Received: 11 November 2016 | Accepted: 21 February 2017
DOI: 10.1002/cpe.4139

WILEY

SPECIAL ISSUE PAPER

GRAPLEr: A distributed collaborative environment for lake ecosystem modeling that integrates overlay networks, high-throughput computing, and WEB services

Kensworth C. Subratie[1] | **Saumitra Aditya**[1] | **Srinivas Mahesula**[1] | **Renato Figueiredo**[1] | **Cayelan C. Carey**[2] | **Paul C. Hanson**[3]

[1]University of Florida, Gainesville, 32611, FL, United States
[2]Virginia Tech, Blacksburg, 24061, VA, USA
[3]University of Wisconsin-Madison, Madison, 53706, WI, USA

Correspondence
Kensworth C. Subratie, Larson Hall, University of Florida, Gainesvile, FL 32611, USA.
Email: kcratie@acis.ufl.edu

Funding information
National Science Foundation, Grant/Award Number: 1527415, 1339737, and 1234983

Summary

The GLEON Research And PRAGMA Lake Expedition—GRAPLE—is a collaborative effort between computer science and lake ecology researchers. It aims to improve our understanding and predictive capacity of the threats to the water quality of our freshwater resources, including climate change. This paper presents GRAPLEr, a distributed computing system used to address the modeling needs of GRAPLE researchers. GRAPLEr integrates and applies overlay virtual network, high-throughput computing, and WEB service technologies in a novel way. First, its user-level IP-over-P2P overlay network allows compute and storage resources distributed across independently administered institutions (including private and public clouds) to be aggregated into a common virtual network, despite the presence of firewalls and network address translators. Second, resources aggregated by the IP-over-P2P virtual network run unmodified high-throughput-computing middleware to enable large numbers of model simulations to be executed concurrently across the distributed computing resources. Third, a WEB service interface allows end users to submit job requests to the system using client libraries that integrate with the R statistical computing environment. The paper presents the GRAPLEr architecture, describes its implementation and reports on its performance for batches of general lake model simulations across 3 cloud infrastructures (University of Florida, CloudLab, and Microsoft Azure).

KEYWORDS

climate change, distributed computing, general lake model, HTCondor, IPOP-VPN, lake modeling, overlay networks

1 | INTRODUCTION

The GLEON Research And PRAGMA Lake Expedition—GRAPLE—aims to improve our understanding and predictive capacity of water quality threats to our freshwater resources, including climate change. It is predicted that climate change will increase water temperatures in many freshwater ecosystems, potentially increasing toxic phytoplankton blooms.[1,2] Consequently, understanding how altered climate will affect phytoplankton dynamics is paramount for ensuring the long-term sustainability of our freshwater resources. Underlying these consequences is complex physical-biological interactions, such as phytoplankton community structure and biomass responses to short-term weather patterns, multiyear climate cycles, and long-term climate trends.[3,4] New data from high-frequency sensor networks (eg, GLEON) provide easily measured indicators of phytoplankton communities, such as in situ pigment fluorescence and show promise for improving predictions of ecosystem-scale wax and wane of phytoplankton blooms.[5] However, translating sensor data to an improved understanding of coupled climate-water quality dynamics requires additional data sources, model development, and synthesis, and it is this type of complex challenge that requires increasing computational capacity for lake modeling.

Searching through the complex response surface associated with multiple environmental starting conditions and phytoplankton traits (model parameters) requires executing and interpreting thousands of simulations, and thus substantial compute resources. Furthermore,

the configuration, setup, management, and execution of such large batches of simulations is time-consuming, both in terms of computing and human resources.

This puts the computational requirements well beyond the capabilities of any single desktop computer system, and to meet the demands imposed by these simulations, it becomes necessary to tap into distributed computing resources. However, distributed computing resources and technologies are typically outside the realm of most freshwater science projects. Designing, assembling, and programming these systems are not trivial and require the level of skill typically available to experienced system and software engineers. Consequently, this imposes a barrier to scientists outside information technology and computer science disciplines and presents challenges to the acceptance of distributed computing as a solution to most lake ecosystem modelers.

GRAPLE is a collaboration between lake ecologists and computer scientists that aims to address this challenge. Through this interdisciplinary collaboration, we have designed and implemented a distributed system platform that supports compute-intensive model simulations, aggregates resources across an overlay network spanning collaborating institutions, and presents intuitive WEB service-based interfaces that integrate with existing programming environments that lake ecologists are familiar with, such as R.[6]

This paper describes GRAPLEr, a cyberinfrastructure that is unique in how it integrates a collection of distributed hardware resources through the IP-over-P2P (IPOP)[7,8] overlay virtual network, supports existing models and the high-throughput-computing middleware (HTCondor) distributed computing middleware,[9] and exposes a user-friendly interface that integrates with R-based desktop environments through a WEB service. As a multitiered distributed solution, GRAPLEr incorporates several components into an application-specific solution. Some of these components are preexisting solutions, which are deployed and configured for our specific uses, while others are specifically developed to address unique needs.

The rest of this paper is organized as follows: Section 2 describes driving science use cases and motivates the need for the GRAPLEr cyberinfrastructure. Section 3 describes the architecture, design, and implementation of GRAPLEr. Section 4 describes a deployment of GRAPLEr and summarizes results from an experiment that evaluates its capabilities and performance. Section 5 discusses related work, and Section 6 concludes the paper.

2 | SCIENCE USE CASES

Simulation modeling is a powerful tool for examining the effects of many different climate change scenarios on water quality in lakes. Here, we are specifically focusing on scenarios that look at both linear and stochastic changes in climate drivers to predict changes in harmful algal blooms. Algal community dynamics can be highly nonlinear because of the large diversity of algae and their functional traits in lakes, as well as the dynamic physical-chemical environment in which the algae live.[10,11] Consequently, small incremental changes in different climate drivers could potentially reveal threshold effects that result in disproportionately large changes in responses. Thus, it could be possible that a small change in climate drivers creates an ideal set of environmental conditions for blooms to occur in silico.

How do algae and lake ecosystem processes respond to small changes in the timing, frequency, and magnitude of air temperature, precipitation, and wind? Can our models simulate the algal growth that is observed in lakes with high-frequency sensors in the field? To answer these questions, we use the simulation software, General Lake Model—Aquatic Eco Dynamics (GLM-AED).[12,13] GLM-AED simulates the vertical dimension (ie, 1-dimensional [1D]) of lake hydrodynamics in response to meteorological and hydrologic forcing and lake chemistry in response to external loading and physical and biological fluxes. The 2 biological trophic levels modeled here, phytoplankton (4 functional groups) and zooplankton (3 functional groups), are modeled according to a set of functional traits that govern growth and death in response to the physical and chemical environment. All 3 components—physical, chemical, and biological—interact to form a highly dynamic and vertically heterogeneous environment. To study phytoplankton blooms, we adjust the model parameters representing functional traits in the phytoplankton and zooplankton. There are thousands of viable trait combinations, and therefore, thousands of simulations needed to determine the outcomes from these hypothetical communities. To study climate change effects on phytoplankton, we adjust meteorological forcing driver data to represent future climate scenarios.

We explored 2 use cases of the GRAPLEr cyberinfrastructure to address the questions above. In the first use case, we ran hundreds of thousands of GLM-AED simulations in which we incrementally altered air temperature driver data by ±0.1°C (in a range between −3°C to +3°C from observed data), wind speed by ±0.1 m/s (in a range between 0 to +5 m/s), and precipitation by ±0.1 mm in precipitation (in a range between 0 to 5 mm) at each hourly time step throughout a year. Analyzing all possible combinations of these climate drivers necessitated running many thousand simulations to determine where threshold effects in algal growth exist in the model. This information was critical for identifying which meteorological conditions would best promote algal blooms. In the second use case, we defined distributions of potential parameter values for different phytoplankton functional traits governing nutrient uptake, light and temperature sensitivity, and growth rates and then randomly pulled different parameter values from the distributions. Analyzing all possible combinations of the parameter values allowed us to determine which parameter values best recreated observed field data of algal abundance in the lake. Having this information consequently allowed us to improve the parameterization of the model to predict future algal blooms and water quality.

3 | ARCHITECTURE AND DESIGN

3.1 | System architecture (GRAPLEr)

The system architecture of GRAPLEr is illustrated in Figure 1. Users interact with the system via a client-side library that is invoked from an R development environment (eg, R Studio) running on their personal computer. User requests are created using the R language and mapped to the GRAPLEr application programming interface (API) calls, which in turn transforms and transmits them to the GRAPLEr WEB Service

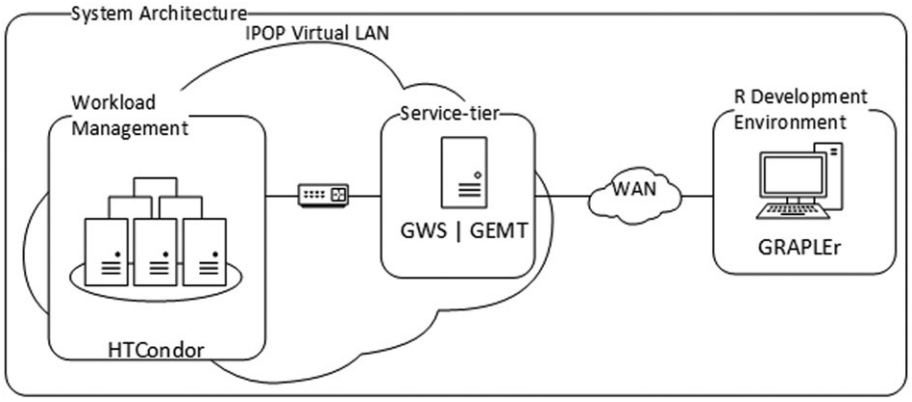

FIGURE 1 System Architecture (GRAPLEr). Users interact with GRAPLEr using R environments in their desktop (right). The client connects to a WEB service tier that exposes an endpoint to the public Internet. Job batches are prepared using GEMT and are scheduled to execute in distributed HTCondor resources across an IPOP virtual private network. HTCondor, high-throughput-computing middleware; GEMT, GRAPLEr experiment management tools; GWS, GRAPLEr WEB Service; IPOP, IP-over-P2P

(GWS). The GWS tier is responsible for interpreting the user requests, invoking the GRAPLEr experiment management tools (GEMT) to set up and prepare the simulations, and queuing jobs for submission to the HTCondor pool. The HTCondor workload management tier is responsible for scheduling and dispatching model simulations across the compute resources, which are interconnected through the IPOP virtual network overlay. All the GRAPLEr components are described in the following sections.

3.2 | Overlay virtual network (IPOP)

Rather than investing significant effort in development, porting, and testing new applications and distributed computing middleware, GRAPLEr has focused on an approach in which computing environments are virtualized and can be deployed on-demand on cloud resources. While virtual machines (VMs) available in cloud infrastructures provide a basis to address the need for a user-provided software environment, another challenge remains: how to interconnect VMs deployed across multiple institutions (including private and commercial cloud providers) such that HTCondor and the simulation models work seamlessly? The approach to address this problem is to apply virtualization at the network layer.

The IPOP[7] network overlay is a flexible and dynamic virtual private network (VPN). It frees the administrator to define the virtual network by simply specifying relationships among the participating nodes. IP-over-P2P then transparently builds logical communication links to facilitate seamless and secure communication within this ad hoc group. Additionally, the IPOP overlay is self-healing as it automatically adjusts to changes in the underlying network—the hosts continue to function without user intervention and with minimal disruption.

This dynamic function of IPOP is essential for addressing the complexities associated with the intranetworking within the hybrid cloud system composed of local and public cloud resources. By using tunneling protocols to extend discrete network segments between hosts, a virtual LAN is created. A virtual LAN simplifies the design and layout of the network by grouping hosts with common requirements regardless of their actual location. For example, HTCondor hosts can be configured to communicate over public internet infrastructure. However, the deployment of such a cluster has considerable more complexities than a corresponding cluster deployed within a LAN.

IP-over-P2P allows GRAPLEr to define and deploy its own VPN that can span physical and VMs distributed across multiple collaborating institutions and commercial clouds. To accomplish this, IPOP captures and injects network traffic via a virtual network interface or "TAP" device. The TAP device is configured within an isolated virtual private address subnet space. IP-over-P2P then encrypts and tunnels virtual network packets through the public Internet. The "TinCan"[8] tunnels used by IPOP to carry network traffic use facilities from WEB real-time computing to create end-to-end links that carry virtual IP traffic instead of audio or video.

To discover and notify peers that are connected to the GRAPLEr "group VPN", IPOP uses the eXtensible messaging and presence protocol (XMPP). XMPP messages carry information used to create private tunnels (the fingerprint of an endpoint's public key), as well as network endpoint information (IP address: port pairs that the device is reachable). For nodes behind network address translators (NATs), public-facing address:port endpoints can be discovered using the session traversal utilities for NAT protocol, and devices behind symmetric NATs can use traversal using relays around NAT to communicate through a relay in the public Internet. Put together, these techniques handle firewalls and NATs transparently to users and applications and allow for simple configuration of VPN groups via an XMPP server.

By using IPOP's network virtualization technologies, unmodified distributed computing resources can be integrated to implement a workload services cluster.

3.3 | Workload management (HTCondor)

A key motivation for the use of virtualization technologies, including IPOP, is the ability to integrate existing, unmodified distributed computing middleware. In particular, GRAPLEr integrates HTCondor,[9] a specialized workload management system for compute-intensive jobs. Like other full-featured batch systems, HTCondor provides a job queueing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. Users submit their serial or parallel jobs to HTCondor, HTCondor places them into a queue, chooses when and

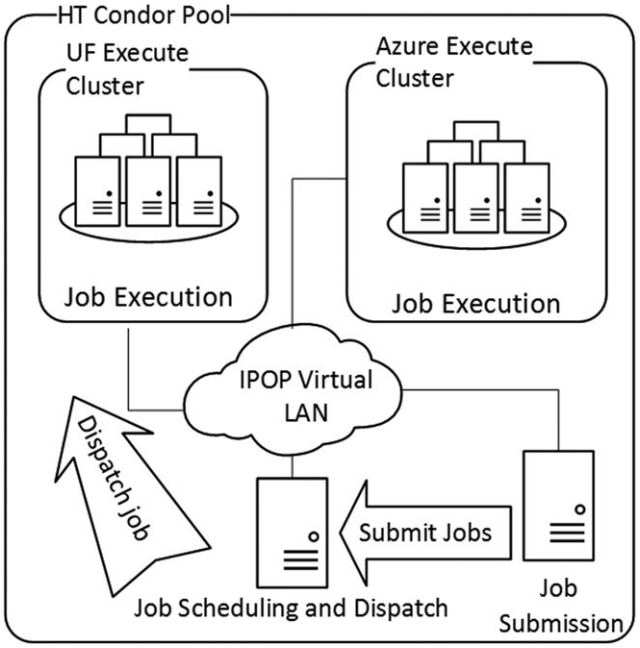

FIGURE 2 Workload Management (HTCondor). GRAPLEr supports unmodified HTCondor software and configuration to work across multiple sites (eg, a private cloud at UF and a commercial cloud at Azure). HTCondor, high-throughput-computing middleware; IPOP, IP-over-P2P

where to run the jobs based upon a policy, carefully monitors their progress, and ultimately informs the user upon completion. Figure 2 illustrates how HTCondor has been deployed for implementing the GRAPLEr workload execution and management tier.

An HTCondor resource pool, running across distributed resources and connected by an IPOP network, provides a general-purpose capability where it is possible to run a variety of applications from different domains. Furthermore, application-tailored middleware can be layered upon this general-purpose environment to enhance the performance and streamline the configuration, of user simulations.

3.4 | Experiment management tools (GEMT)

GRAPLEr experiment management tool provides a suite of scripts for designing and automating the tasks associated with running general lake model (GLM)–based experiments on a very large scale. Here, we use the term "experiment" to refer to a collection of simulations that address a science use case question, such as determining the effects of climate change on water quality metrics. GRAPLEr experiment management tool is both the guidelines for the design and the layout of individual simulations in the experiment, as well as a library of executable code for creating and managing an experiment over its lifetime in the system. The primary responsibility of GEMT is to identify and target the task-level parallelism inherent in the experiment by generating proper packaging of executables, inputs, and outputs; furthermore, GEMT seeks to effectively exploit the distributed compute resources across the HTCondor pool by performing operations such as aggregation of multiple simulations into a single HTCondor job, compression of input and output files, and the extraction of selected features from output files.

For the simulations in an experiment, GEMT defines the naming convention used by the files and directories as well as their layout. The user may interact with GEMT in 2 possible ways: (1) directly, by using a desktop computer configured with the IPOP overlay software and HTCondor job submission software or (2) indirectly, by issuing requests against the GRAPLEr WEB service. In the former case, once the user has followed the GEMT specification for creating their experiment, executing it and collecting the results becomes a simple matter of invoking 2 GEMT scripts. However, the user is left the responsibility of deploying and configuring both IPOP and HTCondor locally. Additionally, the user is now a trusted endpoint on the VPN that carries its own security implications. A breach of the user's system is a potential vulnerable point to accessing the VPN. The latter case alleviates the user from both these concerns. This paper focuses on the latter approach, where GEMT scripts are invoked indirectly by the user through the WEB service.

There are 3 distinct functional modes for GEMT, which pertain to the different phases of the experiment's lifetime. Starting with its invocation, on the submit node, GEMT selects a configurable number of simulations to be grouped as a single HTCondor job. Multiple simulations are grouped into a single HTCondor job as the costs of job scheduling and network transfer of short-running simulations can be significant. By grouping simulations into a single HTCondor job, redundant copies of the input can be eliminated to reduce the bandwidth transfer cost and only a single scheduling decision is needed to dispatch all the simulations in the job. The inputs and executables pertaining to a group of simulations are then compressed and submitted as a job to the HTCondor scheduler for execution. When this job becomes scheduled, GEMT is invoked in its second phase, this time on the HTCondor execute node. The execute-side GEMT script coordinates running each simulation within the job, and preparing the output so it can be returned to the originator. Finally, in its third phase, back on the submit node side, GEMT collates the results of all the jobs that were successful and presents them in a standard format to the end user.

GRAPLEr experiment management tool implements user configurable optimizations to fine tune its operations for individual preferences. It can limit how many simulations are placed in a job, and it will compress these files for transfer. GRAPLEr experiment management tool can also overlap the client side job creation with the server side execution to further minimize the user's wait time for results. These features can be set via a configuration file, and together they combine to provide a simplified mechanism to execute large numbers of simulations.

3.5 | GRAPLEr web service

The GWS module, as illustrated in Figure 3, is a publicly addressable WEB service available on the Internet and serves as a gateway for users to submit requests to run experiments. GRAPLEr WEB service is a middleware service-tier that provides a WEB API, implemented over the HTTP protocol, to its remote users. This API encompasses the server side functionality of GRAPLEr; each method of this API represents a discrete unit of computation capability, which is to be executed on the distributed cluster. GRAPLEr WEB service accepts and interprets user requests, configure and queues jobs, and consolidate and prepares

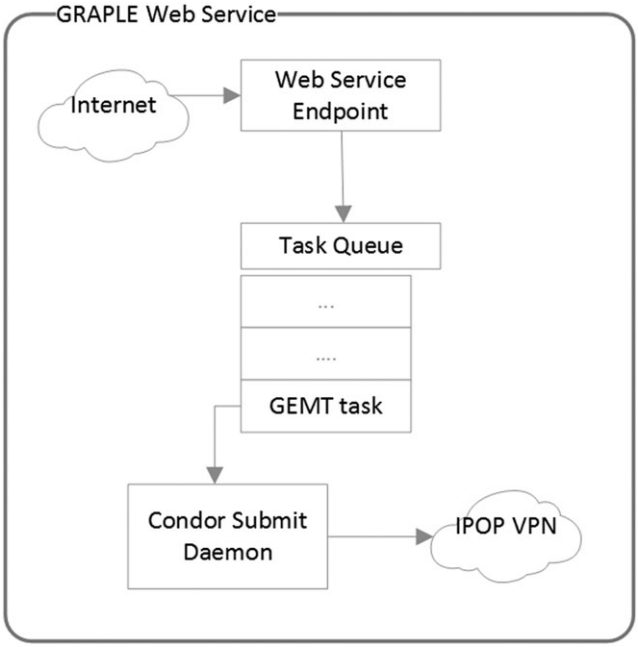

FIGURE 3 GRAPLEr WEB Service (GWS). The GWS is responsible for taking WEB service requests from users, interpreting them, and creating tasks for remote execution using GRAPLEr experiment management tool. IPOP, IP-over-P2P; VPN, virtual private network

results for retrieval. For example, create and execute an experiment consisting of N simulations by varying air temperature according to a statistical distribution for a climate change scenario. GRAPLEr WEB service extensively uses the functionality of GEMT for simulation processing and is collocated in the same host as the GEMT library. This host acts as the submit node to the HTCondor pool, where it monitors job submission and execution.

Representational state transfer, or REST, is an architectural style for networked hypermedia applications that is primarily used to build lightweight and scalable WEB services. Such a WEB service, referred to as RESTful, is stateless with a uniform interface and representation of entities and communicates with clients via messages and address resources using URIs. GRAPLEr WEB service implements the RESTful paradigm and is designed to treat every job submission independently from any other. Note that there is per-experiment state that is managed by GWS, such as the status of each HTCondor job submitted by the GWS. The state of the experiment is maintained on disk, within the local filesytem, leaving the service itself stateless. GRAPLEr WEB service implements the public-facing interface using a combination of open-source middleware for WEB service processing—Python Flask[14]—and an asynchronous task queue—Python Celery.[15] The application is hosted using uWSGi (an application deployment solution) and supplemented by a Nginx reverse proxy server to offload the task of serving static files from the application. The used technology stack facilitates rapid service development and robust deployment.

The GWS workflow begins when a request is received from an R client by the service interface, which is handled by Flask. The request to evaluate a series of simulations can be provided in one of several ways, as discussed in detail in Section 3.6. However, only data files are accepted as input—no user provided executable binaries, or scripts are executed as part of the experiment. A single client-side request can potentially unfold into large numbers (eg, thousands) of jobs, and GWS places these requests into a Celery task queue for asynchronous processing. Provisioning a task queue allows GWS to decouple the time-consuming processing of the input and task submission to HTCondor, from the response.

A 40-character unique identifier (UID) is randomly generated for each simulation request received by GWS; it is used as an identifier to reference the state of an experiment and is thus used for any further interactions with the service for a given experiment. Using the UID returned by GRAPLEr, an R client can not only configure the job but also monitor its status, download outputs, and terminate the job. Once the input file has been uploaded to the service, GWS puts the task into the task queue and responds promptly with the UID. Therefore, the latency that the R developer experiences, from the moment the job is submitted to when the UID is received, is minimized. A GWS worker thread then dequeues GEMT tasks from the task queue and processes the request according to the parameters defined by the user. Figure 3 shows the internal architecture and setup of GWS.

3.6 | GRAPLE R language package (GRAPLEr)

The user-facing component of GRAPLEr is an R package that serves as a thin layer of software between the WEB service and the R programming language. It provides an R language application programming interface that can be programmatically consumed by client programs wanting to utilize the GRAPLEr functionality. It acts as a proxy to translate user commands written in R into WEB service requests that are sent to GWS. It also transfers data between the client and WEB service as necessary.

The following examples illustrates the sequence of R calls to program a GRAPLEr experiment. In the first example, a new GRAPLEr instance is created, and the GWS IP address, the length of time the results should remain available for download, and the local directories for storing various experiment files are specified. The availability of the service is checked to make sure it is running, and finally, the list of post processing filters is retrieved.

```
batchExp <- new("Graple", Retention, ExpRootDir, ResultsDir, TempDir)
batchExp <- GrapleCheckService(batchExp)
batchExp <- GrapleListPostProcessFilters(batchExp)
```

The second example executes a batch type experiment of 1 or more simulations, which have previously been created and placed in the directory *ExpRootDir*. A human readable experiment name is set, and the command to run them on the cluster is issued. The experiment completion status is checked and then the results are downloaded when it becomes available.

```
batchExp <- setExpName(batchExp, "BatchExperiment1")
batchExp <- GrapleRunExperiment(batchExp)
batchExp <- GrapleCheckExperimentCompletion(batchExp)
batchExp <- GrapleGetExperimentResults(batchExp)
```

The third example shows how a user can specify a parameter-sweep experiment with simulations, which are derived from a baseline set. A new experiment instance is again created, and the friendly name and post processing filter are assigned. Both the baseline simulation and the experiment description have been created and stored in the *ExpRootDir* directory at the client. However, the filter specified by *Filtername* is stored remotely in the service. The command to run the sweep experiment is invoked, and the progress is checked periodically until it indicates 100% completion. The results are then retrieved to be used locally.

The GRAPLEr source code can be found at https://github.com/graple and is made available under the MIT Software License. The project website, along with tutorials and usage guides, is available at http://graple.org/.

4 | USE CASE WORKFLOW

A key feature of GRAPLEr is to automatically create and configure an experiment by generating a range of simulation scenarios, see Figures 4 and 5. This is accomplished by varying simulation inputs, based on the user's request and application-specific knowledge. In particular, the service uses application-specific information to identify data in the input file (such as air temperature or precipitation), and apply transformations to them to generate multiple discrete simulation scenarios. This removes the onus from the user to generate, schedule,

```
sweepExp <- new("Graple", ExpRootDir, ResultsDir, TempDir)
sweepExp <- setExpName(sweepExp, "SweepExperiment")
sweepExp <- GrapleRunSweepExperiment(sweepExp, "Filtername")
sweepExp <- GrapleCheckExperimentCompletion(sweepExp)

while (sweepExp@StatusMsg != '100.0% complete') {
  Sys.sleep(10);
  sweepExp <- GrapleCheckExperimentCompletion(sweepExp)
  cat(paste(sweepExp@ExpName, sweepExp@StatusMsg, sep=":"))
}
sweepExp <- GrapleGetExperimentResults(sweepExp)
```

To prevent the use of the WEB service interface to execute arbitrary code, we find that custom code—whether binary executables or R scripts—cannot be sent as part of the simulation requests; instead, users only provide input files and parameters for the GLM simulations. The scenarios that can be run are currently restricted to using GLM tools and our own scripts.

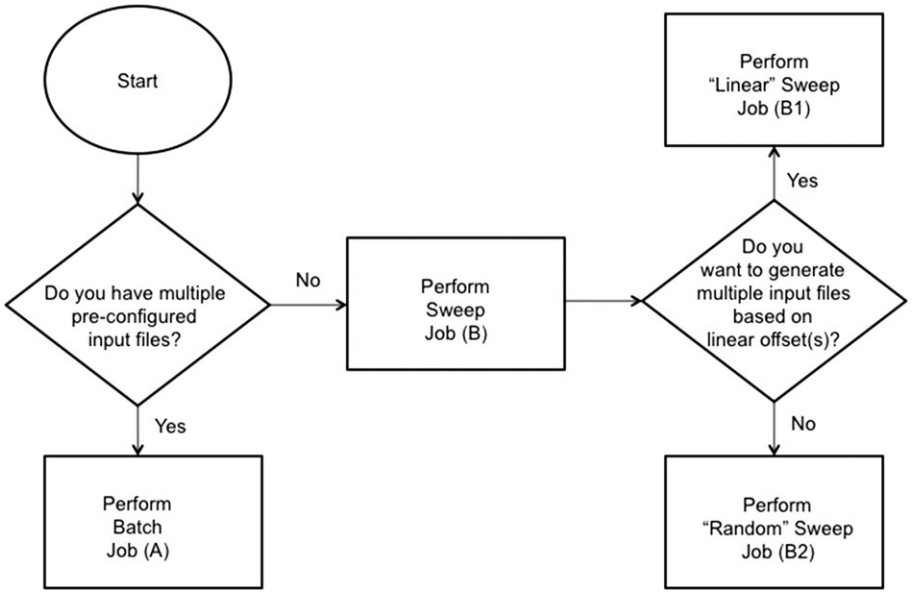

FIGURE 4 GRAPLEr top level workflow chart

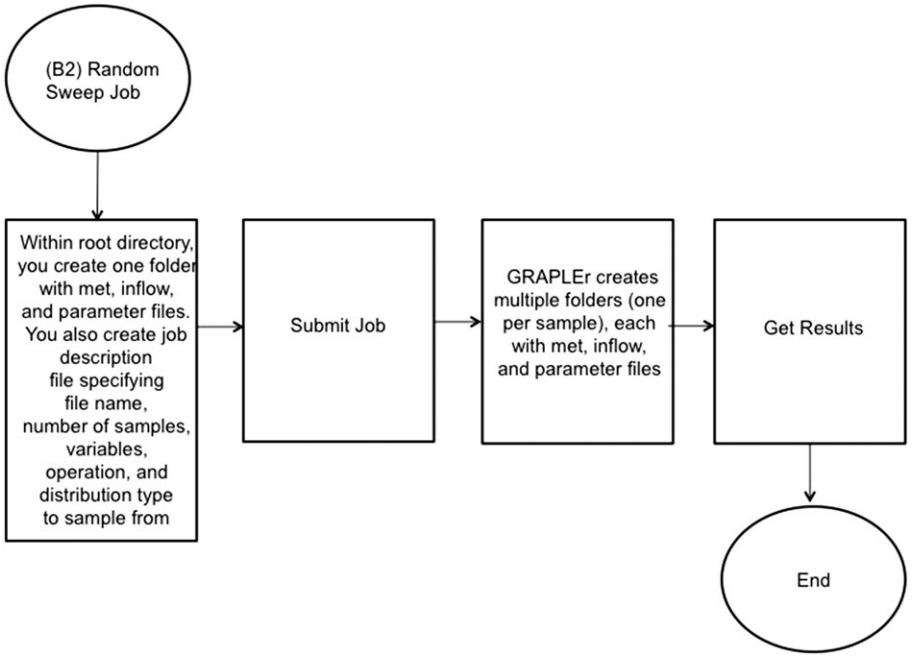

FIGURE 5 GRAPLEr sweep job workflow chart

TABLE 1 GRAPLEr WEB service API

Endpoint	Input	Response
ServiceStatus	N/A	WEB service availability
GrapleRun	Simulations to execute	Job UID
GrapleRunStatus	UID of a previously submitted job	Job status message (completed/processing)
GrapleRunResults	UID of a previously submitted job	URI of job results
GrapleRunLinearSweep	The base simulation and job definition file	Job UID
GrapleRunMetSample	The base simulation and job definition file	Job UID
GrapleEnd	UID of previously submitted job	Status message (success/retry)
GrapleListFilters	N/A	The list of available post processing filters

and collate the outputs of thousands of simulations within their desktops and allows them to quickly create expressive experiment scenarios from a high-level description that simply enumerates which input variables to consider, what function to apply to vary them, and how many simulations to create. The user also has the flexibility to specify a postprocessing operation for each simulation, and to retrieve and download only a selected subset of the results back to their desktops, thereby minimizing local storage requirements and data transfer times.

On the basis of the science use case introduced and discussed in section 2, and our understanding of the GRAPLEr infrastructure and APIs described in section 3, we can now concretely illustrate how GRAPLEr would be used to solve this problem. In the second use case, we defined distributions of potential parameter values for different phytoplankton functional traits governing nutrient uptake, light and temperature sensitivity, and growth rates, and then randomly pulled different parameter values from the distributions.

Within the R programming environment, the GRAPLEr function *GrapleRunSweepExperiment* is used to set the stage for creating an experiment derived from a baseline simulation, which was created from actual senor data. The coding pattern is exactly as previously described in subsection 3.6 "GRAPLE R Language Package." In addition to the baseline simulation, an experiment description file and optional postprocessing filter name is specified. The experiment description file specifies which distribution (linear, random, uniform, binomial, or Poisson) to choose samples from, the number of samples, the variable(s) to be modified, and the operation applied to a variable for each randomly generated value (add, subtract, multiply, or divide). The postprocessing filter name specifies a selection from a collection of operations, which is stored within the GEMT library, to run after the successful completion of each individual simulation.

The invocation of *GrapleRunSweepExperiment* results in a request being sent to the WEB service API method *GrapleRunMetSample* in Table 1, which goes about generating the previously described experiment consisting of 'N' simulations. From this single input and description, GWS utilizes GEMT to generates a detailed description of the experiment along with a partitioning of how the of jobs should be distributed to the "M" worker nodes in the cluster. Each worker node then creates N/M simulations, which comprises its respective subset of the experiment and executes them sequentially in turn.

5 | EVALUATION

In this section, we present a quantitative evaluation of a proof-of-concept deployment of GRAPLEr. The goal of this evaluation is to demonstrate the functionality and capabilities of the framework by deploying a large number of simulations to an HTCondor pool. The HTCondor pool is distributed across multiple clouds and connected by the IPOP virtual network overlay. Rather than focusing solely on the reduction in execution times, we evaluate a setup that is representative of an actual deployment composed of execute nodes with varying capabilities.

A GLM simulation is specified by a set of input files, which describe model parameters and time-series data that drive inputs to the simulation, such as air temperature over time, derived from sensor data. The resulting output at the completion of a model run is a netCDF file containing time series of the simulated lake, with many lake variables, such as water temperatures at different lake depths. In our experiments, we use the 1-D GLM Aquatic Eco-Dynamics (AED) model with a single GLM-AED simulation of a moderately deep lake, run for 8 months at an hourly time step. The test experiment was designed to run reasonably quickly, yet of sufficiently long duration where the timing results would not be skewed by extraneous timing factors. The input folder size was approximately 3 MB, whereas the size of the resulting netCDF file after successful completion of the simulation was 90 MB. We note that simulations run over decades and with more frequent time steps may increase simulation run time and result output by an order of magnitude.

We conducted simulation runs on different systems to obtain a range of simulation runtimes. With the baseline parameters, GLM-AED simulation times ranged from the best case of 6 seconds (on a CloudLab system with Intel Xeon CPU E5-2450 with 2.10 GHz clock rate and 20 MB cache) to 57 seconds (on a University of Florida system with virtualized Intel Xeon CPU X565 with 2.60 GHz clock rate and 12MB cache). Note that individual 1-D GLM-AED simulations can be short-running; the GEMT feature of grouping multiple individual simulations into a single HTCondor job leads to increased efficiency by reducing the overhead time incurred from job scheduling and placement.

Description of experiment setup: The GRAPLEr system deployed for this evaluation was distributed across 3 sites: University of Florida, NSF CloudLab, and Microsoft Azure. The GWS/GEMT service front-end, HTCondor submit node, and HTC-Central Manager were hosted on VMs running in Microsoft's Azure cloud. We deployed 3 HTCondor worker nodes, each with 16 cores and 16 GB of RAM. Two nodes were hosted in VMs on a VMware ESX server at the University of Florida and one on a physical machine in the CloudLab Apt cluster at University of Utah. All the nodes in this experiment ran Ubuntu-14.04, HTCondor version 8.2.8, and IPOP GroupVPN version 15.01.

To conduct the evaluation, we conducted executions of 3 different experiments containing 3000, 5000, and 10 000 simulations of an example lake with varying meteorological input data. Figure 6 summarizes the results from this evaluation. As a reference, we also present the estimated best-case sequential execution time on a single, local machine, taken the CloudLab, and UF machines as a reference. For 10 000 simulations, we achieved a speedup of 2.5 (with respect to sequential execution time of the fast workstation) and 23x speedup (with respect to the sequential execution time at a UF-VM).

FIGURE 6 Job runtimes for GRAPLEr high-throughput-computing middleware pool, compared to sequential execution times on CloudLab (SEQ Fast) and UF (SEQ Slow) slots

It is observed that the time taken to complete the experiment depended greatly on how the jobs were executed by the HTCondor scheduler; as the GRAPLEr cluster is comprised of heterogeneous nodes with varying capabilities. It therefore follows, the achieved speedup is smaller when compared to the best-case baseline on the fastest node as opposed to that on the slower nodes. Furthermore, because HTCondor is best suited for long running jobs, the user-perceived speedup of GRAPLEr over local stand-alone systems will increase as longer-running experiments are executed through the service. We expect that as demand for modeling tools by the lake ecology community increases, so will the complexity, time series resolution, and simulated epochs of climate change scenarios, further motivating users to move from a local processing workflow to distributed execution through GRAPLEr.

Submission of a job to the HTCondor pool involves processing of input (for sweep requests) and packaging of generated simulations into GEMT. To evaluate this step, we conducted experiments to account for the time taken by GRAPLEr to respond to a request to generate a given number of simulations and submit them for execution. The results are presented in Table 2. The column service response time captures the time taken by GRAPLEr to respond to a request with a UID, which is slightly more than the time required to upload the baseline input. The column input processing time captures the time taken to generate and compress all "N" inputs for job submission.

Though not fully explored yet in the design of GRAPLEr, another benefit of remote execution through a WEB service interface is the leveraging of storage and data sharing capabilities of the collaborative infrastructure aggregated by distributed resources connected through the IPOP virtual network. For instance, this experiment resulted in unfiltered result output of 900 GB. By keeping the results on the GRAPLEr cloud and allowing users to share simulation outputs and download selected subsets of the raw data, the service can provide a powerful capability to its end users in enabling large-scale, exploratory scenarios, by both reducing computational time and relaxing local storage requirements at the client side.

TABLE 2 Input handling times

Job size (# of simulations)	Service response time, sec	Input processing time, sec
3000	0.72	1790
5000	0.88	3039
10 000	0.59	6854

6 | RELATED WORK

Several HTCondor-based high-throughput computing systems have been deployed in support of scientific applications. One representative example is the open science grid,[16] which features a distributed set of HTCondor clusters. In contrast to open science grid, which expects each site to run and manage its own HTCondor pool, GRAPLEr allows sites to join a collaborative, distributed cluster by joining its virtual HTCondor pool via the IPOP virtual network overlay. This reduces the barrier to entry for participants to contribute nodes to the network—eg, by simply deploying one or more VMs on a private or public cloud. Furthermore, GRAPLEr exposes a domain-tailored WEB service interface that lowers the barrier to entry for end users.

WS-PGRADE[17] is a workflow design and execution tool that has a broad scope of functionalities geared at p¨arameter sweepäpplications. Users can design workflows, specify how input is combined and on what type of resources job execution takes place. Workflows can also be hosted in a repository and shared with other end users.

A designer selects a workflow node that matches his algorithm characteristics and attaches to it data ports, which determine how the input is combined for execution as well as generated for output. These workflow nodes can be combined and nested to create more complex ones and the desired application. Depending on the workflow node, execution can be mapped onto the local system, a desktop grid, or service grid. According to the WS-PGRADE workflow and parameter set classification, GRAPLEr would be a combination of single regular node, single parameter style node, and a generator output port. The generator port produces multiple output files derived from its input and a user specified algorithm. A GRAPLEr job, which consists of multiple simulations, is mapped to a single parameter style workflow and each executing simulation a single regular workflow node.

However, the highly generalized approach used by WS-PGRADE, which provides extensive flexibility, introduces 2 of the very problems GRAPLEr was designed to solve the need to redesign and reimplement existing applications, and learn new technologies outside the user's knowledge domain.

GRAPLEr provides an intuitive and easy to learn workflow and user interface. Through a collaboration between a core group of domain scientists and cyberinfrastructure experts, GRAPLEr codifies typical usage patterns into workflows that are exposed through simple interfaces to the broader set of target end users, including students. Furthermore, rather than a WEB-based presentation layer, domain scientists can quickly become productive as GRAPLEr's client API integrates directly into their existing R/Rstudio development environment. The GRAPLEr workflows are built around the existing processes, and instead leverage the existing opportunities for concurrency.

The NEWT[18] project also provides a RESTful-based WEB service interface to High-Performance Computing (HPC) systems. IT is a gateway to access HPC computing and data resources at the National Energy Research Scientific Computing Center and is designed to make these resources highly usable via a WEB browser. The NEWT WEB service provides an API over HTTP, which are used by client side browser technologies to WEB applications. Newt, however, does not describe any mechanism for incorporating a heterogeneous set of distributed computing resources within its HPC cluster. GRAPLEr differentiates itself with this ability to leverage virtualized cloud resources that are interconnected by virtual networks.

7 | CONCLUSION

GRAPLEr, a distributed computing system that integrates and applies overlay virtual network, high-throughput computing, and WEB service technologies, is a novel way to address the modeling needs of interdisciplinary GRAPLE researchers. The system's contribution is its combination of power, flexibility, and simplicity for users who are not software engineering experts but who need to leverage extensive computational resources for scientific research. We have illustrated the system's ability to identify and exploit parallelism inherent in GRAPLE experiments. Additionally, the system scales out, by simply adding additional worker nodes to the pool, to manage both increasingly complex experiments as well as larger number of concurrent users. GRAPLEr is best suited for large numbers of long-running simulations as the distribution and scheduling overhead will increase the running time for such experiments. As lake models demand increased resolution and longer time scales to address climate change scenarios, GRAPLEr provides a platform for the next generation of modeling tools and simulations to better assess and predict the impact to our planet's water systems.

The GRAPLE endeavor additionally includes an active involvement in training its end users, as this is perceived as a critical aspect of the penetration and impact of its software infrastructure on its community. This training and inclusion extends beyond the collaboration between the working group's engineers and domain scientists, to undergraduate and graduate students, and postdoctoral researchers. This sets the stage for an iterative process of learning and refinement of the cyberinfrastructure—where each subsequent iteration facilitates tackling problems of a much larger scale, and scientists trust and reply on the tool and methodologies to address broader water research issues.

ACKNOWLEDGEMENTS

This material is based upon work supported in part by the National Science Foundation under Grants No. 1527415, 1339737, and 1234983. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. We thank Amy Hetherington for making the GRAPLEr workflow figures.

REFERENCES

1. Paerl HW, Huisman J. Blooms like it hot. *Science*. 2008;320(5872):5–7.
2. Brookes JD, Carey CC. Resilience to blooms. *Science*. 2011;334 (6052):46–47.
3. Flynn KJ. Castles built on sand: dysfunctionality in plankton models and the inadequacy of dialogue between biologists and modellers. *J Plankton Res*. 2005;27(12):1205–1210.
4. Carey CC, Hanson PC, Lathrop RC, St. Amand AL. Using wavelet analyses to examine variability in phytoplankton seasonal succession and annual periodicity. *Journal of Plankton Research*. 38:27–40. https://doi.org/10.1093/plankt/fbv116.
5. Weathers K, Hanson P, Arzberger P, et al. The Global Lake Ecological Observatory Network (GLEON): the evolution of grassroots network science. *Bull Limnol Oceanogr*. 2013;22(3):71–73.
6. R Core Team. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing; 2015. http://www.R-project.org/. Accessed March 2017.
7. Ganguly A, Agrawal A, Boykin PO, Figueiredo R. IP over P2P: Enabling self-configuring virtual IP networks for grid computing. *International Parallel and Distributed Processing Symposium*, Rhode Island, Greece; 2006.
8. Juste PSt, Jeong K, Eom H, Baker C, Figueiredo RJO. Tincan: User-defined p2p virtual network overlays for ad-hoc collaboration. *EAI Endorsed Trans Collaborative Computing*. 2014;2:1–15,e4.
9. Thain D, Tannenbaum T, Livny M. Distributed computing in practice: the condor experience. *Concurrency Computat: Pract Exper*. 2005;17(2-4):323–356.
10. Hansen GJA, Carey CC. Fish and phytoplankton exhibit contrasting temporal species abundance patterns in a dynamic north temperate lake. *PloS One*. 2015;10(2):e0115414.
11. Rigosi A, Carey CC, Ibelings BW, Brookes JD. The interaction between climate warming and eutrophication to promote cyanobacteria is dependent on trophic state and varies among taxa. *Limnol Oceanogr*. 2014;59(1):99–114.
12. Hipsey MR, Bruce LC, Hamilton DP. Glm general lake model. model overview and user information, Perth, Australia, The University of Western Australia Technical Manual; 2013.
13. Hipsey MR, Hamilton DP, Hanson PC, et al. Predicting the resilience and recovery of aquatic systems: a framework for model evolution within environmental observatories. *Water Resour Res*. 2015;51(9):7023–7043.
14. Grinberg M. *Flask Web Development: Developing Web Applications with Python*. 1st ed. Sebastopol, CA: O'Reilly Media, Inc.: Sebastopol, CA; 2014. ISBN 1449372627, 9781449372620.
15. Solem A. Celery: Distributed Task Queue. http://www.celeryproject.org/; 2013. Accessed March 2017.
16. Pordes R, Petravick D, Kramer B, et al. The open science grid. *Journal of Physics: Conference Series*, vol. 78, IOP Publishing; 2007:012057.
17. Kacsuk P, Karoczkai K, Hermann G, Sipos G, Kovacs J. WS-PGRADE: Supporting parameter sweep applications in workflows. *2008 Third Workshop on Workflows in Support of Large-Scale Science*, Austin, TX; November 2008:1–10.
18. Cholia S, Skinner D, Boverhof J. Newt: A restful service for building high performance computing web applications. *Gateway Computing Environments Workshop (GCE), 2010*, IEEE: New Orleans, LA; 2010:1–11.

How to cite this article: Subratie KC, Aditya S, Mahesula S, Figueiredo R, Carey CC, Hanson PC. GRAPLEr: A distributed collaborative environment for lake ecosystem modeling that integrates overlay networks, high-throughput computing, and WEB services. *Concurrency Computat: Pract Exper*. 2017;29:e4139. https://doi.org/10.1002/cpe.4139