A Roadmap to Robust Science for High-throughput Applications: The Developers' Perspective

M. Taufer*, E. Deelman†, R. Ferreira da Silva†, T. Estrada‡, M. Hall§, and M. Livny¶
*U. Tennessee Knoxville, †U. Southern California,
‡ U. New Mexico, §U. Utah, ¶U. Wisconsin–Madison

***Abstract*—Scientists using the high-throughput computing (HTC) paradigm for scientific discovery rely on complex software systems and heterogeneous architectures that must deliver robust science (i.e., ensuring performance scalability in space and time; trust in technology, people, and infrastructures; and reproducible or confirmable research). Developers must overcome a variety of obstacles to pursue workflow interoperability, identify tools and libraries for robust science, port codes across different architectures, and establish trust in non-deterministic results. This poster presents recommendations to build a roadmap to overcome these challenges and enable robust science for HTC applications and workflows. The findings were collected from an international community of software developers during a Virtual World Café in May 2021.**

***Index Terms*—Performance Scalability, Trustworthiness, Reproducibility**

I. Problem and Contributions

High-throughput applications, in which application workload consists of a large ensemble of self-contained tasks and application performance is measured by the number of tasks completed per unit of time, are vital for scientific discovery. These applications combine multiple components into increasingly complex multi-modal workflows (i.e., data generation; data collection and merging; data pre-processing and feature extraction; data analysis and modelling; and data verification, validation, and visualization) that are executed in concert on large-scale heterogeneous systems including high performance computing (HPC), distributed systems, and cloud platforms. Managing an HTC workload is not an easy task: it requires automation services that include workflow capabilities for managing all the tasks and all the input/output files across different runs of an HTC ensemble. These increasing complexities hinder the ability of scientists to generate robust science, which we define as the capacity of high-throughput applications to scale in performance (i.e., meeting both hardware and software performance expectations when executed on heterogeneous resources and large scale systems); to exhibit trustworthiness (i.e., depend upon hardware and software technology, collaborators across scientific domains, and platform providers to behave as specified or expected); and to assure reproducibility (i.e., drawing the same scientific conclusions using the knowledge encapsulated in the original computational experiment). This poster presents recommendations for designing and implementing software systems for robust science across critical high-throughput applications. The findings were collected through one virtual mini-workshop in May 2021 [2] called a Virtual World Café (VWC). In the VWC, we engaged communities of software developers to share state of the art recommendations through structured conversational processes: participants were distributed across several breakout sessions in an online meeting, with participants switching sessions periodically and getting introduced to the previous discussion at their new session by a session lead.

The work in this poster is funded by the National Science Foundation (NSF) under grants #2028881, #2028923, #2028930, #2028955, and #2028956.

II. Findings and Recommendations

We sort findings and recommendations from the VWC into four categories:

1. Interoperability through Application Programming Interfaces (APIs). *Findings:* HTC workflows are becoming more and more complex. Software systems supporting those workflows are multilayered, expanding both vertically (e.g., multiple software layers) and horizontally (e.g., multiple alternative platforms and software systems for each specific platform). Interoperability is thus a multilayer challenge. General APIs can serve as interoperability solutions, but scientists tend to prefer bespoke software and hardware fixes rather than deploying more general solutions. The needs of different domains can complicate the definition of any general APIs.

Recommendations: (a) Study what level of abstraction is suitable for users across a broad range of domains (i.e., what parts of a system should be made transparent to the user and what should be hidden by an API). (b) Identify the proper location of APIs, determining if APIs should be located at specific layers (e.g., SLURM, HTCondor, and EC2) or should globally wrap multiple layers.

2. Cataloging tools and libraries. *Findings:* Relevant tools and libraries are available to support robust sciences for workflows orchestration (e.g., Airflow, Kuberflow); for workflows performance monitoring (e.g., Hatchet); for workflows annotations and reproducibility (e.g., Popper, Reprozip. Verificarlo); and for workflows management (e.g., Pegasus, DASK, MOCHI, Galaxy, binder.org, Ray), including commercial workflow management platforms (e.g., Amundsen and Databook). When selecting available tools, users often face applicability challenges. Very few tools work across domains; tools popular and effective in one domain may not be so in other domains. When using different tools or different tool versions, users face interoperability challenges; even when

using Spack and containers, installations and deployments are not always easy across platforms and architectures. Ultimately, users may not be able to identify the most suitable ecosystem for their application.

Recommendations: (a) List tools and libraries that are available and used in different domains or for specific goals. (b) Define a searchable taxonomy of the tools and libraries, including platform and type of support (e.g., reproducibility and trustworthiness) within a domain and across domains.

3. Code portability and algorithm correctness. *Findings:* Scientists desire codes to be written once and then run on different architectures. Portability frameworks such as Kokkos and Raja fit into this vision but support limited data layouts. For example, Kokkos allows the execution of codes such as VPIC on both GPUs and CPUs but lacks the capability to handle vectors effectively. Consequently, VPIC on Kokkos can no longer match historical performance observed with optimized vectorization tailored for specific architectures. In general, data layouts impact data movement, and data movement impacts performance. For example, a domain-specific framework targeting stencil computations can offer a broad set of data layout abstractions, data layout transformation, and code generations. Such frameworks use higher level descriptions of the computation, generate specific data layout optimizations targeting different platforms, and combine different software components, some provided by the application programmer and some by manually tuning specifications. In both cases, when porting codes across platforms, scientists value correct algorithms over numerical reproducibility. Tools for assessing correctness by comparing results across platforms, architectures, compilers, or compiler upgrades are missing.

Recommendations: (a) Define tolerable result differences when moving a code from one platform to another, or from one architecture to another (e.g., from CPU and GPU). (b) Build the infrastructures so that algorithms can be trusted to deliver results across platforms within the defined error tolerances.

4. Trust in data, software, infrastructures, and people. *Findings:* When it comes to trustworthiness, HPC and HTC have orthogonal views. HPC runs on centrally controlled resources that allow system administrators to capture immediate states during the execution. While log files exist capturing the history of jobs, HPC centers and national labs are reluctant to pursue trace or system logs sharing. HTC, on the other hand, runs on resources that are locally controlled by different autonomous entities and thus create a different trust model based on blockchain and crypto techniques (with all logs of every transaction made and jobs signed by every participant) or, more simply, job passports and credentials attached to a job. Scientists trust the resource's access point but do not trust the different execution points. Under these circumstances, tracing data provenance is essential for establishing trustworthiness. Annotating workflows and containerizing their executions can automatically capture data, software, and infrastructure information, but it may be difficult to identify appropriate granularity of annotations and containerizations. System conditions (such as file systems, networking, and resource usage) play a key role when addressing aspects of result reproducibility, performance reproducibility, result explainability, and data provenance, but these system conditions are often neglected. Furthermore, there is the danger of metadata explosions (in size) when metadata can become larger than the data itself and annotations can substantially complicate the workflows. There are also potential security risks when sharing metadata: the metadata may reveal potential vulnerability of an annotated system and facilitate system attacks. Last, collected metadata needs to be readable, parsable, and useful for both human and ML tools with their different format requirements, and it is necessary to co-design the metadata collection with the support of both scientists and software developers. Workflow annotations, metadata curation, metadata management, and scientific outreach must all be directly connected to what the scientific community needs; scientists must be equal partners in the annotation process.

Recommendations: (a) Provide users (both scientists and software developers) with different levels to capture metadata (e.g., a systems developer is more interested in capturing metadata related to the network storage while a scientist may be more interested in the execution of the workflow). (b) Talk with different communities and define what types of metadata are relevant for each of them before systematically building abstractions to collect and coordinate metadata. The collection and coordination process should be automated; best practices for packaging and sharing metadata should prevent security breaches. (c) Create a marketplace of reusable annotated artifacts (including metadata), and identify the business model for the management of the marketplace (e.g., where to store the content, how to provide access, what is the lifespan of an artifact and its data, who is in charge for the validation and security of the artifact and its data).

III. Conclusions and Relevance of the Work

The findings presented in this poster can support communities of software developers that work together to define, design, implement, and use general system software solutions for robust science. Any solution should span five critical areas: architecture; systems; high performance computing; programming models and compilers; and algorithms and theory. In our future work, we aim to combine these areas into an integrated continuum through AI-orchestrations, policies, and practices.

Acknowledgment

The authors thank the participants in the May 2021 VWC for the vibrant discussions. Findings and recommendations in this poster are the results of those discussions.

References

[1] M. Livny, J. Basney, R. Raman, and T. Tannenbaum, Todd "Mechanisms for High Throughput Computing" In SPEEDUP Journal, 11(1), 36–40, 1997.

[2] M. Taufer, E. Deelman, R. Ferreira da Silva, T. Estrda, and M. Hall "Performance Scalability, Trust, and Reproducibility: A Community Roadmap to Robust Science in High-throughput Applications," In https://robustscience.org/