Future Generation Computer Systems 159 (2024) 307–322

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

# Asynchronous modeling workflows in CyberWater with on-demand HPC/Cloud access

Ranran Chen [a,1], Feng Li [a,1], Daniel Luna [b], Isuru Ranawaka [c], Fengguang Song [a], Sudhakar Pamidighantam [c,d], Xu Liang [b], Yao Liang [a,*]

[a] Department of Computer and Information Science, Indiana University-Purdue University Indianapolis, 723 W. Michigan Street, SL 280, Indianapolis, 46202, IN, United States
[b] Department of Civil and Environmental Engineering, University of Pittsburgh, 728 Benedum Hall, 3700 O'Hara Street, Pittsburgh, 15261, PA, United States
[c] Cyberinfrastructure Integration Research Center, Pervasive Technology Institute Indiana University, 2709 E. 10th Street, Bloomington, IN, 47408, United States
[d] Center for AI in Science and Engineering (ARTISAN), Georgia Institute of Technology, 1283B CODA Building, 756 W Peachtree St NW, Atlanta, GA 30308, United States

ARTICLE INFO

*Keywords:*
Scientific workflow
Asynchronous workflow control
On-demand HPC access
Site recommendation
Hydrologic modeling

ABSTRACT

Workflow management systems (WfMSs) are commonly used to organize/automate sequences of tasks as workflows to accelerate scientific discoveries. During complex workflow modeling, a local interactive workflow environment is desirable, as users usually rely on their rich, local environments for fast prototyping and refinements before they consider using more powerful computing resources. However, existing WfMSs do not simultaneously support local interactive workflow environments and HPC resources. In this paper, we present a mechanism for on-demand access to remote HPC resources from desktop/laptop-based workflow management software to compose, monitor, and analyze scientific workflows in the CyberWater project. CyberWater is an open-data and open-modeling software framework for environmental and water communities. In this work, we extend the open-model, open-data design of CyberWater with on-demand HPC accessing capacity. In particular, we design and implement the LaunchAgent and JobManager, which can be integrated into a local desktop/Laptop environment to allow on-demand usage of remote HPC resources for computational modeling workflows effectively and efficiently. LaunchAgent manages user authentication to remote resources, prepares computation-intensive or data-intensive tasks as batch jobs, submits jobs to remote resources, and monitors quality of services for users. LaunchAgent interacts seamlessly with other components in CyberWater, providing advantages of user-friendly feature-rich desktop software experience and increased computing power through on-demand HPC/Cloud usage. In our evaluations, we demonstrate how a hydrological modeling workflow that consists of both local and remote tasks can be constructed and show that our new on-demand HPC/Cloud usage helps speed up hydrology workflows while allowing asynchronous HPC/Cloud access from workflows using a desktop graphical user interface.

## 1. Introduction

Scientific discovery often requires the execution of various coupled computational tasks using diverse data from local and remote resources. These tasks can be organized into stages, based on their data dependencies. A workflow management system (WfMS) is a type of software system with which an end-user can describe the data dependencies of tasks, compose workflows, and launch such workflows for execution in designated computing environments. Nowadays, different types of computing environments are supported by WfMSs. In a typical WfMS, workflows are described as directed acyclic graphs (DAG) where each vertex is a computation task, and the edges describe the data dependency between tasks. Such DAGs are then submitted to an execution environment, which is either a High-Performance Computing (HPC) system, or a Cloud system, or a local desktop/server computer. For example, in the Pegasus WfMS [1], one can provide an abstract workflow as a "DAX" file, and Pegasus translates it to an "execution workflow", which is then submitted to one of the supported execution environments. For a simple, small size workflow, a local computer can be used as the execution environment. For larger workloads, a

* Corresponding author.
*E-mail address:* yaoliang@iu.edu (Y. Liang).
[1] Equal contributors.

https://doi.org/10.1016/j.future.2024.04.023
Received 9 September 2023; Received in revised form 12 April 2024; Accepted 15 April 2024
Available online 13 May 2024

Pegasus/HTCondor pool of worker nodes [2] can be used instead, in which the tasks are mapped to a collection of worker nodes.

Although widely-used WfMSs such as Pegasus allow users to utilize various types of execution environments to launch computation tasks, there are three major limitations in practice. First, the choice between local and remote execution sites is not flexible: workflows are typically only allowed to run in their entirety in either local or remote environments. For workflows running in a local environment, computation power is limited; for workflows running remotely, it can take much longer time to prototype, design, develop, and debug. Second, from a workflow user's perspective, correctly preparing an abstract workflow requires a lot of effort for complex workflows. For example, Pegasus WfMS users must either manually create the DAX file, or use one of the supported programming interfaces to generate the DAX file. In contrast, desktop-based WfMSs such as VisTrails [3], provide a feature-rich GUI-based frontend, which allows users to drag and drop widget boxes to form a complex workflow, and gives users comprehensive and timely information such as execution provenance. Third, there is lack of systematic real-time assistance for users to knowledgeably select an appropriate HPC platform for their jobs to maximize users' benefits in terms of performance/cost ratio when multiple HPC platforms are available.

To address these limitations, we present on-demand access to HPC/-Cloud resources from desktop-based WfMSs, a novel approach to provide desktop-based (or laptop-based) workflows with an automated mechanism for on-demand access to remote computing resources, so that rich configurations and relatively small computation tasks can be performed in a user's local environment, and only computationally expensive tasks are offloaded to powerful HPC/Cloud resources whenever needed. On-demand access to HPC/Could is achieved by the development of *LaunchAgent*, handling the offloading of computational tasks (i.e., workflow items) to remote HPC/Cloud resources, so that users can select specific computationally expensive tasks to be offloaded to HPC/Cloud platforms automatically during the workflow configuration via its local graphical user interface (GUI), to achieve significant speedup. *LaunchAgent* supports both direct Slurm-based [4] access and Airavata gateway [5] access to remote computing resources. Such a more flexible design allows users to make use of both mid-size campus-based clusters and large-size grid computing resources (e.g., from NSF XSEDE/ACCESS [6]).

This work is part of the CyberWater project [7–9], which aims to create an open-modeling and open-data framework to accelerate collaborative water research. CyberWater framework system currently adopts VisTrails, a Python-based desktop workflow management software, to support functionalities such as provenance management and reproducible computing for exploratory computation tasks. With CyberWater various data agents, model agents, and generic model agent toolkit [8], users can compose and configure their workflows to integrate heterogeneous environmental data sources and diverse computational models, and enable model coupling through their desktop/laptop GUI. However, the original VisTrails software only supports a synchronous workflow controller, which schedules workflow tasks/items using a predefined pipelined structure and disallows independent computational tasks in the workflow to be offloaded to HPC platform(s) and run in parallel, resulting in an unnecessary delay for the entire workflow computation. To address this issue, we design and develop an asynchronous HPC task scheduling middleware, which enables independent computational tasks to be scheduled and submitted to remote HPC/Cloud platforms simultaneously with non-blocking *LaunchAgent*. Furthermore, to aid users in better selecting HPC platforms and allocations for their jobs at hand, we also design and develop an HPC site-recommendation mechanism to provide users with useful information including expected job execution time and cost for available choices.

To demonstrate our approach and development in CyberWater framework, our CyberWater case study uses hydrological models including the Variable Infiltration Capacity (VIC) model [10–14] and the Distributed Hydrology Soil Vegetation Model (DHSVM) [15] over several watersheds in Pennsylvania, USA. Our experiments show not only a drastic acceleration of computation of hydrological model applications via on-demand access to HPC resources, but also a significant speedup of using asynchronous workflows in CyberWater compared to the synchronous ones. The initial work on HPC on-demand access for synchronous modeling workflows in CyberWater was reported in Ref. [16], which has been significantly extended. The extensions include the new development of asynchronous workflow control, a non-blocking version of *LaunchAgent*, an HPC site-recommendation method, and comprehensive performance study on asynchronous versus synchronous modeling workflows with on-demand HPC access. The major contributions of our work are as follows:

- We present on-demand access to HPC/Cloud resources from desktop-based workflow systems, a novel approach that enables offloading computationally expensive tasks to remote HPC/Cloud platforms, while utilizing the desktop-based WfMS' feature-rich GUI frontend to facilitate users' complex workflow construction and configuration as well as workflow provenance for reproducible computing.
- We design and develop an asynchronous workflow control mechanism to support asynchronous workflows in CyberWater with on-demand HPC/Cloud access to maximize the capacity of on-demand HPC/Cloud access for independent workflow tasks and thus further speed up workflow computations in the CyberWater system.
- We develop a non-blocking *LaunchAgent* library in CyberWater to support on-demand access to remote computing resources including various HPC platforms, Google Cloud, and campus-based clusters/servers.
- We design a mechanism for HPC site recommendations to aid users in selecting appropriate HPC platforms/allocations to optimize the performance/cost ratio for their offloaded computational jobs.
- We present a real-world hydrological modeling use case study to validate and evaluate our approach in the CyberWater framework system, ranging from entirely local desktop/laptop workflow computation versus on-demand HPC computation to asynchronous workflow versus synchronous workflow with on-demand HPC access.

The remainder of this paper is organized as follows. Section 2 briefly overviews the background of CyberWater to set up the context. Section 3 presents our approach, design, and implementation. Section 4 provides use cases of reproducible end-to-end model simulations to demonstrate the use of the HPC module and asynchronous workflow feature for open data and open model integration in CyberWater. Finally, Section 5 concludes the work and gives planned future work.

## 2. Background

### 2.1. CyberWater

CyberWater is a collaborative project for creating an open-data and open-modeling software framework [7–9,16]. The CyberWater project aims at reducing user time and effort needed for hydrologic modeling studies by enabling flexible integration of diverse data sources and users' computational models needed for executing complex workflows with on-demand remote HPC/Cloud resources. The CyberWater system is based on the Meta-Scientific-Modeling (MSM) framework [7] to address the challenges of accessing heterogeneous data sources and integrating individual models. The CyberWater MSM framework consists of four parts: a core (the MSM core), an interface to the Workflow Engine, Data Agents, and Model Agents, as shown in the dashed box in Fig. 1. Data Agents are dynamically loaded components that handle how to connect to and retrieve data from different external data

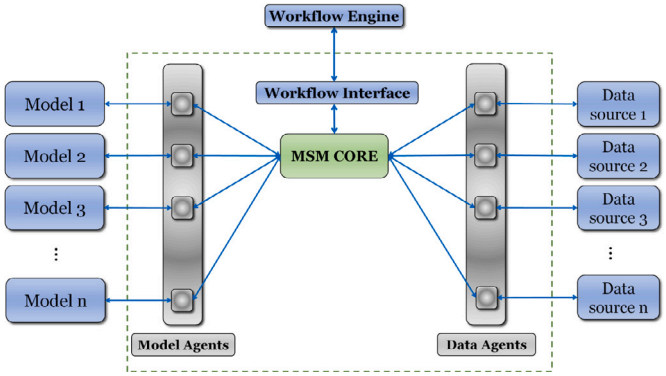



**Fig. 1.** An overall architecture of the MSM framework system.

```
//Topological Sort Algorithm
BEGINFUNCTION topologicalSort
CREATE a temporary stack
CREATE a Boolean array named as visited
//Initialize the values of all the elements in the visited array as False
SET all the nodes in the graph as not visited
CALL the topologicalSortUtil function to traverse all nodes in the graph
ENDFUNCTION

BEGINFUNCTION topologicalSortUtil
//Mark the current node as visited
SET the value of the current node in the visited array as True
CALL the topoloaicalSortUtil function itself recursively for all the adjacent nodes to the current node
PUSH the current node to the temporar stack to store the result
ENDFUNCTION
```

**Fig. 2.** The pseudo-code flattens the acyclic graph into a linear list in topologically sorted order.

providers through the Internet. Model Agents are dynamically-loaded components that handle the input/output and execution specifications of various hydrological models. The Core interacts with the Workflow Engine through the Workflow Interface to prepare and trigger individual tasks (e.g., data retrieving tasks and model execution tasks) specified in workflows.

VisTrails [3], a Python-based graphical science workflow system running in desktop environments, is currently adopted in the CyberWater project as the workflow management system and provides the Workflow Engine that MSM core interacts with. Through the VisTrails workflow system within the CyberWater framework, users can simply drag and drop component modules in their rich desktop environments to compose complex workflows. When the workflow is executed, each component module (i.e., workflow item) of the workflow is executed locally on a desktop/laptop. VisTrails is an open-source scientific workflow and provenance management system that supports simulations, data exploration, and visualization. Like other existing Scientific Workflow Management Systems (WfMSs), VisTrails[2] processes a sequence of tasks by stringing individual VisTrails modules altogether into a pipeline and executing each pipelined independent computational task in a sequence. Users can compose complex hydrology workflows using VisTrails graphic user interfaces. Then, the tasks defined in the workflow are captured by the MSM Core, which triggers actions such as data fetching, model execution, and data processing/transformation, by means of Model Agents and Data Agents.

### 2.2. Synchronous workflow in VisTrails

VisTrails adopted in the CyberWater MSM only supports synchronous blocking workflow scheduling. In VisTrails, each module is represented by a Python class, which specifies the computation and implements a set of ports for connection in data and control flow [17]. For software and programming middleware, the pipelined execution model has always been considered fundamental [18]. It can be represented as an algorithmic pipeline skeleton, where a series of data enter the input module and pass through successive modules via ports until the final result is computed. This type of pipeline workflow executes in a synchronous mode, where each task is completed within a fixed period, corresponding to the longest cycle-time to operate a module [19]. Each module performs computation and passes its produced data set to the following-connected module in a period.

Orthodoxly, the execution order of the workflow is driven by data requests, and in VisTrails, the modules are executed in a bottom-up mechanism, with which each input is generated on-demand by recursively executing upstream modules, and the execution order is scheduled in topologically sorted order by fitting each module as a node in an acyclic graph and traversing them with the Depth-First Search (DFS) algorithm.

To support on-demand access to HPC/Cloud resources from desktop-based WfMS, CyberWater develops an HPC module in the generic model agent toolkit to provide a bridge between the local workflow and the HPC resources, which submits a job of the computational model to be executed in remote HPC clusters, and retrieves the result data back to local workflow environment for provenance, analysis, and visualization in the same workflow. Although the capability of on-demand access to HPC computation power and storage already drastically enhanced the performance, the synchronous blocking pipelined workflow scheduling mechanism of the original VisTrails still significantly limits the effectiveness and efficiency of workflows fundamentally.

There are three major components in the CyberWater framework to ensure the workflow can be executed in topologically sorted order: (1) Module, (2) Pipeline Scheduler, and (3) Workflow Controller. In the following discussion, we provide a concise overview of these three components.

**Module:** A module is represented as a node in the workflow graph, and CyberWater adopts heterogeneous execution models encapsulated within Python packages in these modules to perform computation, data generation, and data transfer facilitated through the interconnected ports existing between modules.

**Pipeline Scheduler:** The pipeline scheduler is implemented by a class named "pipeline" derived from the base class "***DBWorkflow***", which defines the fundamental attributes of the pipeline. It also depends on the "Graph" class, which provides the method for validation of workflow as an acyclic graph and flatten the graph into a linear list in topological sorting order as illustrated in Fig. 2.

**Workflow Controller:** In CyberWater's VisTrails, the execution of a workflow is controlled by the execution engine/workflow controller, which serves as the interface with database (***db***) layer. The ***db*** layer consists of three components: the domain objects, the service logic, and the persistence methods. The workflow controller is responsible for keeping track of invoked operations and capturing the provenance of workflow execution. The underlying principle of pipelined workflow execution in CyberWater's VisTrails involves the pipeline scheduler assigning a unique ***moduleId*** to each module in workflow configured with parameters in the graphic interface. The pipeline scheduler then generates a list of modules in the topological sorting order for the workflow controller to execute sequentially.

### 2.3. Airavata gateway framework

Apache Airavata [5] is a science gateway software framework to compose, execute, and monitor distributed applications running in

[2] www.vistrails.org.

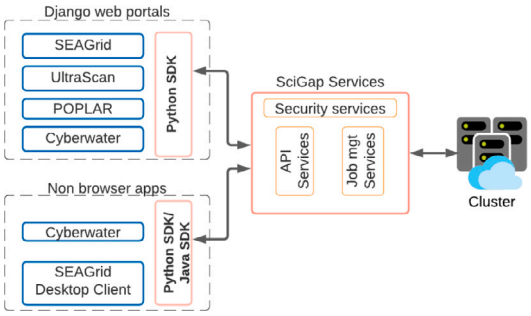



**Fig. 3.** SciGaP integration overview. SciGaP can expose API services to both browser-based and non-browser applications.

environments from local clusters to computational grids and clouds. The Airavata framework is a collection of distributed microservice components of identity management, application and experiment management, job and workflow management, and digital object-sharing management. For instance, Science Gateway Platform (SciGaP) [20] provides Apache Airavata software as a hosted middleware service on Indiana University (IU) Intelligent Infrastructure systems.[3] SciGaP exposes public APIs that science gateways can use to outsource those general capabilities, as shown in Fig. 3. Through the API services, researchers can register an application by specifying information such as executable script path, environment variables, input/output arguments, and data files. The API services also allow the gateway administrator to add computing resources (e.g., clusters) so that when a SciGaP gateway user requests an experiment execution, the corresponding jobs will be created, launched on a designated HPC cluster, and monitored by the job management services.

As illustrated in Fig. 3, Science Gateways may have web portals, non-browser desktop/device-based apps, or a combination of both for their end-user researchers. To this end, the Airavata framework provides software development kits (SDKs) to connect with SciGaP and Apache Airavata services. The API services provided by SciGaP have been successfully used in different domains.[4] Web browser-based interfaces using the Django web framework have been developed and provided as a reference to enable users to configure, launch, and monitor jobs/workflows. However, the browser-style integration sometimes is not suitable for certain scientific workflow applications such as CyberWater applications that require feature-rich desktop-based VisTrails workflow management tools. On one hand, CyberWater uses the Django web interfaces for tasks such as user registration and computing resource management, but on the other hand, it utilizes the Python SDK to configure, launch, and monitor applications from programs. Our integration method of the SciGaP API services into the CyberWater system is described in Section 3.1.

## 3. Methodology

The concept of on-demand access to HPC/Cloud in the CyberWater system refers to the flexibility and capability of users to offload those specific computationally expensive tasks to HPC/Cloud at individual workflow item/module granularity instead of at the entire workflow level, in which the overall workflow still runs locally on a desktop/laptop with only the selected computationally expensive workflow item(s) to be offloaded to remote HPC/Cloud platforms in an automated manner. To achieve on-demand access to HPC/Cloud resources, an HPC module is developed in the generic model agent toolkit in CyberWater [8], which provides users with the unique on-demand capacity to

[3] https://uits.iu.edu/services/intelligent-infrastructure.

[4] SciGaP collaborators and clients: https://scigap.org/pages/collaborations.

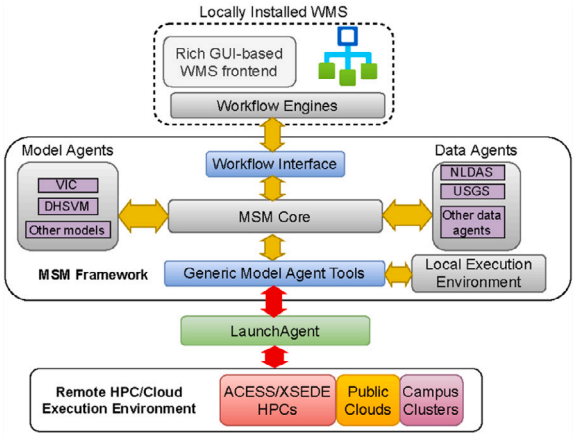



**Fig. 4.** The new HPC/Cloud-enabled CyberWater software framework with *LaunchAgent* integration. *LaunchAgent* extends the open-model and open-data design of the CyberWater MSM framework and allows the selection of computationally expensive tasks to be launched as remote jobs in different remote computing environments.

launch jobs of their model execution to the selected remote HPC/Cloud platform, and then to retrieve results back to the workflow when the execution is completed. In the following subsections, we present the design of *LaunchAgent* (a core component of our HPC module), the design of asynchronous workflow control in CyberWater, and the design of HPC site-recommendation to help users select appropriate HPC sites to maximize their performance/cost ratio.

### 3.1. Design of LaunchAgent

*LaunchAgent* is a core component for on-demand HPC/Cloud access in the CyberWater software framework. *LaunchAgent* is designed to help workflow engines offload computationally intensive tasks to remote HPC/Cloud resources on demand through Python programming APIs. As shown in Fig. 4, a locally installed workflow engine (e.g., VisTrails) on a desktop can manage a workflow as a graph of computational tasks/modules. For the default local computation setup, tasks are scheduled by the local computer's operating system scheduler, and communication between tasks happens in the form of memory objects/local files.

With the help of *LaunchAgent*, specific tasks along with their input files can be offloaded to remote computing resources to accelerate the user's workflow via the use of the HPC module in the generic model agent toolkit. *LaunchAgent* manages user authorization so that users have proper access to remote resources. It also composes, submits, and monitors the computation tasks for the users. Through the design of a universal Python API, a local workflow engine can periodically check the submitted tasks' status in remote sites and download output files when the tasks are finished. Listing 1 below illustrates how a VIC application can be deployed in the BigRed3 cluster using the *LaunchAgent* interface. This example assumes a user has already been allocated with an account and credentials for the BigRed3 HPC system. At step 1 in Listing 1, a user initializes the *LaunchAgent* library by specifying a registered HPC site ('bigred3'), with the provided username and password. Then at step 2, the local "vic" folder is uploaded to the remote site, and this folder stores the running environment of VIC5 (including input files and configuration files). After that, at step 3, the configure_slurm _job function generates the remote batch job script for the HPC site, based on the given execution logic specified in the "execute_script". Then, the run_monitor_job function submits the generated job scripts to the remote HPC site and returns from the function until the job is finished on the HPC/Cloud side. At last, the model outputs generated on the remote site are downloaded back to the local environment.

```
# 1. HPC resources authentication
agent = LaunchAgent(site_name = 'bigred3',
    project_name = 'cyberwater', username = args
    .username, passwd= args.passwd)
# 2. Upload local folder.
agent.upload_folder('vic')
# 3. Configure commands and resources for
    experiment
job_id = agent.configure_slurm_job(nodes = 1,
    ntasks_per_node = 2, walltime_in_mins= 10,
    execute_script = 'vic/run.sh')
# 4. Run the job remotely and wait until job
    finishes.
agent.run_monitor_job(job_id)
# 5. Download all results to local directory.
agent.download_folder('./results_vic_ssh')
```

Listing 1: An example of *LaunchAgent* applied to the VIC hydrological model.

The initial prototype implementation of the *LaunchAgent* library shown in Listing 1 requires a blocking call "agent.run_monitor_job (job_id)": the workflow engine needs to wait for the completion of the remote jobs before it can proceed to submit/execute other jobs. Such a design works well for simple synchronized workflows that have a sequential execution order. In a more complex asynchronous workflow, however, multiple components can execute concurrently when there are no data dependencies between them. In such cases, the initial blocking approach limits the potential performance gain. For this reason, we designed a non-blocking (i.e., asynchronous) approach to launching remote jobs. Compared with the blocking approach, the steps of *LaunchAgent* library initialization, job configuration and data uploading/download in the non-blocking approach are the same as the steps in the blocking approach. Only the "agent.run_monitor_job" call (step 4 in Listing 1) is changed and split into two pieces:

1. agent.launch_job(job_id)
2. status = agent.get_job_status(job_id)

This separation of job launching and monitoring allows users to launch multiple remote jobs without blocking and users only check job status when needed.

To coordinate the execution order of multiple asynchronous jobs, *LaunchAgent* assigns each Slurm-based or Gateway-based job with a unique job ID and maintains a JSON-based document database that records the updated status of all configured remote jobs. Each remote job goes through a sequence of statuses: CREATED, PENDING, RUNNING, COMPLETED/FAILED. For the Slurm-based method, the remote job status is fetched by *LaunchAgent* using Slurm "scontrol show job" command; and for the Gateway-based method, an equivalent function "api_server_client.get_experiment_status" is used. Note that each time when the job status changes, the updated status is persisted to the user's local filesystem, and such job state data allows the workflow engine to either redo (resubmit) or undo (cancel) an unfinished remote job when there is a failure (e.g., the internet connection drops, gateway service becomes unavailable). A use case of the new non-blocking job launching design is shown in Section 4.1, where we run a workflow that couples the VIC5 model with the DHSVM model.

Typically, researchers have access to two types of cluster resources: on-campus clusters and remote clusters accessible through services such as ACCESS (i.e., the previous XSEDE[5]). At Indiana University, there are high-performance/high-throughput clusters such as BigRed 3,[6] and Karst. Those resources typically require University IDs to operate and usually have lesser computing capacity. On the other hand, extreme-scale computing infrastructure provided by ACCESS,[7] such as TACC Stampede2[8] and PSC Bridges-2[9] provide significantly higher computing capacity. To this end, *LaunchAgent* has been designed to suit both settings through two channels: a direct Slurm-based channel, and an Airavata gateway-based channel. The details of these two channels can be found in [16]. Here we mainly focus on the direct Slurm-based channel, as it is a more actively updated channel through which we develop new functionalities such as asynchronous job management.

We developed the direct Slurm-based channel for *LaunchAgent* using Paramiko SSH2 Python library.[10] The direct Slurm-based *LaunchAgent* provides functions such as remote authentication, file management, and job control/monitoring for user tasks offloaded to remote slum-based HPC systems. With the direct Slurm-based method, each user has his/her own login credentials to remote resources, which allows them to access HPC/Cloud systems directly through *LaunchAgent*. A typical use case of the Slurm-based *LaunchAgent* is campus-based clusters, to which most university students/faculties have direct access.

With the programming interface shown in Listing 1, a user needs to configure the folder to be uploaded, which includes input data and running configurations. *LaunchAgent* automatically archives the user's folder, submits it to remote computing resources, and retrieves output data once the job finishes. For authentication, Slurm-based *LaunchAgent* allows users to authenticate themselves using their HPC login user name, with either passwords or SSH key pairs. Initially, we developed our prototype on Indiana University's BigRed3 supercomputer. Using the generic Slurm-based agent, we were able to launch parallel programs from personal computers. To accommodate more compute-intensive modeling applications on broader HPC platforms, we added support for ACCESS/XSEDE supercomputers, such as PSC Bridges-2 and TACC Stampede2. We realize that different systems have different requirements for user authorization and authentication and then add new support for them in the direct Slurm-based *LaunchAgent* implementation. For example, PSC Bridges-2 requires a user to upload his/her SSH login public key through a specific key management web page (operated by PSC), and Stampede2 requires multi-factor authentication (MFA) for each SSH session.

Apart from the ACCESS/XSEDE HPC resources, the direct Slurm-based *LaunchAgent* also supports cloud computing sources such as Google Cloud Platform and JetStream Cloud. To initialize Slurm clusters from those Cloud providers, we utilize the Slurm-GCP tool[11] for the Google Cloud Platform, and use the JetStream Elastic Slurm Cluster tool[12] for JetStream Cloud. Both Slurm-GCP and JetStream Elastic Cluster tools allow dynamic resizing of clusters by allocating additional cloud virtual machines on demand.

Alternatively, *LaunchAgent* can support the Gateway-based channel, developed based on the SciGaP framework. The Gateway channel allows a gateway administrator to pre-configure a set of remote resources for all CyberWater community users. A new user can register him/herself through the Gateway web portal (https://cyberwater.scigap.org). Once a user joins the CyberWater gateway and is approved by the administrator, he/she gains access to a group of pre-configured HPC systems defined by the group resource profile. The interface of using the gateway channel is similar to that shown previously in Listing 1 (however, with the GateWay channel, the CyberWater gateway username/password is used during the agent initialization function, instead

[5] https://www.xsede.org.

[6] Supercomputers for academic research at IU, https://kb.iu.edu/d/alde.

[7] https://access-ci.org/.

[8] Texas Advanced Computing Center. Stampede2 HPC system, https://www.tacc.utexas.edu/systems/stampede2.

[9] Pittsburgh Supercomputing Center. The Bridges-2 HPC System, https://www.psc.edu/resources/bridges-2.

[10] https://www.paramiko.org.

[11] https://cloud.google.com/solutions/deploying-slurm-cluster-compute-engine.

[12] https://github.com/XSEDE/CRI_Jetstream_Cluster.

of per-HPC-system credentials). With an appropriate HPC site selected, the Gateway-based *LaunchAgent* can use the selected site in the group resource profile to launch computational tasks. More details of the Gateway channel design can be found in [16].

The *LaunchAgent* component is invoked by the use of our developed HPC module in the CyberWater generic model agent toolkit, which provides a straightforward HPC configuration mechanism with a GUI interface, hiding all detailed operations to access HPC remote resources from users. A real-world example using the HPC module is described in Section 4.1, where we configure VIC5 model and DHSVM model with the HPC module of generic model agent toolkit for remote execution.

### 3.2. Asynchronous workflow control

It becomes more common for modern WfMSs to support adaptive scheduling and parallel workflow execution [21]. In the following, we will focus on the design of pipeline parallelism in CyberWater, i.e., the design of asynchronous workflow control in CyberWater to overcome the limitation of VisTrails' workflow control, and maximize the effectiveness of CyberWater's on-demand HPC access.

To increase the degree of parallelism in CyberWater for on-demand access to HPC, a new asynchronous HPC pipeline scheduling middleware is designed and developed to enable independent tasks residing in different branches of the workflow to be offloaded to different HPC resources simultaneously, through scheduling workflow items in an asynchronous mode. In contrast to synchronous workflow scheduling, multiple independent workflow items/jobs are submitted to the remote HPC/Cloud resources through a non-blocking job submission mechanism, without waiting for the first submitted job to finish before submitting another one.

To achieve asynchronous workflow control in CyberWater for on-demand HPC job submissions, our design encompasses the following components:

1. The *LaunchAgent* is designed to be non-blocking, returning control to the CyberWater workflow immediately after a user's task is submitted to a selected remote HPC/Cloud platform. Whether the submitted job is executed or not in the remote site does not block the workflow progression.
2. The Workflow Controller is extended to facilitate the rescheduling of pipelined workflow items by rearranging the execution order of modules in the pipeline queue. This enhancement enables dynamic adjustments to the workflow based on the status of offloaded jobs to HPC platforms.
3. A *JobManager* is introduced to monitor (via *StatusManager*) and manage the execution of submitted remote jobs. It is responsible for tracking the progress of offloaded user tasks and handling the completion of these tasks.

The non-blocking behavior of the *LaunchAgent* was discussed in the previous subsection. This subsection focuses on the design of the *JobManager*. The Workflow Controller undergoes modifications to incorporate the newly developed *JobManager* into the process of checking the status of submitted jobs on the remote HPC platform(s) before processing the modules. The non-blocking asynchronous mechanism, positioned between the pipeline scheduler and the modified VisTrails workflow controller, operates as follows: When the controller is ready to execute the next module, it calls the *JobManager*. The *JobManager* first checks the status of any submitted but incomplete job at the corresponding remote HPC site via the *StatusManager*, then updates the job status table accordingly, and returns the job status.

There are two scenarios in which the Workflow Controller invokes the Rearrangement procedure, as demonstrated by the pseudo-code in Fig. 5, to rearrange the pipeline queue: (1) When a job has just been submitted by the current HPC module; and (2) when a previously submitted job by the pending HPC module has not been completed

```
// Rearrange the workflow pipeline list
BEGINFUNCTION rearrange
GET the current workflow pipeline list
GET the subgroup list in topological sorting order
//Put the modules in the subgroup list back to the end of the workflow pipeline list altogether
FOR every module in the current workflow pipeline list
    IF the module is also in the subgroup list
        MOVE the module to the end of the workflow pipeline list
    ENDIF
ENDFOR

// Rearrange routine
PROGRAM Rearrange
START
//Rearrange the pipeline after the HPC module executes and submits a job to the remote HPC platform
IF the current HPC module hasn't submitted a job before
    CALL LaunchAgent.configure_job function to configure the remote platform and generate a job script
    CALL LaunchAgent.upload_folder function to upload a folder of forcing data files and parameter files to remote working directory
    CALL LaunchAgent.run_monitor_job function to submit the job and monitor this job status and return to the workflow controller
    CALL LaunchAgent.check_status function to check the status of the submitted job.
    CALL JobManager.update_status function to update the status of the submitted job in job status table
    CALL PipelineScheduler.topologicalSort function to get the subgroup list of the current module and its successor modules
    CALL WorkflowController.rearrange function to rearrange the workflow pipeline list
ENDIF
// Rearrange the pipeline if the job submitted by the current HPC module has not been completed on the remote HPC platform
IF the current HPC module has submitted a job before and this job has not been finished on the HPC platform
    CALL PipelineScheduler.topologicalSort function to get the subgroup list of the current module and its successor modules
    CALL WorkflowController.rearrange function to rearrange the workflow pipeline list
ENDIF
END
```

**Fig. 5.** The pseudo-code for the rearrangement procedure in the modified workflow controller component to rearrange the workflow pipeline list.

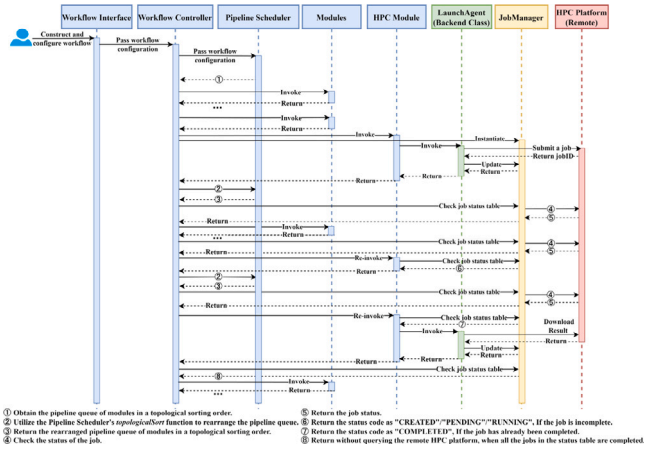



**Fig. 6.** A typical sequence diagram of non-blocking asynchronous-controller workflow in CyberWater.

on the remote HPC platform. In these situations, the modified Workflow Controller invokes the ***topologicalSort*** function of the pipeline scheduler to retrieve the subgroup list of successor modules from the current/pending HPC module. The pipeline list is then rearranged by moving the current/pending HPC module, along with its successor modules, to the end of the pipeline list.

Once the submitted job has been completed with a status code of "COMPLETED" in the job status table, the workflow controller triggers the *LaunchAgent* to download the result files. This process is illustrated in a typical sequence diagram depicted in Fig. 6.

The *JobManager* monitors the status of submitted jobs (refer to Fig. 7), tracks and manages jobs on remote HPC sites by utilizing the *StatusManager* backend class within the *LaunchAgent* to send query requests. This occurs prior to the execution of each workflow module, specifically when there are pending jobs on the remote HPC platform. Upon receiving a response from an HPC site, the *JobManager* updates the job status table accordingly. Three main functions of *JobManager* are described below. The entire operation process of *JobManager* is illustrated in Fig. 9.

> **Create:** During the initiation of workflow execution, an abstract data type(ADT) named job status table is created. This ADT is instantiated as a Python dictionary data type and consists of three key fields: module ID, job ID and status. Table 1 describes the structure of the job status table.

**Table 1**
The table describes the usages of each field of the job status table within the *JobManager* component.

| Field | Description |
|---|---|
| moduleId | The Id of the HPC module submitting the job. |
| jobId | The Id of the job submitted within the HPC module. |
| jobStatus | The status of the job on the HPC module. The state-transition diagram of the submitted job is depicted in Fig. 8. |

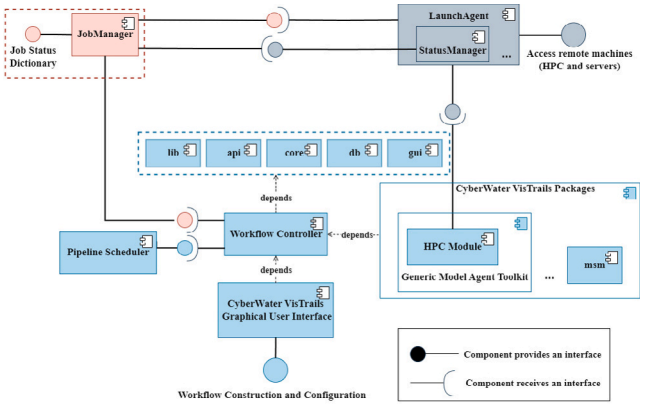



**Fig. 7.** Component diagram of HPC asynchronous control middleware of CyberWater VisTrails.

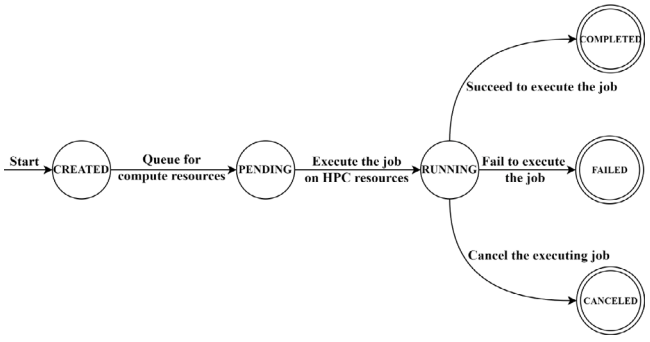



**Fig. 8.** The state-transition diagram of submitted job described in the job status dictionary of *JobManager* middleware.

**Check and Update:** Whenever there is an incomplete job on a remote HPC site, the *JobManager* queries its status and updates the job status table. This action is performed each time a module in the pipeline list is about to be executed by our modified workflow controller, specifically for jobs that have not been marked as *"COMPLETED"* in the status table.

**Clear:** Upon the completion of the entire workflow execution, the job status table maintained within the *JobManager* instance is automatically deleted, ensuring the cleanup of resources.

To elaborate on the mechanism of the dispatcher in *JobManager*, we give a simple workflow example as shown in Fig. 10, which includes 8 module nodes with corresponding moduleId ranging from 1 to 8. At the very beginning of the workflow execution, the order to execute the modules is determined by topological sorting represented by the sequence of moduleId(s). Since there are two HPC modules (corresponding to two independent model computations respectively) in the workflow, with the previous synchronous workflow control method, module 4 cannot be processed until module 3 finishes its job execution and retrieves the result files via a blocking submission. In contrast, by applying the new asynchronous workflow control, module 3 will return once it submits the job via a non-blocking submission; module 4 then can submit its job without waiting for the completeness of the job submitted by module 3. This way, the two HPC jobs are being parallelly

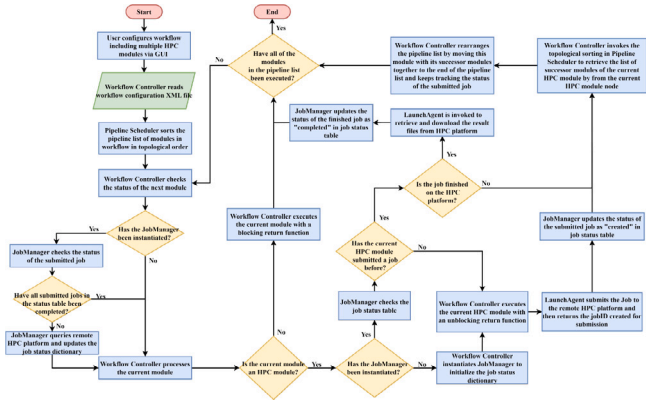



**Fig. 9.** Flowchart description of *JobManager* for asynchronous workflow control in CyberWater.

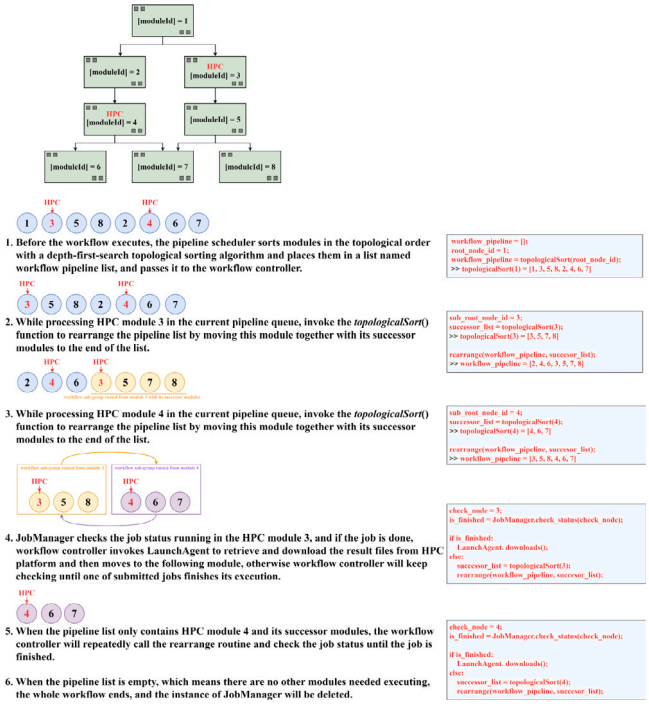



**Fig. 10.** Illustration of the mechanism of dispatching routine in *JobManager* with the example of 8 nodes in a workflow including 2 HPC modules with dependency in their successor modules.

offloaded to and executed on remote HPC platforms from modules 3 and 4 respectively, speeding up the total execution time of the entire workflow.

### 3.3. HPC site recommendations

As different HPC platforms have different computational powers, available resources, and charge rates, it would be desirable if some aid can be provided to users about how to intelligently select an appropriate HPC site and allocation for a given job at hand, to optimize

**Table 2**
Statistics collected from a remote execution configured by *LaunchAgent*.

| Attribute | Meaning | Example |
|---|---|---|
| app_name | The name of an application. | "dhsvm" |
| app_constraints | The resource planning constraint. | {"max_cores":1} |
| app_config | Application runtime configuration. | {"sim_days":365, "sim_cells":30 686, "step_size_hr":1} |
| resource_config | Choice of site and resource allocation used for remote execution | "{site_name":"BigRed3", "launch_info":{"nodes":1, "ntasks_per_node":1, "partition":"general", "walltime":"00:10:00"}} |
| $T_{exec}$ | Time used for remote execution. | 387 s |
| $T_{upload}$ | Time spent for uploading local artifacts. | 2.5 s |
| $T_{download}$ | Time spent for downloading remote results. | 2.9 s |
| $T_{queue}$ | Time spent for queuing for batch system. | 30 s |

users' performance/cost ratio. In addition to providing the flexibility for a user to configure a remote run with different resource allocations, *LaunchAgent* has been extended to give site recommendations based on execution records of historical jobs and resource descriptions of available execution sites. If an application (e.g., DHSVM or VIC5) has been previously configured by *LaunchAgent* and finished successfully, *LaunchAgent* can use the recorded job histories to estimate the execution time and data transfer time for future remote executions of the same computational model. Note that a "future execution" can have different application configurations (e.g., spatial and time resolution, study area, and time period).

To make site recommendations, a generic performance model has been added in *LaunchAgent* so that we can infer the execution time and data transfer time of the same computational model application when using different (computation) resource allocations. *LaunchAgent* then ranks all allocation plans using predefined rules (e.g., minimized total time) to give realistic site suggestions. We first introduce how *LaunchAgent* collects execution data, then we describe how we model and predict execution time and data transfer time and make reasonable resource planning suggestions. Finally, we demonstrate how our time prediction method works in practice and how the site recommendation integrates with CyberWater front end.

### 3.3.1. Job statistics collection

Table 2 shows a non-exhausted list of attributes currently gathered by *LaunchAgent* for each configured remote job. The recorded information reveals the information of the application itself (app_name, app_constraints, app_config), the used resource application (resource_config), and performance timing data ($T_{exec}$, $T_{upload}$, $T_{download}$, $T_{queue}$).

Besides the information in Table 2, *LaunchAgent* also maintains a certain static description of different execution sites. For each site, such static information includes the login address and an optional "core speed" field. The login address allows the *LaunchAgent* to contact remote sites to get the updated queue/partition usage information, and the "core speed" field gives hints about the relative computation speed of each CPU core in an HPC site. Currently, *LaunchAgent* has preconfigured several sites including IU BigRed3, TACC Stampede2, and PSC Bridges2, and users can also add other sites from the GUI front end.

We have also added credential management in *LaunchAgent*, so credentials used for remote login are safely saved locally using the system's keyring service (e.g., Windows Credential Locker). With saved credentials, *LaunchAgent* iterates over each candidate site and predicts the time needed for a given workload, and recommends the best resource plan based on users' preferences.

If the user wants to cancel a job during the execution of a workflow, canceling the workflow will not cancel the jobs that were already submitted to the remote HPC platforms. Given that, CyberWater provides an interface displaying the job ID, site name, status, created time, and remote ID, by which a user can check multiple jobs and then click the 'cancel' button to cancel the job as illustrated in Fig. 11.

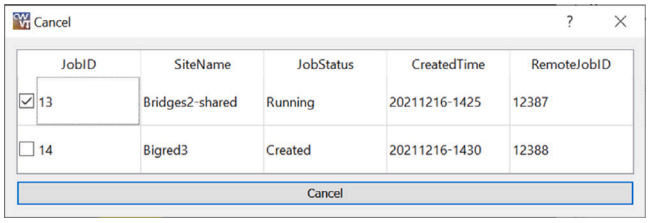



**Fig. 11.** The cancel interface of CyberWater VisTrails to cancel the jobs running on the remote HPC platform.

### 3.3.2. Site recommendations

As *LaunchAgent* gathers more execution statistics of user model applications with different resource allocation and runtime configurations, it can then estimate the execution, upload, download, and queue time of the same computational model under different resources and application configurations. An optimization goal for site recommendation is to minimize the total waiting time experienced by users for job completion ($T_{total} = T_{exec} + T_{upload} + T_{download} + T_{queue}$). Site recommendation involves two primary considerations: (1) determining the optimal site for job launch, and (2) allocating resources, specifically the number of CPU cores. While the choice of site affects the upload/download time, considering both site choice and resource allocation impacts queue time and execution time. Within CyberWater, the site recommendation incorporates factors such as the availability and compute capacity across geologically dispersed computing resources. Below, we outline the current estimations of the four time components within CyberWater.

- **Execution time** ($T_{exec}$) of a computational model (e.g., DHSVM) at a specific site is modeled by Eq. (1) as below:

$$T_{exec} = T_{ref} \times \frac{24 \times sim\_days \times sim\_cells}{nr\_cores} \quad (1)$$

  This simplified linear model assumes that simulation execution takes less time when the problem size is smaller (fewer sim_days or sim_cells), or when there are more computing resources (larger nr_cores). The reference execution time $T_{ref}$ is calculated based on the execution history of the computational model (e.g., DHSVM) application at the target site. More advanced execution-time modeling may be incorporated into our tool in the future.
  Even if the application has never been executed at a target site but has execution histories on other sites, CyberWater can still provide realistic predictions of the execution time. Initially, it estimates the $T_{ref}$ based on the execution histories on the other sites and then iteratively refines the prediction for the target site as more performance data become available. These cross-site execution time prediction capabilities are elaborated further in Fig. 12.
- **Upload/Download time** ($T_{upload}$ / $T_{download}$) is estimated in a similar way, but without considering the effect of nr_cores in Eq. (1).

- **Queue time** ($T_{queue}$) estimation in *LaunchAgent* is implemented using Slurm's "srun –test-only" feature. For any job with a specific allocation configuration (e.g., number of nodes, choice of partition, and walltime), this specific Slurm feature estimates the job start time based on the current job queue information on remote systems. We acknowledge that such estimation can be biased due to the effects of early-terminated jobs, job queue back-filling, and job reservations, but this is the only generally accessible method at present to the best of our knowledge.

The devised predictive function of users' job execution time generalizes across different HPC sites and varying job workloads. While we utilize a Slurm feature for queue waiting time estimation, we note that, for computationally expensive tasks, their execution time typically may constitute a significantly more substantial portion of the total time compared to queue time.

The main logic of the site recommendation works by iterating all candidate sites and estimating the ($T_{exec}, T_{upload}, T_{download}, T_{queue}$) when the same application is configured with different numbers of cores. The choice of the number of cores starts at 1, and gets doubled each time (e.g., 2, 4, 8, ...), until:

1. The core number reaches the maximum number of allowed cores for this application (the "max_cores" is from the app_constraints field in Table 2), or
2. The core number reaches the number of available cores of the selected site, or
3. The execution time no longer decreases (using more cores can no longer speed up the application due to the limited scalability)

After all candidates are tried, a list of all possible plans will be generated, where the plan with the shortest total time ($T_{total} = T_{exec} + T_{upload} + T_{download} + T_{queue}$) will rank the highest and is ready to be launched. Note that *LaunchAgent* currently also support other ranking preferences such as minimizing normalized service units (NUs) based on XSEDE SU converter,[13] this allows researchers to make the best use of their ACCESS/XSEDE allocations when considering service units between different systems are transferable.

Fig. 12 shows a demonstration of the execution time predictions for 4 subsequent DHSVM jobs in two sites (IU BigRed3 and PSC Bridges-2). The four jobs are all configured for the same geographic area with the same amount of cells (sim_cells = 30 686). Job 1 is launched in the BigRed3 HPC system for the simulation of the first 6 months of 2010 (sim_days = 181, site_name = "BigRed3"), and Job 2, 3, and 4 are launched in a different HPC system for a longer full-year period (sim_days = 365, site_name = "Bridges2"). For each job, we plot the predicted execution time for the target HPC systems in dashed lines, and the actual execution time ($T_{exec}$) in bars. Job 1 is the only job that has no predicted execution time, since it is the first DHSVM job that *LaunchAgent* has recorded and there are no previous DHSVM execution histories available to make predictions. For Job 2, even though the DHSVM model is configured at a different site with a different and longer simulation period, *LaunchAgent* is able to make a relatively accurate execution time prediction (288s vs. the actual 290 s execution time for Job 2) based on the execution history of Job 1. The prediction is based on the execution time modeling shown in Formula (1) which considers the effects of core speeds of different sites, and is capable of predicting the performance of various simulation configurations. The $T_{ref}$ in Formula (1) also gets updated when there are more execution histories of the same application available: both Job 3 and 4 are also submitted to Bridges2 with the same configuration as Job 2. The increasing collected execution data makes the predictions on Bridges2 more reliable over time.

[13] XSEDE SU converter: https://portal.xsede.org/su-converter. A newer version of the converter is now accessible at https://allocations.access-ci.org/exchange_calculator.


**Fig. 12.** Demonstration of execution time prediction with *LaunchAgent*. Job 1 is a half-year DHSVM execution on BigRed3, and job 2,3,4 are full-year DHSVM execution on Bridges2.

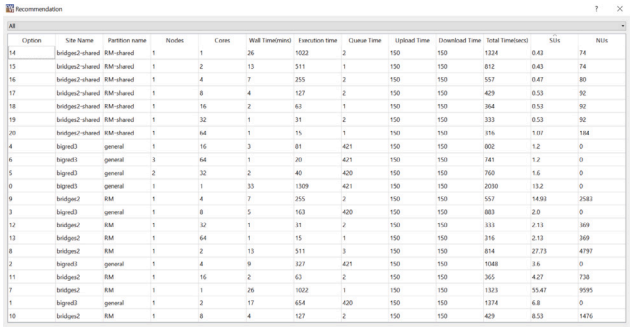


**Fig. 13.** The recommendation system interface for 3 platforms including bridges2-shared, bridges2, and BigRed3 for VIC5 model sorted by SUs ascendingly.

With the recommendation function integrated with the CyberWater frontend, users can choose the best site/platform. Fig. 13 shows the recommendation list for 3 platforms by demonstrating each site's name, the partition name, the number of nodes applied, the number of cores applied, the wall time in minutes, the queue time, the upload time, download time, total time, and SUs and NUs. Users can sort the list by a certain column by simply clicking the header tab, and then select the option by clicking the row where the chosen option locates, e.g., users can click the "SUs" (service units) header tab to get the options list sorted by service units charged as Fig. 13.

In Fig. 13, the "Wall Time" is the Slurm walltime to allocate for the remote execution. We currently configure this walltime to be 1.5 times the estimated execution time, so that applications are less likely to be terminated early due to under-estimated execution time. Also, *LaunchAgent* checked resource plans ranging from 1 to 64 cores accross various sites. Although VIC5 is capable of utilizing more than 64 cores, *LaunchAgent* determined that using additional cores did not further reduce execution time (measured in minutes). Consequentently, *LaunchAgent* stopped checking with more than 64 cores.

## 4. Use case and performance analysis

### 4.1. Use case

In this section, we demonstrate how CyberWater framework's convenient and on-demand access to HPC/Cloud systems works through real-world hydrological modeling workflow examples. We then focus on the comparisons of computational performance for synchronous versus asynchronous modeling workflows in different local/HPC execution environments.

CyberWater offers various data agents for users to retrieve diverse forcing data online, generic model agent toolkit to execute user's

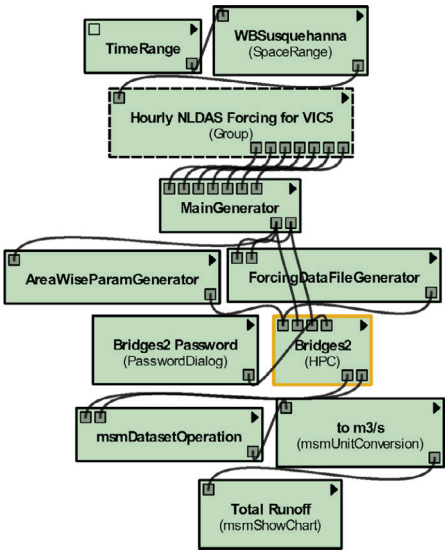



**Fig. 14.** The overall HPC-enabled workflow running VIC5 module in Bridges2 platform with one-year data of the WBSusquehanna river basin.

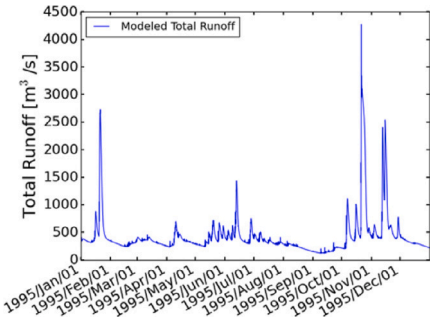



**Fig. 15.** The resulting VIC5 simulated total runoff for the West-Branch Susquehanna river basin.

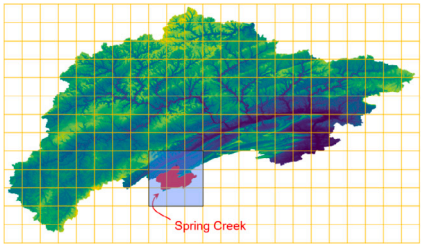



**Fig. 16.** The case study space range for the VIC5 model and DHSVM model, the yellow grids indicate the spatial area for the VIC5 model and the blue square marks the space boundary for the DHSVM model.

models either locally or remotely on High-performance computing platforms, and visualization utilities to display the model simulation results. Note that users interact CyberWater with its graphical user interfaces (GUI) in their local desktop/laptop workflow environments. By utilizing the *HPC* module in CyberWater, we can launch the selected tasks on remote HPC systems. Importantly, CyberWater allows users to integrate the input/output of such on-demand remote execution seamlessly with the local workflow management system. Also, it is convenient and desirable to run the user's model locally by using *RunModuleAgent* instead of the *HPC* module in the CyberWater generic model agent toolkit, if the user's model does not need much computation power during the model prototyping and debugging stages.

#### 4.1.1. Synchronous HPC on-demand workflow

To demonstrate on-demand access to HPC in the CyberWater framework, we first compose a synchronous hydrological modeling workflow through CyberWater GUI. This workflow uses the VIC5 model to study the West Branch Susquehanna[14] river basin for the period 1995–1996. This study area covers more than 17,700 square kilometers.

As shown in Fig. 14, we define the time/space range of the studied problem by configuring the *TimeRange* and *SpaceRange* module for the WBSusquehanna river basin. Then, from the NLDAS (North American Assessment-Land Data Assimilation System)[15] [22], we use the Hourly NLDAS Forcing for VIC5 group module, which internally contains an *NLDASAgent* data agents to fetch 7 variables including temperature, longwave radiation, shortwave radiation, precipitation, pressure, water vapor pressure, and wind speed, and the corresponding unit conversions. This way, CyberWater will pull the seven chosen types of datasets of the specified time/space range from the NLDAS site to the local cache directories. After that, we use several "Generator" modules to prepare the forcing data and parameter files for the VIC5 execution. Then, the *HPC* module is used to launch the VIC5 hydrological model to the PSC Bridges-2 system. During the remote launch, all prepared data are sent to the remote HPC/Cloud system; the VIC5 model is then executed in the remote HPC/Cloud environment; and, finally, all output results are downloaded to the specified local output directory. This entire launch process is conducted automatically by the execution of the *HPC* module in the workflow.

CyberWater also provides post-analysis and visualization modules, such as the *msmShowChart* module. The *msmShowChart* module is used to display useful information such as surface runoff and baseflow for view. For example, Fig. 15 shows the total runoff, which is the sum of baseflow and surface runoff time series of the studied example workflow. The "baseflow" means the portion of the streamflow that is sustained between precipitation events, and it is contributed by slowly moving water within the porous media due to soil moisture or groundwater. "Surface runoff" describes the excess amount of water from rain, snowmelt, or other resources that move over the land surface. The *msmDatasetOperation* module in Fig. 14 sums the baseflow and surface runoff computed from VIC5 to obtain the total runoff.

#### 4.1.2. Asynchronous HPC on-demand workflow

In this section, we demonstrate the asynchronous modeling workflow capability offered by CyberWater through constructing and executing the VIC (version 5) model and The Distributed Hydrology Soil Vegetation Model (DHSVM) in two different High-Performance Computing platforms respectively, and parallelly. In this use case, the river basin selected to illustrate is the West-Branch Susquehanna (WBS) river basin, which covers more than 17,700 square kilometers including 299 modeling cells with 5-year data. The case study area for the DHSVM model is a subarea of the one used for the VIC5 model, as Fig. 16 displays, and the *ForcingDataFileGenerator* module of the generic model agent toolkit can implement the extract of the subrange of the DHSVM model from the original space range for the VIC5 model.

Workflow construction initializes with a *TimeRange* module for a simulation between 2010/01/01 00:00:00 (timeini) and 2015/01/01 00:00:00 (timeend). Since this simulation is hourly, it requires 43,825 data files as each forcing data file represents one hour (i.e., one-time step) covering the entire watershed with 299 modeling cells at a spatial resolution of 1/8 degree per cell. In other words, each forcing data file is a map with 299 cells representing one hour time step to be retrieved from NASA; then add a *SpaceRange* box with the limits:

[14] USGS information used: https://waterdata.usgs.gov/pa/nwis/uv?siteno=01553500.

[15] The NASA Goddard Earth Sciences Data and Information Services: https://ldas.gsfc.nasa.gov/nldas/v2/models.

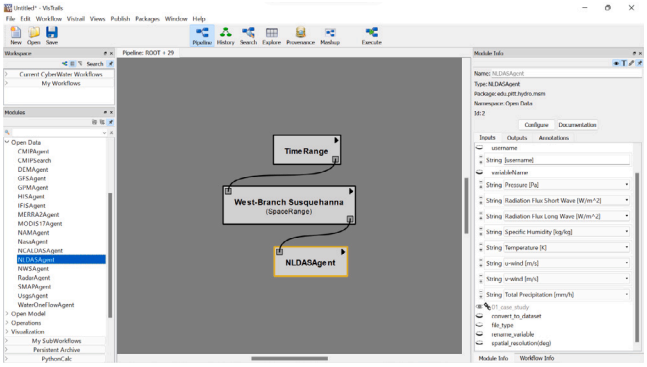



**Fig. 17.** Workflow used to bring all forcing data required to run the VIC5 model and DHSVM model.

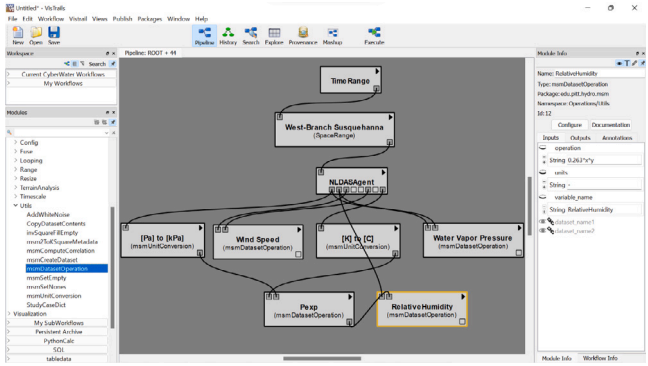



**Fig. 18.** Adding *msmUnitConverison* modules and *msmDatasetOperation* modules to perform data unit conversions and mathematic operations on datasets.

−76.213, −78.9155, 41.933, and 40.454 (x_max, x_min, y_max, y_min respectively) to specify the four bounds of the study area.

*NLDASAgent* module is used to access the forcing data required for the simulation from the NASA Earth database. An *NLDASAgent* module is added to bring the following forcing variables: Temperature [K], Pressure [Pa], Radiation Flux Long Wave [W/m$^2$], Radiation Flux Short Wave [W/m$^2$], Total Precipitation [mm/h], Specific Humidity [kg/kg], u-wind [m/s], and v-wind [m/s]. The resulting workflow is shown in Fig. 17.

For the VIC5 model, the forcing data input should be in specific units determined by the individual variables. In this scenario, the temperature must be in Celsius, and the unit of pressure should be kilo Pascal, and for this case study, the total precipitation will need to be in meters per hour. Thus, users need to check whether the units of the data retrieved from the data sources meet the requirements of the executable model. If not, unit conversion is necessary, and the *msmUnitConversion* module is provided in CyberWater to perform the unit transformation. For instance, to convert the temperature from the NASA Earth database in Kelvin to Celsius, the user needs to configure the input parameter "operation" in the *msmUnitConverison* module as "$x - 273.15$". This "operation" setting allows users to conduct mathematical formula operations using "x" as the variable representing the dataset given in the input port. Similarly, the "operation" can be set into "$x/1000$" to transform the pressure from Pascal to kilo Pascal and filling "operation" as "$x/1000$" to convert the total precipitation from millimeters per hour to meters per hour via other two *msmUnitConverison* boxes. CyberWater offers the *msmDatasetOperation* module to facilitate mathematical operations between two datasets, which can be applied in computing the magnitude of the wind speed based on the information of the two components associated with two directions of the wind speed $(x2 + y2)/2$, where x and y represents U-Wind and V-Wind separately obtained by *NLDASAgent* module. Additionally, we can also use the *msmDatasetOperation* module to calculate the Water Vapor Pressure which is required for the energy-balance model of the VIC model, by performing the operation $(x * y)/0.62$, where x is the Specific Humidity [kg/kg], and y is Pressure [PA]. Besides, Pexp as a needed forcing variable can be output via the *msmDatasetOperation* module with the equation below:

$$Pexp = \frac{x}{e^{\left(\frac{17.3y}{y+237.3}\right)}}, \tag{2}$$

where x is Pressure [kPa], y is Temperature [°C] read from the antecedently connected *msmUnitConverison* module boxes, and e is a mathematical constant approximately equal to 2.7183 for here. The last forcing variable necessary for VIC model is Relative Humidity, which can be obtained by conducting the operation "$2.63 * x * y$", in which x and y represent Specific Humidity [kg/kg] retrieved by *NLDASAgent* module and the Pexp variable yielding from *msmDatasetOperation* module, and the resulting workflow as Fig. 18 shows.

*4.1.2.1. MainGenerator.* To start constructing the VIC5 model with generic model agent toolkit, bring a *MainGenerator* module to set up the working directory for the model simulation where it receives all the forcing datasets as inputs. In addition, all forcing data are collected for VIC5 model by this *MainGenerator* module. The working directory path will be passed through the output port WD_Path to all the following connected components in the generic model agent toolkit to construct the VIC5 model.

*4.1.2.2. AreaWiseParamGenerator.* Add an *AreaWiseParamGenerator* to set up the parameter files (e.g., soil parameter files and vegetation parameter files) for the model. For the illustration in this example, these files need to be created by the user before using the *AreaWiseParamGenerator* module in advance.

*4.1.2.3. ForcingDataFileGenerator.* Add a ForcingDataFileGenerator, responsible for the creation of the forcing data files of the model. This component takes the forcing information gathered from *Data Agents* and the *MainGenerator*, and saves it into the user's specified folder.

*4.1.2.4. InitialStateFileGenerator.* Add an *InitialStateFileGenerator*. This module is responsible for placing the initial state files in the right folder in the working directory.

*4.1.2.5. HPC.* Add an *HPC* module, which is responsible for accessing remote High-Performance Computing (HPC) facilities on demand. It retrieves the results back to the workflow when the execution of the user's model on the remote HPC platform is completed. Users need to select the HPC platform and its corresponding credentials and configure the executable model and arguments for execution, and formats of the resulting data file.

Generally, researchers have access to two types of remote HPC resources: academic HPC clusters such as Bridges2, BigRed3, etc., and commercial cloud computing sources like Google Cloud Platform. With the HPC module, users can conveniently select the viable HPC platform from the drop-down list in the input panel of the HPC module as Fig. 19 shows, either through the SSH direct connection channel or Airavate gateway-based channel. Furthermore, the *HPC* module also offers the flexibility for users to add and adopt their own new HPC platform by simply providing the IP address or domain name of their customized platform. Then the *HPC* module will store the platform configuration and list this newly added platform in the drop-down list for the user to choose in the next run. The *HPC* module can only be conducted when all of its inputs are ready, which is guaranteed by connecting the output port of the three previous components in Section 4.1.2.2, Section 4.1.2.3, and Section 4.1.2.4 to the Ready_List input port of the *HPC* module.

Similar to unit conversion for the original forcing data file, the user can add two *msmUnitConverison* modules to convert the units of datasets of variables "Surface Runoff" and "Baseflow" separately. To

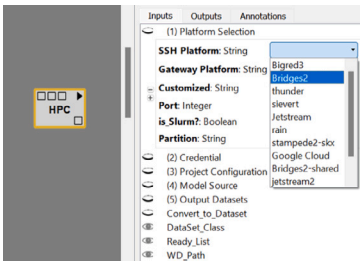



**Fig. 19.** The input panel configuration interface of the *HPC* module of the generic model agent toolkit.

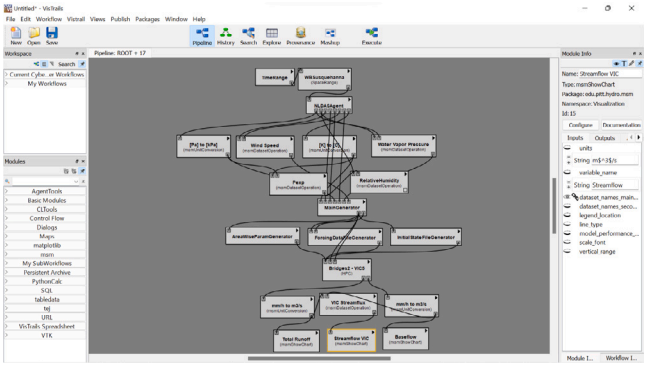


**Fig. 20.** Workflow to execute VIC5 model on Bridges2-share HPC platform and visualize the resulting datasets.

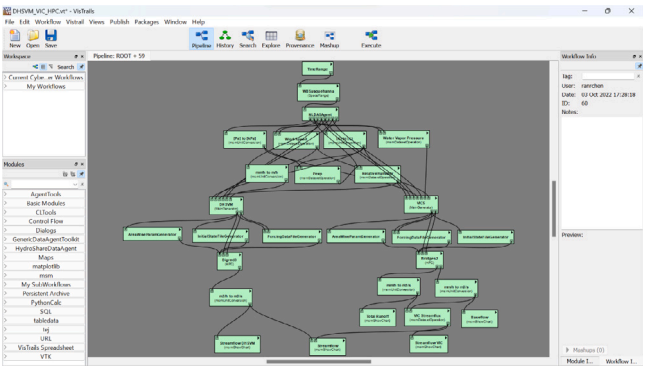


**Fig. 21.** Overall workflow to execute VIC5 model on Bridge2 HPC platform and DHSVM model on BigRed3 platform asynchronously.

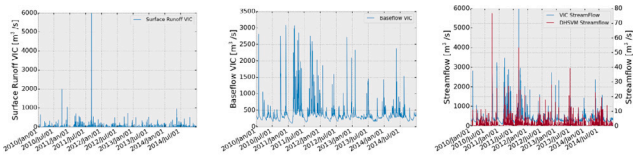



**Fig. 22.** **(a)** The resulting surface runoff predicted by the VIC5 model in WBSusquehanna retrieved from Bridges2-shared platform, **(b)** The resulting baseflow predicted by the VIC5 model in WBSusquehanna retrieved from Bridges2-shared platform, and **(c)** The comparison between the resulting streamflow predicted respectively by VIC5 model (left y-axis) from Brdiges2-shared platform and by DHSVM model (right y-axis) from BigRed3.

compute the "Streamflow" variable, the *msmDatasetOperation* module can be used again to perform a mathematical operation "x + y", where x represents Surface Runoff and y is Baseflow, both in cubic meters per second. In addition, for the purpose of visualizing the resulting datasets computed from the VIC5 model by adding three *msmShowChart* modules to create time series charts for the resulting datasets of "Surface Runoff", "Baseflow" and "Streamflow" by connecting the "resulting_dataset_name" output port from the two previous *msmUnitConverison* modules for "Surface Runoff" and "Baseflow", and the "resulting_dataset_name" output port from the *msmDatasetOperation* module for "Streamflow", with the input port of "dataset_names_mainAxis" in three *msmShowChart* modules respectively, and the current workflow is shown in Fig. 20.

To construct the DHSVM model with the generic model agent toolkit, repeat the steps described in Section 4.1.2.1 to Section 4.1.2.5 to add modules including *MainGenerator*, *AreaWiseParamGenerator*, *ForcingDataFileGenerator*, *InitialStateFileGenerator*, and *HPC* module in the workflow for DHSVM model simulation.

Likewise, to convert the unit for the resulting variable "Streamflow" calculated by DHSVM model, the user can now add an *msmUnitConverison* model and configure the "operation" parameter as "x/3600", where x represents Streamflow in cubic meters per hour to transform the new unit into cubic meter per second. To visualize the resulting datasets computed by the DHSVM model, the user can add an *msmShowChart* module to create time series charts for the resulting "Streamflow" dataset. This is done by connecting the "resulting_dataset_name" output port from the first *msmUnitConverison* module, which processes the "Streamflow" data calculated by the DHSVM model. The overall workflow is depicted in Fig. 21. The resulting surface runoff, baseflow predicted by the VIC5 model brought from Bridges2-shared platform and the comparison between the resulting streamflow predicted by the VIC5 model and the DHSVM model from BigRed3 are plotted as Fig. 22. Results shown in Figs. 15 and 22 are based on default parameters with VIC5 and DHSVM models without any calibration.

### 4.2. Performance analysis

The performance of workflow-parallelism of Cyberwater is evaluated by comparing the elapsed time of executing the *HPC* modules synchronously and asynchronously. In this section, we present the performance experiments on the modeling use case described in Section 4.1.1 with different synchronous versus asynchronous workflow setups, to analyze the performance advantage offered by asynchronous workflow capability within CyberWater framework.

Fig. 23 presents the statistical time consumption for each step, clearly illustrating the performance differences among the local runs, synchronous remote executions, and asynchronous remote executions. Firstly, we start with the local run of VIC5 and DHSVM models in synchronous workflow in CyberWater. The baseline runtime environment for our operations is a Windows 10 local PC machine with 16 GB of RAM. Loading and reading 8 cached raw forcing datasets, each exceeding 100 MB in size and spanning 5 years of data in the WBSusquehanna watershed area, retrieved by the *NLDASAgent* module (illustrated in Fig. 23 under the legend 'Loading Cached Forcing Dataset), takes proximately 15 min. Following that, the *ForcingFileGenerator* module of CyberWater generates the processed forcing data files required for the VIC5 model in an additional 10 min, as indicated in Fig. 23 under the label 'Generating Forcing Data Files for VIC5 Model'. These initial stages of model preparation, involving loading the dataset cache and generating the forcing data file, are consistently performed on the user's local machine. Transferring these preparatory steps to an HPC platform, while marginally enhancing the performance, would impose additional computational demands on the HPC system. To optimize resource utilization, we allocate only the model execution phase to the HPC platform, while retaining the model preparation processes on local machines, to maximize the utilization of local desktop computing capacities. Subsequently, the *RunModuleAgent* module is used in the local run scenario to execute the VIC model binary code in CyberWater, consuming a total of 41 min for the entire VIC5 execution and

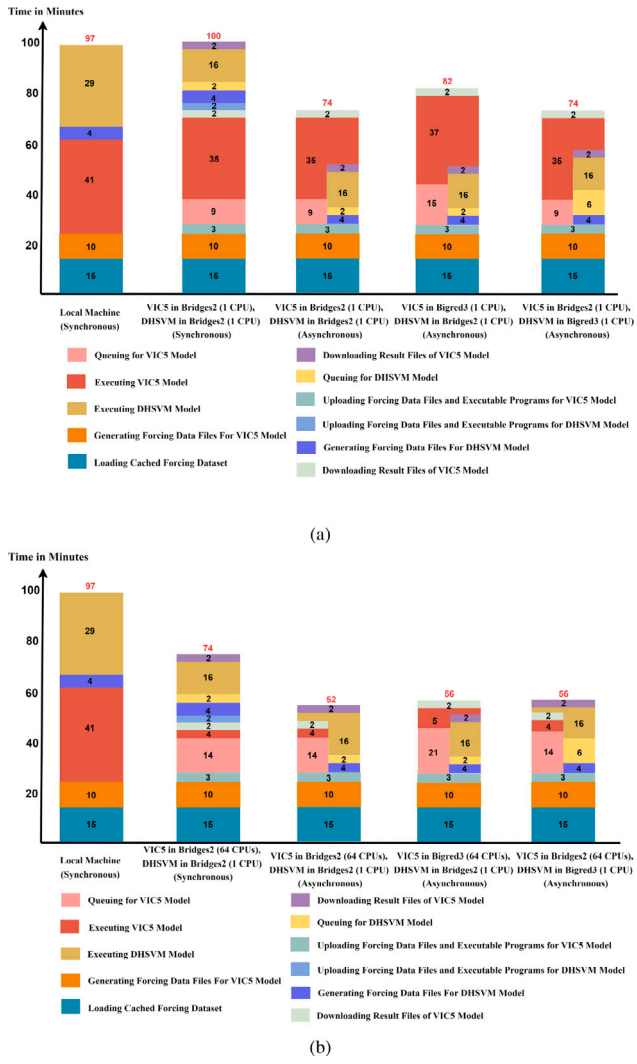



**Fig. 23.** **(a)** Time breakdown for the workflow coupled with the VIC5 model running with 1 CPU and the DHSVM model running with 1 CPU in different HPC setup environments. **(b)** Time breakdown for the workflow coupled with the VIC5 model running with 64 CPUs and the DHSVM model running with 1 CPU in different HPC setup environments.

66 min for the first VIC model execution. Following this, the second *ForcingDataFileGenerator* module generates the forcing data files accustomed to the DHSVM model within 4 min, representing a significantly smaller space range as a subset of the previous date. The inclusion relationship between the two space range sets is depicted in Fig. 16 in Section 4.1. The second *RunModuleAgent* module executes DHSVM, requiring 29 min for the DHSVM execution. Thus, the overall elapsed duration of the use case of synchronous workflow running on the local machine amounts to 97 min, with model execution constituting a substantial proportion. Next, to establish a baseline of on-demand access to HPC in the synchronous workflow, the *RunModuleAgent* modules are replaced by *HPC* modules within this workflow, facilitating the offloading and execution of the VIC5 model and DHSVM model in Bridges2 HPC system. Despite the time required for uploading inputs, waiting for remote execution, and downloading result outputs, the elapsed time to run the model is reduced to 35 min and 16 min for the VIC5 model and DHSVM model. Notably, employing 64 CPU cores in a node for VIC 5 model execution on the Bridges2 platform yields approximately 9 times speedup. As a result, the total execution time for the synchronous workflow involving both the VIC5 and DHSVM models has decreased by a factor of 2.5.

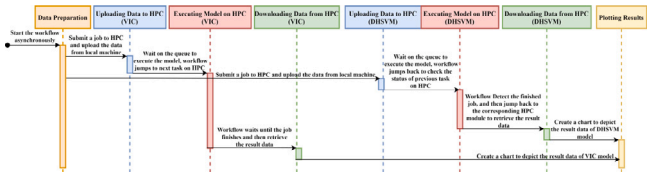



**Fig. 24.** The sequence diagram illustrates the mechanism of how the workflow engine executes the modules synchronously.

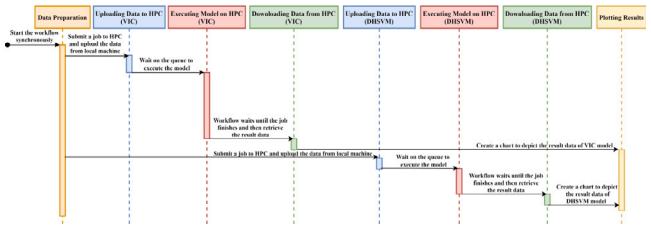



**Fig. 25.** The sequence diagram illustrates the mechanism of how the workflow engine executes the modules asynchronously.

While introducing on-demand access to HPC in CyberWater reduces the overall workflow execution time, further enhancements can be made by extending the LaunchAgent tool to a non-blocking version for job submission and implementing a pipeline parallelism mechanism in CyberWater to execute *HPC* module(s) asynchronously. Fig. 24 shows, in a straightforward way, how each module works and collaborates within the CyberWater framework in a synchronous workflow: the entire cycle for a model to run remotely with the *HPC* module includes the model's input data uploading, queuing, executing, and output results downloading; the next model can be processed only when the whole cycle of the former model finishes. In contrast, the asynchronous workflow of CyberWater enables to offload of multiple models to HPC site(s) in an asynchronous manner without blocking the workflow, and then to execute these models on HPC in parallel, where monitoring the status of remote jobs with the JobManager component periodically. As indicated in Fig. 25, the newly extended workflow engine will move forward to data preparation for the successive model after it submits the previous job to the HPC platform. Therefore, the time for queuing and executing the former model overlaps with the time for preparing the data, uploading the input, submitting, queuing, and executing the job for the following model. To demonstrate, we conduct the third experiment with CyberWater's new asynchronous workflow for on-demand access to Bridges2 HPC system for offloading the VIC5 model and DHSVM model execution. As shown in Fig. 23(b), we observe that when the VIC5 model task waits to start and executes, the second model, DHSVM, generates its forcing data files, uploads the inputs, queues for allocation, and begins computation in the meantime, which results in the overall elapsed time of the entire workflow drops to 52 min from the previous 74 min consumed using the synchronous workflow with the same HPC resource configurations, achieving 1.42 times speedup.

The results of the last two experiments illustrate minor overall performance variations of the same asynchronous workflow using two different HPC platforms, which is mainly due to the difference in queuing time among various HPC platforms. In the fourth experiment, we run the VIC5 in the BigRed3 system, which also includes 64 cores in a node, but always needs more queue time, in this situation, the total time for the VIC5 model running in the HPC platform increases.t As a result, the DHSVM model finishes earlier, which causes the workflow engine turns to the other sub-branch where DHSVM model locates and downloads the result files of DHSVM model from Bridges2 platform, and then hangs on until the VIC5 model job status becomes "completed". This case costs 56 min, 4 min longer than the one to run the

models on the Bridges2 platform in the third experiment. The last one switches the HPC platforms for two models, i.e., the DHSVM model is offloaded to BigRed3 platform, and the VIC5 model is offloaded to Bridges2 platform. In general, the new asynchronous workflow in CyberWater allows independent offloaded models to be executed in parallel with on-demand HPC access, which significantly improves the performance of the corresponding synchronous workflow with the same HPC resource configurations.

## 5. Related works

An overview of related works and tools is helpful for understanding the landscape and highlighting the key distinctions between our proposed framework and existing "state-of-the-art" tools. Notable among these tools are FirecRest [23], NEWT [24], PSI/J [25], Globus GRAM [26], SAGA [27], DRMAA [28], and GISandbox [29], each contributing to the evolving field of HPC.

FirecREST, developed by the Swiss National Supercomputing Centre (CSCS), is a RESTful Web API designed to enhance the accessibility of HPC resources for communities using web applications. While sharing similarities with NEWT in principle, FirecRest differs in its architecture design and API implementation. PSI/J is an open, language-agnostic Job Abstraction API (JAAPI), which enables the portability of HPC applications across diverse HPC platforms with varying scheduler implementations. Other significant JAAPI efforts include Globus GRAM, SAGA, and DRMAA. GISandbox, a science gateway constructed upon Jupyter Notebooks, is specifically tailored for geospatial computing. Its objective is to empower users to execute notebooks on cloud or supercomputer resources. These existing works within HPC communities predominantly focus on developing "state-of-the-art" tools with three main objectives: (1) Web API Accessibility: Offering Web APIs to facilitate user access to HPC resources. (2) Portability: Enabling the portability of HPC applications across different HPC systems and schedulers. (3) Science Gateway Approach: Providing a science gateway approach for users to adopt HPC as their computing platform within specific domains.

It is important to note that these tools are primarily centered around HPC platform sites, where workflows are executed on HPC platforms, and schedulers operate on HPC sites. Additionally, the tools utilizing JAAPI do not directly engage with the utility of users' local desktops/laptops. This is a key distinction from our proposed framework, which places a primary emphasis on the desktop side. Our work takes a user-centric perspective, concentrating on computing systems with a desktop orientation. This distinctive approach establishes a new computing ecosystem that leverages both local and HPC resources seamlessly, diverging from the prevalent HPC-oriented paradigm emphasized by "state-of-the-art" tools. Within this novel computing framework, HPC resources are accessed on-demand exclusively for computationally intensive tasks (e.g., workflow items). The entire workflow, however, is orchestrated and overseen by a desktop workflow system. Our CyberWater framework incorporates non-blocking LaunchAgent, asynchronous workflow control, and JobManager, residing on desktops/laptops. While these functionalities, such as job submission and status monitoring, parallel those found in existing HPC tools, they offer additional benefits, such as supporting offloading computationally expensive operations from local computers to remote HPCs.

The limitations of prevailing HPC-oriented approaches in existing tools are twofold:

1. Underutilization of Local Desktop Computing Capacity: Existing tools often fail to leverage the computing capacity of users' ubiquitous local desktop computers effectively. Access to HPC/Cloud resources is not cost-free and is typically tied to project funding, raising sustainability concerns. Our approach advocates accessing HPC on-demand solely for computationally expensive tasks, allowing the remaining workflow to be executed on desktops. This optimizes the use of HPC resources, directing saved computing power towards demanding tasks within the HPC community. Notably, certain workflow items, such as GUI-based configurations, data visualization, plotting, and interactive computing, are better suited for desktops than HPCs. CyberWater also facilitates reproducible computing, with provenance information managed more conveniently and efficiently on users' desktops.
2. Restricted Accessibility of HPC-Oriented Tools: Present HPC-oriented tools are confined to specific HPC sites, limiting their accessibility to users who have access to those particular supercomputers. In contrast, our new computing ecosystem imposes no specific requirements on HPC sites. CyberWater extends flexibility to users by providing a diverse array of HPC platform options, including Bridges2, Stampede2, Jetstream from ACCESS, and Google Cloud. Users can even add new HPC facilities themselves, fostering adaptability and workload balance across various HPC resources.

Furthermore, unlike existing tools, our CyberWater framework introduces a novel feature: site recommendation based on predicting user task execution time. This assists users in selecting the most appropriate HPC platform for their specific tasks, thereby optimizing performance/cost ratios. This unique capability sets CyberWater apart from current "state-of-the-art" HPC tools. In summary, our CyberWater framework is a valuable complement to existing HPC-oriented tools, offering enhanced flexibility, accessibility, sustainability, and predictive site recommendation for optimal performance and cost efficiency.

## 6. Conclusions

This paper presents our novel work for CyberWater, a framework system for open data and model integration, to enable seamless on-demand access to HPC platforms via an asynchronous workflow mechanism developed on a desktop workflow computing system. In this framework, HPC modules in the workflow are processed asynchronously, enhancing the performance of the CyberWater workflow system through workflow parallelism. In a nutshell, we design and develop the LaunchAgent tools in CyberWater to utilize either a direct Slurm-based channel or gateway-based SciGaP channel to generally offload computational expensive tasks to various high-performance computing resources selected by users. We introduce the JobManager into CyberWater and extend the original VisTrails synchronous workflow mechanism into an asynchronous workflow mechanism to significantly improve the performance of independent models offloaded to remote HPC platforms at workflow level. The JobManager is responsible for re-queuing the modules in the workflow pipeline to facilitate asynchronous workflow scheduling and control in CyberWater. A GUI-based HPC module is developed as a component of the generic model agent toolkit in CyberWater for users to construct their workflow and submit the job to the remote HPC platform on demand effortlessly and effectively, without users' writing any HPC scripts/codes or knowing any specific HPC platforms' operation details. Furthermore, an HPC site recommendation system has been designed and developed to help users select the optimal remote HPC platform and configuration, aiming to save time and/or reduce costs. To the best of our knowledge, this study represents the first work of its kind in open data and modeling framework with general on-demand access to HPC resources through asynchronous workflow, facilitated by intelligent HPC site recommendation. We believe that these unique system features characterize the future workflow computing systems to form a new computing ecosystem where desktops/laptops can seamlessly access HPC resources on demand in a general and systematic manner. Such a future computing ecosystem will not only take advantage of user-friendly and rich GUI functionalities and ubiquitous computing power of desktops/laptops, but also offer valuable HPC resources to those truly computational expensive jobs on an on-demand basis, significantly

enhancing the usability, scalability, and sustainability of the overall computing ecosystem composed of desktops/laptops and various HPC resources. We are currently working on more data-intensive computational tasks and expanding the functionality of the LaunchAgent utility to submit multiple tasks within a job to further optimize the performance of models for repetitive computations over modeling grids with CyberWater framework.

The CyberWater framework, designed to address the unique modeling challenges of water science and engineering, introduces innovative methods and techniques. Particularly, its approach of establishing an efficient and sustainable new computing ecosystem, harnessing the computational power of users' personal desktops/laptops with on-demand access to high-performance computing (HPC)/Cloud resources, offers a versatile solution applicable beyond the realm of water science. The methodologies honed within the CyberWater framework, including asynchronous workflow control, LaunchAgent, JobManager, and HPC site recommendation on desktop workflow system, possess inherent adaptability and can be readily extended to other desktop computing systems for diverse domains. For instance, the integration of user-owned computing resources with dynamic and on-demand access to HPC/Cloud capabilities holds promise for facilitating scientific computing in fields such as agriculture, transportation, biology, and medicine. By leveraging the principles developed in CyberWater, organizations and researchers in various domains stand to benefit from a robust, scalable, and sustainable computing infrastructure.

## CRediT authorship contribution statement

**Ranran Chen:** Writing – review & editing, Writing – original draft, Validation, Software, Investigation, Formal analysis. **Feng Li:** Writing – review & editing, Writing – original draft, Validation, Software, Investigation, Formal analysis. **Daniel Luna:** Software, Investigation, Formal analysis. **Isuru Ranawaka:** Software, Investigation. **Fengguang Song:** Validation, Supervision, Investigation, Formal analysis. **Sudhakar Pamidighantam:** Supervision, Investigation, Formal analysis. **Xu Liang:** Supervision, Project administration, Investigation, Funding acquisition, Formal analysis. **Yao Liang:** Writing – review & editing, Writing – original draft, Supervision, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

Data used are publicly available.

## Acknowledgments

This work was supported by the U.S. National Science Foundation under [OAC-1835817, OAC-2209835] to Indiana University-Purdue University Indianapolis (IUPUI), United States, and [OAC-1835785, OAC-2209833] to the University of Pittsburgh, respectively.

## References

[1] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M. Su, K. Vahi, M. Livny, Pegasus: Mapping scientific workflows onto the grid, Lecture Notes in Comput. Sci. 3165 (2004) 11–20.

[2] D. Thain, T. Tannenbaum, M. Livny, Distributed computing in practice: the Condor experience, Concurr. Comput. (2005) 323–356, http://dx.doi.org/10.5555/1064323.1064336.

[3] S.P. Callahan, J. Freire, E. Santos, C.E. Scheidegger, C.T. Silva, H.T. Vo, Vistrails: visualization meets data management, in: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, 2006, pp. 745–747, http://dx.doi.org/10.1145/1142473.1142574.

[4] A.B. Yoo, M.A. Jette, M. Grondona, SLURM: Simple linux utility for resource management, in: Springer Berlin Heidelberg, 2003, pp. 44–60, http://dx.doi.org/10.1007/109689873.

[5] S. Marru, L. Cunathilake, C. Herath, P. Tangchaisin, M. Pierce, C. Mattmann, R. Singh, T. Gunarathne, E. Chinthaka, R. Gardler, A. Slominski, A. Douma, S. Perera, S. Weerawarana, Apache airavata: a framework for distributed applications and computational workflows, in: Proceedings of the 2011 ACM Workshop on Gateway Computing Environments, 2011, pp. 21–28, http://dx.doi.org/10.1145/2110486.2110490.

[6] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. Peterson, R. Roskies, J. Scott, N. Wilkens-Diehr, XSEDE: Accelerating scientific discovery, Comput. Sci. Eng. 16 (2014) 62–74, http://dx.doi.org/10.1109/MCSE.2014.80, Publisher Copyright: © 2014 IEEE.

[7] D. Salas, X. Liang, M. Navarro, Y. Liang, D. Luna, An open-data open-model framework for hydrological models' integration, evaluation and application, Environ. Model. Softw. 126 (2020) 104622, http://dx.doi.org/10.1016/j.envsoft.2020.104622.

[8] R. Chen, D. Luna, C. Yuan, Y. Liang, X. Liang, Open data and model integration through generic model agent toolkit in CyberWater framework, Environ. Model. Softw. 152 (2022) 105384.

[9] R. Chen, F. Li, D. Bieger, F. Song, Y. Liang, D. Luna, R. Young, X. Liang, S. Pamidighantam, CyberWater: An open framework for data and model integration in water science and engineering, in: Proceedings of the 31st ACM International Conference on Information & Knowledge Management, 2022, pp. 4833–4837, http://dx.doi.org/10.1145/3511808.3557186.

[10] J. Hamman, B. Nijssen, D. Gergel, Y. Mao, The Variable Infiltration Capacity model version 5 (VIC-5): infrastructure improvements for new applications and reproducibility, Geosci. Model Dev. 11 (2018) 3481–3496.

[11] X. Liang, D.P. Lettenmaier, E.F. Wood, S.J. Burges, A simple hydrologically based model of land surface water and energy fluxes for general circulation models, J. Geophys. Res.: Atmos. 99 (D7) (1994) 14415–14428, http://dx.doi.org/10.1029/96JD01448.

[12] X. Liang, E.F. Wood, D.P. Lettenmaier, Surface soil moisture parameterization of the VIC-2L model: Evaluation and modification, Glob. Planet. Change 13 (1–4) (1996) 195–206, http://dx.doi.org/10.1016/0921-8181(95)00046-1.

[13] X. Liang, D.P. Lettenmaier, E.F. Wood, A one-dimensional statistical dynamic representation of subgrid spatial variability of precipitation in the two-layer variable infiltration capacity model, J. Geophys. Res. 101 (D16) (1996) 21,403–21,422, http://dx.doi.org/10.1029/96jd01448.

[14] X. Liang, Z. Xie, Important factors in land-atmosphere interactions: Surface runoff generations and interactions between surface and groundwater, Glob. Planet. Change 38 (1–2) (2003) 101–114, http://dx.doi.org/10.1016/S0921-8181(03)00012-2.

[15] M.S. Wigmosta, L.W. Vail, D.P. Lettenmaier, A distributed hydrology-vegetation model for complex terrain, Water Resour. Res. 30 (6) (1994) 1665–1679, http://dx.doi.org/10.1029/94WR00436.

[16] F. Li, R. Chen, Y. Fu, F. Song, Y. Liang, I. Ranawaka, S. Pamidighantam, D. Luna, X. Liang, Accelerating complex modeling workflows in CyberWater using on-demand HPC/Cloud resources, in: The 17th IEEE EScience Conference, 2021, p. 10, http://dx.doi.org/10.1109/eScience51609.2021.00030.

[17] A. Brown, G. Wilson, The architecture of open source applications, 2011.

[18] A. Benoit, U.V. Catalyurek, Y. Robert, E. Saule, A survey of pipelined workflow scheduling: Models and algorithms, ACM Comput. Surv. 50 (2013) 36, http://dx.doi.org/10.1145/2501654.2501664.

[19] A. Benoit, V. Rehn-Sonigo, Y. Robert, Multi-criteria scheduling of pipeline workflows, in: 2007 IEEE International Conference on Cluster Computing, 2017, pp. 515–524, http://dx.doi.org/10.1109/CLUSTR.2007.4629278.

[20] M. Pierce, S. Marru, E. Abeysinghe, S. Pamidighantam, M. Christie, D. Wannipurage, Supporting science gateways using apache airavata and scigap services, in: Proceedings of the Practice and Experience on Advanced Research Computing, 2018, pp. 1–4, http://dx.doi.org/10.1145/3219104.3229240.

[21] M. Bux, U. Leser, Parallelization in scientific workflow management systems, 2013, CoRR abs/1303.7195.

[22] S.V. Kumar, M. Jasinski, D.M. Mocko, M. Rodell, J. Borak, B. Li, H.K. Beaudoing, C.D. Peters-Lidard, NCA-LDAS land analysis: Development and performance of a multisensor, multivariate land data assimilation system for the national climate assessment, J. Hydrometeorol. 20 (8) (2019) 1571–1593, http://dx.doi.org/10.1175/JHM-D-17-0125.1.

[23] F.A. Cruz, A.J. Dabin, J.P. Dorsch, E. Koutsaniti, N.F. Lezcano, M. Martinasso, D. Petrusic, FirecREST: a RESTful API to HPC systems, in: 2020 IEEE/ACM International Workshop on Interoperability of Supercomputing and Cloud Technologies, SuperCompCloud, 2020, pp. 21–26, http://dx.doi.org/10.1109/SuperCompCloud51944.2020.00009.

[24] S. Cholia, T. Sun, The NEWT platform: an extensible plugin framework for creating ReSTful HPC APIs, Concurr. Comput.: Pract. Exper. (2015) 4304–4317, http://dx.doi.org/10.1002/cpe.3517.

[25] M. Hategan-Marandiuc, A. Merzky, N. Collier, K. Maheshwari, J. Ozik, M. Turilli, A. Wilke, J. Wozniak, K. Chard, I. Foster, R. Ferreira da Silva, S. Jha, D. Laney, PSI/J: A portable interface for submitting, monitoring, and managing jobs, 2023, pp. 1–10, http://dx.doi.org/10.1109/e-Science58273.2023.10254912.

[26] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke, A resource management architecture for metacomputing systems, in: Job Scheduling Strategies for Parallel Processing, 1998, pp. 62–82, http://dx.doi.org/10.1007/BFb0053981.

[27] T. Goodale, S. Jha, H. Kaiser, T. Kielmann, P. Kleijer, G. von Laszewski, C. Lee, A. Merzky, H. Rajic, J. Shalf, SAGA: A simple API for grid applications. High-level application programming on the grid, Comput. Methods Sci. Technol. (2006) 7–20, http://dx.doi.org/10.12921/cmst.2006.12.01.07-20.

[28] P. Tröger, H. Rajic, A. Haas, P. Domagalski, Standardization of an API for distributed resource management systems, 2007, pp. 619–626, http://dx.doi.org/10.1109/CCGRID.2007.109.

[29] E. Shook, D.D. Vento, A. Zonca, J. Wang, GISandbox: A science gateway for geospatial computing, in: Proceedings of the Practice and Experience on Advanced Research Computing, 2018, http://dx.doi.org/10.1145/3219104.3219150.

**Ranran Chen** is a Ph.D. candidate in the Department of Computer and Information Science at Indiana University–Purdue University Indianapolis. She received her Bachelor Degree in Software Engineering in 2018 from Tongji University, China.

**Feng Li** is a Ph.D. candidate at Indiana University-Purdue University Indianapolis. His research works are mainly on high-performance computing and scientific workflow systems, with special interests in performance analysis, modeling, and optimization for in-situ workflows. He works closely with domain scientists from fields such as Computational Fluid Dynamics and Climate/Hydrology research, and his research work helps them accelerate scientific discoveries through efficient computation execution, better data reuse, and strategic resource planning.

**Daniel Luna** receives his Ph.D. in hydrology from the University of Pittsburgh in 2023.

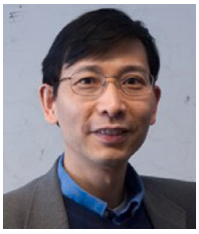
**Isuru Janith Ranawaka** is a full-stack software engineer at Indiana University. He receives his Master degree in Intelligent Systems Engineering (2022) at Indiana University, Master degree in Telecommunication (2017) at Moratuwa University, Sri Lanka, and his Bachelor degree in Computer Science and Engineering (2014) at Moratuwa University, Sri Lanka. He has over six years of professional software development experience and over four years of academic and research experience.

**Fengguang Song** is currently an Associate Professor in Luddy School of Informatics, Computing, and Engineering at Indiana University. He earned his Ph.D. in Computer Science from the University of Tennessee at Knoxville under the direction of Jack Dongarra. After receiving his Ph.D., he continued to work with Jack Dongarra in the Innovative Computing Lab as a Post-doctoral Research Associate (2010-2012), then worked as a Senior Research Scientist in the Computer Science Lab at Samsung Research America–Silicon Valley. Between 2013 and 2023, Dr. Song has been working as an Assistant Professor then as Associate Professor of Computer Science at Indiana University Purdue University.

**Sudhakar Pamidighantam** has been developing and deploying production cyberinfrastructures for various disciplines such as chemistry and physics supported by NSF. He is a project management committee (PMC) member of the Apache Airavata Science Gateway middleware framework. He deployed Chemviz, the chemistry educational portal with integration of NCSA Condor resources during 90s and developed GridChem cyber infrastructure during 2005 and his most recent cyberinfrastructure project is the Science and Engineering Applications Grid (SEAGrid, SEAGrid.org). His research involves Ab initio and molecular dynamics modeling and reaction mechanism, and cyber infrastructure for complex workflows. His interests are in providing production quality infrastructures for multi-disciplinary research using scientific and engineering modeling and simulation workflows.

Sudhakar Pamidighantam obtained his B. Sc. From Madras University and M.Sc. from University of Hyderabad. After spending couple of years at Indian Institute of Science, Bangalore as a UGC Junior Research Fellow he moved to University of Alabama at Birmingham to complete his Ph. D in the department of chemistry. He joined Indiana University in 2014 as a senior scientist in the Science Gateways Research Center in the Pervasive Technology Institute. He has been a consulting and affiliate research scientist for high performance computing and material science applications at the National Center for Supercomputing Applications (NCSA) at the University of Illinois since the last 30 years. He is an affiliate of the IEEE, member of the American Chemical Society, American Association of Advancement of Sciences, and the Association of Computing Machinery.

**Dr. Xu Liang** is Professor of Hydrology in the Department of Civil and Environmental Engineering at the University of Pittsburgh. Her core research spans three main areas: land surface modeling and ecohydrology, hydroinformatics using advanced machine learning methodologies, and cyber system development. She actively engages in interdisciplinary collaboration work with atmospheric scientists, plant biologists, and computer scientists. Professor Liang has been instrumental in the initial and subsequent development of the VIC land surface model and the VIC+ model. She is an elected Fellow of the American Meteorological Society (AMS) since 2016. She is also the recipient of the Chancellor's Distinguished Research Award (senior category) of the University of Pittsburgh in 2016, the recipient of the 2014 Carnegie Science Environmental Award, and the recipient of the Hellman Foundation Junior Faculty Research Award of the University of California at Berkeley in 2000. She held the William Kepler Whiteford Professorship in 2014–2019. Prior to join the University of Pittsburgh, she was faculty at the University of California, Berkeley. Dr. Liang received her Ph.D. in hydrology from the University of Washington (Seattle) and did her postdoctoral work at Princeton University.



**Yao Liang** is a Professor in the Department of Computer and Information Science at Indiana University Purdue University, Indianapolis (IUPUI), USA. His research interests include wireless sensor networks, Internet of Things, cyberinfrastructure, open data and model integration, machine learning, neural networks, data engineering, and distributed systems. His research projects have been funded by NSF, NASA, and DOT. He has received the 2019 Glenn W. Irwin, Jr., M.D., Research Scholar Award, IUPUI. He served as the General co-Chair of The International Conferences on Big Data Engineering (BDE) in 2022, 2021, 2020, and 2019. Dr. Liang has given invited talks and lectures at various universities in US, Europe and China. He is a Senior Member of IEEE, and a Member of ACM. He also had extensive industrial R&D expertise as a Technical Staff Member in Alcatel USA. Yao Liang received his B.S. degree in Computer Engineering and M.S. degree in Computer Science from Xi'an Jiaotong University, China. He received his Ph.D. degree in Computer Science from Clemson University, Clemson, USA, in 1997.