AFRL-RD-PS-
TR-2020-0043

USER WORKFLOW MANAGER

Terrence McGuinness, Mark Potts

Redline Performance Solutions, LLC
2275 Research Blvd, Suite 500
Rockville, MD 20850

22 August 2019

Final Report

AIR FORCE RESEARCH LABORATORY
Directed Energy Directorate
3550 Aberdeen Ave SE
AIR FORCE MATERIEL COMMAND
KIRTLAND AIR FORCE BASE, NM 87117-5776

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by AFMC/PA and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RD-PS-TR-2020-0043 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

//Richard W. H. Chong//	//J. Chris Zingarelli//
RICHARD W. H. CHONG, DR-04, USAF	JOHN C. ZINGARELLI, Lt Col, USAF
Program Manager, MHPCC	Commander and Materiel Leader, AFRL Det 15

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
22-08-2019	Final Report	1 Mar 2019- 22 Aug 2019

4. TITLE AND SUBTITLE

User Workflow Manager

5a. CONTRACT NUMBER

FA9451-19-C-0006

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Terrence McGuinness, Mark Potts

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
D0EU

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

RedLine Performance Solutions, LLC
2275 Research Blvd, Suite 500
Rockville, MD 20850

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory
3550 Aberdeen Ave SE
Kirtland AFB, NM 87117-5776

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/RDSM

11. SPONSOR/MONITOR'S REPORT NUMBER(S) AFRL-RD-PS-TR-2020-0043

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

13. SUPPLEMENTARY NOTES

AFRL-2020-0417 Approved 2 Dec 2020

14. ABSTRACT

The overall objective of the project is to recommend, then validate, an end-to-end proof of concept solution for a general, domain agnostic approach towards workflow systems. Phase I entailed a set of ostensibly viable candidate solutions mapped against the requirements, which RedLine then derived a final recommendation based on sliding-scale evaluation criteria. Phase II objective (prototype testing) validated FireWorks against the trade study requirements and evaluation criteria in the MHPCC technical environment. Software discussed in the document is still a prototype system, and specific recommendations are offered to bring FireWorks into operations.

15. SUBJECT TERMS

Portal; Workflow; High Performance Computing (HPC); load balancing; data visualization

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			RICHARD W. H. CHONG
Unclassified	Unclassified	Unclassified	SAR	34	**19b. TELEPHONE NUMBER** *(include area code)*

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. 239.18

TABLE OF CONTENTS

1.0 SUMMARY

Within the Defense Supercomputing Resource Center (DSRC) Portal framework, most workflow systems are currently tightly coupled within specific domain areas, and have become inherently complex or do not readily adapt to other domain areas. The overall objective of this project is to recommend and validate an end-to-end proof-of-concept solution of a more general approach that is domain agnostic while still being a viable utility for managing workloads and job orchestration across the DSRC resources.

This project has two distinct phases:

- Phase I: Conduct a trade study to recommend a User Workflow Manager for the Maui High Performance Computing Center (MHPCC)
- Phase II: Perform prototype testing of the recommended User Workflow Manager to validate requirements and evaluation criteria in MHPCC's technical environment.

Phase I – Trade Study: There are a number of user workflow management solutions available for MHPCC implementation, but MHPCC needs a user workflow management solution that meets their specific mission and technical requirements. RedLine Performance Solutions, LLC (RedLine) worked with MHPCC to identify their specific detailed objectives, requirements, evaluation criteria, priorities, and constraints. The site survey, and subsequent discussions and documents provided by MHPCC, are the primary sources of the requirements and criteria in the trade study.

A set of ostensibly viable candidate solutions were identified and mapped against the requirements. Requirements were considered as strictly mandatory, pass/fail, and were used to narrow the field of candidate solutions to the compliant, most viable ones. An initial selection based on requirements compliancy was achieved via document review and basic viability testing to reduce the list of candidate solutions to a manageable number. There were three compliant solutions: Pegasus, Cylc, and FireWorks.

RedLine then derived a final recommendation from this smaller field of compliant candidate solutions by determining how each solution fared with MHPCC's evaluation criteria. Evaluation criteria were based on a sliding-scale (e.g., maturity, ease of use/administration, compatibility with existing solutions/environment). The final recommendation was to take FireWorks forward for prototype testing.

Phase II – Prototype Testing: The overall objective of the prototype testing was to validate FireWorks against the trade study requirements and evaluation criteria in the MHPCC technical environment. RedLine followed a structured approach to software development for the delivery of this prototype system for MHPCC. The prototyping plan was designed to validate the criteria used to select FireWorks as the solution best suited for MHPCC as determined by an exhaustive

trade study. Unit tests were designed and built to establish basic functionality of the system within MHPCC, as well as demonstrate the ability to perform a number of desired capabilities (e.g., load balancing, reproducibility, data provenance, parallel execution). During the development of the unit tests, a design decision was made to abstract most of the specifics of both FireWorks and high-performance computing (HPC) from the user, which resulted in developing a series of helper functions. Use cases were subsequently developed to demonstrate the simplicity of setting up multiple common tasks such as using Singularity containers, staging and operating on external data, running parameter scans, and running minimization/optimization scenarios using Python as the driver and FireWorks to coordinate and stage the computations behind the scenes.

Protype testing validated that FireWorks is the best fit as a User Workflow Manager for MHPCC and potentially other DSRCs. The sample code provided with this software will provide a solid foundation upon which future MHPCC users can build and expand their own workflows for use within the MHPCC and eventually across all the DSRCs.

Given that the software discussed herein is still a prototype system, there are a number of hurdles that must be overcome to bring FireWorks into operations. Specific recommendations are offered within this document. Overall, FireWorks provides the full functionality of a mature workflow solution for advanced users while also offering the ability to design workflows that require little to no knowledge of the system itself, which makes it a highly flexible and attractive solution for the varied MHPCC user community.

2.0 INTRODUCTION

Within the Defense Supercomputing Resource Center (DSRC) Portal framework, most workflow systems are currently tightly coupled within specific domain areas, and have become inherently complex or do not readily adapt to other domain areas. The overall objective of this project is to recommend and validate an end-to-end proof-of-concept solution of a more general approach that is domain agnostic while still being a viable utility for managing workloads and job orchestration across the DSRC resources.

RedLine Performance Solutions, LLC (RedLine) has developed this Contract Line Item 2, Data Item A005 – Scientific and Technical Report as the means to:

- Recommend a Maui High Performance Computing Center (MHPCC) User Workflow solution for prototype testing
- Summarize the prototype testing of the recommended User Workflow Manager that has validated the solution selection, and positions MHPCC for continued and expanded use of the User Workflow Manager solution.

This project has two distinct phases:

- Phase I: Conduct a trade study to recommend a User Workflow Manager for MHPCC
- Phase II: Perform prototype testing of the recommended User Workflow Manager to validate requirements and evaluation criteria in MHPCC's technical environment.

There is a technical and time sequence to the two phases (i.e., a solution had to be selected prior to initiating prototype testing), so this report is structured to reflect the two distinctly separate phases of the project.

The recommended proof-of-concept user workflow solution is expected to be a client-based, meta workflow management system that leverages existing systems and capabilities while also improving the ease-of-use and time-to-solution for users of the MHPCC.

3.0 PHASE I: TRADE STUDY

The first phase of this project is to apply an operationally proven trade study process to identify viable User Workflow Managers in the context of MHPCC, eventually leading to a recommendation for the best available solution. The second phase, Prototype Testing, is discussed in section 4 below.

3.1 Trade Study - Methods, Assumptions, and Procedures

This section presents an overview of the trade study evaluation approach, the constraints that realistically exist in the MHPCC environment, and the available candidate solutions.

3.1.1 Evaluation Approach.

There are a number of user workflow management solutions available for MHPCC implementation, but MHPCC needs a user workflow management solution that meets their specific mission and technical requirements. RedLine's technical approach was developed with these objectives in mind. The first step was to assess the current environment, including mission objectives, technical aspects, and business constraints. Toward that end, RedLine was at the customer location in Maui, HI for a week conducting a site survey. RedLine worked with MHPCC to identify their specific detailed objectives, requirements, evaluation criteria, priorities, and constraints. The site survey, and subsequent discussions and documents provided by the customer, are the primary sources of the requirements and criteria in this trade study. This approach ensures the selection of a user workflow management solution is objectively based on the needs of the MHPCC environment.

A set of ostensibly viable candidate solutions was identified and mapped against the requirements. Requirements were applied as strictly mandatory, pass/fail, and used to narrow the field of candidate solutions to the compliant, most viable ones. An initial selection based on requirements compliancy was achieved via document review to reduce the list of candidate solutions to a manageable number. Basic viability testing was conducted in the MHPCC environment to further down-select potential solutions based on the ability to meet basic requirements that could not be pre-determined via a document review. RedLine provided this preliminary analysis to MHPCC for discussion and interim approval.

RedLine then derived a final recommendation from the smaller field of compliant candidate solutions by determining how each solution fares with MHPCC's evaluation criteria. Evaluation criteria were based on a sliding-scale (e.g., maturity, ease of use/administration, compatibility with existing solutions/environment). Evaluation has been qualitative, not quantitative, since assigning weights to criteria is itself subjective.

As part of the evaluation criteria, it was essential to identify several use cases of particular interest to the MHPCC. This enabled RedLine to establish a baseline end-to-end capability that was inherently required for any generalized workflow. Examples of components within these use cases included aspects such as data object management with necessary constructs for establishing provenance and persistence of the workflow itself, and its results for collaborative usability. Use cases of interest to MHPCC include:

- Simple workflow for digital engineering (DE) clients involving data staging and processing, with results sent to mass storage and/or on for final analysis
- Simple parameter scan involving 1-5 parameters
- Optimization/minimization over 1-5 parameters
- Sample use case involving conditional branch/merge capability.

The end product of this task is a user workflow management solution objectively selected on MHPCC's needs and constraints to take forward for prototype testing.

3.1.2 Constraints.

The following items are realistic circumstances and limitations in which the recommended user workflow solution must function. They can be the source of requirements and evaluation criteria, but must be accounted for in the evaluation of candidate solutions.

- There is currently no generic way for a *persistent service to generate new credentials* without a user in the loop. DSRC security requires a two-factor authentication scheme to achieve this, and it is possible to obtain approval for a new process, but only on a per-case basis. Jack Harris from Mind Modeling/Clarity, has a solution that depends on a limited one-time token, but this has not been fully demonstrated or tested, and is not yet cleared by security.

- The High Performance Computing (HPC) Portal Application uses the Tomcat server on the MHPCC Portal Appliance. There is a shared file system, but communication via file is slow.

- There is no way to connect from the portal appliance on to Hokulea (Power8 system in use at MHPCC) without a yubikey or Common Access Card (CAC). However, a virtual desktop with an xterm on Hokulea can be launched via the portal.

- Potential users of the system will be Department of Defense (DoD) personnel who have access ONLY to a Windows desktop computer that cannot be modified. All solutions must therefore be available through a web browser. This should be possible using the web portal with a virtual desktop to Hokulea.

- There is currently no DoD-wide portal, though that is something that may eventually become available. As a result, each portal is isolated inside its own enclave/DSRC.

- There is no DoD-wide file system that would allow users to move/access files across DSRCs.

- The Center-Wide File System (CWFS) exists at each DSRC, but it is not accessible from the supercomputer compute nodes. As a result, using the CWFS to stage data must be done from the transfer queue (separately from any worker processes).

3.1.3 Candidates.

Based on industry research and information gleaned to date, the following were the candidate user workflow management solutions considered:

3.1.3.1 Current Workflow Systems and Projects at the DSRCs.

- Galaxy
- Clarity Bioanalytics/MindModeling
- Resonate/Girder
- FireWorks.

3.1.3.2 Other Open Source Candidates.

- Arvados: https://arvados.org/
- NextFlow: https://www.nextflow.io/tags/workflow.html
- Cuneiform: https://cuneiform-lang.org/
- JMS: https://github.com/RUBi-ZA/JMS
- Workspace: https://desktop.arcgis.com
- Swift/T/EMEWS: https://emews.github.io/
- Cylc: https://cylc.github.io/
- ecFlow: https://software.ecmwf.int/wiki/display/ECFLOW/ecflow+home
- Rocoto: https://github.com/christopherwharrop/rocoto/wiki/documentation
- LAW: https://law.readthedocs.io/en/latest/
- Pegasus: https://pegasus.isi.edu/documentation/
- SAW: https://www.sandia.gov/saw
- Galaxy: https://galaxyproject.org/
- Jupyter: http://jupyter.org/
- Kepler: https://kepler-project.org/
- noWorkflow: https://github.com/gems-uff/noworkflow
- Sumatra: https://pythonhosted.org/Sumatra/
- Taverna: http://www.taverna.org.uk/
- VisTrails: http://www.vistrails.org/

3.2 Trade Study – Results and Discussion

This section defines MHPCC's mandatory requirements and evaluation criteria. The initial down-select of candidate solutions via requirements compliancy is also discussed.

3.2.1 Requirements.

This section identifies the requirements for the user workflow solution. Requirements are mandatory (pass/fail) and are used to narrow the field of candidate solutions to the compliant, most viable ones.

3.2.1.1 Functionality Requirements.

- **FR.1:** The solution shall provide a tool for users to build processing workflows that access data from data stores (e.g., RDBMS, object stores, file stores), run HPC compute processing tasks, and return results to some report and/or an object-based data collection system.

- **FR.2:** The solution shall provide users and projects the ability to repeatedly connect multiple processing steps (in series and parallel) to achieve a single result.

- **FR.3:** Results (other than heavy data) shall be storable in a database that can be accessed and shared with other users. The information should include performance metrics and be self-descriptive for reproducibility. This may be a stand-alone database, but it may also be part of a solution. Pointers to heavy data shall also be stored in the database.

- **FR.4:** The solution shall be able to dispatch, manage and control multiple HPC jobs (processing tasks).

- **FR.5:** The solution shall allow both branch and merge capability.

- **FR.6:** The solution shall have fault tolerance, specifically the ability to detect tasks that fail to complete, and then restart them. The solution shall also have fail-over capability with current state of system being saved for restart from point of failure.

- **FR.7:** The solution shall interact and dispatch jobs with PBS Pro and SLURM batch scheduling systems.

- **FR.8:** The solution shall be usable for users who cannot install software on their desktop/laptop. The solution shall work in an X-windows environment or in a container.

- **FR.9:** The solution shall not depend on any client software (e.g., Java or Python).

- **FR.10:** There shall be a no-cost, viable source for ongoing product maintenance (e.g., patches) and customer support. Licensing shall be at no cost or unreasonable restriction, nor shall there be any ITAR restrictions.

- **FR.11:** The solution shall be mature, with at least two sites using it in production. Beta software is not acceptable. Likewise, the solution shall be currently evolving, with ongoing development.

3.2.1.2 Usability Requirements.

- **UR.1:** The solution shall be available through the MHPCC portal, but also should provide as an easily accessible front-end that can be downloaded and installed on a desktop or laptop computer.

- **UR.2:** The solution shall allow users to easily and quickly build, maintain, run and reuse a sequence/collection of processing tasks. The product shall be easy to learn, use, and understand.

- **UR.3:** The solution shall provide a graphic representation of the processing status to the user owning the jobs. There shall be an easy-to-understand dashboard interface to monitor the status of the workflow.

- **UR.4:** The solution shall be useable without advanced knowledge of HPC resources and scripting. (i.e., can be generally configured without requiring user to have knowledge of job scheduling tools and file system layouts).

- **UR.5:** There shall be a clear API that allows users/developers to interact with the workflow elements and its corresponding data.

3.2.2 Compliancy Assessment.

Compliancy with the mandatory requirements was established through documentation review and basic testing. Figure 4-1 summarizes the compliancy of the candidates. Based on this analysis, the compliant candidates are Pegasus, Cylc, and FireWorks.

To save time and effort, research typically ceased once RedLine established non-compliancy. Cells in the matrix noted with a “U” indicates “Unknown”. These are situations in which non-

compliancy was established for that candidate prior to determining if the candidate met that particular requirement.

3.2.2.1 Requirements Compliancy Matrix.

	Requirements															
	FR. 1	FR. 2	FR. 3	FR. 4	FR. 5	FR. 6	FR. 7	FR. 8	FR. 9	FR. 10	FR. 11	UR. 1	UR. 2	UR. 3	UR. 4	UR. 5
Candidates																
Pegasus	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FireWorks	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Cylc	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Arvados	Y	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
Galaxy	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	Y	Y	Y
Galaxy Project	Y	Y	Y	Y	Y	Y	Y	U	U	Y	Y	Y	N	Y	Y	Y
NextFlow	Y	Y	Y	Y	Y	Y	Y	U	U	Y	Y	Y	Y	N	N	Y
JMS	Y	Y	N	Y	Y	U	Y	Y	U	N	U	U	Y	Y	Y	Y
ecFlow	N	Y	N	Y	N	Y	N	Y	Y	Y	Y	N	N	Y	Y	Y
Rocoto	N	Y	N	Y	N	Y	Y	Y	Y	Y	Y	N	Y	N	Y	Y
SAW	Y	Y	Y	Y	Y	U	Y	N	Y	Y	Y	Y	Y	Y	Y	Y
Kepler	Y	Y	Y	Y	Y	N	Y	U	U	Y	Y	Y	N	Y	Y	Y
Taverna	Y	Y	Y	N	Y	Y	N	Y	N	Y	Y	Y	N	N	Y	Y
MindModeling	Y	U	U	U	U	U	U	U	U	U	U	U	N	U	N	U
Cuneiform	U	U	U	U	U	U	U	U	U	U	U	U	N	U	U	U
Swift/T/ EMEWS	U	U	U	U	U	U	U	U	U	U	U	U	N	U	U	U
Workspace	U	U	U	U	U	U	U	N	U	U	U	U	U	U	U	U
Resonate/ Girder	Y	U	U	U	U	U	U	U	U	U	N	U	U	U	U	U
LAW	U	U	U	U	U	U	U	U	U	U	N	U	U	U	U	U
noWorkflow	U	U	U	U	U	U	U	U	U	U	N	U	U	U	U	U
Sumatra	U	U	U	U	U	U	U	U	U	U	N	U	U	U	U	U
VisTrails	U	U	U	U	U	U	U	U	U	U	N	U	U	U	U	U

Figure 3-1 Requirements Compliancy Matrix

3.2.2.2 Compliancy Notes of Interest.

- Jupyter is integrated with Pegasus, so it was not separately evaluated.
- Cylc was evaluated with the Rose plug-in present.
- Running CWF workflows with Arvados requires setting up "Arvados Clusters", which will not work in a general HPC environment.
- Getting Galaxy Project to operate in a production environment requires a number of non-trivial efforts. It is highly integrated for bio-informatics and would be challenging to be used as a generic solution.
- NextFlow has no apparent built-in dashboard capability, so it requires fairly advanced scripting skills. All components seem to be written in groovy (Java-based).
- SAW is available only to Sandia employees and to Sandia partners through a government-use agreement and is under ITAR restrictions.
- MindModeling, Resonate/Girder, and Galaxy do not meet all the requirements, but have potential as subsystems in the recommended solution.
- The following candidates failed the product maturity requirement:
  - LAW – current release is Alpha
  - noWorkflow – no updates in over a year
  - Sumatra – last update was in 2015
  - VisTrails – last update was in 2016.

3.2.3 Evaluation Criteria.

Candidate workflow solutions meeting the mandatory requirements are mapped against the evaluation criteria defined in this section. Evaluation criteria are sliding scale, not pass/fail. This approach allows for the relevant advantages/disadvantages of compliant solutions to be considered to derive a recommendation for the best solution for MHPCC.

3.2.31 Evaluation Criteria Definitions.

- **EC.1 – Persistence:** The data store of a workflow instance should be self-described for reproducibility.

- **EC.2 – Debugging**: The user should be able to set up and debug a workflow on local desktop with appropriate stubs, if necessary, before actually running all the processes on an HPC system. The process of staging any workflow solution to the MHPCC should also be as simple as possible.

- **EC.3 – Data Accessibility:** There should be direct links (pointers) to both inputs and outputs from each element of the workflow, as well as any heavy data placed into mass storage or elsewhere.

- **EC.4 – Data Visualization:** There should be built-in options or easily adaptable APIs for visualization of results from workflow elements and interim/final results.

- **EC.5 – Load Balancing:** The solution should allow for load balancing, movement, and distribution of tasks across less busy HPC systems.

- **EC.6 – Reporting System:** The solution should provide periodic (e.g., user specified daily, hourly) notification of processing status to user/owner within the security constraints of the customer environment.

- **EC.7 – Performance Statistics:** The solution should be able to collect and visualize workflow system usage and performance statistics.

- **EC.8 – Authentication:** The solution should work with any DoD authentication framework (e.g., UIT+, OSS, Kerberos, Yubikey).

- **EC.9 – Ease of Use:** There are numerous user workflow solutions at the DSRCs. A consistent obstacle for some of them is their effective use requires significant HPC

knowledge and skills. A primary goal for this user workflow solution is to introduce a capability that is full-featured and as easy as possible for users to create use cases. Requirements in this area have been imposed to limit the field to those solutions that are relatively easy to use, but due to the criticality of this aspect, this criterion will help identify those solutions excelling in this area.

- **EC.10 – Ease of Administration**: The user workflow solution should be as easy as possible to administer. Evaluation in this area includes solution features; training required, recommended, and offered; and whether expertise with the candidate already exists in the customer arena.

- **EC.11 – Scalability:** The solution should have the ability to schedule jobs across multiple resources within a single DSRC. A factor to consider is how viable and difficult it would be to extend that capability to multiple DSRCs.

- **EC.12 – Support:** There is a requirement for solution maintenance and support. This criterion measures the quality and potential costs of the offered support.

- **EC.13 – Repeatability:** A key component for unifying workflows as a service is establishing data permanence and long-term repeatability by holding results through an object-based metadata approach and housing this information via established self-describing standards. This establishes easy access to experiments workflow provenance and can create a multi-disciplinary workflow perspective.

- **EC.14 – Machine Learning:** The ability to use machine learning (vs. more traditional techniques) to drive workflow system (optimization/minimization) is a desired feature.

- **EC.15 – Encapsulation:** The ability to encapsulate parts of another established workflow tool or to run another tool from within. One example of this would be encapsulating a Galaxy optimization solution that utilizes DAKOTA as part of a larger use case.

- **EC.16 – Support Long-running Workflows:** The ability to manage workflows that take more than a month to complete is a desired feature. This would likely be a requirement were it not for security control constraints.

- **EC.17 – Command Line Support:** All GUI-driven action should also have command-line/scripting interfaces. This would ease/simplify certain user actions. This capability allows for rapid prototyping and for jobs and services to be chained together to automate the system as a whole.

- **EC.18 – Container Support:** The system should be able to establish and orchestrate tasks as containers (e.g., Docker, Singularity). This capability will allow components, including user compiled, MPI-based applications, to be run seamlessly on different platforms within the DSRCs where they otherwise would need to be compiled on every different platform.

- **EC.19 – Cloud support:** The solution should have the capability to interface with external cloud services. An increasing number of cloud services are becoming available, and it would be useful to be able to leverage those using the chosen solution.

3.3 Trade Study – Conclusions

The three compliant solutions (Pegasus, Cylc, and FireWorks) were assessed for how well they met the evaluation criteria. This analysis includes two general components:

- Achieving basic functionality in the MHPCC environment – Basic functionality is defined as being able to successfully submit a job from the portal appliance to Hokulea.

- Assessing how well each candidate meets the evaluation criteria. This was determined to the extent possible with empirical experience. When testing was not feasible, documentation review was used.

Figure 3-2 defines the colors used in Figure 3-3, which summarizes the evaluation criteria analysis.

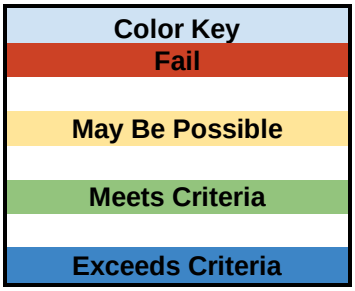

Figure 3-2 Color Key for Evaluation Criteria Assessment

Evaluation Criteria	Pegasus	Cylc	FireWorks
EC.1: Persistence	.Dax files meet this criterion.	Cylc suites can be transferred from user to user and Rose allows workflows (including metadata) to be saved and shared.	FireWorks stores job instances (a specific run of the workflow) automatically in a MongoDB database, guaranteeing its persistence.
EC.2: Debugging	Installs and runs on Mac, Windows, and Linux.	GUI's "dummy" mode runs through a workflow without executing long running jobs. Can be run locally or on host.	FireWorks holds its workflow state and job related JSON data objects in a MongoDB. With local MongoDB instance, can locally develop and "test" FireWorks Python scripts.
EC.3: Data Accessibility	Dashboard capability has links to input and output files, as well as other metadata.	Job logs are available through Cylc-review. Data accessibility available via Rose.	Workflow data objects have their data associations "stored", managed, shared, and queried as objects in that database via descriptors in the respective YAML files associated to the respective controlled jobs.
EC.4: Data Visualization	Dashboard provides this capability.	Cylc-7 does not seem to offer this, but new version will have a better API for visualization of results via web browser.	A FireWorks extension for VASP provides data sharing of results. It sits on top of FireWorks objects and shares the results with other Workflow Job instances through Pymatgen services. Could possibly be generalized.
EC.5: Load Balancing	Load balancing is enabled.	The command 'rose host-select' allows a Rose suite to be configured to submit jobs to different hosts. Can be set up to do load balancing at the task level.	No specific load balancing mechanism, but the 'qlaunch' command can issue workflows and job instances to a list of hosts over multiple configurations using schedule adapters.
EC.6: Reporting System	Updates available via dashboard. Notifications can be sent via email or jabber/Gtalk.	This is configurable and the web interface for running jobs is well constructed.	The built-in web interface lacks dependency information and does not directly link with the schedulers, but third party MongoDB tools and schedule viewers could provide a better dashboard interface.

Evaluation Criteria	Pegasus	Cylc	FireWorks
EC.7: Performance Statistics	Available feature	Not sure if this is built in, but the data could easily be collected. All job statistics are saved in the same location.	FireWorks JSON job objects in the MongoDB service encapsulate many performance metrics. It should be possible to formulate queries across thousands of past executions. Further automation using data mining techniques has been developed under a system called Custodian.
EC.8: Authentication	Pegasus (via HTcondor) can make an initial connection, but cannot establish a return and is unable to connect to the PBS server.	Uses an existing Kerberos tickets via ssh socket connections to communicate between hosts and manager. Testing has shown that the return connection is not made when using a socket to launch jobs. This may be fixable, but requiring a valid Kerberos ticket to launch is not unreasonable. Once a UIT+ system is in place, ssh should work with that as well.	Authentication is achieved via ssh and works with either a valid kerberos ticket or a socket connection created from a valid kerberos ticket. Connections to the MongoDB are made via port connections on the host running the job.
EC.9: Ease of Use	At this juncture, Pegasus is non-functional in the DSRC environment. Unable even to submit a job from a local host.	In stand-alone mode, Cylc is easy to both set up and run. Rose has been more difficult to set up and configure, but ultimately would be installed as a service on the appliance if it is the selected solution. Rose, Rosie, and the Subversion server can all be hosted in a VM. This would simplify setup, but it is unclear if that VM could run on the portal.	FireWorks is Python-based, so it is easy to use if a user is familiar with python. GUI support is somewhat limited, but incorporation with Jupyter Notebooks is an advantage.
EC.10: Ease of Administration	Significant problem. Cannot make job control manager connect through Kerberos. Process of setting up an HTcondor server is too complicated to make this a viable solution.	Cylc does not have much overhead for administration. Rose seems to be more complicated and requires a Subversion server and web access that will be more complicated to set up. Once in place, it should not be too difficult to keep running. There is a pre-built VM that is available and provides everything pre-installed.	The core of FireWorks WfLS is Python-based and the install is straightforward using pip. The main challenge is standing up a MongoDB for either personal use or for multiple users, but once running, these services stay up indefinitely.

Evaluation Criteria	Pegasus	Cylc	FireWorks
EC.11: Scalability	Cannot connect to either Poi or Hokulea.	Adding and submitting to various hosts is supported. Successfully submitted jobs to both Hokulea and Poi from the portal appliance.	FireWorks can connect to multiple hosts, which facilitates scaling. The main challenge will be allowing connections to a single MongoDB instance to monitor the status of the jobs.
EC.12: Support	Pegasus has 15 years of funding and development from a variety of sources including DOE, DARPA, NSF. Host online "office hours" twice a month. Unfortunately, no response to requests for assistance in site configuration.	There is active development on these projects out of the Met office in the UK. They also responded promptly when asked for information regarding the configuration of Rose.	FireWorks is mainly used and developed by the Material Science community and is funded at ARFL, NERSC, with some activity at Oak Ridge, Argonne, and SDSC.
EC.13: Repeatability	Workflow description and metadata are saved in a replica catalog that can be either file based or a RDBMS.	When running with Rose, the workflow configuration and associated files are saved in a Subversion repository and can be shared or checked out to run again at a later date.	A PETTT special project has developed a "facade" that inherits the FireWorks Python library, thus having data inputs as Python objects via the MongoDB data store. This can be generalized and, with development, combined with the Kitware Girder system that also is established via a shared MongoDB instance.
EC.14: Machine Learning	It should be possible to use machine learning to drive a workflow, though it is not feasible to test this in the short term.	It should be possible to use machine learning to drive a workflow, though this will be difficult to test in the short term.	FireWorks is one of the few systems that is inherently well suited for dynamic control of workflows by allowing the system to adapt from results. Workflows can be developed directly in Python with access to data objects per task along with its built-in functions that can modify the workflow specifications.
EC.15: Encapsulation	Pegasus supports Open Science Grid workflow representations which would expose a common interface to other workflow representations.	If the other workflow can be started with a batch script and inputs and outputs are defined, this should be possible.	Runtime encapsulation of a workflow would be easily achieved if a single script, along with its inputs and outputs are defined and added to the controlling database via 'lpad add'.

Evaluation Criteria	Pegasus	Cylc	FireWorks
EC.16: Support Long-running Workflows	Support is available.	Cylc is set up to handle this kind of scenario, but there may be problems keeping the Kerberos connections open.	The data store task management system is conducive for this type of process control. The state of the workflow (provenance included) is stored in MongoDB.
EC.17: Command Line Support	Available feature	Available feature	Available feature
EC.18: Container Support	Available feature	Available feature	Available feature
EC.19: Cloud Support	Cloud support is available, though it is unclear if it would be reachable from the portal appliance.	Cloud support is available, though it is unclear if it would be reachable from the portal appliance.	Cloud support is available, though it is unclear if it would be reachable from the portal appliance.

Figure 3-3 Evaluation Criteria Summary Analysis

3.4 Trade Study – Recommendations

As indicated in Figure 3-3, the final ranking of the candidate solutions is FireWorks, Cylc, and finally Pegasus. FireWorks is the recommended solution. Critical findings and recommendations for each are summarized in this section.

Pegasus (Rank #3) – Pegasus is a comprehensive and established Scientific Workflow Management System that includes key elements such as provenance, a dataflow subsystem, and a consistent management framework. It best met and/or exceeded the mandatory requirements, but failed to achieve basic functionality in the MHPCC environment. Specifically, the Condor/Bosco subsystem used to authenticate and connect to hosts, as well as interact with the PBS scheduler, could not readily be adapted to the Kerberized High Performance Computing Modernization Program (HPCMP) environment. It is probably possible to adapt the system to work within the HPCMP environment, but despite best efforts and reaching out to the support community multiple times, the path forward was too uncertain to continue with this analysis. Given Pegasus' mature suite of features and capabilities, and its use within other large environments, it may be worth re-assessing Pegasus at a future date.

Cylc (Rank #2) – Cylc proved to be the simplest system to set up within the MHPCC environment, and can successfully launch and monitor jobs on multiple systems. The dependency graphing and cycle-based approach to managing workflows is intuitive and provides an excellent interface for both debugging and monitoring active workflows. Its fully featured

GUI includes a graphical representation of the workflow itself. A disadvantage of being cycle-based with controls that manage dependences between specific time intervals is that it may be better suited for workflow types other than those typically used by MHPCC (e.g., weather services). By itself, however, Cylc is not a fully integrated workflow management system. It depends on Rose as an add-on toolkit to write, edit, run, and store application configurations. While Cylc is entirely capable of standing on its own for many workflow applications, the complications of using Rose for more diverse workflows did not seem to be as intuitive or as streamlined as either Pegasus or FireWorks. In addition, Rose leverages a Subversion (SVN) repository to handle provenance and collaboration, which adds an extra layer of complexity to the administration of the system. Due to Cylc's inherent simplicity and intuitive structure, it may well be the preferred candidate for some use cases and certainly bears further consideration in the future.

FireWorks (Rank #1) – FireWorks is similar to Pegasus in that it has many features of a fully integrated workflow management system while still being a light-weight workflow manager. It is Python-based, which makes installation extremely simple, and also allows for potential integration with the full Python suite of applications. This includes Jupyter notebooks, which has become popular among scientific programmers and analysts in recent years. Although there were initially some problems handling the ssh connections between the portal and both Hokulea and Poi, some minor modification of the code allowed jobs to be launched and monitored successfully on both systems.

While the dashboard interface of FireWorks is not as robust as either Pegasus or Cylc, there are third party applications that improve this experience. FireWorks relies on a MongoDB instance to manage and store workflow states and data, which adds a layer of complexity. However, this introduces advantageous features:

- FireWorks tracks and stores all jobs instances through a database with configurable descriptors that can be supported and utilized directly by the research developers themselves.
- The framework intrinsically has placeholders within the stored job descriptors for persistent performance-based provenance.
- Because FireWorks jobs are held as objects in Python along with its associated data, it allows the development of adaptive techniques using machine learning that can dynamically drive the workflow.

During testing, it was possible to set up an instance of MongoDB in user space, but it might be preferable to host a server on the portal appliance that would be available to all users.

Another key advantage to FireWorks is the potential to leverage work done previously on a PETTT pre-planned effort to wrap VASP within a FireWorks façade. Similarly, the work being

done by KitWare to develop the Girder framework for HPCMP may also integrate well with the FireWorks system.

Due to the fully featured nature of FireWorks, along with the key advantages of being Python-based and its ability to leverage past and ongoing work within the HPCMP, FireWorks is recommended as the best option to prototype for this project. It is the surest and most diverse option to lower barriers to HPC for users of the MHPCC systems.

4.0 PHASE II: PROTOTYPE TESTING

The second phase of the User Workflow Manager project was to perform prototype testing with FireWorks, the recommended workflow manager solution.

4.1 Prototype Testing – Methods, Assumptions, and Procedures

After selection of FireWorks as the prototype solution, both development and testing proceeded in tandem according to a general plan as follows:

- Establish a unit testing framework based on pytest.

- Design unit tests that validate as many trade study requirements and evaluation criteria as possible:
    - Verify MongoDB connectivity
    - Verify connectivity to Poi and Hokulea
    - Verify ability to launch to PBS queues
    - Verify ability to relaunch after failure
    - Verify ability to run multiple jobs in parallel
    - Veriy ability to reproduce a previous workflow
    - Verify ability to run on multiple remotes
    - Verify ability to specify dependencies
    - Verify ability to load balance long running jobs.

- Build the basis for the unit tests—an iterative estimator for pi
    - Integrate with MongoDB
    - Integrate with FireWorks framework
    - Integrate into pytest.

- Design and build use cases that demonstrate the trade study evaluation criteria and stated desires of MHPCC for a workflow solution:
    - Simple data transfer template
    - Simple parameter scan leveraging pythonic Pscan module
    - Simple minimization leveraging scipy.optimize.minimize capability
    - Simple Workflow utilizing Singularity Containers
    - Sample Complex workflow.

- Build simple tools/templates that abstract the complexity of both FireWorks and HPC from users who are unfamiliar with one or both

- Integrate common tools like Jupyter Notebooks and Robo-3T to demonstrate extensibility.

4.1.1 FireWorks Functionality.

An overview of the FireWorks user workflow manager is presented in this section.

The FireWorks workflow design includes these basic components:

- FireTasks – individual tasks to be computed
- FireWork – collection of tasks with specified data and metadata
- Workflow – interlocking set of Fireworks set to run on one or more platforms.

FireWorks operates in the following ways:

- Fireworks are "launched" one at a time or as a group
  - Locally (rlaunch)
  - Remotely (rlaunch via ssh/fabric)
  - Via a queue (qlaunch locally/ssh)
- Launches are grouped by Launchpad, which specifies the MongoDB name for workflow, provides user verification, etc.
- Qlaunches have a queue adapter that specifies queue details (time, nodes, etc.)

Users interact with FireWorks via the command line or a Python-based interface. The command line interface includes:

- lpad (add fireworks/workflows to the system)
- rlaunch (local launch – no queue)
- qlaunch (local launch via PBS/Slurm)

The Python-based interfaces to lpad, rlaunch and qlaunch are serialized functions that can be passed between hosts and include the following:

- Firetaskbase class
- Input requires a firework specification be defined
- FWAction allows for outputs to be defined and returned to the MongoDB.

4.1.2 Test Methodology.

A key component of the test methodology was the development and incorporation of unit testing designed to validate as many as possible of the evaluation criteria and requirements from the trade study. RedLine mapped the quantifiable/testable requirements and evaluation criteria to five primary unit tests, which were subsequently developed and refined. Some evaluation criteria

were validated by use cases instead of unit tests. The unit tests are described in more detail in section 4.2.1.

4.1.3 Test Environment.

Since FireWorks is Python-based software, RedLine's approach to unit testing leveraged the pytest and unittest frameworks built into Python itself. The tests all incorporate an iterative estimator for the value of pi.

At its base level, the calculator uses random numbers to "throw darts" at a unit square and estimate the value of pi based on the number of darts that lie within the circle as compared to the total number of darts thrown. The calculation is not very efficient, but it provides an opportunity to store results in an external database. This can be compared to results reported to the workflow, which is how the simple_pi test runs, and tests the ability to store data from a workflow.

The calculator can be invoked with a number of different options, which allows it to be used in various ways by the different unit tests. For example, for the load balancing unit test, the calculator is directed to run slowly (sleep for 20 seconds) when it is running on Hokulea so that the turnaround time there will be slower than on Poi and more work will be sent to Poi as a result.

Similarly, the calculator can be directed to run in parallel and collect results at the end of a prescribed number of operations, which allows the parallel unit test to validate FireWorks parallelism capabilities. The failover test is accomplished by directing the calculator to "fail" via division by zero, which results in a "FIZZLED" firework that the unit test detects and relaunches. The calculator can also be instructed to re-run a previous simulation, which is how reproducibility of a workflow is tested using the framework.

All unit tests are designed to run from the portal appliance (vapp), which is also where the MongoDB instance is hosted in user-space. While some tests are run locally on vapp, the majority leverage either Hokulea or Poi to run FireWorks components through the PBS queues located on each machine.

MongoDB ties everything together in the MHPCC environment:

- MongoDB instance is run in User space (port 27017) on Vapp2
- SSH Master connections using SSO authentication are created from Vapp2 to Poi and Hokulea– "ssh –YMNfq –p 2020 –i ~/.ssh/$user.id_rsa_sso machine"
- Reverse Port forwarding established through master connection from Vapp2 back from Poi, Hokulea, and elsewhere via "ssh –R 27017:localhost:27017 machine"
- Upon launching any firework through a queue (PBS or Slurm) port 27017 must be forwarded from the compute node back to head node.

- Launches to queues are executed from Vapp2, so direct password-less connection via ssh must be available to any compute resource (no return connection is required, however).

4.2 Prototype Testing – Results and Discussion

While developing the unit tests and use cases, it became clear that use of the FireWorks system could be simplified with the addition of helper functions tailored to the MHPCC environment. These simple tools and functions abstract the complexity of both FireWorks and the supercomputing environment from users who are unfamiliar with one or both.

Early on, a design decision was made to create a separate Launchpad and associated workflow name for each workflow and have these workflows run from a launch directory located under the $WORK directories of Poi and Hokulea. This implementation was tailored specifically to MHPCC and was made to keep the workflow system as simple as possible for the user and keep different workflows disambiguated. As such, every workflow ran under a different name and all associated fireworks were stored in a MongoDB collection that pertained only to that workflow.

This approach does not preclude storing multiple workflows in a single MongoDB collection, if necessary, but when using the tools developed for MHPCC, separate collections and Launchpad names are the default.

This approach is accomplished via the simpleLauncher helper function located in panacea/fireworks/scripts/site-packages/mhpcc_fireworks.py, along with a number of other helper functions created to tailor FireWorks to MHPCC. The simpleLauncher function requires only a workflow name and a remote system upon which to run, but it also allows a user to specify details such as the run directory, and batch queue specific information (e.g., wall time, processor count, queue name), among others.

Other helper functions (remote_launch and rqueue_launch) replaced the built-in rlaunch function of FireWorks, which could not operate within the MHPCC security constraints. These functions provided the capability to launch workflows on remote machines either on the head node or through the PBS queue. A more detailed description of all the developed helper functions is provided in section 4.2.2.

4.2.1 Unit Tests.

Unit tests meeting multiple requirements were developed to:

- Verify integrity of the MHPCC install
- MongoDB connectivity
- Test/confirm connectivity to/from Poi and Hokulea
- Test ability to launch fireworks via queue (PBS)

- Verify Software Requirements and Evaluation Criteria are met
- Test DB retrieval and storage
- Test repeatability, failover, and scalability
- Test load balancing.

The basis of the unit testing is an iterative Pi calculator that:

- Leverages Pytest
- Base firework created to use within unit testing
- Python-based, so works on Power and x86_64
- Iterative calculation of the value of Pi
- Stores calculations in a separate MongoDB
- Returns updated values to FireWorks MongoDB instance for data store
- Designed to run in serial or parallel
- Designed to "fail" to test fault tolerance
- Designed to run slow to test load balancing capability.

Specific unit tests and their functions were:

- **test_pi_simple** - Test capability to connect to MongoDB, create a workflow, launch the workflow, retrieve a value from the associated workflow database and confirm it matches value computed and stored elsewhere.

- **test_load_bal** - Test capability to launch workflows that span both Poi and Hokulea and show the ability to automatically load balance when one remote is running slowly.

- **test_fizzle** - Test capability to detect a "fizzled" (failed) firework, recover and re-launch the same firework.

- **test_rerun_pi -** Test capability reproduce output from a previously run workflow.

- **test_parallel_pi -** Test capability to run multiple fireworks in parallel as well as ability to set up dependencies of one firework upon another.

4.2.2 Helper Functions.

The following helper functions were developed to aid users by abstracting the complexities of the HPC environment and FireWorks:

- **simpleLauncher** - A class to initialize a new launchpad locally and on any remote computational machines. Determines appropriate settings for each remote, creates

my_launchpad.yaml and my_qadapter.yaml files and stages them to remote machines. Sets requested PBS queue parameters.

- **run_Pscan** - A function to encapsulate running an executable in conjunction with PScan package. Works with a wrapper (see scanParameters sample below) that will need to be modified if there are a different number of parameters. The PScan package expects only the parameters being varied to be part of the function call, so extra information is included in the wrapper for the executable name/location and the remote to use for fireworks. Although the parameters in scanParameters must be explicitly declared, the run_Pscan function will work on an arbitrary number as long as they are embedded in a dictionary.

- **run_exe** - A function to encapsulate running an executable in conjunction with scipy.optimize.minimze with user defined, json-style inputs. Executable will be run as a firework on a remote but return an evaluation value for the minimize function running on the controller.

- **run_exe_with_inputs** - A function to encapsulate running an executable with user defined inputs as a firework on a remote.

- **run_singularity -** A function to encapsulate running a Singularity container as a firework on a remote system.

- **getResults** - A class to handle collecting results from a remote machine and placing them at another location.

- **fileTransfer** - A class to handle transferring files to a remote machine for use in a workflow.

- **remote_launch -** A helper function to launch workflows/fireworks on a remote machine on the head node (primarily file transfers) Designed primarily to be used internally by functions and classes of the MHPCC fireworks software.

- **rqueue_launch** - A helper function to launch workflows/fireworks on a remote machine via the PBS/Slurm queue. Designed primarily to be used internally by functions and classes of the MHPCC fireworks software.

4.2.3 Use Cases.

RedLine developed several use cases:

- File Transfer template – Stages data from a remote source, operated on the data and returns the results to the remote destination.

- Simple Parameter Scan – Identify the function/executable to run and the parameters to scan, and the use case returns results.

- Optimization/Minimization – Encapsulates FireWorks within the scipy.optimize.minimize ecosystem. It provides a simple template for users to set up experiments.

- Run a workflow employing executable inside a Singularity container with the data outside the container.

- Run a complex workflow entirely within Singularity.

Specific use cases that were developed:

- **datatransfer.py** - Simple template to transfer files from an external source (s4.ssec.wisc.edu) to a remote (Poi/Hokulea) for use in a workflow.

- **transferOperateReturn.py** - Slightly more complex template that transfers files (input.params and threebody_params.py executable) from an external source to a remote (Poi), creates a simpleLauncher to handle the workflow, runs the executable on Poi with the transferred input parameters, and copies the results from Poi back to vapp upon completion.

- **parameter_scan.py** - Sample parameter scan case designed to be driven by the Scan function of the pscan package using a firework launched through the queue on poi to evaluate an executable (threebody.py) for each individual set of parameters defined by Scan.

- **minimize.py** - Sample use of the scipy.optimize.minimize module using a firework run through the queue on poi to evaluate a sample executable in order to find the parameters that result in the minimum value of the executable.

- **runSingularity.py** - Sample use case showing how to set up FireWorks to transfer a Singularity container to and input files to Poi, unpack the input files and run an executable inside the container against that data using 6 nodes and 150 cores.

- **runSingularityWRF.py** - Sample use case of a complex workflow executed by Fireworks using a singularity container with multiple executables. Multiple stages of the workflow are completed while other portions await dependencies to be completed before running.

4.2.4 Notebooks.

Because FireWorks is Python-based, it can be readily adapted to work within Jupyter notebooks or iPython. Several Jupyter examples were developed and presented to MHPCC for the demonstration of RedLine's solution. These examples can be found in panacea-workflows/fireworks/scripts/notebooks.

- **Minimize Example.ipynb** - Minimize use case that graphically plots the solution of a 2-dimensional function shown as a surface and then shows the use of a FireWorks embedded scipy.optimize.minimize function to find the parameters that produce the minimum value of that surface.

- **Parameter Scan.ipynb** – Notebook-based use case that shows the results of a parameter scan of solutions of the three-body problem computed using both FireWorks and the Pscan module.

In addition to validating the results of the trade study, the unit tests also served to confirm the proper installation of the FireWorks system, including connectivity to all remotes and the MongoDB server running on Vapp. When combined with the Singularity use cases (runSingularity.py and runSingularityWRF.py in panacea-workflows/fireworks/scripts/ use_cases), nearly all the evaluation criteria could be validated as well.

4.3 Prototype Testing – Conclusions

After a thorough trade study review, including hands-on testing, the FireWorks workflow solution was selected for MHPCC prototype development. With some adaptation to the Kerberized environment, FireWorks proved to be a robust system capable of handling a diverse number of workflow solutions for MHPCC users. With the addition of several helper functions and sample use cases, FireWorks also proved to be easily extensible and offered a great opportunity to help lower the barriers to using HPC systems for new users. Similarly, with the adaptation of scipy.optimize.minimize and pscan to the FireWorks framework, RedLine demonstrated how simple it can be to use FireWorks to run large-scale parameter scans and optimization problems utilizing MHPCC systems without a need for users to set up or understand how the workflow system operates.

For the delivery of this prototype system for MHPCC, RedLine followed a structured approach to software development. The prototyping plan was designed to validate the criteria used to select FireWorks as the solution best suited for MHPCC as determined by an exhaustive trade study. Unit tests were designed and built to establish basic functionality of the system within MHPCC, as well as demonstrate the ability to perform a number of desired capabilities (e.g., load balancing, reproducibility, data provenance, parallel execution). During the development of the unit tests, a design decision was made to abstract most of the specifics of both FireWorks and HPC from the user, which resulted in a series of helper functions. Use cases were subsequently developed to demonstrate the simplicity of setting up multiple common tasks such as using Singularity containers, staging and operating on external data, running parameter scans, and running minimization/optimization scenarios using Python as the driver and FireWorks to coordinate and stage the computations behind the scenes. The sample code provided with this software will provide a solid foundation upon which future MHPCC users can build and expand their own workflows for use within the MHPCC and eventually across all the DSRCs.

Protype testing validated that FireWorks is the best fit as a User Workflow Manager for MHPCC and potentially other DSRCs. Key FireWorks features of importance to MHPCC include:

- Reliance on SSO credentials, port forwarding, and ssh configurations to handle
- communications
- Python-based, so can be easily extended and encapsulated
- Helper templates greatly improve accessibility for novice HPC users interested in parameter scans, optimization/minimization, and file staging/operation/reporting
- Easily handles containers and complex workflows for more advanced users
- Provides data provenance and accessibility via MongoDB and Robo-3T
- Data for performance analysis and reporting is all stored and readily accessible in MongoDB.

4.4 Prototype Testing – Recommendations

Given that the software discussed herein is still a prototype system, there are a number of hurdles that must be overcome to bring FireWorks into operations. First of all, the MongoDB system is currently run in user space and is open to all users. For secure operations, individual authentication to the database will be required. This could be accomplished either via a centralized service or via individual instances of MongoDB with disambiguated ports (possibly based on user id + constant). Considering the ease with which FireWorks interacts with Jupyter, it would be an excellent opportunity to create a tile on the MHPCC web portal that utilizes both Jupyter and the MHPCC FireWorks installation. Sample notebooks containing parameter scans and optimization examples could be included as well. If it is decided to move FireWorks into operations, it would also make sense to expand upon the MHPCC specific helper functions to further abstract the workflow specifics from the user and make common tasks as simple as

possible. Overall, FireWorks provides the full functionality of a mature workflow solution for advanced users while also offering the ability to design workflows that require little to no knowledge of the system itself, which makes it a highly flexible and attractive solution for the varied MHPCC user community.

Redline recommends the following as the most logical next steps:

- Install MongoDB in a central location on vapp or design a system to launch individual instances of the server using unique ports based on user ID
- Implement user-based logins (already available) to MongoDB to improve security
- Improve the scripts that ensure that MongoDB connectivity is forwarded from all vapps(1/2/3), Hokulea, and Poi to host where individual's MongoDB is running.
- Add checks for ssh and MongoDB connectivity into the module used to load FireWorks to ensure an operational system whenever the module is loaded.
- Create a Portal tile with Jupyter set up to interact with FireWorks within MHPCC
- Create Portal tiles with templates for parameter scans and minimization/optimization runs at MHPCC.
- Extend the capability of FireWorks to interact with other DSRCs via SSO.
- Develop common templates/helper functions for use with FireWorks@MHPCC and incorporate them into training and workshops for new users.