SAGA: A standardized access layer to heterogeneous Distributed Computing Infrastructure

Article · March 2015

DOI: 10.1016/j.softx.2015.03.001

Andre Merzky
Rutgers, The State University of New Jersey
81 PUBLICATIONS **1,228** CITATIONS
SEE PROFILE

Ole Weidner
Louisiana State University
12 PUBLICATIONS **137** CITATIONS
SEE PROFILE

Shantenu Jha
Rutgers, The State University of New Jersey
252 PUBLICATIONS **1,989** CITATIONS
SEE PROFILE

Available online at www.sciencedirect.com
ScienceDirect
SoftwareX ▮ (▮▮▮▮) ▮▮▮–▮▮▮
www.elsevier.com/locate/softx

SAGA: A standardized access layer to heterogeneous Distributed Computing Infrastructure

Andre Merzky, Ole Weidner, Shantenu Jha*

RADICAL & ECE, Rutgers University, NJ, USA

Received 23 February 2015; received in revised form 16 March 2015; accepted 16 March 2015

Abstract

Distributed Computing Infrastructure is characterized by interfaces that are heterogeneous—syntactically and semantically. SAGA represents the most comprehensive community effort to date to address the heterogeneity by defining a simple, uniform access layer. In this paper, we describe the basic concepts underpinning its design and development. We also discuss RADICAL-SAGA which is the most widely used implementation of SAGA.

RADICAL-SAGA

`License:`	`MIT`
`Release Name:`	`SAGA-Python`
`Programming Language:`	`Python` (> = 2.7)
`Dependencies:`	`apache-libcloud`, `radical.utils`
`Repository:`	http://www.github.com/radical-cybertools/saga-python/ git@github.com:radical-cybertools/saga-python.git
`Documentation:`	http://saga-python.readthedocs.org/
`Tutorial:`	http://saga-python.readthedocs.org/en/latest/tutorial/
`Developers List:`	saga-devel@groups.google.com
`User List:`	saga-user@groups.google.com
`Bug Tracker:`	http://www.github.com/radical-cybertools/saga-python/issues/

1. Introduction

High-Performance Distributed Computing Infrastructure (DCI) is a important enabler of science and engineering applications. A fundamental technical challenge towards a comprehensive, balanced and flexible DCI is the requirement to support a broad range of application scenarios and modalities on a range of platforms with varying performance. Middleware services and tools have to be designed to support a wide-range of access and usage modalities over a shared set of resources. In turn, the choice of middleware services and tools are thus an important determinant of how these infrastructures will be utilized.

In spite of the impressive scientific advances in DCI capabilities, (the Higgs-discovery being a prominent example), the usage modes of DCI have not evolved significantly. As demonstrated by D. Katz et al. [1], the usage modes of applications leveraging the TeraGrid (in its final year, before transitioning to XSEDE) are narrow, and in many cases limited to single localized resources.

There are several contributing factors. One primary factor is heterogeneity: heterogeneity of interfaces (syntactical) and capabilities (semantic) is a fundamental attribute of DCI. Heterogeneity arises at multiple levels from the need to support diverse application requirements and usage scenarios over a shared set of resources which is after all the *raison d'etre* of DCI.

* Corresponding author.
E-mail addresses: andre@merzky.net (A. Merzky), ole.weidner@rutgers.edu (O. Weidner), shantenu.jha@rutgers.edu (S. Jha).

http://dx.doi.org/10.1016/j.softx.2015.03.001
2352-7110/© 2015 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

Another contributing factor to the problem of DCI complexity and limited "novel usage" is the fundamental issue that individual systems that constitute DCI retain a static model of resource utilization. This is evidenced by the fact that supercomputing centers have their user and runtime environments tuned to support mostly single-job oriented workloads, and thus still present a batch-queue, job-centric and static perspective of resources. The requirements and world view of applications however, have changed significantly [2–4], e.g., applications comprising multiple simulations such as those composed of a set of ensemble-members [1] and variants thereof such as multiple replica-based [5], now account for a non-negligible fraction of demand.

The economics and sociological factors that determine the success or uptake of middleware services and tools is complex. Although it is not possible to discuss or analyze the full complexity of this ecosystem, we point to a specific and salient feature: the need for a sustainable ecosystem of tools and services which balances the creativity of "a thousand flowers bloom" approach, with the reality of the cost of maintaining a diverse and large suite of tools that address partially overlapping and similar needs.

This is a starting motivation for RADICAL Cybertools [6] which provide hitherto missing capabilities to address the aforementioned three fundamental requirements: (i) a mechanism to manage the heterogeneity inherent in DCI, whilst retaining the flexibility to exploit it for customized solutions necessary for performance at extreme-scales; (ii) flexible resource management techniques as required by current and future generation of applications, and (iii) lightweight, semantically tight and functionally well-defined building blocks, which in turn are extensible and can be interfaced with other building blocks and thus upon which a sustainable ecosystem of services and tools can be built.

RADICAL Cybertools are an integrated, abstractions-based suite of well-defined capabilities that are architected for scalable, interoperable and sustainable support of science on a range of HPDC systems. It currently consists of two components: (i) RADICAL-SAGA [7]: a lightweight interface that provides standards-based interoperable capabilities to the most commonly used functionalities required to develop distributed applications, tools and services, and (ii) RADICAL-Pilot [8]: a scalable and flexible system that supports application-level resource management. In this paper, we focus on the first requirement (to manage heterogeneity) that is provided as an implementation of the *'Simple API for Grid Applications'* (SAGA), referred to as RADICAL-SAGA.

2. SAGA API specification: motivation, scope and design

DCIs typically need to serve a considerably large, and importantly *diverse*, set of user communities, usage modes and application characteristics. Heterogeneity at the hardware resource layer, is also a reflection of the large set of resource providers with a diverse set of objectives. Collectively, this diversity imposes a large number of functional requirements on the DCI, which evolve over time and translate to a large number of (sometimes contradictory) design constraints. That makes defining a *simple* DCI software stack very challenging. Thus to engineer distributed computing systems effectively, we need abstractions that provide conceptual simplicity whilst needing abstractions to manage complexity inherent in distributed systems.

A well-defined, stable general purpose API is one critical element of the solution, as it exposes the basic distributed capabilities as an uniform access layer to the infrastructure layer as well as a building block for higher-level applications, capabilities and tools. We define an *Access Layer* as a software layer which provides abstractions to a (sub)set of capabilities of lower layers. In the given context, a DCI access layer provides *uniform* access to *heterogeneous* DCI.

The objective of the Simple API for Grid Applications (SAGA) is to provide this missing critical component in the distributed cyberinfrastructure ecosystem. While the API's name suggests its strong ties to Grid based DCIs, it is in fact a general purpose API for distributed resources—which is historically rooted in the Grid community. SAGA provides a level of semantic harmonization, hides resource access and usage complexity, supplements the incompleteness and lack-of-extensibility of DCI layers whilst promoting interoperability as first-class design objective for applications and tools.

2.1. *Design of the SAGA API*

There have been several earlier attempts to address the problem of heterogeneity using suitable access layers. The most notable of these are the Globus project (Globus API [9], CoG [10]), the RealityGrid [11] project and the Grid Application Toolkit (GAT) [12]. Extending the collective experiences from these projects with a number of use cases collected from end users, SAGA has the following design objectives:

- **Manage system heterogeneity:** usable on a wide variety of DCIs; handle the underlying system heterogeneity via a small but well-defined semantic scope and mandatory semantics.
- **Ease of use:** provide consistent and intuitive abstractions; build upon existing standards and best practices; support established best-practices.
- **Support common usage patterns:** *"make commonly used functionality simple"*; use-case driven design and scope; balance higher level abstractions versus semantic expressiveness; recognizable and intuitive abstractions and patterns.

Towards those design objectives, SAGA maintains POSIX and Unix shell semantics where appropriate, and builds upon other standards and implementations where possible, for example, SAGA employs commonly known abstractions (such as file systems, data streams, jobs). The API layout is oriented on other well-known APIs and standards, such as POSIX [13], Java CoG [10], GAT [12], GridRPC [14], DRMAA [15,16], JSDL [17] and others.

Table 1
The scope of the capability SAGA API as defined in the SAGA Core API specification [20], and in the current set of specifications of API extension packages [21–24].

API package	Scope (approximate)
Saga core API	
Jobs	`job description, create, suspend/resume, state, cancel, inspection, stage-in/out`
Namespace	`open, close, create, copy, move, delete, find`
Files	`stat, read, write, seek, scattered I/O, pattern based I/O, extended I/O`
Replicas	`list/add/remove/update_location, replicate, annotate, find`
Stream	`serve, connect, close, read, write, wait`
RPC	`open, call, in/out/inout parameter`
Saga API extensions	
Service discovery	`discover, describe`
Information service navigation	`search, describe, relate`
Adverts	`create, delete, annotate, find, attach`
Messages	`publish/subscribe, multicast, (un)reliable, (un)ordered, send, test, recv`
Resource	`acquire, release, inspect`

2.2. *Semantic scope of the SAGA API*

The *capability* scope of the SAGA API (Table 1) was derived from a set of use cases [18,19], and thus defined by the set of operations considered essential for the requirements of and beneficial to the development of typical distributed applications and tools [18,19]. That scope includes most prominently job and data management (the latter both as physical and logical data), and further covers resource information queries and communication abstractions, both essential to the programming of applications for heterogeneous DCIs. Table 1 documents the API scope by listing API packages and exemplary method names.

Applications outside the set of considered use cases may require (slightly or significantly) different functionality. The scope of the API was thus widened in an attempt to provide coverage beyond the scope of considered use cases. For most parts, that approach worked reasonably well, but it also led to some pollution of the API with semantics which has been rarely used, and/or has been difficult to implement. We nevertheless feel that this approach added to the stability of the API—only on very rare occasions, the semantic scope has shown to be limiting the applicability of the API.

To accommodate new use cases with significantly different semantic requirements, SAGA was designed to be extensible using new capability packages which were defined on top of existing packages. That mechanism has been used, for example, to add messaging capabilities to the API scope, and the ability to interact with information services.

2.3. *SAGA look-&-feel*

Orthogonal to the capability packages, the SAGA API specification addresses general programming concerns across all packages which provides a common *look-&-feel* to the capability packages for both current and future API extensions. This covers:

- asynchronous operations and events,
- multithreading and concurrent execution,
- caching (or latency hiding in general),
- auditing, logging and inspection.

As an example, the code in Listing 1 shows how asynchronous operation is rendered in SAGA (Python), a mechanism that applies to *all* API calls of the SAGA API. Table 2 lists some of these orthogonal API elements. The next section describes the parts of the SAGA API specification implemented in RADICAL-SAGA, and how the API semantics is mapped to the various distributed infrastructures and services.

2.4. *SAGA standardization and impact*

The SAGA API and its extensions have been published as a set of standard specifications in the Open Grid Forum (OGF) [18–20,25,21,22,24,23]. Several independent implementations exist, in Java [26], C++ [27], and Python [28].

SAGA is being used by projects which have to routinely implement and execute applications on heterogeneous resources. While alternative approaches exist (such as VINE [29], GAT [30], COG [31], and many projects which provide similar concepts in a point-wise manner [32]), the use of a standardized API promises to be less brittle, and to be more resilient to churn in both the resource layer and in the application and tool development itself. To cite one representative example, KEK researchers use SAGA as an abstraction layer for data access and management [33], and for application execution [34,35], on the heterogeneous NAREGI DCI [36]. KEK researchers have contributed to both the specification of SAGA, and to the implementation of SAGA.

3. RADICAL-SAGA as SAGA implementation

3.1. *Implementation architecture*

Managing backend heterogeneity, hiding system complexities, providing common patterns and simple, high-level abstractions, are the dominating objectives for RADICAL-SAGA. From an architecture perspective, there exist established design patterns to serve those objectives: providing one interface over

Table 2
The 'property' part of the SAGA API.

Property	Use cases	Syntactic rendering

Asynchronous Operations, Bulk Operations, Thread Safety	Latency hiding, Concurrency	`task = File.copy (target, ttype=saga.ASYNC)`
		`print task.state`
		`print task.progress`
		`task.wait ()`

Monitoring	Notifications, Inspection, Events	`job.add_callback (saga.job.STATE, my_cb)`
		`print job.get_attribute ('Memory')`

Permissions	Access control	`file.permission_allow (user_id, READ)`
		`file.permission_deny (user_id, WRITE)`

Session	Authentication Authorization	`c = saga.Context ('ssh', {'UserID':'name'})`
		`s = saga.Session (); s.add_context (c)`
		`js = saga.job.Service (url, session=s)`

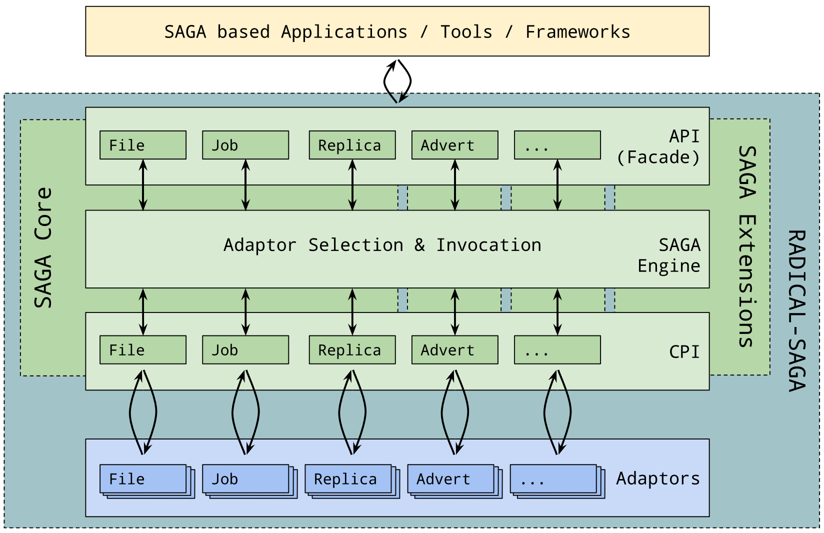

Fig. 1. Combining the *Facade* and *Adaptor* pattern yields a basic architecture: applications use a facade layer (the SAGA API); an adaptor layer translates the facade layer syntax to the respective backend syntax, while preserving the API semantics. The CPI (capability provider interface) is the internal interface to the adaptors, and structurally reflects the SAGA API.

another pre-existing interface is supported by the *Facade* pattern; providing an interface over multiple backend implementation is supported by the *Adaptor* pattern.

The high-level RADICAL-SAGA architecture is shown in Fig. 1—any architecture which conforms to both patterns will essentially be similarly structured, and it is indeed the basic architecture that all SAGA implementations follow. Beyond the assumption that SAGA is a *client side* API, neither the SAGA API structure nor any other part of the API specification prescribes nor implicitly suggests a specific architecture for its implementations.

3.2. *Implementation properties*

RADICAL-SAGA implements adaptors at the granularity of SAGA packages. The adaptor interface mirrors the SAGA package API, which makes the implementation of adaptors conceptually very simple. Those principles are followed for the SAGA Core API packages, as well as for API extension packages.

Available adaptors are *registered* via a static registry. Users can enable and disable adaptors via config files at runtime. Adaptor *selection* is performed dynamically, based on properties of the SAGA API objects (such as file or service URLs, communication protocols etc.). Adaptor instances are *bound* to individual API object instances, for the life time of those objects (object level binding). Other SAGA implementations use method level binding, which increases adaptor invocation flexibility, at the price of much higher implementation complexity (object state management cannot be localized to the adaptor layer).

The object level binding also motivates the package level granularity: adaptors usually cover the functionality of all API objects from a specific API package. Adaptor invocation is performed on a per-method basis, with dynamic call forwarding by the RADICAL-SAGA `saga.engine`. The engine also performs basic parameter checks, transparently translates synchronous to/from asynchronous calls, manages bulk operations, manages security credentials and configurations, etc. The engine is that part of RADICAL-SAGA which contains most of the semantic complexity of the "property" part of the SAGA API. Some code elements which are shared between different sets of adaptors are organized into a `radical.saga.utils` module, such as the management and sharing of secure con-

Table 3
RADICAL-SAGA adaptors.

API package	RADICAL-SAGA adaptors
Jobs	SGE, Torque, LSF, PBS, LoadLeveler, Slurm, Condor, SSH, GSISSH, Fork
Filesystem	GlobusOnline, SFTP, GSISFTP, GridFTP, HTTP, Local
Replica	iRods
Adverts	Redis
Resource	AWS/EC2, Local
API properties	async operations, bulk operations, async notifications (callbacks), inspection, authentication, authorization, concurrency, thread-safety

nection channels, or the implementation of callback and notification mechanisms. While applications are not expected to directly use that utility module, it greatly simplifies the implementation of adaptors.

```
def progress_cb (metric, value):
  print "progress: %s" % value
file = saga.filesystem.File (source_url)
task = file.copy (target_url, ttype=saga.ASYNC)

task.add_callback ('progress', progress_cb)
task.run()
task.wait()  # will print progress updates asynchronously

      if task.state != saga.Task.DONE :
      print "file copy failed: \%s" \% task.exception
      else :
      print "file copy done"
```

Listing 1: SAGA example for asynchronous operations and notifications (in Python)

The facade/adaptor approach implies a certain call stack overhead due to dynamic adaptor selection. The overhead is negligible compared to the normal call stack, and irrelevant in DCI environments. Other performance metrics are rate of operations (per second), and scaling with the number of sequential and concurrent operations. All those were found to be either dominated by (unavoidable) network latencies, bound by system limits, or to behave reasonably for the set of target use cases. Selected performance and scalability metrics and measurements for RADICAL-SAGA are discussed in [37].

The adaptors available in RADICAL-SAGA cover a wide range of back-ends; the set is being constantly extending based on internal and external user requirements i.e., adaptor development by request. Table 3 provides an overview of currently supported backends, and also lists the supported set of orthogonal API properties (covering all defined by GFD.90 with exception of the `Permission` interface). The mailing list [38] is used for exchanges between developers.

3.3. The RADICAL-SAGA Community

The SAGA community consists of a global set of developers and users; sometimes the distinction between developers and users is blurred. A quick survey of the contributors to the project shows contributions from Asia (Japan, Korea), US, UK and Europe. RADICAL-SAGA is currently used by a wide range of projects ranging from tool developers to individual users. Illustrative but not exhaustive examples include the PANDA group (ATLAS project) [39], Gateways projects [40–42], as well as multiple Bioinformatics projects [43,44]. A combination of standards-based, open source and distributed development, supported by multiple independent contributors, has contributed to the sustainability of RADICAL-SAGA.

Acknowledgments

SAGA is currently funded by National Science Foundation (NSF) SSE award "RADICAL Cybertools" NSF ACI-1440677. SAGA has been funded by a number of financial sources over the years, including UK EPSRC grant number GR/D0766171/1 (via OMII-UK) and the Cybertools project (PI Jha) NSF/LEQSF (2007-10)-CyberRII-01, HPCOPS NSF-OCI 0710874 award. SAGA has benefited from computer resources provided by TeraGrid TRAC award TG-MCB090174 (Jha), and support from the US NSF under Grant No. 0910812 to Indiana University for "FutureGrid: An Experimental, High-Performance Grid Testbed".

Appendix

Quick installation

```
$ virtualenv ve
$ source ve/bin/activate
(ve) $ pip install saga-python
(ve) $ sagapython-version
0.25
```

Listing 2: SAGA-Python installation steps

References

[1] Katz DS, Hart D, Jordan C, Majumdar A, Navarro JP, Smith W, et al. Cyberinfrastructure usage modalities on the TeraGrid, 2011 High-performance grid and cloud computing workshop, Proceedings of 2011 IPDPS Workshops, p. 927–934, 2011. http://dx.doi.org/10.1109/IPDPS.2011.239. http://www.ci.uchicago.edu/~dsk/papers/Cyberinfrastructure_Usage_Modalities.pdf.
[2] NSF XSEDE Annual Report. 2012, https://www.xsede.org/documents/10157/169907/2012+Q2+Annual+Report.pdf.
[3] Survey of cyberinfrastructure needs and interests of NSF-funded principal investigators, https://scholarworks.iu.edu/dspace/handle/2022/9917.
[4] XROADS User Requirements, http://www.ci.uchicago.edu/research/papers/CI-TR-10-0811.
[5] Rhee YM, Pande VS. Multiplexed-replica exchange molecular dynamics method for protein folding simulation. Biophys J 2003;84(2):775–86. [Online]. Available: http://www.biophysj.org/cgi/content/abstract/84/2/775.
[6] RADICAL Cybertools, http://radical-cybertools.github.io/.

[7] RADICAL-SAGA, https://github.com/radical-cybertools/radical-saga.
[8] RADICAL-Pilot, https://github.com/radical-cybertools/radical-pilot.
[9] The Globus Project, http://www.globus.org/.
[10] Von Laszewski G, Foster I, Gawor J, Lane P. A java commodity grid kit'. Concurrency Comput Pract Exp 2001;13(8–9):645–62.
[11] The RealityGrid Project, 2005, http://www.realitygrid.org/.
[12] Allen G, Davis K, Dramlitsch T, Goodale T, Kelley I, Lanfermann G, et al. The GridLab Grid Application Toolkit. In: High-performance distributed computing, international symposium on. IEEE Computer Society; 2002. p. 411ff.
[13] The Open Group, http://pubs.opengroup.org/onlinepubs/9699919799/.
[14] Nakada H, Matsuoka S, Seymour K, Dongarra J, Lee C, Casanova H. A Grid RPC Model and API for End-User Applications, Grid Forum Document GFD.52, Open Grid Forum, OGF Recommendation Document, 2007, http://www.ggf.org/documents/GFD.52.pdf.
[15] Troger P, Rajic H, Haas A, Domagalski P. Standardization of an API for distributed resource management systems. In: Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE international symposium on, IEEE; 2007. p. 619–26.
[16] Rajic H, Brobst R, Chan W, Ferstl F, Gardiner J, Haas A, et al. Distributed resource management application API specification 1.0. In: Grid Forum Document GFD.133, Open Grid Forum, OGF Recommendation Document, 2008, http://www.ggf.org/documents/GFD.133.pdf.
[17] Anjomshoaa A., Brisard F., Drescher M., Fellows D., Ly A., McGough S., et al. Job submission description language (JSDL) specification, version 1.0. In: Grid Forum Document GFD.56, Open Grid Forum, OGF Recommendation Document, 2005, http://www.ggf.org/documents/GFD.56.pdf.
[18] Jha S, Merzky A. A collection of use cases for a simple API for grid applications. In: Grid Forum Document GFD.70, Open Grid Forum, OGF Informational Document, 2006, http://www.ggf.org/documents/GFD.70.pdf.
[19] Jha S, Merzky A. A requirements analysis for a simple API for grid applications. In: Grid Forum Document GFD.71, Open Grid Forum, OGF Informational Document, 2006, http://www.ggf.org/documents/GFD.71.pdf.
[20] Goodale T, Jha S, Kaiser H, Kielmann T, Kleijer P, Merzky A, et al. A Simple API for grid applications (SAGA). In: Grid Forum Document GFD.90, Open Grid Forum, OGF Proposed Recommendation Document, 2007, open Grid Forum.
[21] Merzk A, SAGA API extension: Advert API. In: Grid Forum Document GFD.177, Open Grid Forum, OGF Proposed Recommendation Document, 2011, http://www.ggf.org/documents/GFD.177.pdf.
[22] Merzky A, SAGA API extension: Message API. In: Grid Forum Document GFD.178, Open Grid Forum, OGF Proposed Recommendation Document, 2011, http://www.ggf.org/documents/GFD.178.pdf.
[23] Fisher S, Wilson A, Pavnethan A, SAGA API dxtension: Servic discovery API. In: Grid Forum Document GFD.144, Open Grid Forum, OGF Proposed Recommendation Document, 2011, http://www.ggf.org/documents/GFD.144.pdf.
[24] Fisher S., Wilson A., SAGA API extension: Information system navigator. In: Grid Forum Document GFD.195, Open Grid Forum, OGF Proposed Recommendation Document, 2011, http://www.ggf.org/documents/GFD.195.pdf.
[25] Merzky A, Luckow A, Simmel D, SAGA extension: Checkpoint and recovery API (CPR). In: Grid Forum Document GFD.XX, Open Grid Forum, Draft for an OGF Proposed Recommendation Document, 2008, http://www.ogf.org/OGF27/materials/1767/saga_cpr.pdf.
[26] J-SAGA. [Online]. Available: software.in2p3.fr/jsaga/.
[27] SAGA-C++. [Online]. Available: http://saga-project.github.io/saga-cpp/.
[28] SAGA-Python, http://www.github.com/radical-cybertools/saga-python/.
[29] Russell M, Dziubecki P, Grabowski P, Krysinki M, Kuczyski T, Szjenfeld D, Tarnawczyk D, Wolniewicz G, Nabrzyski J. The Vine Toolkit: A Java Framework for Developing Grid Applications. In: Parallel processing and applied mathematics. Wyrzykowski R, Dongarra J, Karczewski K, Wasniewski J, editors. Lecture notes in computer science, vol. 4967. Berlin, Heidelberg: Springer; 2008. p. 331–40.
[30] The Grid Application Toolkit (Java-GAT). [Online]. Available: http://www.cs.vu.nl/ibis/javagat.html.
[31] The Java-CoG Kit. [Online]. Available: http://wiki.cogkit.org/.
[32] JGlobus. [Online]. Available: https://dev.globus.org/wiki/CoG_jglobus.
[33] Kawai Y, Iwai G, Sasaki T, Watase Y. SAGA-based file access application over multi-filesystem middleware. In: Proceedings of the 19th ACM international symposium on high performance distributed computing, ser. HPDC'10. New York, NY, USA: ACM; 2010. p. 622–6.
[34] Kawai Y, Iwai G, Sasaki T, Watase Y, SAGA-based application to use resources on different grids. In: Proc. international symposium on grids and clouds (ISGC), 2011.
[35] Iwai G, Kawai Y, Sasaki T, Watase Y. SAGA-based user environment for distributed computing resources: A universal grid solution over multi-middleware infrastructures. Procedia Comput Sci 2010;1(1):1545–51.
[36] National Reserach Grid Initiative, Japan. [Online]. Available: http://www.naregi.org/project/index_e.html.
[37] RADICAL-SAGA Performance (Wiki), http://github.com/radical-cybertools/saga-python/wiki/Performance-of-saga-python.
[38] SAGA Developers Mailing List. [Online]. Available: saga-devel@groups.google.com.
[39] Klimentov A, Buncic P, De K, Jha S, Maeno T, Mount R, et al. Next generation workload management system for big data on heterogeneous distributed computing. In: 16th international workshop on advanced computing and analysis techniques in physics research (ACAT), September 2014, Keynote and paper, https://indico.cern.ch/event/258092/page/17.
[40] Maddineni S, Kim J, El-Khamra Y, Jha S. Distributed application runtime environment (DARE): A standards-based middleware framework for science-gateways. J Grid Comput 2012;10(4):647–64.
[41] Ardizzone V, Barbera R, Calanducci A, Fargetta M, Ingrà E, La Rocca G, et al. A European framework to build science gateways: Architecture and use cases. In: Proceedings of the 2011 teraGrid conference: Extreme digital discovery. ACM; 2011. p. 43.
[42] Ardizzone V, Barbera R, Calanducci A, Fargetta M, Ingrà E, Porro I, et al. The DECIDE science gateway. J Grid Comput 2012;10(4):689–707.
[43] Raghotham A, S S, Kim N, Jha S, Kim J. Developing eThread pipeline using SAGA-pilot abstraction for large-scale structural bioinformatics. In: BioMed Research International, Vol. 2014. 2014. http://dx.doi.org/10.1155/2014/348725. Article ID 348725.
[44] Kim J, Maddineni S, Jha S. Advancing next-generation sequencing data analytics with scalable distributed infrastructure. Concurr Comput Pract Exp 2014;26(4):894–906.