grid Forum initiative that standardizes the naming of distributed resources that form an infrastructure. It has a simple set of operations for managing grid and cluster services mapped as file system operations such as `rm`, `mkdir` or `cp`. The RNS specification provides client applications to couple WS-Addressing [22] based endpoints with human readable notations. For instance, a job execution service managing multiple jobs can be represented as a parent directory and individuals jobs are child directories which may further contain the contents of their working directories.

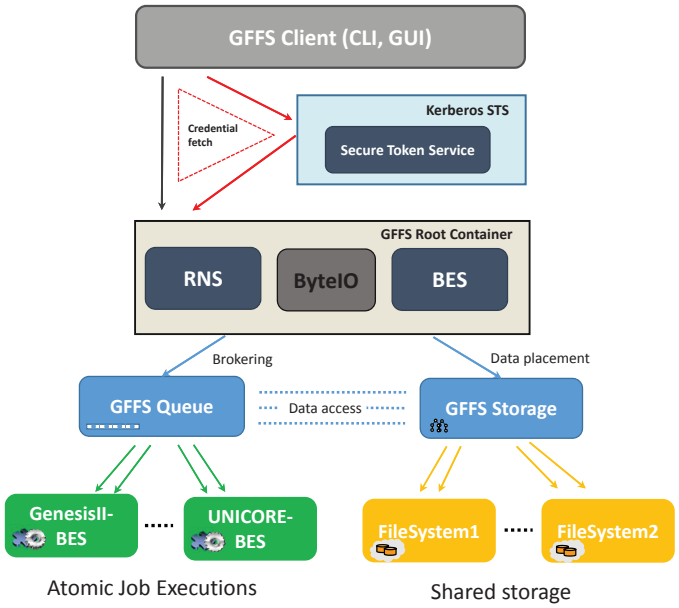


FIG. 2.2. *The Global Federated File System (GFFS) architecture.*

Considering the stateful service based endpoints, wherein web service resources can have a nested heirarchical structure, the RNS representation is very helpful in providing an access and location transparency to underlying resources. The GFFS mentioned earlier implements the core RNS and its Web Services Resource Framework (WSRF) [37] rendering to access service endpoints. The statefulness gives individual access which is very much analogous to the domain of distributed remote objects. As far as the data transfer is concerned the GFFS primarily uses the ByteIO [35] standard. ByteIO provides a set of interfaces to interact with bulk data sources and sinks. The ByteIO standard enables large amount of data in an efficient way. It has two interfaces, RandomByteIO and StreamableByteIO. RandomByteIO provides an interface to access bulk data in a stateless and random manner. This interface is normally being called when a client transfer large files. StreamableByteIO interface allows data transfer data in a stateful manner. It is normally used for accessing short files, in most cases standard outputs of managed jobs.

The Job Submission and Description Language (JSDL) [8] is an XML-based comprehensive data model for specifying computational job requirements consisting of application, resources and data concepts. UNICORE and Genesis II clients specify job requirements in the JSDL format. UNICORE server side implements most of the JSDL, and also its related profiles and extensions.

The JSDL specification has a generic model for representing multiple type of resource settings, such as HPC and HTC. The JSDL model further provides a set of profiles which imposes constraints on requirements according to the type of resource architecture. These requirements may include, file staging modalities, parallel execution environments and parametric jobs. This paper mainly targets HPC resource types which normally use parallel execution environments and a resource management system. HPC resource profile [12] enable users to specify more internal HPC architecture specific requirements in combination with restrictions on the execution service. The HPC file staging profile [44] captures data movement specific elements to be used within heterogeneous cluster environments. This profile impose constraints on using HPC specific data staging attributes as part of the job submission request. For instance, the request may contain FTP user name and password credentials for the BES instance to carry out third-party file transfers on user's behalf.

The related technologies presented above provide a base for providing a seamless and robust middleware platform to tackle big data challenges. One of the common big data processing machinery requirement is to have an iterative execution for discovering optimal set of algorithm parameters. Specifically, data clustering algorithms such as K-Means or DBSCAN require certain parameters before their processing. In the case of

iterative execution multiple runs with a varying set of parameters are required. If we combine these runs into a single composite job then this kind of job is called parametric. UNICORE as our base execution middleware supports parametric jobs through the JSDL Parameter Sweep extension [23]. This specification provides an intuitive model to parametrize multiple parts of job request per se. It may allow client application to sweep over a list of arguments, file transfer locations and environment variables specified under the JSDL request. The sweep model can represent different kind of iterations, either be it a element-wise iteration on a set or a counter with configurable stepping factor. There are two major kind of sweeps the specification provides, Document sweep and File sweep. The Document sweep provides a model to modify a requested JSDL instance. The File sweep is an advanced model which presents a data structure to modify the contents of the files imported before the job execution phase. This kind of sweep is applicable to text based files. UNICORE middleware implements both kind of sweeps through its client API and command line client (UCC). This specification is used to automate the iterative data clustering on the points cloud data set we use in this paper.

OGSA-Basic Execution Service (BES) [8] is an Open Grid Forum (OGF) initiative which provides a web services-based interface for managing and monitoring computational jobs in HPC and HTC environments. For a job submission use case BES interface accepts a JSDL instance and its related profiles as a parameter and then runs it on back-end resource. The focus of this paper is based on a scenario in which UNICORE server expose it computing capabilities via BES model, and Genesis II client use this interface to invoke remote calls on the UNICORE endpoint.

2.4. Unsupervised Learning. While the overall architectural design in this paper is applicable to many learning models, our work is using parallel version of the Density Based Spatial Clustering for Application with Noise (DBSCAN) algorithm [21]. The focus is rather on the access of the algorithm implementation through the UNICORE middleware and the GFFS based file system. Therefore, this section briefly introduces the DBSCAN method. Goetz et al. describes more details on the algorithm implementation in [27], which gives more detail on what parallelization strategies are used to achieve scalability and high performance while analyzing large data sets. DBSCAN [21] is an unsupervised density based clustering algorithm. The cluster based on density is represented by a number of points MinPoints within a specified radius Epsilon. These are the important user defined parameters of the algorithm to identify the clusters.

i) Core point: A central point in a dense region, it has more than a specified number of minimum points MinPoints within its neighborhood (or radius) Epsilon.

ii) Border point: A point that lies on the border of the dense region, it fewer than minimum points MinPoints within its neighborhood (or radius) Eps

iii) Noise point: A point that is neither a core point nor a border point

DBSCAN intrinsically enables maximizing the local point density recursively. It sets apart from other clustering algorithms as it detects the clusters of arbitrary shapes and sizes. Notably, it is resistant to noise and suitable for finding anomalies or filter specific noise signals from the data. We use the parameter-based DBSCAN learning algorithm as a specific example of how our architectural design and its implementation can be generically used by a wide variety of learning algorithms in this paper.

3. Requirement Analysis. During the course of transparent integration for bridging both technologies, we identified multiple requirements which not only aims at superficially combining them, but also some extensions in the UNICORE services layer which will help to tackle "big data" challenges from machine learning and data mining. The integration has taken XSEDE infrastructure as an example, but the implementation is applicable to any distributed computing and storage infrastructure. The major integration requirements from both the technologies' perspective are summarized together with relevant technology environments in Table 3.1. They lie in the following areas. R1) Secure Trust Delegation: As in a distributed service interaction a user interacts with a portal or meta scheduler which then forwards the request to a service that takes care of the job execution and also calls upon data management services to pull and fetch data. While the user is participating in the very beginning, then the following phases are to be done by other services, require some kind of trust delegation which the user entity assigns to the target job execution and data management services to act on her behalf. In our scenario a user communicates a data oriented job request with a set of input data staging elements, therefore trust delegation has to be implemented by the UNICORE platform to understand the GFFS user requests. R2) Openness: In any kind of communication between a user and the GFFS or UNICORE, it

TABLE 3.1
Summary of Requirements

#	**Requirements**	
	Description	*Environment*
R1	Secure Trust Delegation	GFFS and UNICORE
R2	Openness	OGSA-BES, JSDL
R3	Data transport	RNS, BYTEIO
R4	Infrastructure integration	XSEDE, MYPROXY
R5	Transparency	UNICORE Server
R6	Parameter sweep	JSDL Parameter Sweep
R7	Extensible resource and job model	GFFS and UNICORE

should support standards-based protocols, so that users or services from different middleware backgrounds can easily interact through UNICORE and the GFFS client-side APIs. R3) Data transport: As UNICORE jobs are intended to use data from the GFFS, the running jobs should be able to upload and download data from the file system space. R4) Infrastructure integration: XSEDE-based identity management should be understandable to both layers of job submission and data management middlewares. R5) Transparency: The jobs managed by UNICORE in an HPC environment which accesses data from the GFFS data should not know the physical location and also on how the data is structured across data nodes within the file system space. R6) Parameter sweep: This requirement is very specific to jobs which require re-running the same application but with different parameters:these are called composite or parametric jobs. In a parametric job, the execution middleware iterates through a set of parameters provided by a user job submission request and creates a separate job internally for each parameter combination. In this case UNICORE middleware should be capable of interpreting and incarnating parametric jobs. R7) Extensible resource and job model: As supercomputing architectures are evolving to support data and network intensive applications, the hosting middleware should be adaptive to new changes and thus possess an extensible model for users specifying sophisticated requirements. For instance number of GPGPUS or use of execution environment.

4. Interoperable Security Model. The GFFS security model is based on the Security Assertion Markup Language (SAML) [41] standard, and takes the UNICORE SAML profile [14] as a reference implementation. The GFFS extends the UNICORE's SAML profile to represent the trust delegation chains of the Genesis II security model. A delegation chain encompasses that a user has delegated some level of trust to a service in the GFFS in order to achieve a task, such as processing a job. Mostly, delegation operations include three entities, (1) a grid user (for our example, called U, (2) a TLS connection identified by an X.509 certificate (called C), and (3) a grid resource (called R). Longer delegation chains usually contain all three of these types of entities as the first links in the chain.

In the GFFS, the first entity U is always a user defined as a grid STS (Secure Token Service) object. This entity is the prime mover for any operations that are performed in the GFFS. The user's rights within the grid's access control list permission system dictates what that user can and cannot do with regard to every grid resource.

The client software must authenticate as the grid user U to obtain services from a GFFS container. This is where the second security entity C comes in; it is the connection by the client software at the behest of the user. Initially, the client credentials only contain C, as one makes the connection before STS authentication occurs. In the XSEDE login process, this connection is always based on X.509 credentials obtained from a certificate authority service, the so called XSEDE MyProxy server. Thus, it can be a well-known identity within both Genesis (via a Kerberos-based STS) and UNICORE (via the grid-mapfile). In the example, the client software first authenticates to MyProxy by using user name and password to obtain the certificate C. Then, the client authenticates to the Kerberos STS in the GFFS to obtain grid user U.

After the user authenticates, the client software's credential wallet will contain the first delegation of trust, U→C. This states that the grid user U trusts the TLS connection C to act on its behalf. Afterward, all of the actions taken by the TLS connection C are understood to be U's actions. Any access that provides U with

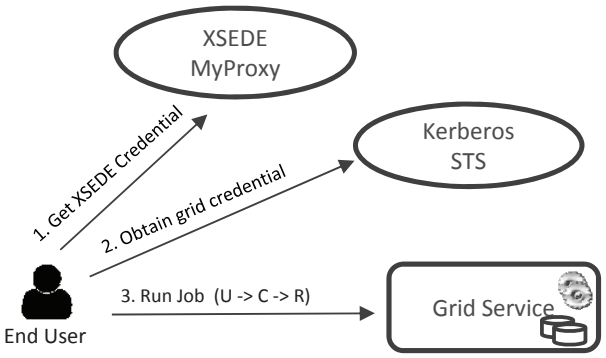


FIG. 4.1. *End User interaction with security and grid services.*

permissions will also be granted to C. That may include submitting a job to a BES named R. This is the second point where trust is delegated; the certificate for TLS connection C signs a new trust delegation that expresses "C trusts R" to perform a job execution. This extends the length of the delegation chain by one, so that it now has all three entities involved in two trust delegation objects. This can be represented with delegation arrows such as:

- First delegation: U→C (The grid user trusts the TLS connection)
- Second delegation: C→R (The TLS connection trusts the resource)
- Full Chain: U→C→R (The user U trusts the connection C which trusts R to run a job)

This new delegation chain can be presented by resource R when it needs to do further actions on the grid user's behalf. Moreover, actions might include storing file staging results back to RNS space (U→C→R→D, where D is a Data folder or possibly submitting the job to another BES for final processing. The important point is that entity D is just another resource to which trust can be delegated, and the chain can be continued in that manner for as long as its delegation depth limit allows. Figure 4.1 depicts a typical interaction of an end user with the security services and a target grid (execution or data management) service.

One challenge while interoperating between the GFFS and UNICORE integration arose due to a difference in interpretation of the SAML assertions. The delegation chains in the GFFS are tightly-coupled, and do not allow mixing and matching of individual entities. This is not directly provided by the UNICORE SAML implementation, which permits the receiver to mix and match any delegations provided in a message (U→C→R is considered to be two separable delegations U→C and C→R). In the GFFS model, a delegation chain must be used in its entirety or not at all.

To address this difference, the GFFS implementation of SAML adds a unique identifier to each SAML assertion. A chain such as U→C→R is built by embedding the identifier of the U→C assertion in the C→R assertion. To make this cryptographically secure, the signature of the U→C assertion is also embedded in the C→R assertion before C→R itself is signed. This enforces the connections between the GFFS delegation chains while still leveraging the UNICORE's Security Assertion Markup Language (SAML) implementation. Upon reception, the chains are reassembled and any assertions that are referenced by a longer delegation chain are removed from the pool of available assertions.

The Genesis delegation chain model supports having multiple chains in a credential wallet. This supports the user possessing multiple different types of identity and authorization on resources. The users will always have their own identity as a credential, which allows them access to resources where the user has been given explicit permission. The user will also usually have at least one group credential, which allows them access to portions of the grid file system. Additional group credentials may convey access to different BES or queue resources within the grid. Thus the credential wallet approach supports a flexible authorization appropriate to the variety of grid resources, possibly across multiple administrative domains, that may be required for the user's work.

The signing of credentials ensures that it is computationally infeasible to create a fraudulent credential chain where a new identity is inserted into the credential chain. Each credential records the signature of the

prior element in the chain, along with its unique identifier. Thus an attacker would have to compute a valid XML digital signature inside a valid trust delegation object, where the unique id is also properly signed by that signature.

To ensure that the credential wallet cannot be easily compromised and used for playback attacks (where the valid credentials of a user are stolen and used by a different user), all credentials must be "anchored" with the current TLS session credential of the grid client. At least one link in the credential chain must be identical to the TLS session certificate. This ensures that playback is very difficult indeed, since the stolen credentials must be based on the TLS session key that the user was employing at the time the valid credentials were minted. This mitigates attacks upon the server, where the container database is compromised. Attacks using an entire set of stolen client credentials are also somewhat mitigated, since the TLS session certificate is based on a short-lived key pair.

With respect to the requirements mentioned in Table 3.1, the given security model implementation covers R1-Secure Trust Delegation. XSEDE's MyProxy access is provided to allow XSEDE users run job with their infrastructure credentials. This feature relates to R4-XSEDE MyProxy integration.

5. Integrated Architecture and Implementation. We have extended UNICORE's server tier to accept the incoming requests incoming from Genesis II remote clients. The remote clients here implies the GFFS's GFFS-Queue component which is an entry point for a user to submit job. The GFFS-Queue acts as a meta scheduler that schedules the user's request based on its resource requirements on a set of available BES-based computing endpoints. Even though UNICORE understands BES protocol, but still the execution service should know how to interpret, authenticate, and authorize the incoming GFFS Queue requests. A separate UNICORE server extension is implemented that is invoked when server finds a security token containing GFFS–related information in the incoming client request. The extension validates the SAML chain by looking into every element of the chain. These elements are entities (described in the previous section) which contain every stakeholder including end user or service through which the request was passed. The standards-based access and the validation of incoming requests required to trigger the data transfers serve the requirements R2-Openness and R3-Data transport. The user doesn't need to provide the actual physical location of the GFFS hosted data, instead she uses the symbolic RNS qualified hierarchical paths. This feature is inclined to support R5-Transparency.

Before a Genesis II client is able to send jobs to a UNICORE BES endpoint, a Genesis II container should recognize and link the UNICORE BES instance into the RNS space. The linking is achieved through the Endpoint Reference Minting process. An EPR (Endpoint Reference) is the basic component of the RNS. Every location in the GFFS namespace has an EPR that identifies (1) where the resource lives and (2) the X.509 certificate that represents the resource. Minting an EPR is the process of creating a new EPR as an XML document that represents an external resource, such as a UNICORE BES instance. The process of minting an EPR combines the URI where the resource is located with the X.509 certificate expected as the resource's identity (which it would report over a TLS connection). Once an EPR is minted, the EPR's XML document can be stored locally as a file or added as a new link to the grid namespace. When linked into the grid, a user with appropriate credentials can see the entry in the GFFS files system and can use it to obtain whatever services the resource provides.

The user's XSEDE identity is extracted from the delegation chain, and the retrieved identity is validated against the authorization store of the UNICORE server deployment. If the user is found under the authorization store then the required user context is created for carrying out GFFS data staging invocations. After the context creation phase, the server extension releases its control and job moves to the next phase of execution. In the beginning of the execution phase the request is processed further to carry out the GFFS-based data stagings of jobs. Figure 5.1 shows the job request encoded in JSDL containing application requirements and data staging elements pointing to the GFFS space. Note that the job will be executed on the UNICORE site, therefore Genesis II-BESes are not involved in this sequence.

The GFFS-based data transfer is realized through an XNJS extension. The GFFS download component prepares a command to pull data from the GFFS. The command (such as '`grid cp remote-source job-working-directory`') will be forwarded to the Target System Interface (TSI) that invokes the Genesis II '`grid`' command gracefully and downloads the data to the job's working directory. In a likewise manner the

GFFS upload takes care of uploading output files to the remote GFFS location. Any failure will make UNICORE fail the job and abandon any further processing related to it. For every job which is sent by the Genesis II client UNICORE server extension maintains an additional folder in each of the job's working directory that contain user's contextual information.

The pictorial representation of a simple job submission sequence is shown in Figure 5.1. In the first step, the Genesis II GUI client asks a user to log-in through an XSEDE provided credentials. After the authentication phase (step 2), she uploads data to the GFFS folder (step 3). In step 4 the user then submits a job request with application details, and location of the data to be downloaded. In a similar manner, output data paths are also specified. The user then selects a UNICORE endpoint and run the job. Steps 5 and 7 shows a submission of request to the TSI and the target batch system. As soon as the job has been submitted to the execution service, the client continuously monitors the job until it reaches to a terminal state. Once the job is finished successfully the output is fetched back from the remote job working directory which is located on cluster's file system to the GFFS space. Steps 6 and 8 depicts the TSI and the GFFS interaction. For the sake of brevity only a sequence of major steps are being highlighted.

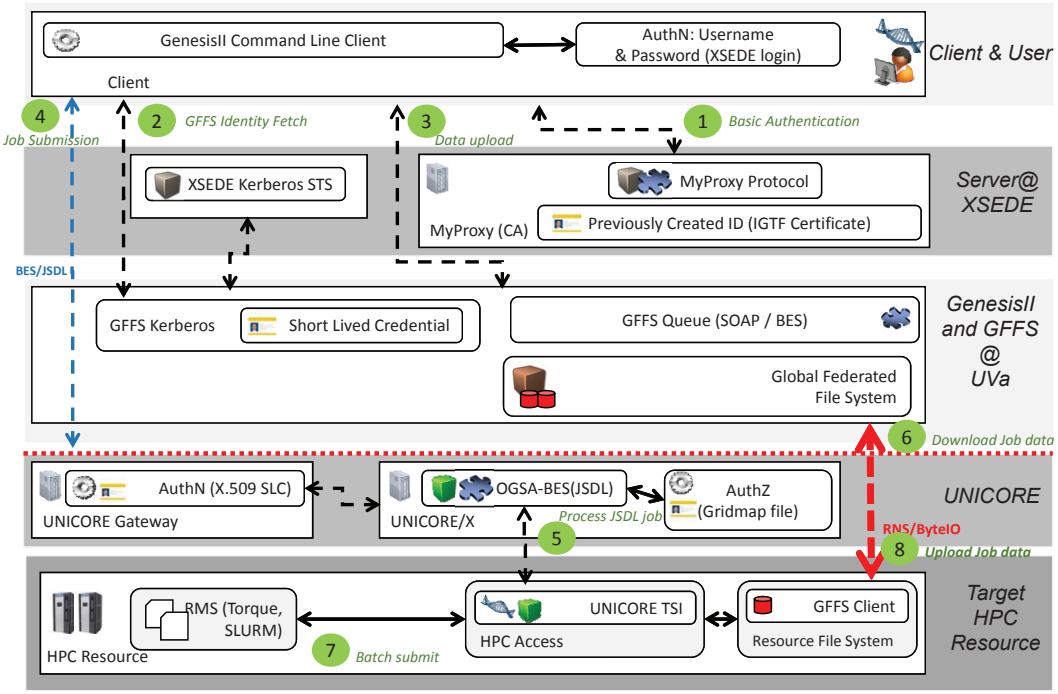


FIG. 5.1. *The UNICORE and the GFFS integration showing a job submission sequence with data staging.*

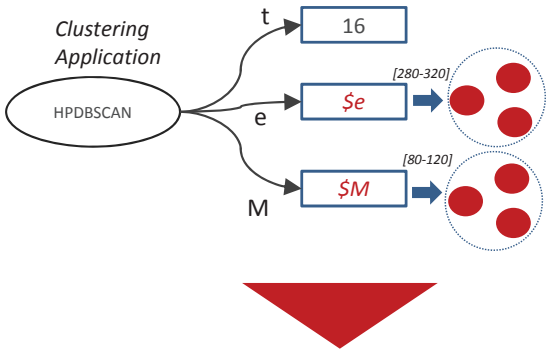
Another building block for supporting semi-automated data analytics is to allow user running jobs of parametric nature. Specially, the use case presented in the next section needs multiple runs required to identify optimal set of application parameters. By using the JSDL Parameter Sweep extension implementation of UNICORE [32] users can run multiple jobs as a single request. This is a very useful feature that has a positive impact on the overall data analysis life cycle. Considering, if manually running many jobs and compiling resultant outputs by hand, it will take user's considerable time just for book keeping the previous and next set of runs. We experimented by sending parametric jobs via Genesis II client and UNICORE BES implementation, and compared this model with manual SSH based submissions. By following this approach the user's administrative and usability overhead has significantly reduced. Figure 6.1 shows the snapshot of the used JSDL Parameter Sweep instance for the application.

6. Usecase: Point Cloud Anomaly Detection. Anomaly or outlier detection algorithms primarily identify a set of data points that appear to be different than the remaining data. There are different data

clustering approaches which help data scientists to discover anomalies and a meaningful set of clusters from data. Several methods exist, for instance K-Means and Agglomerative clustering, have been used in commercial and scientific domains.

In this paper we place our focus on the DBSCAN [21]] algorithm which allows to reduce noise factor of the 3D point cloud dataset. A point cloud dataset captures objects in three dimensional space representing the external surface of objects by a point cloud. In our case, we use a data set that contains a point cloud for landscape elements, such as different kind of buildings, monuments or bridges, of the city of Bremen, Germany. This points cloud has approximately 81 million data points. We use DBSCAN to detect outliers in particular noise artefacts produced by the 3D scanner when recording the 3D point cloud. In practice if the dataset is processed using serial algorithm, it may take a couple of days. Therefore, it is imperative to have a parallel implementation of DBSCAN, which not only improves the performance, but also uses storage and memory requirements in an efficient manner. Another requirement is to have an implementation that adequately exploits execution environments of HPC. In order to support the application, HPDBSCAN [27] implementation is used. It is an initiative of Juelich that provides parallel implementation of DBSCAN. For efficient data storage and access, it uses the HDF5 data format.

We deployed this application on XSEDE infrastructure. We specifically used the BlackLight cluster that is deployed at Pittsburgh Supercomputing Center (PSC). A UNICORE service instance has been deployed and linked with the XSEDE-wide GFFS. Before the job execution, the dataset is placed on the GFFS node at the Indiana University's Mason cluster. The data staging was done by using the UNICORE's GFFS extension that copies data from the file system space to the local job's working directory.




```
<sweep:Sweep>
    <sweep:DocumentNode> ..
    <sweep:Match>//jsdl-posix:Argument[$e]</sweep:Match> </sweep:DocumentNode>
    <sweepfunc:LoopInteger start = "280" end="320" step="10">
    <sweep:Sweep>  ..
     <sweep:DocumentNode> ..
      <sweep:Match>//jsdl-posix:Argument[$M]</sweep:Match> </sweep:DocumentNode>
      <sweepfunc:LoopInteger start = "80" end="120" step="10">
    </sweep:Sweep>
</sweep:Sweep>
```

FIG. 6.1. *HPDBSCAN representation in the JSDL Parameter Sweep format depicting application with arguments: epsilon (e) and MinPoints (M) sweeping through a range of values.*

The identification of anomalies from the point cloud dataset is the main objective of the clustering application. This requires to find an optimal set of application arguments: MinPoints M, and epsilon (also called radius) e which influence the clustering of new point cloud instances or different variants of the same data, respectively. In terms of data mining this phase is called post-processing. The discovery of optimal arguments is achieved by analyzing each of the completed job's output which contains the cluster distribution and noise factor. Within the output, the criterion is to select the job configuration containing the minimum noise factor combined with the best cluster distribution. The whole process of optimization requires multiple manual runs

TABLE 6.1
The user perspective of the Data analysis lifecycle using manual and automated mechanisms.

Access Mode / Execution Phase	**Data Transfer**	**Data Processing**	**Post Processing**
Manual	SCP, GridFTP, ByteIO, FTP	Job script for every different resource and batch system	Create script manually for every variation
Middleware (UNICORE & the GFFS)	Automated through the supported data transfer protocols	One JSDL instance for all kind of backends	Single JSDL-PS template for the specified parametric variations

of the same application but with different M and e values. In order to avoid that users need to run these multiple jobs manually, the extended JSDL Parameter Sweep implementation which is provided by UNICORE's execution back-end was used. This allows using just a single job request which is not only more convenient for the user, but also faster, reproducible and less error-prone. The parameter sweep implementation serves the requirement R6-Parameter Sweep. Even though the user submits only one job, multiple child jobs are automatically generated according to the number of parameter iterations and nested sweeps. Figure 6.1 shows the sample HPDBSCAN JSDL job description making use of the parameter. The sweep factor of epsilon (e) and MinPoints (M) shown in Figure 6.1 will spawn 25 jobs in total with each generating a separate output.

Table 6.1 summarize the steps user need to perform data analysis in manual (script-based) and middleware-hosted environment. It is also evident from the illustration that the use of JSDL and JSDL-PS is more intuitive and avoids a need to write custom job requests for each flavor of the target resource management system. Furthermore, the data transfer event here applies to the pre-execution and fetch outputs phase.

7. Related Work. In this section we present the related job execution middle- ware technologies which are integrated with distributed file systems as well as work related to DBSCAN.

GridFTP [31] is one of the major data transfer protocols used in today's scientific and commercial data infrastructures. Specifically, GlobusOnline [25] data transfer service is mainly using this protocol to move data across widely distributed endpoints. UNICORE's GridFTP extension [7] helps scientists to submit job executions on UNICORE and using GridFTP-based data endpoints for data stagings. From the implementation perspective, both the GridFTP and GFFS extensions are integrated following the same approach, that is by using the XNJS programmatic interfaces. In the case of GridFTP integration, the clients requiring UNICORE servers to perform data staging on user's behalf, need to send at least X.509 proxy certificate chain along with the job submission request. For the GFFS the entities communicating the job request to the server must send a set of SAML assertions.

The GlobusOnline [25] service is a web portal to help end users perform GridFTP based high performance data transfers across different data endpoints. From the data management aspect, GlobusOnline and the GFFS are sharing a common set of features. By bridging the data access and processing (i.e. the job submission and execution, and execution service mount-point) services simultaneously distinguish the GFFS from GlobusOnline. The processing part is capable of attaching high performance (GenesisII) and high throughput computing (UNICORE) in a standard way. A very positive aspect of GlobusOnline is usability as it offers a ready to use data transfer service through a common web brower, whereas the GFFS user interaction is native desktop-based, which is not very intuitive and responsive as compare to browser-based applications.

ARC [20] is a middleware suite used by high throughput computing communities. ARC's integration with [26] and DDM (Distributed Data Management) [13] solutions are mostly used by the ATLAS [13] particle physics community at the Large Hadron Collider. dCache and DDM are distributed data management platforms providing storage and retrieval of huge amounts of data. dCache and the GFFS share mostly the same set of scenarios, but the major difference is that the GFFS expose its interface via RNS and ByteIO, whereas dCache is accessed through the SRM [9] interface.

WS-PGRADE / gUSE (grid and cloud user support environment) [30] is an open source scientific gateway framework that allows access to heterogeneous grid and cloud resources. gUSE provides a client extension in the form of DCI bridge [1] to the GFFS by invoking Genesis II clients. It is much similar to the way UNICORE integrates the GFFS. The frame- work provides access to UNICORE and ARC job submission services through

the OGSA-BES interface.

In the context of worfklow (e.g. Taverna [38], Kepler [11], etc.) enabling data mining methods on distributed computing infrastructures. Da Silva et al. [17] describe workflows with serial implementation of DBSCAN. According to our understanding their approach is not using the parallel DBSCAN implementation and in contrast to our approach that is intended for production usage in a high performance computing environment, the paper rather describes a research project than a production implementation.

PDSDBSCAN-D [39] is an implementation of DBSCAN, based on the MPI and OpenMP frameworks. According to [27] the HPDBSCAN application is more performant on various earth science data sets, among which the points cloud data is one. It performs better due to efficient pre-processing of spatial cells and use of density-based chunking to balance the local computation load on each node. Furthermore, HPDBSCAN uses the HDF5 [4] data format to store data and uses its library for achieving better parallel input and output performance.

8. Conclusion. In this paper, we have derived and implemented an integrated architecture which covers a set of requirements for providing transparent, secure and interoperable data processing tasks. Also provide these tasks access to the datasets managed by the Genesis II's Global Federated File System (GFFS). This is mainly achieved by the technical integration of UNICORE and the GFFS. The most important requirements are: R1 expresses a need for a secure trust delegation model, but should be standards-based and extensible (R2). As part of the integration the GFFS uses the SAML-based profile provided by the UNICORE's execution service. On the other hand, we extended UNICORE's identity validation that understand the GFFS-based requests containing multi-chain delegation assertions. R2 is also fulfilled by having the standards-based job execution and data management interfaces through the OGSA-BES and RNS specifications, respectively. The ByteIO standard is used to manage the data transport, thus the functionality implements R3.

While jobs are managed by UNICORE execution services, its internal service implementation is taking care of any status update delays through time out based probes against the target resource management system. In addition to that, the execution service also handles gracefully if the parallel file system on which the job's working data is stored becomes temporarily unresponsive, quite normal in production environment. The requirement R5 is served in this case.

For the Extensible resource and job model requirement R7, the resource model of the OGSA-Basic Execution Service (BES) and Job Submission and Description Language (JSDL) standards are extensible. But it will be only helpful if the compliant implementations are with minimal effort supporting the standard-allowed extensions. The technologies in our focus, UNICORE and the GFFS, are providing server and client side APIs to easily extend the resource model. This feature will be much more useful for community specific science gateways and next generation infrastructures with varying requirements. The XSEDE infrastructure has been used to demonstrate our implementation and data analysis excursion. This would require any technology and users entering the domain of an infrastructure should abide by its security model and its policies. With the GFFS client and UNICORE-based server, we used XSEDE-provided credentials to execute data processing jobs on a production deployment.

In our observation, most of the machine learning and data mining job submissions are parametric in nature, thus they need to be running multiple times. UNICORE's standard-based parameter sweep implementation helps to support our point cloud data clustering tasks. If we are able to represent the HPDBSCAN application requirement through JSDL and its parameter sweep extension, then any other data mining application can easily be supported. For the sake of implementation validity, we are analysing other methods of data mining, for example classification algorithms. One usability issue with the UNICORE's parametric sweep implementation is to produce a single job output based on some user specified criteria, which is currently not supported. The realization of this feature will reduce an overhead for data scientists to manually sort and merge the resultant job outputs. We intend to support this feature through a rule-based convergence of all the results from different parametric jobs into a single meaningful output. Another useful aspect is to avoid submitting multiple jobs to the batch system and rather use its internal feature of chaining multiple jobs. By enabling this feature the management and monitoring of complex job composites can be much more intuitive and usable.

REFERENCES

[1] *DCI Bridge Manual - v3.7.4.* http://sourceforge.net/projects/guse/files/3.7.4/Documentation/. [Online; accessed 29-Dec-2015].

[2] *FUSE (Filesystem in userspace).* https://github.com/libfuse/libfuse. [Online; accessed 29-Dec-2015].

[3] *GNU Octave.* https://www.gnu.org/software/octave/. [Online; accessed 31-Dec-2015].

[4] *Hierarchical data format version 5.* http://www.hdfgroup.org/HDF5. [Online; accessed 29-Dec-2015].

[5] *The OASIS Web Services Interoperability (WS-I).* http://www.ws-i.org/. [Online; accessed 29-Dec-2015].

[6] *TORQUE Resource Manager.* http://www.adaptivecomputing.com/products/open-source/torque-resource-manager/. [Online; accessed 31-Dec-2015].

[7] *UNICORE/X Manual.* https://www.unicore.eu/documentation/manuals/unicore6/files/unicor ex/unicorex-manual.html. [Online; accessed 29-Dec-2015].

[8] A. Anjomshoaa et al., *Job Submission Description Language (JSDL) Specification, Version 1.0.* Open Grid Forum, GFD-R.136, July 2008.

[9] A. Sim et al., *The Storage Resource Manager Interface Specification, Version 2.2.* Open Grid Forum, GFD-R-P.129, May 2008.

[10] A. Streit et al., *Unicore 6 recent and future advancements*, Annals of Telecommunications, 65 (2010), pp. 757–762.

[11] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock, *Kepler: an extensible system for design and execution of scientific workflows*, in Proceedings. 16th International Conference on Scientific and Statistical Database Management., June 2004, pp. 423–424.

[12] B. Dillaway et al., *HPC Basic Profile, Version 1.0.* Open Grid Forum, GFD-R-P.114, Aug 2007.

[13] G. Behrmann, D. Cameron, M. Ellert, J. Kleist, and A. Taga, *ATLAS DDM integration in ARC*, Journal of Physics: Conference Series, 119 (2008).

[14] K. Benedyczak, P. Bała, S. van den Berghe, R. Menday, and B. Schuller, *Key aspects of the UNICORE 6 security model*, Future Generation Computer Systems, 27 (2011), pp. 195 – 201.

[15] A. Bialecki, M. Cafarella, D. Cutting, and O. O'MALLEY, *Hadoop: a framework for running applications on large clusters built of commodity hardware.* http://hadoop.apache.org/. [Online; accessed 29-Dec-2015].

[16] D. Box et al., *Simple Object Access Protocol (SOAP) 1.1.* http://www.w3.org/TR/2000/NOTE-SOAP-20000508/. [Online; accessed 29-Dec-2015].

[17] R. F. da Silva, G. Juve, E. Deelman, T. Glatard, F. Desprez, D. Thain, B. Tovar, and M. Livny, *Toward fine-grained online task characteristics estimation in scientific workflows*, in Proceedings of the 8th Workshop on Workflows in Support of Large-Scale Science, WORKS '13, New York, NY, USA, 2013, ACM, pp. 58–67.

[18] J. Dean and S. Ghemawat, *Mapreduce: Simplified data processing on large clusters*, Commun. ACM, 51 (2008), pp. 107–113.

[19] B. Demuth, B. Schuller, S. Holl, J. Daivandy, A. Giesler, V. Huber, and S. Sild, *The unicore rich client: Facilitating the automated execution of scientific workflows*, 2013 IEEE 9th International Conference on e-Science, 0 (2010), pp. 238–245.

[20] M. Ellert, M. Grønager, A. Konstantinov, B. Kónya, J. Lindemann, I. Livenson, J. L. Nielsen, M. Niinimäki, O. Smirnova, and A. Wäänänen, *Advanced resource connector middleware for lightweight computational grids*, Future Gener. Comput. Syst., 23 (2007), pp. 219–240.

[21] M. Ester, H. peter Kriegel, J. Sander, and X. Xu, *A density-based algorithm for discovering clusters in large spatial databases with noise*, AAAI Press, 1996, pp. 226–231.

[22] D. B. et al., *Web Services Addressing (WS-Addressing).* http://www.w3.org/Submission/ws-addressing/. [Online; accessed 29-Dec-2015].

[23] M. D. et al., *JSDL Parameter Sweep Job Extension.* Open Grid Forum, GFD-R-P.149, May 2009.

[24] F. Bachmann et al., *XSEDE Architecture Level 3 Decomposition*, Dec 2012.

[25] I. Foster, *Globus online: Accelerating and democratizing science through cloud-based services*, IEEE Internet Computing, 15 (2011), pp. 70–73.

[26] P. Fuhrmann and V. Gülzow, *dCache, Storage System for the Future*, in Euro-Par 2006 Parallel Processing, W. Nagel, W. Walter, and W. Lehner, eds., vol. 4128 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2006, pp. 1106–1113.

[27] M. Götz, M. Richerzhagen, C. Bodenstein, G. Cavallaro, P. Glock, M. Riedel, and J. A. Benediktsson, *On Scalable Data Mining Techniques for Earth Science*, Procedia Computer Science, 51 (2015), pp. 2188–2197.

[28] A. Grimshaw, M. Morgan, and A. Kalyanaraman, *GFFS - The XSEDE Global Federated File System*, Parallel Processing Letters, 23 (2013), p. 1340005.

[29] I. Foster et al., *OGSA Basic Execution Service (BES), Version 1.0*, Nov 2008.

[30] P. Kacsuk, Z. Farkas, M. Kozlovszky, G. Hermann, A. Balasko, K. Karoczkai, and I. Marton, *WS-PGRADE/gUSE Generic DCI Gateway Framework for a Large Variety of User Communities*, Journal of Grid Computing, 10 (2012), pp. 601–630.

[31] I. Mandrichenko, W. Allcock, and T. Perelmutov, *GridFTP v2 Protocol Description.* Open Grid Forum, GFD-R-P.047, May 2005.

[32] S. Memon, S. Holl, B. Schuller, M. Riedel, and A. Grimshaw, *Enhancing the performance of scientific workflow execution in e-science environments by harnessing the standards based parameter sweep model*, in Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery, XSEDE '13, New York, NY, USA, 2013, ACM, pp. 56:1–56:7.

[33] S. Memon, M. Riedel, F. Janetzko, B. Demeler, G. Gorbet, S. Marru, A. Grimshaw, L. Gunathilake, R. Singh,

N. Attig, and T. Lippert, *Advancements of the ultrascan scientific gateway for open standards-based cyberinfrastructures*, Concurrency and Computation: Practice and Experience, 26 (2014), pp. 2280–2291.

[34] S. Memon, M. Riedel, C. Koeritz, and A. Grimshaw, *Interoperable job execution and data access through unicore and the global federated file system*, in Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2015 38th International Convention on, May 2015, pp. 269–274.

[35] M. Morgan, *ByteIO Specification 1.0.* Open Grid Forum, GFD-R-P.87, Oct 2006.

[36] M. Morgan, A. Grimshaw, and O. Tatebe, *RNS Specification 1.1.* Open Grid Forum, GFD-R-P.171, December 2010.

[37] M. Morgan and O. Tatebe, *RNS 1.1 OGSA WSRF Basic Profile Rendering 1.0.* Open Grid Forum, GWD-R.172, December 2010.

[38] T. Oinn, M. Greenwood, M. Addis, M. N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. R. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe, *Taverna: lessons in creating a workflow environment for the life sciences*, Concurrency and Computation: Practice and Experience, 18 (2006), pp. 1067–1100.

[39] M. A. Patwary, D. Palsetia, A. Agrawal, W.-k. Liao, F. Manne, and A. Choudhary, *A New Scalable Parallel DBSCAN Algorithm Using the Disjoint-set Data Structure*, in Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC '12, Los Alamitos, CA, USA, 2012, IEEE Computer Society Press, pp. 62:1–62:11.

[40] M. Riedel, M. Goetz, M. Richerzhagen, P. Glock, C. Bodenstein, A. Memon, and M. Memon, *Scalable and parallel machine learning algorithms for statistical data mining - practice amp; experience*, in Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2015 38th International Convention on, May 2015, pp. 204–209.

[41] S. Cantor et al., *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0.* OASIS Standard saml-core-2.0-os, March 2005.

[42] S. Graham et al., *Web Services Resource 1.2 (WS-Resource)*, April 2006.

[43] B. Schuller, R. Menday, and A. Streit, *A versatile execution management system for next-generation unicore grids*, in Euro-Par Workshops, 2006, pp. 195–204.

[44] G. Wasson and M. Humphrey, *HPC File Staging Profile, Version 1.0.* Open Grid Forum, GFD-R-P.135, Jun 2008.

[45] A. B. Yoo, M. A. Jette, and M. Grondona, *Slurm: Simple linux utility for resource management*, in Job Scheduling Strategies for Parallel Processing, Springer, 2003, pp. 44–60.

Edited by: Enis Afgan
Received: December 21, 2015
Accepted: March 31, 2016