Continuous Infrastructure Components for HPC-IO Projects

Author: Kenneth Casimiro (University of California, San Diego, Lawrence Berkeley National Laboratory)

Primary Mentor: Suren Byna
Associate Mentor: Houjun Tang
Associate Mentor: Jean Luca Bez

Abstract

Continuous infrastructure enables better transparency and visibility in the process of software development and delivery. Software development teams need to streamline the development process by breaking down the process into small manageable tasks. One way to improve the workflow is through continuous integration. Continuous integration is the software development practice of regularly integrating code changes into a shared code repository. This practice encourages committing small changes more often over committing large changes infrequently. Each commit triggers a build during which tests are run that help to identify if anything was broken by the changes. The significance of continuous integration is that it reduces risk, promotes communication, reduces waiting time and results in a higher product quality for the project. My role in the SDM Group is to create and deploy continuous infrastructure components to several repositories from HPC-IO. I created scripts that validate code changes to the repositories with automated testing through GitHub Actions. I developed documentation using Sphinx and web hosting on ReadtheDocs. Additionally, I made Spack package recipes to distribute the software with ease and ran scientific applications and benchmarks on high performing computing (HPC) systems to evaluate their performance. Throughout the internship, I learned the best practices in HPC systems in building and developing HPC software infrastructures for ECP software and applications.

Introduction

Software development is an ever-growing industry, and the need for continuous infrastructure is required to accelerate the development and delivery of software features without compromising quality. Traditional software development processes are no longer appropriate. Those processes are prone to error due to the slow-release rate of the product, meaning slower fixes. Because the feedback from the customers is less frequent, it is harder to evaluate if the delivered product fulfills all the needs and requirements. With continuous integration practices, it is possible to provide new features more frequently to clients. Continuous integration offers several benefits: getting more and quick feedback from the software development process; having frequent and reliable releases, which lead to improved satisfaction and product quality; the strengthened connection between developers and automation can eliminate manual tasks. Although this practice advocates integrating work-in-progress multiple times per day, continuous infrastructure is about quickly and reliably releasing software to clients by bringing automation support as much as possible. A growing number of industrial cases indicate that incorporating continuous infrastructure components makes inroads in software development practices across various domains and sizes of organizations [1]. Continuous practices are rapidly growing in popularity. Therefore, for this internship, I'll be building infrastructure components for HPC-IO projects for the Scientific Data Management Research (SDM) Group.

The SDM Group is one of the research groups of the Data Science and Technology Department of the Computational Research Division at Lawrence Berkeley National Laboratory. The role of the group is to face significant challenges in organizing, managing, and analyzing data by developing computer applications to meet those challenges. The SDM Group develops technologies and tools for efficient data access, data storage, data analysis, and management of massive scientific data sets. The current projects in the group include data querying technologies, in situ feature extraction algorithms, data analysis, machine learning algorithms, storage resource management tools, and software platforms for exascale data. Additionally, the group works closely with application scientists to address their data management challenges. These tools and application development activities are backed by active research efforts on novel algorithms for emerging hardware platforms.

HPC-IO is a GitHub organization that contains various software tools that improve storing and accessing data (i.e., performing I/O) efficiently on high-performance computing (HPC) systems. This GitHub organization contains many software repositories, such as Proactive Data Containers (PDC), asynchronous I/O, and caching Virtual Object Layer (VOL) connectors for HDF5, and H5bench (an HDF5 benchmark suite). In this project, I built numerous infrastructure components for the software repositories in the HPC-IO organization, including continuous integration (CI), documentation, software packaging with spack, and implementing coding standards.

Materials & Methods

Materials

For this internship, I contributed to the ExaIO project in the Exascale Computing Project (ECP) and the Proactive Data Containers (PDC) project by creating and deploying continuous integration (CI) to several code repositories, which drives code developers to check in validated code changes to repositories frequently with automated testing; setting up documentation hosting websites other user-facing interactions, which provides useful information to the users of our software; and running scientific applications and benchmarks on high performance computing (HPC) systems to evaluate their performance. I learned about the best practices in HPC systems as well as build and develop HPC software infrastructures for ECP software and applications. The projects that I worked with are the following: Proactive Data Containers (PDC), asynchronous I/O, and caching Virtual Object Layer (VOL) connectors for HDF5, and H5bench (an HDF5 benchmark suite).

Proactive Data Containers (PDC):
PDC is a data management system with an object-centric API and a runtime system with a set of data object management services. These services allow placing data in the memory and storage hierarchy, performing data movement asynchronously, and providing scalable metadata operations to find data objects and interested regions in those objects. PDC revolutionizes how data is stored and accessed by using object-centric abstractions to represent data that moves in the high-performance computing (HPC) memory and storage subsystems that are often heterogeneous and complex[2].

HPC-IO VOL connectors (Async I/O, Cache, Provenance):
The Virtual Object Layer (VOL) is an abstraction layer within the HDF5 library that redirects I/O operations into a VOL "connector", immediately after an API routine is invoked. These connectors are Asynchronous I/O, Node Local Storage Cache, and Provenance, which will implement different features and capabilities for HDF5 within the VOL framework. These VOL connectors are the mechanisms to implement storage for those HDF5 objects. Async I/O allows asynchronous operations for HDF5, Node Local Storage Cache connector allows efficient parallel I/O through caching data on node-local storage, and Provenance VOL transparently intercepts HDF5 calls and record operations at multiple levels, namely file, group, dataset, and data element levels [3].

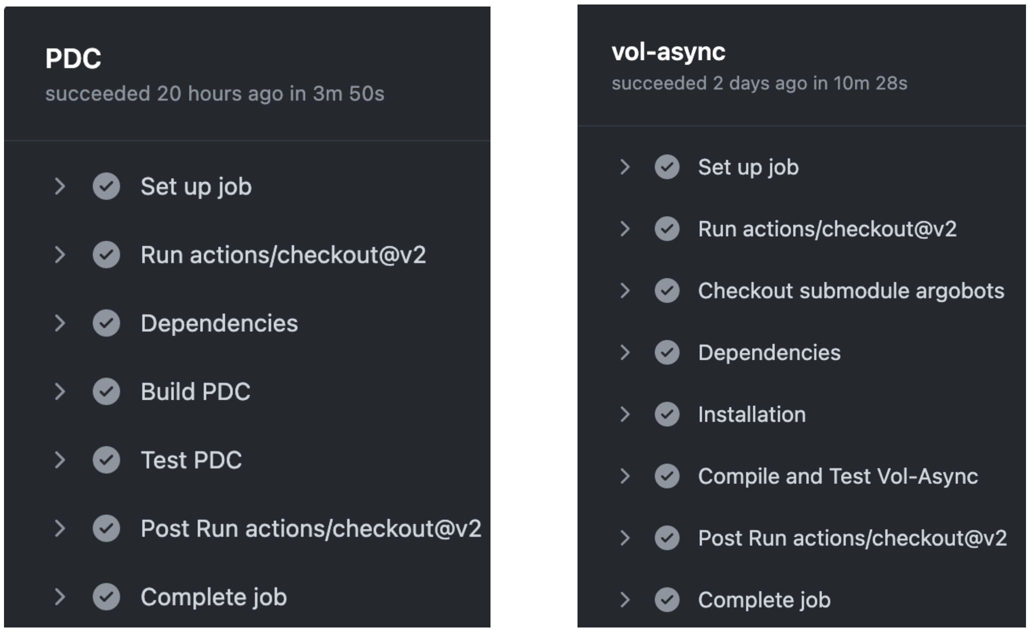
H5bench:
A suite of parallel I/O benchmarks or kernels representing I/O patterns that are commonly used in HDF5 applications on high performance computing systems. H5bench measures I/O performance from various aspects, including the I/O overhead, observed I/O rate, etc.

Methods

Through this project, I explored building continuous infrastructure for ECP software repositories. My main goal of the internship is to build and develop HPC software infrastructure for software hosted on GitHub. I built upon the initial design of continuous integration effort where I explored testing and running continuous integration (CI) scripts automatically on servers. The infrastructure components that I built for the ECP software repositories will be the following: continuous integration, documentation, and software packaging.

Continuous Integration:

Continuous integration is the software development practice of regularly integrating code changes into a shared code repository. This practice encourages committing small changes more often over committing large changes infrequently. Each commit triggers a build during which tests are run that help to identify if anything was broken by the changes. The significance of continuous integration is that it reduces risk, promotes communication, reduces waiting time and results in a higher product quality for the project. CI reduces risk by having frequent testing and deployment of code. Code defects and bugs can be detected earlier. These bugs and errors can be easily fixed and take less time, making the overall process easier. CI also creates the workflow easy and regularized. Resulting in better communication that allows the process to be more transparent and collaborative among team members. The CI I've implemented was done through GitHub Actions, where it automated the build, installation, and testing of the projects in a Linux development system.


PDC
succeeded 20 hours ago in 3m 50s

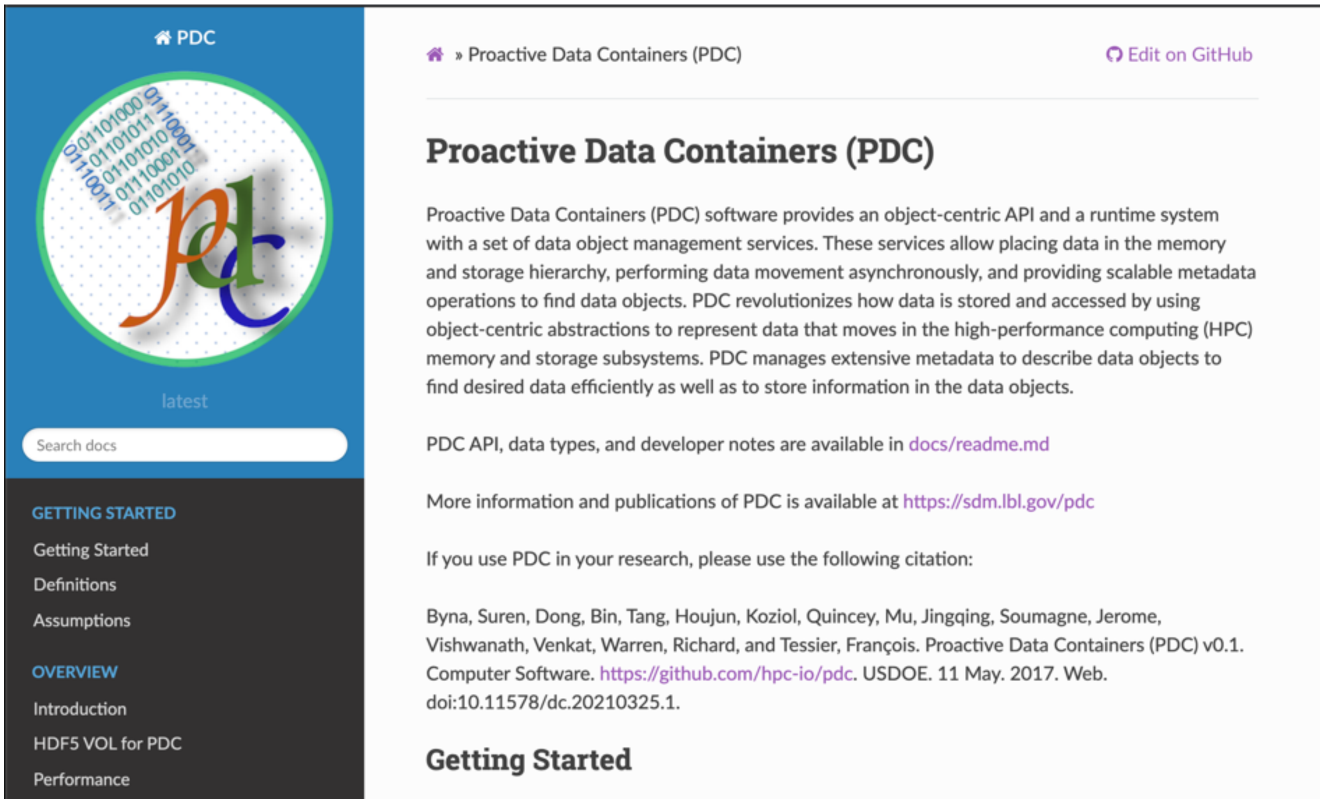
- Set up job
- Run actions/checkout@v2
- Dependencies
- Build PDC
- Test PDC
- Post Run actions/checkout@v2
- Complete job

vol-async
succeeded 2 days ago in 10m 28s

- Set up job
- Run actions/checkout@v2
- Checkout submodule argobots
- Dependencies
- Installation
- Compile and Test Vol-Async
- Post Run actions/checkout@v2
- Complete job

Figure 1: GitHub Actions for building, installing, and testing on a Linux development system. The left image shows the CI for PDC and the right shows the CI for Vol-Async.

Documentation:
Documentation is also essential to software development. The presence of documentation helps keep track of all aspects of an application and it improves the quality of a software product. Its main focuses are development, maintenance and knowledge transfer to other developers. The significance of successful documentation will make information easily accessible and allow new users to learn quickly on how to use your install and deploy your software. This is done through Sphinx, a documentation generator written. Through Sphinx, I used reStructuredText (RST) files for technical documentation where I hosted these documentations on the project's respective ReadtheDocs domain.



Proactive Data Containers (PDC)

Proactive Data Containers (PDC) software provides an object-centric API and a runtime system with a set of data object management services. These services allow placing data in the memory and storage hierarchy, performing data movement asynchronously, and providing scalable metadata operations to find data objects. PDC revolutionizes how data is stored and accessed by using object-centric abstractions to represent data that moves in the high-performance computing (HPC) memory and storage subsystems. PDC manages extensive metadata to describe data objects to find desired data efficiently as well as to store information in the data objects.

PDC API, data types, and developer notes are available in docs/readme.md

More information and publications of PDC is available at https://sdm.lbl.gov/pdc

If you use PDC in your research, please use the following citation:

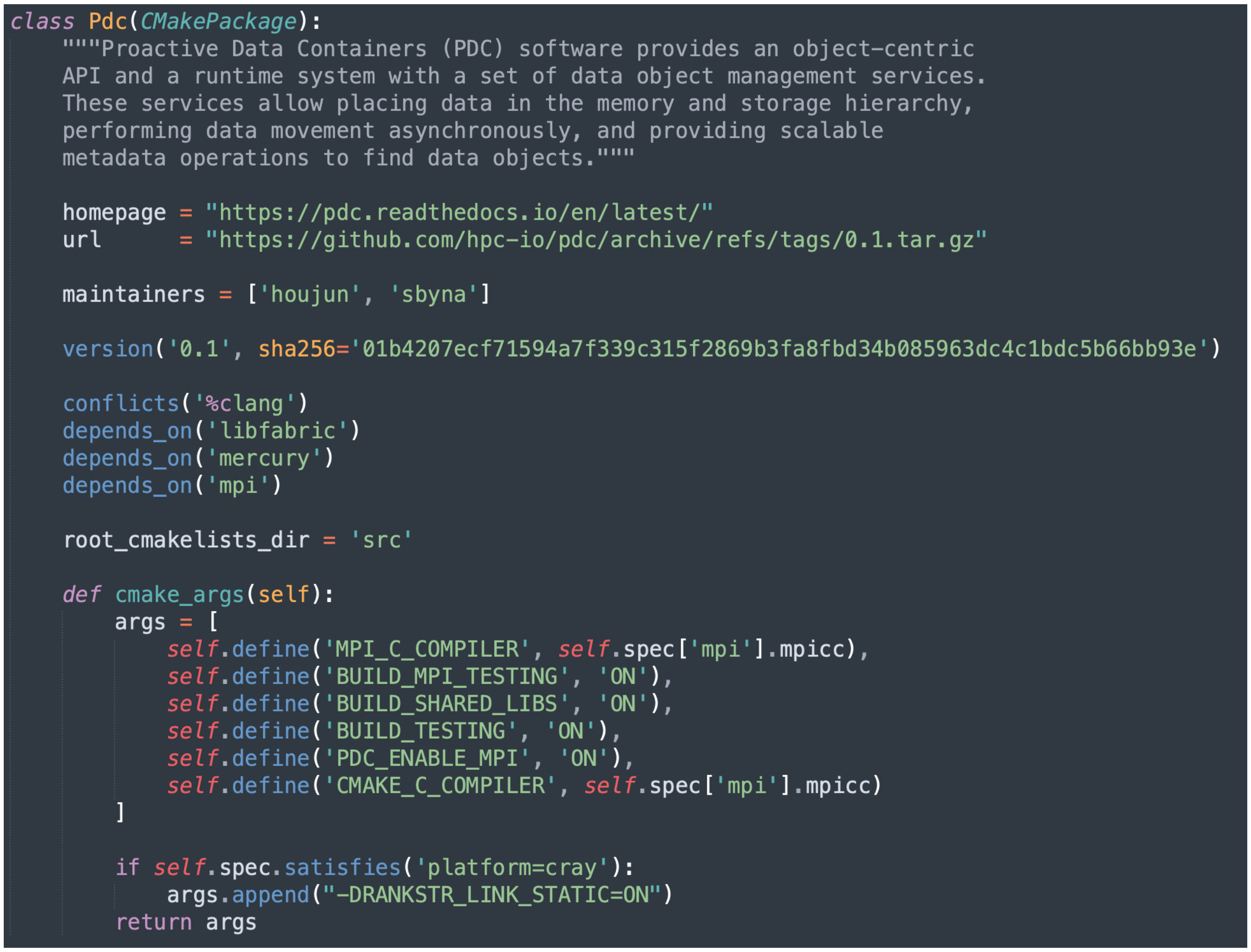
Byna, Suren, Dong, Bin, Tang, Houjun, Koziol, Quincey, Mu, Jingqing, Soumagne, Jerome, Vishwanath, Venkat, Warren, Richard, and Tessier, François. Proactive Data Containers (PDC) v0.1. Computer Software. https://github.com/hpc-io/pdc. USDOE. 11 May. 2017. Web. doi:10.11578/dc.20210325.1.

Getting Started

Figure 2: The image shows a screenshot of the Proactive Data Containers ReadtheDocs, a web hosting service for technical documentation of software projects.
The PDC documentation can be found here: https://pdc.readthedocs.io/en/latest/

Software packaging:
An additional essential is software packaging for software applications as it will ensure a stable and standard environment for users. By creating software packages, one can distribute everywhere with a simple command on your command-line. Rather than letting the user manually download and install the dependencies and the software itself, the developers can

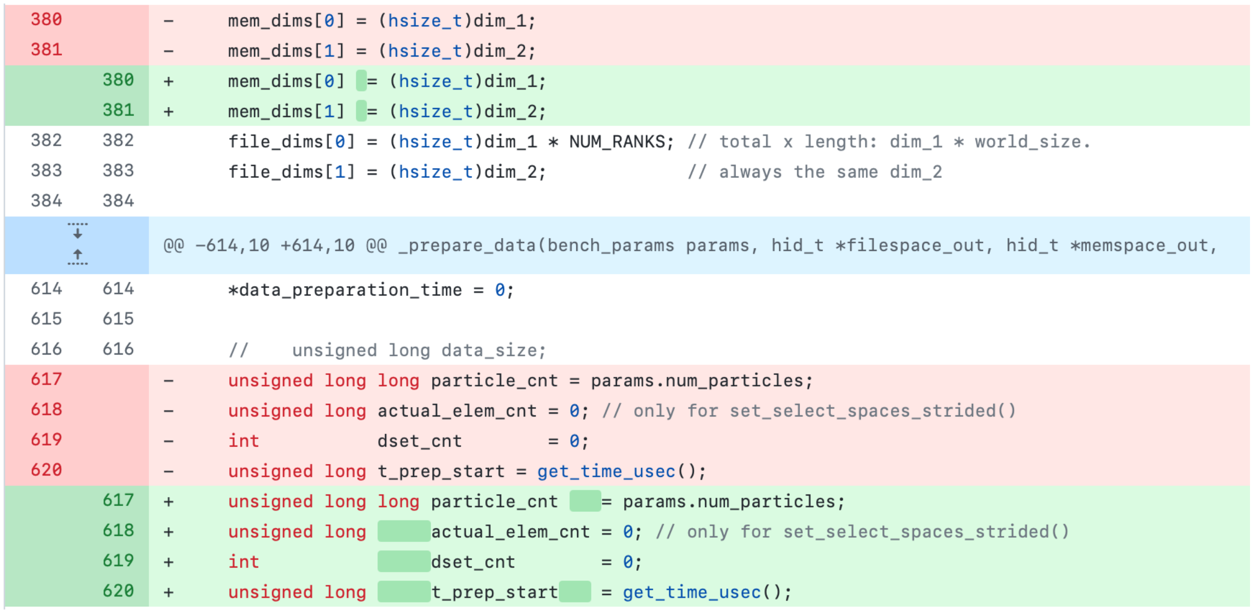
create a package for the user to install that will do all that without any disruptions or any errors along the way to use the respective software. For this component, I used Spack, a package manager for supercomputers where it makes installing scientific software easy.



```python
class Pdc(CMakePackage):
    """Proactive Data Containers (PDC) software provides an object-centric
    API and a runtime system with a set of data object management services.
    These services allow placing data in the memory and storage hierarchy,
    performing data movement asynchronously, and providing scalable
    metadata operations to find data objects."""

    homepage = "https://pdc.readthedocs.io/en/latest/"
    url      = "https://github.com/hpc-io/pdc/archive/refs/tags/0.1.tar.gz"

    maintainers = ['houjun', 'sbyna']

    version('0.1', sha256='01b4207ecf71594a7f339c315f2869b3fa8fbd34b085963dc4c1bdc5b66bb93e')

    conflicts('%clang')
    depends_on('libfabric')
    depends_on('mercury')
    depends_on('mpi')

    root_cmakelists_dir = 'src'

    def cmake_args(self):
        args = [
            self.define('MPI_C_COMPILER', self.spec['mpi'].mpicc),
            self.define('BUILD_MPI_TESTING', 'ON'),
            self.define('BUILD_SHARED_LIBS', 'ON'),
            self.define('BUILD_TESTING', 'ON'),
            self.define('PDC_ENABLE_MPI', 'ON'),
            self.define('CMAKE_C_COMPILER', self.spec['mpi'].mpicc)
        ]

        if self.spec.satisfies('platform=cray'):
            args.append("-DRANKSTR_LINK_STATIC=ON")
        return args
```

Figure 3: The image shows the Spack recipe for Proactive Data Containers. The recipe consist of a summary, where to find the homepage and url of the project, who the maintainers are, the dependencies, and the cmake arguments to build and install the project.

Coding Standard:
The last infrastructure component that I'll be contributing to these projects is the coding standard script, which is done through GitHub Actions. Therefore whenever new code is added to the project on GitHub, it will check the code first and if the coding style is not the same, it will automatically format the code in the specified coding style and commit those changes to the project.

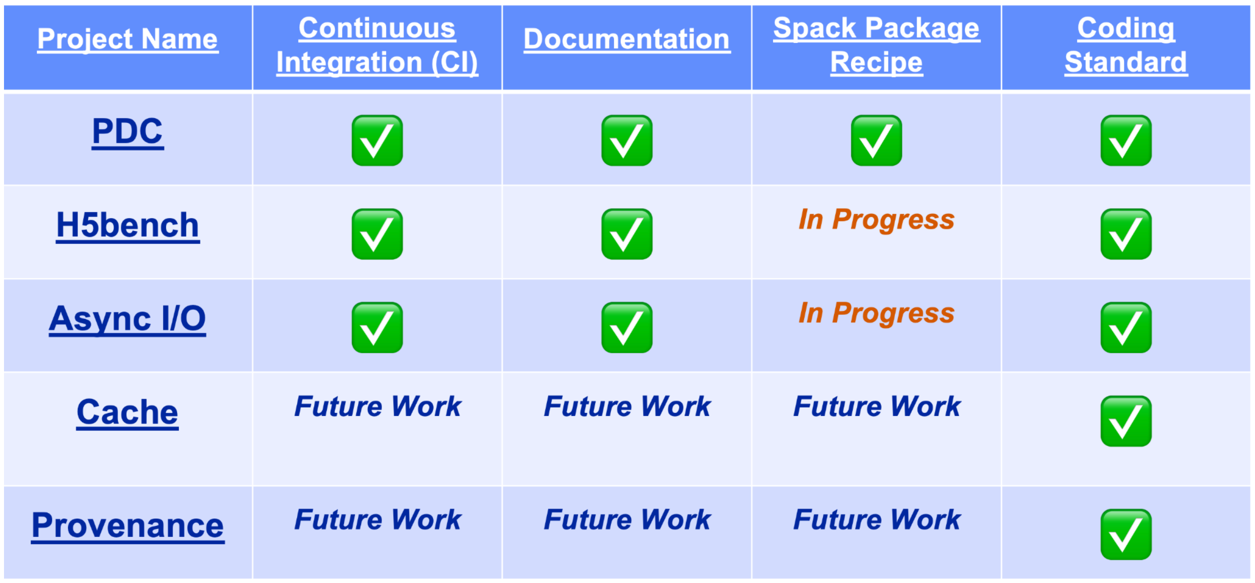


```
-     mem_dims[0] = (hsize_t)dim_1;
-     mem_dims[1] = (hsize_t)dim_2;
+     mem_dims[0]  = (hsize_t)dim_1;
+     mem_dims[1]  = (hsize_t)dim_2;
      file_dims[0] = (hsize_t)dim_1 * NUM_RANKS; // total x length: dim_1 * world_size.
      file_dims[1] = (hsize_t)dim_2;             // always the same dim_2

@@ -614,10 +614,10 @@ _prepare_data(bench_params params, hid_t *filespace_out, hid_t *memspace_out,
      *data_preparation_time = 0;

      //    unsigned long data_size;
-     unsigned long long particle_cnt = params.num_particles;
-     unsigned long actual_elem_cnt = 0; // only for set_select_spaces_strided()
-     int           dset_cnt        = 0;
-     unsigned long t_prep_start = get_time_usec();
+     unsigned long long particle_cnt    = params.num_particles;
+     unsigned long      actual_elem_cnt = 0; // only for set_select_spaces_strided()
+     int                dset_cnt        = 0;
+     unsigned long      t_prep_start    = get_time_usec();
```

Figure 4: This screenshot shows the changes that the coding standard script did after checking a push request to the project. It detected a different coding style which then the script automatically reformatted and committed those changes to the project.

Results

For my results I was to implement each infrastructure component at least once through the term. I was able to complete each of the components for the PDC project. I created a GitHub Action for building, installing, and testing the project in a Linux development environment. The documentation was generated and can be found on the respective ReadtheDocs for the project. The Spack package recipe was completed and submitted to the Spack GitHub repository. It was successfully submitted and is now available to be installed through Spack. Additionally, the coding standard script is now incorporated into the project. With each new code change, the code will be automatically checked and reformatted if needed to conform to the project's specific coding style. After PDC was completed, I started working with the VOL-Async I/O and H5bench projects. I first implemented the coding standard script and the documentation using Sphinx and hosting the information on ReadtheDocs. I was able to get feedback from my mentors and the developers from each project to successfully meet their requirements on what to put in the documentation. Next, I completed the continuous integration component for each project. Although I had some struggles, I was able to overcome them by asking my mentors for guidance on what approaches I should take to complete the tasks. I was able to complete the GitHub Actions and the projects are now automatically building, installing, and testing the software whenever there is a pull request on the master branch on GitHub. Currently I am still in the process of finishing up the Spack package recipes, with those prospectively being completed, I will be done with H5bench and Async I/O projects before my internship term ends.



Project Name	Continuous Integration (CI)	Documentation	Spack Package Recipe	Coding Standard
PDC	✓	✓	✓	✓
H5bench	✓	✓	*In Progress*	✓
Async I/O	✓	✓	*In Progress*	✓
Cache	*Future Work*	*Future Work*	*Future Work*	✓
Provenance	*Future Work*	*Future Work*	*Future Work*	✓

Figure 5: The table shown shows my current progress in the internship. I was able to fully complete all of the components for PDC and I am currently finishing up H5bench and Async I/O.

Discussion

There were numerous obstacles that I faced during this internship. First, I'll be talking about the challenges of starting a new job in a field where I have no professional experience. As a sophomore in college, I never had a professional software engineering experience under my belt. This was the first time where I met with developers and attended project meetings to talk about my progress in developing project features. After the first few weeks, it became a norm and I lost all of my anxiety of being at a new job. I went to each meeting confident and was able to show progress in my work and meet the expectations of my mentors. I took the initiative to discuss with my mentors about any concerns or problems that I had and they were able to direct me in the right direction. For the work itself in the internship, there were numerous obstacles that I had to overcome as I had no experience in the software I was using. I had no professional experiences before with software testing, continuous integration, and using Spack. To resolve these issues, I completed tutorials and read documentation on the software I was using. I learned how to use reStructuredText files and how to generate software documentation with Sphinx. I learned how to host that documentation on ReadtheDocs. I worked with my mentors on how to use GitHub Actions on automatically building tests for the projects. I did have trouble as I never used CMake and my lack of knowledge in Linux, but I was able to take the initiative and work with my mentors to catch me up to speed to complete the given tasks. With Spack, I worked with my mentors on how to resolve my debugs and how to successfully make a package recipe. With the challenges that my mentors couldn’t help me with, I was able to get in contact with the Spack developers to point me in the right direction. Although I still have issues in Spack, I am actively resolving these issues and attempting to complete my given tasks. Currently I am creating the Spack package recipes for H5bench and Async I/O. The next steps with the project, after completing those components, would be to complete the same components for Cache and Provenance projects. I was able to complete the coding standard scripts for those projects but they still require continuous integration, documentation, and the Spack package recipe. For the next intern to do this project I recommend knowing CMake, working in a Linux development environment, and the ins and outs of using Spack.

Conclusion

In conclusion, I was able to actively contribute to the projects and meet the learning objective of my internship, being able to build and develop HPC software infrastructure for projects hosted on GitHub. I produced and deployed continuous infrastructure components to several repositories in the HPC-IO organization. I validated code changes with automated testing and developed documentation for user-facing interactions. Some of my obstacles were technical proficiency with new software that I never experienced before. But my mentors were able to support and guide me in the correct direction to complete my tasks. Hopefully I am able to finish up the last component for H5bench and Async I/O. The next step for this project would be to complete the infrastructure components for Cache and Provenance. This internship has been overall an amazing experience. As this is one of the first professional experiences I have, I’m grateful to have this opportunity to first-hand experience in software engineering. I learned about the best practices in HPC computing and the essential skills required for software engineering. This has been an awarding experience which allowed me to grow as an individual. I'm excited to see how this experience and the knowledge that I acquired will shape the next chapters of my professional and academic career.

References

[1] Shahin, M., Ali Babar, M., & Zhu, L. (2017). Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. *IEEE Access*, *5*, 3909–3943. https://doi.org/10.1109/access.2017.2685629

[2] Houjun Tang, Suren Byna, Francois Tessier, Teng Wang, Bin Dong, Jingqing Mu, Quincey Koziol, Jerome Soumagne, Venkatram Vishwanath, Jialin Liu, and Richard Warren, "Toward Scalable and Asynchronous Object-centric Data Management for HPC", 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid) 2018 [Pre-print version]

[3] Houjun Tang, Quincey Koziol, Suren Byna, John Mainzer, and Tonglin Li, "Enabling Transparent Asynchronous I/O using Background Threads", PDSW 2019, in conjunction with SC19. [Pre-print version]

[4] Byna, Suren, Dong, Bin, Tang, Houjun, Koziol, Quincey, Mu, Jingqing, Soumagne, Jerome, Vishwanath, Venkat, Warren, Richard, and Tessier, François. Proactive Data Containers (PDC) v0.1. Computer Software. https://github.com/hpc-io/pdc.

[5] Li, Tonglin, Byna, Suren, Koziol, Quincey, Tang, Houjun, Bez, Jean Luca, Kang, Qiao. H5bench v0.1. Computer Software. https://github.com/hpc-io/h5bench.

[6] Byna, Suren, Koziol, Quincey, Tang, Houjun, Bez, Jean Luca. Vol-Async v0.1. Computer Software. https://github.com/hpc-io/vol-async.

[7] Zheng Huiho, Ravi John, Lee, Joe H., Koziol, Quincey. Vol-Cache v0.1. Computer Software. https://github.com/hpc-io/vol-cache.

[8] Li, Tony, Koziol, Quincey. Vol-Provenance v0.1. Computer Software.

https://github.com/hpc-io/vol-provenance .

Acknowledgements

This work was prepared in partial fulfillment of the requirements of the STEM Core Program, managed by Workforce Development & Education at Berkeley Lab. This work used resources of the National Energy Research Scientific Computing Center.

Mentorship and guidance were given by Suren Byna, Houjun Tang, and Jean Luca Bez from the Scientific Data Management Research (SDM) Group in the Data Science and Technology Department of the Computational Research Division (CRD) at Lawrence Berkeley National Laboratory.