University of Nevada, Reno

Framework for Large Data Processing under Constrained Resources

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy
in Computer Science and Engineering

by

Rui Wu

Advisors: Dr. Sergiu M. Dascalu and Dr. Frederick C. Harris, Jr.

May, 2018

THE GRADUATE SCHOOL

We recommend that the dissertation
prepared under our supervision by

RUI WU

Entitled

Framework for Large Data Processing under Constrained Resources

be accepted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

Sergiu Dascalu, Advisor

Frederick C Harris, Jr., Co-advisor

Lei Yang, Committee Member

Yantao Shen, Committee Member

Sage Hiibel, Graduate School Representative

David W. Zeh, Ph. D., Dean, Graduate School
May, 2018

Abstract

Data processing is used to uncover, transform, and classify information inside of data. Data-intensive research topics, such as environmental parameter prediction and sensor data imputation, require abundant computing power. To process big data efficiently, a server cluster is used for most cases. On one hand, a more powerful server cluster should be better. On the other hand, the powerful cluster will require a greater budget. How to balance this tradeoff is a challenge. Another challenge is how to improve communication between different nodes in a server cluster. The communication is usually through network and transportation speed is very slow.

In this thesis, we propose a data processing framework that can provide stable service with a limited budget. Stable service means the average waiting time and queue length do not change massively. The key of this framework control strategy is to import budget and local server computing power concepts into the $M/M/1/1/\infty/\infty$ queue model. To tackle the data communication challenge, data is compressed before transportation and decompressed when it arrives at its destination. An improved compression algorithm is proposed for this data transportation workflow, which leverages multiple GPUs and, to the best of our knowledge, is much faster than most other algorithms. Three data processing services that rely on the proposed framework are also presented in detail, to illustrate and prove the capabilities of our solution.

Dedication

I dedicate this thesis to my family and my academic family members who have supported me.

Acknowledgments

I would like to thank my advisers, Dr. Sergiu Dascalu and Dr. Fred Harris, and my committee members Dr. Lei Yang, Dr. Yangtao Shen, and Dr. Sage Hiibel for their time and suggestions. Dr. Dascalu and Dr. Harris treated me as their own family member. They assisted me financially and encouraged me to pursue my dream to be a professor. I really appreciate that my advisors and Dr. Lei Yang supported my academic job applications during the last year and I finally obtained my dream job. I also would like to thank my collaborators Chao Chen, Jose Painumkal and Moinul Hossain for the initial work on the system that led to the dissertation. Last but not least, I would like to thank my family and also my academic family members, such as Connor Scully-Allison, Alex Redei, Vinh Le, and Hannah Munõz, for their support.

This material is based upon work supported by the National Science Foundation under grant numbers IIA-1329469 and IIA-1301726.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

Contents

List of Tables

List of Figures

Chapter 1

Introduction and Background

1.1 Overview

Third party companies, such as Amazon and Microsoft, offer a stable cloud computing service (virtual machine usage service) based on a pay-as-you-go model [78]. This makes setting up large scale servers much easier than before. For example, Netflix leverages Amazon Web Services (AWS) to offer quality service for global users by renting virtual machines [109]. By using AWS virtual machine rental services, servers do not require technical hardware maintenance, such as power management and device management.

However, a company usually rents more resources (virtual machines or storage) from the cloud computing service companies than actual needed to guarantee the high-quality services. It is very hard to set up accurate rules to control server sizes with virtual machine usage service companies. There are two commonly-used control strategies for this problem:

- Event trigger: if something happens, server sizes will be changed. For example, when CPU usage is more than 70%, more servers will be added to the cluster. When network usage is less than 30%, some servers will be removed from the cluster.

- Schedule trigger: the number of servers are decided by time. For example, based on log files, there are more users between 10:00 am and 4:00 pm. Therefore,

more machines are rented during this time period.

These two control strategies, available from some platforms, such as Amazon Web Service (AWS) and Azure, are called "auto-scale". However, these two strategies are not perfect as explained by our previous research introduced in [95]. The main issues are:

- When the servers are auto-scaled, it is hard to control the budget. The service may scale up the servers and go over the budget.
- The auto-scale service can shut down some servers when the CPU utilization or network utilization are below a threshold. However, these "events" sometimes cannot truly represent that extra servers are not needed. It is possible that they are just busy with some low-CPU or low-network jobs.

Thus, the best way should be for the servers to control themselves because no one knows better than the servers about what is going on inside of the machines. Therefore, the method proposed in this paper shuts down the servers when they finish their assigned jobs.

In this dissertation, we propose a framework to change server size based on budget and user feedback. The framework has five components that provision resources based on rules set up by system administrators. The main advantages of this framework are:

- It can provide stable services. The average waiting time and queue length do not change massively.
- It can control the budget within the plan.
- It is able to show users' opinions about the current system and suggest to system administrators how to change rules to increase user satisfaction.

To prove the framework concept, three services are introduced using the framework: Service 1 *large datasets visualization and interaction* (Chapter 4.1), Service 2

model accuracy improvement (Chapter 4.2), and Service 3 *nitrate prediction* (Chapter 4.3). Service 1 visualizes large datasets in the backend in parallel and presents visualization results in the frontend. This is different from traditional workflows, which transfer and visualize data in the frontend. Our proposed method reduces network load and is stable. Service 2 can improve a hydrologic model accuracy based on the correlation between the model inputs and errors. It also utilizes the "window" strategy to process non-stationary data. Service 3 can predict nitrate in water with a prediction model developed by us. The model is tuned with genetic algorithms and is more accurate compared to other prevelant models.

The services and the framework are integrated as Figure 3.1 shows. It is simple to extend the framework with other services. A system administrator only needs to finish some configuration files and prepare a service Docker image.

The rest of this section introduces: service provisioning definitions; what are elastic servers; what has been done about the elastic servers; and some problems of current prevalent methodologies.

1.2 Service Provisioning

In our opinion, the size of a server cluster should be updated based on needs. If there are more nodes rented than the actual needs, extra money will be paid to rent or maintain the servers. If there are fewer machines than the actual needs, the users may feel latency or they need to wait in a queue to use the servers.

Many companies, such as Netflix [109] and Expedia [110], choose to use "cloud provisioning services" of third party companies, which are defined as: Cloud infrastructures (for example Microsoft Azure and Amazon Web service) that expose their capabilities as a network of virtualized IT resources [16]. Cloud provisioning is very prevalent because the advantages are clear and attractive:

1. Security: Most companies have different security models for different purposes. For example, Azure [6] does very well as "Infrastructure as a Service. The

model does not only protect the platform, it also secures the end user. The multiple-level security models are very necessary for sensitive projects, such as financial websites and user information databases.

2. Easy to Scale: Amazon Web Service (AWS) and Azure offer intuitive graphical user interfaces (GUIs) to scale servers up and down. It is very easy to learn and understand. The user may take around one hour to read tutorials or watch Youtube videos to do basic server scale operations.

3. Low Maintenance: Because all the rented virtual machines are set up in the third-party company physical machines, the maintainer does not need to worry about the machine power supply, machine security (someone may steal the machine), or machine damage problems. These are taken care of by these third party companies.

However, nothing is perfect. There are also some disadvantages with cloud provisioning:

1. Accessibility: When people rent servers, it is hard to control the rented server physically. If there is something wrong, in most cases the user can only shut down the machine and spin up a new one.

2. Privacy: It is not possible to keep absolute private information if you rent servers from a company. The company staff can access the servers if they have high-level authority.

3. Downtime: Azure provides a 99.95% service level agreement (SLA), which means there are only 4.38 downtime hours of the whole year [6]. This is better than most other companies. However, it is still not perfect. Downtime issue can causes server failures.

We believe it is worthy to use third party companies' cloud provisioning service and local machines for storing some sensitive information or if we want to access the

physical machine. Therefore, the framework proposed in this dissertation uses both local and rented servers (hybrid servers). The hybrid servers can change size based on the user needs and job execution events.

1.3 Elastic Server

Before we discuss further details about how to set up the hybrid servers, it is important to understand the definition of elasticity servers. There are many definitions of elasticity. Here are three of them:

1. Capabilities can be elastically provisioned and released, in some cases, automatically to scale rapidly, outward and inward, commensurate with demand [84].

2. Provider can dynamically assign the amount of memory, CPU, and disk space to a specific job and therefore, their performance and capabilities can vary based on the set up [1].

3. Elasticity is the capacity at runtime by adding and removing resources without service interruption to handle the workload variation [97].

There are some necessary characteristics based on these three definitions. First, the server cluster should be able to change its own size based on demand. Second, the size changing should be based on some events, such as hardware use. Third, during the size changing phase, the service offered by the server should not be stopped. In this dissertation, we propose ideas about how to build an elastic server based on these ideas.

There are two basic ways to scale up or down servers:

1. "Scale up" is also called vertical scaling. The basic idea is to add more resources into the server. For example, adding more memory and buying a more capable CPU belong to this category. The user can increase the service tier to scale up the service.

2. 'Scale down" is also called horizontal scaling. The basic idea is to increase the number machines in the database system. Then each individual machine needs to take care of less data.

Most third party companies support both methods and have a different price for different control strategies. In this dissertation, we only focus on how to use horizontal scaling to fulfill the actual demand from users. Specifically, we propose an innovative server-usage optimization approach to facilitate on-demand provisioning of computing resources to ensure reduced waiting time for jobs consistently over a predefined period of time within the allocated budget constraints.

1.4 Dissertation Structure

This thesis, in its remaining chapters, is organized as follows: Chapter 2 presents an overview of the related work; Chapter 3 describes in detail our proposed framework for optimizing elastic servers; Chapter 4 provides comprehensive descriptions of three data processing services that rely on our proposed solution; and Chapter 5 contains our conclusions and outlines possible future works.

Chapter 2

Related Work

There are numerous studies conducted in the field of dynamic provisioning of computing resources in a cloud environment. Some of the successful works are briefly discussed in this section.

Rodrigo *et al.* [16] proposed an adaptive provisioning of computing resources based on workload information and analytical performance to offer end users the guaranteed Quality of Services (QoS). The QoS targets were application specific and were based on requests, service time, the rejection rate of requests and utilization of available resources. The proposed model could estimate the number of VM instances to be allocated for each application by analyzing the observed system performance and the predicted load information. The efficiency of the proposed provisioning approach was tested using application-specific workloads, and the model could dynamically provision resources to meet the predefined QoS targets by analyzing the variations in the workload intensity. However, the approach offers no control over the expenses as it does not consider budget constraints and update control strategies while provisioning resources to ensure guaranteed QoS.

Qian Zhu *et al.* [140] proposed a dynamic resource provisioning algorithm based on feedback control and budget constraints to allocate computational resources. The goal of the study was to maximize the application QoS by meeting both time and budget constraints. The CPU cycles and memory were dynamically provisioned between multiple virtual machines inside a cluster to meet the application QoS targets. The proposed approach worked better than the static scheduling methods and con-

serving strategies on resource provisioning. The flaw with this approach was that it requires the reconfiguration of computing resources within the machine instances, which is not recommended in the current cloud environment where resources could be efficiently managed by the addition and removal of virtual machines from the cloud host providers. Moreover, dynamic allocation of resources based on CPU cycle and memory usage could become inaccurate more often, as the parameters cannot truly indicate the need for more resources. There are chances that the virtual machine is just busy with some low-CPU or low network jobs.

Similar to "dynamic provisioning", the concept of elastic servers has been introduced recently. Some third-party web service companies such as Amazon provide this service. Amazon EC2 (Elastic Compute Cloud), is one of the most commonly used elastic services, updated in December 2016 to support auto-scaling [108]. Previously, the rented cluster was removed on termination. This means that when a machine is removed in the cluster, everything will be removed and the project manager will need to manually scale up and scale down the number of servers. Now, Amazon EC2 Auto Scaling enables the servers to scale up and scale down automatically. There are two main methods to achieve this:

- Based on events. For example, if the CPU utilization passes a certain threshold, Amazon will spin up EC2 instances to lower the CPU utilization and when the CPU utilization comes down, the EC2 instances will be shut down.

- Based on schedule. For example, when normally most users use a website during the day and consequently the website manager sets a rule based on time: from 7:00 am to 7:00 pm more servers should be rented.

The Amazon EC2 Auto Scaling is also very easy to use. A project manager needs to group instances into auto scale groups and set operation rules. However, this service is not perfect, especially when auto-scaling servers based only on events or schedules. The main issues are: (1) When the servers are auto-scaled, it is also hard to control the budget. The service may scale up the servers and go over the

budget; and (2) The auto-scale service can shut down some servers when the CPU utilization or network utilization are below a threshold. However, these events cannot truly represent that extra servers are not needed (e.g. some low-CPU or low-network jobs). Besides these two issues, it is very hard to answer the following two questions using current web services based our knowledge:

- If the budget for one hour is $100, the workload can be estimated based on the history data, and there are five local host machines. How powerful the server we can build? Amazon does offer some brief survey about how much is the budget and suggestion the user some plans (CPU and memory types). However, this cannot offer an approximate idea about the job waiting time and the queue length.

- How to adjust the server size based on the user opinions? It is a hard question because the user may not even know their actual demands sometimes. Even though the demand is clear, it is hard to decide how to set up rules based on the feedback. Thus, one appropriate way is that the servers to control themselves based on the job completion events. Based on this idea, the framework proposed in this dissertation shuts down servers based upon an evaluation performed when each job is completed.

To deploy services in different nodes, "containerization" techniques are used in this dissertation. "Containerization" descirbes operating-system-level virtualization and the node provided by Docker is called a Docker container [85]. Docker is one of the most prevalent containerization programs [25]. The main reasons why Docker is very popular are: 1)it is lightweight. Docker containers, running in the same host, share the same operating system kernel. The containers use less memory and boot faster than traditional virtual machines. Therefore, Docker containers are more effective; 2) Docker containers are designed based on open standards. Docker supports most operating systems and each container is similar to a single machine. Therefore,

it is easy to maintain security of each container by adding a layer of application protections [25].

Because of the huge workload, a project manager always uses more than one nodes (i.e. Docker containers) to implement a service in a server cluster. To organize these nodes together, an orchestration tool is needed. Three of the most prevalent orchestration tools are Docker Swarm, Kubernetes, and Mesos. They all use Docker containers instead of virtual machines or real machines and they can scale up/down applications and manage docker containers. Here are detailed introductions between these three popular orchestration tools:

Docker Swarm: This is provided by the Docker company. Because it uses standard Docker APIs, it is compatible with most docker tools. Docker Swarm performance won't be affected by a large number of servers. The Docker company has done experiments to prove the concept with 50,000 nodes [26]. It is very easy to mount volumes on Docker containers and customize a Docker network. If one of the node fails in the Docker Swarm, the orchestration tool can start a new container to replace the old one. Docker Swarm has a default scheduler and is compatible with other plugins such as Kubernetes or Mesos [26]. However, Docker Swarm is not perfect. For example, the scaling function is not automated. The users can specify the number of containers in different services. However, if they want to define different rules about scaling, they need to write their own code.

Kubernetes: This was started by Google and now is supported by Docker and other companies, such as Microsoft and IBM. Kubernetes does support automatic scaling. The users can specify resource (such as CPU and memory usage) limits. The tool can automatically spin up containers based on the resources requirements and without affecting other containers performance. It is very easy to do horizontal scaling with Kubernetes. The users can use a UI to change the number of Pods (Docker container groups). If something goes wrong, the tool can rollback containers to a previous version. Also, the users can keep private information when they deploy and update application configurations without rebuilding application images [66].

Apache Mesos: This was developed at University of California, Berkeley. It supports Docker containers and also other prevalent big data cluster frameworks, such as Apache Spark and Apache Hadoop. This orchestration tool uses a different method to connect individual containers compared to most other popular tools. It does not use a Docker overlay network or some third-party networking tools such as vSwitch [86]. Apache Mesos uses ZooKeeper to handle different containers' IPs, ensure container availabilities, and manage failures [141].

We implemented our own orchestration tool because we would like to control the server size based on job completion events and none of the existing tools or platforms have this feature based on our knowledge.

Section 3 introduces a framework to improve the communication between different nodes. The key of the framework is to use a fast and data-lossless compression/decompression algorithm. There are many mature and efficient CPU compression algorithms. Some of them are designed for image compressions, such as JPEG [125], some of them are designed for audio and video compression, such as MPEG [69], and some of them are for general use, such as LZ4 [22]. Some scientists tried to take advantage of GPU to increase the speed of CPU compression algorithms. For example, paper [21] tried to improve the Huffman compression algorithm using a GPU. GPU is short for Graphics Processing Unit. It is originally designed for computer graphics and image processing, and and have become very popular in high-performance computing today.

Most of existing compression and decompression algorithms are not suitable for the data transportation framework introduced in Section 3.7 because they are not fast enough. For example, LZ4 is around 14.56 gigabits/s [22], which is much slower than wide-band network speed. In other words, the algorithm will slow down the throughput of data transportation.

Chapter 3

Optimization of Elastic Servers

3.1 Overview

This chapter introduces how we design our elastic servers and manage resources for each service. The main challenge is how to provide a stable service with a limited budget. Parts of our previous work has been introduced in paper [52].

Figure 3.1 shows an overview of the whole framework. All the services share the same Queue Manager, Rule Manager, and Feedback Collector. Each service has their own Docker container cluster and each container works independently.

3.2 Queue Master

The framework places job requests in different queues. All the user requests are handled by a Docker container named "Queue Master" (see Figure 3.1). This container classifies the requests into different groups based on the required services, Then different requests go into different queues. For example, if a user wants to execute a PRMS model, the request belongs PRMS service. Therefore, it enters the PRMS queue.

Each queue uses a modified queuing method. The original $M/M/1/1/\infty/\infty$ can estimate job waiting time by using Equation 3.1 and queue length by using Equation 3.2. This queuing model represents the following situation, with a Poisson distribution, where jobs arrive at a rate of λ/hour and the server processes the jobs at a rate of μ/hour (typically, this is considered exponential). There is one server;

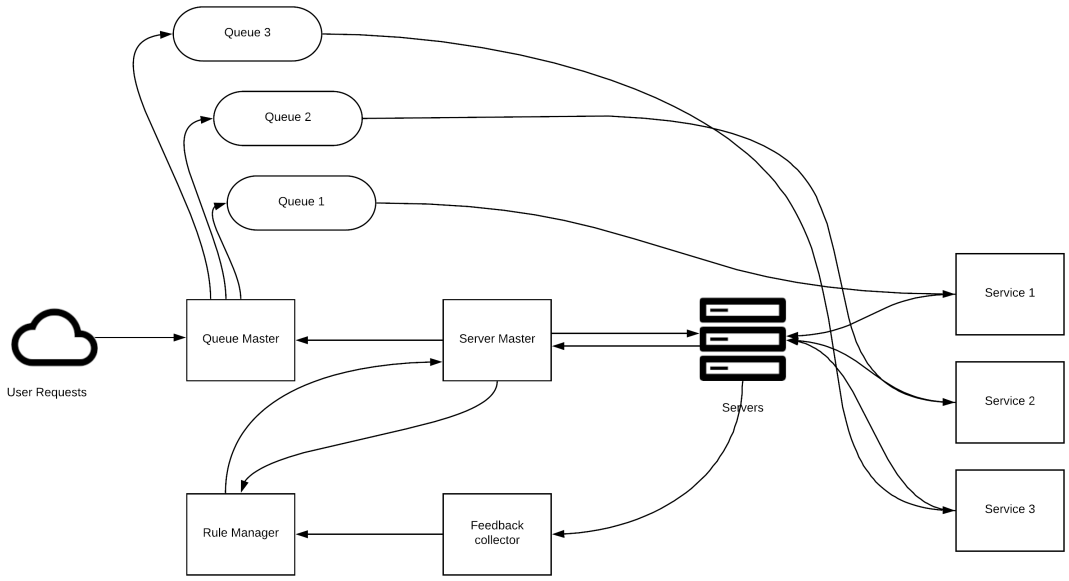

Figure 3.1: Architecture design.

the queue length can be infinite, and the population (maximum number of the jobs at the same time) can be infinite. This model also assumes that λ (job arrivals rate) is less than μ (server processes rate).

$$T = \frac{\lambda}{\mu^2 - \lambda * \mu} \tag{3.1}$$

$$L = \frac{\lambda^2}{\mu^2 - \lambda * \mu} \tag{3.2}$$

The $M/M/1/1/\infty/\infty$ queuing model was modified and applied in our prototype system. The idea is to develop a formula to estimate the average job waiting time and queue length in a hybrid server environment using the budget amount, budget period, cost of rented instances, and the average time for job execution. The modified queuing model processes jobs with owned servers and rented servers. For the given budget, B, and average job execution time with local servers, T_{own}, the owned servers can process a maximum of $(N_0 * T_b)/T_{own}$ jobs during the budget period T_b, where N_o denotes the number of owned servers.

If a rented instance costs $\$P$ for an hour of usage, then $B/(P * T_{rent})$ is the total number of jobs that can be processed with rented servers for the given budget amount B.

To achieve a stable service, the usage of rented servers are distributed uniformly during the budget time period T_b, which means the project manager should rent a server at every time interval, $T_{int} = (T_b * T_{rent} * P)/B$, if the owned servers are busy. At every T_{int} interval, if the owned servers are available (which means the job queue is empty), then the system needs not spin up a rental server for the incoming job. The system will also increment a counter variable, so that later, if a job comes in and the owned servers are busy, the system will rent a server immediately. This way, the proposed approach ensures that rented workers were utilized judiciously throughout the entire budget period. During the budget period, the hybrid server system could process a maximum of $(N_0 * T_b)/T_{own} + B/(P * T_{rent})$ jobs. The modified model is shown in Equation 3.3 and Equation 3.4. Owned machines do not consume budget in this model.

$$L = \frac{\lambda^2}{(\frac{N_0}{T_{own}} + \frac{B}{P+T_{rent}})^2 - \lambda * (\frac{N_0}{T_{own}} + \frac{B}{P+T_{rent}})} \tag{3.3}$$

$$T = \frac{\lambda}{(\frac{N_0}{T_{own}} + \frac{B}{P+T_{rent}})^2 - \lambda * (\frac{N_0}{T_{own}} + \frac{B}{P+T_{rent}})} \tag{3.4}$$

Results

The experimental study was conducted on four machines, each with Intel i7 CPU, 16 GB DDR4 RAM, and 256 GB SSD. Multiple threads were used to handle the continuous monitoring of queue length and job status, the creation of rented workers, and the simulation of the Poisson job arrival stream. The proposed approach was evaluated by simulating a Poisson job arrival stream on the job queue. Each job constitutes one PRMS model run with climate data from NRDC [51]. We have conducted experiments with different models and input data files, but because of the

space limitation only one of them is shown. To execute one job, the worker takes an average of 34 seconds (this simulates one-month of climate modeling). The initial execution time is obtained from experience and it is replaced with the average job execution time after the server starts working. Since the experiment was conducted with physical machines instead of machine instances from cloud providers, the cost of the host machine and the budget amount were simulated. For the experimental study, the system was allocated with a budget amount of $1.63 for a budget period of 20 minutes and the cost of the rented instance was considered to be $4.256/hour which is the current cost for a high end computing node on AWS. With the provided budget and price of instances, a maximum of 40 models could be processed with rented workers and the time interval T_{int} was estimated to be 30 seconds. i.e. during the budget period of 20 minutes, the system would use a rented worker to execute the job every 30 seconds, provided the owned worker is busy at that time.

Figure 3.2 shows the comparison of the waiting time between the FIFO approach and the proposed elastic server approach. In the proposed hybrid elastic-server approach, a rented container will be used only at regular time intervals. Whereas in the FIFO approach a new rented worker container is created and used to execute the job whenever the owned worker is busy. The drawback of the FIFO approach was that the rented workers may not last until the end of the budget period. Therefore, once the rented models are over, the incoming jobs have to wait longer in the queue causing a drastic increase in waiting time. By comparison, in the proposed approach, the rented jobs were used judiciously and hence the waiting time was maintained at a controlled level throughout the budget period.

Figure 3.3 shows the comparison of the queue length between the FIFO approach and the proposed approach. In the FIFO approach, all the rented workers were finished around the 13_{th} minute, which resulted in a steep increase of the queue length. In contrast, in the proposed approach the queue length was consistently low throughout the budget period.

The waiting time was calculated as the time taken by the worker to start the

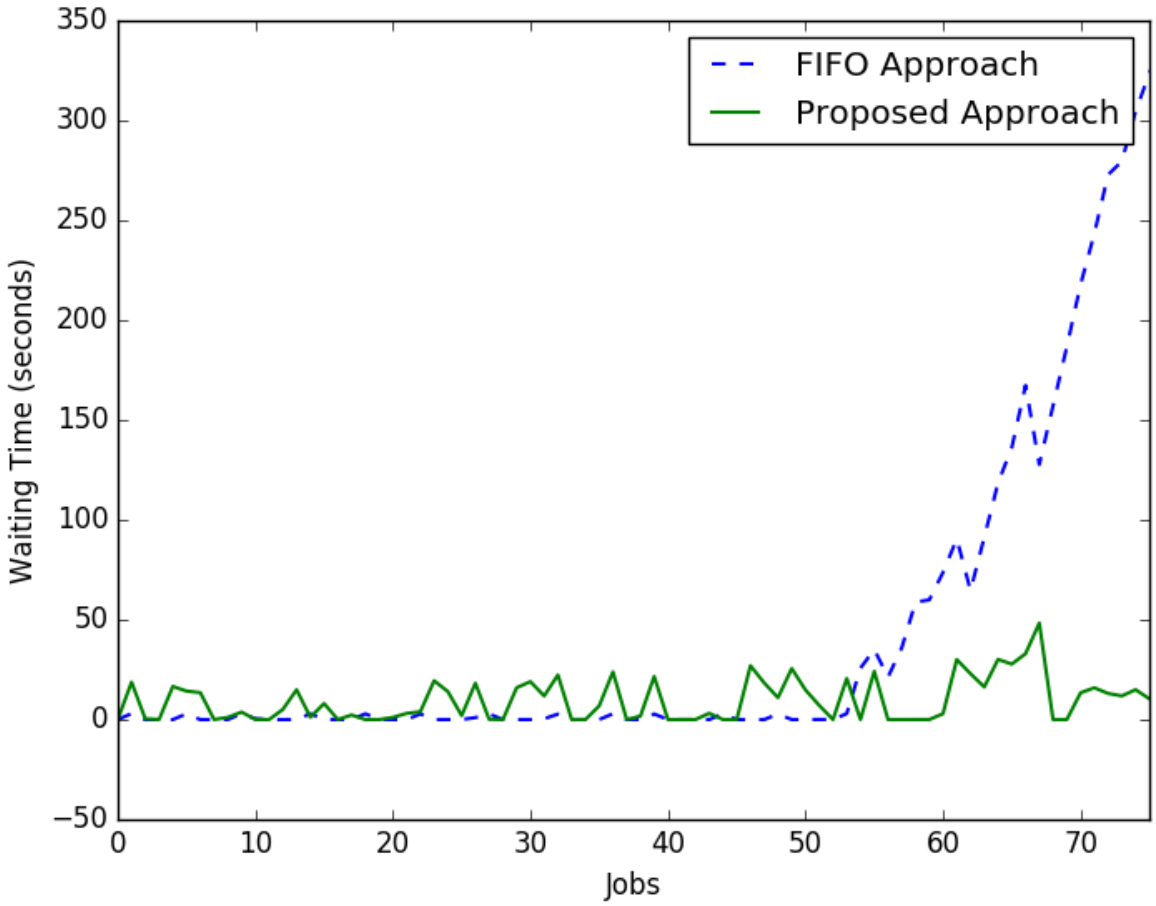

Figure 3.2: Comparison of waiting time of jobs. Waiting time does not change dramatically with the proposed framework.

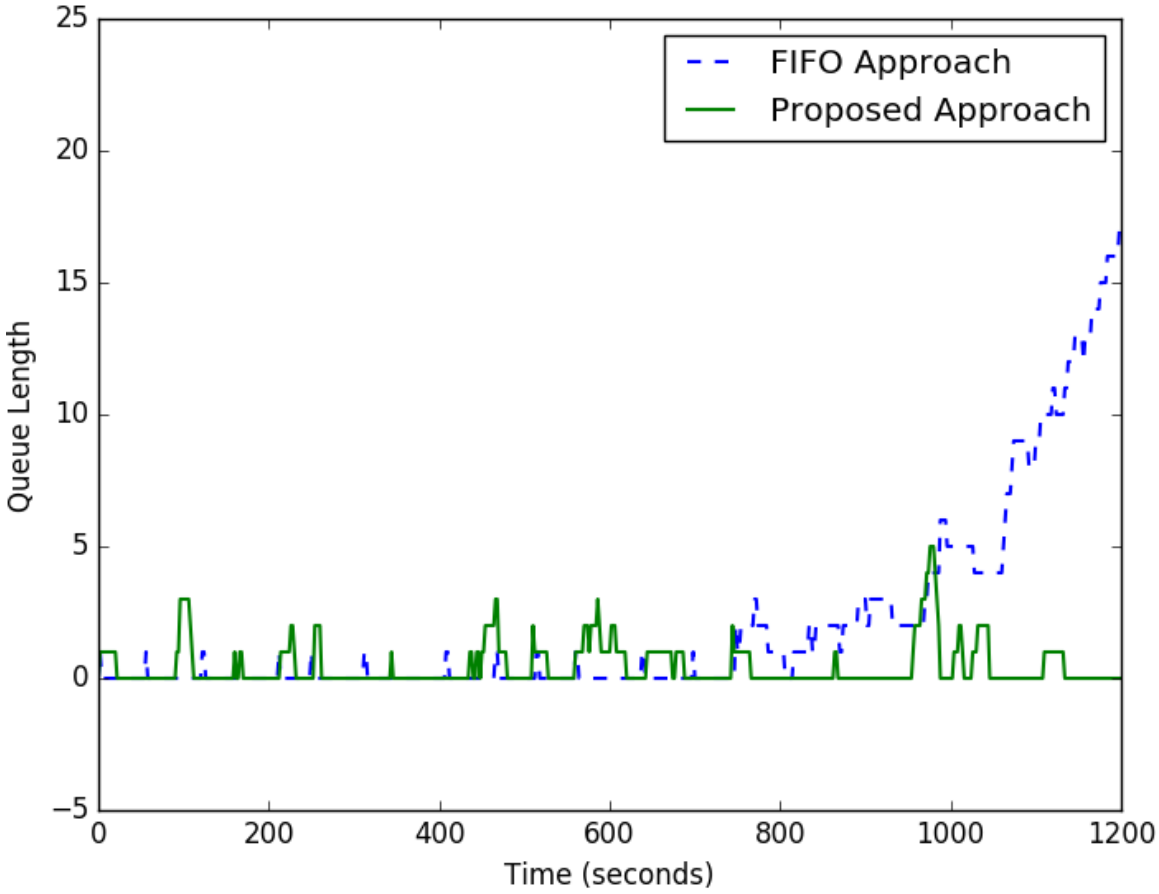

Figure 3.3: Comparison of number of jobs waiting in the queue. Queue length has consistently low throughout the budget period with the proposed framework.

job once the job is added to the queue. As seen in Figure 3.2, the waiting time of the jobs showed several fluctuations during the monitoring period, some of which impacting both the FIFO and the proposed methods. In the experiment, the rented worker containers were created from scratch using the base image. Depending on the resource utilization on the host machine, the container creation consumed several milliseconds to seconds. After the creation of a worker container, the worker took a few more seconds to establish a connection with the configured job queue and pick a job from it. We also see fluctuations in the proposed approach since the rented workers were created at regular time intervals T_{int}. This would also explain observed variations in the waiting time.

In the experiment, the FIFO approach ran out of rented workers in 747.1 seconds. During this period, it finished 54 jobs, with 14 of them being completed by the owned workers. Therefore, the utilization rate of the owned workers was 25.93%. In the same time period (747.1 seconds), the proposed system finished 40 jobs, 21 of them being completed by the owned workers. Thus, the utilization rate of the owned workers was 52.5%. It is evident from the results that the proposed method had a higher utilization rate of owned workers and it saved more rented workers for later use.

Based on Equation 3.3 and Equation 3.4, the expected queue length (number of job arrivals in the queue) was 1.46 and the real queue length was 0.622. Theoretically, each job needed to wait 0.39 minute and, in fact, each job waited 0.34 minute on average. This shows that the proposed method worked well in this job queue case. The time slicing between the different threads could also be a reason for fluctuations in the queue length and observed waiting time.

The time consumption for starting and stopping a rented instance varies with the work load and the cloud hosting service. Normally, the starting time of an instance ranges between 30 seconds to 6 minutes. Since our goal was to prove the applicability of the proposed approach, the experiment was conducted with comparatively shorter jobs and hence T_{int} value is also relatively small (less than a minute). However in real world scenarios, while dealing with high time consuming jobs, the estimated Tint

value would be sufficiently large enough to accommodate the varying VM start time.

3.3 Server Master

The server master Docker container is used to inspect all the host machines' health information. For example, the CPU, memory, and network usage percentages. It can automatically rent another host machine from a third party company, such as Amazon Web Service. The rented host machine event can be triggered based on the rules stored in Rule Manager container.

In each host machine, there are worker containers. These worker containers are arranged into different groups based on the service they provide. For example, Container 1, 4, 7, and 10 are in the same group as shown in Figure 3.4 because they provide the same service. All the workers in this group are created with the same Docker image and they know how to finish their jobs based on configuration files. For example, the users can run PRMS models in the prototype system. They need to upload three input files to start a model run. The workers in PRMS service group obtain the input files from the database based on the job ID and store the model output files into the database after the job is finished.

To connect different Docker containers in different host machines, we setup a key-value store node. If Docker containers are in the same host machine, they can ping each other directly without any further operations. However, a Docker container cannot ping another if they are in different machines. The key-value store node stores different host machine IPs, networks, and endpoints. After the node is setup, different host machines can view each other. Then, it is easy to create a Docker overlay network across different host machines.

For each group, there should be a task manager node and a feedback collector. The task manager node is used to create worker containers in different host machines and delete the worker container after the job is finished. We did not use any orchestration tool, such as Docker Swarm and Apache Mesos, because it is not possible to stop a container in a certain machine with these tools based on our knowledge. These

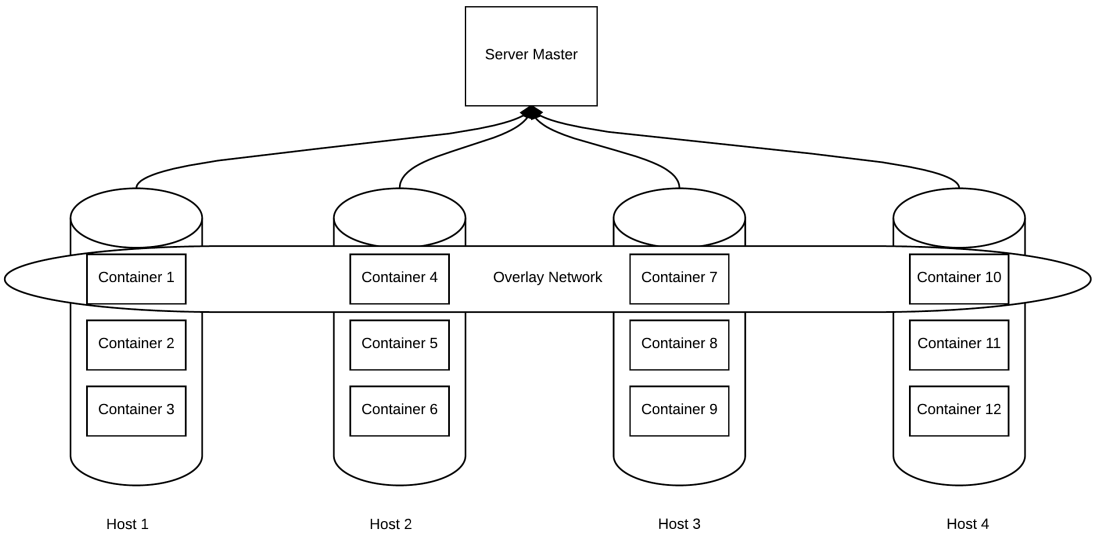

Figure 3.4: Server master.

tools can change the server size based on some events, such as CPU and memory usage percentages. However, this occurs at a high level (server as a whole). Based on our experiences, only the worker container itself knows the job status. It is not reasonable to shrink the server size based on CPU usage or time. Sometimes, the CPU usage is low and the container is busy. For example, file transportation jobs mainly use I/O bandwidth instead of CPU or memory. Accordingly, we can stop the container only when the container finishes the last line of the script. Therefore, each worker in our prototype system sends a termination request to the task manager after it finishes the job and then the task manager will stop the container. For a large cluster, it is possible that many containers report termination events to the server master container at the same time. It can cause problems if the network bandwidth is not big enough. There are two solutions for this problem:

- More than one server master containers are set up in the system to process the reports. Each master container is only in charge a group of containers and this method reduces the burden.

- Jittering APIs can be applied. Each container does not send the termination

information to the server master through the API directly after the container finish the job; the worker container waits a random time (e.g. 100 ms) and then send the information. This avoids network congestion.

The feedback collector collects the user opinions and can affect the rules stored in the rule manager. More details are introduced in Section 3.5 and Section 3.6.

3.4 Servers

Servers are physical machines in the system. These machines are set into different groups based on the needs of different services. For example, there are more PRMS model run requests than ISnowbal model run requests. Therefore, more server nodes are assigned in PRMS service group than in ISnowbal service group. The server master runs Docker configure files to download Docker images and setup services in different nodes. The server master may rent more nodes automatically based on the rules stored in the rule manager.

3.5 Feedback Collector

User feedback is very important for the project manager to set up reasonable rules. There are different methods to collect the user feedback. In the prototype system proposed in this dissertation, two methods are used to collect user's opinions: survey and comments analysis. Based on these two rules, the system can change the server size automatically or offer suggestions to the project manager.

3.5.1 Survey

The feedback collector sends a survey invitation to the user after he/she uses the service. Each question has a different weight and the options of each questions are worth different scores. A project manager can set up a threshold for each question. If the score passes the threshold, it means the project manager needs to do something. Here is an example: a project manager wants to know a user's opinion about the service's

performance. Therefore, the project manager puts two questions in the survey: 1) What's your opinion about the waiting time? (question weight 0.8) Options: A. Too long (4 points) B. Long (3 point) C. Not Sure (2 point) D. Short (1 point) E. Very short (0 points) 2) Do you want to pay more to have a faster service? (question weight 1.2) Options: A. Strongly Agree (4 points) B. Agree (3 point) C. Not Sure (2 point) D. Disagree (1 point) E. Strongly disagree (0 points). If a user chooses A for the first question and D for the second question, then it contributes 0.8*4+1.2*(1)=4.4 to the global feedback score. If the global feedback score passes the threshold it means the users believe the server is slow and they want to pay more for a faster service. Therefore, the project manager may need to inform the IT manager to rent more computational power from cloud computing service companies.

Each service has a feedback collector. The feedback collector contains a survey predefined by the project manager. Based on the feedback score, the feedback collector can affect rules stored in the rule manager. More details are introduced in Section 3.6.

3.5.2 Sentiment Analysis

The basic idea of comment analysis is to teach a machine to classify user feedback into different categories based on the words in the comments. For example, the categories can be: very bad, bad, intermediate, good, very good. Then, the feedback collector can change the server size based on the size of different categories. To achieve this goal, there are two steps: clean the data, and perform the machine learning.

Clean Data

It is essential to clean the data before training the model. This is because noise data can mislead or confuse computers. Assume the user feedback is ***The server performs terribly. I can't use this. It's too slow.*** Here are some steps used in this phase:

- Remove punctuation (period, comma etc.). The prototype system extracts the

words by specifying the separator (usually it is space). If the sentences have punctuation, these will be mixed in words with this strategy. For example, if the system extract words from "I love the service.", the words array will be ["I", "love", "the", "service."]. The last word is "service.", which is definitely not what we want. Then the feedback turns into ***the server performs terribly i can't use this it's too slow***.

- Lower case all the words. Our prototype system is case sensitive. Lower cases will not affect the meaning of the sentences for most cases and the machine will not treat the same word differently for the case problem. Then the feedback turns into ***the server performs terribly. i can't use this. it's too slow.***.

- Replace synonyms in the sentences. In fact, machine learning techniques can detect words with similar meanings. However, this requires enough data to train the model. This step reduces the burden of the machine learning step. In our prototype system, we have built a small synonym database and we plan to use a third party synonyms through RESTful APIs in the future. After this step the feedback turns into ***the server performs very bad i not use this it's too slow***.

- Remove unnecessary words. The main purpose of this step is to keep sentimental and negative (e.g. not, none, nothing) words. The main purpose of comment analysis is to analyze the user opinions. Therefore, only the sentimental and negative words are meaningful. We also have an unnecessary word database in the prototype system. After this step the feedback turns into ***performs very bad not use too slow***.

Machine Learning

After the feedback is cleaned, we use different machine learning techniques to create models and choose the best performed one for the future use.

Here is an example of how the system evaluates a machine learning method. One

of the machine learning techniques applied in our prototype system is the decision tree. The system randomly separates the history data into a training set (70%) and a testing set (30%). It creates the decision tree using the training set and tests the performance with the test data. The system repeats the process 30 times and gets the average error rates to evaluate the performance.

3.5.3 Other Possible Methods

In our opinion, users' feedback can also be implied by their actions. For example, if the user quits the queue before the request is handled by the server, it may be because the user is impatient and unhappy. If an event like this occurs, a brief question can be provided to the user to ask why he/she quits the queue. For example, if the user leaves the queue before the services are rendered, in the survey, a question can be asked if the user misclicks or was unhappy with the waiting time.

To simplify the survey and obtain user feedback in an effective way, we want to use check boxes (each one has a keyword, such as bad, good, slow) to collect user feedback. The users need to choose the best words to describe their opinions.

Also different users may have different opinions about the same thing (such as "bad service"). For example, if a job is finished in 10 seconds, for some users it is fast. However, others may believe this is very slow. To differentiate between user expectations, in the future we could create a profile for each users and leverage the profile to make a more informed decision from each user's feedback.

3.6 Rule Manager

Queue master and server master follow rules stored in the rule manager. "Rule" means the constraints to control a system. For example, a possible rule can be "at most, 3 nodes can be rented per hour". The rules are applied to these two masters through RESTful APIs. When the project managers want to add a new rule, they may also need to modify RESTful APIs in the queue master and server master. This is because the RESTful APIs may not include the functions required in the rules. For

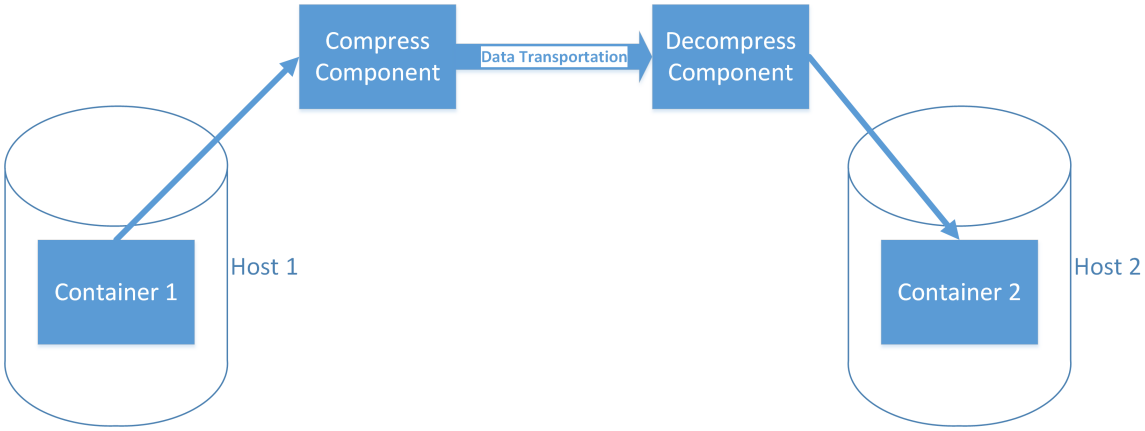

Figure 3.5: Data transportation optimization.

example, there is a rule requiring average job waiting time in Service 1 queue should be 10 seconds. However, the queue master does not have a RESTful API to change the average job waiting time. Then, the project managers should work on the API first.

3.7 Data Transportation Optimization

The communication between two workers from two different servers can be very time consuming because the two workers are connected via network for most cases. For example, container 1 and container 4 are connected through overlay network in Figure 3.4. This can be even worse if the two containers process a data intensive project. To solve this challenge, we used a framework shown in Figure 3.5. Before the data is tranported to container 2 from container 1, the data is compressed. When the data arrives container 2, it is decompressed. This idea is inspired by the work of Dr. Holub and his colleagues. In their paper [50], a method is provided to transmit HD, 2K, and 4K videos with a low-latency network.

The key idea is to choose a fast and data lossless compression and decompression algorithm. An improved version of GFC algorithm is used in the framework and here are some reasons that we prefer GFC over of other algorithms. First, original GFC is one of the fastest existing lossless compression algorithms. The algorithm is

75 gigabits/s [94]. It is gigabit, instead of gigabyte, because the core ideas of GFC algorithm are based on bitwise operations. The speed is much faster than most other compression algorithms. Second, the original GFC is designed for GPU directly. To contrast to GFC, most of the GPU algorithms are converted from CPU algorithms, which means some compromises have to be made and that has a negative impact on the algorithm performance most of the time. Third, the original GFC aims to compress large datasets, which is critical for both business and scientific uses. To further boost the performance, we improve the original GFC algorithm.

Some basic concepts about GPU, such as grid, block, warp, and thread can be found in [76] and Figure 3.6 displays a common GPU structure that presents the relations between threads, blocks, and grids. Different GPU video card structures may be different from each other, but they all share some common features: if users want their GPU algorithms to perform best, they have to use all the threads in a warp. If different threads in the same block need to communicate with each other, programmers can use shared memory; but if different threads in different blocks need to communicate with each other, programmers can use global memory.

3.7.1 Original GFC Algorithm

GFC is a lossless double-precision floating-point data compression algorithm. It is designed specifically for GPU computation. By using the GFC algorithm from [59], it replaces 64-bit floating-point values with 64-bit integers. Therefore, the GFC needs only integer operations, although it can compress a floating-point dataset.

An overview of warp, block and chunk assignments of GFC are displayed in Figure 3.7. The uncompressed data is separated into r chunks and each chunk contains 32 doubles. Each chunk is processed by one warp in the GPU. After all warps finish compressing the assigned chunk, GFC combines all the results together, which is compressed data. The reason that each chunk contains 32 doubles is that there are 32 threads in each warp for most of GPU video cards and it is most effective when a program uses all the threads in a warp. Figure 3.8 presents the details about the GFC

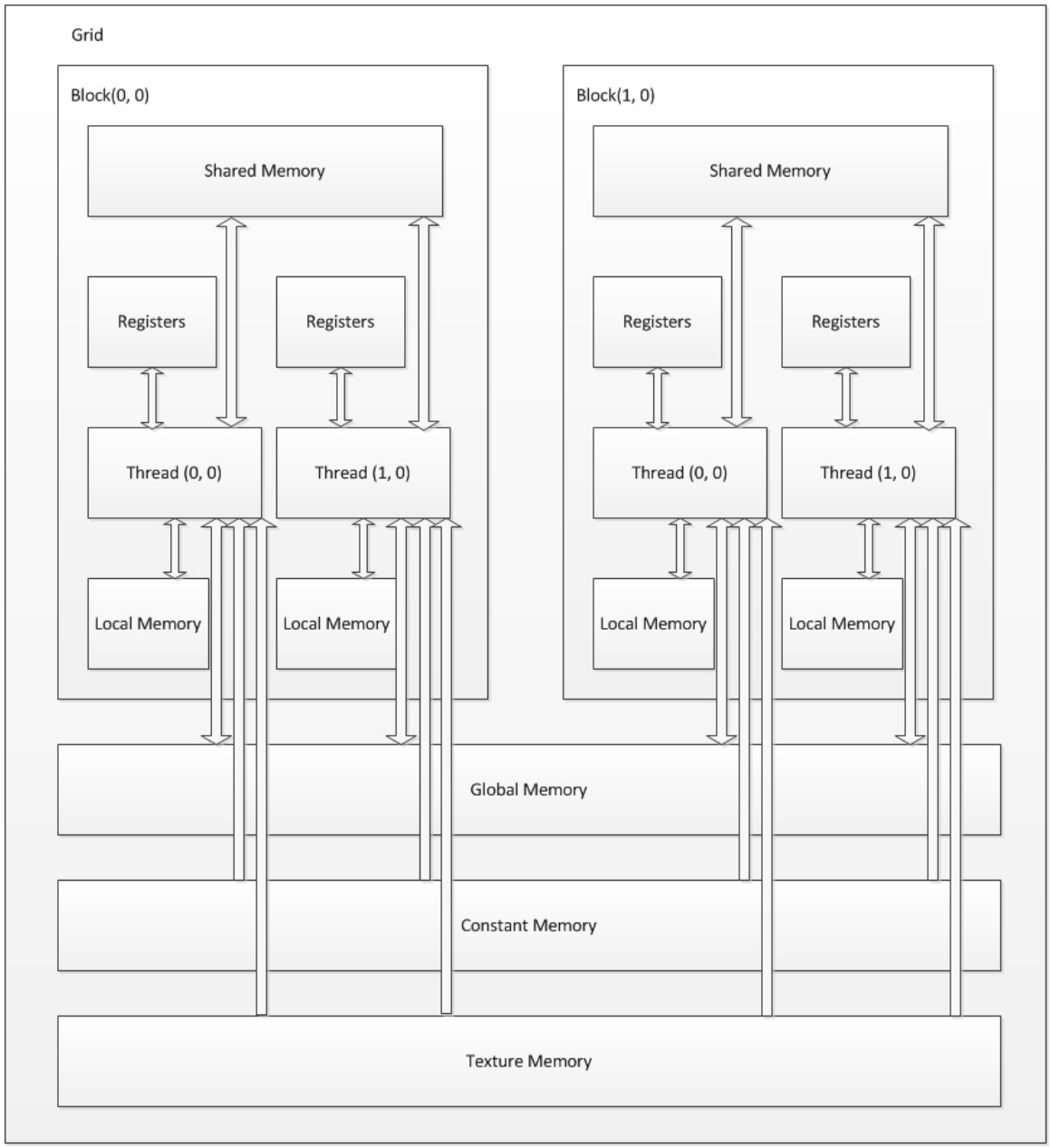

Figure 3.6: GPU structure.

compression algorithm. According to GFC, we need to subtract p, which is in the previous chunk, from i, which is in the current chunk, and $p = 32-(dim-i\%dim)$ [94]. Dim means dimension in this equation. If the subtraction is negative, we need to use operation "negate" to turn it positive. The key of GFC is the rectangle named residual in the bottom part of Figure 3.8. By counting the leading zeros of this part, removing these zeros, and adding the leading zeros metadata, GFC compresses the original datasets. The most significant theory behind GFC algorithm is that most scientific datasets interleave values from multiple dimensions [94]. For example, weather temperature will follow a pattern each year for most of the time, which means temperature scientific data can have many leading zeros by using GFC compression algorithm. Users need to find the interleave orders, get the maximum leading zeros and remove them to have the highest compression ratio.

It is possible that the compressed data is larger than the original data using GFC compression algorithm if an inappropriate interleave dimensionality is chosen. For example, all the eight bytes of residuals are non-zeros and it results in the output sub-chunk being 16 bytes larger than the original chunk, which is 6% larger than the original part [94]. Before the GFC compression is used, the data should be proprocessed and find out the suitable data interleave dimensionality to obtain the best performance.

O'Neil and Burtscher created GFC and published this algorithm on paper [94]. They avoided using long if-else statements and assigned datasets reasonably according to the structure of GPU to improve the performance of their algorithm. if-else statements can slow down a program, especially a GPU program. This is because of the structure of video cards. Each warp has 32 threads (for most video cards) and all these threads (in the same warp) must execute the same instruction in one cycle [76]. When these threads execute if-else statements, some threads may fulfill the if statement and execute that part of the code, and the remaining threads will stay idle, which means threads are not fully used. Therefore, GFC avoids using long if-else statements.

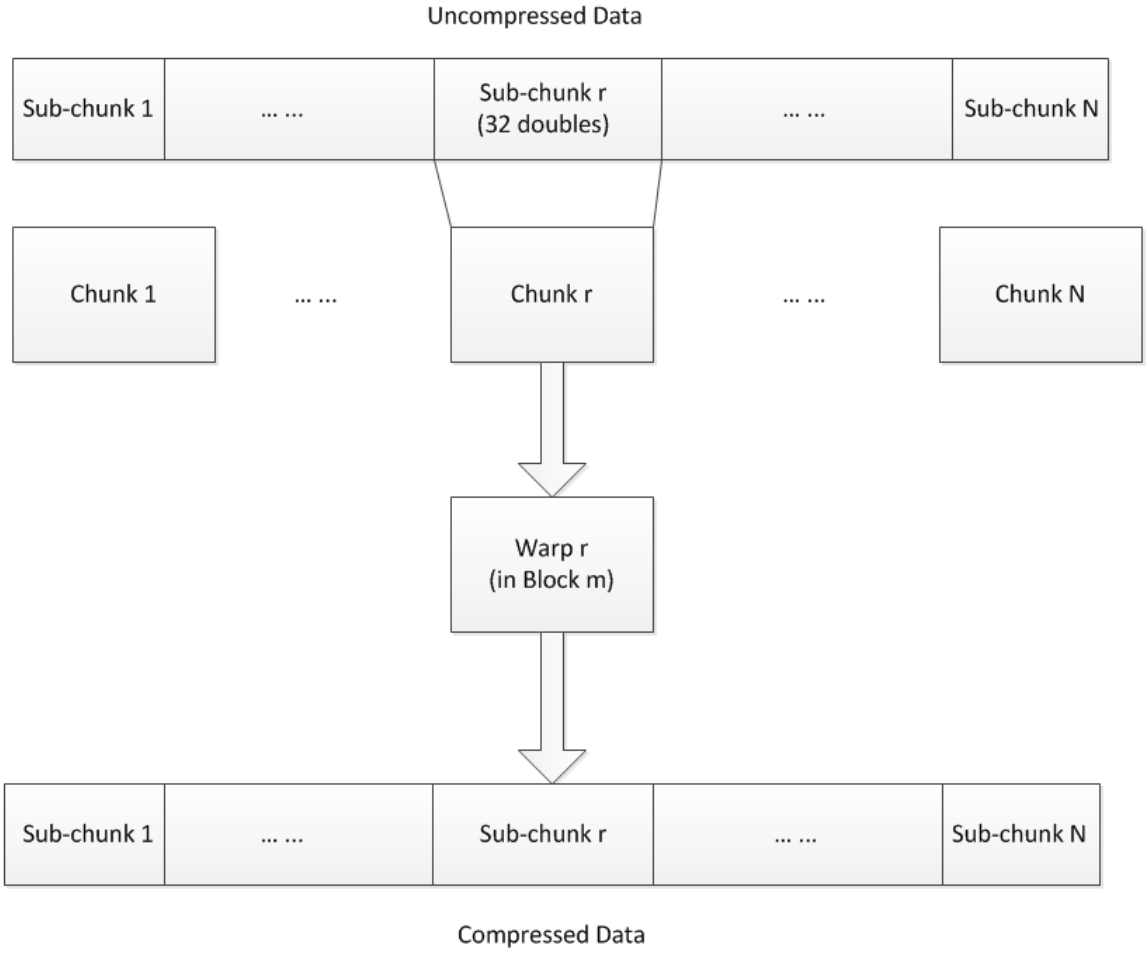

Figure 3.7: Overview of GFC algorithm warp, block, and chunk assignment. Each warp is assigned 32 doubles because there are usually 32 threads in each warp.

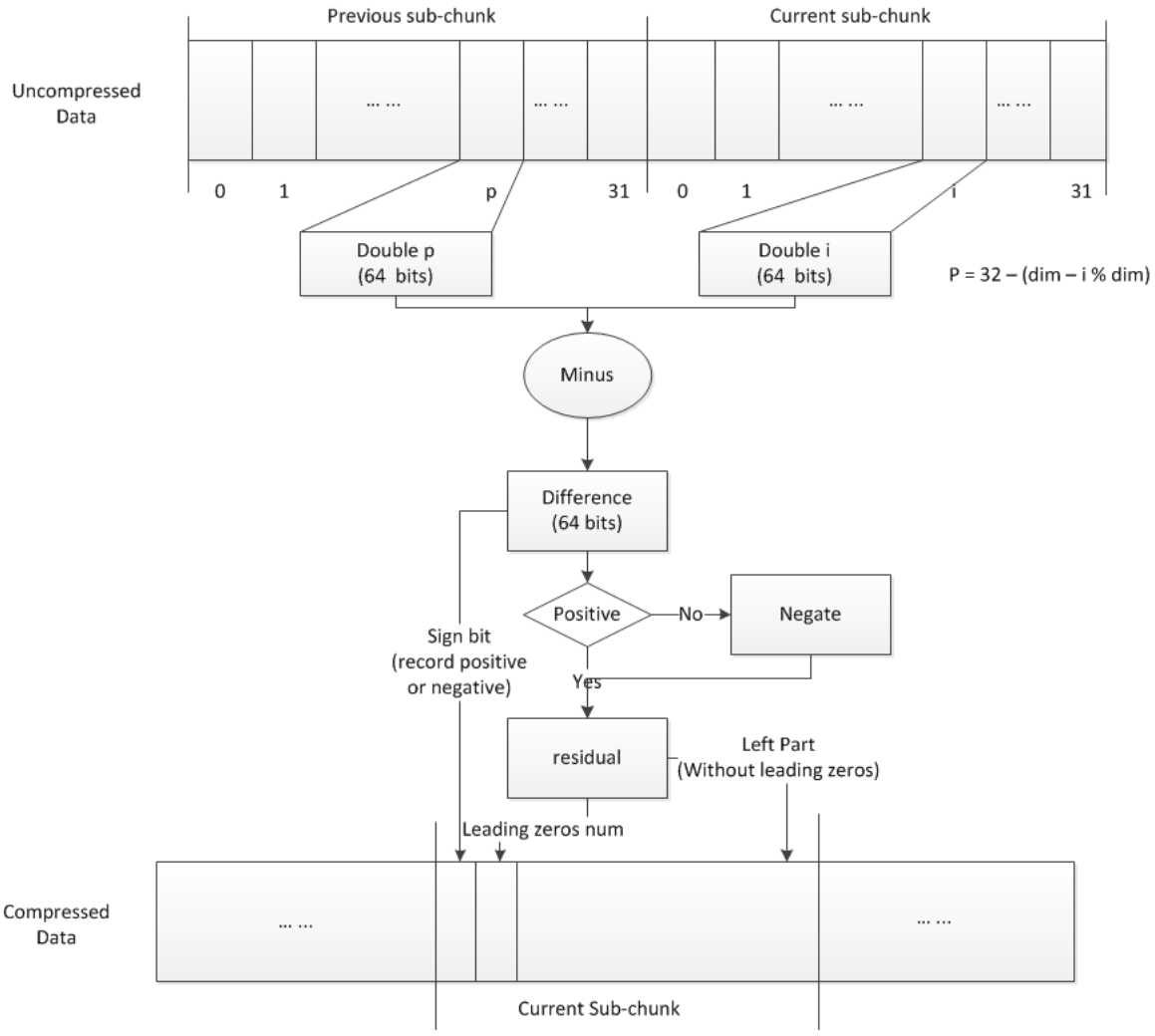

Figure 3.8: GFC compression algorithm. The original file is shrunk by removing the leading zeros.

```
if a < 3 then
  a = 7
else
  if a >= 3 then
    a = 5
  end if
end if
```

Figure 3.9: If-else statement example.

3.7.2 Improved GFC Algorithm

We tried to improve the performance of GFC algorithm with three methods: 1) using clzll to count the leading zeros; 2) removing if-else statements in the program; 3) using multi-GPUs.

Clzll

In the summary and conclusions part of [94], the authors mentioned that they wrote their own function to count the leading zeros because their video card was GTX-285 and it did not support clzll, which is used to count the number of consecutive leading zeros bits, starting at the most significant bit (bit 63) [91]. They believe GFC could be improved by using clzll to count the leading zeros to replace their code. We agree with their idea because professional programmers in Nvidia know how to take the advantage of their video cards. Therefore, it is not strange that their GPU functions are more suitable to the structure of video cards and more effective than our code. The results in Section 5 provide further verification of this idea.

If-Else-Removals

In our opinion, if-else statements can slow down programs, especially for GPU programs. This is because if-else statements will make some of the threads in a warp idle, when these threads within the same warp cannot fulfil the if-else statement. Here is an example presented in Figure 3.9:

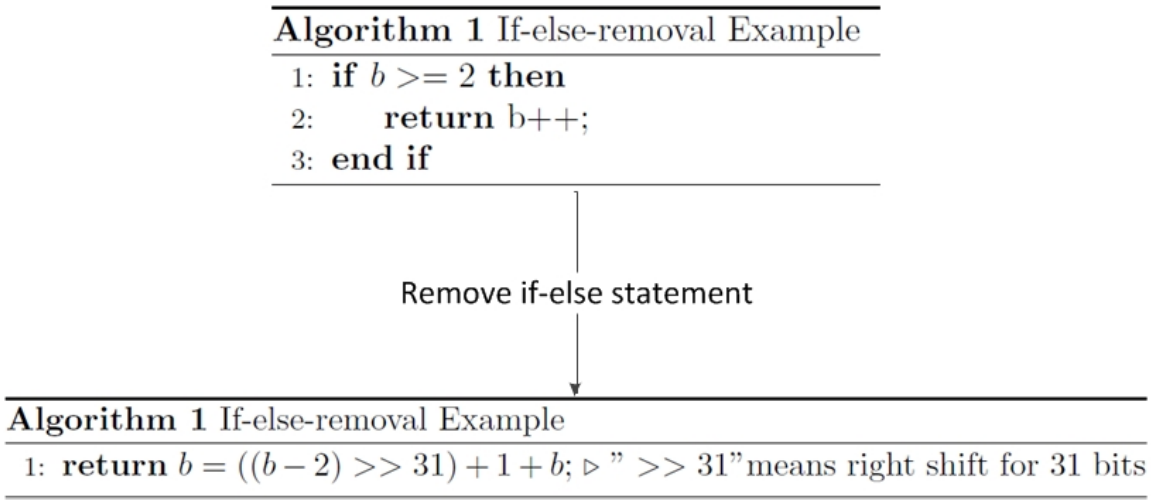

Figure 3.10: If else-removal example, less lines but more complex.

Each warp has 32 threads for most current video cards. Only the threads that fulfil the condition, $a \leqslant 3$, they will execute $a = 7$. Other threads will be idle until the whole warp goes through this if-statement.

There are some materials, such as [74], proving long if-else statements will also have a negative impact on the performance of normal programs. Therefore, we tried to remove if-else statements in GFC algorithm by using bitwise operations. Here is an example, as Figure 3.10 displays. $>> 31$ means a right shift for 31 bits. For most cases, signed integers have 32 bits and the left most bit is used for a sign (positive or negative). $(b2) >> 31$ is -1 when $b2$ is negative and it is 0 when $(b2) >> 31$ is positive. Therefore, the two statements are the same in Figure 3.10.

However, we found when if-else statements are short (for example, there is just one line of statement under if), the replacement of if-else statements with bitwise operations will slow down the program. We believe it may be because something undisclosed in the compiler to optimize the program. The authors of paper [94] also tried to avoid long if-else statements in their program, except one part in the decompress kernel. Therefore, we replaced that part with bitwise operations as Figure 3.11 shows.

But the method cannot guarantee better results all the time. Figure 3.12 dis-

Algorithm 2 If-else-removal in GFC Decompress

```
1: if (lane&1) == 0 then
2:     code = dbufd[off + (lane >> 1)];
3:     ibufs[iidex] = code;
4:     ibufs[iidex + 1] = code >> 4;
5: end if
```

Remove if-else statement

Algorithm 2 If-else-removal in GFC Decompress

```
1: code = dbufd[off + lane >> 1] * ((−1) * (lane&1) + 1) + code * (lane&1);
2: ibufs[iindex] = ((−1) * (lane&1) + 1) * code + ibufs[iindex] * (lane&1);
3: ibufs[iindex + 1] = ((−1) * (lane&1) + 1) * code >> 4 + ibufs[iindex + 1] *
   (lane&1);
```

Figure 3.11: If-else-removal in GFC decompress.

plays the delta time between the original algorithm and the improved algorithm for a dataset named obs_info. When the line is above zero, it means the improved algorithm is faster. Even if the improved algorithm is better, the improvement is not really obvious. Therefore, we do not apply this method in the final improved algorithm. In our opinion, the reasons that this method does not improve the performance are that each thread needs to spend more time than before because the code is more complex and the total processing time is worse, even if there are no idle threads in the wrap.

Multi-GPUs

After reading some GPU technique papers, we found that some authors try to improve the performance of an algorithm by parallelizing the algorithm and others try to enhance an algorithm by parallelizing tasks. For example, in paper [58], the author proposed to separate strings and assign a thread for each segment to increase the speed of Boyer-Moore algorithm. We also found there was a trend that scientists used multi-GPUs instead of a single GPU to improve their algorithms.

We found the taskcompression is parallelizable. Parallelizable means that we can separate the task into several parts and each part can be processed independently.

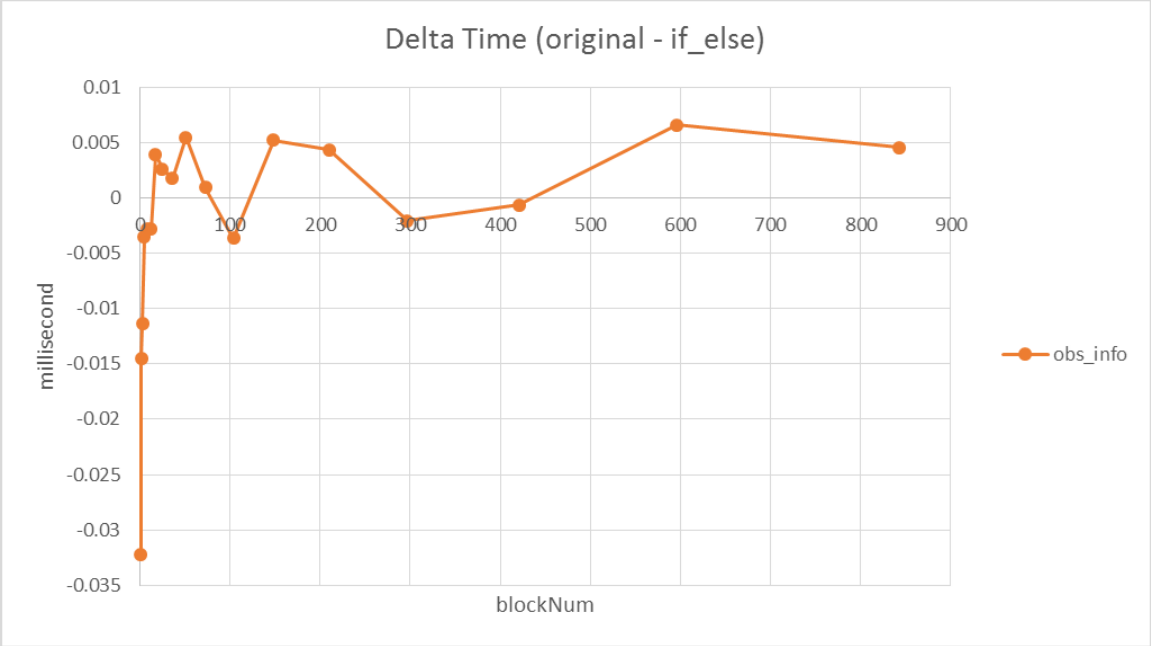

Figure 3.12: If-else-removal time delta.

GFC is a GPU algorithm and it uses both blocks and threads. Therefore, we need to assign a GPU for every segment to enhance the performance. So we tried to use multi-GPUs instead of single GPU and the basic idea is displayed in Figure 3.13. The uncompressed dataset is separated into N chunks, each chunk is processed by a GPU, and each GPU processes the assigned data with GFC algorithm. After all the GPUs finish their jobs, a CPU will combine the results together, which is the compressed data.

3.7.3 Experiments and Result Analysis

Our experiments were run on a Cubix machine, which has eight GeForce GTX 780 video cards, Intel(R) Xeon(R) CPU E5-2620 @ 2.00GHz, and PCI 3.0.

All the flowing experiment datasets are offered by Martin Burtscher, who is one of the authors of [14]. The datasets can be downloaded on the website [15]. From our experience on GPU programming, the best results of different problems need different numbers of blocks and threads. After experiments with four of these datasets, we found that we need to use all the threads in the chosen number of blocks to get

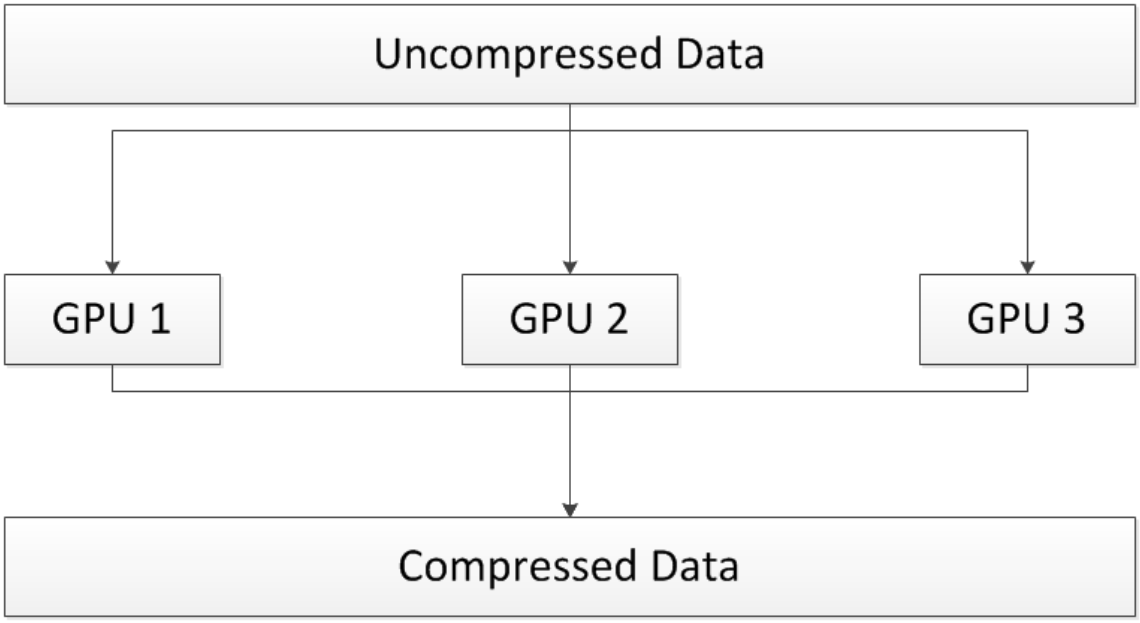

Figure 3.13: Multi-GPUs method.

the best results (throughput). Therefore, we only did experiments to find the best number of blocks for each dataset and used all the threads. All the experiments were ran 11 times and we chose the median value of these 11 results to be the final result. For example, in multi-GPUs part, we tested different numbers of blocks for a dataset named obs_info. We did the same experiment 11 times and finally found we should use 51 blocks and all the threads in these blocks to get the maximum throughput 1073.376 gigabits/s.

Because paper [94] mentioned that PCIe bus is too slow for GFC (compression speed is limited to 8GB/s [34]), O'Neil and Burtscher did not record the time of transferring data from CPU to GPU. Therefore, we did not do that for all the following experiments. We also compared decompressed files with original files to make sure that our methods do not change files.

Clzll

The first improvement is to use __clzll(), which is used to count the number of consecutive leading zeros bits, starting at the most significant bit (bit 63) of x [91]. The

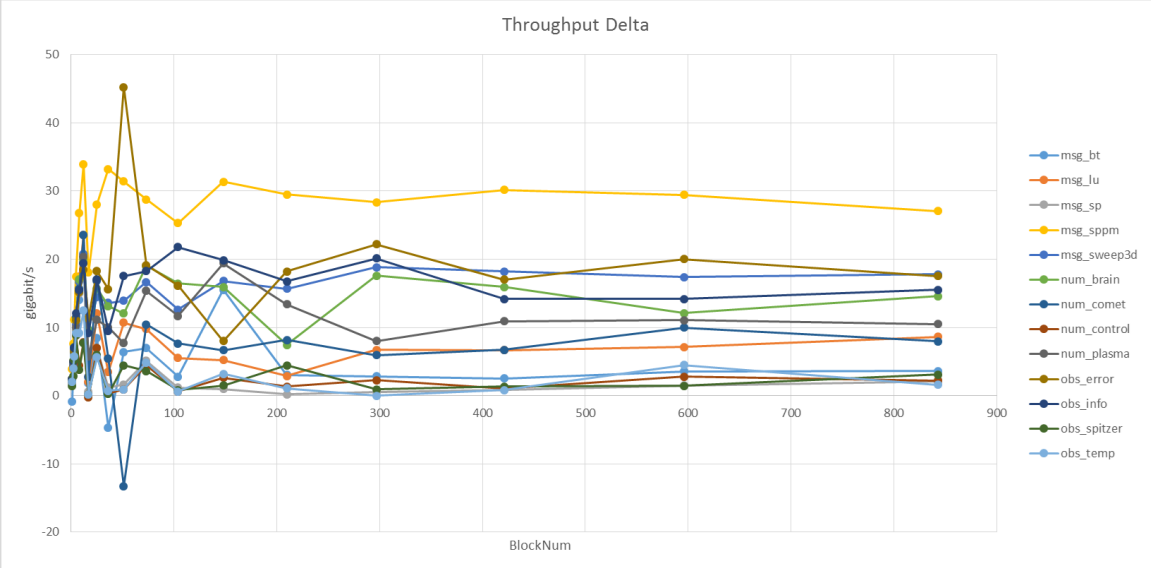

Figure 3.14: Clzll throughput delta. Most cases on the line charts are above zero. This means Clzll function can improve the performances.

results are presented in Figure 10.

In Figure 3.14, we subtracted original GFCs throughput from improved GFCs throughput. Most of the time, the deltas are above zero, which means the improved algorithms throughput are better. This validates the idea that is introduced in Section 3.7.2.

Multi-GPUs

We did the experiments with one, two, four, and eight GPUs to study the relation between the number of GPUs and the speedup. Time consumptions were recorded of each GPU and we used the maximum time consumption to be the final time consumption. For example, we used 8 GPUs and GPU1 spent T1, GPU2 spent T2 GPU8 spent T8. The final time consumption was Max(T1, T2, T8). We used the maximum time for the final time because we set up a synchronizing point, which resulted in GPUs waiting for others until all the GPUs finish their jobs. Table 1 displays the throughputs (gigabits/s) of a dataset named num_plasma. To save time, we did not do the experiment with block number from 1 to 1024. The step of BlockNum in Table 3.1 is int(sqrt(2)).

Table 3.1: Num_plasma Throughputs

Methods Names	**P-value**	**T-value**		
BlockNum	8-GPU	4-GPU	2-GPU	1-GPU
1	159.26	81.40	41.18	21.25
2	304.68	158.78	81.51	42
3	436.46	233.39	120.21	62.06
5	668.01	376.61	196.12	102.86
8	987.06	572.33	304.24	159.44
12	1233.31	804.09	438.55	233.19
17	1214.7	768.86	420.02	219.24
25	1219.77	715.74	386.17	202.43
36	1212.84	803.42	438.75	233.02
51	1268.60	815.37	465.57	250.08
73	1365.97	955.67	541.73	261.71
104	1381.89	876.36	481.61	258.48
148	1312.64	871.23	523.82	264.98
210	1266.23	860.8	500.14	274.39
297	1214.87	838.03	496.44	274.89
421	1170.78	818.49	480.72	266.15
596	1140.8	743.96	457.01	264.19
843	1079.34	715.57	439.54	253.56

Table 3.2: Maximum Throughput

Methods Names	**P-value**	**T-value**	
Name	Max Throughput (gigabits/s)	BlockNum	Speedup
8-GPU	1381.89	104	5.03
4-GPU	955.67	73	3.48
2-GPU	541.73	73	1.97
1-GPU	274.89	297	1

Table 3.2 presents the maximum throughput of different number of GPUs. From this table, we can tell that the speedup is better with more GPUs. However, the relationship between the speedup and the GPU number is not linear. For example, 8-GPU speedup does not equal eight times 1-GPU speedup. In our opinions, this is because of the more GPUs we have, the more segment file will be generated (our program will separate the original file into N parts and each GPU is in charge of a segment). Our program needs to combine all the segment files together to be the final compressed file in the last compression step, which is done by a CPU sequentially. This step will use more time if we have more segment files.

Figure 3.15 visualizes the relation between the throughput of each number of GPUs with a line chart. For each line in Figure 11, we found they went up first and then went down, which means that too many blocks will reduce the throughputs (gigabit/s) after a certain threshold. When the blocks number is small, N GPUs will increase the throughput almost N times. However, when the blocks number is increased, the speedup is less than N times. We think it may be because of the impact of blocks, as we just discussed. This negative impact will reduce the gap between each of the multi-GPUs results. Therefore, the final results are less than N times, when the blocks number is large.

Final Improved GFC Algorithm

In conclusion, we combined two methods: clzll and multi-GPUs together to improve the GFC alforithm. We performed experiments on datasets acquired from the website [15] and obtained a significant speedup result (the improved GFC algorithm over

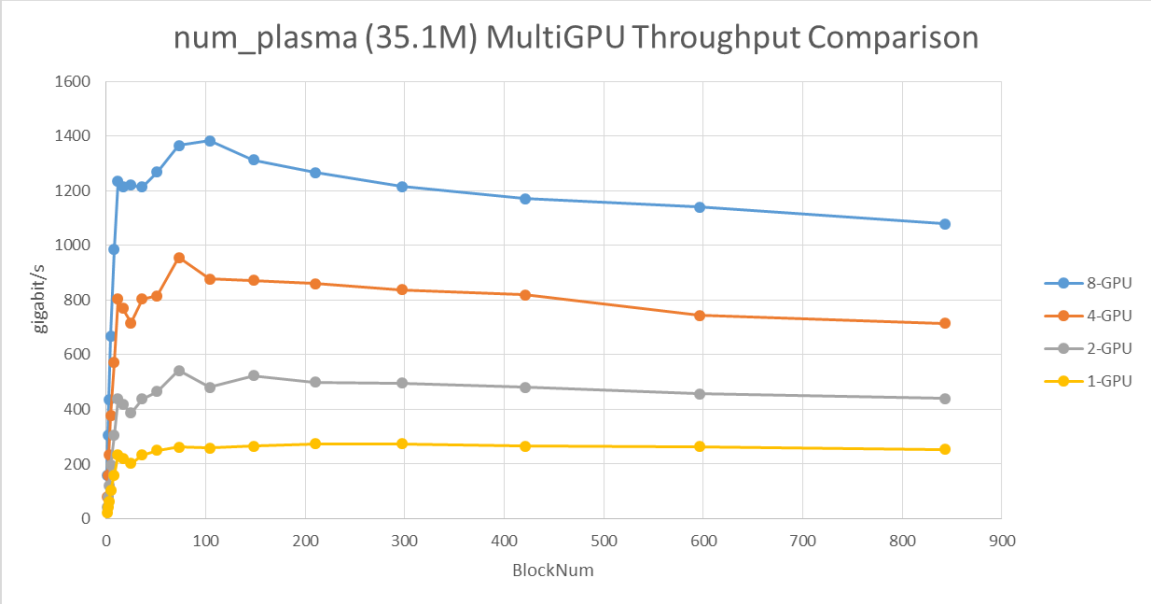

Figure 3.15: Multi-GPUs throughput of num_plasma.

the original GFC algorithm) as Figure 3.16 presents.

The maximum speedup of the improved GFC algorithm is 8.705 and the maximum throughput of the improved GFC algorithm is 2454.603 gigabits/s, which is much faster than original GFC throughput present in paper [69]. Of course, the good result is partially because we used better hardware than the original GFC paper.

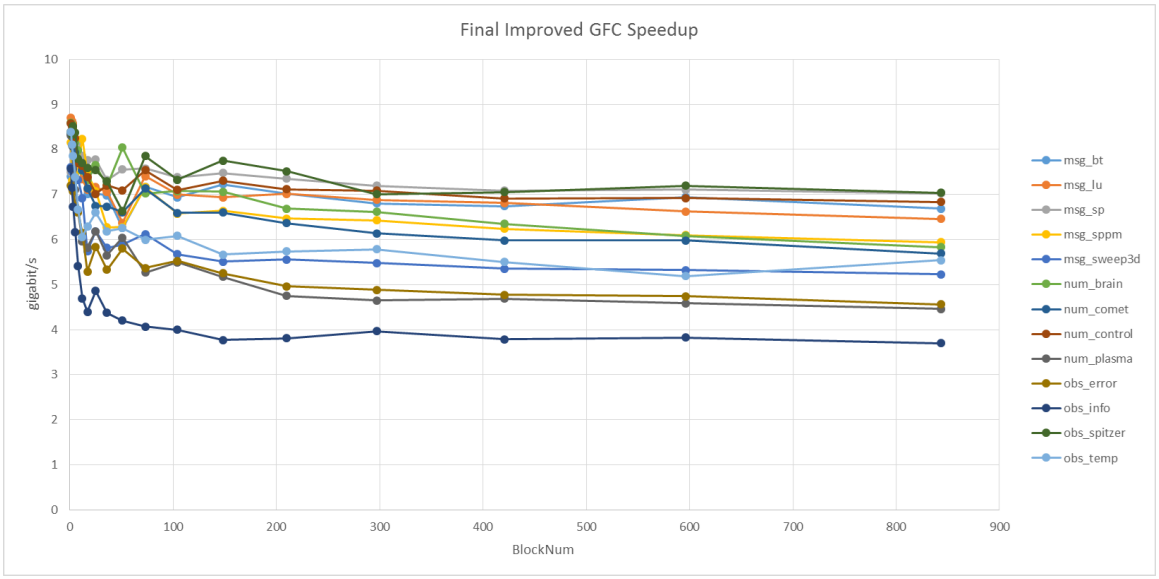

Figure 3.16: Speedup of improved GFC algorithm. The speedups of most cases are above 4 for all the datasets.

Chapter 4

Services

In this chapter, we introduced three services to prove the concept of the proposed framework. These three services are: Service 1: Large Datasets Visualization and Interaction, Service 2: Model Accuracy Enhancement, and Service 3: Nitrate Prediction Model, which are widely used in the interdisciplinary research.

4.1 Service 1: Large Datasets Visualization and Interaction

4.1.1 Overview

In this section, we introduce several algorithms to visualize large size datasets through a client/server architecture and allow for an interactive component with the results. There are many existing similar algorithms and libraries, but to our knowledge, most of them consume too long time to finish the job.

4.1.2 Motivation

Interaction and visualization are two significant methods for both business and science oriented individuals [132, 96, 135, 131] to find interesting features and trends buried within raw data. These two methods provided in this section can simplify complex theories and make it easier for people from different research areas to cooperate. Many prevalent web-based data interaction and visualization tools and libraries are not as effective as before because of big data. Most of the traditional client/server web

application visualization tools and libraries handle the visualization and interaction aspects on the client side. This workflow requires the server side to transfer data to the client side. If the data size is very big, such as the case with big data, the data transferring time is nigh unbearable. In this section, we propose a fast and new method for client/server web application to interact and visualize big data. The method visualizes data in the server side with multiple CPU cores and transfers result images to the client side. The client side collects users interaction information and the server side updates visualization results based on the interaction information. We tested the workflow with large volume datasets and it has shown a much faster speed than traditional workflows.

If users have large datasets, it is inconvenient and sometimes impossible to store all the data in one computer. The web-based client/server architecture is a good choice to interact and visualize large datasets. Instead of storing the data on the client side, users can utilize the storage capabilities of the server for larger datasets. Despite the abundance of traditional tools and libraries, they are not as effective as before because of the big data. As [71] points out, the most prevalent visualization tools and libraries can only be run on the client side, such as Google Chart, Open Flash Chart (OFC), Adobe Flex, and D3.js. And among those tools and libraries, only a few choice sections of them can even process large datasets. Generally, the performance tends to drop dramatically when there are more than 200,000 data points. For Adobe Flex, the visualization results occasionally stops responding occasionally when there are more than 50,000 records. To solve this problem, we designed a new workflow to move as much work as possible to the server side to improve the performance.

Our research aims to help both the business and science communities to interact and visualize their large volume data more quickly and effectively. In this section, we showcase our new workflow and a prototype system, as well as some experiments that compare the performance between our workflow and the traditional workflow systems. The results proved our workflow is better for big data interaction and visualization.

4.1.3 Proposed Visualization and Interaction Workflow

The web-based client/server architecture is used in the proposed workflow. We prefer to use this architecture because it does not require users to install anything for most cases. Users only need to open a browser and visit our website. Also, this architecture can always guarantee reliable performance, as long as it is designed reasonably (allocate most heavy tasks to servers). For example, some websites may offer images processing services. Users can upload batches of images to the websites and choose the process methods. After the websites finish the jobs, it sends an alert email to the users and then the users can download the result images from the website. Users can also process the images with their own computers. However, the procedure may be very slow if their computer is not good enough.

Most traditional web-based client/server applications use the workflow as Figure 4.1 presents [71]. When the client side sends a visualization request to the server side, the server side will send data to the client side. All interaction and visualization operations are done in the client side. If we use this workflow to interact and visualize big data, there will be two problems: 1) data transfer time can be unacceptable. For example, if users have 1TB data and 10MB/s downloading speed, it will take around a half an hour to transfer data; 2) most traditional tools and libraries load data into the client machines local memory to guarantee the performance. This does not work for big data. For example, if the chosen data is 1TB, this would require the client side machine to have 1TB of available memory spaces. This is impractical for most machines. We designed an improved workflow to move the interaction and visualization tasks to the server side that reduce the client side machines burden. We introduce more details about this workflow in New Workflow section.

The basic idea about the new workflow is that it separates all interaction & visualization tasks into subtasks and arranges as many subtasks as possible in the server side. The most remarkable difference between the new workflow and the traditional workflow is that it transfers visualization result images from the server side to the

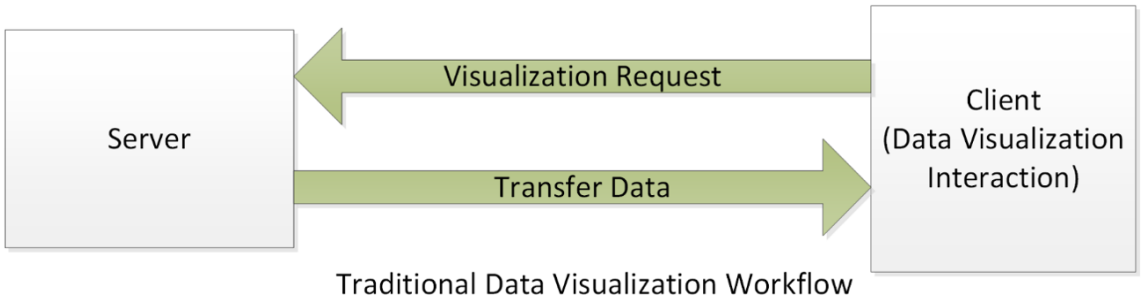

Figure 4.1: Traditional workflow.

client side instead of data. The server side updates the visualization result images based on the interaction information collected from the client side.

Figure 4.3 displays the workflow details. The server side is a distributed system containing many nodes. Each node can be a computer or a CPU-core. The large volume dataset is separated into small pieces. Each node oversees a slice of data and generates a result image of the assigned data. One of the nodes combines all the result images as a final result image. The image combination method is shown in Figure 4.2. This can be done in parallel. In Step 0, we have n images in total. Each step we use one thread to combine two images. After all the threads combine images, the next step will be started. The image combination is done in parallel too.

After the server side generates the final result image, it sends the final result image to the client side. The client side collects users interaction information (such as mouse coordinates, mouse button click, and mouse button release) and transfers the information back to the server side. The server side updates the result images based on the users interaction information.

The new workflow relies more on the server-side machines and that results in the system performing more steadily. It is hard to predict machines situated on the client side. If we use traditional workflow and push most of the data interaction and visualization jobs to the client side, we may have different problems to deal with when it comes to big data. For example, the client-side machine does not have advanced hardware and it performs more slowly to interact and visualize large datasets. In contrast to the traditional workflow, our proposed workflow uses distributed systems

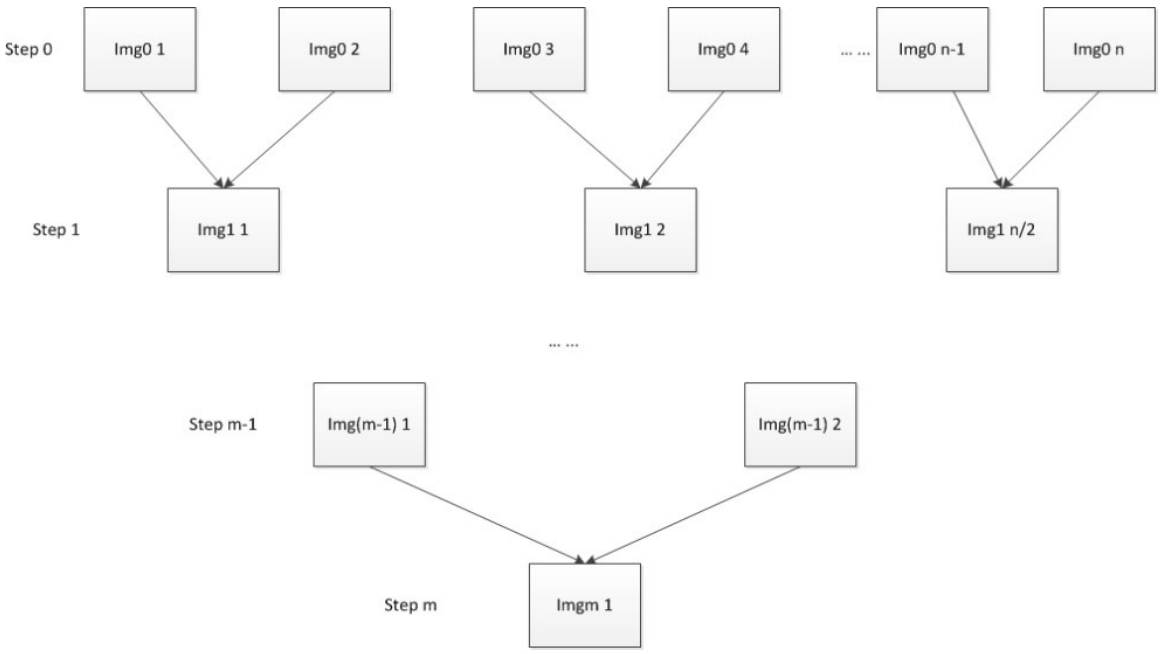

Figure 4.2: Image combination method.

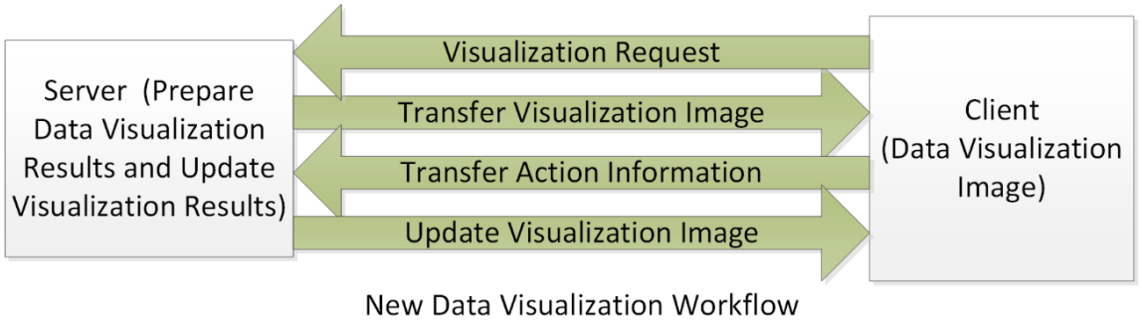

Figure 4.3: New workflow.

and process data in parallel. Therefore, the new workflow is always faster to deal with big data. Users do not need to have an advanced computer because the client-side machine only needs to display result images and collect the user interaction information.

Interaction can help a user to study further into visualization results. However, it is not very easy to interact with a large dataset visualization result. To our previous proposed method, we need to collect the user interaction information and interpret the information. For example, the most challenging part of enabling the users to interact with a line chart visualization result on the front end is to convert the users cursor information into appropriate values. Here are the four basic steps for this algorithm:

- Convert the cursor coordinates into percentages, which are named as CP. For example, when the user clicks on the middle part of the line chart, then CP should be 50% (from the images left side).

- Get the left and right image margin percentages of the whole image. The left margin is named LMP and the right margin is named RMP. For example, assume the left margin is 10% of the width of the whole image and the right margin is 15% of the width of the whole image. Then, in the following LMP is 10% and RMP is 15%.

- Obtain the corresponding x-axis values. The minimum value of the x-axis is named Min and the maximum value of the x-axis is named Max.

- Calculate x, the location of the cursor on x-axis, from the equation:

$$\frac{CP - LMP}{x - Min} = \frac{1 - LMP - RMP}{Max - Min} \tag{4.1}$$

  x is the users cursor x-axis value. (CP-LMP) means the percentage between the users mouse cursor and Min; (x-Min) means the x-axis value between the users cursor and Min; (1-LMP-RMP) means the x-axis range percentage; (Max-Min)

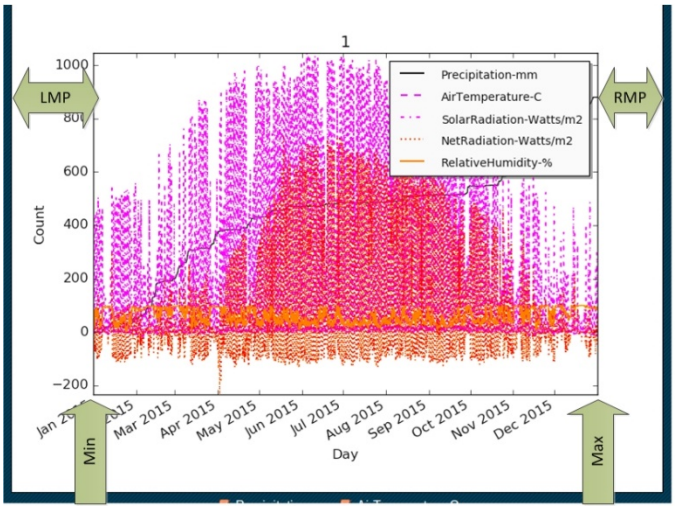

Figure 4.4: CSV file visualization example.

means the x-axis range. The ratio between percentage and x-axis values should be the same.

Figure 4.4 shows more details about these steps. More examples are introduced in Section 4.1.4.

Overall, visualize and interact datasets steps:

- Users choose a file from the server database or upload a file to the server by themselves.
- The server prepares the visualization result image and sends it to the client side.
- Users interact with the result image and the client computer collects the interaction information.
- The server side updates the visualization result image based on the interaction information.

4.1.4 Prototype Service

We have created a prototype service with eight CPU cores with the proposed framework introduced in section 3 and the server side can only visualize csv files with

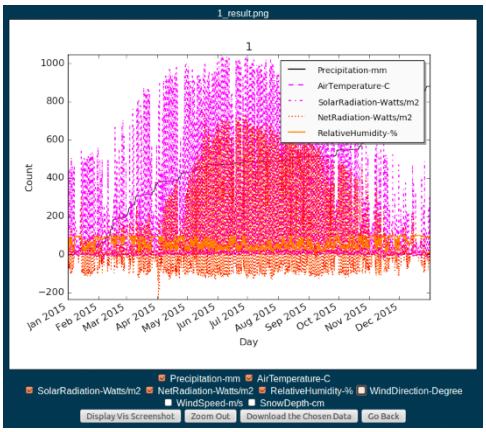

Figure 4.5: CSV file visualization example.

line charts for the current version. In this section, we introduce how to zoom in, zoom out, download the chosen part of data, and add/remove a csv column from the visualization result image.

Figure 4.5 is a screenshot of a csv file visualization. The line chart part image is created by merging eight images shown in Figure 4.6. Because each CPU core generates a result image, we have eight images in total. The procedure is that one of the CPU-cores separates the chosen csv file into eight parts with the help of a Python library named Pandas [100]. Then each CPU-core visualizes a sub csv file with Matplotlib [55]. The result images are named: img0, img1, img2, , img7. In the image merging stage, the system uses four CPU-cores as seen in Step One. CPU-core 0 is in charge of combining img0 and img1; CPU-core 1 is in charge of combining img2 and img3; ; CPU-core 3 is in charge of combining img6 and img7. Similarly, the final result image is generated after Step Four. This stage uses the method introduced in Figure 4.2.

There are some checkboxes below the line chart part in Figure 4.5. Each of the checkboxes represents a column in the csv file. If the user checks one checkbox, the system adds the corresponding csv column in the line chart, which includes adding a line in the line chart and a matching legend in the right top corner of the resulting image. If the user wants to remove the corresponding csv column, they need to uncheck the checkbox, which includes removing the line from the line chart and the

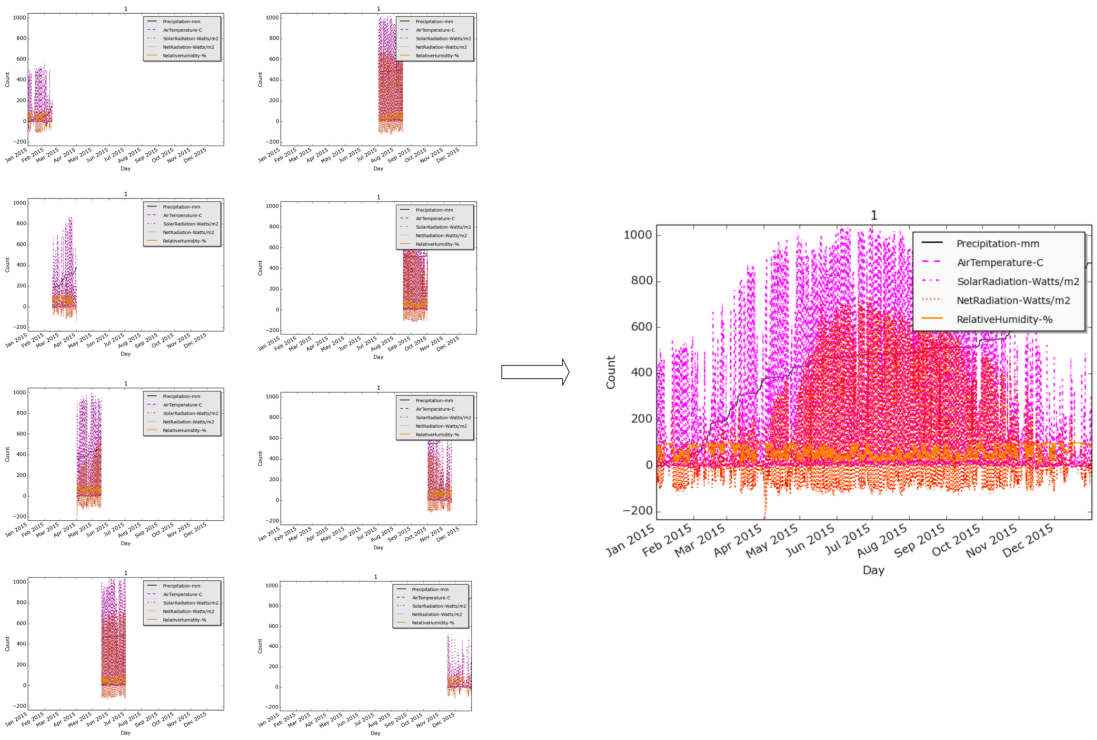

Figure 4.6: Visualization result images created by eight CPU cores.

matching legend in the right top corner of the final result image.

Users can zoom in the line chart by clicking the mouse left button, dragging along the line chart, and releasing the mouse left button. The system draws a rectangle on the line chart as Figure 4.7 shows. After the client side receives the updated visualization result image from the server side, it displays the updated line chart image, as Figure 4.8 displays. If users want to zoom out on the line chart, they need to click the Zoom out button. We implemented the Zoom in function by tracking the coordinates of users cursors and mouse click and mouse release events. Based on the information, the client side draws rectangles on the visualization result images and transfers the coordinates to the server side.

4.1.5 Results

This subsection showcases some experiments that were done to prove that our workflow is faster than the traditional workflow. Here are our server and client machines

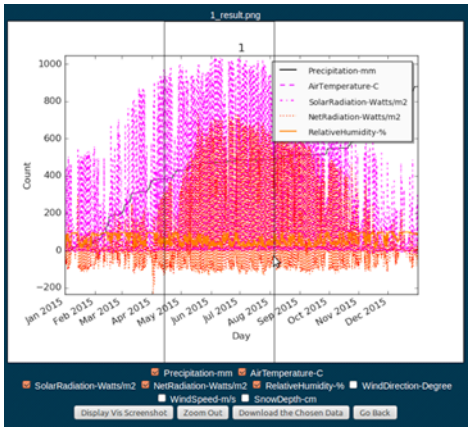

Figure 4.7: Users choose an area on the line chart.

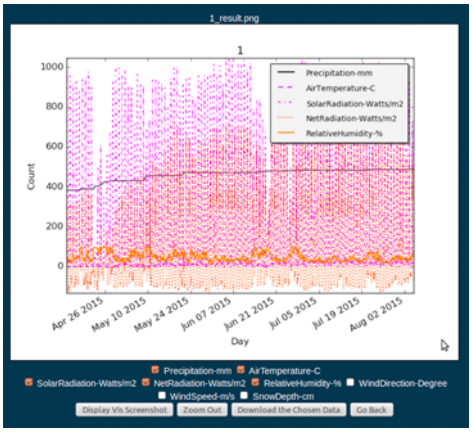

Figure 4.8: Zoomed in Line Chart.

hardware and operating system descriptions:

- 8 Intel (R) Core (TM) i7-4770 CPU @ 3.4GHz
- 12.0 GB DDR3 RAM
- Ubuntu 12.04

For our new workflow, we used Flask [39] to build the backend and Matplotlib [55] to visualize data. Flask is a powerful Python-based microframework and Matplotlib is an effective Python based 2D data visualization tool. For the traditional workflow, we used Flask [8] to build the backend and dygraph.js [33] to interact and visualize data in the frontend. Digraph.js is a JavaScript library that is easy to use and efficient. There are more details about how we built the traditional workflow system in [133]. All the data files used in this dissertation are generated by a scientific model named Isnobal [79].

For all of the following experiments, the server and the client are installed on the same machine. Therefore, the data transfer time is very short for both the new and the traditional workflows. In real life, data transfer time is one of the most crucial factors that affects the user experience. For most of the big data cases, the traditional workflow costs more time than our new workflow. This is because the visualization result image is usually smaller than the visualized data itself. For example, 100 MB data can be visualized with a 5 MB jpg file. The image size is decided by resolutions and image formats. Therefore, the new workflow is faster than the traditional workflow in data transfer.

Figure 4.9 compares the traditional workflow and our workflow data visualization time consumptions. When there are less than 100,000 records (floats), the traditional workflow uses less time than the new one. This is because data size is small and data transfer time is short for the traditional workflow. The new workflow needs to separate files into small pieces, visualize each of them, and then merge all the visualization result images. These steps are not effective for small data files. However,

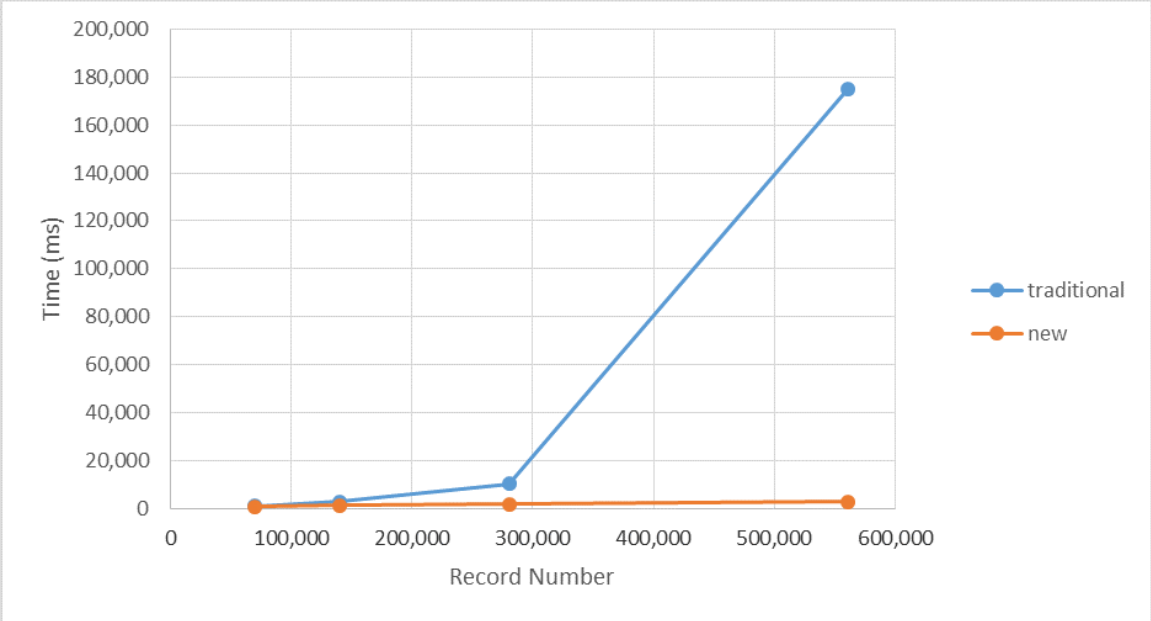

Figure 4.9: Traditional vs new workflow data visualization time consumption.

when data size grows, the traditional workflow turns slower than the new workflow. Especially when we have more than 290,000 records (floats), the traditional workflow performance drops dramatically. In fact, when we tried a large file with more than 10,000,000 records, the traditional workflow ran for more than one hour and popped up an error.

Figure 4.10 presents the new workflow time consumptions. It is almost linear which means the more data we have, the more time it takes. To improve the performance, we want to use Hadoop with more nodes.

We also tested our system with a different number of processes. Figure 4.11 shows that the time consumption goes down first and then goes up, which means when we used more processes, the performance of the system turns better first and then turns worse. The system performs best when we used 16 processes. This is because we used 8 * Intel (R) Core (TM) i7-4770 CPU @ 3.4GHz. The Intel core uses hyper-threading technique, which allows a computers operating system or hypervisor to access two logical processors for each physical core [83]. Therefore, the 8 Intel core equals 16 logical processors.

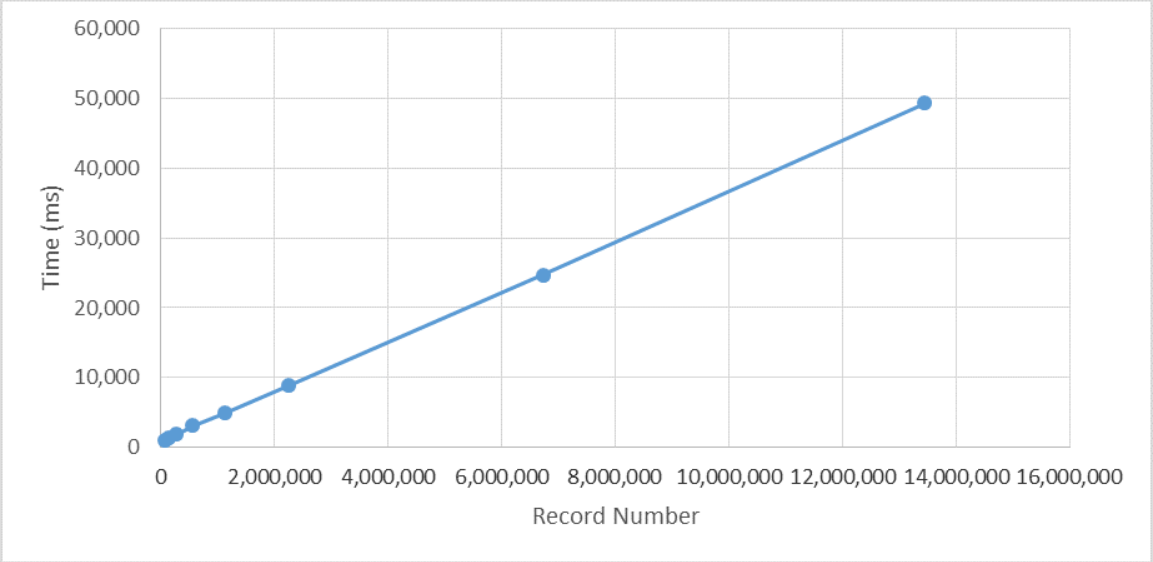

Figure 4.10: New workflow time consumptions.

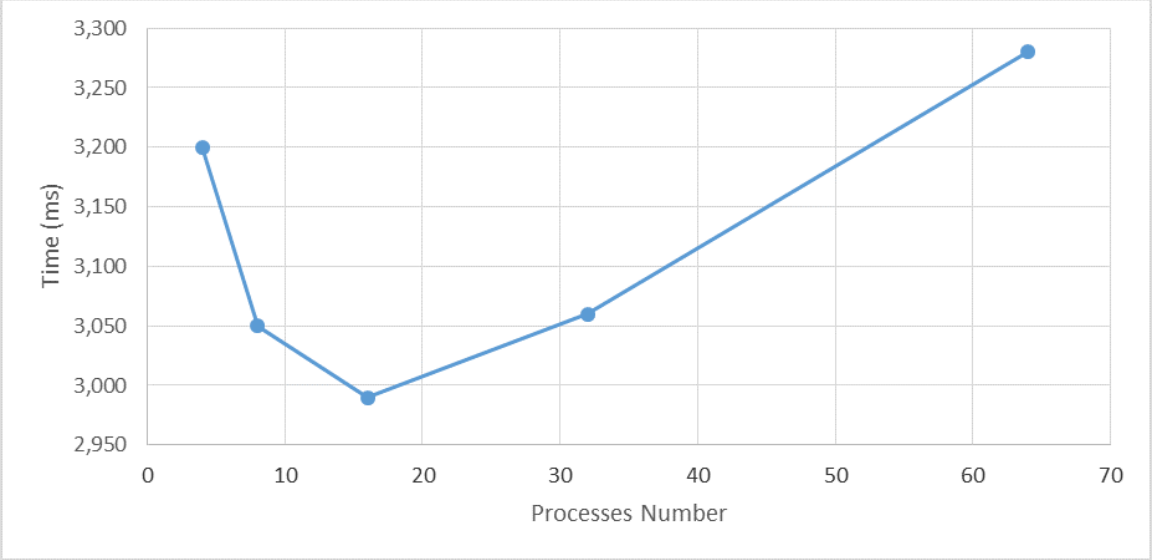

Figure 4.11: Visualize 560640 records with different number processes.

4.1.6 Conclusion

In this section, we proposed a new workflow to interact and visualize big data for web-based client/server applications. The basic idea is to visualize and update the visualization results in the server side and only transfer visualization result images to the client side. This is different from the traditional workflow in that it transfers data and finishes interaction and visualization tasks in the client side. We did some experiments that showed that the traditional workflow is better than the new workflow if users have a small dataset (less than 100,000 floats). If users have a large dataset, the new workflow performs much better than the traditional workflow. When we used the traditional workflow to process a very large amount of data, the performance drops dramatically.

4.2 Service 2: Model Accuracy Enhancement

4.2.1 Overview

This section studies how to improve the accuracy of hydrologic models using machine learning models as post-processors and presents possibilities to reduce the workload to create an accurate hydrologic model by removing the calibration step. It is often challenging to develop an accurate hydrologic model, due to the time-consuming model calibration procedure and the non-stationarity of hydrologic data. Our findings show that the errors of hydrologic models are correlated with model inputs. Thus motivated, we propose a modeling error learning based post-processor framework by leveraging this correlation to improve the accuracy of a hydrologic model. The key idea is to predict the differences (errors) between the observed values and the hydrologic model predictions by using machine learning techniques. To tackle the non-stationarity issue of hydrologic data, a moving window based machine learning approach is proposed to enhance the machine learning error predictions by identifying the local stationarity of the data using a stationarity measure developed based on Hilbert-Huang transform. Two hydrologic models, the Precipitation-Runoff Mod-

eling System (PRMS) and the Hydrologic Modeling System (HEC-HMS), are used to evaluate the proposed framework. Two case studies are provided to exhibit the improved performance over the original model using multiple statistical metrics.

4.2.2 Introduction

Motivation

Hydrologic models are commonly used to simulate environmental systems, which help to understand the water systems and their responses to external stresses. They are also widely used in scientific research for physical-process studies and environmental management for decision support and policy-making [35]. One of the most important criteria for model performance evaluations is prediction[1] accuracy. A reliable model is able to capture the hydrologic features with robust and stable prediction. However, it is challenging to develop a reliable hydrologic model with low biases and variances. In this section, we aim to develop a post-processor framework to improve the relability of hydrologic models.

Hydrologic models are typical environmental models for hydrologic process studies and water resources evaluations. Among all types of hydrologic models, physically based parameter-distributed hydrologic models have become increasingly prevalent as they are able to capture detailed features within hydrologic systems. However, in regions with high hydrologic heterogeneities, a large number of parameters are required to represent both temporal and spatial variation. This requests a large amount of computational resources, which substantially increases difficulties in a model development, data assimilation, and model calibration [137]. The resulting high cost of computation makes it challenging to implement data assimilation techniques such as Ensemble Kalman Filters [73, 113], or use an optimization method such as Shuffled Complex Evolution [31, 32]. On the other hand, the post-processing methodology dealing with model results can potentially mitigate such computation requirements

[1] In the hydrologic modeling results, while the term "simulations" is widely used for both concepts of historical records replication and future prediction, the term "predictions" is used in this study for the purpose of keeping a consistency with term used in the numerical post processing.

and improve the performance [137]. Therefore, the post-processor approach is studied and used in this section. By studying many hydrologic scenarios, we observe that the hydrologic model errors often follow some patterns that are highly correlated with model inputs (see Figure 4.14). Such patterns can be learned via machine learning (see Section 4.2.3) and applied in predictions. Thus motivated, we propose a machine learning based post-processor framework that can learn the modeling error to enhance the prediction accuracy.

Despite the potential improvement brought by machine learning techniques, it is worth noting that pure machine learning techniques cannot completely replace hydrologic models. When we compare the performance of the environmental model and machine learning methods, it turns out that the accuracy of the Precipitation-Runoff Modeling System (PRMS) [70, 80, 82] is much higher than that of commonly used machine learning techniques (e.g. random forest tree [72] and gradient-boosted tree [48]). Compared to hydrologic models developed using domain knowledge, pure machine learning models with limited training data cannot accurately characterize all the features of the underlying physical process. Nevertheless, based on hydrologic simulation machine learning approaches are able to further enhance hydrologic model results, by predicting the original modeling errors via learning the relationships between model inputs and output simulation results.

Major Contributions

In this section, we develop a modeling error learning based post-processor framework to enhance the prediction accuracy of hydrologic models. Based on the results in Section 4.2.4, the proposed framework can ease the parameter tuning processes and achieve accurate predictions. The key idea is to leverage the correlation between the hydrologic model inputs and model output errors. There are two main challenges of building the proposed framework: 1) how to improve the efficiency and accuracy in a hydrologic model in terms of model simulation and development and 2) how to deal with the non-stationary hydrologic data. To solve the first challenge, we propose

a machine learning based post-processor, which can capture and characterize model errors to improve hydrologic model predictions. This can help avoid the misleading effects of irrelevant model inputs. Also, we propose to clean and normalize the data, which enable better characterization of the correlation. To solve the second challenge, we propose a window size selection method, which identifies local stationary regions of the data by using a stationarity measure based on Hilbert-Huang transform (HHT) [53]. The key idea is to first find all possible window sizes by using data autocorrelation and then select the best window size, which contains the most stationary data. The stationarity measure is proposed to calculate the data stationarity within a window. The two major contributions of this section are summarized as follows:

- A machine learning based post-processor framework is developed to improve the prediction accuracy and flexibility of hydrologic models. One common issue of existing hydrologic simulation studies is that the development of hydrologic models, in terms of calibration processes, often requires long research time cycles but ends up with barely-satisfied model accuracy. To tackle these challenges, the proposed framework can significantly simplify the parameter tuning processes by learning and calibrating the modeling error using machine learning techniques. Moreover, the proposed framework can use different machine learning methods for different scenarios to obtain the best results, and the model parameters can be dynamically updated using the latest data. Our experiment results in Section 4.2.4 show that our method can significantly improve the prediction accuracy, compared with the simulation results of existing hydrologic models.

- A moving window based machine learning approach is proposed, which can enhance the performance of the machine learning technique when dealing with non-stationary hydrologic data. We observe that the distribution of hydrologic data changes over time and the data exhibits seasonality (see Figure 4.13). The proposed moving window based machine learning approach can characterize the time-varying relationship between the model inputs and model output errors.

The key step is to choose a suitable window size, within which the data is stationary, as most machine learning techniques are designed for stationary data. By leveraging recent advances in the field of nonlinear and non-stationary time series analysis, particularly HHT, we propose the degree of stationarity to measure the local stationarity of the data. Based on the degree of stationarity and the autocorrelation, we propose a window size selection method to optimize the performance of the machine learning techniques.

The proposed framework has been evaluated on the basis of different hydrologic models. The framework can improve the accuracy of the original hydrologic models and the window selection method can find the data pattern and select a suitable window size. Moreover, we find that the accuracy of an uncalibrated hydrologic model is as good as the calibrated one by using the proposed framework, which indicates that the proposed framework can replace the complicated "calibration" step in the traditional hydrologic model developing workflow. Section 4.2.4 introduces more details of the case studies.

Related Work

An appropriate window size is very important for training a machine learning model to deal with non-stationary time series data. Most of the existing work on the window size selection is based on concept drifts and distribution changes. There are some methods that perform well but can only be applied to a certain machine learning method, such as [38, 61]. Figuerol, Carles, and Gavaldà in [38] proposed a concept drift based method to dynamically adapt window size for Hoeffding Tree [28]. To solve the limitation, some methods are proposed that can be applied to different machine learning techniques by using statistical techniques to monitor the concept drifts. In [12, 62, 68], Statistical Process Control (SPC) [92] is leveraged to monitor the data change rate by using error rate. If the error rate change is larger than a threshold, it means the data is not stable, and then the window size should be changed. These methods need to assume that the error rate follows a certain distribution, and

then calculate the threshold by using the error confidence interval. Similarly, in [10, 43], window selection methods are proposed based on the concept of context with the stationary distribution. The proposed methods require the dataset inside a window to follow a certain distribution, and then calculate the confidence interval by using an approximate measurement [10, 43]. However, this requirement may not be satisfied for some hydrologic data because the data distributions may be not known or follow a certain distribution. Different from these works, we choose the window size based on the degree of stationarity of the data, based on the proposed stationarity measure (see Section 4.2.3), which does not assume the data follows any predetermined distribution and is applicable to different machine learning techniques.

There are many methods to improve the performance of hydrologic model simulations by reducing uncertainties from various sources: model input pre-processing, data assimilation, model calibration, and model result post-processing [137]. Model input pre-processing deals with uncertainties from model input variables such as establishing precipitation measurement networks or post-processing meteorological predictions [45]. Data assimilation treats the uncertainties from model initial and boundary conditions. For instance, the assimilation of snow water equivalence data can improve initial conditions in a snow or hydrologic model [3, 113]. Model calibration technique reduces the uncertainty from model parameterization [30, 31], such as using a transformation of model residuals to improve the model parameter estimations [103], or using optimization algorithms to find best parameters that fit the observations [49, 112]. Post-processing quantifies and reduces the uncertainties related to model results. Statistical models are usually used for post-processing, which calculates the conditional probability of the observed flow given forecast flow [107, 137]. Examples include variants of Bayesian frameworks built on model output [65], the meta-Gaussian approach [87], the quantile regression approach [107], and the wavelet transformation approach [115]. Because the post-processing methodology only deals with model results it requires less computations for most cases. Therefore, we propose to use the post-processing method in this framework.

There are many different post-processing approaches being used for hydrologic modeling. According to Brown and Seo (2013), the existing algorithms generally varies, in terms of: 1) the source of bias and uncertainties; 2) the way of predictor developed using prior available data; 3) the assumptive relationship between predictors and model simulations; 4) the uncertainty propagation techniques; 5) the model method used in spatial, temporal, and cross-dependencies simulation; and 6) the parameterization means. Specifically, Zhao, Duan, Schaake, Ye, and Xia (2011) introduced a general linear model, which leveraged and removed the mean bias from the original model outputs, to improve the original model predictions. Quantile Mapping (MQ) method was used as an effective method, which uses Cumulative Density Functions (CDFs) of observations and simulations to remove corresponding differences on quantile basis [47, 129]. Based on this, Madadgar, Moradkhani, and Garen (2014) proposed a couple equations of univariate marginal distributions joint CDFs that further improved the representation of the inherent correlations between observations and simulations, and the separation of the marginal distribution of random variables. Brown and Seo (2010) designed an advanced data transformation method for non-parametric data using Conditional Cumulative Density Function (CCDF) [105], which has been successfully applied to nine eastern American river basins [13]. Krzysztofowicz and Maranzano (2000) proposed a Bayesian based methodology using normal quantile transform in a Meta-Gaussian distribution as a way to remove model biases. However, these methods that rely on the original model calibration are limited to the applied basins [139], variable uncertainties, the static dataset in use, and instabilities by data outliers and "ancient" dataset [14]. which can substantially reduce the result performance and reliability of the post-processing algorithms.

4.2.3 Modeling Error Learning Based Post-processor Framework

Hydrologic models are based on the simulation of water balance among principal hydrologic components. With different study purposes, the selected hydrologic model

varies and so as the parameters used in the simulation algorithm. It is challenging to develop an accurate hydrologic model and traditional hydrologic models can often have high biases and variances in the outputs. By studying many hydrologic scenarios, we observed that the hydrologic model errors often follow some patterns that highly correlate with the model inputs and such patterns can be learned via machine learning. Thus motivated, we propose a machine learning based post-processor framework that can learn the modeling error to enhance the prediction accuracy. The details of the proposed framework are provided in the follow section.

Observations and Motivations

We study the prediction errors of a PRMS model [70, 80, 82] using 10-year historical watershed data collected from USGS [121]. The study area is the Lehman Creek watershed in eastern Nevada and the data is collected every 24 hours. Figure 4.12 illustrates the error distribution of streamflow prediction from the PRMS model. The distribution is very close to a normal distribution with a close-to-zero mean value and a low variance. However, when taking a closer look at the prediction errors across time (see Figure 4.13), we observe a large discrepancy between the model outputs and the ground truths in the middle of each year. It implies that the current PRMS model cannot accurately characterize the streamflow in the middle of a year. Therefore, there is a need to better capture the dynamics of the streamflow in this time period.

Intuitively, the prediction errors contain important information, which can be leveraged to reduce the hydrologic model errors, so as to improve the prediction accuracy. Therefore, we explore the information contained in the prediction errors and find that the prediction errors are actually highly correlated with the model inputs. As shown in Figure 4.14, during May, June, July, and August of the year 2011, the streamflow prediction errors are highly correlated with the temperatures and time (month and day). The larger correlation values and stars in Figure 4.14 in the upper side mean the closer relations between two variables. By leveraging the correlations, we aim to predict the original model errors and thereby improve the

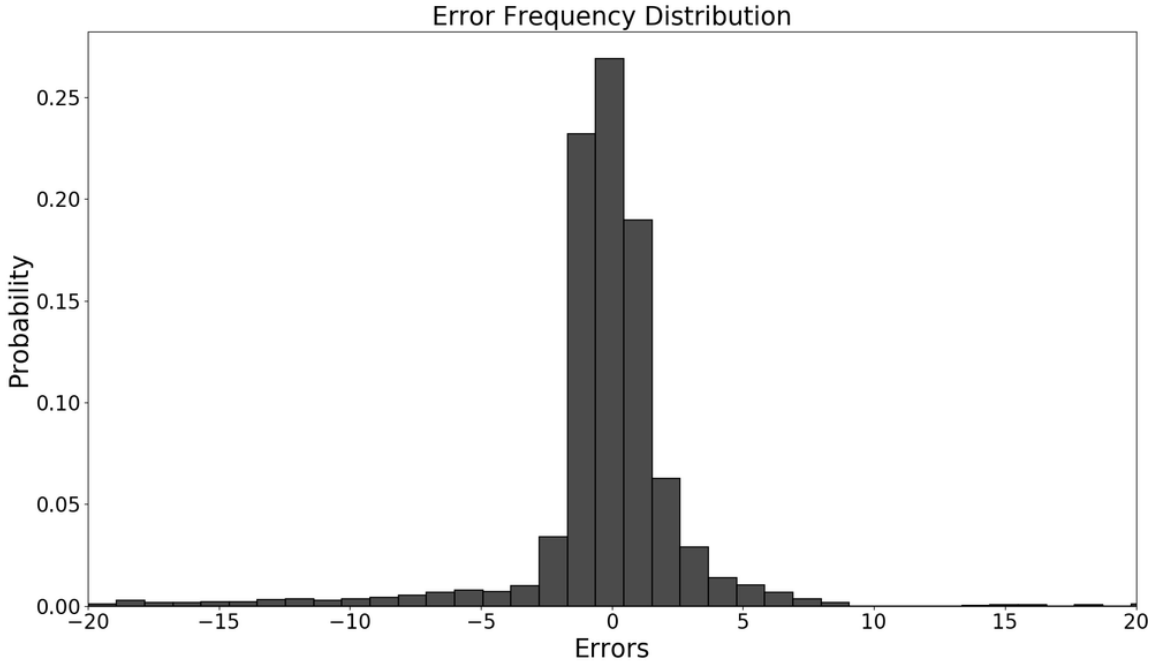

Figure 4.12: A traditional calibrated prms model streamflow prediction errors histogram (example of the Lehman Creek).

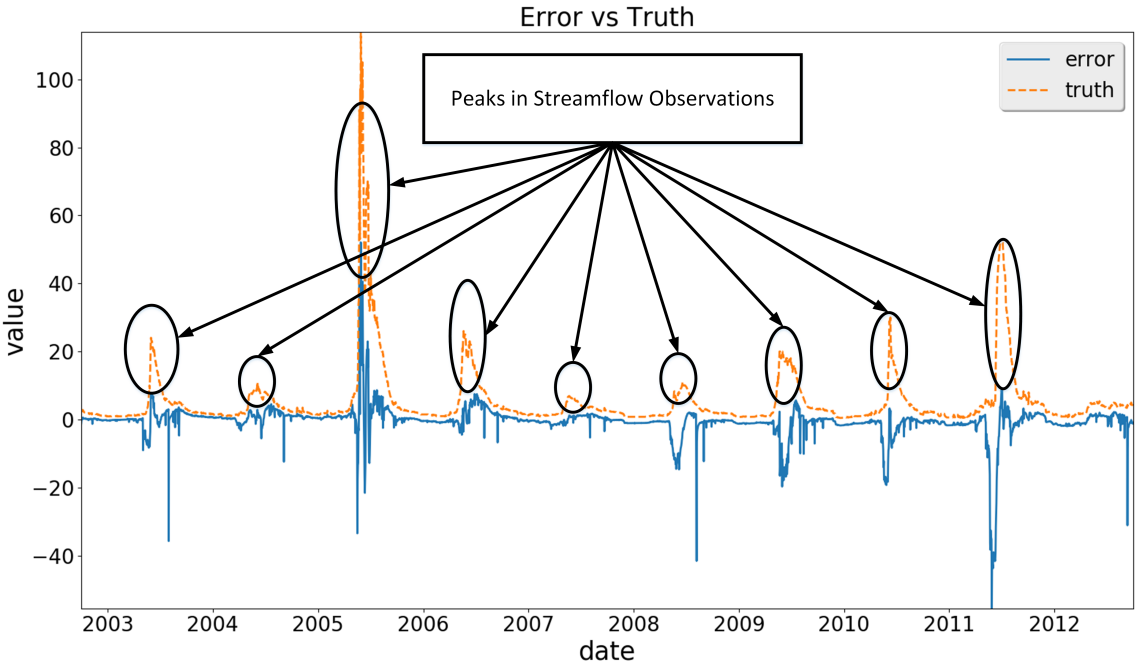

Figure 4.13: Comparisons between streamflow observations and prediction errors from a traditional calibrated PRMS model (example of the Lehman Creek).

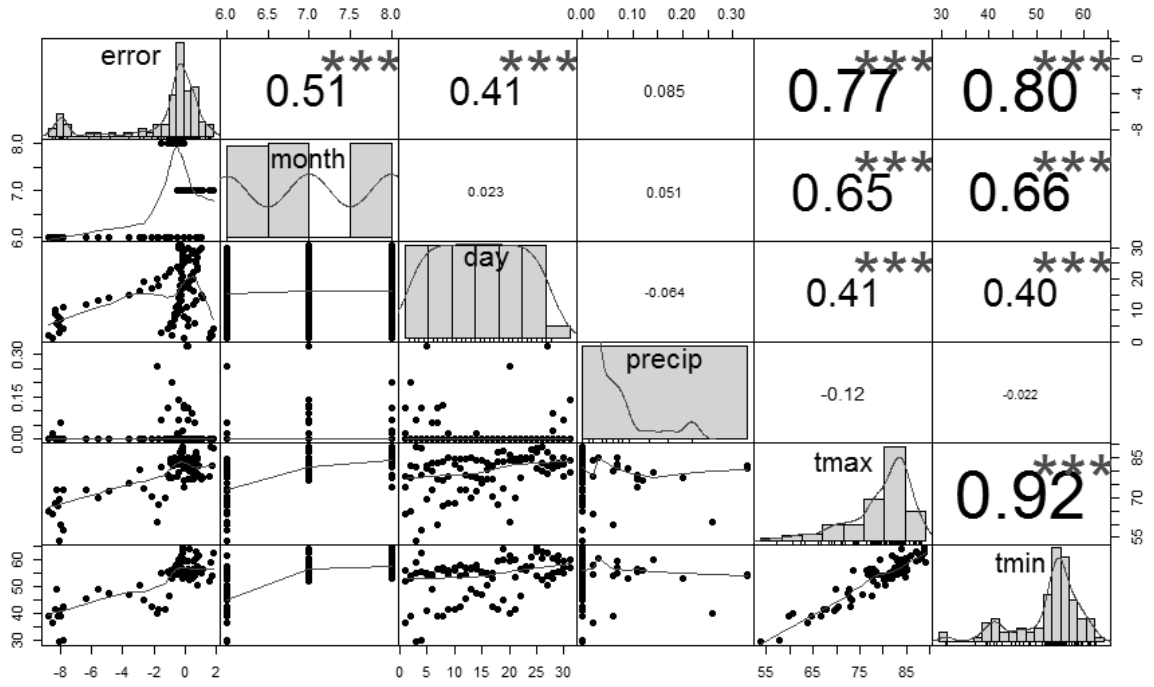

Figure 4.14: Correlations between PRMS inputs (i.e. *precip*, *tmax*, and *tmin*) and streamflow prediction *Errors*, during May, June, July, and August (2011): The diagonal graphs show the variable distributions, the lower side graphs show the scatter plots between the corresponding row and column variables, and the upper side values are the correlation values between the corresponding row and column variables. (*precip*: precipitation; *tmax*: maximum temperature; *tmin*: minimum temperature); *errors*: streamflow prediction errors.

prediction accuracy.

Along this line, we propose to use machine learning techniques to learn the modeling errors by leveraging the strong correlations between the prediction errors and the model inputs, in order to improve the accuracy in streamflow predictions. The proposed framework is illustrated in Figure 4.15. It mainly consists of three steps:

- Step 1: Develop a hydrologic model, such as PRMS. The model can generate predictions (e.g., streamflow prediction) based on the inputs (e.g., temperature, time, and precipitation).

- Step 2: Obtain the hydrologic model errors. By comparing the ground truths with the hydrologic model predictions, the framework can collect historical hy-

drologic model errors.

- Step 3: Preprocess history errors and build a machine learning model. The hidden correlations between the model errors and the model inputs can be enhanced after preprocessing and be characterized by a machine learning model.

After these three steps, the trained machine learning model is integrated with the original hydrologic model to enhance the prediction accuracy. It produces the improved results by adding the predicted errors with hydrologic model predictions. Different methods in each "Preprocessor", "Machine Learning Model", and "Hydrologic Model Errors" component can be selected based on the needs of applications. The details of each component as shown in Figure 4.15 are described in the following sections.

Remarks: In practice, the development of a hydrologic model needs to be calibrated based on hydrogeologic conditions and meteo-hydrologic characteristics. The calibration procedure is a process that finalizes parameters used in the model numerical equations that determining the hydrologic process simulation. With temporal and spatial heterogeneity, these parameters could either be characterized with both these features, such as in a physically based parameter-distributed hydrologic model PRMS, or be averaged to represent a mean level while still maintaining the capability of capturing the streamflow variation, such as in the Hydrologic Modeling System (HEC-HMS). In this study, the default values of each parameters are used in the uncalibrated cases as to compare with the calibrated cases from traditional hydrologic calibration and post-processor methods. As demonstrated in Section 4.2.4, the proposed framework provides a better prediction accuracy with high processing efficiency when compared with the traditional hydrologic calibration method.

Modeling Error Learning enhanced hydrologic Model

The detailed workflow of the designed modeling error learning enhanced hydrologic model is illustrated in Figure 4.16. The basic idea is to use predicted error to calibrate

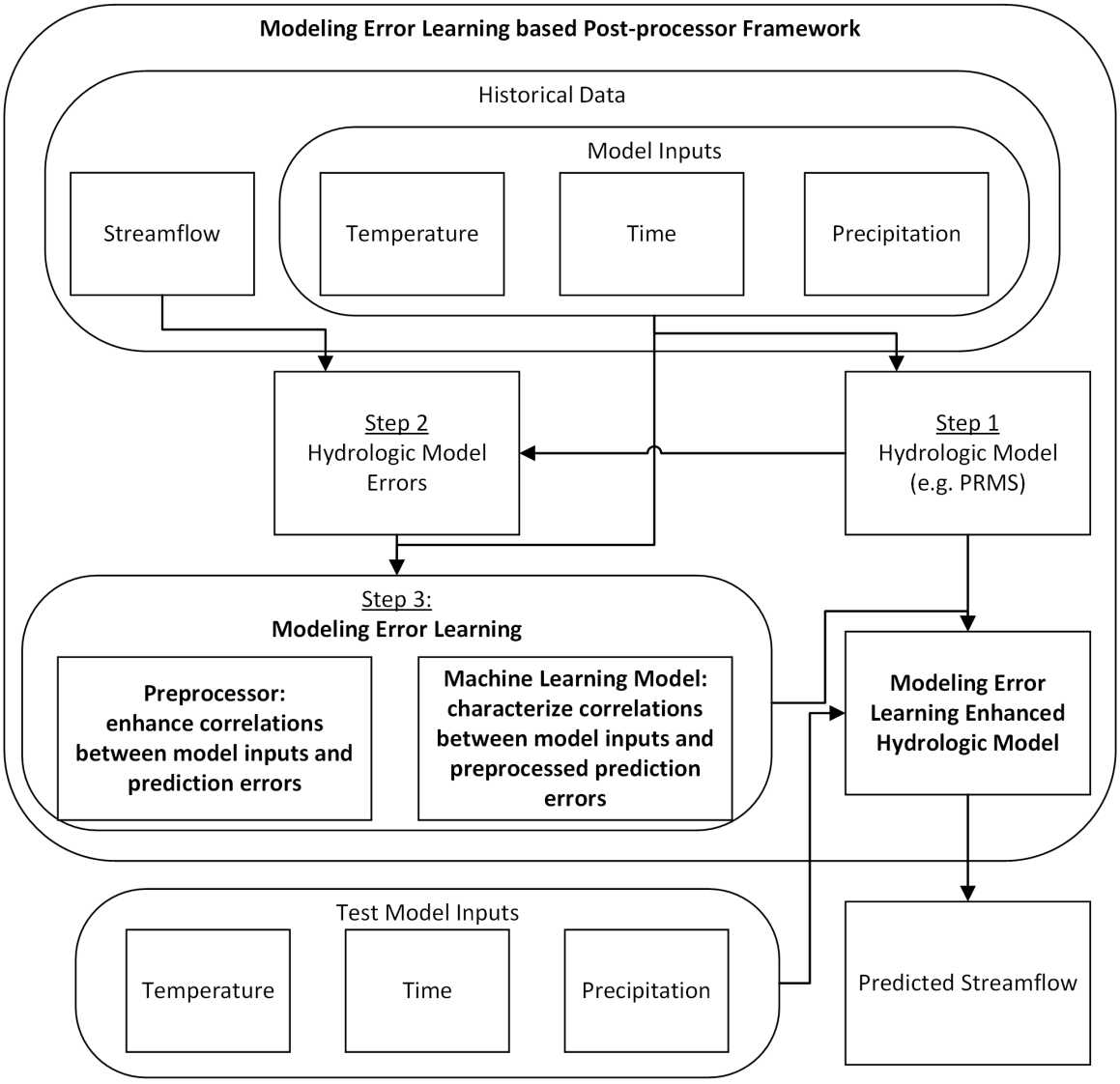

Figure 4.15: The diagram of modeling error learning based model post-processor framework.

original hydrologic model's predictions as shown in Equation (4.2)

$$\hat{p}_t = f(x_t) + g(x_t) \tag{4.2}$$

where $\hat{p}_t$ denotes the improved prediction at time t; x_t denotes the model inputs (i.e., temperature, time and precipitation) at time t; $f(\cdot)$ denotes the hydrologic model, which generates predictions based on x_t; and $g(\cdot)$ denotes the error prediction model learned in the "Machine Learning Model" component, which generates hydrologic model prediction error based on x_t.

As illustrated in Figure 4.16, there are basically three steps to build an enhanced hydrologic model:

- Step 1: Calculate the hydrologic model errors. We calculate errors using differences between the observations and model predictions in the "Hydrologic Model Errors" component.

- Step 2: Enhance the correlation between hydrologic model errors and inputs. This step contains two sub-steps: "scale model error" and "data transformation". "Scale model error" is used to scale error into a certain scope (e.g. between 0 and 1) and "data transformation" is used to normalize hydrologic model errors and stabilize the variances of hydrologic model errors.

- Step 3: Build a machine learning model. The scaled and transformed original hydrologic model errors and model inputs are used to train a machine learning model to predict the hydrologic model errors. The predicted errors need to be back-transformed and back-scaled before being used to compensate the hydrologic model results.

More details of the framework components (rectangles in Figure 4.16) and steps (arrows in Figure 4.16) are introduced in the follow sections.

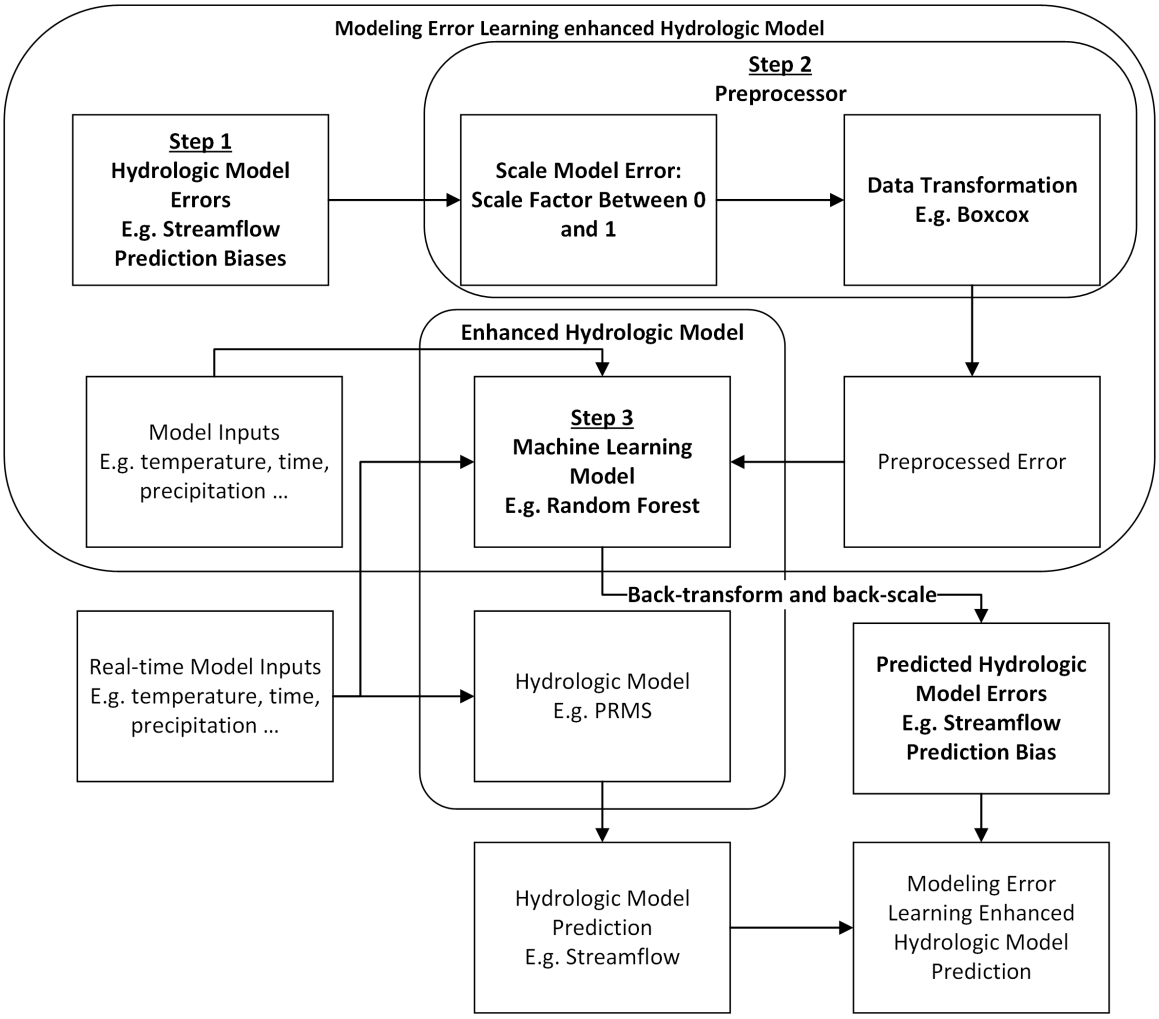

Figure 4.16: Modeling error learning enhanced hydrologic model.

Preprocessor Component

The "preprocessor" component preprocesses the hydrologic model errors, and the outputs of this component are used to train a machine learning model in the "Machine Learning Model" component. The objective of the "preprocessor" component is to normalize errors and reduce error variances. In other words, this component is used to make it easier for the "Machine Learning Model" component to characterize correlation between the hydrologic model inputs and errors. Specifically, this component scales and transforms the hydrologic model prediction errors using Equation (4.3)

$$e_t = tr(\alpha e) \tag{4.3}$$

where e_t denotes preprocessed error; $tr(\cdot)$ denotes transformation function, α denotes the scaling factor; e denotes the original hydrologic error. Based on the case studies in Section 4.2.4, a good scaling factor is often between zero and one.

Note that in this framework different functions can be selected based on the dataset characteristics. For example, if the dataset is positively skewed, log-sinh transformation [126] could be helpful. If the dataset has a large variance, boxcox transformation [127] may be applied. In Section 4.2.4, Case Study 1 uses the log-sinh transformation (see Equation 4.14) and Case Study 2 uses the boxcox transformation (see Equation 4.16). These transformation functions can improve the hydrologic model outputs, as shown in Section 4.2.4.

Remarks: The "Preprocessor" component should be repeated multiple times to find out the best-performed scaling factor and data transformation parameters. For example, %-time cross validation can be used to test all possible parameter combinations' performance [64]. The performance can be measured by using RMSE (Equation 4.9), PBIAS (Equation 4.10), NSE (Equation 4.11), or CD (Equation 4.12). After a good parameter combination is chosen, it will be used in both the "Preprocessor" component and the "Back-transform and Back-scale" step.

Machine Learning Model Component

The "Machine learning model" component aims to predict the transformed hydrologic model error $\hat{g}(x_t)$, using the hydrologic model input x_t. To obtain the original model prediction error $g(x_t)$, $\hat{g}(x_t)$ needs to be transformed back using the inverse of the transformation function, which is discussed in Section 2.2.4. In what follows, we discuss how to find $\hat{g}(\cdot)$ using machine learning techniques.

There are many machine learning techniques that can be applied in this component, such as Support Vector Regression (SVR) [7] and gradient boosted tree [48]. Most of them are designed for stationary environments, in the sense that the underlying process follows some stationary probability distribution. However, hydrologic processes are often non-stationary. As illustrated in Figure 4.13, the streamflow shows seasonality in the sense that the patterns of streamflow in each year are similar but change over time. To address this challenge, we propose to use a moving time window to adapt to the changes due to hydrologic data variations.

The basic idea is to set up a time window and train the machine learning model using the data within the window, which moves over time. By using the time window, we are able to track the changing dynamics of hydrologic data. However, it is challenging to find an appropriate window size. If the window size is too large, it increases model training complexity and the model is not able to quickly adapt to the changes of the hydrologic data. Even though a model with a large window size may generate accurate results during the training phase, it is possible that the accuracy of the model using the test dataset could be very poor, which is due to overfitting issue [27]. If the window size is too small, the model may not be able to capture the pattern of the hydrologic model errors.

In this section, the window size selection is based on the pattern and the degree of stationarity of the data, which can not only capture the data pattern, but also ensure the data stationarity within the window.

To find the data pattern, we leverage the autocorrelation of the data. Due to the seasonality, the autocorrelation shows a peak every year (see Figure 4.18)

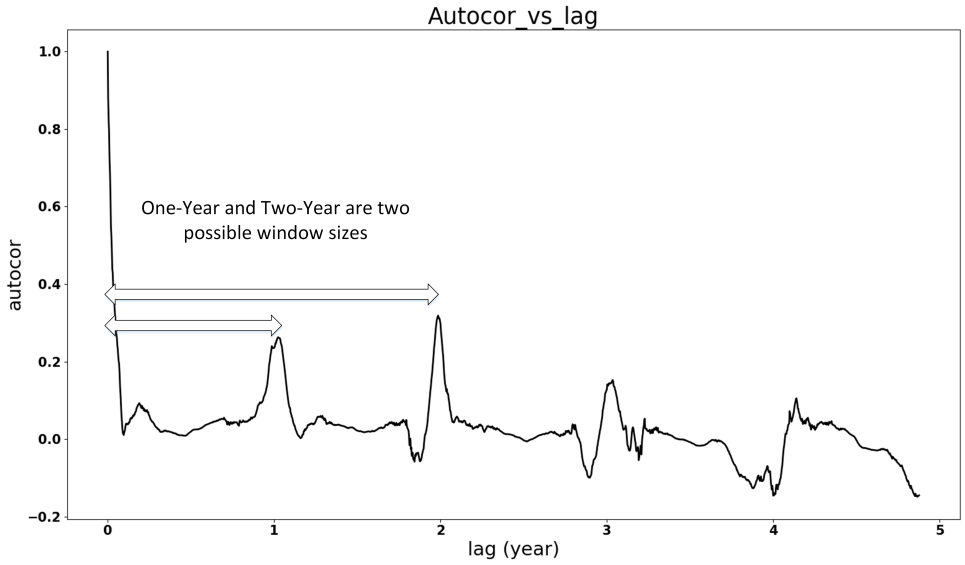

Figure 4.17: Case study 1 training data autocorrelation values vs lag days: one-year and two-year can be the data pattern lengths, because these are the distances between the start point and peaks in the training data.

and the distance between two peaks indicates that the pattern repeats during this period. However, as illustrated in Figure 4.18, there are several peaks, and it remains challenging to determine the window size, i.e., "how many peaks should be chosen?"

To address this challenge, we further calculate the degree of stationarity of the data in a given window size and use this to determine the window size. Specifically, the degree of stationarity (DS) is defined by leveraging recent advances in the field of nonlinear and non-stationary time series analysis, particularly Hilbert-Huang transform (HHT) [53]. DS is defined as:

$$DS(T) = \frac{\sum_{\omega} \hat{DS}(\omega) n(\omega)}{n_{sum}} \tag{4.4}$$

$$\hat{DS}(\omega) = \frac{1}{T} \sum_{t=0}^{T} (1 - \frac{H(\omega, t)}{n(\omega)})^2 dt \tag{4.5}$$

$$n(\omega) = \frac{1}{T}\sum_{t=0}^{T} H(\omega, t) \tag{4.6}$$

where $DS(T)$ denotes the data stationarity value of window size T (Equation 4.4), $\hat{DS}$ can characterize the variation of the data in a certain frequency (ω) bin over time (Equation 4.5), $n(\omega)$ is the average amplitude of the frequency (Equation 4.6).

In Equation 4.4, $n_{sum} = \sum_{\omega} n(\omega)$. $DS(T)$ sums $\hat{DS}$ value of each frequency and weights each of them by using $n(\omega)$. This ensures that small, relatively insignificant oscillations do not dominate the metric. n_{sum} in the denominator part normalizes $DS(T)$ and allow different DSs to be comparable. Note that the larger DS, the more non-stationary the data, and we prefer a small DS in a given time window.

In Equation 4.5, $H(\omega, t)$ denotes the Hilbert spectrum, which is a frequency-time distribution of the amplitude of the data. A large $\hat{DS}$ indicates large variations in the bin, which means non-stationary behavior. A close-to-zero $\hat{DS}$ indicates small variations in the bin, which means stationary behavior.

The $\hat{DS}$ concept is first introduced in section [53] but it only considers the data stationarity of a certain frequency bin and does not characterize the entire time series data stationarity. To improve the $\hat{DS}$ concept, we propose DS that calculates the whole dataset stationarity.

After the possible data patterns are chosen based on autocorrelation, the data pattern that has the minimum DS (the most stable) is chosen to be the final window size.

Figure 4.18 illustrates the values of DS under different window sizes for Case Study 1 in Section 4.2.4. The DS value increases as the window size grows, which means the data becomes more non-stationary when the window size grows. As the one-year DS is smaller than two-year DS, the one-year window size is chosen for the Case Study 1, because it is one of the data patterns and this window size has the minimum DS value. Figure 4.19 compares the prediction performance using different window sizes for Case Study 1. It shows the one-year window size has the best performance. In contrast, the 4.5-year window size is more accurate than one-year

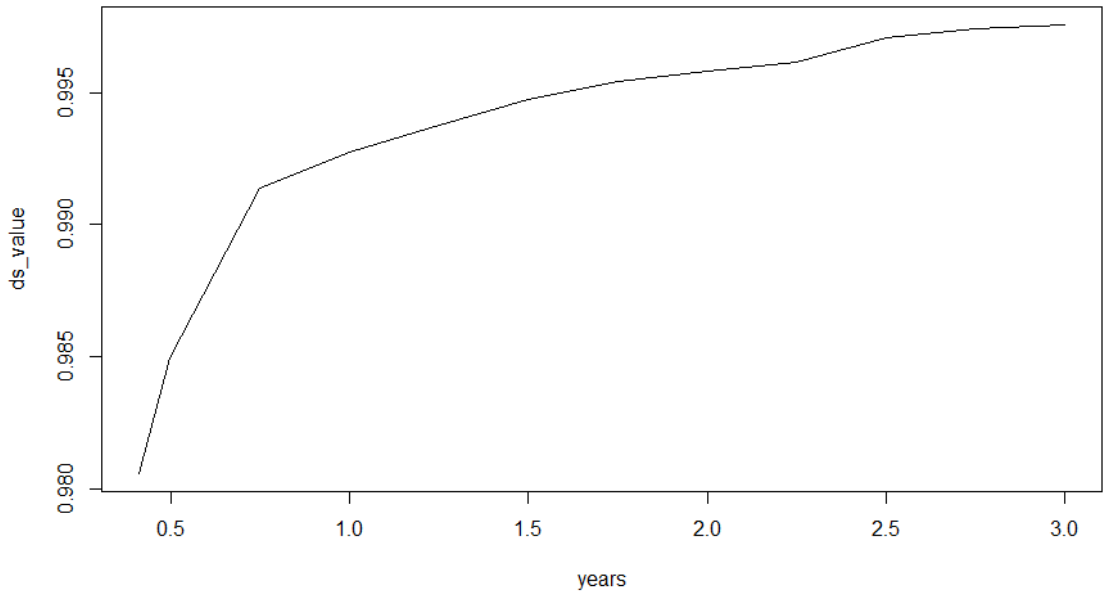

Figure 4.18: Case study 1 training data DS vs window size. One-year DS is slightly less than two-year DS.

window size with the training dataset but the performance is worse with the testing dataset, which means a larger window size can cause overfitting issues.

Back Transform and Back Scale

The predicted errors generated from the "Machine Learning Model" cannot be used directly because the machine learning model is trained with the preprocessed errors. The predicted errors need to be back preprocessed using the corresponding preprocessor methods to obtain the real predicted hydrologic model errors.

Let tr^{-1} denote the inverse of the transformation function. $g(x_t)$ can be computed as follows:

$$g(x_t) = tr^{-1}(\hat{g}(x_t))/\alpha \tag{4.7}$$

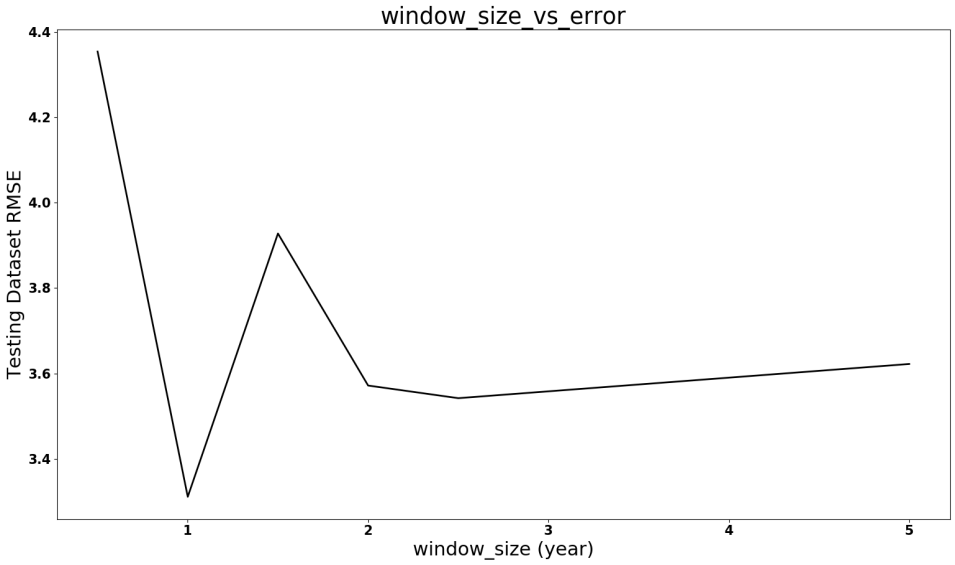

Figure 4.19: Case study 1 testing data RMSE vs window size: the one-year window size is better than other window size based on rmse value.

And the prediction $\hat{p}_t$ can be given as:

$$\hat{p}_t = f(x_t) + tr^{-1}(\hat{g}(x_t))/\alpha \tag{4.8}$$

Discussion of Proposed Methods

The "Modeling Error Learning" is the key component of the framework. If it is able to predict the hydrologic model errors, the framework works. If not, then the framework cannot improve a hydrologic model performance. Therefore, the question "when the framework does not work" equals "when the 'Modeling Error Learning' component cannot predict errors accurately". Because this component leverages the relations between the model inputs and model errors, the component can work when the model inputs are correlated to the model errors. Therefore, a modeler can calculate the correlation values between each model inputs and the preprocessed model errors of the historical data to test if the proposed framework can work. If some model inputs are correlated with the preprocessed model errors, then the proposed framework is

able to improve the hydrologic model accuracy and vice versa.

"How the framework can perform better" is another important question. It depends on the chosen machine learning techniques used in the "Modeling Error Learning" component. The errors contain biases and variances. Based on bias-variance tradeoff theory [42], when bias decreases, variances will increase and vice-versa. Different machine learning techniques have different characteristics. For example, a boosted tree has a high bias, low variance, and performs well when dimensionality is low; A random forest has a low bias, high variance, and performs well when dimensionality is high [17]. Thus, the selection of machine learning method should be determined by study needs and data characteristics.

However, it is hard to determine which machine learning technique works better for a certain problem before performing tests. We suggest to do a pre-test to exam which machine learning technique could work and perform better. The pre-test data should be part of historical data and the size is decided by the data cycle, such as a week, month, and year. For example, the temperature is high in summer and low in winter. Therefore, a "year" can be a cycle. The first two years temperatures of the historical data are chosen to be the pre-test data. The first year temperature values are used in the training phase, and the second year temperature values are used in the testing phase.

Hydrologic data can vary dramatically in a short time period, which is hard to be captured by a hydrologic model. It is also difficult for the "Machine Learning Model" component to accurately predict the hydrologic model errors. To address this issue, we propose a smooth prediction method to regulate the hydrologic model errors are less irregular and therefore enhance the performance of the "Machine Learning Model" component. Figure 4.13 is an example of dramatically-changed streamflow. The streamflow observations grow rapidly in the middle of each year and the vibrations generate small spikes along the uphills and downhills. The original PRMS model cannot characterize the spikes and generate irregular errors. Because the "Machine Learning Model" component is built based on these errors, the framework cannot

perform very well in the middle of every year and generate unnecessary peaks. We propose a method to smooth the hydrologic model predictions to avoid the spikes, which contains three steps:

1. Choose a threshold T, which should be between the maximum and minimum value.

2. Smooth the hydrologic model predictions by using T. If the difference between the previous prediction and current prediction is higher than T, then we use the previous prediction to replace the current prediction.

3. Check if the current T avoid peaks. If the current T cannot avoid any peaks, then choose a smaller T, and then go to Step 1. If there is a "plateau" (flat peak) as Figure 4.20 displays, then choose a larger T.

When an fitting T is finalized, it is used both in the training phase and in the test phase for the hydrologic model predictions. In the training phase, it can help to choose more appropriate scale factors, transformation parameters, and window size. In the test phase, it can to avoid original hydrologic model severe vibration predictions.

4.2.4 Results and Analysis

Experiment Design

Each dataset is separated into training dataset (50%) and testing dataset (50%). We use the quantitative statistics to perform the statistical evaluation of modeling accuracy in the testing step: RMSE, PBIAS, NSE, and CD. The statistical parameters are defined by the following equations:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(P_i - A_i)^2} \tag{4.9}$$

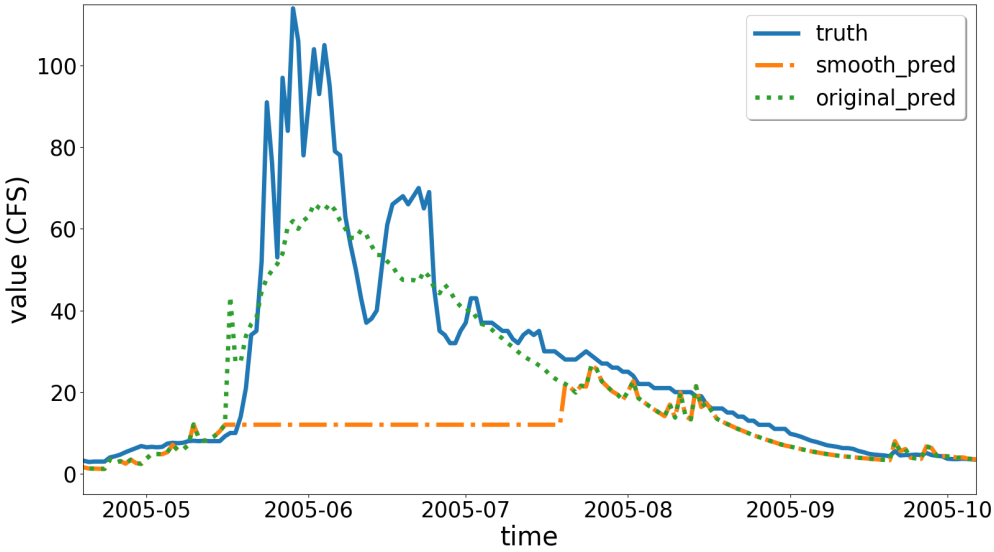

Figure 4.20: Use 10 as threshold: there is a plateau around 2005 june generated. CFS is short for cubic feet per second

$$PBIAS = \frac{\sum_{i=1}^{N}(A_i - P_i)100}{\sum_{i=1}^{N} A_i} \tag{4.10}$$

$$NSE = 1 - \frac{\sum_{i=1}^{N}(A_i - P_i)^2}{\sum_{i=1}^{N}(A_i - \bar{A})^2} \tag{4.11}$$

$$CD = \left\{ \frac{\sum_{i=1}^{N}(A_i - P_i)(P_i - \bar{P})}{\left(\sum_{i=1}^{N}(A_i - \bar{A})^2\right)^{\frac{1}{2}} \left(\sum_{i=1}^{N}(P_i - \bar{P})^2\right)^{\frac{1}{2}}} \right\}^2 \tag{4.12}$$

where P_i and A_i represent the simulated and observed values respectively; $\bar{A}$ is the mean of the observed values and $\bar{P}$ is the mean of simulated values for the entire evaluation period.

RMSE measures how close the observed data are to the predicted values while retaining the original units of the models output and observed data. Lower values of RMSE indicate a better fit of the model. RMSE is one of the important standards that defines how accurately the model predicts the response and it is commonly used in many fields.

PBIAS is a measure to evaluate the model simulations. It determines whether the predictions are underestimated or overestimated, compared to the actual observations. If the PBIAS values are positive, the model overestimates the results; otherwise, the model underestimates the results by the given percentage. Therefore, values closer to zero are preferred for PBIAS.

The Nash-Sutcliffe Efficiency (NSE) is a normalized statistic assessing the models ability to make predictions that fit 1: 1 line with the observed values. The values for NSE range between $-\infty$ and 1. For acceptable levels of performance, the values of NSE should lie close to one, and the higher NSE indicates the better results.

CD stands for coefficient of determination, calculated as the square of the correlation between the observed values and the simulated values. The values for CD ranges between 0.0 and 1.0 and correspond to the amount of variation in the simulated values (around its mean) that is explained by the observed data. Values closer to one indicate a tighter fit of the regression line with the simulated data. Similar to NSE, the higher CD values indicates the better results.

In the following case studies, we also provide Prediction Interval (PI) which offers the possible prediction range. The PI is calculated using Equation 4.13, where $\bar{X}$ is the sample mean, n is the number of samples, T_a is student's t-distribution percentile with $n-1$ degrees of freedom. PI is described with upper bound and lower bound.

$$PI = \bar{X}_n \pm T_a s_n \sqrt{1 + (1/n)} \tag{4.13}$$

Case Study 1

The PRMS Hydrologic Model

The Precipitation-Runoff Modeling System (PRMS) was developed by U.S. Geological Survey in the 1980s, which is a physically based parameter-distributed hydrologic modeling system [70, 80, 82]. The PRMS model used in this study was developed by Chen, Fenstermaker, Stephen, and Ahmad [19] in the study area of Lehman Creek watershed, eastern Nevada. The watershed is located in the Great Basin National Park, occupying an area of 5,839 acres of the southern Snake Valley [99, 123]. More than 78% of land cover were evergreen forest, deciduous forest, and mix forest, 2% of shrubs, 2% were perennial snow and ice, and 17% were barren land [19, 117]. The streamflow is mainly composed by snowmelt, which sourced from the high elevated area in the west, flowing over the large mountain quartzite and recharging the groundwater system through alluvial deposits and karst-limestone in the east [20]. These high hydro-geography variations made it appropriate to use PRMS model to describe the spatial heterogeneity of hydrologic processes. Figure 4.21 displays the study area.

On a grid-based simulation, the Lehman Creek watershed was delineated by 96 columns and 49 rows using 100 x 100-meter cell/grid. A total number of 4074 grids were formed, and based on which, the combinative effects of canopy interception, evapotranspiration, infiltration, overland runoff, and subsurface flow were simulated. The parameter estimation is one of the most critical and challenging parts of the PRMS model development. They were estimated for model algorithms and determining the model performance, using land cover land use, soil information or through literatures for each hydrologic component on each of 4704 units [19]. Among all the parameters required for model runs, some parameters are specifically sensitive and have great influences on the model simulation results. Such as parameters that determine the temporal and/or spatial distribution of precipitation, requires specification on every one of 12 months and/or every one of 4704 cells (e.g., $tmax\_allsnow$, monthly maximum air temperature when precipitation is assumed to be snow; $snow\_adj/rain\_adj$, monthly factor to adjust measured precipitation on each

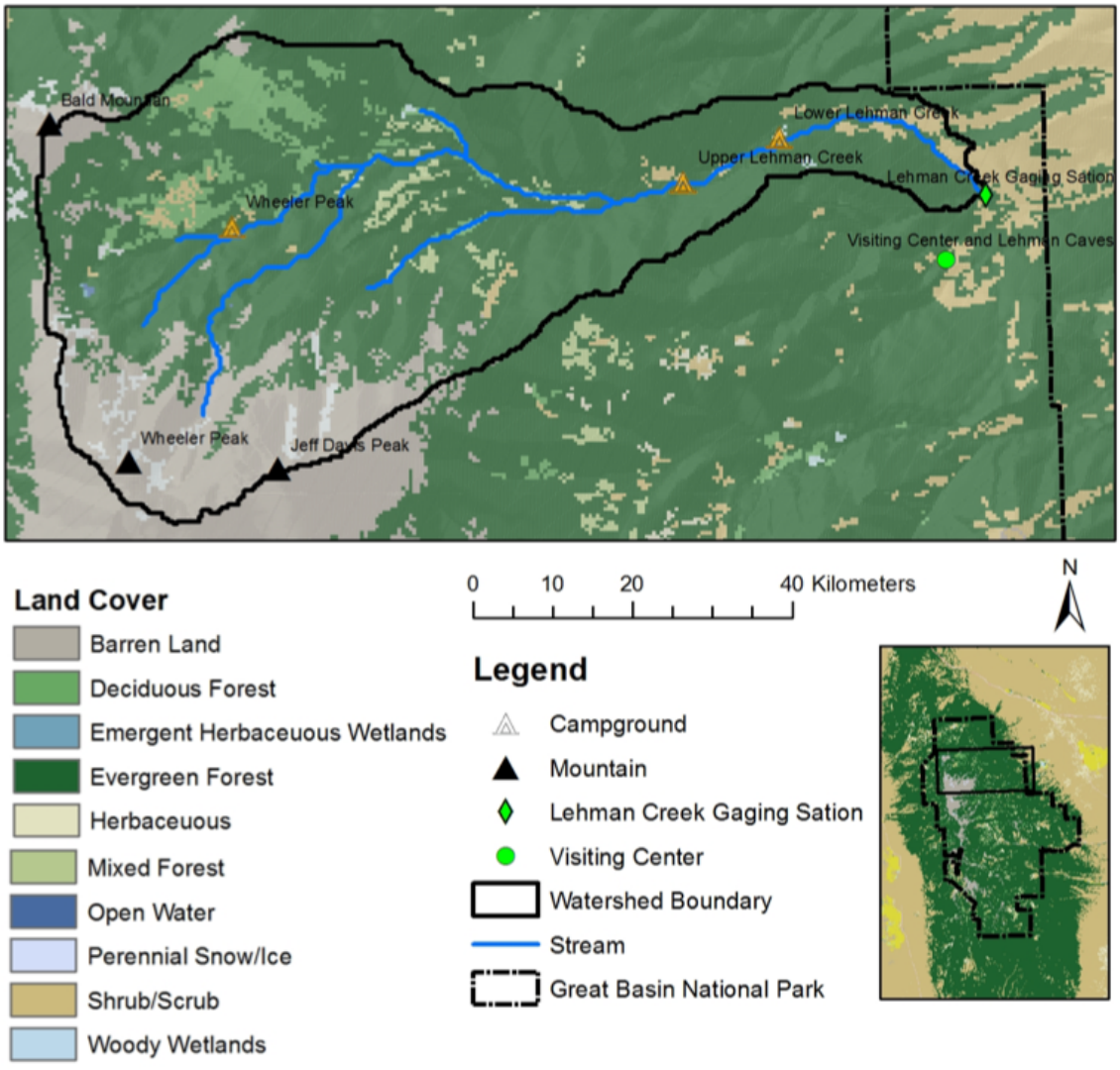

Figure 4.21: PRMS hydrologic model study area.

HRU to account for differences in elevation, and so forth; *tmin_lapse*, monthly values representing the change in minimum air temperature per 1,000 *elev_units* of elevation change).

One station meteorologic data were used as the driving forces to the developed model in the study area of Lehman Creek watershed. Daily precipitation, maximum temperature, and minimum temperature from October 1, 2003 to September 30, 2012 were collected from the meterologic station (#263340, Great Basin NP). Daily streamflows at the Lehman CK Nr Baker gauging station (#10243260) were collected for model calibration and validation [19].

Results

First, the training dataset is transformed by using log-sinh transformation, which is introduced in [126]. Equation 4.14 is the transformation equation and Equation 4.15 is the back transformation equation.

$$\hat{y} = \frac{log(sinh[a + by])}{b} \tag{4.14}$$

$$y = \frac{sinh^{-1}(10^{\hat{y}b}) - a}{b} \tag{4.15}$$

where a and b are transformation parameters. By using log-sinh transformation, the original randomly distributed errors are normalized for the convenience of correlation characterization.

During the training process as evaluated by using cross validation, we found the best scale factor α is 0.5, the best transformation parameter a is 0.0305 and b is 0.0605, where α is used in Equation 4.3; a and b are used in Equation 4.14. Gradient Boosted Trees [48] is used in the "Machine Learning Model" component and the initial window size is one-year.

Note that we find that the improved PRMS model predictions do not well follow the observations during the water recession period after the peak flow. This is

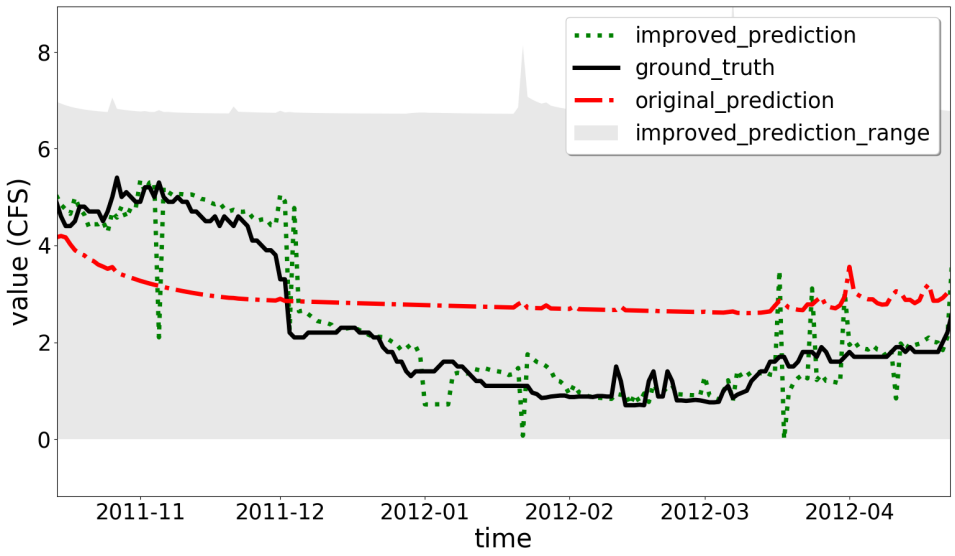

Figure 4.22: Case study 1 final PRMS model improvements. CFS is short for Cubic Feet per Second

caused by unstable historical data. By using the smooth method introduced in Section 4.2.3, the RMSE is further improved to be 2.032 with $T = 10$. The comparisons between parts of the data are shown in Figure 4.22. It is clear that the improved predictions are closer to the ground truths than the original PRMS predictions. All the statistical measurement results summarized are shown in Table 4.1. As results show, the improved predictions have lower RMSE indicating they are closer to the observed data. The PBIAS value is larger than the original PRMS model suggesting an over-estimation compared with the observations. The NSE value is closer to one, which means the improved model has a more acceptable level of performance. The CD value is closer to one means the improved model fits more to the observations. As suggested by the comparison results of model performance evaluation indicators, the proposed framework can improve the original PRMS model's results.

As suggested by statistical measurement comparisons in Table 4.2, our proposed framework can also improve uncalibrated PRMS model predictions. With the same PRMS model and input data, the RMSE is improved from 8.439 to 3.092 by using

Table 4.1: Calibrated PRMS Model Results Comparisons.

	Indicators			
Model	**RMSE**	**PBIAS**	**CD**	**NSE**
Original PRMS	4.585	***7.205***	0.769	0.768
Improved PRMS	***2.032***	10.808	***0.936***	***0.926***

Table 4.2: Uncalibrated PRMS Model Results Comparisons.

	Indicators			
Model	**RMSE**	**PBIAS**	**CD**	**NSE**
Original PRMS	8.439	-82.658	0.001	-0.292
Improved PRMS	***3.092***	***3.054***	***0.837***	***0.826***

1.0, 0.0905, 0.0805, and 10 for α, a, b, and the smooth threshold respectively. The RMSE is very close to the improved calibrated model RMSE (2.032), which indicated the proposed framework can be an effective replacement the traditional complex time-consuming calibration procedure, providing a competitive level of model performance.

Case Study 2

Hydrologic Modeling System

The Hydrologic Modeling System (HEC-HMS), released by U.S. Army Corps of Engineers in 1998, is designed to simulate the hydrologic processes of dendritic watershed system [8, 104]. Different from the PRMS model that focuses on the hydrologic components based on user-defined unit, the HEC-HMS uses a dendritic-based precipitation-runoff model with integrations in water resources utilization, operation, and management [104]. The case study of HEC-HMS was the Little River Watershed, which is an example application model in HEC-HMS program for the demonstration of the continuous simulation with the soil moisture accounting method [9]. As introduced by Bennett and Peters [9], the Little River Watershed is a 12,333-acre (19.27 m^2) basin near Tifton, Georgia. More than 50% of the land is covered by forest with remaining land used for agricultural purposes [120]. The annual precipitation is 48 inches [18].

One single-station data of precipitation observation were used, which was from

the Agricultural Research Service (ARS) rain gauge (#000038) [44]. The precipitation records were on a 15-min basis for the same model running period of January 1 1970-Jun 30 1970. The streamflow observations were from ARS gauge #74006 [44] on an hourly basis, which were used for the calibration and validation of this hydrologic model performance.

Results

In case study 2, we use Boxcox transformation [127] to transform the dataset and choose decision tree in the Machine Learning Model component to improve the hydrologic model accuracy. Boxcox transformation is simple but efficient method and able to reduce dataset variances. A decision tree consumes much less time than most machine learning methods (such as gradient boosted trees) with the same inputs in the training phase. Equation 4.16 is the Boxcox transformation equation and Equation 4.17 is the back Boxcox equation function.

$$\hat{y} = \frac{y^{\lambda} - 1}{\lambda} \tag{4.16}$$

$$y = \sqrt[\lambda]{\hat{y}\lambda - 1} \tag{4.17}$$

where λ is the transformation parameter. During the training process as indicated by using cross validation, the best α is 0.3 and the best λ is 9.0 for this case study. The window size of one-week is selected. By using our proposed method, the RMSE is 39.844 compared to 44.9833 resulting from the original HEC-HMS PRMS model. Figure 4.23 shows the prediction comparisons of parts of the data between the lower bound, upper bound, improved prediction, ground truth, and original prediction. Clearly, the improved prediction is more accurate than the original hydrologic model predictions.

As summarized in Table 4.3, RMSE of the improved model is 39.844 and it is lower than the original HEC-HMS RMSE (44.983), which means the outputs are

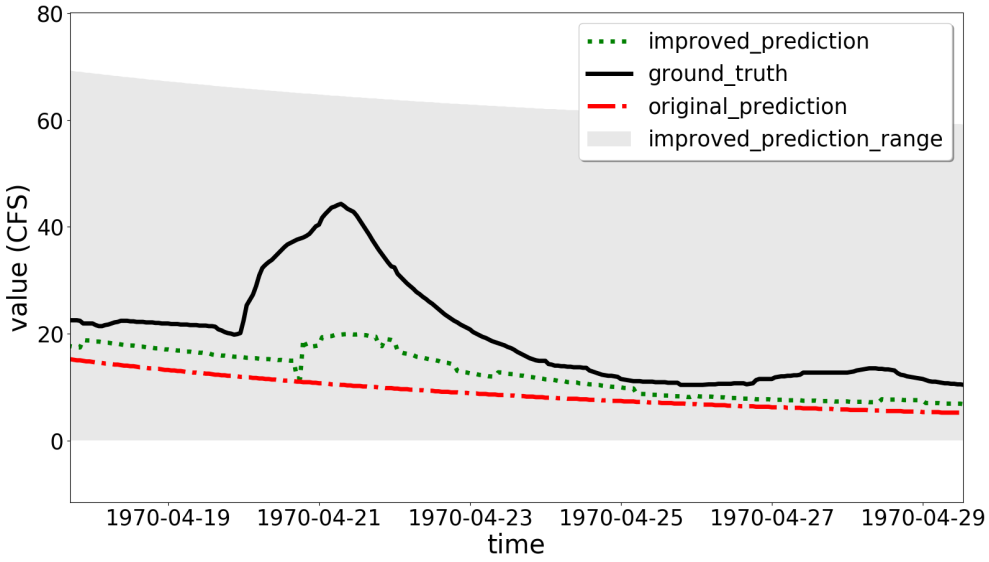

Figure 4.23: Case study 2 HEC-HMS PRMS model improvements. CFS is short for Cubic Feet per Second

Table 4.3: Calibrated HEC-HMS Model Results Comparisons.

Model	**Indicators**			
	RMSE	**PBIAS**	**CD**	**NSE**
HEC-HMS	44.983	***4.657***	0.842	0.808
Improved HEC-HMS	***39.844***	8.590	***0.884***	***0.850***

closer to the observed data. PBIAS (4.657) of the original model is closer to zero than the improved HEC-HMS PBIAS (8.590), which means the improved method over-estimates the observations. The NSE and CD values (0.850 and 0.884) of the improved HEC-HMS are closer to one than the original HEC-HMS values (0.808 and 0.842), which means the improved model has a more acceptable level of performance and fits more to the observations. The smooth method, which is introduced in Section 4.2.3, cannot improve the results. This because there are not many spikes along the uphills and downhills.

As suggested by the statistical measurement comparisons in Table 4.4, the proposed method can also improve the uncalibrated HEC-HMS model. By inputting the

Table 4.4: Uncalibrated HEC-HMS Model Results Comparisons.

Model	Indicators			
	RMSE	**PBIAS**	**CD**	**NSE**
HEC-HMS	134.610	45.943	0.768	-0.716
Improved HEC-HMS	***89.882***	***29.876***	***0.823***	***0.235***

same data, the RMSE is reduced from 134.610 to 89.882 by using 0.8 and 11 for α and λ respectively. The time window is one-week.

4.2.5 Discussion

The current study used two typical hydrologic models, PRMS and HEC-HMS, and demonstrated the performance of the proposed post-processing framework. To have a comprehensive evaluation, these two models are selected as representations from hydrologic models categories that differentiate in terms of simulation scopes, structures, and applications. As a representation of physically-based parameter-distributed hydrologic models, PRMS is widely used for research purposes, which requests large sets of parameters to simulate the physical processes; comparatively, as a representation of empirical-based lumped-parameter hydrologic models, HEC-HMS is widely used in industrial engineering purposes, which conceptualized physical bases towards result-oriented simulation.

While implementing the pre-developed hydrologic simulation, the calibrated hydrologic models were restored to the original uncalibrated status for a comparison purpose. During the restoration, the calibrated parameters were adjusted to default values either from program manuals or authors personal suggestions. This may lead to a varying restoration status of uncalibrated model performance depend on parameters suggested. However, in this study, the main goal for the development of uncalibrated hydrologic models is to compare model simulation/post-processing performance in a qualitative sense. Thus, the detail of uncalibrated model development is not the main focus in the study.

There is one thing should be aware of in the PRMS simulation of the Lehman

Creek watershed. According to Prudic, Sweetkind, Jackson, Dotson, Plume, Hatch, and Halford (2015), during 2011 summer, the peak flow observation was under-recorded due to the large overland flow bypassing the gauge station. The actual peak flow rate should be as great as the peak flow rate in 2005, since the precipitation in these two years are comparable. However, the current calibrated PRMS model was not able to capture the actual high peak flow but the observed peak flow. Nevertheless, this results in a better fitness with observations instead of over estimation and making the fitness evaluation in PRMS model and post-processor more comparable.

4.2.6 Conclusion

In this section, a post-processor framework is proposed to improve the accuracy of hydrologic models with a window size selection method embedded to solve the non-stationary concern in hydrologic data. The proposed post-processor framework leverages machine learning approaches to characterize the role that the model inputs play in the model prediction errors so as to improve hydrologic model prediction results. The proposed window size selection method enhances the performance of the proposed framework when dealing with non-stationary data. The results of two different hydrologic models show that the accuracy of calibrated hydrologic models can be further improved; without efforts of the calibration, the results of uncalibrated hydrologic models using the proposed framework can be as accurate as the calibrated ones by leveraging the proposed framework, which means that our proposed methods are able to ease the traditional complex and time-consuming model calibration step.

4.3 Service 3: Nitrate Prediction Model

4.3.1 Overview

In this section, the nitrate prediction service is introduced. The key of this service is to use an accurate nitrate prediction model. Instead of leveraging existing nitrate

prediction models, we built a model by ourselves. This is because most nitrate prediction models are not very accurate based on our survey. Our collaborators from University of Nevada, Reno Hydrology Department built the initial model and we tuned the model parameters using genetic algorithms (GAs) [136].

4.3.2 Introduction

Nitrogen is a major nutrient that is essential for plant and animal growth. Although nitrogen is often a limiting nutrient, an abundance of inorganic species such as nitrate (NO_3) causes excessive growth among primary producers that often results in low levels of dissolved oxygen, fish kills, toxic algal blooms, and toxicity to aquatic organisms. [11, 88, 101]. Nutrient enrichment from nonpoint sources, such as fertilizer runoff, was identified as one of the largest impairments to surface water quality in the United States [36]. Daniel et al. suggest 70% of the fertilizers and feed applied to farms in the US are either lost to soil storage or transported to surface or groundwater [24]. Additionally, sewage effluent, burning of fossil fuels, energy production, and industrial activities also can lead to increased nitrate in the environment [5, 29, 106]. Nitrate is mobile in groundwater, and drinking nitrate contaminated water has been linked to infant methemoglobinemia (MetHb), among other human health issues [54, 63]. For these reasons, it is critical to measure and predict nitrate loads in rivers to better inform governmental and non-governmental agencies such as policy makers, environmental groups, and water suppliers.

There are different models utilized by hydrologists to determine nitrate content in water. These prediction models use other constituents present in water to predict the NO_3 content. The models mainly differ from each other in the number of constituents they need to make the predictions. In this paper, an improved nonlinear prediction model is developed by using a GA to predict the NO_3 content in water. The proposed model uses six constituents - organic nitrogen, orthophosphate, pH level, dissolved oxygen, temperature, and discharge to make the predictions on NO_3 content. The model contains 12 parameters that need to be calibrated effectively to improve the

accuracy of the predictions. Due to the nonlinearity of the model, the calibration of the model parameters is highly complex.

GA is powerful adaptive search techniques that use the concepts of natural selection to mimic the process of biological evolution to efficiently solve optimization problems [46]. When searching over a large multidimensional state space, GAs can outperform conventional search techniques due to simplicity, effectiveness, versatility, and robustness of GA [75]. For the calibration of the proposed model, a GA was found to be a feasible approach due to the following reasons: 1) the nonlinearity of the model, 2) presence of many parameters, 3) vast search space, and 4) high possibility of convergence towards optimal values for the parameters.

To evaluate the performance of the proposed approach we compared results from the GA-tuned model with the prevalent environmental tool, LOADEST applied for predicting nitrate loads in Hellbranch Run, which is a protected stream in central Ohio [124]. LOADEST is a software tool offered by the United States Geological Survey (USGS), and is widely used by hydrologists to estimate the constituent loads [M/T] [102]. The results of the GA were compared with four other regression methods: generalized linear regression, gradient boosted tree regression [41], random forest regression [72], and decision tree regression [40].

4.3.3 Prior Work

Much research has been done to study how effectively nitrate content in water can be predicted. Almasri et al. proposed the use of Modular Neural Networks (MNN) to predict the nitrate distribution in water [2]. The MNN-based approach was simple and economical. Although it could efficiently predict the distribution of nitrate concentration in water, its performance deteriorated drastically with noisy data due to high sensitivity to errors in the input data. Yesilnacar et al. used an Artificial Neural Networks (ANN)-based approach to predict the nitrate concentration in 24 observation wells in the Harran Plain, located in Turkey [138]. The developed model was cost-effective and gave a satisfactory fit to the experimentally obtained nitrate data.

Poor et al. proposed the use of tree analysis to improve the predictions of low-flow nitrate in Willamette River [98]. Although regression tree analysis greatly improved the predictability compared to multiple linear regression, the results show that this approach was highly inaccurate with smaller datasets and shows an inconsistent relationship between nitrate and some other parameters. Arabgol et al. proposed the use of Support Vector Machine (SVM) models in predicting the nitrate concentration in ground water resources [4]. SVM models were fast, reliable and cost-effective. The prediction accuracy of SVM was better than ANN. However, the prediction accuracy of SVM models with noisy data has not yet been proven. To acquire more accurate results, we proposed a numerical equation and tune the parameters with GA in this paper.

The four objectives of our study are: 1) Use GA to optimize the parameters of the nonlinear NO_3 prediction model 2) Evaluate the performance of GA with the results from LOADEST software. 3) Compare the performance of GA approach with four other machine learning techniques such as gradient boosted tree regression, random forest regression, decision tree regression and generalized linear regression. 4) Deal with missing fields in the dataset and evaluate how it affects the prediction capability of the model.

Our results show that our GA-based approach produced nitrate level predictions that were closer to the observed values than LOADEST and statistically significantly $(t - test, p = 8.19 * 10^{-47})$ different from LOADEST predications. Furthermore, the GA-tuned model performed better than the four other estimation methods described earlier. Therefore, using our proposed approach hydrologists can make more accurate predictions of nitrate content in water.

4.3.4 Methodology

We used a GA with rank based selection. "Two-point crossover" was used as the crossover technique, and "bit-wise mutation" was used as the mutation strategy. We also compared the performance of rank based selection strategy with other prominent

selection techniques used in the GA. The results of the comparison between different selection strategies are given in Section 4.3.5. The proposed NO_3 prediction model uses six constituents present in water to make predictions on the NO_3 level. Organic nitrogen, orthophosphate, pH level, dissolved oxygen, temperature, and flow rate (discharge) are the six constituents required by the model. The model maintains a nonlinear quadratic relationship with the various constituents and contains 12 parameters whose value lies in [-10.24,10.24]. The proposed model is represented as below:

$$\Psi = a_0 + a_1 * LnQ + a_2 * (LnQ)^2 + a_3 * sin(2 * dtime) + a_4 * cos(2 * dtime) + a_5 * dtime + a_6 * dtime^2 + a_7 * DO + a_8 * T + a_9 * ON + a_{10} * OP + a_{11} * TP \tag{4.18}$$

where Q denotes discharge; DO denotes dissolved oxygen; T denotes temperature; ON denotes organic nitrogen; TP denotes pH level; OP denotes orthophosphate; *dtime* (decimal time - center of decimal time); Ψ denotes nitrate load at *dtime*. Decimal time (calculated as decimal years in LOADEST) is an important explanatory variable for load modeling. In the LOADEST model, the third and fourth terms represent a first-order Fourier series in *dtime* to capture seasonal variations and the fifth and sixth terms in *dtime* are meant to capture linear and quadratic temporal trends [11]. Decimal time is the decimal equivalent of the date and time. To convert the date and time to its decimal equivalent, one year is represented as one revolution around the unit circle. Therefore, the values within a year are converted to their respective values between 0 and 2π. Center of decimal time is the average of all the decimal equivalents for the entire time period.

The objective function in the GA for the estimation of optimal parameters (a_0 to a_{11}) in the proposed nonlinear NO_3 prediction model is taken as minimizing the mean square root of sum of squares between the observed and predicted nitrate content in

water and is given by:

$$minRMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(P_i - A_i)^2} \tag{4.19}$$

where P_i and A_i represent the predicted and observed values of nitrate content respectively and n is the total number of observations. In GAs, the fitness function often defined for the canonical GA. To meet this requirement, the fitness function is represented as the reciprocal of the objective function. Therefore, the RMSE values will be minimized on maximizing the fitness function.

A string length of 11 was chosen to represent each variable encoded as binary digits. This is because the value of each variable lies in [-10.24, 10.24] and the precision is 0.01, which means there are 211 possible numbers in total. Since there were 12 variables, from a0 to a11, the total chromosome length of the individual was 132. The population size was chosen as 200 and the number of generations was fixed to 500. A mutation probability (P_m) of .01 and a crossover probability (P_c) of 0.9 were used in the GA to estimate the optimal values of the variables in the proposed nonlinear nitrate prediction model. After many experiments were done, we found these chosen GA parameters guarantee good results for this problem. The GA was ran 30 times with different random seeds. The best (minimum) RMSE of the 30 runs was selected as the solution to the problem.

There are some existing hydrologic tools or libraries that can predict NO_3 content in water based on the available constituent details. However none of them can guarantee accurate predictions. To evaluate the performance of the model, several quality metrics were used. RMSE, percent bias (PBIAS), and Nash-Sutcliffe efficiency (NSE) are some of the popular quantitative statistics used to perform the statistical evaluation of model accuracy. These parameters have been introduced in Section 4.2.

The dataset used in the study was collected from the United States Survey for Big Darby Creek Watershed in Ohio. The dataset comprised 435 water samples monitored during the period of 20 years between December 1, 1996 and August 25, 2016. There

were two challenges faced with this dataset. 1) cleaning the data and 2) filling the missing fields in the dataset. The dataset contains many constituent details that are not relevant for the proposed model. Finding the required constituents information and removal of unwanted constituent details from this USGS dataset was the first challenge. Out of the 435 water samples, only 140 samples had the measurements for all the constituents required by the model. For the remaining 235 samples, at least one of the constituent readings were missing. Missing fields in dataset is a common issue with environment data and researchers have employed various strategies to deal with the problem; artificial neural networks, support vector machines, interpolation or regression techniques, Bayesian approaches, and multiple imputations are a few of them. However, for this experimental study, simple linear regression was used to fill the missing fields. When using a deterministic linear regression approach it is easy to pinpoint whether possible issues lie in the data or the GA approach. We chose to include incomplete samples because we wanted to test if the linear regression technique works well for this problem. It is very common that there are some missing data in the real world environmental observations. Thus, these incomplete samples can be very important to studies.

The proposed approach was compared with the results from LOADEST, which is a prominent load estimation tool used by hydrologists. On specifying the input constituents, LOADEST performs its own calibration and estimation procedures using several statistical estimation methods and forms a regression model to predict the estimated constituent. Besides LOADEST, the results of the GA were compared with four other machine learning techniques: generalized linear regression, gradient boosted tree regression, decision tree regression, and random forest regression. Linear regression is a popular modeling technique used to estimate values for an unknown parameter [67]. The data for the known variables (features) are used to map a linear relationship with the parameter to be estimated. Linear regression is not suited for problems that maintain a nonlinear relationship between predicted parameter and features. Generalized linear regression is more accurate than linear regression, as it

allows transform predictors and interactions [90]. Decision tree regression uses decision tree as the predictive model and is widely used in data classification research [93]. It breaks down data into smaller datasets by incrementally developing an associated decision tree. Random forest regression is similar to decision tree regression, where random forest regression uses multiple decision trees to improve the regression results [72]. Gradient boosted tree regression is another machine learning technique that follows a stage-wise fashion to build an additive prediction model using the combination of other predictive models [41]. It is a popular technique that is used by Google and Yahoo for page ranking in search engine.

In the next section, we have done several experiments to compare the performances of the introduced six methods and analyze the results.

4.3.5 Results and Analysis

We used three performance measurements (RMSE, PBIAS, and NSE) to compare the six approaches. The six prediction methods are a generalized linear regression, gradient boosted tree regression, random forest regression, decision tree regression, GA, and LOADEST. LOADEST contains many methods and we chose the best results to compare with other methods. We tested these methods with 30-fold cross-validation and used 70% of the data for training and used 30% of the data for testing. Furthermore, the models have been run 30 times and the average values are used as the final result. Some interesting phenomena are found from these results.

Table 4.5 displays our results from all six methods. From the table, it is clear that the random forest regression has the lowest RMSE, which means this regression method prediction is closest to the observed values. GA has the best PBIAS, which shows that GA overestimates or underestimates at least compared to other methods. The gradient boosted tree regression has the best NSE. This means the method is more efficient and its prediction fits 1:1 line with the observed values. The GA is not as good as other machine learning methods, but it is slightly better than LOADEST based on RMSE and much better based on PBIAS and NSE. However, this does not

Table 4.5: Results of Different Techniques

Name of the method	**RMSE**	**PBIAS**	**NSE**
Generalized linear regression	2.020	2.356	0.275
Gradient boosted tree regression	1.961	1.974	0.730
Random forest regression	1.894	1.808	0.587
Decision tree regression	1.953	2.250	0.551
GA	2.036	-0.801	0.376
LOADEST	2.890	59.958	-0.474

Table 4.6: Comparison of Different Selection Strategies

Selection strategy	**RMSE(Average of 30 runs)**
Fitness proportionate selection	2.077
Truncation selection	2.925
Tournament selection	2.191
Rank based selection	2.036
Elitism	2.100

mean GA is less useful. The best result of the 30 GA model runs RMSE is only 2.036, which is better than most other methods. This means GA can obtain good results but it is not very robust.

Different selection strategies were tried to improve the execution of the genetic algorithm. We compared the performance of rank based selection with other popular selection strategies such as truncation selection, fitness proportionate selection, tournament selection, and elitism. For truncation selection, the candidate individuals were sorted in the decreasing order of their fitness value, and the individuals were picked from the first half of the population to generate offspring. To perform tournament selection, a set of 40 individuals were randomly selected from the population and the individual with the best fitness was chosen. To implement elitism, the fittest 25 individuals in the population were copied to the next generation and thus ensure that the best chromosomes are not being lost during the evolution process. Table 4.6 shows the results obtained with different selection strategies. Among the five selection strategies implemented, rank based selection performed the best, whereas truncation selection was the worst.

Different years have different characteristics. Some years are very dry (droughts)

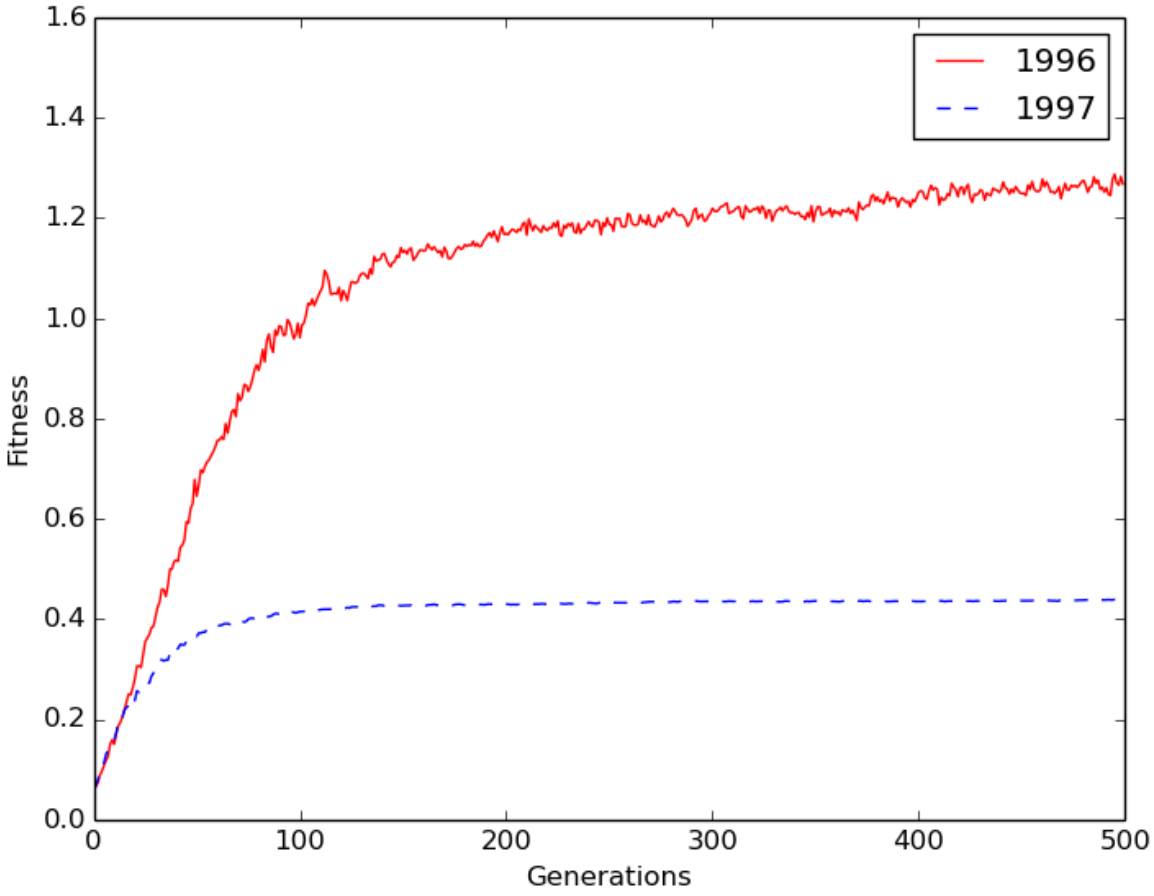

Figure 4.24: Comparison of the best (year 1996) and worst (year 1997) results.

Table 4.7: Results of T-TEST

Methods Names	**P-value**	**T-value**
GA vs LOADEST	$8.19*10^{-47}$	1.648
GA vs random forest	0.026	1.648
GA vs generalized linear regression	0.102	1.648
GA vs decision tree	0.274	1.648
GA vs gradient boosted tree	0.486	1.648

Table 4.8: Means AND Variance

Methods Names	**Mean**	**Variance**
GA	2.458	4.250
LOADEST	4.082	11.585
Random forest	2.621	2.492
Generalized linear regression	2.545	1.279
Decision tree	2.515	3.536
Gradient boosted tree	2.461	4.317

and some years are very wet (floods). Even though the year information is built in dtime in our fitness, the GA model performs different year by year. Figure 4.24 shows the best and worst results of year-wise comparison. This means that the year information is not well-built in the current fitness function. We have run the GA model with and without year information. The result shows that the year information can improve the results (with year RMSE is 2.035 and without year RMSE is 2.145).

To prove that GA method is significantly different from the LOADEST estimations, the one-tailed T-Test has been done and the p-value obtained was 8.19*10-47, which shows that the predictions of these two methods are significantly different. Also from Table 4.7, it is clear that even though some methods, such as gradient boosted tree, have better RMSE and NSE, their T-Test values show that the predictions of these methods were not significantly different from GA predictions. Therefore, it is wrong to state that the performance of those techniques was superior to GA approach. Table 4.8 contains more statistics results of the introduced methods.

Some other fitness functions were also tried. However, most of them did not perform very well. For example, one of the fitness function is created with the assumptions that the nitrate has non-linear quadratic relations with all the parameters.

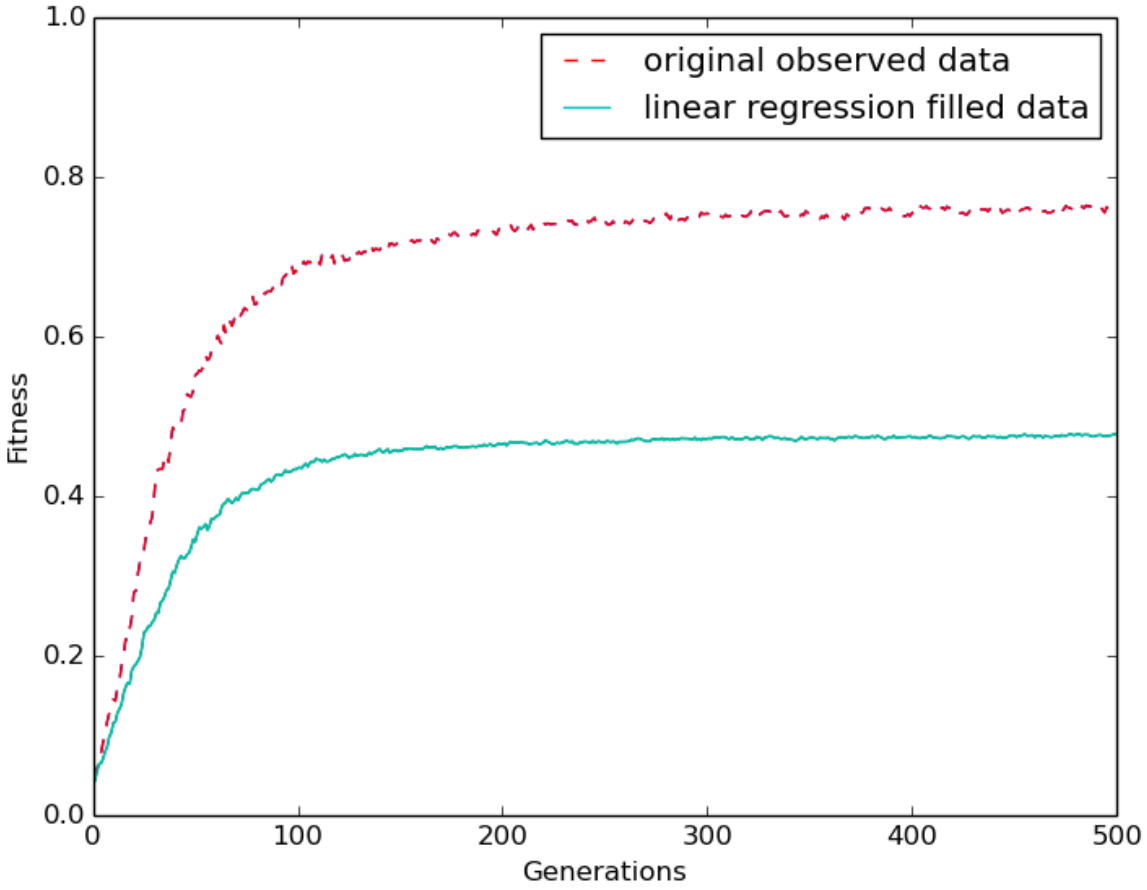

Figure 4.25: Comparison of the results using original and filled data.

The results show that this assumption cannot guarantee good results for all the occasions and, in most cases, it can lead to a worse result than the previously introduced fitness functions. This experiment shows that professional knowledge of the target problem is very necessary to build a good fitness function.

For all the experiments mentioned above, we used the original observed dataset from USGS and filled the missing data gaps using linear regression. To test the accuracy of data filling original dataset and the modified dataset. From Figure 4.25, it is evident that the performance of GA deteriorated with the filled data values. Thus, we could conclude that linear regression is not a reliable technique for filling missing data fields in environmental datasets and could be replaced with other efficient data filling techniques.

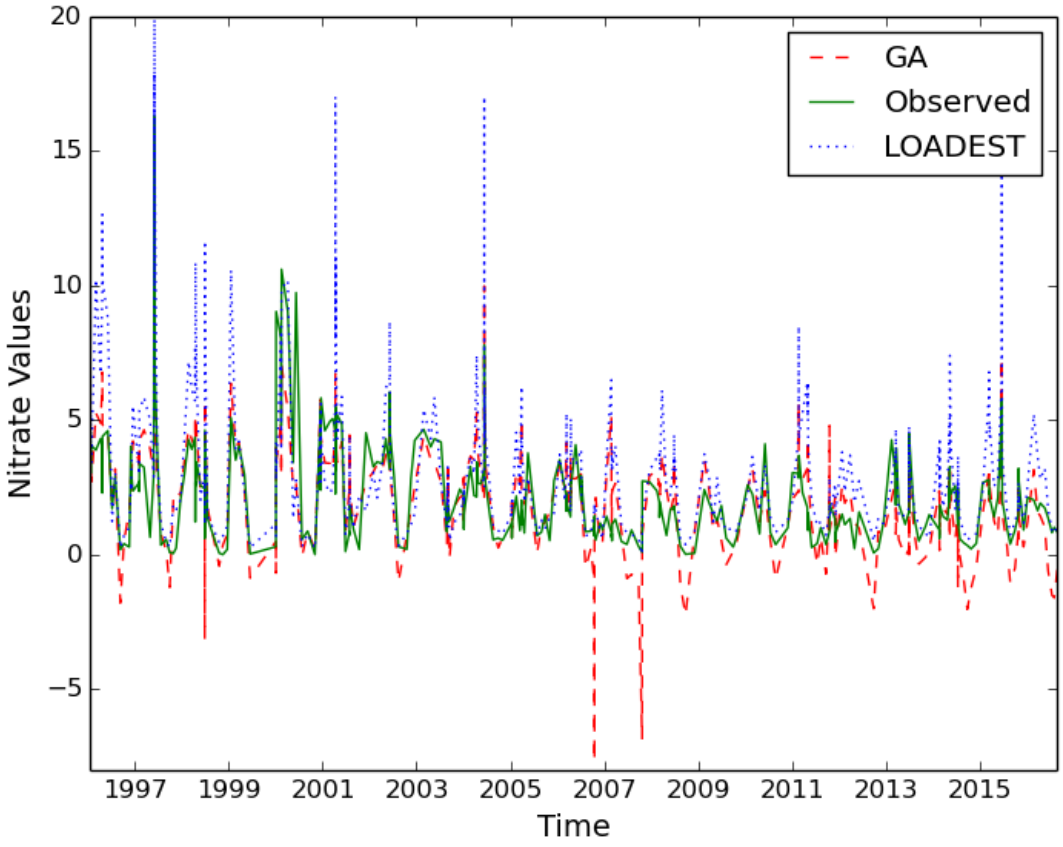

Figure 4.26: Comparison of the results using original and filled data.

Table 4.9: Parameter Estimates of Big Darby Creek Watershed USGS Dataset

Parameter	**GA**	**LOADEST**
a0	1.26	2.625
a1	-6.06	0.263
a2	3.15	-0.003
a3	0.72	0.695
a4	0.43	0.333
a5	-0.17	-0.036
a6	2.95	0.004
a7	-0.02	-0.151
a8	0	-0.100
a9	1.28	-0.034
a10	6.61	5.363
a11	-5.55	-1.714

Table 4.9 shows the parameter estimations of our GA model and LOADEST model for the Big Darby Creek Watershed. Figure 4.26 compares the LOADEST predications, GA predictions, and the observed data. Only two methods results are shown in the graph because it is clearer than crowding all the results in a single graph. From the comparisons, we can tell that GA predictions are closer to the observed than the LOADEST predications. However, GA method generates some predictions below zero and NO_3 values cannot be negative. In the future, we plan to set some extra rules to make predications more accurate, such as turn negative value into zero to make the results more accurate.

4.3.6 Conclusion

In this section, we have proposed a GA method to calibrate the parameters of an improved nonlinear hydrologic nitration prediction model. From RMSE, PBIAS, and NSE, the GA method is better than LOADEST. GA predictions are significantly different from the LOADEST predictions based on T-test p-value (< 0.001). We have also used some other popular machine learning techniques (generalized linear regression, gradient boosted tree regression, random forest regression, and decision tree regression) to predict nitrate content with the same dataset. The results show GA has the best PBIAS value than other methods. This means GA does least over-estimates and underestimate compared with other six introduced methods. GAs best results are as good as random forest regression predications based on RMSE.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this dissertation, we have proposed a new framework for elastic server optimization that changes a server size semi-automatically. It contains a queue model to estimate queue lengths and waiting time. The model includes a schema for elastic scale hybrid servers that are a combination of owned and rented servers. The experimental results showed that our proposed approach was more stable than the traditional FIFO queue model in regards to the average waiting time and the expected queue length. We also proposed an improved compression and decompression algorithm, named GFC, to increase data transportation speeds. The algorithm is used to improve the data transportation between different nodes. Based on our experiment results, the improved GFC is 8.7 times faster than the original algorithm. Three services were also presented to prove the capabilities of the proposed framework. These three services are: large dataset visualization and interaction, model accuracy enhancement, and a nitrate prediction model. We proposed new solutions for each of these services. According to the experimental results, a large dataset can be visualized and interacted with a much shorter delay; a hydrologic model accuracy can be improved and the calibration step can be replaced with our proposed new methods; nitrate prediction results can be more accurate than those obtained using LOADEST, one of the most prevalent prediction tools currently available.

5.2 Future Work

The current prototype system can shut down the rented servers when they finish their jobs to avoid unnecessary budget spending. However, the project manager still needs to change the budget based on the user's feedback. In the future, we plan to use machine learning techniques to improve this part. We will also improve our current queue model by specifying the virtual machine starting time, which is very important in the real world.

Furthermore, we plan to use Hadoop with more nodes to improve the new workflow performance presented in Service 1. Also, we would like to use GPUs in each of the nodes to further accelerate the server's performance. As most of our collaborators in the past have been environmental scientists, they often utilize both CSV and NetCDF files for most environmental readings. Therefore, we should improve our system to support more scientific data file types, such as NetCDF.

Two case studies are described in Service 2 and we will examine the framework with other models and study fields. Also, it is interesting to study the peak values and better prediction algorithms for them in the future. The key idea is to separate the peak values and then apply another model for them. Extreme-value-theory-based methods are widely used to separate the peak values. However, the results are not very accurate based on our experiments. Thus, we would like to develop better algorithms in the future to address this problem.

For the nitrate model parameter calibrations introduced in Service 3, GA is not perfect. From the experiments, it is clear that the year information is not well-built into the model. In the future, we plan to modify our fitness function to improve the "dtime". Also, based on the results obtained, it is clear that data filling using linear regression can make the predictions less accurate. This means the linear regression method is not very good for this problem. Therefore, we plan to replace it with other methods, such as neural networks and compare their performance. Last but not least, we want to add some other useful limitations or customize some conditions to obtain

better results. For example, the predictions should always be positive, as they express stream values.

Bibliography

[1] F. Aisopos, K. Tserpes, and T. Varvarigou. Resource management in software as a service using the knapsack problem model. *International Journal of Production Economics*, 142(2):465–477, 2011.

[2] M. N. Almasri and J. J. Kaluarachchi. Modular neural networks to predict the nitrate distribution in ground water using the on-ground nitrogen loading and recharge data. *Environmental Modelling & Software*, 20(7):851–871, 2005.

[3] K. M. Andreadis and D. P. Lettenmaier. Assimilating remotely sensed snow observations into a macroscale hydrology model. *Advances in Water Resources*, 29(6):872–886, 2006.

[4] R. Arabgol, M. Sartaj, and K. Asghari. Predicting nitrate concentration and its spatial distribution in groundwater resources using support vector machines (svms) model. *Environmental Modeling & Assessment*, 21(1):71–82, 2016.

[5] B. Arheimer and R. Liden. Nitrogen and phosphorus concentrations from agricultural catchmentsinfluence of spatial and temporal variables. *Journal of Hydrology*, 227(1-4):140–159, 2000.

[6] Microsoft Azure. Microsoft azure: cloud computing platform & service. URL: `https://azure.microsoft.com/en-us/?wt.mc_id=AID529439_SEM_YlMuLt4h&gclid=CjwKEAiA8JbEBRCz2szzhqrx7H8SJAC6FjXXjLdPe_dzC1-AHXipf-Dd0_gLs2uUd_EIHVpqEw5msxoCdRLw_wcB`. [Accessed on 8 April 2018].

[7] D. Basak, S. Pal, and D. C. Patranabis. Support vector regression. *Neural Information Processing-Letters and Reviews*, 11(10):203–224, 2007.

[8] T. H. Bennett. Development and application of a continuous soil moisture accounting algorithm for the hydrologic engineering center hydrologic modeling system (hec-hms). *Thesis (M.S.), University of California, Davis*, 1998.

[9] T. H. Bennett and J. C. Peters. Continuous soil moisture accounting in the hydrologic engineering center hydrologic modeling system (hec-hms). *World Environmental and Water Resources Congress.*, 8806(1):110, 2004.

[10] A. Bifet and R. Gavalda. Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, pages 443–448. SIAM, 2007.

[11] B. J. F. Biggs. Eutrophication of streams and rivers: dissolved nutrient-chlorophyll relationships for benthic algae. *Journal of the North American Benthological Society*, 19(1):17–31, 2000.

[12] A. Bouchachia. Fuzzy classification in dynamic environments. *Soft Computing*, 15(5):1009–1022, 2011.

[13] J.D. Brown and D.J. Seo. A nonparametric postprocessor for bias correction of hydrometeorological and hydrologic ensemble forecasts. *Journal of Hydrometeorology*, 11(3):642–665, 2010.

[14] J.D. Brown and D.J. Seo. Evaluation of a nonparametric postprocessor for bias correction and uncertainty estimation of hydrologic predictions. *Hydrological Processes*, 27(1):83–105, 2013.

[15] M. Burscher. Martin burscher/fpdouble. URL: `http://cs.txstate.edu/~burtscher/research/datasets/FPdouble/`. [Accessed on 8 April 2018].

[16] R. N. Calheiros, R. Ranjan, and R. Buyya. Virtual machine provisioning based on analytical performance and qos in cloud computing environments. In *2011 international conference on Parallel processing (ICPP)*, pages 295–304. IEEE, 2011.

[17] R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised learning algorithms using different performance metrics. In *Proceedings of 23 rd International Conference on Machine Learning (ICML06)*, pages 161–168, 2005.

[18] Southern Regional Climate Center. Southern regional climate center. 1998. URL: `http://www.srcc.lsu.edu/srcc.html/`. [Accessed on 8 April 2018].

[19] C. Chen, L. Fenstermaker, H. Stephen, and S. Ahmad. Distributed hydrological modeling for a snow dominant watershed using a precipitation and runoff modeling system. *World Environmental and Water Resources Congress*:2527–2536, 2015.

[20] C. Chen, A. Kalra, and S. Ahmad. A conceptualized groundwater flow model development for integration with surface hydrology model. *World Environmental and Water Resources Congress*:175–187, 2017.

[21] R. L. Cloud, M. L. Curry, H. L. Ward, A. Skjellum, and P. Bangalore. Accelerating lossless data compression with gpus. *arXiv preprint arXiv:1107.1525*, 2011.

[22] Y. Collet. Lz4-extremely fast compression algorithm. URL: `https://code.google.com/p/lz4/`. [Accessed on 8 April 2018].

[23] P. Cudré-Mauroux, H. Kimura, K.-T. Lim, J. Rogers, R., Simakov, E. Soroush, P. Velikhov, D. L. Wang, M. Balazinska, J. Becla, D. DeWitt, B. Heath, D. Maier, S. Madden, J. Patel, M. Stonebraker, and S. Zdonik. A demonstration of scidb: a science-oriented dbms. *Proceedings of the VLDB Endowment*, 2(2):1534–1537, 2009.

[24] T. C. Daniel, A. N. Sharpley, and J. L. Lemunyon. Agricultural phosphorus and eutrophication: a symposium overview. *Journal of Environmental Quality*, 27(2):251–257, 1998.

[25] Docker. Docker-build,ship, and run any app, anywhere. URL: `https://www.docker.com/`. [Accessed on 8 April 2018].

[26] Docker. Docker swarm — docker. URL: `https://www.docker.com/products/docker-swarm`. [Accessed on 8 April 2018].

[27] P. Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.

[28] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80. ACM, 2000.

[29] C. T. Driscoll, G. B. Lawrence, A. J. Bulger, T. J. Butler, C. S. Cronan, C. Eagar, K. F. Lambert, G. E. Likens, J. L. Stoddard, and K. C. Weathers. Acidic deposition in the northeastern united states: sources and inputs, ecosystem effects, and management strategies: the effects of acidic deposition in the northeastern united states include the acidification of soil and water, which stresses terrestrial and aquatic biota. *AIBS Bulletin*, 51(3):180–198, 2001.

[30] Q. Duan, J. Schaake, V. Andreassian, S. Franks, G. Goteti, H.V. Gupta, Y.M. Gusev, F. Habets, A. Hall, L. Hay, and T. Hogue. Model parameter estimation experiment (mopex): an overview of science strategy and major results from the second and third workshops. *Journal of Hydrology*, 320(1):3–17, 2006.

[31] Q. Duan, S. Sorooshian, and V. Gupta. Effective and efficient global optimization for conceptual rainfallrunoff models. *Water Resources Research*, 28(4):1015–1031, 1992.

[32] Q. Duan, S. Sorooshian, and V.K. Gupta. Optimal use of the sce-ua global optimization method for calibrating watershed models. *Journal of Hydrology*, 158(3):265–284, 1994.

[33] Dygraphs. Dygraphs.com. URL: `http://dygraphs.com/`. [Accessed on 8 April 2018].

[34] A. Eirola. Lossless data compression on GPGPU architectures. *arXiv preprint arXiv:1109.2348*, 2011.

[35] EPA Environmental Protection Agency. Environmental modeling — US EPA, 2017. URL: `https://www.epa.gov/modeling`. [Accessed on 8 April 2018].

[36] EPA. National water quality inventory: Report to Congress. Technical report, Washington, DC: Environmental Protection Agency, 2009.

[37] A. M. Fan and V. E. Steinberg. Health implications of nitrate and nitrite in drinking water: an update on methemoglobinemia occurrence and reproductive and developmental toxicity. *Regulatory Toxicology and Pharmacology*, 23(1):35–43, 1996.

[38] B. Figuerol, A. Carles, and M. R. Gavaldà. Adaptive parameter-free learning from evolving data streams, 2009.

[39] Flask. Welcome — FLASK (a Python microframework). URL: `http://flask.pocoo.org/`. [Accessed on 8 April 2018].

[40] M. A. Friedl and C. E. Brodley. Decision tree classification of land cover from remotely sensed data. *Remote Sensing of Environment*, 61(3):399–409, 1997.

[41] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*:1189–1232, 2001.

[42] J. H. Friedman. On bias, variance, 0/1loss, and the curse-of-dimensionality. *Data mining and knowledge discovery.*, 1(1):55–77, 1997.

[43] J. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with drift detection. In *Brazilian Symposium on Artificial Intelligence*, pages 286–295. Springer, 2004.

[44] Georgia. Agricultural research service hydrology laboratory. 2007. URL: `https://hrsl.ba.ars.usda.gov/wdc/ga.htm`.

[45] B. Glahn, M. Peroutka, J. Wiedenfeld, J. Wagner, G. Zylstra, B. Schuknecht, and B. Jackson. Mos uncertainty estimates in an ensemble framework. *Monthly Weather Review*, 137(1):246–268, 2009.

[46] D. E. Goldberg and J. H. Holland. Genetic algorithms and machine learning. *Machine Learning*, 3(2):95–99, 1988.

[47] T. Hashino, A. A. Bradley, and S. S. Schwartz. Evaluation of bias-correction methods for ensemble streamflow volume forecasts. *Hydrology and Earth System Sciences Discussions*, 3:561594, 2006.

[48] T. Hastie, R. Tibshirani, and J. H. Friedman. Boosting and additive trees. *The Elements of Statistical Learning (2nd ed.) New York: Springer.*:337384, 2009.

[49] L. E. Hay and M. Umemoto. Multiple-objective stepwise calibration using luca, open-file report. *US Geological Survey.*:20061323, 2007.

[50] P. Holub, M. Šrom, M. Pulec, J. Matela, and M. Jirman. Gpu-accelerated dxt and JPEG compression schemes for low-latency network transmissions of hd, 2k, and 4k video. *Future Generation Computer Systems*, 29(8):1991–2006, 2013.

[51] M. Hossain, H. Munoz, R. Wu, E. Fritzinger, S. M. Dascalu, and F. C. Harris. Becoming dataone tier-4 member node: Steps taken by the Nevada Research Data Center. In *Optimization of Electrical and Electronic Equipment (OPTIM) & 2017 Intl Aegean Conference on Electrical Machines and Power Electronics (ACEMP), 2017 International Conference on*, pages 1089–1094. IEEE, 2017.

[52] M. Hossain, R. Wu, J. T. Painumkal, M. Kettouch, C. Luca, S. M. Dascalu, and F. Harris. Web-service framework for environmental models. In *Internet Technologies and Applications (ITA), 2017*, pages 104–109. IEEE, 2017.

[53] N. E. Huang, Z. Shen, S. R. Long, M. C. Wu, H. H. Shih, Q. Zheng, N. Yen, C. C. Tung, and H. H. Liu. The empirical mode decomposition and the hilbert spectrum for nonlinear and non-stationary time series analysis. In *Proceedings of the Royal Society of London: Mathematical, Physical and Engineering Sciences*, volume 454 of number 1971, pages 903–995. The Royal Society, 1998.

[54] P. F. Hudak. Regional trends in nitrate content of Texas groundwater. *Journal of Hydrology*, 228(1-2):37–47, 2000.

[55] J. D. Hunter. A 2d graphics environment. *Computing in Science and Engineering*, 9(3):90–95, 2007.

[56] R. J. Hyndman and A. B. Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4):679–688, 2006.

[57] Google Inc. Statisticsyoutube. URL: `https://www.youtube.com/yt/press/statistics.html`. [Accessed on 8 April 2018].

[58] M. Jaiswal. Accelerating enhanced boyer-moore string matching algorithm on multicore gpu for network security. *International Journal of Computer Applications*, 97(1), 2014.

[59] W. Kahan. Ieee standard 754 for binary floating-point arithmetic. *Lecture Notes on the Status of IEEE*, 754(94720-1776):11, 1996.

[60] S. Karlin and J. McGregor. Many server queueing processes with poisson input and exponential service times. *Pacific J. Math*, 8(1):87–118, 1958.

[61] R. Klinkenberg and T. Joachims. Detecting concept drift with support vector machines. In *ICML*, pages 487–494, 2000.

[62] R. Klinkenberg and I. Renz. Adaptive information filtering: learning in the presence of concept drifts. *Learning for Text Categorization*:33–40, 1998.

[63] L. Knobeloch, B. Salna, A. Hogan, J. Postle, and H. Anderson. Blue babies and nitrate-contaminated well water. *Environmental Health Perspectives*, 108(7):675, 2000.

[64] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *IJCAI*, volume 14(2) of number 2, pages 1137–1145. Stanford, CA, 1995.

[65] R. Krzysztofowicz and C.J. Maranzano. Bayesian system for probabilistic stage transition forecasting. *Journal of Hydrology*, 299(1):15–44, 2004.

[66] Kubernetes. Kubernetes production grade container orchestration. URL: `https://kubernetes.io/`. [Accessed on 8 April 2018].

[67] M. H. Kutner, C. Nachtsheim, and J. Neter. *Applied linear regression models*. McGraw-Hill/Irwin, 2004.

[68] C. Lanquillon. *Enhancing text classification to improve information filtering*. PhD thesis, Otto-von-Guericke-Universität Magdeburg, Universitätsbibliothek, 2001.

[69] D. Le-Gall. MPEG: a video compression standard for multimedia applications. *Communications of the ACM*, 34(4):46–58, 1991.

[70] G. H. Leavesley, R. W. Lichty, B. M. Troutman, and L. G. Saindon. *Precipitation-runoff modeling system: User's manual*. USGS Washington, DC, 1983.

[71] Jo-J. Y. Lee S. and Y. Kim. Performance testing of web-based data visualization. In *2014 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 1648–1653. IEEE, 2014.

[72] A. Liaw and M. Wiener. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[73] Y. Liu, W. Wang, Y. Hu, and W. Cui. Improving the distributed hydrological model performance in upper huai river basin: using streamflow observations to update the basin states via the ensemble kalman filter. *Advances in Meteorology*, 61(5):1–14, 2016.

[74] S. Loinel. Does a lot of if else statements slow down the code? URL: `https://software.intel.com/en-us/forums/topic/283268`. [Accessed on 8 April 2018].

[75] C. B. Lucasius and G. Kateman. Understanding and using genetic algorithms part 1. concepts, properties and context. *Chemometrics and Intelligent Laboratory Systems*, 19(1):1–33, 1993.

[76] J. Luitjens and S. Rennich. CUDA warps and occupancy. *GPU Computing Webinar*, 11:2–19, 2011.

[77] S. Madadgar, H. Moradkhani, and D. Garen. Towards improved postprocessing of hydrologic forecast ensembles. *Hydrological Processes*, 28(1):104–122, 2014.

[78] J. Madhavan, S. R. Jeffery, S. Cohen, X. Dong, D. Ko, C. Yu, and A. Halevy. Web-scale data integration: you can only afford to pay as you go. In CIDR, 2007.

[79] D. Marks, J. Domingo, and J. Frew. Software tools for hydro-climatic modeling and analysis: image processing workbench, ars-usgs version 2t. *ARS Tech. Bull*, 99(1), 1999.

[80] S. L. Markstrom, R. G. Niswonger, R. S. Regan, D. E. Prudic, and P. M. Barlow. Gsflow coupled ground-water and surface-water flow model based on the integration of the precipitation-runoff modeling system (PRMS) and the modular ground-water flow model. *Water-Resources Investigations Report.*, 2005.

[81] S. L. Markstrom, R. S. Regan, L. E. Hay, R. J. Viger, R. M. Webb, R. A. Payn, and J. H. LaFontaine. Prms-iv, the precipitation-runoff modeling system, version 4. *US Geological Survey Techniques and Methods*:6–B7, 2015.

[82] S. L. Markstrom, S. Regan, L. E. Hay, R. J. Viger, R. M. T. Webb, R. A. Payn, and J. H. LaFontaine. *the Precipitation-Runoff Modeling System*. Version 4. U.S. Geological Survey Techniques and Methods, Book 6, Chap. B7, http://doi.org/http://dx.doi.org/10.3133/tm6B7. Clarendon Press, 2015, page 158.

[83] D. Marr. Hyper-threading technology in the netburst microarchitecture. *14th Hot Chips*, 2002.

[84] P. Mell and T. Grance. The NIST definition of cloud computing. *National Institute of Standards and Technology*, 6(50):53, 2009.

[85] D. Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2, 2014.

[86] Apache Mesos. Apache mesos. URL: `http://mesos.apache.org/`. [Accessed on 8 April 2018].

[87] A. Montanari and A. Brath. A stochastic approach for assessing the uncertainty of rainfallrunoff simulations. *Water Resources Research*, 40(1):W01106, 2004.

[88] R. B. Moore, C. M. Johnston, K. W. Robinson, and J. R. Deacon. Estimation of total nitrogen and phosphorus in new england streams using spatially referenced regression models. *US Geological Survey Scientific Investigations Report*, 5012:1–42, 2004.

[89] J. Nelder and R Wedderburn. Generalized linear models. *Journal of the Royal Statistical Society*, 135(3):370–384, 1972.

[90] John Ashworth Nelder and R Jacob Baker. *Generalized linear models*. Wiley Online Library, 1972.

[91] NuDoq. Nudoq cudafy.net. URL: `http://www.nudoq.org/#!/Packages/CUDAfy.NET/Cudafy.NET/IntegerIntrinsicsFunctions/M/clzll`. [Accessed on 8 April 2018].

[92] J. S. Oakland. *Statistical process control*. Routledge, 2007.

[93] C. Olaru and L. Wehenkel. A complete fuzzy decision tree technique. *Fuzzy Sets and Systems*, 138(2):221–254, 2003.

[94] M. A. O'Neil and M. Burtscher. Floating-point data compression at 75 gb/s on a gpu. In *Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units*, page 7. ACM, 2011.

[95] J. Painumkal, R. Wu, S. Dascalu, and H. Frederick. Self-managed elastic scale hybrid server using budget input and user feedback. *Workshop on Feedback Computing*, 2017.

[96] L. Palathingal, R. Wu, R. Belkhatir, S. M. Dascalu, and F. Harris. Data processing toolset for the virtual watershed. In *Collaboration Technologies and Systems (CTS), 2016 International Conference on*, pages 281–287. IEEE, 2016.

[97] F. Perez-Sorrosal, M. P. Martinez, R. Jimenez-Peris, and B. Kemme. Elastic sicache: consistent and scalable caching in multi-tier architectures. *International Journal of Production Economics*, 20(6):841–865, 2011.

[98] C. J. Poor and J. L. Ullman. Using regression tree analysis to improve predictions of low-flow nitrate and chloride in willamette river basin watersheds. *Environmental Management*, 46(5):771–780, 2010.

[99] D. E. Prudic, D. S. Sweetkind, T. L. Jackson, K. E. Dotson, R. W. Plume, C. E. Hatch, and K. J. Halford. Evaluating connection of aquifers to springs and streams. *Eastern Part of Great Basin National Park and Vicinity, Nevada 2015*.

[100] Python. Python data analysis librarypandas. URL: `http://pandas.pydata.org/`. [Accessed on 8 April 2018].

[101] A. C. Redfield. The biological control of chemical factors in the environment. *American Scientist*, 46(3):230A–221, 1958.

[102] R. L. Runkel, C. G. Crawford, and T. A. Cohn. Load Estimator (LOADEST): A FORTRAN program for estimating constituent loads in streams and rivers. Technical report, 2004.

[103] A. Safari and F. De Smedt. Improving the confidence in hydrologic model calibration and prediction by transformation of model residuals. *Journal of Hydrologic Engineering*, 20(9):04015001, 2015.

[104] W. A. Scharffenberg and M. J. Fleming. Hydrologic modeling system hec-hms: user's manual. *US Army Corps of Engineers, Hydrologic Engineering Center.*, 2006.

[105] F.C. Schweppe. *Uncertain dynamic systems.* Prentice-Hall, 1973, page 576.

[106] SP Seitzinger, GP Asner, CC Cleveland, PA Green, EA Holland, DM Karl, AF Michaels, J. H. Porter, A. R. Townsend, and C. J. Vorosmarty. Nitrogen cycles: past, present, and future. biogeochemistry 70: 153226gan hj, zak dr, hunter md (2013) chronic nitrogen deposition alters the structure and function of detrital food webs in a northern hardwood ecosystem. *Ecol Appl*, 23:13111321, 2004.

[107] D. J. Seo, H. D. Herr, and J. C. Schaake. A statistical post-processor for accounting of hydrologic uncertainty in short-range ensemble streamflow prediction. *Hydrology and Earth System Sciences Discussions*, 3(4):1987–2035, 2006.

[108] Amazon Web Service. Aws — auto scaling. URL: `https://aws.amazon.com/autoscaling/`. [Accessed on 8 April 2018].

[109] Amazon Web Services. Netflix case study. URL: `https://aws.amazon.com/solutions/case-studies/netflix/`. [Accessed on 8 February 2017].

[110] Expedia Web Services. Netflix case study. URL: `https://aws.amazon.com/solutions/case-studies/expedia/`. [Accessed on 8 February 2017].

[111] K. Sierra and B. Bates. *Head first java*. "O'Reilly Media, Inc.", 2005.

[112] B. E. Skahill, J. S. Baggett, S. Frankenstein, and C. W. Downer. More efficient pest compatible model independent model calibration. *Environmental Modelling & Software*, 24(4):517–529, 2009.

[113] A. G. Slater and M. P. Clark. Snow data assimilation via an ensemble kalman filter. *Journal of Hydrometeorology*, 7(3):478–493, 2006.

[114] Apache Spark. Spark programming guide - spark 2.2.0 document, 2017. URL: `https://spark.apache.org/docs/latest/rdd-programming-guide.html#resilient-distributed-datasets-rdds`. [Accessed on 9 April 2018].

[115] A. K. Srivastava, M. Rajeevan, and S. R. Kshirsagar. Development of a high resolution daily gridded temperature data set (19692005) for the indian region. *Atmospheric Science Letters*, 10(4):249–254, 2009.

[116] M. Stonebraker, J. Becla, D. J. DeWitt, K. Lim, D. Maier, O. Ratzesberger, and S. B. Zdonik. Requirements for science data bases and SciDB. In *CIDR*, volume 7, pages 173–184, 2009.

[117] Study of Lehman Creek watersheds hydrologic response to climate change using downscaled cmip5 projections. *World Environmental and Water Resources Congress*:508517, 2016.

[118] R.S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[119] Stanford University. Fortran 77 tutorial. URL: `https://web.stanford.edu/class/me200c/tutorial_77/`. [Accessed on 8 April 2018].

[120] USDA. Agricultural research service hydrology laboratory. 1997. URL: `http://hydrolab.arsusda.gov/`.

[121] USGS. Usgs.gov — science for a changing world, 2017. URL: `https://www.usgs.gov/`. [Accessed on 8 April 2018].

[122] P. Vagata. and K. Wilfong. Scaling the facebook data warehouse to 300 pb. URL: `https://code.facebook.com/posts/229861827208629/scaling-the-facebook-data-warehouse-to-300-pb/`. [Accessed on 8 April 2018].

[123] J. M. Volk. Potential effects of a warming climate on water resources within the lehman and baker creek drainages. *Great Basin National Park, Nevada.*, 2014.

[124] J. M. Volk. Spatial and temporal variations of water quality in hellbranch run: a historical perspective. *Senior Honors Thesis*, 2011.

[125] G. K. Wallace. The JPEG still picture compression standard. *IEEE Transactions on Consumer Electronics*, 38(1):xviii–xxxiv, 1992.

[126] Q. J. Wang, D. L. Shrestha, D. E. Robertson, and P. Pokhrel. A logsinh transformation for data normalization and variance stabilization. *Water Resources Research*, 48(5), 2012.

[127] Q. J. Wang, D. L. Shrestha, D. E. Robertson, and P. Pokhrel. An analysis of transformations. *Journal of the Royal Statistical Society Series B (Methodological)*:211–252, 1964.

[128] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996.

[129] A. W. Wood and D. P. Lettenmaier. A test bed for new seasonal hydrologic forecasting approaches in the western united states. *Bulletin of the American Meteorological Society*, 87(12):1699–1712, 2006.

[130] J. P. Wu and S. Wei. *Time series analysis*. Hunan Science and Technology Press, ChangSha, 1989.

[131] R. Wu. *Environment for Large Data Processing and Visualization Using MongoDB*. University of Nevada, Reno, 2015.

[132] R. Wu, C. Chen, S. Ahmad, J. Volk, C. Luca, F. Harris, and S. M. Dascalu. A real-time web-based wildfire simulation system. In *Industrial Electronics Society, IECON 2016-42nd Annual Conference of the IEEE*, pages 4964–4969. IEEE, 2016.

[133] R. Wu, S. Dascalu, and F. Harris. Environment for datasets processing and visualization using scidb. In *the 24th International Conference on Software Engineering and Data Engineering (SEDE 2015)*, pages 223–229, 2015.

[134] R. Wu and J. T. Painumkal. Model accuracy improvement system. URL: `https://github.com/ruiwu1990/machine_learning_prms_accuracy`. [Accessed on 8 April 2018].

[135] R. Wu, J. T. Painumkal, N. Randhawa, L. Palathingal, S. Hiibel, S. M. Dascalu, and F. Harris. A new workflow to interact with and visualize big data for web applications. In *Collaboration Technologies and Systems (CTS), 2016 International Conference on*, pages 302–309. IEEE, 2016.

[136] R. Wu, J. T. Painumkal, J. Volk, S. Liu, S. J. Louis, S. Tyler, S. M. Dascalu, and F. Harris. Parameter estimation of nonlinear nitrate prediction model using genetic algorithm. In *Evolutionary Computation (CEC), 2017 IEEE Congress on*, pages 1893–1899. IEEE, 2017.

[137] A. Ye, Q. Duan, X. Yuan, E.F. Wood, and J. Schaake. Hydrologic post-processing of mopex streamflow simulations. *Journal of Hydrology*, 508:147–156, 2014.

[138] M. I. Yesilnacar, E. Sahinkaya, M. Naz, and B. Ozkaya. Neural network prediction of nitrate in groundwater of harran plain, turkey. *Environmental Geology*, 56(1):19–25, 2008.

[139] L. Zhao, Q. Duan, J. Schaake, A. Ye, and J. Xia. A hydrologic post-processor for ensemble streamflow predictions. *Advances in Geosciences*, 29:51–59, 2011.

[140] Q. Zhu and G. Agrawal. Resource provisioning with budget constraints for adaptive applications in cloud environments. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 304–307. ACM, 2010.

[141] Apache Zookeeper. Apache zookeeper - home. URL: `https://zookeeper.apache.org/`. [Accessed on 8 April 2018].