

Hydrology@Home: a distributed volunteer computing framework for hydrological research and applications

Ramil Agliamzanov, Muhammed Sit and Ibrahim Demir 


ABSTRACT

Web-based distributed volunteer computing enables scientists to constitute platforms that can be used for computational tasks by using potentially millions of computers connected to the internet. It is a widely used approach for many scientific projects, including the analysis of radio signals for signs of extraterrestrial intelligence and determining the mechanisms of protein folding. User adoption and clients' dependence on the desktop software present challenges in volunteer computing projects. This study presents a web-based volunteer computing framework for hydrological applications that requires only a web browser to participate in distributed computing projects. The framework provides distribution and scaling capabilities for projects with user bases of thousands of volunteers. As a case study, we tested and evaluated the proposed framework with a large-scale hydrological flood forecasting model.

Key words | Citizen Science, distributed computing, flood forecasting, hydrological modeling, scientific computation, volunteer computing

Ramil Agliamzanov
Higher Institute of Information Technology and
Information Systems (ITIS),
Kazan Federal University,
18 Kremlyovskaya Street, Kazan 420008,
Russian Federation

Muhammed Sit (corresponding author)
Informatics Graduate Program,
The University of Iowa,
100 C. Maxwell Stanley Hydraulics Laboratory,
Iowa City, IA 52242-1585,
USA
E-mail: muhammedali-sit@uiowa.edu

Ibrahim Demir 
Department of Civil and Environmental
Engineering,
The University of Iowa,
100 C. Maxwell Stanley Hydraulics Laboratory,
Iowa City, IA 52242-1585,
USA

INTRODUCTION

With the recent advances in high-resolution sensing and monitoring capabilities, large-scale spatial and temporal modeling became computationally challenging and mostly beyond the capabilities of a single ordinary computer (Nakada *et al.* 1999). Surpassing this limitation is possible only with high-performance computing (HPC) clusters or a distributed system of many computers working in parallel. Distributed systems and computing rely on physically separate participant computers that make up a large system (Lampert & Lynch 1989). Considering, nowadays the computing power is not clustered only in computing centers (Anderson 2004), and the distributed systems form an evidentiary opportunity for scientific computing purposes. Blockchain-based systems, ARPANET (the ancestor of the internet) and the internet itself, are common examples of distributed systems (Stone & Bokhari 1978; Coulouris *et al.* 2005). In parallel computing systems, physically separated computers, or computing instances, carry out computations

for a task as directed by a central system (Culler *et al.* 1999). Similarly, in distributed computing, instances use a distinct messaging system to work asynchronously with all instances, with scheduling support from a central system (Almasi & Gottlieb 1989).

Forming the base for distributed computing, distributed systems are widely used in both academia and industry (Attiya & Welch 2004). As an efficient way of computation (Korpela 2012), distributed computing depends on the computational power of connected instances to solve the assigned computational tasks (Coulouris *et al.* 2005). Consequently, a distributed system is able to achieve tasks that are beyond the capabilities of a single computing instance. Such a distributed computing system is considered a volunteer system when it is comprised of volunteers with capable devices that contribute to the acceleration of the cumulative process of the completion of computational tasks needed to solve the problem (Anderson & Fedak 2006). In such

volunteer computing systems, volunteers with idle computational power within their desktops and portable devices (smartphones and tablets) take part in a scientific process of solving problems.

In a regular centralized system, all computation is carried out by a dedicated HPC instance owned by the computing project's operator. The HPC instance is, most of the time, expensive to both setup and maintain. Similarly, in grid computing or distributed computing without volunteer participation, the cardinal factor on the cost becomes the financing of the infrastructure. In contrast, in volunteer computing, a project operator only needs to maintain a web server to form a central authority on the computation medium, while the computation is done by connected third-party computers. This nature of distributed voluntary computing makes it a suitable option for CPU-intensive tasks (Anderson & Fedak 2006).

Citizen Science, as a term, describes scenarios in which people who are not scientists participate or conduct research. Citizen volunteers and researchers can cooperate to address and answer scientific questions (Cohn 2008). For instance, volunteer test subjects can contribute to research by providing data (Bonney *et al.* 2009; Wiggins & Crowston 2011; Fiene & Lowry 2012; Ferster & Coops 2013; Jonoski *et al.* 2013; Horita *et al.* 2015). Volunteers can even serve as field assistants in scientific efforts (Cohn 2008). The goal of this study is to enable people to participate in scientific studies through a web browser by letting researchers use the idle computational power of their computers. A crowdsourcing-oriented platform provides a channel for this 'volunteer computing' (Granell *et al.* 2016). Thus, a volunteer in volunteer computing is a user with a computationally capable machine that is connected to the distributed computation system to let the system use its resources in various computing tasks. Occasionally, the volunteer is also referred to as the 'client', borrowing the term from web terminology.

Related work

Volunteer computing has been used in many large-scale scientific projects for more than two decades. One of the earliest examples of a distributed volunteer computing project is the Great Internet Mersenne Prime Search (Durrani

& Shamsi 2014), which aimed to find Mersenne Prime numbers using hardware provided by volunteers. In the early 2000s, the few web-based volunteer computing efforts met with limited success because of the slow performance of web systems and the lack of capabilities for multicore and background process support. One volunteer computing project known as Bayanihan (Sarmanta & Hirano 1999) interprets Java usage behaviors on web browsers to create a distributed computation system. In the same year, a project called Charlotte (Baratloo *et al.* 1999) used web browsers to create a parallel distributed computing framework.

SETI@Home is an early example of distributed computing that uses software to access computer's resources to work on assigned tasks only when the computer is idle (Sullivan *et al.* 1997). The software was designed as a screensaver (Shirts & Pande 2000) that used volunteer computers to search for extraterrestrial existence in the universe by analyzing radio signals that reached Earth as a part of universal SETI effort. In 2000, the Pande Lab at Stanford University introduced the Folding@Home project, which aims to carry out protein folding simulations to enhance disease research processes. Similar to SETI@Home, Folding@Home also began as a screensaver (Shirts & Pande 2000).

In 2004, researchers released BOINC (Berkeley Open Infrastructure for Network Computing), an open source generic framework for volunteer computing (Anderson 2004). BOINC consists of two separate entities, one of which uses the server for centrally managed task distribution, while the client software uses volunteer computing resources. Client software enables volunteers to choose a project from several options on the BOINC framework, download assigned tasks from the server, and send the computed results back to the server after computation. After BOINC was released, distributed computing projects and studies that involved volunteer participants gained momentum. Many volunteer computing projects such as SETI@Home and Folding@Home started to take advantage of the BOINC infrastructure, including MilkyWay@Home (Cole *et al.* 2010), Rosetta@Home (Das *et al.* 2007), and Einstein@Home (Einstein@Home 2005). Several studies also expanded BOINC to different implementation levels, some of which use BOINC middleware on mobile devices (Black & Edgar 2009; Theodoropoulos *et al.* 2016) to

enhance their ability to handle and distribute data (Costa et al. 2008).

The first conceptual studies worked on web browsers (Cappello et al. 1997; Sarmenta & Hirano 1999) they were using Java, which requires a plugin. After many years, these were followed by studies using the native JavaScript language (Boldrin et al. 2007; Cushing et al. 2013). The focus of these studies was to use JavaScript to get distributed data for computation and to have computed results on the client-side (deThread 2016). Recent developments in new JavaScript engines (e.g., V8) and revised web standards improved JavaScript's efficiency and enhanced performance that was close to low-level languages. Early limitations on web-based distributed computing studies included performance and multicore support. In 2016, the Atlas framework was proposed as a solution to the performance problems of these web systems (Gullapalli 2016). Atlas' main contribution was the use of Emscripten (Emscripten 2015) as middleware before the computation. Emscripten is a technology created by Mozilla to compile LLVM bitcode into JavaScript, with the goal of accelerating the JavaScript code, bringing it closer to the native computer code. Emscripten can be used with many programming languages that can be compiled into LLVM bitcode, such as C and C++. Besides the conventional computing, the distribution of computation tasks is done on Blockchain-based systems as well (Golem 2015; GridCoin 2013). While Golem focuses on renting out idle computing power, GridCoin presents a way of utilizing BOINC with Blockchain. Even though the early examples of blockchain centered computing do not involve browsers, this can be made possible by arranged implementation choices.

Many distributed computing projects require volunteers to download the software (Larson et al. 2009). Volunteers must install and maintain client software, which raises privacy and security issues. The use of web browsers that are pre-installed on most computers rather than custom standalone applications that have limited cross-platform compatibility looks like a promising approach. The heterogeneous dissemination of different hardware across a large spectrum is one of the biggest problems in software adaptation (Durrani & Shamsi 2014). JavaScript is the most widely used scripting language on the web and it works on a virtual machine within any web browser. A distributed

volunteer computing system based on JavaScript runs on a web browser and can handle the same tasks that a desktop framework does without any client software. In this paper, we present a generic scientific volunteer computing framework, *Hydrology@Home*, that runs on web browsers and an evaluation study to demonstrate the capabilities of the framework with a real-time large-scale hydrological model.

This paper starts with a review of the literature on distributed computing, volunteer computing, and citizen science. The Methods section explains the framework for the server-side as well as the client-side, with corresponding components and technologies. The Results and Discussion section includes a case study on hydrological modeling for the evaluation of the framework with benchmarks and results that demonstrate the performance and capabilities of the framework. The paper concludes with a summary of our objectives and findings, as well as future perspectives about the distributed computing framework.

METHODS

In this section, we will present the web-based distributed volunteer computing framework and its technical aspects, explaining the stack and technologies that power the framework and how these technologies use both server-side and client-side resources.

Framework

The goal of the framework is to be able to distribute any kind of scientific computing task among participant devices and logging the computation results. The current version of the framework expects scientists to generate an analysis or modeling code written in JavaScript to run on volunteer computers. New web standards (e.g., WebAssembly) enable the framework to use the native C/C++ code directly in the framework. Computation tasks in the framework rely on inputs (i.e., data and model parameters) provided to the model code. Input data for the intended computation task can be integrated directly into the framework by the database connection interfaces to be distributed to the volunteer computers with computing subtasks.

The framework's infrastructure consists of two major parts (Figure 1): one that deals with task scheduling and task distribution on the server-side and another that handles computations and communication with the server on the client-side. 'User interface' describes the panel volunteers (or users) use to start participating in scientific computing tasks. The framework asks users to give their consent to join a selected scientific effort and contribute computer time for the project. The volunteer can always revoke this permission later. Also, users can pause a computation when they need the computational power for a personal task, resuming their participation at any time. The framework can be embedded on any website with a single line of code. This allows researchers to easily enable this framework on websites with a heavy user base (e.g., organization or university websites).

The framework allows users to select multiple projects or a single project. The framework will make an on-demand stress test on the volunteer computer with multiple threads to understand the multi-CPU configuration of the volunteer system and to optimize the model runs. This allows the framework to utilize JavaScript Web Workers (also known as workers) to run multiple tasks in parallel on a single volunteer computer. In the framework, each computation run over a set of parameters is called a task.

When a set of tasks bundled together, they form a job. Each volunteer system gets a job at one time. The framework can estimate the optimal job size, or in other words, the optimal number of tasks for each job by a stress test. It is also possible for the framework to optimize the computation process depending on the participant device's capabilities in terms of data storage, CPU, and GPU time. The framework requires the user to keep the browser session active during runs and continue running if the session breaks and resumes. When starting to run the computation, besides the recommended values by the stress tests, users can also provide job size and worker count to the framework. Since each JavaScript worker uses only one CPU core at one time, contributors who prefer to not to have all computational power they possess to be utilized by the framework can limit this utilization by setting the worker count.

Since each job contains a number of tasks and they are acquired and sent results by the client when the computation is done, they do not depend on each other. A volunteer instance that is set to contribute by taking 10 jobs with 1,000 tasks runs the computation functions with given parameters 10,000 times ($10 \times 1,000$). In other words, if 10 different jobs were conveyed to a client one by one each with 1,000 separate tasks, this client computer

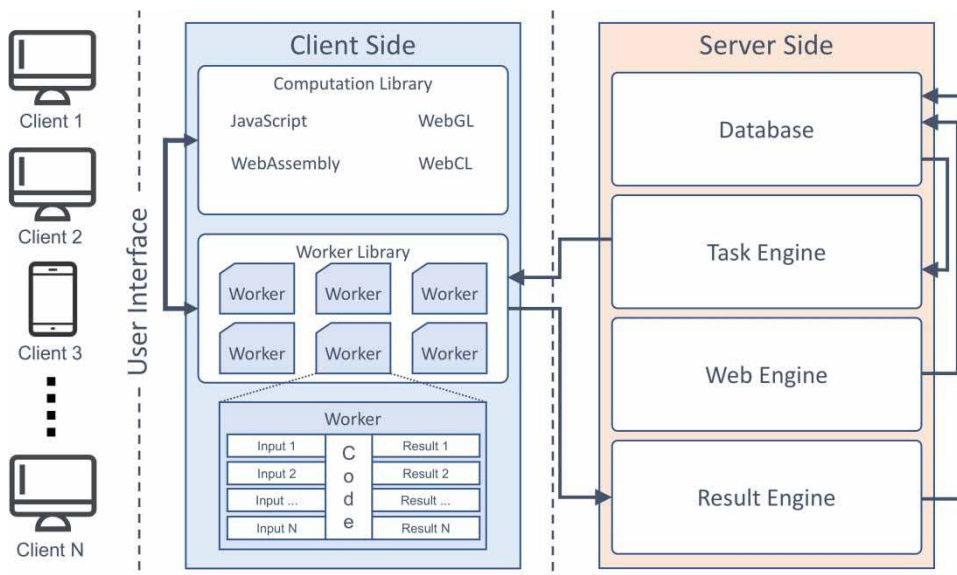


Figure 1 | Architecture of the volunteer distributed computing framework.

computes 10,000 separate function runs and generates results.

Project creation

Researchers can create projects using the server-side interfaces of the framework. Project creation consists of two separate steps. The first step is the submission of the computation code for modeling or analysis of the data. The code is needed to be written in JavaScript and it should be stored in the framework's database. The second step is to define communication methods to provide input data to the framework. The framework implements communication application programming interfaces (APIs) for input and output data. The data are expected to be supplied from a PostgreSQL database table using these APIs. Each parameter that the computation function needs must have its own column on the database table with its respective name that it has on the JavaScript function's parameter list.

One important aspect that affects data loading into the framework is task scheduling. The scheduling system in the framework needs a priority scheme regarding the data tables. The scheme provision is optional and depends on the computation task itself. If the priority scheme is provided, it must be included in the input data table with an integer column of 'priority' starting from the value '0'. This value represents the level of data entry on a dependency table. The lower the value of the 'priority' column, the more priority a data entry is given. Thus, the data entries with a priority value of '0' will be the first to be sent to the volunteers, and entries with the value of '1' will be second and so forth. Besides the additional column of 'priority', each data entry also should have a Boolean column in order to keep track of the completed tasks namely 'is_completed' and an array column of 'connections'.

The connection between upstream and downstream nodes in the priority list is established by the 'connections' column. This column incorporates an array of indices which are the IDs of future computation tasks. By using this connections array, each task can be linked to any number of future tasks and this information enables the framework to use outputs of previous tasks as inputs in future tasks. Output saving to the output table and inserting the outputs to the input table for future computations handled

by the results mechanism. The results mechanism needs to be defined using the database connection interfaces that the framework proposes. The result engine that will be described in the following subsection uses this interface to make the project database store the computation outputs. If outputs of any task will be used as an input for a further computation task, the column name that the individual outputs will be saved within the input table must be specified within the interface.

After the communication medium between the server-side of the framework and the data source established via database interfaces, volunteers are able to load the website and to start contributing to the project.

Server-side task handling

The stack powering the framework consists of the Linux operating system, PostgreSQL database management system, PHP server-side scripting engine, and Apache web server. PHP is one of the most common web programming languages for server-side development. PostgreSQL is an open source database management system that handles tasks, input datasets, and output from the computation. The framework runs on Red Hat Enterprise Linux with Apache 2. Apache is an open source project that enables developers to serve their web applications. The framework depends entirely on open source and free systems to facilitate reproduction and adaptation by other research groups. Input data and modeling codes should be stored in a PostgreSQL database. Once the data are stored in the designated database and afore-mentioned connections are done, they are available to be shared with connected volunteer computing instances.

Task handling on the server essentially relies on three components of the framework: task engine, result engine, and web engine (Figure 1). The web engine handles interactions when a user visits the user interface and triggers the computation. It broadly handles the new volunteer device and makes arrangements to enable the task engine to transfer the data and the computation code to the user. After the web engine initiates the process to enable the new volunteer client to contribute to the project, the task engine gathers tasks from the project database based on their dependencies and their availability. The task engine

also restricts the assignment of tasks that have any unmet dependencies to volunteers. After the selection is done, task engine bundles assigned tasks as a job package, then it transfers the data and the code to the client. The data consist of parameters and the actual code. Each parameter entry in its corresponding column from the connected database is conveyed to the volunteer devices with other parameters. The term distribution comes into play with the task engine. The task engine manages all tasks and data distributions to volunteer computing instances. After the client has processed the data, it sends the results of the analysis, as a set of task results, to the result engine. The result engine's job is to save the computed results to the database and mark the assigned tasks as completed using the column 'is_completed'.

All server-side tasks handling within the framework is done on a first-come-first-served basis. A client computer is taken to a queue when it makes a request to get tasks. After the serving for all clients who requested tasks before them is completed, the computation data they need to contribute to any project are conveyed to them. If there are fewer tasks on the database than the requested amount, then all remaining tasks are conveyed to the volunteer computer and the next requests get the message of 'No tasks available' from the server-side until new data are made available.

Client-side task handling

Client-side task and job management consists of two components: the worker library and the computation library. The worker library is the installation of the framework that operates JavaScript Web Workers for computations. The computation library is the core foundation of the infrastructure that integrates different web technologies to provide a versatile approach to the computation. The workflow of the computation library implicitly depends on the implementation. For example, the analysis code can be written with the latest technologies, such as WebGL which enables web applications to take advantage of the graphical processing power of the GPUs. The computation library is flexible and supports many high- and low-level computation libraries and technological choices. After the task engine on the server-side sends the input data and the computation

code to the client, the worker library with the JavaScript Web Worker implementation on the framework handles client-side processing. Because the user has already determined the number of workers and job sizes, the worker library creates the desired number of workers to share the tasks. The tasks are sent directly to the client to be distributed to workers.

Normally, a webpage becomes irresponsive to user interaction or interference when a JavaScript code is running on the page. HTML5 Web Workers enable JavaScript codes to run in the background while keeping the webpage responsive to the user. With the help of Web Workers, computational tasks on a web browser run within the JavaScript Virtual Machine without affecting the interface. Also, because each worker is created as a separate thread, tasks assigned and distributed to a computing instance can easily be computed by the independent workers, i.e., threads. This approach enables the framework to use multiple cores simultaneously for different tasks. It should be noted that all created web workers work on the same job at any given time.

The data sent from the server to the client by the task engine must be stored on the client-side for an organized and orderly computation process. The framework uses IndexedDB, a transactional database that handles large-scale data on the web browser. After the task engine pushes the data that will be used in distributed tasks, the Web Worker script stores them locally on an IndexedDB database. After the data are stored, the tasks within a job are randomly distributed to the workers, the computation begins, and all data are drawn from this new local IndexedDB to workers. Computed results are also saved to the same storage to be pushed back to the server, i.e., result engine. This structure allows local organization and scheduling of the tasks and handles interruption and resumption of the computing session on the client-side. After all tasks in the IndexedDB are completed, the client-side of the framework can either ask for a job or terminate the connection with the server-side.

Interrupted and corrupted task handling

The above-mentioned distribution, computation, and communication pipeline assume that all parts of the framework

do their defined jobs properly. However, since the client-side cannot be completely managed by the framework itself, there are some overhauling mechanisms within the framework for cases that the client-side fails to convey the expected output. When volunteer computer disconnects from the server-side of the framework by browser window closure, the server-side still waits some time for the response from the client. If the response that includes computation results is not issued within a time interval that is defined by the project admin, the tasks assigned are not marked as complete on the database. Incomplete tasks are later sent to new volunteer instances by being taken into account in the regular scheduling system.

Another problem that may occur on the client-side is that malevolent attempts can be made through regular API routes of the framework using client computers. For instance, computations can be simulated to be seen as the right results by the server-side while not providing actual computation results. This potential problem can be surpassed by validating the results by distributing the same tasks to multiple random volunteer instances but with each extra validation task, the overall computation time increases. Consequently, the advantages of using a distributed approach may be lost if validation is overdone. Nevertheless, the framework presented in this paper enables researchers to define a validation threshold for distributed tasks via the scheduling system. It can be zero or a non-zero value, and this decision is left to the project administrator.

Case study on hydrological modeling

The Iowa Flood Information System (IFIS) was developed at the Iowa Flood Center (Demir & Krajewski 2013; Krajewski *et al.* 2017) using a generalized water cyberinfrastructure. The IFIS is based on an integrated and extensible architecture that allows researchers, instructors, and students to create custom cyberinfrastructures for their own research projects and curriculum (Jones *et al.* 2018; Sermet & Demir 2018; Weber *et al.* 2018), enabling reproducible scientific workflows (Duffy *et al.* 2012; Gil *et al.* 2016). The IFIS features many data management, scientific visualization, and communication tools, as well as data resources to facilitate flood risk management and research at the watershed scale (Demir & Beck 2009; Demir *et al.* 2009, 2015, 2018).

The Iowa Flood Center operates a real-time, HPC-based flood forecasting model (Cunha *et al.* 2012; Small *et al.* 2013). This model transforms the rainfall from the climate projections and provides quantitative stage and discharge forecasts and a 10-day flood risk outlook in the IFIS for more than 1,500 locations in Iowa (e.g., communities and stream gauges). It is a distributed model running on HPC (Quintero *et al.* 2016; Krajewski *et al.* 2017) that builds on the concept of the landscape decomposition into hillslopes and channels (Mantilla & Gupta 2005). The mass transport equations for each hillslope in the network are defined as a power law relation describing flow velocity as a function of discharge and drainage area (Ayalew & Krajewski 2017). Details about the model equations, configuration, and numerical solver are provided in Quintero *et al.* (2016) and Krajewski *et al.* (2017). The model runs for more than 600,000 hillslopes in Iowa (Figure 2). These hillslopes consist of 10 dependency levels and scheduling was done by this order of dependency (Table 1). Each level in dependency needs previous levels to be computed before being sent to the volunteer instances. Each independent task focuses on a single hillslope and needs minimal data transfer of 5–10 kilobytes. Running such a comprehensive model in an operational setting requires extensive resources for data processing, computation, and communication of results. Operational systems like the IFIS can benefit from the proposed framework to distribute computations for its operational flood forecasting model to its large userbase. We used a simplified version of this hydrological model for the case study to understand the scalability and performance of the volunteer computing framework. The model is simplified to reduce the data needs but not affect the computational evaluation of distributed computing. Simplification is only done on the system parameters to minimize the effort on data preparation for the case study. Average values are used at the hillslope level for these parameters instead of unique values. The simplifications are implemented in a way to not affect the computation and data transfer complexity for objective evaluation of the performance of the framework. This framework will allow volunteers to contribute to hydrological research and applications from their homes, as in the case study on flood forecasting.

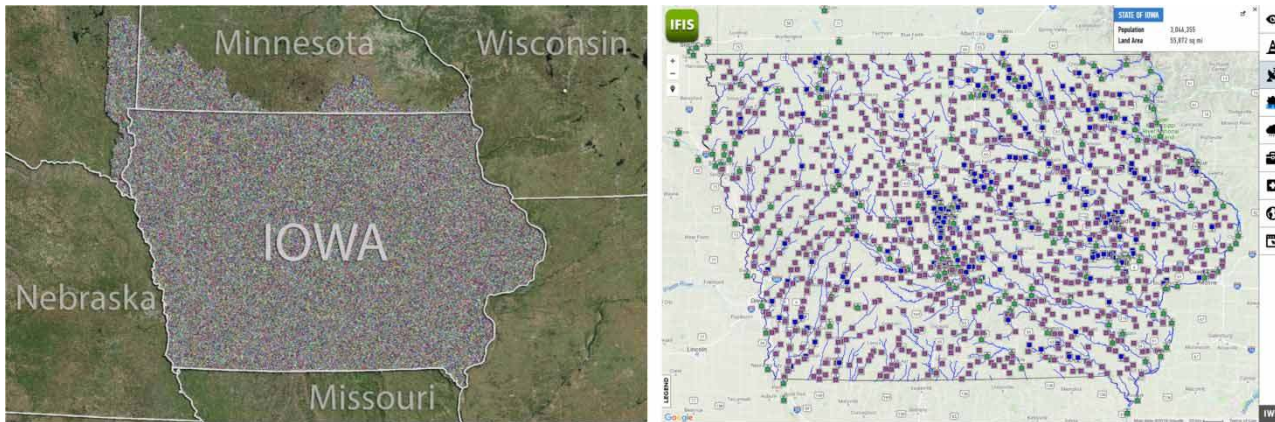


Figure 2 | Computational tasks for 620,172 individual hillslopes (left) and 1,500 points of interests reporting model results (right).

Table 1 | Dependency layers and their respective hillslope numbers

Level number	Number of hillslopes
0	318,205
1	140,975
2	76,002
3	41,496
4	23,837
5	10,867
6	5,582
7	2,423
8	763
9	22

RESULTS AND DISCUSSIONS

As a part of the evaluation and testing of the distributed computing framework, we carried out several tests to better understand the performance and robustness of the framework. We evaluated the performance of the framework and the model separately. Because the initial model implementation will be in JavaScript, we wrote the model's code in both JavaScript and C language to compare the performance of the JavaScript engine (e.g., V8) on web browsers to the native code. To carry out this test, we prepared a script to run both codes in JS and C languages 1,000 times each and to return the time elapsed to execute the model code with data. According to these results, C-based code performed twice as fast as JS code. This

shows promise for web-based computing based on JavaScript and implementation using new technologies (e.g., WebAssembly) that will enable native C code to run on web systems close to the speed of desktop-level computations.

Since browsers do not allow a webpage to run a script in the background when the tab is closed on the client computer, the computation process could be interrupted suddenly without notice. This would interrupt the volunteer computing project. The framework can handle such interruptions and continue where it left off when the user returns to the page. One solution uses browser extensions to fully eliminate interruption and allow clients to continue running even after the browser is closed. The extension can easily be installed in the client browser when the user visits the project website.

Another challenge we encountered when running the model and testing the framework was about data and task dependencies of the specific model used. Some models might have a prioritized hierarchy for the computation of certain tasks. Thus, the next steps of the computations need the results of prior computations. When a lack of data is experienced in the model, volunteers will need to wait until new data become available for computation. This is more common for hydrological applications in which computations from upstream nodes are required by downstream nodes. Priority scheduling of the tasks by prioritizing tasks from upstream nodes to the volunteer computing instances can reduce the dependency and improve the time-efficiency of the computations.

Because the idea of distributed computing using volunteer instances depends on data transfers, a reliable data transfer process is crucial. Even though the internet provides a proper medium for such interactions, the data size creates a limitation in a system that heavily depends on these transfer operations. Thus, in this study, we selected the use case; these limitations and data transfer processes were optimized for this use case's needs. Considering the trade-off and effects on total run time, web-based volunteer computing systems are more suitable for computationally intensive tasks and studies with limited or low data needs.

Benchmarks with the model and framework

We conducted various tests to evaluate the server and client communication in terms of performance and timing and to observe the behavior of the framework and the model. We used a moderate client system for tests with Intel(R) Core(TM) i5-4258U CPU @ 2.40 GHz. Selected tests and their goals can be listed as follows: (a) tests with a single client and different numbers of workers and job size combinations to understand the performance changes; (b) tests to measure average and total elapsed time by each step of the computation; (c) performance on the utilization of separate browser windows versus the utilization of workers on one browser window; (d) tests with only job size changes on the same client to observe the linearity of the performance gains when job size is increased; (e) tests with the heterogeneity of clients to evaluate the performance of both the client and the server; and (f) server performance when multiple clients are connected to the server to evaluate the task retrieving durations as the number of clients increased.

In the case study, we defined each task as solving rainfall-runoff model equations for a hillslope (watershed) for discharge predictions. Test results for different numbers of jobs and job size combinations (a) and elapsed times for each step of the process (b) can be seen in Figure 3. These tests are conducted using a client computer with a two-core CPU (Intel(R) Core(TM) i5-4258U CPU @ 2.40 GHz). Our benchmarks showed that an increase in worker count did not dramatically decrease the computation time elapsed. Nevertheless, there is an indisputable improvement when the worker number is increased. This parallel increase

continues until the CPU cannot handle the number of workers selected.

In another test (c), when the client computer uses two different browser windows instead of two separate workers within a single window, the efficiency increases steadily in terms of time. This discrepancy can be linked with threading of the selected web browser (e.g., chromium-based) used in the tests because of its JavaScript Engine (e.g., V8). Another client test (d) focuses on retrieving tasks and sending results – when the total number of tasks being computed is kept at 10,000 and the job size is increased (in result the number of jobs is decreased), the total elapsed time while tasks were being fetched by the client decreases. This is basically because the data for each hillslope are small and the network overhead requires more time than processing the data itself. The trade-off between job size and elapsed time starts to become trivial when job size is set to 2,500. It is not practical to choose small job sizes with the goal of enhancing efficiency because retrieving tasks and sending results for smaller job sizes increase network overhead. We recommend a job size that is neither too small nor too big for a distributed computing project that works on the framework.

We carried out heterogeneity tests (e) to see if framework-related issues could affect the performance of clients with a variety of hardware and software combinations. Because the computation code provided runs on JavaScript Virtual Machine, the main performance limitation is due to having a non-native code. Depending on the JavaScript engine, browser selection can also affect the performance of the framework. We conducted tests with several modern browsers and achieved mixed performance results. There is no strong candidate for the best web browser and/or engine selection. The operating system choice has a limited or minor effect on the performance. A major and inherent limitation is the hardware components including CPU- and GPU-based on the computation engine structure. In particular, the volunteer computers' CPU speed and cores affect the performance of the computation and the number of workers that the computer can handle concurrently. The geographic location of the volunteer is a natural obstacle for data transfer in retrieving tasks and sending results, but not an essential problem in most cases. Network

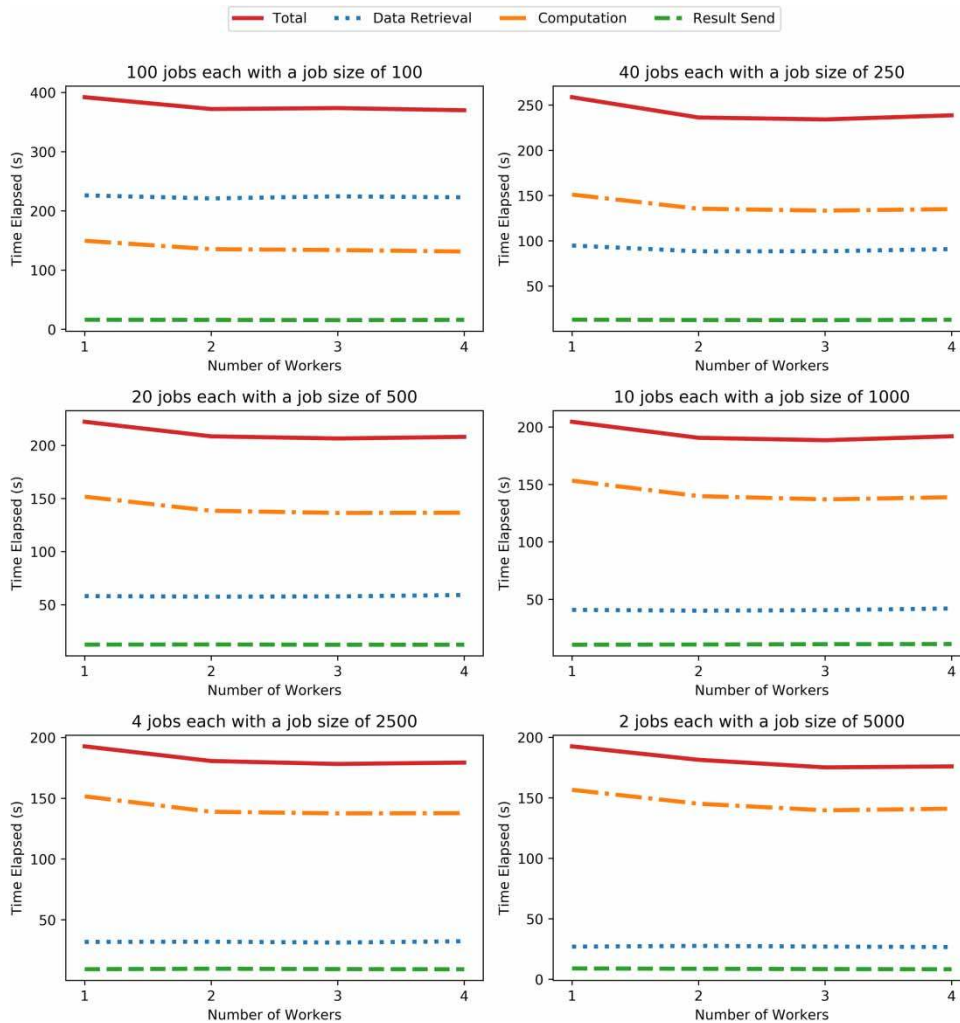


Figure 3 | Elapsed time for each step of the task management process and several worker counts.

connection is another bottleneck for an overall voluntary computation process.

The server's ability to handle users concurrently is the only factor that depends on the server specifications and the framework code itself. Tests conducted (f) with multiple clients simultaneously requesting tasks does not produce a significant overhead on the server and its efficacy in the task distribution process. Besides the server's scaling abilities, we also tested the scalability of the framework by utilizing an open source load test tool (i.e., [Locust 2011](#)). Tests were done by concurrent and swarmed users as well. A Locust file was created containing possible scenarios within the framework with different job sizes (e.g., 100, 200, 500, and 1,000), number of jobs (e.g., 10, 50, and

100), and worker counts (e.g., 1, 2, and 4). Different numbers of users (e.g., 100, 500, 1,000, and 2,000) were tested on the framework's server-side abilities, and consequently, we clearly saw that framework was successful at handling various swarming rates. The success rate of the made requests to the framework starts to decrease at the swarming rate of 200 users per second.

Finally, we made a test run in operational settings for the entire system with seven computers that have various specifications utilizing various numbers of workers ([Table 2](#)). For simplicity, the job sizes for every computer were kept fixed at 1,000 tasks at a time, and when there were less than 1,000 tasks remaining at the server-side, all remaining tasks were conveyed to the requesting party without trying

Table 2 | Computers used in the experiment, their utilized CPUs, worker counts, and time-wise performance for each of them for a job with the job size of 1,000

Computer	CPU	Worker count	Elapsed time for 1,000 tasks (ms)
Computer 1	Intel(R) Core(TM) i5 – 4258U CPU @ 2.40 GHz	2	8,909.3
Computer 2	Intel(R) Core(TM) i7 – 2600 CPU @ 3.40 GHz	2	6,435.3
Computer 3	Intel(R) Core(TM) i7 – 2600 CPU @ 3.40 GHz	4	6,186.5
Computer 4	Intel(R) Core(TM) i5 – 6200U CPU @ 2.30 GHz	2	6,928.8
Computer 5	Intel(R) Core(TM) i7 860 CPU @ 2.80 GHz	2	9,586.9
Computer 6	Intel(R) Core(TM) i7 – 7700HQ CPU @ 2.80 GHz	4	6,363.7
Computer 7	Intel(R) Core(TM) i7 – 6800 K CPU @ 3.40 GHz	4	9,216.8

to fulfill the requested job size. There were no specified number of jobs and all machines were run until no other tasks have remained on the database. It should also be noted that the validation mechanism was not used for this experiment and all computers were assumed as trusted volunteers.

The time required for the overall process (task retrieval, computations on the client, and sending results to the server) varies based on the computer. The model used in the framework runs with data for a total of 620,172 hillslopes in the State of Iowa with 10 levels of dependencies (Table 1). Task distribution and the computation were started with the first level of dependency (Level 0), and any following level was not initiated while there are still unfinished tasks at a previous level. For this case study that needs to compute 620,172 tasks, working with a user base of seven volunteer computers with various hardware, software, and network capabilities, the model was solved approximately in 11 min (exactly 661.02 s) for the entire case study. During the computation, the number of tasks left on the database decreased near-linearly.

To understand both the performance constraints and scalability of the framework, we also ran separate simulations with 20, 30, 50, 70, and 100 volunteer computers identical to the Computer 1 in Table 2 each contributing two workers. These simulations took 280.2, 187.1, 115.8, 80.1, and 62.3 s to finish all hillslope computations, respectively. It should be noted that the simulation may not reflect the real-world settings completely but considering that the framework is able to handle up to 200 requests per second, it nevertheless provides concise remarks about the performance of the framework.

It warrants mention that the use case we selected for test purposes in this study relies on a time-sensitive task; time-sensitive tasks need to be done almost simultaneously. By its very nature, distributed volunteer computing depends on the number of participants, and this may be a drawback. Nevertheless, we expect to have more participants when flood prediction models are needed most due to extreme weather events and when it is important to finalize all computations. Because of this fact, the volunteer computing systems can be used more efficiently when it is more important to have intended computations made without any time

constraints. Even though the limited number of volunteer participants seems to be crucial for the success of such a distributed volunteer computing task, as mentioned earlier, the required number of volunteers for this study can easily be met through user base of the IFIS (over 300,000 unique users as of December 2018).

CONCLUSIONS

The main goal of this study is to develop a web-based voluntary scientific computing framework and a platform with the capability to use multiple cores and take advantage of low-level technologies to enhance computation speeds. There are many computational approaches such as grid computing and cloud computing to achieve the similar goals to distribute the computational load to multiple nodes. These approaches can provide more stable environment than distributed volunteer computing with regard to accessibility of the computational power. However, relative low cost of the distributed volunteer computing makes it a favorable choice for non-time-sensitive scientific computing tasks. Subsequently, time-sensitive tasks such as running a flood forecasting model on recent measurements still can be done by distributed volunteer computing via operational web systems with large user bases. The web-based nature of the proposed framework makes it easy for any researcher to adapt the framework and use for their project and utilize existing user base of their labs or institutes with limited effort.

While the framework is specialized to work efficiently with hydrologic computational tasks, it is also designed for use by scientists from other disciplines. The model chosen for the study allows easy parallelization of the tasks and is large enough to be used as an example for a distributed computing project. This browser-based approach provides easy adaptation and maintenance for both project managers and volunteers. The results of performance tests suggest that the framework is capable of handling scientific computation tasks and scales efficiently with multicore systems and multiple users. The hierarchical dependency structure of the models or data can be handled at the server level during scheduling. The framework empowers the use of low-level computing and programming technologies, which provides

a significant improvement in computational performance. Since the overhead caused by large numbers of volunteers is shown to be not significant empirically, the work here has inferred that web browsers and usage of computationally capable devices around the world can facilitate the participation of volunteers in scientific efforts, including data processing and modeling.

Our large-scale hydrological case study shows that the framework can even handle an operational real-time system using a limited number of volunteer computers, supporting flood prediction efforts for disaster preparedness. Although it is not ideal or recommended to rely on volunteer computing for operational systems, this case study shows the potential of the framework for resource-constrained communities. The current version of the framework can run with models and analysis written in JavaScript, and codes powered by WebAssembly and compiled by Emscripten. This allows a significant improvement in the performance of the framework as new technologies are allowed in web-based systems. The web-based nature of the platform will enable new capabilities without updating the client-side software. The generic structure of the framework also allows researchers to use the framework for any distributable scientific task by loading their model and datasets to the framework by creating communication medium as mentioned in the Project Creation subsection under Methods. Besides being open to any web-based low-level technology implementations over JavaScript, it is worth mentioning that the framework presented here can be extended by implementing it in a serverless decentralized fashion as a future perspective. As another potential future work on the framework, in order to ensure data security and discard possible malware attacks on the server, a medium could be implemented between the user and the server to have the users sign their outputs with a unique key given to them.

REFERENCES

- Almasi, G. & Gottlieb, A. 1989 *Highly Parallel Computing*. Benjamin/Cummings, Menlo Park, CA.
- Anderson, D. P. 2004 Boinc: A system for public-resource computing and storage. In: *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*, pp. 4–10.

- Anderson, D. P. & Fedak, G. 2006 The computational and storage potential of volunteer computing. In: *CCGRID '06 Proceedings of the Sixth IEEE International Symposium*, 1, pp. 73–80.
- Atiya, H. & Welch, J. 2004 *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*, Vol. 19. John Wiley & Sons.
- Ayalew, T. B. & Krajewski, W. F. 2017 Effect of river network geometry on flood frequency: a tale of two watersheds in Iowa. *Journal of Hydrologic Engineering* **22** (8), 06017004-1–06017004-7.
- Baratloo, A., Karaul, M., Kedem, Z. M. & Wijckoff, P. 1999 Charlotte: metacomputing on the web. *Future Generation Computer Systems* **15** (5), 559–570.
- Black, M. & Edgar, W. 2009 Exploring mobile devices as grid resources: Using an x86 virtual machine to run BOINC on an iPhone. In: 2009 10th IEEE/ACM International Conference on Grid Computing, pp. 9–16.
- Boldrin, F., Taddia, C. & Mazzini, G. 2007 Distributed computing through web browser. In: Proceedings of the 66th Vehicular Technology Conference, VTC-2007 Fall, IEEE, pp. 2020–2024.
- Bonney, R., Cooper, C. B., Dickinson, J., Kelling, S., Phillips, T., Rosenberg, K. V. & Shirk, J. 2009 Citizen science: a developing tool for expanding science knowledge and scientific literacy. *BioScience* **59** (11), 977–984.
- Cappello, P., Christiansen, B., Ionescu, M. F., Neary, M. O., Schausser, K. E. & Wu, D. 1997 Javelin: Internet-based parallel computing using Java. In: ACM Workshop on Java for Science and Engineering Computation.
- Cohn, J. P. 2008 Citizen science: can volunteers do real research? *BioScience* **58** (3), 192–197.
- Cole, N., Desell, T., González, D. L., de Vega, F. F., Magdon-Ismail, M., Newberg, H., Szymanski, B. & Varela, C. 2010 Evolutionary algorithms on volunteer computing platforms: The milkyway@home project. In: *Parallel and Distributed Computational Intelligence*. Springer, Berlin, pp. 63–90.
- Costa, F., Silva, L., Fedak, G. & Kelley, I. 2008 Optimizing the data distribution layer of boinc with bittorrent. In: 2008 IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2008. IEEE International Symposium, pp. 1–8.
- Coulouris, G. F., Dollimore, J. & Kindberg, T. 2005 *Distributed Systems: Concepts and Design*. Addison-Wesley, New York.
- Culler, D. E., Pal Singh, J. & Gupta, A. 1999 *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann Publishers, San Francisco.
- Cunha, L. K., Mandapaka, P. V., Krajewski, W. F., Mantilla, R. & Bradley, A. A. 2012 Impact of radar-rainfall error structure on estimated flood magnitude across scales: an investigation based on a parsimonious distributed hydrological model. *Water Resources Research* **48** (10). <https://doi.org/10.1029/2012WR012138>.
- Cushing, R., Putra, G. H. H., Koulouzis, S., Belloum, A., Bubak, M. & De Laat, C. 2013 Distributed computing on an ensemble of browsers. *IEEE Internet Computing* **17** (5), 54–61.
- Das, R., Qian, B., Raman, S., Vernon, R., Thompson, J., Bradley, P., Khare, S., Tyka, M. D., Bhat, D., Chivian, D. & Kim, D. E. 2007 Structure prediction for CASP7 targets using extensive all-atom refinement with Rosetta@home. *Proteins: Structure, Function, and Bioinformatics* **69** (S8), 118–128.
- Demir, I. & Beck, M. B. 2009 GWIS: a prototype information system for Georgia watersheds. In: Georgia Water Resources Conference: Regional Water Management Opportunities, UGA.
- Demir, I. & Krajewski, W. F. 2013 Towards an integrated flood information system: centralized data access, analysis, and visualization. *Environmental Modelling & Software* **50**, 77–84.
- Demir, I., Jiang, F., Walker, R. V., Parker, A. K. & Beck, M. B. 2009 Information systems and social legitimacy scientific visualization of water quality. In: *IEEE International Conference on Systems, Man and Cybernetics, SMC 2009*, pp. 1067–1072.
- Demir, I., Conover, H., Krajewski, W. F., Seo, B. C., Goska, R., He, Y., McEniry, M. F., Graves, S. J. & Petersen, W. 2015 Data-enabled field experiment planning, management, and research using cyberinfrastructure. *Journal of Hydrometeorology* **16** (3), 1155–1170.
- Demir, I., Yildirim, E., Sermet, Y. & Sit, M. A. 2018 FLOODSS: Iowa flood information system as a generalized flood cyberinfrastructure. *International Journal of River Basin Management* **16** (3), 393–400.
- DeThread 2016 Available from: <https://github.com/DeThread/dethread/> (accessed 14 June 2018).
- Duffy, C., Gil, Y., Deelman, E., Marru, S., Pierce, M., Demir, I. & Wiener, G. 2012 Designing a road map for geoscience workflows. *Eos, Transactions American Geophysical Union* **93** (24), 225–226.
- Durrani, M. N. & Shamsi, J. A. 2014 Volunteer computing: requirements, challenges, and solutions. *Journal of Network and Computer Applications* **39**, 369–380.
- Einstein@Home 2005 Available from: <https://einsteinathome.org/> (accessed 14 June 2018).
- Emscripten 2015 Available from: <https://github.com/kripken/emscripten/> (accessed 14 June 2018).
- Ferster, C. J. & Coops, N. C. 2013 A review of earth observation using mobile personal communication devices. *Computers & Geosciences* **51**, 339–349.
- Fienen, M. N. & Lowry, C. S. 2012 Social. water – a crowdsourcing tool for environmental data acquisition. *Computers & Geosciences* **49**, 164–169.
- Gil, Y., David, C. H., Demir, I., Essawy, B. T., Fulweiler, R. W., Goodall, J. L., Karlstrom, L., Lee, H., Mills, H. J., Oh, J. H. & Pierce, S. A. 2016 Toward the geoscience paper of the future: best practices for documenting and sharing research from data to software to provenance. *Earth and Space Science* **3** (10), 388–415.
- Golem 2015 Available from: <https://golem.network/> (accessed 26 August 2018).
- Granell, C., Havlik, D., Schade, S., Sabeur, Z., Delaney, C., Pielorz, J., Usländer, T., Mazzetti, P., Schleidt, K., Kobernus, M. &

- Havlik, F. 2016 [Future internet technologies for environmental applications](#). *Environmental Modelling & Software* **78**, 1–15.
- GridCoin 2013 Available from: <https://gridcoin.us/> (accessed 26 August 2018).
- Gullapalli, S. 2016 Atlas: an intelligent, performant framework for web-based grid computing. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 1154–1156.
- Horita, F. E., de Albuquerque, J. P., Degrossi, L. C., Mendiondo, E. M. & Ueyama, J. 2015 [Development of a spatial decision support system for flood risk management in Brazil that combines volunteered geographic information with wireless sensor networks](#). *Computers & Geosciences* **80**, 84–94.
- Jones, C. S., Davis, C. A., Drake, C. W., Schilling, K. E., Debionne, S. H., Gilles, D. W., Demir, I. & Weber, L. J. 2018 [Iowa statewide stream nitrate load calculated using in situ sensor network](#). *JAWRA Journal of the American Water Resources Association* **54** (2), 471–486.
- Jonoski, A., Almoradie, A., Khan, K., Popescu, I. & Van Andel, S. J. 2013 [Google android mobile phone applications for water quality information management](#). *Journal of Hydroinformatics* **15** (4), 1137–1149.
- Korpela, E. J. 2012 [SETI@ home, BOINC, and volunteer distributed computing](#). *Annual Review of Earth and Planetary Sciences* **40**, 69–87.
- Krajewski, W. F., Ceynar, D., Demir, I., Goska, R., Kruger, A., Langel, C., Mantilla, R., Niemeier, J., Quintero, F., Seo, B. C. & Small, S. J. 2017 [Real-time flood forecasting and information system for the State of Iowa](#). *Bulletin of the American Meteorological Society* **98** (3), 539–554.
- Lampert, L. & Lynch, N. 1989 *Chapter on Distributed Computing* (No. MIT/LCS/TM-384). Massachusetts Inst of Tech Cambridge Lab for Computer Science, Cambridge, MA.
- Larson, S. M., Snow, C. D., Shirts, M. & Pande, V. S. 2009 [Folding@ Home and Genome@ Home: using distributed computing to tackle previously intractable problems in computational biology](#). *arXiv preprint arXiv:0901.0866*.
- Locust 2011 Available from: <https://locust.io/> (accessed 17 September 2018).
- Mantilla, R. & Gupta, V. K. 2005 [A GIS numerical framework to study the process basis of scaling statistics in river networks](#). *IEEE Geoscience and Remote Sensing Letters* **2** (4), 404–408. doi:10.1109/LGRS.2005.853571.
- Nakada, H., Sato, M. & Sekiguchi, S. 1999 [Design and implementations of Ninf: towards a global computing infrastructure](#). *Future Generation Computer Systems* **15** (5), 649–658.
- Quintero, F., Krajewski, W. F., Mantilla, R., Small, S. & Seo, B.-C. 2016 [A spatial–dynamical framework for evaluation of satellite rainfall products for flood prediction](#). *Journal of Hydrometeorology* **17** (8), 2137–2154. doi:10.1175/JHM-D-15-0195.1.
- Sarmenta, L. F. & Hirano, S. 1999 [Bayanihan: building and studying web-based volunteer computing systems using Java](#). *Future Generation Computer Systems* **15** (5), 675–686.
- Sermet, Y. & Demir, I. 2018 [An intelligent system on knowledge generation and communication about flooding](#). *Environmental Modelling & Software* **108**, 51–60.
- Shirts, M. & Pande, V. S. 2000 [Screen savers of the world unite!](#). *Science* **290** (5498), 1903–1904.
- Small, S. J., Jay, L. O., Mantilla, R., Curtu, R., Cunha, L. K., Fonley, M. & Krajewski, W. F. 2013 [An asynchronous solver for systems of ODEs linked by a directed tree structure](#). *Advances in Water Resources* **53**, 23–32.
- Stone, H. S. & Bokhari, S. H. 1978 [Control of distributed processes](#). *Computer* **11** (7), 97–106.
- Sullivan, W. T., Werthimer, D., Bowyer, S., Cobb, J., Gedye, D. & Anderson, D. 1997 [A new major SETI project based on Project Serendip data and 100,000 personal computers](#). *International Astronomical Union Colloquium* **161**, 729–734.
- Theodoropoulos, D., Chrysos, G., Koidis, I., Charitopoulos, G., Pissadakis, E., Varikos, A., Pnevmatikatos, D., Smaragdous, G., Strydis, C. & Zervos, N. 2016 [mCluster: a software framework for portable device-based volunteer computing](#). In: 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 336–341.
- Weber, L. J., Muste, M., Bradley, A. A., Amado, A. A., Demir, I., Drake, C. W., Krajewski, W. F., Loeser, T. J., Politano, M. S., Shea, B. R. & Thomas, N. W. 2018 [The Iowa Watersheds Project: Iowa's prototype for engaging communities and professionals in watershed hazard mitigation](#). *International Journal of River Basin Management* **16** (3), 315–328.
- Wiggins, A. & Crowston, K. 2011 [From conservation to crowdsourcing: a typology of citizen science](#). In: 2011 44th Hawaii International Conference on System Sciences (HICSS), pp. 1–10.

First received 19 June 2018; accepted in revised form 1 November 2019. Available online 19 December 2019