A Container-Based Framework to Facilitate Reproducibility in Employing Stochastic Process Algebra for Modeling Parallel Computing Systems

William S. Sanders
Computer Science & Engineering
Mississippi State University
Mississippi State, MS USA
wss2@msstate.edu

Srishti Srivastava
Computer Science
University of Southern Indiana
Evansville, IN USA
fsrishti@usi.edu

Ioana Banicescu
Computer Science & Engineering
Mississippi State University
Mississippi State, MS USA
ioana@cse.msstate.edu

Abstract—**Scientific applications are increasingly complex and domain specific, and the underlying architectures of the parallel and distributed systems on which they are executed also continue to grow in complexity. As these high performance parallel and distributed computing applications and environments continue to grow both in complexity and computing power, there is an increasing financial cost associated with both the acquisition and maintenance of those systems. Therefore, the ability to model the performance of these applications and systems before and during their development and deployment to guide cost-effective decisions about their resources and configurations is highly important to the designers of those applications and systems. Performance Evaluation Process Algebra (PEPA) is a modeling language and framework for modeling parallel and distributed computing and communication applications and systems, and numerous examples are present in the literature where PEPA has been utilized to model these systems for evaluating or predicting their performance using various metrics, including throughput, utilization, and robustness. Since its development, the PEPA modeling framework has been expanded to model biological systems and networks (Bio-PEPA), and massive (on the order of $\sim 10^{129}$ components) homogeneous systems with Grouped PEPA (GPEPA). PEPA and its derivatives are implemented in a variety of ways, ranging from plug-ins integrated with the Eclipse integrated development environment to standalone command-line based interpreters, each with their own unique and often challenging installation and configuration requirements. To help enable other researchers to more easily utilize these frameworks and facilitate increased and robust reproducibility across end-user platforms, we present and make available containerized versions of a number of these PEPA frameworks. We have validated the functionality of these containers by testing them with models available from the research community that utilizes PEPA. These containers serve as a readily available resource for the community and can be executed on any environment capable of executing the underlying containerization framework.**

Index Terms—**Application virtualization; Reproducibility of results; Performance modeling; Performance evaluation; Robustness analysis; Parallel computing; Process algebra; Scalability**

I. Introduction

The growing and evolving size and complexity of scientific applications and parallel and distributed computing environments are often associated with a related growth in the expense necessary to develop, deploy, and operate these applications and systems. This complexity and expense gives rise to the need to model and test these application and architecture designs in a cost-effective manner, to determine if a given system will meet or exceed certain performance metrics and criteria early in the design and development of such systems. Performance Evaluation Process Algebra (PEPA) [20] has arisen as one of a number of process algebras / process calculi to model these applications and systems, and has gained in popularity as a formalism to model these dynamic environments, as it offers significant advantages over previous modeling formalisms [2], [3]. PEPA has been used to model, evaluate, and predict the performance of a number of complex environments including the robustness of resource allocations in parallel and distributed computing systems [2], [5]. Since its release, PEPA has continued to be updated and adapted, and derivatives have spawned for specific use cases. Bio-PEPA was developed [10], [11] as an extension to the PEPA framework to model biological systems, while Grouped PEPA (GPEPA) was developed [18], [19] to overcome limitations in the number of states modeled via PEPA for homogeneous systems.

The PEPA formalism and its derivatives are made available as software implemented in application environments and these modeling frameworks have their own configurations and dependencies that can make them both challenging for new users to configure and overcome. This can result in a significant barrier to entry to use these formalisms for researchers that want to create their own models or validate existing system designs. Both PEPA and Bio-PEPA are implemented as plug-ins for the Eclipse Integrated Development Environment (IDE), a programming framework implemented in Java, and require a number of very specific dependencies to function, including specific versions of the Java Development Toolkit, specific versions of the Eclipse IDE, and various specific software components and add-ons within the Eclipse IDE [31], [12]. GPEPA is implemented through the standalone GPAnalyser tool, which also requires very specific Java and visualization package dependencies [30]. As these implementations of PEPA, Bio-PEPA and GPEPA are often not maintained and updated on the same cycle as the underlying systems on which individual researchers utilize them, the various configuration inconsistencies can result in these applications being both difficult to configure and pose problems when trying to replicate results on slightly different or updated platforms.

Reproducibility, the capability to replicate published research results, represents one of the fundamental aspects of the scientific method. Recently, there have been several high profile instances of research results failing to be adequately reproduced and replicated across a variety of domains, and scientific publishing in general has been said to be in the midst of a reproducibility crisis [28]. This crisis is amplified by the current ease of data generation and collection, and can be compounded by failure to publish precise details of the custom scripts and application environment configurations used to assist with data capture, processing, and analysis [28]. Concurrently with this reproducibility crisis, technology continues to evolve, and parallel and distributed computing applications, systems, and architectures continue to grow increasingly more complex and expensive.

Containerization frameworks have attracted significant attention lately, as they facilitate the software development practice of DevOps, where continuous software development (Dev) is combined with integrated information technology operations (Ops). To facilitate DevOps, the development of microservices is prioritized. Microservices are computationally lightweight, virtualized environments generally utilized to run a single application in a portable framework, independent of the underlying physical architecture [24], [25]. These microservices are generally implemented as containerization frameworks (lightweight virtual machines (VMs)) where the application runtime environment is separated from the underlying operating system resources on the execution host. This is different than the execution of traditional VMs, where the guest operating system and applications are executed with both the hardware and operating system of the underlying system abstracted [25]. One such containerization framework, Docker, has gained significant utilization as the de facto containerization framework for cloud service providers and cloud computing ecosystems, but concern about potential security flaws have hindered its uptake for on-premise data centers and high performance computing facilities serving multiple tenants [24]. To address these concerns, the Singularity containerization framework was developed, verified to be compatible with Docker, and has significant traction for on-premise, non-cloud container deployments [24], [25].

In this paper, we present containerized modeling frameworks for PEPA and some of its derivatives as Singularity containers, freely available online to the scientific community. We provide background on PEPA and its derivatives, discuss the need for reproducibility and various challenges that can be encountered when replicating previous work, and describe containerization frameworks. We then create containers for PEPA and its derivatives and validate those containers against previously reported models. Both the instructions to build these containers (the build recipes) and the containers themselves are available for download through GitHub and Singularity-Hub, respectively. These containers help facilitate scientific reproducibility and lower the barrier to utilizing PEPA and its derivatives for modeling increasingly complex parallel and distributed applications and systems. To the best of our knowledge, this work represents the first efforts to make a shared container resource available to the process calculi / process algebra communities.

II. Background and Related Work

A. Performance Evaluation Process Algebra (PEPA)

Performance Evaluation Process Algebra (PEPA) was developed in the early 1990s by Jane Hillston of the University of Edinburgh to evaluate systems best described through the use of dynamic properties, for example, modeling the number of resources available to fulfill requests in a queue-based scheduling system, or modeling the runtime performance on a cluster of parallel computing resources [3], [20]. The PEPA formalism is part of a larger group of process algebras or process calculi, which include the Calculus of Communicating Systems (CCS) [26] and Communicating Sequential Processes (CSP) [23]. Process calculi offer significant improvement in formal framework for predictive modeling over the previous best methods of stochastic Petri Nets (SPNs) [20] and queuing networks [20].

Process calculi became adopted over these previous methods because the formalism of process calculi offer the benefits of compositionality, quantitative analysis, and wide acceptance [21]. Having the property of compositionality allows researchers to break highly complex systems into their component parts and construct the models using the various smaller constructed components [21]. With the ability to perform qualitative analysis checks on systems being modeled, verification that the modelled system is performing correctly and responds to queries in a reasonable time is obtained [21]. Wide acceptance of process calculi for modeling allows the constructed models to be incorporated early in the design process (instead of late in the design process or neglected all together), allowing systems to be designed for scalability and stable performance [21].

The standard implementation of PEPA utilizes stochastic Markov processes to model the changes in state of the random variables that represent the evolution of dynamic systems. Markov processes are probabilistic in nature and model the probability of a component being in its current state based on input from its possible previous states [20], [21]. In PEPA, this implementation serves as an extension of classical process calculi through the association of the Markov process with the random variables of a model, primarily a time step, which are associated with the states of the model [20], [21]. Each component of a PEPA model can perform a set of actions. An action in PEPA is characterized by two components - the type of action being performed and the duration of the action as modeled by the negative exponential expression governed by the Continuous Time Markov Chain (CTMC) which helps to model the likelihood and probability of the model being in a given state [21]. PEPA's implementation, like other process calculi, defines five structured operators or operational semantics typical to process calculi (listed in order of increasing complexity): Prefix, Choice, Cooperation, Hiding, and Constant [21].

Continuous time Markov chains (CTMC) were chosen instead of discrete time Markov chains (DTMC) for describing stochastic system evolution in PEPA and Bio-PEPA because a continuous time (CT) representation more accurately addresses modeling parallel and distributed systems with events that are both countably finite and that occur at non-specific time intervals than does a discrete time (CT) representation [3]. This is because events in models utilizing a CTMC representation are evaluated as each event occurs, while those events in models with a DTMC representation are evaluated at predefined or discrete time intervals [3]. DTMCs do not support modeling of systems with concurrent behavior, while CTMCs allow a more accurate time representation for concurrent systems [22]. By evaluating a CT model at the occurrence of each event, one can more accurately model systems with many parallel agents acting independently [3].

As Markov chains, including continuous time Markov chains, grow increasingly larger, there can be an corresponding increase in the number of finite states which must be evaluated, and this leads to the *"state space explosion"* problem, where an increasing amount of states must be evaluated in order to generate a solution [20]. This explosion in the number of states that have to be explored to generate a solution leads to an increase in the time and memory required for that solution, making standard, brute-forced based, solutions generally unfeasible and limiting the scalability of the algorithm used [27].

PEPA was initially used to model the performance of grid-based scheduling algorithms that were utilized to process data from Britain's National Weather Service (NWS) [6]. This NWS data processing was modeled and tested using PEPA by decomposing the processing pipeline into "skeleton" algorithm components and "grid" resource components, which were then used to model the time to process data based on the algorithmic performance on the available resources [6]. After the effectiveness of this performance modeling was shown for the NWS data, the framework was extended to a more generic framework for modeling algorithmic performance on sets of possibly limited resources [7], and then was further extended to include the incorporation of online resources in the "grid" component of the PEPA modeling framework [8]. Additional work extended the PEPA framework to include the incorporation of components that are aware of their own state, allowing the construction of component specific feedback into the constructed model [14]. A recent study demonstrated effectiveness of PEPA framework to predict the robustness of scheduling irregular applications in parallel and distributed environments [2], [5].

In 2008, PEPA was extended to create Bio-PEPA by Federica Ciocchetta and PEPA's creator Jane Hillston to allow the formal modeling of biochemical networks [10], [11]. The components of biochemical networks can be mapped back to the components of an algorithmic pipeline (the processes and directionality of substrate and product formation versus the inputs and outputs of an algorithmic pipeline) and limited computational resource availability (enzyme and inhibitor concentrations versus the availability of a given computational resource (processing, storage, memory, etc.)) [10], [11]. This gives the modeling of biochemical networks with Bio-PEPA a direct correlation to performance modeling and task scheduling in high performance computing environments. The Bio-PEPA extensions of PEPA allow for the modeling of chemical stoichiometry and the incorporation of general enzyme kinetics laws [10]. Additionally, at the same time automatic mapping procedures were developed to allow Bio-PEPA models to be converted into the syntax of Systems Biology Markup Language, a common framework that significant portions of the biological research community use to describe biochemical reactions in structured manner [16].

The PEPA/Bio-PEPA frameworks were further expanded through extensions independent of the initial PEPA development efforts to attempt to increase the size and scope of systems being modeled and to overcome the limitations imposed when using CMTCs. Grouped PEPA (GPEPA) [18], [19] and Process Algebra with Layers (PAL) [29] were both developed to overcome limitations identified in the existing PEPA/Bio-PEPA implementations. GPEPA disposes of the underlying CTMCs and replaces them with differential equations, allowing for models with $\sim 10^{129}$ states to be evaluated [18], [19]. The GPEPA execution framework GPAnalyser [30] is separate and standalone from the existing PEPA and Bio-PEPA plugins for Eclipse. PAL allows for the modeling of more complex models with multiple scales (or layers) within a single model, but both PAL and Bio-PEPA are typically executed within the Bio-PEPA plugin for Eclipse, which uses continuous time Markov chains (CTMCs) for modeling the stochastic processes of the systems being modeled, and the Bio-PEPA plugin is localized to a single, non-distributed computer system. The use of CTMCs as the basis for modeling the stochastic processes limits models that can be evaluated to $\sim 10^{11}$ states.

B. *Reproducibility*

There is rising concern about a lack of repeatability, replicability, and reproducibility in science and engineering [1] with a large number of public failings across domains ranging from algorithm development and design to cancer genomics, medicine and economics, where the results of a number of high profile publications failing to be replicated when reproducibility studies were conducted [28]. Variation in data collection methodologies, experimental environments, computational configuration, the lack of detailed and intricate documentation, and more are leading to a call for enhanced peer review and validation of experimentally produced artifacts [1], [28]. These experimentally produced artifacts represent the digital artifacts produced during the course of research that are either the inputs or outputs of the study, and can take the form of input data sets, raw data, software systems, scripts to run experiments or analyze results [1].

As part of this process to provide better validation of experimental works, a number of publishers are engaged in efforts to encourage and promote better scientific reproducibility [1], [28]. The Association of Computing Machinery has begun and embraced a campaign to produce digital badges representing various levels of digital artifact validation to accompany publications [1]. These badges indicate if the digital artifacts have been found to be functional, reusable, available, and if the underlying studies have validated results and reproduced results [1].

C. Containers

In recent years, containers and containerization technologies have emerged and gained significant adoption and importance among developers within industry and within the software development landscape [25]. Applications distributed as containers have the same installation overhead as a normal installation of the container, but with a slight additional overhead requiring knowledge of the containerization framework. However, once the applications are containerized and the containers are made available for others, the full burden of application installation and configuration is abstracted from the end users of those containers, so long as they can utilize the underlying containerization framework on their systems [34]. This allows users to download and run these pre-built containers without the need for often lengthy and detailed application installation procedures. Containers operate as a virtualization layer within the operating system of the host machine, and provide a separation of the containerized application environment from the physical resources of the hosts. Previous work has shown that applications bundled and distributed as containers and executed through containerization frameworks have very little execution overhead and exhibit almost no difference in application performance than the same application executing non-containerized on a bare-metal system [32] [33]. This separation and abstraction allows containers to be easily migrated from system to system as long as the underlying containerization framework is capable of being executed and has given significant traction to their adoption, particularly in cloud computing ecosystems. They typically include all the software and dependencies to run a single application, allowing them to facilitate DevOps principles and support for microservices (an environment dedicated to a single task) [24], [25].

The microservice development strategy, encapsulating all the necessary dependencies for a single application into a lightweight virtualization environment makes containers a natural fit for the software environments, processing scripts, and applications produced as digital artifacts in the course of scientific and academic research. A large portion of the artifacts generated by research groups are produced by graduate students and postdoctoral associates, are developed in unique or isolated environments, would not be considered production-level in terms of support and documentation, and often, after publication or when the student or postdoctoral associate moves on to other positions, are at best poorly updated and maintained. This leads to situations where researchers working to reproduce or build on this previous work are forced to engage in almost archaeological expedition level efforts with the identification and configuration of various undocumented application dependencies [13].

Docker has emerged as the clear leader in containerization frameworks, particularly for cloud computing ecosystems, but information security concerns about possible root privilege escalation exploitations have slowed its adoption within on premise data centers and academic high performance computing centers. To alleviate these potential security concerns, an alternative containerization framework, Singularity, was developed at Lawrence Berkeley National Laboratory, and has begun to make significant inroads as the a leader in containerization frameworks for non-cloud deployments [13], [25]. The Singularity containerization framework differs from the Docker containerization framework in that by design it does not provide a mechanism to allow for privilege escalation within the runtime environment, so a user within the container is the same user as outside the container [34]. This makes it an ideal candidate for execution on large multi-tenant HPC systems, such as those found in academic and research institutions.

III. Containerization for Reproducibility

To facilitate uptake of PEPA and its derivatives and to lower the barriers to entry for researchers while facilitating scientific reproducibility, Singularity containers were developed for PEPA, Bio-PEPA, and GPAnalyser. We chose Singularity over other containerization frameworks as Singularity has been developed to alleviate information security concerns of unauthorized privilege escalation and as such is becoming more widely available at national high performance computing facilities and on-premise data centers [13], [25]. Containers were constructed by constructing container build recipes for each application and all its dependencies through a process of following each application′s installation guidelines and often times, in the case of lack of detail about underlying dependencies, identifying the most likely application packages and versions used at the time of the application′s last revision by its maintainers. Each container was initially constructed using Singularity v2.5.2 running on CentOS Linux v7.4 deployed HP Proliant SL series servers with 2.5 GHz 20-core 64-bit Intel processors with 256GB RAM.

After initial construction, each container was verified to provide correct operation by comparing its functionality and output against non-containerized versions of each application using example code on other operating system platforms (CentOS Linux v7.6, Ubuntu LTS v18.04 *"Bionic Beaver"*, Ubuntu v16.04 LTS *"Xenial Xerus"*, LinuxMint v19.1 *"Tessa"*, Debian v9.6 LTS *"Stretch"*,) running Singularity v2.5.2. To confirm compatibility with cloud computing resources, we also verified our containers on a `n1-standard-8` machine instance (8 virtual CPUs and 30GB RAM) running CentOS

TABLE I
MAPPING A AND MAPPING B OF APPLICATIONS (a_i) TO MACHINES (M_j) FROM [5]

Machine	Mapping A	Mapping B
M_1	$a_5, a_9, a_{12}, a_{17}, a_{20}$	$a_3, a_4, a_5, a_{17}, a_{18}, a_{20}$
M_2	a_6, a_{16}	$a_2, a_{11}, a_{14}, a_{19}$
M_3	a_1, a_3, a_7	a_1, a_7, a_{13}
M_4	$a_2, a_4, a_{10}, a_{13}, a_{15}, a_{19}$	a_9, a_{12}, a_{15}
M_5	$a_8, a_{11}, a_{14}, a_{18}$	a_6, a_8, a_{10}, a_{16}

v7.6 and Singularity v2.5.2 on the Google Cloud Platform. To validate the functionality of our container for The PEPA Eclipse Plug-in, a number of example models (including The PEPA Active Badge Model [17], The Alternating Bit Protocol Model [15], and the PC LAN 4 Model) were downloaded from http://www.dcs.ed.ac.uk/pepa/examples/ and tested both with and without container functionality. A small PEPA model successfully running in our container environment is shown in Fig. 1.

The interactive IDE nature of PEPA makes detailed performance comparison of the containerized versions of these applications and their non-containerized versions more challenging than standard command-line interface applications, but previous work has shown very minimal impact of containerization on application performance [32] [33]. Our experiences with these containers are consistent with our experience using the native, non-containerized applications, but it is worth noting that both the containerized and non-containerized versions of these applications, when executed on a remote system through a terminal (for example, with X-forwarding enabled) are subject to the standard issues around network latency when utilizing GUI-based applications over a network.

To fully validate that our containers provide the same functionality as the non-containerized versions of PEPA and its derivatives, we chose to replicate portions of previously published work utilizing PEPA and The PEPA Eclipse plug-in to model a mapping of applications to machines to determine the robustness of resource allocations in an environment with variable processor availability [5]. This research is based on a simulation study of robustness of static resource allocations in an environment where 20 parallel applications are mapped to a cluster of 5 parallel and heterogeneous machines for two distinct cases (Mapping A and Mapping B) [5]. This mapping of applications to machines is shown in Table I.

Fig. 2 is a replication the activity diagram for M_3 from [5], generated using our PEPA Singularity container. This provides verification of functionality of the PEPA Eclipse plug-in, but to highlight that even the numerical analysis provided PEPA is consistent, we also replicated the cumulative distribution functions for M_1 under both cases in that study (Mapping A and Mapping B) and this is shown in Fig. 3 and 4 respectively. The results from our containerized application are consistent with the results presented in that study, and the same cumulmative distribution function was obtained when using the containerized PEPA Eclipse plug-in.

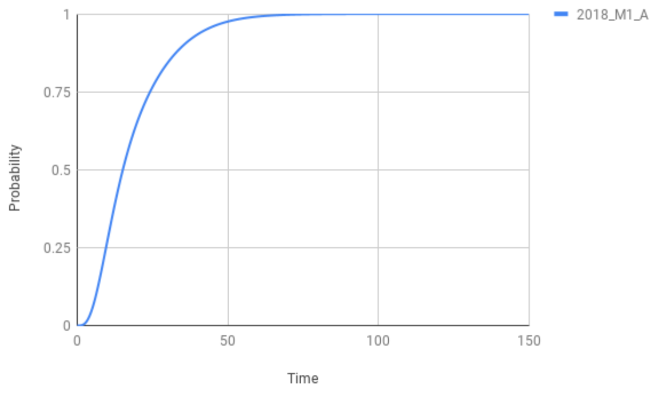


Fig. 3. Results from the containerized replication of the cumulative distribution function (CDF) of the finishing time of Machine M_1 for executing applications $a_5, a_9, a_{12}, a_{17}, a_{20}$ as given by Mapping A from [5]

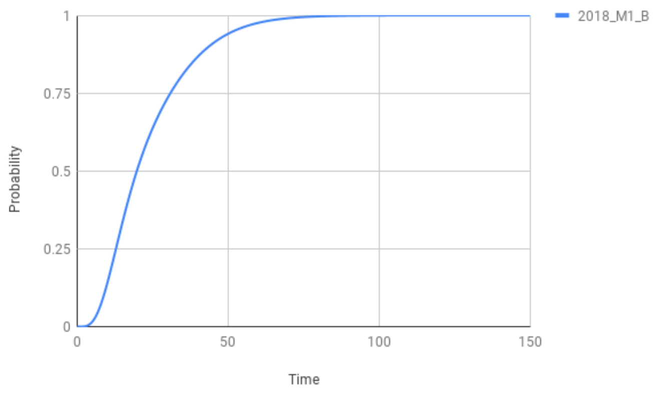
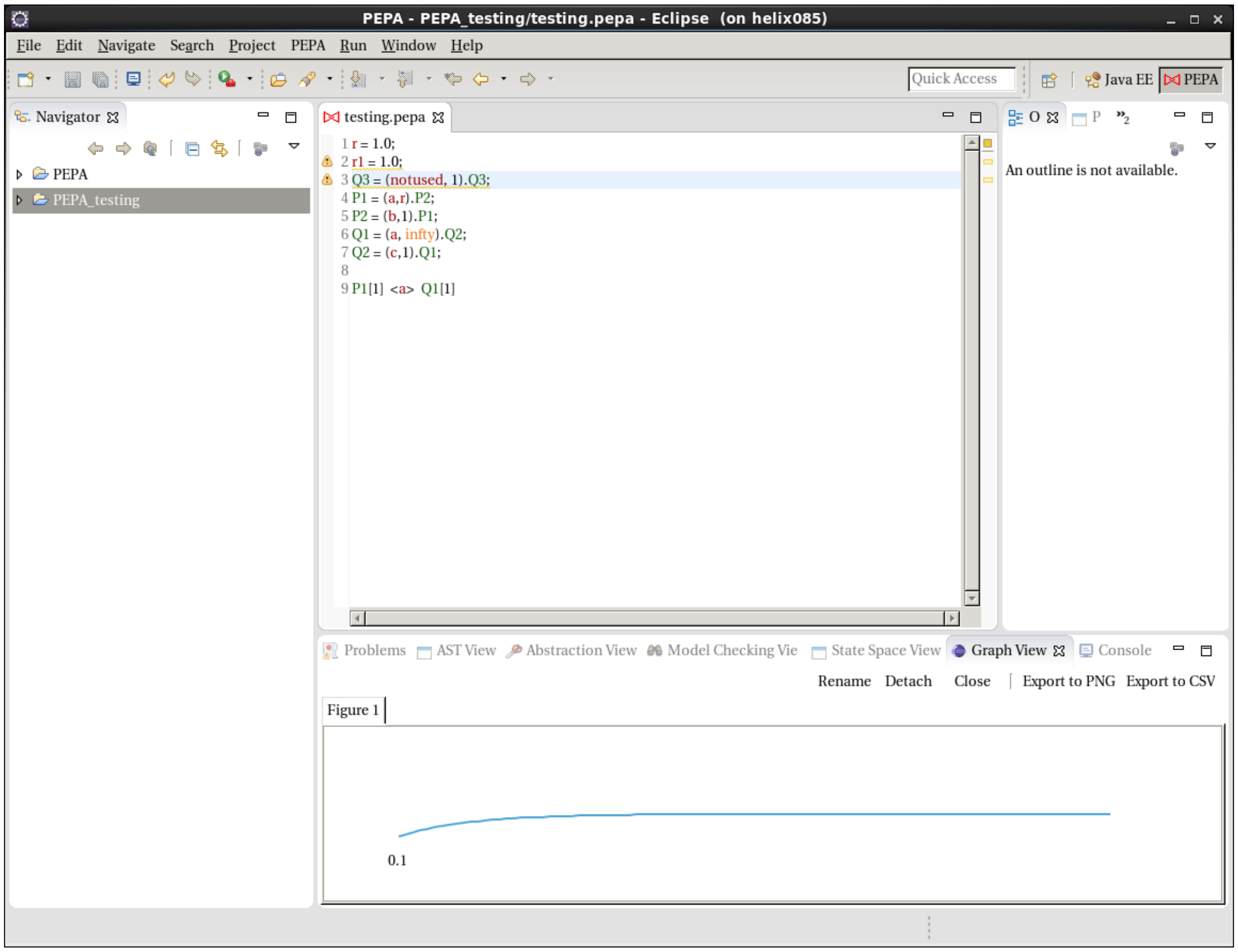


Fig. 4. Results from the containerized replication of the cumulative distribution function (CDF) of the finishing time of Machine M_1 for executing applications $a_3, a_4, a_5, a_{17}, a_{18}, a_{20}$ as given by Mapping B from [5]

For testing our Bio-PEPA container, example models from the Bio-PEPA Users Manual [9] showing Bio-PEPAs ability to model basic biochemical enzyme kinetics systems converting various molecular products to substrates with and without the presence of reaction inhibitors were used to validate correct implementation. The GPAnalyser container was also validated by confirming identical output of the container compared to the non-containerized implementation by using all the GPEPA example model files (these include models of homogenous systems for client-server request scalability and client-server power consumption) available at https://code.google.com/archive/p/gpanalyser/source/models/source. Fig. 5 shows the `clientServerScalability.gpepa` GPEPA model, provided as an example with the GPAnalyser tool, executing as expected with our Singularity container. This system models the scalability of a varying number of client systems making requests to a variable number of servers, where the servers are rewarded for satisfying requests within a given time period.

The build recipes for all Singularity containers described in



```
r = 1.0;
r1 = 1.0;
Q3 = (notused, 1).Q3;
P1 = (a,r).P2;
P2 = (b,1).P1;
Q1 = (a, infty).Q2;
Q2 = (c,1).Q1;

P1[1] <a> Q1[1]
```

An outline is not available.

Fig. 1. Validation of The PEPA Eclipse Plug-in Singularity Container. Application output for a simple PEPA model was verified by comparing against non-containerized version of the application.

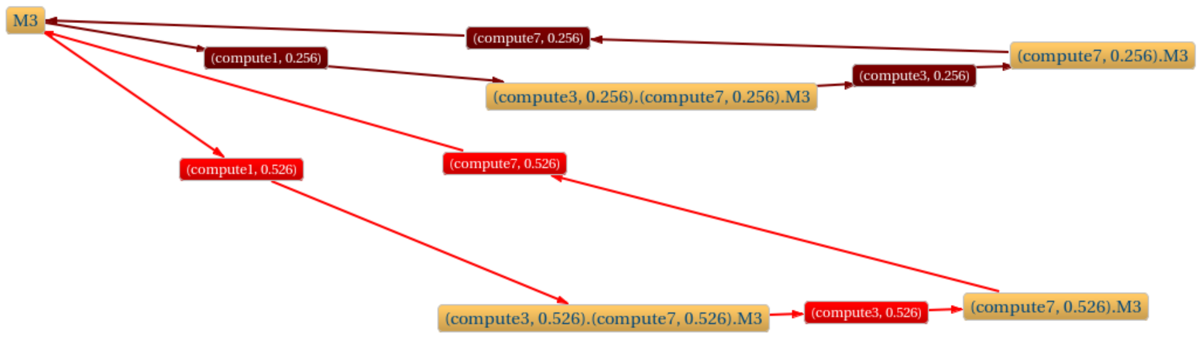
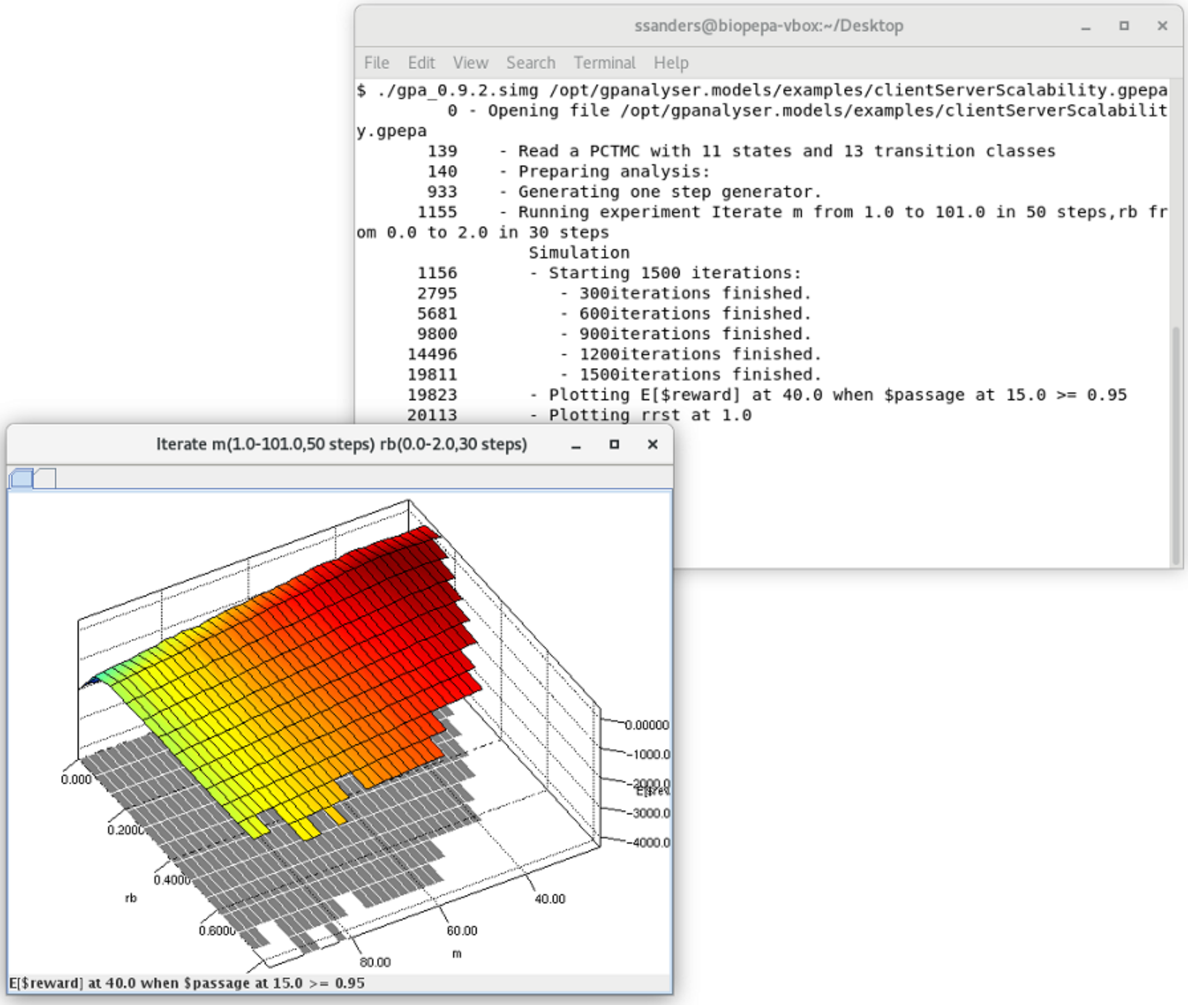


Fig. 2. Containerized replication of the activity diagram generated by the PEPA workbench for machine (M_3) component of the PEPA model for Mapping A from [5].

this manuscript are available on GitHub at https://github.com/williamssanders/pepa and are also pre-compiled and available for download at https://www.singularity-hub.org/collections/2351 (shown in Fig. 6).

IV. Conclusions and Future Work

In this work, we have described and made available a resource to the community to facilitate increased ease-of-use of PEPA and some of its derivatives by developing and providing


ssanders@biopepa-vbox:~/Desktop
File Edit View Search Terminal Help
Iterate m(1.0-101.0,50 steps) rb(0.0-2.0,30 steps)
0.00000
-1000.0
-2000.0
E[$re
-3000.0
-4000.0
0.000
0.2000
0.4000
rb
0.6000
40.00
60.00
80.00
m

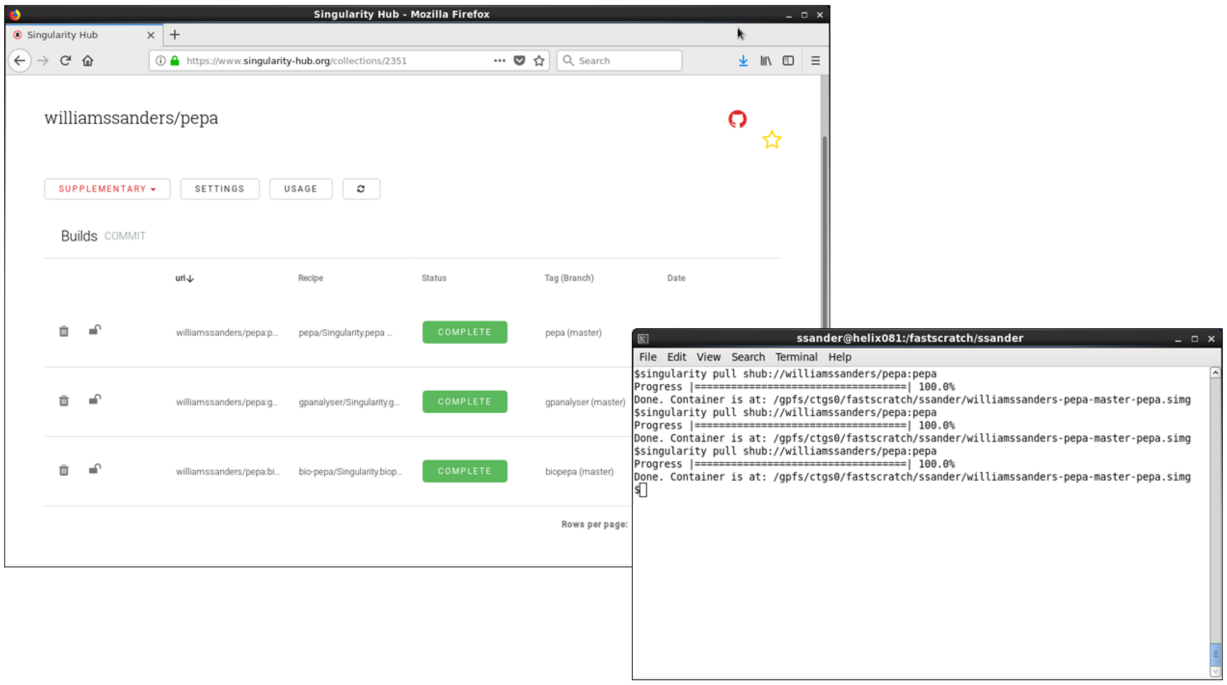
```
$ ./gpa_0.9.2.simg /opt/gpanalyser.models/examples/clientServerScalability.gpepa
         0 - Opening file /opt/gpanalyser.models/examples/clientServerScalabilit
y.gpepa
       139    - Read a PCTMC with 11 states and 13 transition classes
       140    - Preparing analysis:
       933    - Generating one step generator.
      1155    - Running experiment Iterate m from 1.0 to 101.0 in 50 steps,rb fr
om 0.0 to 2.0 in 30 steps
                 Simulation
      1156       - Starting 1500 iterations:
      2795          - 300iterations finished.
      5681          - 600iterations finished.
      9800          - 900iterations finished.
     14496          - 1200iterations finished.
     19811          - 1500iterations finished.
     19823       - Plotting E[$reward] at 40.0 when $passage at 15.0 >= 0.95
     20113       - Plotting rrst at 1.0
```

```
E[$reward] at 40.0 when $passage at 15.0 >= 0.95
```

Fig. 5. Validation of GPAnalyser Singularity Container by Execution of the `clientServerScalability.gpepa` GPEPA example file. Application output was verified by comparing against non-containerized version of the application.

containerized applications based on the Singularity containerization framework. These containers aim to help overcome some of the many challenges that can arise when working with the digital artifacts of scientific research. Novel and promising applications are often developed during the course of graduate and postdoctoral research, but those graduate students and postdoctoral associates transition to other roles and institutions in the course of their careers, often leaving the applications and other digital artifacts they have generated unsupported, unmaintained, and often not updated. Other researchers seeking to build and expand upon that previous work can struggle to replicate and reproduce that work, particularly as it ages and the underlying systems and dependencies also age, reach the end of their supported lifetimes, and new updated versions are released. Researchers can find themselves engaging in work akin to an archaeological dig, pouring through equally aged documentation trying to determine which versions of which dependencies were prevalent at the time of the initial application publication. Even those applications and digital artifacts that are well documented and supported by their developers can cause difficulties with the replication and reproducibility of previous work built utilizing them, if care is not taken to verify and document backwards computability.

Containerization of these applications reduces the level of effort required to build and maintain working environments for the development and analysis of PEPA models, as the containerized versions of these applications can be utilized on any platform capable of utilizing the underlying container framework. These containers and their build recipes can be stored independently of the application and its source code and they can be version controlled to facilitate reproducibility and replication of past results. These containers provide increased ease-of-use to the end user, minimizing the need for users (and system administrators) to engage in detailed and involved application installations, and have an advantage that the container will consistently produce reproducible results across platforms.


Singularity Hub - Mozilla Firefox
Singularity Hub
https://www.singularity-hub.org/collections/2351
Search
SUPPLEMENTARY
SETTINGS
USAGE
ssander@helix081:/fastscratch/ssander
File Edit View Search Terminal Help
Rows per page:

williamssanders/pepa

Builds COMMIT

	uri	Recipe	Status	Tag (Branch)	Date
	williamssanders/pepa:p...	pepa/Singularity.pepa ...	COMPLETE	pepa (master)	
	williamssanders/pepa:g...	gpanalyser/Singularity.g...	COMPLETE	gpanalyser (master)	
	williamssanders/pepa:bi...	bio-pepa/Singularity.biop...	COMPLETE	biopepa (master)	

```
$singularity pull shub://williamssanders/pepa:pepa
Progress |===================================| 100.0%
Done. Container is at: /gpfs/ctgs0/fastscratch/ssander/williamssanders-pepa-master-pepa.simg
$singularity pull shub://williamssanders/pepa:pepa
Progress |===================================| 100.0%
Done. Container is at: /gpfs/ctgs0/fastscratch/ssander/williamssanders-pepa-master-pepa.simg
$singularity pull shub://williamssanders/pepa:pepa
Progress |===================================| 100.0%
Done. Container is at: /gpfs/ctgs0/fastscratch/ssander/williamssanders-pepa-master-pepa.simg
$
```

Fig. 6. Container Resources for PEPA, Bio-PEPA, and GPAnalyser made publicly available for download from Singularity-Hub.org. The Singularity-Hub collection of PEPA containers, along with successful clone of each container from the collection are shown.

The containerized applications function identically to their non-containerized versions, making them ideal for distribution of applications and code as part of the peer-review process.

Future work will include the identification and containerization of other PEPA derivatives and of other process calculi modeling tools to further enhance this online resource. In addition, these containers will be used to expand on previous work and model resource allocations in parallel computing systems and obtain an analysis of the robustness of the resource allocations in those systems as they are subjected to unpredictable variations in application and systemic characteristics.

ACKNOWLEDGMENT

The authors would like to acknowledge The Jackson Laboratory (JAX) for providing support and helping to make this work possible, and David P. McKenzie at JAX for his assistance with the identification of GPEPA dependencies. Additionally, the authors would like to acknowledge the partial support of the NSF #1034897 grant.

REFERENCES

[1] Association of Computing Machinery. Artifact review and badging. Technical report, Association of Computing Machinery, April 2018.

[2] I. Banicescu and S. Srivastava. Towards robust resource allocations via performance modeling with stochastic process algebra. In *Proc. IEEE 18th Int. Conf. Computational Science and Engineering*, pages 270–277, October 2015.

[3] Srishti Srivastava. *Evaluating the robustness of resource allocations obtained through performance modeling with stochastic process algebra*. PhD thesis, Mississippi State University, 2015.

[4] Srishti Srivastava and Ioana Banicescu. Robust resource allocations through performance modeling with stochastic process algebra. *Concurrency and Computation: Practice and Experience*, 29(7),e3894, 2017.

[5] Srishti Srivastava and Ioana Banicescu. PEPA based performance modeling for robust resource allocations amid varying processor availability. *17th International Symposium on Parallel and Distributed Computing (ISPDC)*, 2018.

[6] A. Benoit, M. Cole, S. Gilmore, and J. Hillston. Scheduling skeleton-based grid applications using pepa and nws. *The Computer Journal*, 48(3), November 2005.

[7] A. Benoit, M. Cole, S. Gilmore, and J. Hillston. Evaluating the performance of pipeline-structured parallel programs with skeletons and process algebra. *Scalable Computing: Practice and Experience*, 6(4):1–16, 2005.

[8] Anne Benoit, Murray Cole, S. Gilmore, and J. Hillston. Enhancing the effecitve utilisation of grid clusters by exploiting on-line performability analysis. 2005.

[9] Bio-pepa users manual. Web, Feb 2019.

[10] F. Ciocchetta, F.ta and J. Hillston. Process algebras in systems biology. In Marco Bernardo, Pierpaolo Degano, and Gianluigi Zavattaro, editors, *Formal Methods for Computational Systems Biology*, volume 5016 of *8th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2008 Bertinoro, Italy, June 2-7, 2008*, pages 265–312. Springer, 2008.

[11] F. Ciocchetta and J. Hillston. Bio-pepa: a framework for the modelling and analysis of biochemical networks. Technical Report EDI-INF-RR-1231, University of Edinburgh, 2008.

[12] F. Ciocchetta and J. Hillston. Bio-PEPA: a framework for the modelling and analysis of biological systems em Theoretical Computer Science 410:(33-34):3065-3084, 2009.

[13] J. Cito and H. C. Gall. Using docker containers to improve reproducibility in software engineering research. In *Proc. IEEE/ACM 38th Int. Conf. Software Engineering Companion (ICSE-C)*, pages 906–907, May 2016.

[14] A. Clark and S. Gilmore. State-aware performance analysis with extended stochastic probes. *EPEW 2008, LNCS 5261*, 2008.

[15] James Edwards. Process algebras for protocol validation and analysis. In *Proceedings of the PREP 2001 Conference*, 2001.

[16] Kanimozhi Ellavarason. *An Automatic Mapping from the Systems Biology Markup Language to the Bio-PEPA Process Algebra*. European masters in informatics, University of Trento, 2008.

[17] Graham Clark, Stephen Gilmore, Jane Hilston. Specifying performance measures for pepa. In *Proceedings of the 5th International AMAST Workshop on Formal Methods for Real-Time and Probabilistic Systems*, 1999.

[18] Richard A. Hayden, Jeremy T. Bradley, and Allan Clark. Performance specification and evaluation with unified stochastic probes and fluid analysis. *IEEE Transactions on Software Engineering*, 39(1), January 2013.

[19] Anton Stefanek, Richard Hayden, and Jeremy Bradley. A new tool for the performance analysis of massively parallel computer systems. *arXiv preprint arXiv:1006.5104*, 2010.

[20] J. Hillston. The needham lecture - tuning systems: From composition to performance. *The Computer Journal*, 48(4), November 2005.

[21] J. Hillston. Process algebras for quantitative analysis. In *Proceedings of the 20th Annual Symposium on Logic in Computer Science (LICS'05)*, 2005.

[22] Andrew Hinton, Marta Kwiatkowska, Gethin Norman, and David Parker. Prism: A tool for automatic verification of probabilistic systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 441–444. Springer, 2006.

[23] Charles Antony, Richard Hoare. Communicating sequential processes. In *The origin of concurrent programming*, pages 413–443. Springer, 1978.

[24] Seth Cook, Spencer Julian, Michael Shuey. Containers in research: Initial experiences withlightweight infrastructure. In *XSEDE16 Proceedings of the XSEDE16 Conference on Diversity, Big Data, and Science at Scale*, 2016.

[25] Piotr Uminski, Madhusudhan Govindaraju, Pankaj Saha, Angel Beltre. Evaluation of docker containers for scientific workloads in the cloud. *Proceedings of the Practice and Experience on Advanced Research Computing Article No. 11*, July 2018.

[26] Robin Milner. *A calculus of communicating systems*. 1980.

[27] Radek Pelánek. Fighting state space explosion: Review and evaluation. In *International Workshop on Formal Methods for Industrial Critical Systems*, pages 37–52. Springer, 2008.

[28] Roger Peng. The reproducibility crisis in science: A statistical counter-attack. *Significance*, 12(3):30–32, 2015.

[29] Erin Gemma Scott. *Process Algebra with Layers: A Language for Multi-Scale Integration Modelling*. PhD thesis, Univeristy of Stirling, 2016.

[30] A. Stefanek, R. A. Hayden, and J. T. Bradley. Gpa - a tool for fluid scalability analysis of massively parallel systems. In *Proc. Eighth Int. Conf. Quantitative Evaluation of SysTems*, pages 147–148, September 2011.

[31] Stephen Gilmore, Mirco Tribastone, Adam Duguid. The pepa eclipse plugin. *ACM SIGMETRICS Performance Evaluation Review*, 36(4):28–33, March 2009.

[32] Yuyu Zhou, Balaji Subramaniam, Kate Keahey, and John Lange. Comparison of virtualization and containerization techniques for high performance computing. In *Proceedings of the 2015 ACM/IEEE conference on Supercomputing*. 2015.

[33] Jie Zhang, Xiaoyi Lu, and Dhabaleswar K. Panda. Is Singularity-based Container Technology Ready for Running MPI Applications on HPC Clouds? In *Proceedings of the10th International Conference on Utility and Cloud Computing*, pp. 151-160. ACM, 2017.

[34] Gregory M. Kurtzer, Vanessa Sochat, and Michael W. Bauer. Singularity: Scientific containers for mobility of compute. *PloS One* 12, no. 5 (2017): e0177459.