Reproducible High Performance Computing without Redundancy with Nix

Rohit Goswami
Science Institute
University of Iceland
Reykjavik, Iceland
rog32@hi.is
& *Quansight Labs*
Austin, TX, USA
rgoswami@quansight.com

Ruhila S.
Department of Biological Sciences
IISER Mohali
Mohali, India
ms20110@iisermohali.ac.in

Amrita Goswami
Science Institute
University of Iceland
Reykjavik, Iceland
amrita@hi.is

Sonaly Goswami
Department of Chemistry
Indian Institute of Technology Kanpur
Kanpur, India
sonaly@iitk.ac.in

Debabrata Goswami*
Department of Chemistry
Indian Institute of Technology Kanpur
Kanpur, India
dgoswami@iitk.ac.in

2022 Seventh International Conference on Parallel, Distributed and Grid Computing (PDGC) | 978-1-6654-5401-8/22/$31.00 ©2022 IEEE | DOI: 10.1109/PDGC56933.2022.10053342

***Abstract*—High performance computing (HPC) clusters are typically managed in a restrictive manner; the large user base makes cluster administrators unwilling to allow privilege escalation. Here we discuss existing methods of package management, including those which have been developed with scalability in mind, and enumerate the drawbacks and advantages of each management methodology. We contrast the paradigms of containerization via docker, virtualization via KVM, pod-infrastructures via Kubernetes, and specialized HPC packaging systems via Spack and identify key areas of neglect. We demonstrate how functional programming due to reliance on immutable states has been leveraged for deterministic package management via the nix-language expressions. We show its associated ecosystem is a prime candidate for HPC package management. We further develop guidelines and identify bottlenecks in the existing structure and present the methodology by which the nix ecosystem should be developed further as an optimal tool for HPC package management. We assert that the caveats of the nix ecosystem can easily mitigated by considerations relevant only to HPC systems, without compromising on functional methodology and features of the nix-language. We show that benefits of adoption in terms of generating reproducible derivations in a secure manner allow for workflows to be scaled across heterogeneous clusters. In particular, from the implementation hurdles faced during the compilation and running of the `d-SEAMS` scientific software engine, distributed as a nix-derivation on an HPC cluster, we identify communication protocols for working with SLURM and TORQUE user resource allocation queues. These protocols are heuristically defined and described in terms of the reference implementation required for queue-efficient nix builds.**

***Index Terms*—high-performance-computing, reproducible-research, nix-lang, functional-derivations, functional-package-management**

I. Introduction

Software management systems which can be scaled up for use on heterogenous computational resources have been recognized by the scientific computing community as a important focus area for software development. The rigorous requirements of scientific computing ensure that these software are optimized for running on large high performance computing clusters[1], [2], [3], [4], [5].

Best practices for various programming languages[6] and computational studies stress the importance of reproducible research[7] and workflows[8], but often implicitly assume that software components can be installed with `root` access.

Heterogeneous computing clusters are difficult to manage, in that traditional approaches tend to lock dependencies as is the norm for industrial use server operating systems like CentOS (now replaced typically with Rocky Linux) or the more commercial RedHat operating systems. Rolling distributions, which circumvent this locking of core-libraries are not typically seen in high performance computing environments, due to the high chance of packages breaking on updates. Cloud computing has become the industry standard which is based on containerization and pod based scaling systems like Kubernetes or `docker`. For personal computers, `docker` [9] has also been widely adopted by the computational community for the ease with which different operating systems and packages may be used interchangeably. Light-weight containers and containerization have become more viable though there have also been extensions to the Linux kernel which favor traditional virtualization. The relative merits of both approaches have been addressed by different groups in the literature [10], [11], [12], [13], [14]. However, performance aspects aside, `docker` is well documented to be plagued with security risks [15], [16], mostly since the inclusion of a user in the `docker` group is presently tantamount to being granted superuser access to the machine. For industrial situations where machines are provisioned on the cloud as virtual machines, this is not an

issue, but for systems which are managed on premise, like for most academic clusters such unrestricted access is a major security vulnerability. In academic settings, the true multi-user nature of the monolithic Linux kernel is still leveraged and user permissions are strictly enforced. Additionally, the model of computation still prevalent in most of the world's High Performance Clusters is that dominated by the process queue. Typically, users have access to only their own home folder, isolated in part from the rest of the machine, and are not granted the ability to install or upgrade packages. To some extent this may be mitigated by the local compilation of packages or by leveraging local-install helpers such as `brew`. In recent times there has also been a proliferation of software systems designed to overcome these traditional design concerns. Previously, there have been some attempts to describe package management with `nix` [17], however, these have not described the theoretical foundations underpinning the viability of `nix`. Here, we establish a systematic methodology for the continued prevalence of `nix` for HPC systems by positing extensions for parallelism.

II. Present Scenario

Most HPC clusters operate via simple UNIX permissions coupled with a queue system for resource allocation. SLURM [18] and TORQUE [19] are two popular choices. These systems are typically used to run a variety of time and resource constrained queues (like small/medium/large queues for say, 24/48/72 hours and 18/32/64 nodes). Users are typically constrained to a `$HOME` directory on a shared file system, as well as access to a temporary, but faster access `/scratch` directory. Though users are able to install packages locally and provide local overlays to the module system (e.g. `lmod`), these endeavors are often constrained by the proliferation of old library headers and compilers on the main system. Rolling distributions like ArchLinux or even the new CentOS Stream are rare in the scientific HPC community, the admins are often unable to comply with any requests to upgrade the system without a full shut-down, which is typically not viable.

III. Alternatives

HPC package management is an active area of research, with the ability to work without superuser access without compromising on security being the most common feature. An overview of some of the more popular package management systems which have been considered for use on HPC clusters are summarized in Table I and described further in this section.

A. Kubernetes

Kubernetes is an outgrowth of the billing cycle, in the sense that it provides an abstraction of the pod ecosystem [20]. The system is meant for fast asynchronous usage, and is thus not very feasible in the academic environment, which, as mentioned above, has multiple queues and the entire system is in constant operation.

B. Docker

The main drawback of the light-weight container system brought by `docker` is the security implications [16], [21]. The nature of `docker` containers allows users to mount arbitrary folders into their `docker` containers. Additionally, building the images themselves is time consuming and makes little sense in the fast development cycle, which is characteristic of scientific code. Furthermore, the `docker` cache is not stored efficiently across machines, causing data redundancy, unless paired with file-systems which are not considered to be stable for production, such as BTRFS (b-tree file system) [22].

C. Spack

Spack [21] is a package management system designed from the ground up for HPC usage. However, though there were considerations made to allow for YAML based package descriptions, it lacks the ability to define immutable build products and has no functional abstractions. Thus, the user has to deal with version dependencies in a non-trivial manner in some situations, making it less than ideal.

D. Virtual Machines

In-spite of increased kernel virtualization support, VMs (Virtual Machines) have been proven to be too slow [20] for viable usage in HPC clusters.

IV. Nix

The idea of reproducible derivations of package management also stems from functional programming considerations. Nix [23] is a functional language for describing packages. It is cross platform, runs on Linux and MacOS, and also has many pre-compiled packages as well. Nix has also hitherto been considered largely as a methodology to improve the industrial standards for continuous integration systems (in the form of Hydra) as a `docker` replacement [24] or as a full blown operating system (NixOS) [25]. The nix system is managed by a daemon. As of this date, the default number of `nixbld` (default name) users created by the multi-install option is thirty two, though this may be overridden by the NIX_USER_COUNT variable at the time of installation. The basic outline of the system [23] is described as follows:

- The nix derivation is read
- The dependency graph is traversed from the derivation
- The items in the dependency graph are built (immutably)
- The derivation is used to create a single path in the nix store

The single path in the list above refers to the unique cryptographic hash, which is generated under the system root level `/nix` store, with user specific versions as shown in Figure 1. Since this hash is linked to the dependency graph which is static, it can be shared across machines and is guaranteed to produce the exact same binaries on each machine. The nix-shell [26] may be used for development purposes as a reproducible way to ensure all bugs and features work as intended, in a more light weight manner than utilizing a virtual machine. Several helper tools like lorri [27] have been

TABLE I
TABULATED COMPARISON OF HPC PACKAGE MANAGEMENT SYSTEMS.

Package Manager	Reproducible builds	Automated Managed Dependencies	Encapsulation	Number of Packages	Cross-Platform Compatibility
Docker	yes	no	yes	-	yes
Kubernetes	yes	no	yes	-	yes
Spack	no	yes	no	>3,000	no
Nix	yes	yes	yes	>40,000	yes

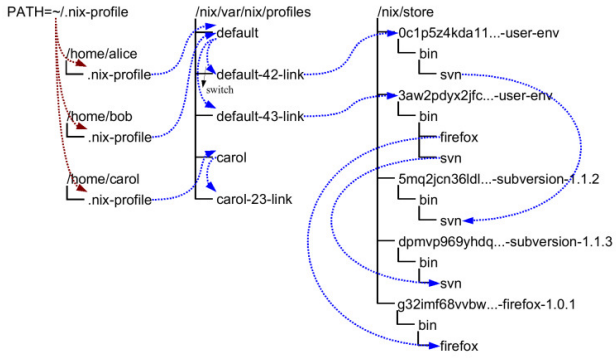

Fig. 1. An illustration of the nix user profile system [26]

developed to further automate the process of utilizing isolated environments in a deterministic and transparent manner. This is conceptually clarified by the depiction in Figure 2.

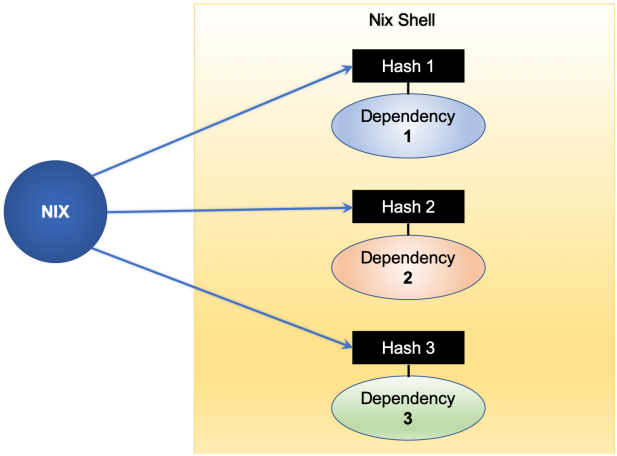

Fig. 2. Qualitative description of nix build guarantees

A. Caveats

Currently, there are still side-effects of users being granted access to the nix-build system. As the nix-build daemon is not managed by the existing queue systems (SLURM or TORQUE), the load borne by the system due to arbitrary builds requested by users are not deterministic and may cause slowdowns. Furthermore, the cache distribution under various file-systems is still ill-defined. There is no official methodology to share the nix-store among machines and the access controls must be better defined for the same. Ad-hoc approaches include forcibly sharing out the /nix store, however, this requires a lot of setup that needs superuser access. A long standing issue which trips up new users is the fact that nix derivations are not FHS (Filesystem Hierarchy Standard) [25] compliant by default, so utilizing the nix-shell to try to install packages will often give unexpected results.

V. PERFORMANCE EXTENSIONS

We have compiled and run a scientific software analysis tool called d-SEAMS [28] which draws on a multitude of `C++` libraries and is distributed as a `nix` derivation. The implementation of this software is such that it requires significant computing resources for the algorithms [4], [1] and large volume of input simulation data to be parsed. The `nix` build ensured that the binary was reproducible, however, due to bottlenecks in the nix-ecosystem, we were required to escalate privileges and modify the existing `nix-store` to ensure access to the same hashes across each system in our cluster. Based on these implementation hurdles, we enumerate the directions in which extensions to the nix ecosystem are required to improve performance and viability for academic HPC clusters.

A. Distributed Builds

The multi-user installation of nix essentially spawns a set of `nix-build` users on the computer, which are then used to communicate with the nix systemd daemon to execute builds in a transparent and immutable manner. The connection between the daemon and the users executing the build is possible to be distributed across machines, however, this requires the addition of a `nix-multi-master` daemon, to ensure access and even resource allocation. The basic idea is to have a poll based consensus instead of a master-slave system, which would best leverage the benefit of the immutability of each nix derivation output. The steps would be roughly as follows:

- The build request is forwarded by an unprivileged user
- The daemon communicates across the network to determine the approximate resource usage
- Based on a round-robin scheme, the build is allocated resources on a machine
- The final output is still a single path under the shared `/nix` store

An important consideration in the system above is the extent to which the `nix-systemd` daemon can be modified to utilize the existing queue infrastructure. HPC queues typically allocate resources as per their queue position and priority, hence it may be best to dedicate a single node for builds, and add it back to the processing queue, only if there are no builds requested across the network. The implementation of the said scheme would require both support at the queue software

end, and modifications to the `nix-daemon`. Furthermore, the nature of the file-system is also of importance for the efficient sharing of the `/nix` store.

B. Nix with SLURM

A simplistic reference implementation of the distributed build system above is to ensure communication between the system-daemon run on each node cluster, and then simply run builds in a queue. Under this setup, a separate SLURM queue may be initialized to run in parallel to the existing user-queue but which only processes the `nixbld` user resource requests, hence offloading much of the complexity of the scheme described above. This implementation is shown in Figure 3.

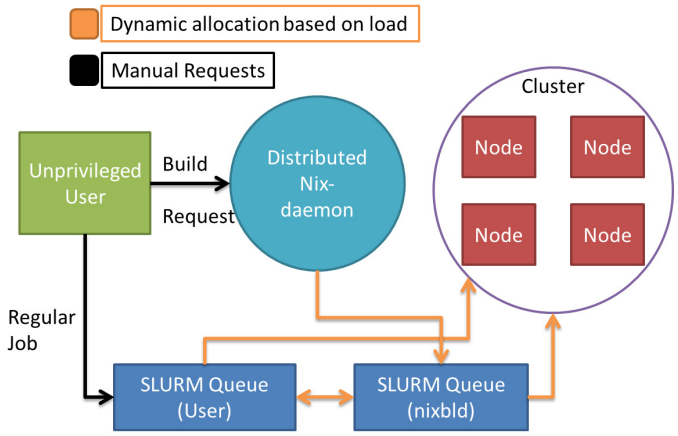

Fig. 3. A reference implementation of the build queue system for parallelization

VI. Conclusion

We have identified gaps in the framework of existing package management systems. We have briefly outlined the manner in which the nix package management system is superior to existing frameworks for High Performance Computing clusters mostly by virtue of the functional nature of the packaging derivations. We from the experience of implementing existing HPC software we identify performance bottlenecks. We show how these might be avoided in distributed systems. Furthermore, the reproducible nature of the derivations and the ability to pin packages forms a language and device agnostic framework to ensure that derivations will always compile. We also assert that the nix multi-user build system is ideal for distributed workloads, especially when paired with simple queuing systems. We demonstrate guidelines and directions for the growth of the nix ecosystem for HPC package management. In particular we have discussed the manner in which the `nix-build` system is to integrate with the existing HPC resource allocation queues like SLURM and TORQUE. We enumerate the goals of this framework performance extension and discuss the basic theoretical proof-of-concept which can be achieved by the existing ecosystem. We thus indicate a new avenue of research, which holds great promise due to the emphasis on security, resource allocation, and provide reproducible results. Thus we expect that the methodology described herein will be widely adopted.

Acknowledgments

R.G. is partially supported by the Icelandic Research Fund, grant no. 217436–052 and A.G. is partially supported by the Icelandic Research Fund, grant no. 228615–051. This work was supported by the Indian Space Research Organization (ISRO) Space Technology Cell (STC) and the Govt. of India Ministry of Electronics and Information Technology (MEITY). We acknowledge support from the HPC cluster of the Computer Center (CC), Indian Institute of Technology Kanpur.

References

[1] A. Goswami and J. K. Singh, "A General Topological Network Criterion for Exploring the Structure of Icy Nanoribbons and Monolayers," *arXiv:1909.09827 [cond-mat, physics:physics]*, Sep. 2019.

[2] R. Goswami, A. Goswami, and D. Goswami, "Space Filling Curves: Heuristics For Semi Classical Lasing Computations," in *2019 URSI Asia-Pacific Radio Science Conference (AP-RASC)*. New Delhi, India: IEEE, Mar. 2019, pp. 1–4.

[3] R. Goswami and D. Goswami, "Quantum Distributed Computing with Shaped Laser Pulses," in *13th International Conference on Fiber Optics and Photonics*. Kanpur: OSA, 2016, p. W4C.3.

[4] Prerna, R. Goswami, A. K. Metya, S. V. Shevkunov, and J. K. Singh, "Study of ice nucleation on silver iodide surface with defects," *Molecular Physics*, pp. 1–13, Aug. 2019.

[5] R. Goswami, "Wailord: Parsers and Reproducibility for Quantum Chemistry," *Proceedings of the 21st Python in Science Conference*, pp. 193–197, 2022.

[6] L. J. Kedward, B. Aradi, O. Čertík, M. Curcic, S. Ehlert, P. Engel, R. Goswami, M. Hirsch, A. Lozada-Blanco, V. Magnin, A. Markus, E. Pagone, I. Pribec, B. Richardson, H. Snyder, J. Urban, and J. Vandenplas, "The State of Fortran," *Computing in Science & Engineering*, vol. 24, no. 2, pp. 63–72, Mar. 2022.

[7] R. D. Peng, "Reproducible Research in Computational Science," *Science*, vol. 334, no. 6060, pp. 1226–1227, Dec. 2011.

[8] S. P. Huber, "Automated reproducible workflows and data provenance with AiiDA," *Nature Reviews Physics*, vol. 4, no. 7, pp. 431–431, Jul. 2022.

[9] D. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and Deployment," *Linux J.*, vol. 2014, no. 239, Mar. 2014.

[10] M. Chae, H. Lee, and K. Lee, "A performance comparison of linux containers and virtual machines using Docker and KVM," *Cluster Computing*, vol. 22, no. 1, pp. 1765–1775, Jan. 2019.

[11] A. M. Joy, "Performance comparison between Linux containers and virtual machines," in *2015 International Conference on Advances in Computer Engineering and Applications*, Mar. 2015, pp. 342–346.

[12] N. Haydel, G. Madey, S. Gesing, A. Dakkak, S. G. de Gonzalo, I. Taylor, and W.-m. W. Hwu, "Enhancing the Usability and Utilization of Accelerated Architectures via Docker," in *Proceedings of the 8th International Conference on Utility and Cloud Computing*, ser. UCC '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 361–367.

[13] A. K. Yadav, M. L. Garg, and Ritika, "Docker Containers Versus Virtual Machine-Based Virtualization," in *Emerging Technologies in Data Mining and Information Security*, ser. Advances in Intelligent Systems and Computing, A. Abraham, P. Dutta, J. K. Mandal, A. Bhattacharya, and S. Dutta, Eds. Singapore: Springer, 2019, pp. 141–150.

[14] M. Plauth, L. Feinbube, and A. Polze, "A Performance Survey of Lightweight Virtualization Techniques," in *Service-Oriented and Cloud Computing*, ser. Lecture Notes in Computer Science, F. De Paoli, S. Schulte, and E. Broch Johnsen, Eds. Cham: Springer International Publishing, 2017, pp. 34–48.

[15] A. Duarte and N. Antunes, "An Empirical Study of Docker Vulnerabilities and of Static Code Analysis Applicability," in *2018 Eighth Latin-American Symposium on Dependable Computing (LADC)*, Oct. 2018, pp. 27–36.

[16] T. Bui, "Analysis of Docker Security," *arXiv:1501.02967 [cs]*, Jan. 2015.

[17] B. Bzeznik, O. Henriot, V. Reis, O. Richard, and L. Tavard, "Nix as HPC package management system," in *Proceedings of the Fourth International Workshop on HPC User Support Tools - HUST'17*. Denver, CO, USA: ACM Press, 2017, pp. 1–6.
[18] A. B. Yoo, M. A. Jette, and M. Grondona, "SLURM: Simple Linux Utility for Resource Management," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds. Springer Berlin Heidelberg, 2003, pp. 44–60.
[19] V. Chlumsky, D. Klusacek, and M. Ruda, "The extension of TORQUE scheduler allowing the use of planning and optimization in grids," *Computer Science*, no. Vol. 13 (2), pp. 5–19, 2012.
[20] V. Medel, R. Tolosana-Calasanz, J. Á. Bañares, U. Arronategui, and O. F. Rana, "Characterising resource management performance in Kubernetes," *Computers & Electrical Engineering*, vol. 68, pp. 286–297, May 2018.
[21] T. Gamblin, M. LeGendre, M. R. Collette, G. L. Lee, A. Moody, B. R. de Supinski, and S. Futral, "The Spack Package Manager: Bringing Order to HPC Software Chaos," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15. New York, NY, USA: ACM, 2015, pp. 40:1–40:12.
[22] O. Rodeh, J. Bacik, and C. Mason, "BTRFS: The Linux B-Tree Filesystem," *Trans. Storage*, vol. 9, no. 3, pp. 9:1–9:32, Aug. 2013.
[23] E. Dolstra, M. de Jonge, and E. Visser, "Nix: A Safe and Policy-Free System for Software Deployment," p. 15, 2004.
[24] E. Dolstra and E. Visser, "Hydra: A Declarative Approach to Continuous Integration," p. 18.
[25] E. Dolstra, A. Löh, and N. Pierron, "NixOS: A purely functional Linux distribution," *Journal of Functional Programming*, vol. 20, no. 5-6, pp. 577–615, Nov. 2010.
[26] "Nix manual," https://nixos.org/nix/manual/.
[27] "Tweag I/O - Introducing lorri, your project's nix-env," https://www.tweag.io/posts/2019-03-28-introducing-lorri.html.
[28] R. Goswami, A. Goswami, and J. K. Singh, "D-SEAMS: Deferred Structural Elucidation Analysis for Molecular Simulations," *Journal of Chemical Information and Modeling*, vol. 60, no. 4, pp. 2169–2177, Apr. 2020.