RADICAL-Cybertools: Middleware Building Blocks for Scalable Science

Vivek Balasubramanian[1], Shantenu Jha[1,2], Andre Merzky[1] and Matteo Turilli[1]

[1] *Electrical & Computer Engineering, Rutgers University, Piscataway, NJ 08854, USA,*
[2] *Brookhaven National Laboratory*

Abstract

RADICAL-Cybertools (RCT) are a set of software systems that serve as middleware to develop efficient and effective tools for scientific computing. Specifically, RCT enable executing many-task applications at extreme scale and on a variety of computing infrastructures. RCT are building blocks, designed to work as stand-alone systems, integrated among themselves or with third-party systems. RCT enable innovative science in multiple domains, including but not limited to biophysics, climate science and particle physics, consuming hundreds of millions of core hours. This paper provides an overview of RCT, their impact, and the architectural principles and software engineering underlying RCT.

Keywords:
Middleware, Pilot System, Building Blocks

1. Motivation and significance

The design of middleware to support scientific computing has never been more challenging. The unprecedented complexity of middleware design and development arises from diversity in application requirements, disruptive changes in the resources and technology landscapes, intermixed with new discovery modalities and need for scalable computing.

Against this dynamic landscape, two critical question must be addressed: How can middleware be designed and implemented to meet the collective challenges of scale, new and diverse functionality, and usability? How can critical middleware components be designed to be sustainable software implementations while being forward looking and enable innovative capabilities?

RADICAL-Cybertools (RCT) [1] are a set of systems developed to address these challenges. RCT are building blocks, which can be used as a standalone system, or integrated with other RCT, or third-party tools to enable diverse functionalities. RCT offer several innovative features to support the design and implementation of middleware.

This paper takes a software perspective to present the overarching architectural paradigm of RCT and discuss the design and implementation of two cybertools: RADICAL-Pilot and Ensemble-Toolkit (EnTK). We outline the direct impact that RCT are having on domain sciences, focusing the discussion on architectural and design paradigms of middleware for scientific computing. In this way, we show how RCT further the "state-of-theory" and practice of scientific computing.

2. Software description

RCT have three main components: RADICAL-SAGA (RS) [2], RADICAL-Pilot (RP) [3] and RADICAL-Ensemble Toolkit (EnTK) [4, 5].

RS is a Python implementation of the Open Grid Forum SAGA standard GFD.90 [6], a high-level interface to distributed infrastructure components like job schedulers, file transfer and resource provisioning services. RS enables interoperability across heterogeneous distributed infrastructures, improving on their usability and enhancing the sustainability of services and tools. Ref. [2] details RS therefore, in the rest of the paper, we focus on RP and EnTK.

RP is a Python implementation of the pilot paradigm and architectural pattern [7]. Pilot systems enable users to submit pilot jobs to computing infrastructures and then use the resources acquired by the pilot to execute one or more tasks. These tasks are directly scheduled via the pilot, without having to queue in the infrastructure's batch system.

EnTK is a Python implementation of a workflow engine, specialized in supporting the programming and execution of applications with ensembles of tasks. EnTK executes tasks concurrently or sequentially, depending on their arbitrary priority relation. Tasks are scalar, MPI, OpenMP, multi-process, and multi-threaded programs that run as self-contained executables. Tasks are not functions, methods, threads or subprocesses.

RCT are designed to work both individually and as an integrated system, with or without third party systems. This requires a "Building Block" approach to their design and development, based on applying the traditional notions of modularity at system level [8]. The Building Block approach derives from the work on Service-oriented Architecture and its Microservice variants, and the component-based software development approaches where computational and compositional elements are explicitly separated [9, 10, 11, 12, 13].

AirFlow, Oozie, Azkaban, Spark Streaming, Storm, or Kafka are examples of tools that have a design consistent with the building blocks approach.

2.1. Software Architecture

Architecturally, all RCT consist of one or more subsystems, each with several components. Some components are used only in specific deployment scenarios, depending on both application requirements and resource capabilities. Components are stateless and some of them can be instantiated concurrently, enabling throughput scaling and fault tolerance. A communication overlay enables coordination of concurrent components, improving scalability at the cost of infrastructure-specific overheads.

2.1.1. RADICAL-Pilot

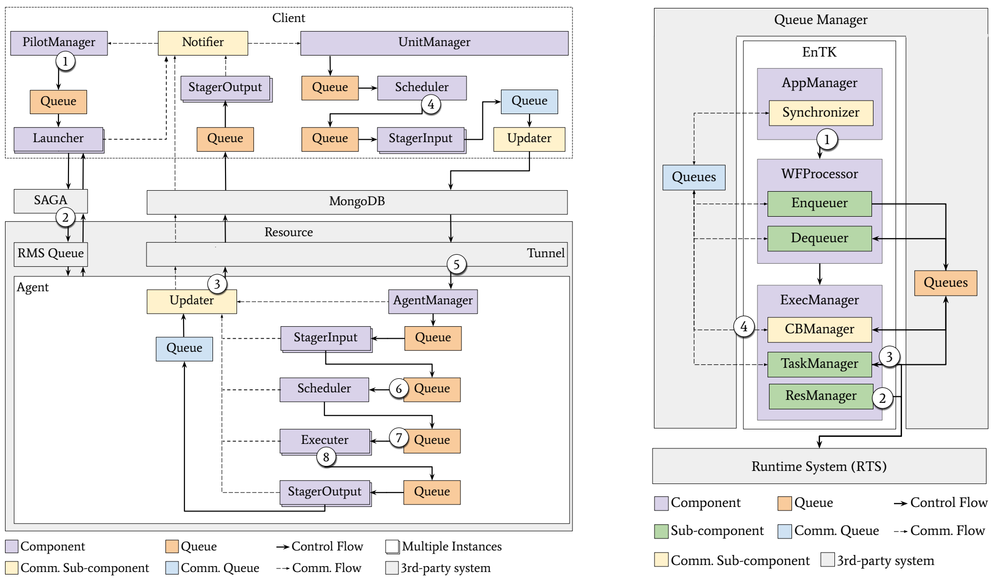
RP implements two abstractions: Pilot and Compute Unit (CU). Pilots are placeholders for computing resources, where resources are represented independent from architecture and topological details. CUs are units of work (i.e., tasks), specified by a program's executable alongside its resource and execution environment requirements.

Fig. 1a depicts RP's architecture. RP has two subsystems (white boxes), each with several components (purple and yellow boxes). Purple components manage pilots and CUs while yellow components manage the communication among components. Subsystems can execute locally or remotely, communicating and coordinating over TCP/IP, and enabling multiple deployment scenarios.

Client has two main components: PilotManager and UnitManager. PilotManager manages pilots and its Launcher component uses configuration files to tailor Agent's components to specific resources. Currently, configuration files are made available for all the XSEDE HPC machines, NCSA Blue Waters, NCAR Cheyenne, and ORNL Rhea, Titan and Summit. UnitManager manages CUs, using Scheduler to late bind CUs onto one or more pilots only when they become available on one or more machines. StagerInput copies CUs' input files to the machines on which each CU was scheduled.

Agent has four main components: StagerInput and StagerOutput for staging CUs' input/output data, Scheduler and Executer to schedule and execute CUs on pilots' resources. For performance optimization at scale, multiple instances of Stager and Executer can coexist in a single Agent, placed on head nodes, MOM/batch nodes, compute nodes, virtual machines, or any combination thereof.

Each Agent's component has a queue to support Pilot or CUs feeding (Fig. 1a, orange queues) and control messaging (Fig. 1a, blue queues). All



(a) (b)

Figure 1: Caption.

Queues support bulk communication and enable load balancing among concurrent components to obtain performance at scale.

RP data management moves, copies or links input files to CUs before execution and output files from CUs after execution. RP can select the most appropriate storage, depending on resource capabilities and CUs requirements. Data can be exclusive to CUs, or shared among them.

Client and Agent communicate via a queue instance rendered as a data structure of a database, preserving the semantic of the queues in Fig. 1a. Database's data are persistent, enabling *post mortem* profiling and analysis of RP executions.

Fig. 1a's numbers illustrate the resource acquisition and task execution process. PilotManager uses RS to queue a pilot as a job on the resource's batch System (Fig. 1a.1-2). Once scheduled, the pilot bootstraps RP Agent's components, using Updater to notify RP Client that is ready to execute tasks (Fig. 1a.3). Upon notification, UnitManager queues all the available tasks onto Client's Scheduler and, after staging files if required, tasks are queued to the RP Agent's Scheduler (Fig. 1a.4-6). Scheduler places tasks on suitable partitions of the pilot's resources and then queues tasks to the Executor so that, when those partitions of resource becomes available, tasks are executed (Fig. 1a.7). Executor sets up the environment required by each task and then spawns each task for execution (Fig. 1a.8).

2.1.2. RADICAL Ensemble Toolkit

EnTK implements three abstractions: task, stage and pipeline. Each task represents an executable program with information about its environment and data dependences. Stage is a set of tasks which have no mutual dependences and thus can execute concurrently. Pipeline is a list of stages where the ordering of the stages represents the ordering of their execution.

As with RP, EnTK enables concurrent and sequential execution at program level, not at function or method level. Each task's program can use parallelism, enabling concurrent execution of scalar, MPI, OpenMP, multi-process, and multi-threaded programs.

Fig. 1b shows the architecture of EnTK (white) with three components (purple), each with subcomponents dedicated to the management of EnTK's entities (green) or coordination of the entities' execution (yellow). The three components are AppManager, WFProcessor and TaskManager, respectively enabling workflow specification, workflow execution management, and workload management.

EnTK's API enables the development of ensemble-based applications with tasks, stages and pipelines, and the specification of resource requirements for the application execution. AppManager initializes EnTK and holds the global state of the application at runtime. AppManager is the sole stateful component of EnTK, allowing to restart other components upon failure, without interrupting the execution of the ensemble-based application.

WFProcessor queues and dequeues tasks pulled to and from the TaskManager in accordance with the task order specified by stages and pipelines. TaskManager interfaces with the RTS via ResManager and ExecManager. RTS provides capabilities to acquire resources and schedule tasks on those resources for execution. ResManager isolates RTS from EnTK, enabling fault-tolerance with persistent information about executed tasks. ExecManager schedules tasks on the resources, tracking the state of each task during execution. Currently, EnTK support only RP as RTS but it is designed to use other task-based RTS as, for example, Coasters or HTCondor.

Fig. 1b's numbers illustrate the execution of workflows in EnTK. Users instantiate an AppManager, define a set of resources on which to run their workflow, describe that workflow in terms of pipelines, stages and tasks and execute it. AppManager passes a copy of the workflow description to WFProcessor that, based on the priorities between stages and pipelines, uses Enqueuer to queue tasks that are ready for execution to TaskManager (Fig. 1b.1). Meanwhile, ResManager users the chosen runtime system to acquire the requested resources (Fig. 1b.2) and, once available, TaskManager uses those resources to execute the queued tasks (Fig. 1b.3) and dequeuing them once they have been executed. ExecManager uses queues to communicate the state of each task execution to AppManager (Fig. 1b.4). Note that AppManager is the only stateful component of EnTK: both WFProcessor and ExecManager can fail without loss of information about pending or completed tasks, but only about tasks that are executing at the instant of failure.

2.2. Software Functionalities

As a pilot system, RP decouples resource acquisition from task execution, enabling execution of tasks on a pilot without using the resource's queuing system. When compared to other pilot systems or tools that enable the execution of many-task applications on HPC systems, RP offers four unique features: (1) concurrent execution of heterogeneous tasks on the same and across pilots, execute single or multi core tasks within a single compute node, or across multiple nodes, isolating the execution of each tasks into a dedicated process and enabling concurrent execution of heterogeneous tasks by design; (2) support of the major HPC batch systems; (3) support of twelve methods to launch tasks; and (4) a general purpose architecture.

EnTK is a workflow engine that, compared to similar systems, allows to codify ensemble applications as pipelines of stages of tasks. Task interdependences may be determined by input/output data or control flow requirements. EnTK provides adaptive capabilities to change the ensemble specifications, creating new pipelines, stages and tasks, or changing the properties of those already defined, based on intermediate results obtained during execution. Further, pipelines can be paused while waiting to perform *ad hoc* computations. This enables the implementation of high-level application patterns as, for example, simulation-analysis [14] or replica exchange [15].

3. Illustrative Examples

An increasing trend is that application developers tend to develop their own tools tailored to their requirements. Ref. [**?**] enumerates in excess of 230 purported workflow systems. The proliferation of workflow systems has many implications for users and developers of workflow systems, and raises important questions: Is it possible to implement workflow systems in an agile fashion to provide flexibility and sharing of capabilities while not constraining functionality, performance, or sustainability? What are the necessary abstractions and capabilities that workflow system developers be provided with so as to prevent the need to develop *ab initio*.

The case of specialized domain-specific tools that support similar but not identical workflows provides an illustrative example. Building logically self-contained software blocks, lowers the technical barriers to their composability and thus allows designing domain-specific tools.

RCT can be extended to support multiple domain-specific tools. EnTK has enabled the development of workflow tools which provide domain specific functionalities. Each tool is characterized by a unique execution and coordination pattern and can serve multiple applications. The four ensemble-based workflow frameworks developed using EnTK and other RCT are: EXTASY [14], RepEx [15], HTBAC [16], and ICEBERG. Details can be found in Ref. [8]

ExTASY, RepEx, HTBAC represent three adaptive ensemble based domain-specific tools. They have benefited by using EnTK by not having to re-implement workflow processing, efficient task management and interoperable task execution capabilities on distinct and heterogeneous platforms. In turn, eight distinct adaptive ensemble workflows have been implemented using EnTK, which suggests that the "last mile customization" capability of the RCT-based building block approach.

4. Impact

The impact of RCT spans domain science, high-performance computing and software systems design. RCT serves a worldwide community of domain scientists and system engineers that actively contribute to the open source development of RCT. Currently, RCT supports a dozen active science projects across the USA and Europe. The size of projects varies from single PIs to very large international collaborations. Thus, there is intrinsic uncertainty in the number of both direct and indirect RCT users.

RCT has supported scientific advancements in computer science [17]; software engineering [8]; chemical [18] and biophysics [19, 16]; materials [16], earth and climate science [20, 5]; particle physics[21]; and drug discovery [22]. RADICAL-SAGA has mostly been integrated into end-to-end middleware solutions, RADICAL-Pilot has been used both as standalone system and integrated with other systems, while EnTK has served as workflow engine for domain-specific applications.

RADICAL-SAGA enables the Production ANd Distributed Analysis (PanDA) system to submit batch jobs to Titan and Summit, the two leadership class machines managed at the ORNL [23]. PanDA is the workload management system used by the ATLAS experiment to execute hundred of millions of jobs a year on both grid and HPC infrastructures [24]. The usage of ORNL resources constitutes $\approx 2 - 3\%$ of all of ATLAS computing. There are several thousand researchers that directly or indirectly use PanDA, and thereby RADICAL-SAGA. RADICAL-Pilot will be part of the PanDA workload management ecosystem on multiple HPC platforms.

RADICAL-SAGA was used to develop Science Gateways as part of the Distributed Application Runtime Environment (DARE) framework [25]. These gateways supported several projects, including DECIDE [?] and neuGRID [26], to study the early diagnosis of Alzheimer and other neurodegenerative diseases. In that capacity, RADICAL-SAGA enabled submission of jobs to distributed computing infrastructures managed by the European Grid Initiative (EGI). Recently, the emergence of toolkits such as Agave [27] which integrate identity management have provided higher-level solutions for Gateway developers, but they retain the RADICAL-SAGA based approach to job submission to distributed computing systems.

Since its first release in 2013, RP has supported around two dozen projects and 100 direct users. Half of these projects used RP has a standalone system to execute applications comprised of multiple tasks. Supporting distinct applications contributed to realize the growing importance of the ensemble computational model: HPC applications were replacing single task applications with ensembles of tasks to achieve significant performance gains on large-scale parallel machines.

As seen in Sec. 3, EnTK has enabled the development of four ensemble-based workflow tools which provide domain specific functionalities. ExTASY and RepEx implement advanced sampling algorithms using biomolecular simulations. Both use the EnTK API to implement diverse coordination patterns amongst ensembles of biomolecular simulations and analysis. HTBAC supports multiple algorithms that compute free-energy calculations that are critical to drug design and resistance studies. HTBAC allows the runtime adaptation at multiple levels: algorithms, pipelines and tasks within a pipeline. This capability has been demonstrated to reduce the time-to-solution by a factor 2.5 in controlled experiments on real drug candidates [28]. ICEBERG [20] supports scalable image analysis applications using multiple concurrent pipelines.

The Building Blocks approach helps to create systems that can support use cases both individually and as integrated, end-to-end solution [8]. This is important when supporting projects with multiple, distinct use cases. For example, ICEBERG has to support five use cases, each investigating a specific problem in the domain of polar science. Four of these use cases require the concurrent execution of pipelines but one requires only the execution of bag of tasks. The first four cases can use EnTK while the latter only RADICAL-Pilot. Importantly, all use cases can be served by the ICEBERG framework with a minor change in the private API to call EnTK or RP, depending on the use cases and therefore without any engineering overheads.

RCT are also a testbed for engineering research, mostly focused on foundational abstractions [29], architectural paradigms [7], application patterns [14, 5], and performance analysis of distributed middleware on diverse computing infrastructures [29, 16]. Among the most representative projects supported by RP as a standalone system, the Abstractions and Integrated Middleware for Extreme-Scale Science (AIMES) project enabled extreme-scale distributed computing via dynamic federation of heterogeneous computing infrastructures. We used RP to execute millions of tasks on both HPC and HTC resources, studying the federated behavior of multiple infrastructures, establishing the importance of integrating task and resource information in scheduling and placement decision making for federated supercomputers [30].

5. Conclusions

RCT is a small operation with at most two developers working on the systems at the same time. In order to implement RCT capabilities while supporting more than ten concurrent projects at any point in time, we adopted a specific methodology for the design, development and maintenance of RCT. This methodology is based on the Building Blocks approach, the use of git for distributed version control of the code base [1], and a tailored project management process.

Based on more than ten years of experience, our approach shows a sustainable and effective way to organize software development, promote community adoption and leverage the specific characteristics of the academic financial model. This signals the transition away from a development model based on end-to-end, monolithic solutions with stringent requirements on infrastructures' software stack. The RCT development model is based on small, independent systems, each with a well-defined capability and composable with other systems developed independently by different development teams. This approach enables a model of sustainability based on smaller and shorter funding sources but requires a certain convergence in the vision of diverse groups competing in the same research field.

Acknowledgments and Contributions

We thank all our users, collaborators, and present and past members of RADICAL. This paper was was supported primarily by NSF 1440677 and DOE ASCR DE-SC0016280. We acknowledge access to computational facilities: XSEDE resources (TG-MCB090174), Blue Waters (NSF-1713749), and DOE leadership machines via multiple INCITE and DD awards. Vivek

Balasubramanian was the lead developer of EnTK; Shantenu Jha is the PI of RADICAL-Cybertools; Andre Merzky is the lead developer of RADICAL-Pilot and RADICAL-SAGA; and Matteo Turilli is the lead architect of EnTK, co-architect of RADICAL-Pilot and wrote the bulk of the paper.

[1] Research in Advanced DIstributed Cyberinfrastructure and Applications Laboratory (RADICAL), RADICAL-Cybertools (RCT), `https://github.com/radical-cybertools`, Last accessed on 2019-03-31 (2019).

[2] A. Merzky, O. Weidner, S. Jha, Saga: A standardized access layer to heterogeneous distributed computing infrastructure, SoftwareX 1 (2015) 3–8.

[3] A. Merzky, M. Turilli, M. Maldonado, M. Santcroos, S. Jha, Using pilot systems to execute many task workloads on supercomputers, in: D. Klusáček, W. Cirne, N. Desai (Eds.), Job Scheduling Strategies for Parallel Processing: 22nd International Workshop, JSSPP 2018, Vancouver, BC, Canada, May 25, 2018, Revised Selected Paper, Springer, 2019, pp. 61–82.

[4] V. Balasubramanian, A. Treikalis, O. Weidner, S. Jha, Ensemble toolkit: Scalable and flexible execution of ensembles of tasks, in: 2016 45th International Conference on Parallel Processing (ICPP), IEEE, 2016, pp. 458–463.

[5] V. Balasubramanian, M. Turilli, W. Hu, M. Lefebvre, W. Lei, R. Modrak, G. Cervone, J. Tromp, S. Jha, Harnessing the power of many: Extensible toolkit for scalable ensemble applications, in: 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE, 2018, pp. 536–545.

[6] T. Goodale, S. Jha, H. Kaiser, T. Kielmann, P. Kleijer, G. Von Laszewski, C. Lee, A. Merzky, H. Rajic, J. Shalf, Saga: A simple api for grid applications. high-level application programming on the grid, Computational Methods in Science and Technology 12 (1) (2006) 7–20.

[7] M. Turilli, M. Santcroos, S. Jha, A comprehensive perspective on pilot-job systems, ACM Computing Surveys (CSUR) 51 (2) (2018) 43.

[8] M. Turilli, V. Balasubramanian, A. Merzky, I. Paraskevakos, S. Jha, Middleware building blocks for workflow systems, Computing in Science

& Engineering (CiSE) special issue on Incorporating Scientific Workflows in Computing Research Processes
https://arxiv.org/abs/1903.10057.

[9] D. Batory, S. O'Malley, The design and implementation of hierarchical software systems with reusable components, ACM Trans. Softw. Eng. Methodol. 1 (4) (1992) 355–398.

[10] D. Garlan, R. Allen, J. Ockerbloom, Architectural mismatch or why it's hard to build systems out of existing parts, in: Proceedings of the 17th International Conference on Software Engineering, ICSE '95, 1995, pp. 179–185.

[11] M. Lenz, H. A. Schmid, P. F. Wolf, Software reuse through building blocks, IEEE Software 4 (4) (1987) 34–42.

[12] S. Clemens, G. Dominik, M. Stephan, Component software: Beyond object-oriented programming, Addison-Westley: Boston, MA, USA.

[13] J.-G. Schneider, O. Nierstrasz, Components, scripts and glue, in: Software Architectures, Springer, 2000, pp. 13–25.

[14] V. Balasubramanian, I. Bethune, A. Shkurti, E. Breitmoser, E. Hruska, C. Clementi, C. Laughton, S. Jha, Extasy: Scalable and flexible coupling of md simulations and advanced sampling techniques, in: 2016 IEEE 12th International Conference on e-Science (e-Science), IEEE, 2016, pp. 361–370. doi:10.1109/eScience.2016.7870921.

[15] A. Treikalis, A. Merzky, H. Chen, T.-S. Lee, D. M. York, S. Jha, Repex: A flexible framework for scalable replica exchange molecular dynamics simulations, in: 2016 45th International Conference on Parallel Processing (ICPP), IEEE, 2016, pp. 628–637.

[16] J. Dakka, M. Turilli, D. W. Wright, S. J. Zasada, V. Balasubramanian, S. Wan, P. V. Coveney, S. Jha, High-throughput binding affinity calculations at extreme scales, BMC bioinformatics 19 (18) (2018) 482.

[17] A. Merzky, M. T. Ha, M. Turilli, S. Jha, Synapse: Synthetic application profiler and emulator, Journal of computational science 27 (2018) 329–344.

[18] C. Sampat, F. E. Bettencourt, Y. Baranwal, I. Paraskevakos, A. Chaturbedi, S. Karkala, S. Jha, R. Ramachandran, M. Ierapetritou, A parallel unidirectional coupled DEM-PBM model for the

efficient simulation of computationally intensive particulate process systems, Computers & Chemical Engineering 119 (2018) 128–142. doi:10.1016/j.compchemeng.2018.08.006.
URL `https://doi.org/10.1016/j.compchemeng.2018.08.006`

[19] A. Shkurti, I. D. Styliari, V. Balasubramanian, I. Bethune, C. Pedebos, S. Jha, C. A. Laughton, Coco-md: A simple and effective method for the enhanced sampling of conformational space, Journal of chemical theory and computation.

[20] I. Paraskevakos, M. Turrili, B. C. Gonçalves, H. J. Lynch, S. Jha, Workflow design analysis for high resolution satellite image analysis, arXiv preprint arXiv:1905.09766.

[21] D. Oleynik, S. Panitkin, M. Turilli, A. Angius, S. Oral, K. De, A. Klimentov, J. C. Wells, S. Jha, High-throughput computing on high-performance platforms: A case study, in: 2017 IEEE 13th International Conference on e-Science (e-Science), IEEE, 2017, pp. 295–304.

[22] D. W. Wright, B. A. Hall, O. A. Kenway, S. Jha, P. V. Coveney, Computing Clinically Relevant Binding Free Energies of HIV-1 Protease Inhibitors, Chemical Theory and Computation 10 (3) (2014) 1228–1241. doi:10.1021/ct4007037.

[23] Oak Ridge National Laboratory (ORNL), Leadership Computing Facility (OLCF), OLCF resources: Compute systems, `https://www.olcf.ornl.gov/olcf-resources/compute-systems/`, Last accessed on 2019-02-28 (2019).

[24] T. Maeno, Panda: distributed production and distributed analysis system for atlas, in: Journal of Physics: Conference Series, Vol. 119, IOP Publishing, 2008, p. 062036.

[25] S. Maddineni, J. Kim, Y. El-Khamra, S. Jha, Distributed application runtime environment (dare): a standards-based middleware framework for science-gateways, Journal of grid computing 10 (4) (2012) 647–664.

[26] A. Redolfi, R. McClatchey, A. Anjum, A. Zijdenbos, D. Manset, F. Barkhof, C. Spenger, Y. Legré, L.-O. Wahlund, C. B. di San Pietro, et al., Grid infrastructures for computational neuroscience: the neugrid example, Future Neurology 4 (6) (2009) 703–722.

[27] R. Dooley, M. Vaughn, D. Stanzione, S. Terry, E. Skidmore, Software-as-a-service: the iplant foundation api, in: 5th IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS), 2012.

[28] J. Dakka, K. Farkas-Pall, M. Turilli, D. W. Wright, P. V. Coveney, S. Jha, Concurrent and adaptive extreme scale binding free energy calculations, in: 2018 IEEE 14th International Conference on e-Science (e-Science), IEEE, 2018, pp. 189–200.

[29] M. Turilli, Y. N. Babuji, A. Merzky, M. T. Ha, M. Wilde, D. S. Katz, S. Jha, Evaluating distributed execution of workloads, in: 2017 IEEE 13th International Conference on e-Science (e-Science), IEEE, 2017, pp. 276–285.

[30] M. Turilli, F. Liu, Z. Zhang, A. Merzky, M. Wilde, J. Weissman, D. S. Katz, S. Jha, Integrating abstractions to enhance the execution of distributed applications, in: 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE, 2016, pp. 953–962.

Code Samples

RADICAL-Pilot (RP) executes set of tasks. The degree of concurrency of the execution depends on the amount of available resources. Consider for example Ref. [14]'s many-task application for the simulation of molecular dynamics with an ensemble of 128 GROMACS simulations, each requiring 24 CPU cores. The user can use RP API to describe a pilot job with 3072 cores (Lis. 1), 128 CUs (Lis. 2) and two managers to coordinate the acquisition of the pilot resources via RADICAL-SAGA on an HPC machine and the concurrent execution of the 128 tasks on those resources (Lis. 3).

```
pdesc = rp.ComputePilotDescription()
pdesc.resource = target
pdesc.cores = 3072
pdesc.runtime = 120
pdesc.project = 'abc'
pdesc.queue = 'xyz'
pdesc.access_schema = 'ssh'
```

Listing 1: RADICAL-Pilot API: define a 3072-core pilot that runs for 120 minutes on resource 'target' accessed via 'ssh'

```
n = 128    # number of units to run
cuds = list()
for i in range(0, n):
    cud = rp.ComputeUnitDescription()
    cud.executable = 'gmx_mpi'
    cud.pre_exec = ['module load gromacs/5.1.2']
    cud.arguments = ['mdrun', '-s', 'min.tpr', '-v', '-deffnm',
        'npt']
    cud.input_staging = ['FRF.itp', 'dynamic.mdp', 'FF.itp',
        'martini_v2.2.itp', '85-20.top', 'init85-20.gro']
    cud.cores = 24
    cud.mpi = True
    cuds.append(cud)
```

Listing 2: RADICAL-Pilot API: define 128 24-cores MPI CU descriptions.

```
pmgr = rp.PilotManager(session=session)
pilot = pmgr.submit_pilots(pdesc)
umgr = rp.UnitManager(session=session)
umgr.add_pilots(pilot)
umgr.submit_units(cuds)
umgr.wait_units()
```

Listing 3: RADICAL-Pilot API: create pilot and unit managers, submit units, and wait for their completion.

RP API cannot express dependences among tasks. For RP, every task that is passed to UnitManager is assumed to be ready for execution. Users can explicitly code priorities among stages in the applications they write with the RP API but they have no dedicated abstractions in that API for expressing those priorities. EnTK offers these abstractions at API level, enabling users to create pipelines as sequence of stages, each with a set of tasks. For example, the 128 GROMACS simulations of the previous example can become a stage of a pipeline followed by another stage performing the analysis of the simulations' result. Users instantiate an AppManager (Lis. 4), define a set of resources on which to run their workflow (Lis. 5), describe that workflow in terms of pipelines, stages and tasks (Lis. 6) and execute it (Lis. 7).

```
appman = AppManager(hostname=hostname, port=port)
```

Listing 4: RADICAL-EnsembleToolkit (EnTK) API: Create AppManager.

```
res_dict = {
    'resource': 'target',
    'walltime': 120,
    'cpus': 3072
}
```

Listing 5: RADICAL-EnsembleToolkit (EnTK) API: Describe a resource request.

```
p = Pipeline()
s = Stage()
n = 128
for i in range(0, n):
    t = Task()
    t.executable = "gmx_mpi"
    t.pre_exec = ['module load gromacs/5.1.2']
    t.arguments = ['mdrun', '-s', 'min.tpr', '-v', '-deffnm',
        'npt']
    t.copy_input_data = ['FRF.itp', 'dynamic.mdp', 'FF.itp',
        'martini_v2.2.itp', '85-20.top', 'init85-20.gro']
    t.cpu_reqs = {
      'processes': 24,
      'process_type': MPI
      }
    s.add_tasks(t)
p.add_stages(s)
```

Listing 6: RADICAL-EnsembleToolkit (EnTK) API: Describe a pipeline with 1 stage with 128 24-cores MPI tasks.

```
appman.resource_desc = res_dict
appman.workflow = set([p])
appman.run()
```

Listing 7: RADICAL-EnsembleToolkit (EnTK) API: Describe execution of a pipeline on specified target resource

Current code version

Nr.	Code metadata description	Please fill in this column
C1	Current code version	0.60
C2	Permanent link to code/repository used for this code version	`https://github.com/radical-cybertools`
C3	Legal Code License	MIT License (MIT)
C4	Code versioning system used	git
C5	Software code languages, tools, and services used	python, shell, C
C6	Compilation requirements, operating environments & dependencies	virtualenv, pip or conda
C7	Developer documentation/manual	`https://radicalpilot.readthedocs.io/`
C8	Support email for questions	`radical-cybertools@googlegroups.com`

Table .1: Code metadata

Current executable software version

Nr.	Exec. metadata description	Please fill in this column
S1	Current software version	0.60
S2	Permanent link to executables of this version	`https://pypi.org/project/radical.pilot/`
S3	Legal Software License	MIT License (MIT)
S4	Computing platforms/Operating Systems	GNU/Linux operating systems
S5	Installation requirements & dependencies	virtualenv, pip or conda
S6	User manual and publications	User manual: `https://radicalpilot.readthedocs.io/`; Publications: Refs [2, 5, 3]
S7	Support email for questions	`radical-cybertools@googlegroups.com`

Table .2: Software metadata