Preliminary version. Final version to appear in 8th IEEE International Conference on eScience 2012. Chicago Illinois. Oct 8-12 2012.

Calibration of Watershed Models using Cloud Computing

Marty Humphrey, Norm Beekwilder
Department of Computer Science
University of Virginia
Charlottesville, VA 22904

Jonathan L. Goodall, Mehmet B. Ercan
Department of Civil and Environmental Engineering
University of South Carolina
Columbia, SC 29208

Abstract— **Understanding hydrologic systems at the scale of large watersheds and river basins is critically important to society when faced with extreme events, such as floods and droughts, or with concerns about water quality. A critical requirement of watershed modeling is model calibration, in which the computational model's parameters are varied during a search algorithm in order to find the best match against physically-observed phenomena such as streamflow. Because it is generally performed on a laptop computer, this calibration phase can be very time-consuming, significantly limiting the ability of a hydrologist to experiment with different models. In this paper, we describe our system for watershed model calibration using cloud computing, specifically Microsoft Windows Azure. With a representative watershed model whose calibration takes 11.4 hours on a commodity laptop, our cloud-based system calibrates the watershed model in 43.32 minutes using 16 cloud cores (15.78x speedup), 11.76 minutes using 64 cloud cores (58.13x speedup), and 5.03 minutes using 256 cloud cores (135.89x speedup). We believe that such speed-ups offer the potential toward real-time interactive model creation with continuous calibration, ushering in a new paradigm for watershed modeling.**

Keywords-cloud computing, watershed modeling, model calibration, SWAT, Windows Azure

I. Introduction

Understanding hydrologic systems at the scale of large watersheds and river basins is critically important to society when faced with extreme events, such as floods and droughts, or with concerns about water quality. Climate change and increasing population are further complicating watershed-scale prediction by placing additional uncertainty on future water resource conditions.

Computational tools are critical to hydrologic analysis and simulation. However, today, many of the models that are used for analysis of hydrologic systems do not make use of recent advances in computing and data resources, making them less effective tools for addressing current water resource challenges. This challenge extends beyond simply developing the simulation code and includes the entire workflow needed to support models from data collection, to data preparation, to modeling, to post-processing model results to support decision makers.

One of the specific goals of the hydrologic modeling community is to scale-up models to better address complex water resource problems (droughts, floods, water quality). This scaling-up essentially results in considering larger and more complex physical domains within hydrologic models. For example, a few decades ago water quality modeling typically only took into account so-called point source discharges of pollution to water bodies. Now models must also take into account non-point sources, i.e. runoff from agricultural and urban lands to water bodies. This difference in scope introduces a significant change in the complexity of the model from considering only river systems to now consider the entire watershed that drains to the river as well.

Because of the complexity introduced by needing to simulate hydrology at a watershed-scale, an important step in the modeling process is the assessment of the model against observed historical conditions measured within the watershed, such as streamflow measured by sensors in the watershed outlet. That is, after a candidate model is produced from available spatial datasets (elevation, land use, soils, etc.), it must be "calibrated" in a computationally-expensive process whereby a subset of parameters of the model thought to be uncertain are selected (each model can have dozens of parameters), values are set for each parameter, the model is used as the basis of a simulation, and the results of the simulation are compared against observed measurements. The number of parameter sets and values considered is typically very large in order to attain sufficient confidence – typically a desktop computer must run for hours or even days in order to run a sufficient number of simulations of the model in order to calibrate. Increasingly, the duration of this necessary calibration step is impeding hydrologic modeling as, simply, scientists and engineers are limited in how well they are able to calibrate their model given available time and computational resources. While there have been some efforts to use multiple computers—in either a networked situation or in a compute cluster configuration—to speed up the calibration [1][2], perhaps the most significant limitation of these efforts is the relatively rigid and static use of computational resources.

In this paper, we report on our cloud-based system for watershed model calibration. We currently support the SWAT watershed model [3][4], and we are using the Microsoft Windows Azure platform [5]. We have created a Web-based portal for the watershed modeling community, and we anticipate that usage of this portal will be sporadic, which maps well to the cloud's ability to ramp up and ramp down dynamically. In addition to the flexibility offered by the cloud model, we are beginning to use cloud storage as the basis for community-based sharing of models and raw (and processed) input data for the hydrology community. More specifically, in this paper we describe:

- the design and implementation of the watershed model calibration Web portal – attempting to achieve a delicate balance between *hiding* and *exposing* parts of the back-end cloud infrastructure

- the experiences of adapting our "MODISAzure" framework [6] for use in watershed model calibration (including a "CloudBurst" approach [7], which means that jobs are attempted to be executed using local enterprise resources first, but – when needed – jobs are pushed to "the cloud")
- the adaptation of an existing watershed model calibration search algorithm (DDS [8][9]) specifically to leverage a cloud environment such as Windows Azure
- the evaluation of Windows Azure for performing this watershed model calibration, specifically focusing on the computational efficiency as a function of the number of Windows Azure cores used.

We believe that our system makes efficient use of cloud resources. With a representative watershed model whose calibration takes 11.4 hours on a commodity laptop, our cloud-based system calibrates the watershed model in 43.32 minutes using 16 cloud cores (15.78x speedup), 11.76 minutes using 64 cloud cores (58.13x speedup), and 5.03 minutes using 256 cloud cores (135.89x speedup). We believe that such speed-ups offer the potential toward real-time interactive model creation with continuous calibration, ushering in a new paradigm for watershed modeling.

The rest of this paper is organized as follows. Section II contains related work. Section III gives background on SWAT model calibration and the Microsoft Windows Azure cloud. Section IV gives the details of our architecture that uses Windows Azure. Section V contains the evaluation and discussion, and Section VI contains the conclusion.

II. RELATED WORK

A number of efforts have been focusing on using the cloud to perform scientific experiments (e.g., CloudBLAST [10], coupled atmospheric-ocean climate models [11], data mining [12], astrophysics [13], astronomy [14][15], high energy physics [16]). Windows Azure has been used as the platform for executing BLAST very effectively [17][18].

While cloud computing has not been applied to hydrologic system modeling, there have been an increasing number of applications of advanced computing architectures and approaches for simulating hydrologic systems. One area of interest has been utilizing modern software engineering paradigms in order to support community-based scientific models (CSDMS [19], ESMF [20], OMS [21] and OpenMI [22] are examples). The recent application of modern software engineering paradigms such as component-based [23] and service-oriented architectures [24] to scientific computing provides a technical means for advancing cross-disciplinary scientific modeling that can be advanced by a large community. Hydrologists have long noted that water resource management requires sophisticated models capable of predicting how coupled hydrologic, climate, and socio-economic systems respond to change in order to estimate response in one system (e.g. water quality) caused by changes within another system (e.g. energy policy) [25][26][27]. Most existing earth science and environmental management models are only starting to consider complex system interactions such as feedback loops across disciplinary models, because each subdiscipline builds their own models, and fully coupling models between disciplinary boundaries remains a major research challenge.

III. BACKGROUND: SWAT MODEL CALIBRATION AND WINDOWS AZURE

We first describe the specific watershed modeling framework we are using (SWAT) and also describe existing approaches to calibrate SWAT models. We then give an overview of the Microsoft cloud: Windows Azure.

A. Soil and Water Assessment Tool (SWAT) Model Calibration

Soil and Water Assessment Tool (SWAT) [3][4] is a comprehensive modeling framework that supports weather, hydrology, soil, plant growth, nutrients, pesticides, bacteria and pathogens, and land management. In SWAT, a watershed is divided into subwatersheds, which are then further divided into hydrologic response units (HRUs). The potential inputs to a SWAT model include daily precipitation, max/min temperature, relative humidity, wind speed data, etc. During the "execution" of a model, the water balance is simulated for each HRU. The runoff generated within each HRU is then aggregated at the subwatershed level and routed through the streams, ponds, wetlands, etc., to the watershed outlet.

A key step in hydrologic modeling is both the calibration and validation of a model. In SWAT, the Nash–Sutcliffe model efficiency coefficient (E) is typically used and represents how well the simulated result matches the observed result (e.g., for streamflow). The "goodness" of the model increases as the E coefficient increases up to a value of 1, which represents a perfect model.

Before a watershed model can be used as the basis for prediction (e.g., of streamflow), it must be calibrated to observed historical conditions within the watershed. Calibration can be manual such as a trial-and-error approach, in which an expert selects which parameters to vary and by how much, or it can be automated. There are a number of examples of techniques to automate the watershed model calibration procedure. We focus on the calibration of SWAT models, but each modeling framework (e.g., HSPF [28]) has similar approaches. Automated approaches are usually Monte Carlo-based, such as the Shuffled Complex Evolution (SCE) [29]. The approach used in our system is called the Dynamically Dimensioned Search (DDS [8]) and has been shown to be more efficient and more effective than SCE (e.g., "In two cases, DDS requires only 15–20% of the number of model evaluations used by SCE in order to find equally good values of the objective function." [8]). The goal of DDS is to begin by searching for a good candidate parameter set in the global search space and then switching to a local search space by gradually reducing the number of candidate parameters to modify as well as the range of acceptable values for each candidate parameter. During the search, the next candidate is chosen via a random and uniform perturbation of a particular parameter. Search continues for either a fixed number of iterations or until termination criteria is satisfied (e.g., a sufficient value of E is achieved).

There have been recent efforts to attempt to use multiple computers to speed up the SWAT model calibration procedure. As discussed later in this paper, the DDS approach is not trivially parallelizable because it uses the results of the most recent parameter set and values as the basis of the *next* parameter set and values. In other words, DDS is implicitly a sequential algorithm. However, DDS has been used as the basis of two parallel implementations: EP-DDS and P-DDS [9]. Other efforts use a computer cluster to speed up the calibration [1][2], including via the use of Genetic Algorithms (GA) [30]. A recent study has investigated DDS, "Particle Swarm Optimization" (PSO), and a Generic Algorithm (GA), showing favorable results for DDS [31]. It is important to note that none of these efforts utilize cloud computing to perform watershed model calibration.

B. *Windows Azure*

Windows Azure was announced by Microsoft as its cloud computing platform at its Professional Developers Conference (PDC) Nov 2008. Initially, Windows Azure focused on providing a .NET-based hosting platform that was integrated into a virtual machine abstraction. Thus, developers who are familiar with .NET application development could take advantage of this homogeneous cloud environment and develop applications for Windows Azure just like ordinary .NET applications by using Visual Studio. In contrast to this "Platform as a Service" (PaaS), Amazon's EC2 has focused on support for virtual machine technology (aka Infrastructure as a Service, or IaaS). Microsoft has more recently augmented its PaaS with IaaS as well. In both EC2 and Windows Azure, users can customize the environment for their application by installing specific software or by purchasing particular machine images.

In Windows Azure, the virtual machine instances can be separated into three different roles: the front-end website hosting server instances are called Web Roles, the back-end computational instances called Worker Roles, and the more recent Virtual Machine (VM) roles. Developers can specify the number of instances for roles at the deployment of their application or can dynamically adjust the number of instances at runtime.

Windows Azure provides three types of cloud storage services, in addition to SQL Azure:

- Blob service, the main storage service for storing durable large data items;
- Queue service, which provides a basic reliable queue model to allow asynchronous task dispatch and to enable service communication;
- Table service, which provides the structured storage in the form of tables and supports simple queries on partitions, row keys, and attributes.

The key aspect of cloud storage is that it is accessible via any virtual machine in Windows Azure (with the proper authentication/authorization). Therefore, while there is local storage available to a particular computation, it is assumed that one of the cloud storage services will be used if the data is to be shared across virtual machine instances.

Windows Azure continues to evolve. For example, on June 6 2012, Microsoft announced support for running Linux-based VMs in Windows Azure [32]. Microsoft also recently announced the wide-scale availability of its "Virtual Network", whereby giving the ability to create a virtual private network (VPN) between the local enterprise and Windows Azure (in later parts of this paper, we describe how we used this feature when it was in its "early adopter" version).

IV. SWAT Model calibration using Windows Azure

In this project, our broad goals are to use Windows Azure for watershed model data preparation (raw data exists, but files are large and require preprocessing, suitable for workflow processing), watershed model calibration, and large-scale watershed model execution (too large to execute on a single workstation). To date, we have focused on watershed model calibration, which is the focus of this paper.

A. *Requirements*

There were a number of critical requirements that we established for our cloud-based watershed model calibration system. The overall requirement of course is to provide a significant speed-up in watershed model calibration as compared to the typical operating environment of a hydrology researcher (which is assumed to be a commodity laptop computer). A number of key additional requirements were established, both from the perspective of the systems implementer as well as from the perspective of the hydrology researcher:

- Local (enterprise) resources should be used first (because latency is less, resources are largely already paid for, local resources are behind the firewall, etc.); if additional resources are needed, cloud computing should provide the capacity needed.
- As per the value/appeal of cloud computing, the level of back-end cloud resource consumption should match the level of activity by hydrology researchers. That is, an idle system should incur little to no recurring cost.
- The hydrology researcher should be able to use our system via a WWW browser, with as little a learning curve as possible.

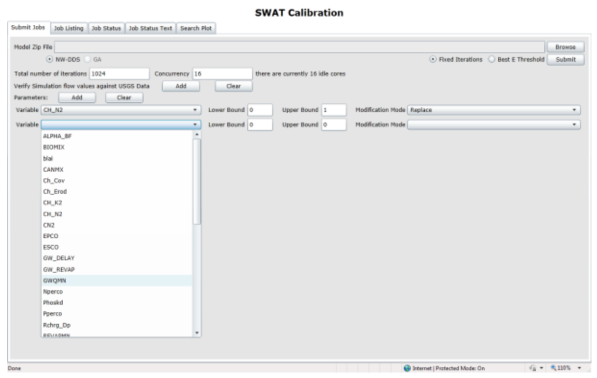
There were key additional constraints that we established to ensure future enhancements. That is, we attempted to create a current design that did not *prevent* the following architecturally:

- To evaluate the effectiveness and feasibility of a cloud-based watershed model calibration system, it was acceptable that the system only support a single (authenticated) user. However, the design should not hinder a multi-user system, in which end users are isolated from each. More specifically, end users should not see each other's models, and the computing resources available for one user should not be impacted by the use of computing resources by another user.

- The architecture should support potential cost-per-use passed on to the end user. This is important for sustainability – while we were certainly not interested in running this *for profit*, we recognize that we wanted to create a potentially long-lived service for the hydrology community, but it was unlikely that we could directly pay for its long-term operation.

B. *User Interface*

The user interacts with the system via a WWW browser. The submission frame is shown in Figure 1. The key aspect of the pane is the ability to select any number of watershed model parameters to modify and the ability to specify how each is to be modified by our system during the search process. The other key feature of the pane is its showing of the number of currently-idle cloud cores (the user can select any number from 1 up to the number of idle cores; note that currently we manually boot cores as part of the system administrator console).


SWAT Calibration

Submit Jobs | Job Listing | Job Status | Job Status Text | Search Plot

Model Zip File: Browse

(• NW-DDS) (GA) (• Fixed Iterations) (Best E Threshold) Submit

Total number of iterations: 1024 Concurrency: 16 there are currently 16 idle cores

Verify Simulation flow values against USGS Data: Add Clear

Parameters: Add Clear

Variable: CH_N2 Lower Bound: 0 Upper Bound: 1 Modification Mode: Replace

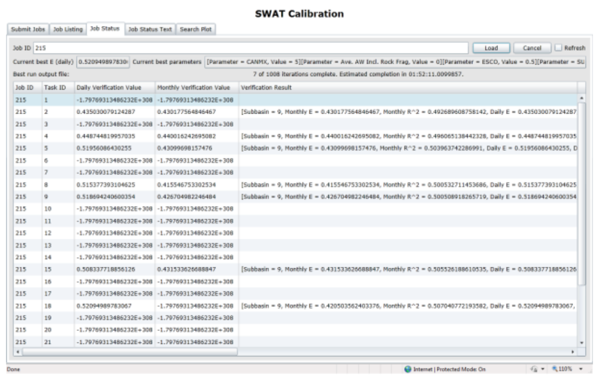
Variable: Lower Bound: 0 Upper Bound: 0 Modification Mode:

ALPHA_BF
BIOMIX
blai
CANMX
Ch_Cov
Ch_Erod
CH_K2
CH_N2
CN2
EPCO
ESCO
GW_DELAY
GW_REVAP
GWQMN
Nperco
Phoskd
Pperco
Rchrg_Dp

Done Internet | Protected Mode: On 110%

Figure 1. Submitting a watershed model calibration request.

After job(s) have been submitted, the user has a number of additional panes available to view the status or previous results. For example, the user has the ability to check the current status of a model calibration submission, as shown in Figure 2. This pane shows the status of all of the model executions, as well as the currently-best model found (e.g., so that the user can choose to manually abort the calibration request if desired.)


SWAT Calibration

Submit Jobs | Job Listing | Job Status | Job Status Text | Search Plot

Job ID: 215 Load Cancel Refresh

Current best E (daily): 0.520949897830 Current best parameters: [Parameter = CANMX, Value = 5][Parameter = Ave. AW Incl. Rock Frag, Value = 0][Parameter = ESCO, Value = 0.5][Parameter = SU

Best run output file: 7 of 1008 iterations complete. Estimated completion in 01:52:11.0099857.

Job ID	Task ID	Daily Verification Value	Monthly Verification Value	Verification Result
215	1	-1.79769313486232E+308	-1.79769313486232E+308	
215	2	0.435030079124287	0.430177564846467	[Subbasin = 9, Monthly E = 0.430177564846467, Monthly R^2 = 0.492689608758142, Daily E = 0.435030079124287
215	3	-1.79769313486232E+308	-1.79769313486232E+308	
215	4	0.448744819957035	0.440016242695082	[Subbasin = 9, Monthly E = 0.440016242695082, Monthly R^2 = 0.496065138442328, Daily E = 0.448744819957035
215	5	0.51956086430255	0.43099698157476	[Subbasin = 9, Monthly E = 0.43099698157476, Monthly R^2 = 0.503963742286991, Daily E = 0.51956086430255, D
215	6	-1.79769313486232E+308	-1.79769313486232E+308	
215	7	-1.79769313486232E+308	-1.79769313486232E+308	
215	8	0.515377393104625	0.415546753302534	[Subbasin = 9, Monthly E = 0.415546753302534, Monthly R^2 = 0.500532711453686, Daily E = 0.515377393104625
215	9	0.518694240600354	0.426704982246484	[Subbasin = 9, Monthly E = 0.426704982246484, Monthly R^2 = 0.500508918265719, Daily E = 0.518694240600354
215	10	-1.79769313486232E+308	-1.79769313486232E+308	
215	11	-1.79769313486232E+308	-1.79769313486232E+308	
215	12	-1.79769313486232E+308	-1.79769313486232E+308	
215	13	-1.79769313486232E+308	-1.79769313486232E+308	
215	14	-1.79769313486232E+308	-1.79769313486232E+308	
215	15	0.508337718856126	0.431533626688847	[Subbasin = 9, Monthly E = 0.431533626688847, Monthly R^2 = 0.505526188610535, Daily E = 0.508337718856126
215	16	-1.79769313486232E+308	-1.79769313486232E+308	
215	17	-1.79769313486232E+308	-1.79769313486232E+308	
215	18	0.52094989783067	-1.79769313486232E+308	[Subbasin = 9, Monthly E = 0.420503562403376, Monthly R^2 = 0.507040772193582, Daily E = 0.52094989783067,
215	19	-1.79769313486232E+308	-1.79769313486232E+308	
215	20	-1.79769313486232E+308	-1.79769313486232E+308	
215	21	-1.79769313486232E+308	-1.79769313486232E+308	

Done Internet | Protected Mode: On 110%

Figure 2. Checking the status of a watershed model calibration.

Our user interface design is meant to be simple, specifically hiding most of the details of the cloud back-end both so that the hydrologist is not required to understand the computing infrastructure as well as to give us the ability to re-architect and re-configure the processing if necessary. The closest we get to exposing the back-end is via the number of cores the user requests (which we expose as "concurrency level"). We further note that we are in the process of replacing this low-level concurrency knob with the more-useful "requested duration to complete" (which we would translate into a "concurrency level"), which will in part be based on our project's research into deadline-based cloud computing [33][34]. This research will also be leveraged when (as part of future research) we auto-scale cloud resources instead of manually controlling the level of cloud resources manually.

C. *System Architecture: Cloudbursting*

The overall architecture of our system is shown in Figure 3. To satisfy the requirement of using local resources first, we leverage our MODIS cloudbursting architecture [7] (which itself highly leveraged our MODISAzure system [6], a Windows Azure-only application we created for large-scale satellite image processing). The nature of our MODIS cloudbursting architecture is to provide a framework by which any number of days/years of the MODIS satellite data could be analyzed in parallel via a researcher-specific application uploaded to the cloud – using one cloud core per day of satellite data (thus, for example, to process 1 year worth of data, we created 365 independent jobs that would be processed in parallel on 365 cores on Windows Azure). We adapted this system for our watershed model calibration system, particularly leveraging our database-centric approach for managing the parameters for a particular set of jobs.

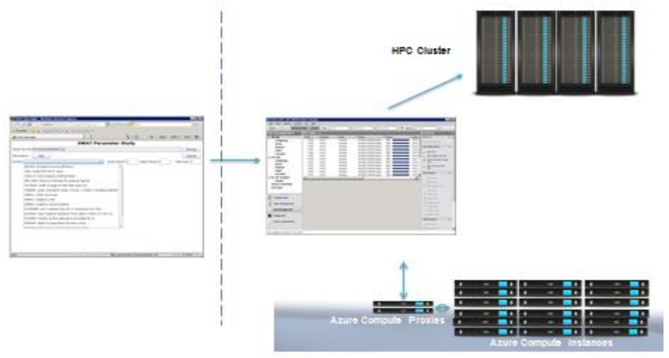


Figure 3. Overall system architecture utilizing cloudbursting.

The center of Figure 3 shows the Microsoft Windows HPC job scheduler [35], which inherently supports a cloudbursting model (whereby Windows Azure nodes can be dynamically "added" to an otherwise enterprise-only Windows Compute Cluster). When a user submits a watershed model calibration request that, for example, requests that 1000 model executions be performed in a search for the best match, 1000 jobs are submitted to the Windows HPC job scheduler. Then, the Windows HPC scheduler dispatches the jobs, preferring the local resources we have at the University of Virginia. When theses nodes are all executing SWAT model executions, we "burst" onto Windows Azure. Upon execution, the first action each job performs is to check back with the database to obtain its parameters for the SWAT model (We modify these parameters in the database, as described next in this section), This architecture has shown to be very flexible, as we are able to spin up new compute nodes in Windows Azure (and shut them down) whenever we desire. Within Windows Azure, we use Blob storage to prepare the nodes – i.e., when they boot, they automatically load our watershed modeling support code – but, because each execution of the model is largely independent, we do not use Blob storage after the VMs have booted (and rather rely on local storage within the VMs). We use the Windows Azure Connect CTP to provide VPN capability between the head node inside UVa and our Windows Azure compute nodes. Specifically, this VPN support allows the Windows Azure nodes to automatically see a disk on the head node as being just another "share" by which to read/write data – i.e., we use this for security and convenience, not generally for high-performance.

To better understand/predict how our system should dispatch jobs between local nodes and cloud nodes, we conducted experiments to measure a typical watershed model execution in our cloudbursting system (to minimize latency, we chose to exclusively use the datacenter closest to us, which was the Chicago Windows Azure datacenter). That is, if the overall execution time for a cloud node was too long, then it wouldn't make sense to *ever* execute this application via cloudbursting. Table I shows the results of executing a typical watershed model on our three available platforms, measuring both the duration to move the model and its input data and also the duration to execute the model (we measured output, but found it to be analogous in cost to the input costs, so we do not report that here). As shown in Table 1, there are no data transfer costs when the model is executed on the scientist laptop (the data and model are assumed to be on the scientist laptop). In the case of the enterprise node, the data and node are transferred within enterprise networks to a network node. The wide-area data transfer from the enterprise node at UVa to the Windows Azure node takes significant time via the VPN – however, the Windows Azure nodes are significantly more recent than our enterprise nodes, so the overall time is comparable. We note that it is not necessary that the overall time in the cloud be less than within the enterprise, as our queuing system will accommodate various dynamic latencies by only dispatching to those nodes that are available (whenever that might be).

TABLE I. EXECUTING A TYPICAL SWAT WATERSHED MODEL ON DIFFERENT PLATFORMS

	Stage-in	*Compute*	*Total*
Scientist laptop	0	55 sec	55 sec
Enterprise node	5 sec	60 sec	65 sec
Windows Azure (ex-large)	53 sec	32 sec	85 sec

D. System Internals

Having the overall system architecture in place, we next focused on how to control the actual search through the potential watershed model parameter sets attempting to find the parameter set that best matches observed measures. As mentioned earlier, because of its excellent results on a single-core system, we focused on the DDS algorithm as the basis of our cloud-based watershed model calibration system. Our adaption of the DDS algorithm is shown in Figure 4. At the top of the figure, the user input is passed to the overall SWAT calibration mechanism. The left of the figure depicts a potentially large number of model executions occurring in parallel, where each simulation commences via modification of the specific SWAT input files to match the chosen parameter values, followed by the execution of the model and the scoring of the model's predictions ("Calculate E" by comparing against actual streamflow measures). The right of Figure 3 shows the mechanism by which the calibration either continues or terminates (at which point the results are returned to the user via a browser page).

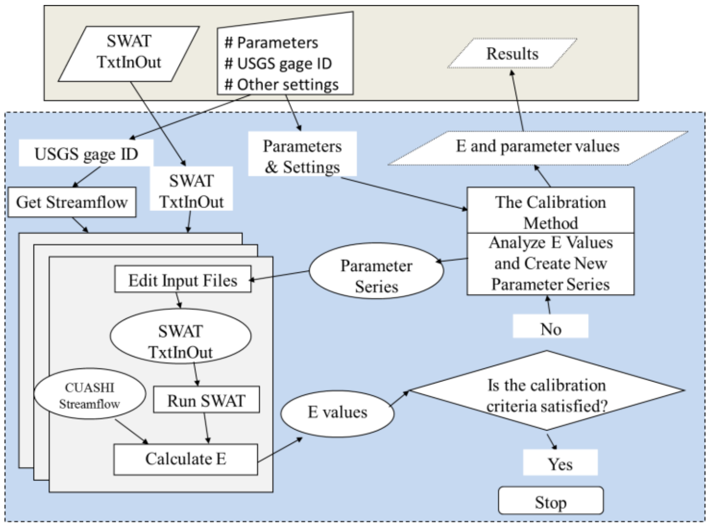


Figure 4. SWAT model calibration using Windows Azure.

As stated previously in this paper, DDS is an implicitly sequential algorithm in which the values for the next model execution are based on the results of the previous model execution. In our first cloud version of the DDS algorithm, we attempted to preserve (as much as possible) the behavior of the sequential algorithm by designing an approach by which *N* model executions would base their parameters on the last (best) model execution and execute in parallel, as shown in Figure 5.

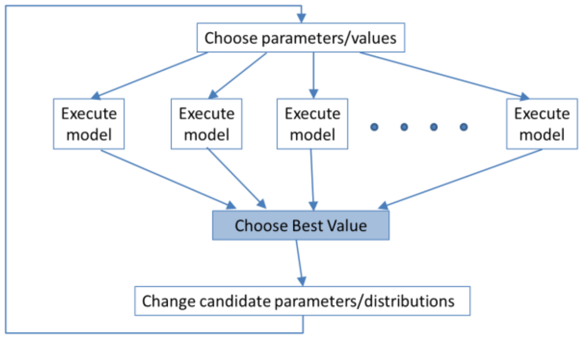


Figure 5. Parallel DDS Implementation, first version

At this point in the development process, we focused on the quality of the resultant model as compared to the watershed calibration mechanism that is distributed with the SWAT modeling framework (which runs on a desktop). Having confirmed that the cloud-based system produced a best-model that was roughly the same quality, we began to evaluate the efficiency of the platform, which we describe next.

V. Evaluation and Discussion

Whereas we were producing high-quality calibrations, our first measurements indicated that the cloud-based system was not as efficient as it should have been. For example, a simple model that required approximately 10 hours to calibrate via the built-in SWAT calibration routine for the desktop required 2 hours and 44 minutes in our cloud-based system with 16 cores – clearly not using the cloud nodes efficiently. (Note: we chose to use only the largest-sized VMs in Windows Azure, to simplify the booting/halting overhead). In this section, we describe how we measured our cloud-based system and improved it. In the rest of this section, we focus on a particular watershed model, but have found similar results for other models, both larger and smaller. In addition, to focus on worst-case, the results described in this section are *without* any local (enterprise) nodes.

We measured the number of cores that were active as a function of time, shown in Figure 6. This data led us to conclude that, while attempting to preserve the behavioral characteristics of the original DDS was important, the synchronization point in our implementation ("choose best value" in Figure 5) was having too big a negative impact on our overall efficiency.

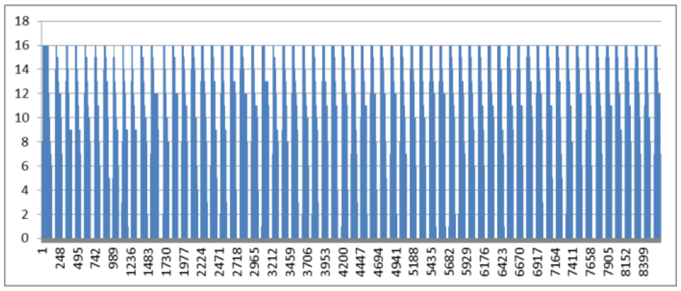


Figure 6. Parallel DDS implementation idle time.

Before potentially changing our implementation of DDS, we also wanted to determine what, specifically, each core was doing. We therefore instrumented our implementation along major categories, resulting in data shown in Figure 7, which shows the breakdown for each of the 1000 executions of the model. The biggest lesson reinforced from the data in Figure 7 was: *be very careful when crossing the cloud/enterprise boundary* (e.g., whenever possible, only move data once). This was important for both the "Stage In" as well as "Stage Out" phases. In "Stage in", we took a simple approach whereby we get the SWAT model and input data into place for model execution on a VM in the cloud by copying the particular SWAT model from our head node (within the enterprise). Although the model was relatively small, many nodes could potentially be attempting this copy in parallel. To remedy this, we changed our implementation so that the model was copied into blob storage and no longer retrieved every time from the head node (it was retrieved from blob storage, and furthermore it was *not* retrieved if it was already present on the cloud VM.) This reduced the number of copies from 1000 to 1+ the number of VMs being used. To reduce the cost of the "Stage Out" we recognized that it was not the actual result of the model execution that was necessary for the hydrologist to see, it was the parameters for that particular model execution. In other words, we already kept the parameter set for each of the 1000 model executions in a SQL database, so we completely eliminated this Stage Out mechanism. This improvement was made possible by first determining exactly where the time in each model execution was spent, and then asking the hydrology researcher if this step was truly necessary (which it wasn't).

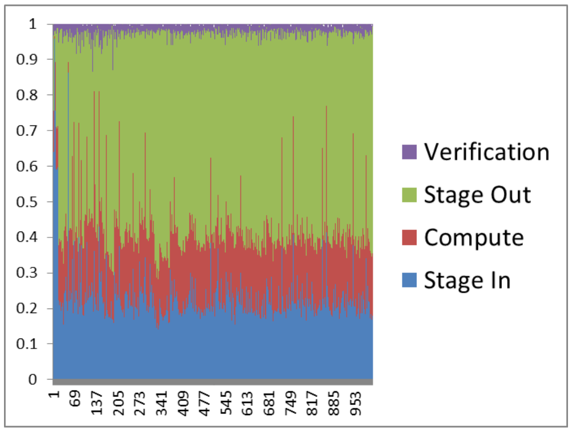


Figure 7. Performance of each model execution.

Next, we took one final step: eliminating the synchronization point in our algorithm (this synchronization point was the sole reason for idle cores in Figure 6). After numerous tests to confirm that the resulting algorithm was able to produce high-quality calibrated models, and confirming 100% utilization on our 16 cores (not shown in this paper), we re-ran our experiments from Figure 7, which resulted in the data shown in Figure 8. By re-architecting our approach whereby almost 95% of the time was spent executing watershed models, we were able to reduce our overall execution time.

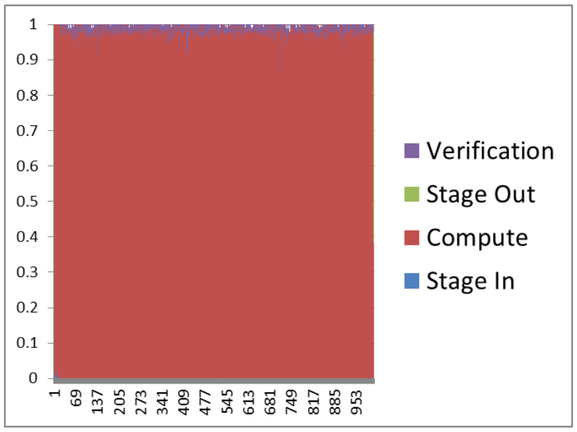


Figure 8. Performance of each model execution, new verstion.

Finally, we conducted experiments to determine the scalability of our approach. Figure 9 shows the same model calibration as used in previous results with 16 cores, Figure 10 shows with 64 cores, and Figure 11 shows with 256 cores.

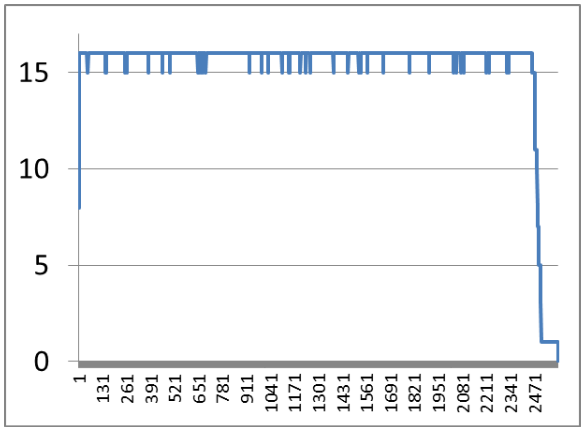


Figure 9. Watershed model calibration using cloud computing (16 cores)

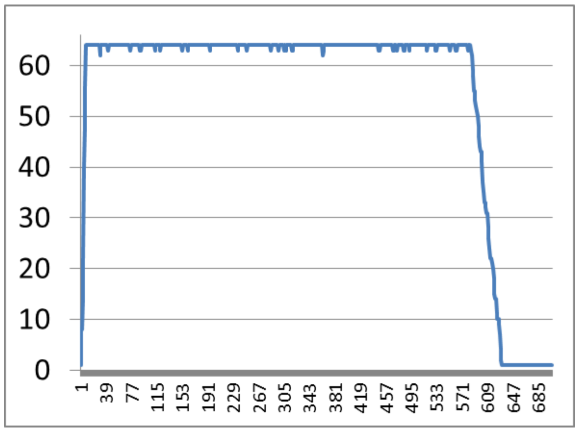


Figure 10. Watershed model calibration using cloud computing (64 cores)

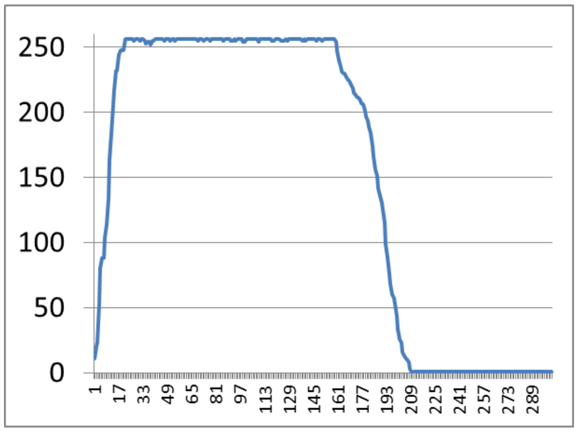


Figure 11. Watershed model calibration using cloud computing (256 cores)

Table 2 summarizes the results. While the speedup over the desktop option is significant, in general as the number of cores increase, the overall duration becomes dominated by *stragglers* (toward the end of the 1000 model executions, there will be idle cores). Additionally, because the hydrologist wants the actual results from the best model execution, we re-run the best model execution at the end to capture the actual results (which are generally discarded during the execution itself). This can be seen particularly in Figure 10 and 11. For these particular experiments, to capture the worst case behavior, we assume a single user and a single model calibration. However, in general practice, there will be multiple users/calibrations, meaning that these *n-1* cores will in general be consumed by another model calibration and will not be idle.

TABLE II. EFFICIENCY/SPEED-UP OF WATERSHED MODEL CALIBRATION SYSTEM USING CLOUD COMPUTING

	Duration	***Speedup***
Desktop, SWAT internal calibration (single threaded)	11.4 hours (41040 sec)	--
Windows Azure, Ex-large VM, 16 cores, NW-DDS	43.32 min (2599 sec)	15.78x
Windows Azure, Ex-large VM, 64 cores, NW-DDS	11.76 min (706 sec)	58.13x
Windows Azure, Ex-large VM, 256 cores, NW-DDS	5.03 min (302 sec)	135.89x

VI. CONCLUSION

Next-generation hydrology modeling will be increasingly sophisticated, encompassing a wide range of natural phenomena. Furthermore, calibrating models will soon cease to be practically feasible on desktop computers. In this paper, we have presented the design, implementation, and evaluation of a cloud-based system for watershed model calibration. With a representative watershed model whose calibration takes 11.4 hours on a commodity laptop, our cloud-based system calibrates the watershed model in 43.32 minutes using 16 cloud cores (15.78x speedup), 11.76 minutes using 64 cloud cores (58.13x speedup), and 5.03 minutes using 256 cloud cores (135.89x speedup). We believe that such speed-ups we achieve in our cloud-based watershed model calibration system offer the potential toward real-time interactive model creation with continuous calibration, ushering in a new paradigm for watershed modeling.

In the future, we plan to expand our support for different watershed modeling frameworks (e.g., HSPF), support additional user-defined calibration models, a visual representation of the search process, and provide support for community-based sharing of hydrologic models.

REFERENCES

[1] M. Sloboda, V. Sharma, S. Hood, D. Swayne, W. Booty, D. Lam and I. Wong. Innovative Autocalibration Techniques Using High Performance Computing. 18th World IMACS / MODSIM Congress, Cairns, Australia 13-17 July 2009.

[2] E. Rouholahnejad, K.C. Abbaspour, M. Vejdani, R. Srinivasan, R. Schulin, A. Lehmann, A parallelization framework for calibration of hydrological models, Environmental Modelling & Software, Available online 5 January 2012, ISSN 1364-8152, 10.1016/j.envsoft.2011.12.001. (http://www.sciencedirect.com/science/article/pii/S1364815211002829)

[3] P. Gassman, M.R. Reyes, C. Green, J. Arnold.The soil and water assessment tool: development, applications, and future research directions. Transactions on the ASABE. 50(4), pp. 1211-1250.

[4] Soil and Water Assessment Tool (SWAT). http://swatmodel.tamu.edu/

[5] Microsoft Windows Azure. http://www.windowsazure.com

[6] J. Li, D. Agarwal, M. Humphrey, C. van Ingen, K. Jackson, and Y. Ryu. eScience in the Cloud: A MODIS Satellite Data Reprojection and Reduction Pipeline in the Windows Azure Platform. In Proceedings of the 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2010), Apr 19-23, 2010. Atlanta, Georgia.

[7] M. Humphrey, Z. Hill, C. van Ingen, J. Jackson, and Y. Ryu. Assessing the Value of Cloudbursting: A Case Study of Satellite Image Processing on Windows Azure. In IEEE e-Science 2011 Conference. Stockholm, Sweden. Dec 5-8, 2011.

[8] B.A. Tolson and C. A. Shoemaker, Dynamically dimensioned search algorithm for computationally efficient watershed model calibration. *Water Resources Research,* 43, W01413, doi:10.1029/2005WR004723.

[9] B.A. Tolson, Sharma, V., Swayne, D., and Fan, L. (2007) "A parallel implementation of the Dynamically Dimensioned Search (DDS) algorithm." International Symposium on Environmental Software Systems (ISESS) 2007. Prague, Czech Republic. May 22 - 25, 2007.

[10] A. Matsunaga, M. Tsugawa, J. Fortes. CloudBLAST: Combining MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications Proceedings of 4th IEEE International Conference on e-Science (eScience 2008), Indianapolis, IN, Dec 2008.

[11] C. Evangelinos and C. Hill. Cloud Computing for parallel Scientific HPC Applications: Feasibility of running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2. Cloud Computing and its Applications (CCA-08). Chicago, IL. Oct 22-23, 2008.

[12] R. Grossman, Y. Gu, Data mining using high performance data clouds: experimental studies using sector and sphere. Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'08). Las Vegas, NV, 2008.

[13] G. Mackey, S. Sehrish, J. Bent, J. Lopez, S. Habib, J. Wang. Introducing Map-Reduce to High End Computing. 3rd petascale data storage workshop (held in conjunction with SC'08). Austin, TX. Mon Nov 17, 2008.

[14] E. Deelman, G. Singh, M. Livny, B. Berriman, J. Good. The Cost of Doing Science on the Cloud: The Montage Example. Proceedings of Supercomputing 2008, Austin, TX, Nov 15-21, 2008.

[15] K. Jackson, L. Ramakrishnan, K. Runge, and R. Thomas. "Seeking Supernovae in the Clouds: A Performance Study". 1st Workshop on Scientific Cloud Computing (ScienceCloud 2010), Chicago, IL, June 21 2010.

[16] M. Palankar, A. Lamnitchi, M. Ripeanu, S. Garfinkel. Amazon S3 for Science Grids: A Viable Solution? International Workshop on Data-Aware Distributed Computing, Boston, Massachusetts, June 2008.

[17] W. Lu, J. Jackson, and R. Barga. "AzureBlast: A Case Study of Developing Science Applications on the Cloud." 1st Workshop on Scientific Cloud Computing (ScienceCloud 2010), Chicago, IL, June 21 2010.

[18] W. Lu, J. Jackson, J. Ekanayake, R. S. Barga, N. Araujo. Performing Large Science Experiments on Azure: Pitfalls and Solutions. IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2010), Nov. 30-Dec.1, 2010, Bloomington, Ind.

[19] Community Surface Dynamics Modeling System (CSDMS). http://csdms.colorado.edu

[20] Earth System Modeling Framework (ESMF). http://www.earthsystemmodeling.org/

[21] Object Modeling System (OMS). http://www.javaforge.com/project/oms

[22] Open Modeling Interface (OpenMI). http://www.openmi.org

[23] B.A. Allan, R. Armstrong, D.E. Bernholdt, F. Bertrand, K. Chiu, T.L. Dahlgren, K. Damevski, W.R. Elwasif, T.G.W. Epperly, M. Govindaraju, D.S. Katz, J.A. Kohl, M. Krishnan, G. Kumfert, J.W. Larson, S. Lefantzi, M.J. Lewis, A.D. Malony, L.C. McInnes, J. Nieplocha, B. Norris, S.G. Parker, J. Ray, S. Shende, T.L. Windus, and S.J. Zhou, "A component architecture for high-performance scientific computing," International Journal of High Performance Computing Applications, vol. 20, 2006, pp. 163-202.

[24] I. Foster, "Service-Oriented Science," Science, vol. 308, 2005, pp. 814-817.

[25] N.S. Christensen, A.W. Wood, N. Voisin, D.P. Lettenmaier, and R.N. Palmer, "The effects of climate change on the hydrology and water resources of the Colorado River basin," Climatic Change, vol. 62, 2004, pp. 337–363.

[26] A.F. Hamlet and D.P. Lettenmaier, "Effects of climate change on hydrology and water resources in the Columbia River basin," Journal of the American Water Resources Association, vol. 35, 1999, pp. 1597–1623.

[27] J.T. Payne, A.W. Wood, A.F. Hamlet, R.N. Palmer, and D.P. Lettenmaier, "Mitigating the Effects of Climate Change on the Water Resources of the Columbia River Basin," Climatic Change, vol. 62, 2004, pp. 233-256.

[28] B.R. Bicknell, J.C. Imhoff, J.L. Kittle, A.S. Donigian and R.C. Johanson, R.C., 1997, Hydrological Simulation Program--Fortran, User's manual for version 11: U.S. Environmental Protection Agency, National Exposure Research Laboratory, Athens, Ga., EPA/600/R-97/080, 755 p., 1997.

[29] K. Eckhardt and J.G. Arnold. Automatic calibration of a distributed catchment model. *Journal of Hydrology.* Vol 251, issues 1-2,15 Sept 2001, pages 103-109.

[30] C.-T. Cheng, Wu, X.-Y. Chau, K. W. Multiple criteria rainfall-runoff model calibration using a parallel genetic algorithm in a cluster of computers. *Hydrological Sciences Journal.* 2005, Vol 50(6), pages 1069-1088.

[31] T. Francke, A. Bronster, and C. A. Shoemaker. Heuristic optimization methods for run-time intensive models (Dynamically Dimensioned Search, Particle Swarm optimization, GA) – a comparison of performance and parallel implementation using R. *Geophysical Research Abstracts.* Vol 12. EGU2010-8741, 2010.

[32] "Announcing New Windows Azure Services to Deliver 'Hybrid Cloud'". http://blogs.msdn.com/b/windowsazure/archive/2012/06/06/announcing-new-windows-azure-services-to-deliver-hybrid-cloud.aspx

[33] M. Mao, Jie Li and M. Humphrey. Cloud Auto-Scaling with Deadline and Budget Constraints. In Proceedings of 11th ACM/IEEE International Conference on Grid Computing (Grid 2010). Oct 25-28, 2010. Brussels, Belgium.

[34] M. Mao and M. Humphrey. Auto-Scaling to Minimize Cost and Meet Application Deadlines in Cloud Workflows. *Proceedings of Supercomputing 2011*, Seattle, WA, Nov 15-20, 2011.

[35] Microsoft Corporation. Windows HPC Server 2008 R2 Suite. http://www.microsoft.com/hpc/en/us/product/cluster-computing.aspx