Novel Applications of Computational Steering

A thesis submitted to the University of Manchester

for the degree of Doctor of Philosophy

in the Faculty of Science & Engineering

2016

By

Junyi Han

School of Computer Science

List of Contents

Word Count: 65297

List of Figures

List of Tables

Abbreviations

4DVAR	Four Dimensional Variational Assimilation
ACM	Association for Computing Machinery
API	Application Programming Interface
AVS	Application Visualization System
AWS	Amazon Web Service
BN	Backend Node
COVISE	Collaborative Visualization and Simulation Environment
CPS	Cyber-Physical Systems
CPU	Central Processing Unit
CSE	Computational Steering Environment
CSWS	Computational Steering Web Service
CUMULVS	Collaborative User Migration, User Library for Visualization and Steering
DA	Data Assimilation
DDDAS	Dynamic Data-Driven Application Systems
DDDCS	Dynamic Data-Driven Computational Steering
DDDCS-HL	Hardware Level Dynamic Data-Driven Computational Steering
DDDCS-SL	Software Level Dynamic Data-Driven Computational Steering
DMF	Demand Multiplier Factor
DDR3	Double Data Rate Type Three
EPA	Environmental Protection Agency
EPS	Extended Period Simulation
FAST	Flow Analysis Software Toolkit

FN	Frontend Node
GA	Genetic Algorithm
GPFS	General Parallel File System
GS	Google Scholar
gViz	Visualization and Computational Steering on the Gird
HDCS	Human-Driven Computational Steering
HPC	High-Performance Computing
HPCE	High-Performance Computing Environment
HTTP	HyperText Transfer Protocol
IaaS	Infrastructure as a Service
MAE	Mean Absolute Error
MPI	Message Passing Interface
NFS	National Science Foundation
OGSI	Open Grid Services Infrastructure
PaaS	Platform as a Service
PFLOPS	Peta Floating Point Operations Per Second
RDS	Relational Database Service
REST	REpresentational State Transfer
RMSE	Root-Mean-Square Error
RTS	Real-Time System
S3	Amazon Simple Storage Service
SaaS	Software as a Service
SQS	Simple Queue Service
SVM	Support Vector Machine

TCP	Transmission Control Protocol
ViSC	Visualization in Scientific Computing
VISON	isual Interactive Simulation
WAMP	Web Application Messaging Protocol
WCT	Wall-Clock Time
WDS	Water Distribution System
WMS	Workflow Management System
WoS	Web of Science
WRC	Water Research Centre
WSN	Wireless Sensor Network
WSRF	Web Services Resource Framework

Name of the University: The University of Manchester

Candidate Name: JUNYI HAN
Degree Title: PhD (Computer Science)
Thesis Title: Novel Applications of Computational Steering

Abstract

Large scale systems in science and engineering are often modelled by numerical simulation to monitor their behaviour and to provide the ability to forecast future state. With the increasing volume of data from observations in the natural and built environments, it is now possible to adjust parameters of such simulations so that they can track more accurately the actual state of the systems. The systems that are tracked by the simulations are named Dynamic Data Driven Application Systems (DDDAS) and the method that adjusts the simulations are already known as the Data Assimilation (DA) process.

The DA utilized in DDDAS is often a compute-intensive and time-consuming process. Hence, existing DA applications can take hours and days to update the simulation. Although this updating speed is acceptable in investigations that are not subject to real-time constraints it is not acceptable in application that are, e.g. weather forecasting, monitoring of forest fires. Thus, a major challenge is to accelerate data assimilation process to prepare the simulation to keep pace with the development physical system being modelled.

Computational steering has been utilized to optimize the exploration of parameter space in simulations. Based on the analysis of the simulation output, we hypothesize that adapting computational steering can also optimize data assimilation. This requires adaptation and automation of methods of utilizing computational steering so that data streams as well as human users can steer simulations.

Consequently, we have developed a steering architecture to implement the steerable data assimilation process. To guarantee the updating frequency required by DDDAS, we introduce time management and computing resource management functions into the new computational steering architecture.

To evaluate our hypothesis and steering architecture, we applied it to the problem of simulating the real-time behaviour of water distribution networks as an example of a DDDAS. Collaboration with the water utilities has provided real observation data for our experiments. We find that, by integrating computational steering with the data assimilation, the running time of the data assimilation method is dramatically decreased. Moreover, using steering results from the data assimilation, human users can also steer high-level parameters of the system. Furthermore, the time management and computing resource management functions are able to manage to control the running time of the steering process to enable real-time prediction of the behaviour of DDDAS.

Date of Submission: 28 December 2016

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

First and foremost, I would like to say special thanks to my supervisor Dr. John Martin Brooke for his consistent help, patience, guidance, motivation and providing opportunities that every PhD student could wish for. To Dr. Jon Shapiro thank you for your timely feedback and priceless suggestions.

Further thanks go to Bruce D'Amora for his generous help during my internship in IBM, and being nice and available whenever I met problems. And to water engineers and scientists in WRc and Northumbrian water company who provide invaluable support on building on use case.

I would also like to express my sincere thanks and appreciation to my parents Hong and Xiyan for their love, support and trust. I cannot thank you enough for raising me up and enabling me to grow as a person. Finally, special thanks to my friend Yangzi for her love, understanding and encouragement. Your unwavering support is the reason this thesis has been written.

Chapter 1 Introduction

1.1 The Problem Domain

Large-scale physical systems, such as meteorological systems, water distribution systems, and electricity grid systems undoubtedly exert a significant influence on our daily lives. In order to gain an insight into such systems, numerical models have been increasingly utilised in simulations to track their behaviours of such systems. Since large-scale physical systems change with time continuously, it becomes essential to adapt those simulations to precisely reflect the real-time states of such systems. To address this issue, observation data is generally taken as the reference for physical systems, in order to calibrate the parameters of simulations.

The development of sensor technologies, as a substantial instrument for obtaining observational data, has facilitated the process of adapting simulations to monitor and predict behaviours of large-scale physical systems. However, the use of sensor data in simulations raises new challenges in computation technologies. For instance, in order to simulate large-scale systems, the uncertainties and flexibilities introduced by the sensor data require intensive computing and “big data” processing capabilities. Despite that High-Performance Computing (HPC) resources are generally used as an infrastructure to provide a high-level computing power for the task, scientists and engineers require more rapid and accurate updates to simulations in order to react to changes in physical systems at the first opportunity.

Computational steering has been emphasised for decades as a tool for increasing the efficiency of running interactive simulations on HPCs. Unlike the traditional cycle, in which human users need to post-process simulation results for analysis and re-run the simulation with changes based on this analysis, computational steering enables human users to change simulation parameters and receive the corresponding feedback from simulations on the fly. However, due to the limitations of modern methods and applications areas, computational steering has not been widely taken up.

Consequently, this thesis is motivated by challenges arising in two domains. The first is derived from simulating physical large-scale systems, which requires a more efficient approach in order to run simulations on HPCs. The second aspect arises from computational

steering since its application needs to be extended to a broader area. This thesis examines the intersection of these two challenges: each of these can aid in solving the other, while together they also raise novel research problems. The remainder of this section discusses separate aspects of the two problems first, and subsequently discusses the challenges which emerge through the effort of integrating the two approaches.

1.1.1 Simulations of Large-Scale Real-Time Real-World Systems

Incorporating observations into computational simulations is one significant method for studying the behaviours of real-world systems [11, 31, 62]. For instance, in a weather forecast system, observational data are obtained from weather stations, airports, ships, buoys, aircraft measurements and radar stations [101], and the observations are assimilated into computational models to track the behaviour of the weather system. Following the development of the Wireless Sensor Network (WSN), more real-time and large-scale information is available for simulating real-world systems. For example, in the United Kingdom, water companies are seeking methods for the integration of real-time sensor data with hydraulic models that can monitor and predict states of real-world distributed water network [81]. In another example [63], real-time data collected by the WSN feeds into a flood monitoring system to produce timely predictions and flood warnings. An additional study [160] describes a price control system in the electricity market that adjusts electricity prices by analysing real-time electricity demand and consumption.

As simulations of real-world systems with real-time observational data begin to attract attentions in a great number of research and engineering fields, including traffic, oil, gas, weather, climate, etc., various issues have been covered as open challenges in active research areas such as Dynamic Data-Driven Application Systems (DDDAS) and Cyber Physical Systems (CPS). As general areas of research, these scientific terms have a scope of application which extends beyond the simulations studied in this thesis. The application area of this thesis focuses only on systems which incorporate dynamic data from the physical world in compute-intensive and real-time tasks. The rest of this section introduces two challenges arising in DDDAS and CPS which are also of interest in this thesis. DDDAS is a term that is related to the application area of this work, and it will be used to indicate the application scope of the rest of this thesis.

The first challenge is that computer simulation systems must have enough flexibility to cope with uncertain changes taking place in physical systems. Dynamic data is a representation

of the flexibility of physical systems. It can indicate specific states and events happening in the real world and requires particular methods, resources and configurations for the cyber systems. The fundamental concept of DDDAS is the incorporation of additional data into executing applications [31]. As a general paradigm related to physical and computational systems, DDDAS explicitly requires systems in its scope to have the ability to dynamically change parameters, algorithms, data selection and computing resources.

The second challenge arises from the time management between cyber and physical worlds. CPS focuses on the intersection between the cyber and physical worlds [36] and is defined as *engineered systems that are built from, and depend upon, the seamless integration of computational algorithms and physical components* [44]. One major concern in this area of research is that the changes of states in the physical systems must be represented on the cyber systems in an accurate and timely manner [80]. The meaning of "accurate" and "timely" depends on specific application areas; however, as the main challenge of CPS, the development of such systems must take these two requirements into consideration. In our case, the accuracy has a correlation with the speed of reaction to changes in physical systems. Since the purpose of the simulations considered in this thesis is to track the behavior of physical systems which vary continuously, a higher frequency of updating leads to higher time-resolution in the simulations results. Consequently, improving the timeliness feature of CPS is the challenge undertaken in this project.

1.1.2 Computational Steering

HPC resources, such as Supercomputers and Cloud computing systems, play an increasingly important role in the field of computational science and have been widely used as infrastructure for data- and compute-intensive tasks in a broad range of application areas. Conventionally, such tasks running on HPCs are non-interactive [19]. Instead, an initial file describing the initial states, parameters and configurations of the task is prepared, and is then submitted to a batch queue to wait for the required computing resources, which are assigned by the task scheduler of the HPC. Afterwards, the simulation runs as a black box based on the prepared initial file, and the states of the running simulation cannot be accessed by users. Finally, the task outputs results to a storage system and the results are post-processed using visualisation to give users direct access. This non-interactive mode, which is also referred to here as batch mode, initially saw wide spread usage in the mid-1980s [59] and is still one important method for executing tasks on modern HPCs [23, 116].

However, with the mounting complexity of computations and the increasing amounts of data in simulations, the post-processing of results, waiting in the job queue and reinitialising simulations in batch mode has inevitably become an intricate task. Since the batch mode conceals the execution process from users, the running simulation may diverge from the region of interest, causing unnecessary usage of computing resources and decreasing the efficiency of analysis of computational results. Consequently, computational steering has emerged as a primary solution to this problem, and provides users with the real-time state of a running simulation. After users have gained insights into the running simulations, computational steering enables the simulation to be steered by modifying the parameters [106]. Thus, computational steering can improve the efficiency of the original feedback loop that consists of two objects, human users and simulations. Figure 1-1 depicts the workflow of computational steering process.

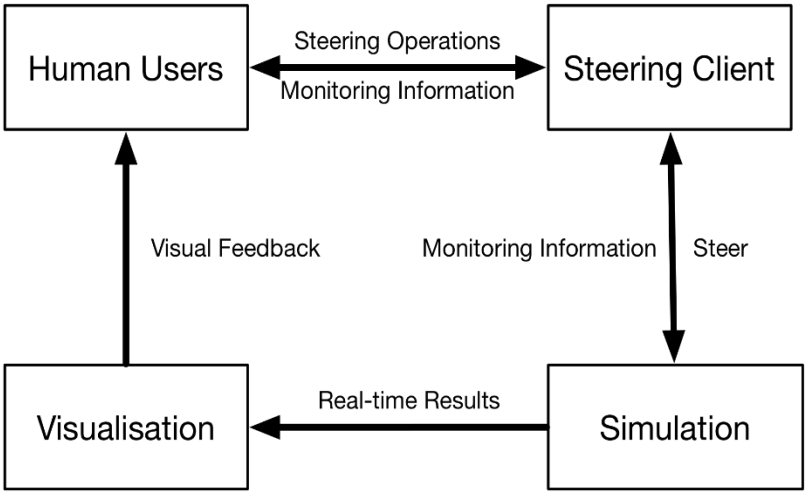


Figure 1-1 Workflow of Computational Steering Process: Arrows indicate communications between components. This figure indicates computational steering forms a feedback loop using Steering Client and Visualisation to help users to get real-time results and change parameters of running simulations.

While a simulation is running, it outputs simulation status and results in real-time. This information is instantly processed by the Steering Client and Visualisation components and presented to human users. According to this timely information, the user can make steering operations to direct the simulation to areas of interest. Due to the benefits arising from this interactive ability, computational steering is gradually gaining popularity and is recognised as a valuable computational tool for exploring computational modelling in fields such as fluid dynamics, molecular dynamics, design optimisation, etc. However, although the concept of computational steering has attracted research interest, the take-up has been

limited in practice. This thesis analyses the reasons for this problem from three aspects: user-friendliness, the human factor and the application environment.

In terms of user-friendliness, computational steering requires specialised computer science expertise from users. For scientists and engineers who are not familiar with HPC infrastructure and network communication, the efforts that they are required to invest in, in order to implement computational steering, may not give sufficient return. Additionally, since computational steering requires a deep integration with HPC infrastructure and software, it is also regarded as a disposable task that can benefit only a few specific projects. Moreover, specific hardware is required by computational steering with regard to security, graphical computing requirements and ad-hoc designed steering clients. Hence, finding an existing computational steering framework that is compatible with users' software and computing resources is an essential but cumbersome task.

Human users are the dominant source driving the entire computational steering process. Through the analysis of real-time feedback from running simulations, they steer the program based on their specialised knowledge, empirical experience and intuition. Although steerable simulations can benefit a lot from human intelligence, the other side of the coin is that computational steering is also limited by human abilities. For example, in simulations with very large parameter spaces or long running times, it is not realistic for users to wait for hours or days in front of a monitor in order to steer hundreds or thousands of parameters. Although the use of algorithms to automate the steering of simulations was raised as an aspect of computational steering as long ago as 1996 [45,123], without a specific application background, existing computational steering projects have insufficient support for automatic steering. As a result, projects using computational steering are rarely driven by automated algorithm.

In terms of the application environment, online simulation is an appealing application area of HPCs. For example, in climate research, scientists utilise the great computing power of HPCs to analyse observations and predict the climate change continuously. As a result, such online simulations require steering ability in order to alter parameters and search for the best fit. However, the real-time nature of online simulations raises new challenges in the corresponding steering process. Computational steering is generally limited by its sole use with offline simulations in practise and existing works rarely have methods to estimate the

time taken by users to conduct steering operations. In this thesis, we [1] denote offline simulations as those that have no real-time interaction with physical systems or other programs.

1.1.3 The Integration of Computational Steering and Dynamic Data-Driven Application Systems

The development of DDDAS has provided opportunities for computational steering, and conversely, computational steering has provided the capability to tackle certain challenges of DDDAS. It is therefore hypothesised that the integration of computational steering with DDDAS can facilitate the development in both areas. By taking the DDDAS as the main application scope of this thesis, this section discusses the reasons for this hypothesis and further challenges raised in the integration. It is also significant to emphasize that computational steering does not create a new real-time computation, it only works with existing real-time computations in DDDAS.

Opportunities and Challenges Raised by the Data Assimilation Algorithms

As discussed above, DDDAS applications have high requirements for real-time features. To meet these requirements, tasks must run in a timely manner. Computational steering is widely used to improve the efficiency of running simulations, and thus, in order to address the problem of DDDAS, we need to find compute-intensive tasks that are required to run on HPCs in existing work on DDDAS. The process of incorporating observational data into simulations is a primary target.

This process is also referred to Data Assimilation (DA), and is well-known as one of the core techniques for modern forecasting in various earth science disciplines including meteorology, oceanography, and hydrology [108]. Data assimilation is a procedure in which a model is periodically adjusted on the basis of physical observations, and the adjustment process must be confined to a limited time window. While the specific algorithm for implementing data assimilation may vary, many widely-used assimilation algorithms are similar in terms of reducing the difference between the estimated states output by the simulations and the real states obtained from the physical world [33]. Such simulations are driven by an estimation of the initial parameters of models; hence, to minimise these

[1] The author of this thesis is the sole author and the sole contributor to this thesis. However, the word *we* is used throughout the thesis, since this is common practice in scientific literature and can be used to refer to a generic third person.

differences, models need a very large ensemble of estimated parameters to have a high probability to generate an accurate simulation. Hence, if the simulation is compute and time-intensive, the overall time consumption required by the DA is greatly increased by the ensemble size, and DA typically leads to high consumption of time and computing resources [121].

The effect of this great time consumption is magnified by the development of sensor technologies, as an increasing number of application areas, such as distributed networks for water, gas and electricity, begin to utilize DA. With a high update frequency of sensors, more timely data collection and analysis becomes possible than in traditional applications of DA. However, although a high frequency of updating brings advantages in terms of richness of data, it also creates a challenge in that the existing DA methods must handle more data in a smaller time window. However, many recent works overlook this aspect of the time management, and this situation gives rise to a “data-rich but knowledge-poor” or “data updated but knowledge outdated” problem.

To reduce the running time of DA, prior work has carried out DA using HPCs. However, in terms of the efficiency of running jobs on HPCs, the ensemble DA methods have efficiency problems similar to those of other batch tasks. The main difference is that in DA, algorithms automatically alter the parameters of models, rather than the human users in conventional computational steering. In this thesis, DA is considered as the main compute-intensive component accounting for a large proportion of the time consumed in DDDAS. Although the time taken by data transport, storage and analysis of results can be important, the running speed of these components forms a separate research topic in DDDAS and is beyond the scope of this thesis. Since the DA workflow on HPCs is in batch mode, this exactly matches the problem domain addressed by computational steering.

Recent work in the area of DA focused on using algorithms such as 4DVAR [28], Kalman filter [62], genetic algorithm [35] to automatically drive programs running on HPCs. However, computational steering is generally driven by human rather than by algorithms, and few existing works have conducted experiments on steering applications using algorithms automatically. Hence, no existing theory and architecture have been developed to support such requirement, and the use of algorithms to automatically drive high-level components with complicated functions and requirements has been a challenging research area for decades. Furthermore, it is reasonable to argue that it is necessary to retain a human

intelligence component in DDDAS until automatic approaches can “perfectly” handle dynamic real-world systems. Therefore, human intelligence and interaction are still valuable in terms of endowing DDDAS with flexibility. Hence, a further problem which arises is how to integrate conventional computational steering with the requirement for driving the applications of DDDAS with an algorithm.

Challenges Raised by DDDAS as a Real-Time Computing Systems

Real-Time Systems (RTS) are defined as systems in which the correctness depends not only on the logical result of computation but also on the time at which the results are produced [131]. As discussed in Section 1.1.1, DDDAS applications are always driven by real-time data from the physical world; DDDAS, as discussed in this thesis, can therefore be considered as a real-time computing system.

RTS can be divided into soft and hard RTS. In a hard RTS, a response is worthless once a deadline has passed. By contrast, in a soft RTS, the value of the response declines as it passes the deadline. Since a delayed result still have some values in a DDDAS application, such as in the case of a delayed monitoring or prediction result, the real-time system discussed in the remainder of this thesis is a soft RTS. Time management in soft RTS has been researched for decades. In order to retain the determinism, correctness and timeliness, methods such as planning schemes [22], running time estimations [37], deadline assignments [78] and scheduling strategies [4] have emerged as individual research subfields. However, although DDDAS is an important application area of soft RTS, time issues are addressed only in CPS, where projects use methods of Workflow Management System (WMS) as an effective approach to conduct time management.

However, time management methods in CPS can only allow a slight deviation from deadlines. This requires the system to have a stable or predictable performance which can only be feasible for low-level computations that are distributed on an embedded network. Nevertheless, with developments in complexity of computation, recent works have begun to transport computation tasks from embedded systems to distributed computing systems such as supercomputers and cloud computing systems. This breaks the original stable and predictable computing environment and introduces uncertainties from distributed and remote computing and network resources [103]. Hence, it is necessary to develop a high-level time management method to handle the time issues in the high-level simulations.

These time issues persist with integration of DDDAS with computational steering. Moreover, integrating computational steering with DDDAS can introduce additional uncertainties to time management. Since this work is the first to attempt an analysis of time management in a steerable DDDAS, questions such as: “Does steering driven by dynamic data or human users have a significant influence on time management?” and “How much do they affect the time management?” need to be answered. A further challenging research question is “What we can do to retain the timeliness when this integration has an inevitable influence on the time management?”

1.2 Research Hypotheses

The central hypothesis of this project is driven by two research questions:

“How can the efficiency of running DDDAS simulations on HPCs be increased?” and “What is a suitable application area to extend the usage of computational steering?”

H1. Computational steering can be applied to DDDAS in which the steering is driven by both human intelligence and dynamic information collected from the physical world. This can improve the efficiency of data assimilation algorithms in DDDAS that usually run in batch mode on HPCs, and can also extend the usage of DDDAS applications.

From the main hypothesis, this work is grounded on the interaction between the cyber and physical worlds. Based on this application environment, three further research questions are developed with corresponding hypotheses. The first concerns the usability of Computational Steering:

How can time management be carried out in computational steering after its integration with DDDAS?

H2. Time is the main concern in the interaction between the cyber and physical worlds. To apply computational steering in applications which involve interactions with the physical world, it is hypothesised that the time consumed by dynamic data-driven computational steering can be estimated according to the dynamic data and can be managed by adjusting computing resources. Unlike dynamic data-driven steering, human users make steering decisions based on human judgement, intuition and

expertise. Hence, it is also hypothesised that the time required by human users is manually determined by users.

What computing infrastructure is suitable for supporting new applications in computational steering?

H3. We can implement a new computational steering architecture on modern HPC resources since HPCs can provide great computing power to support compute-intensive tasks targeted by computational steering.

What real-life applications can benefit from new applications of computational steering?

H4. As computational steering can facilitate the efficiency of the feedback loop in running simulations on HPCs, it is hypothesised that data assimilation and human analysis in DDDAS applications can benefit from computational steering.

1.3 Aims and Objectives

The main aim of this project is to integrate computational steering with DDDAS. This integration has two levels. The first level is the process of integrating computational steering with the data assimilation process of DDDAS. Hence, the first level of integration concerns about driving computational steering by dynamic data from the physical world. The second level involves the integration of conventional human-driven computational steering with the first level of integration. Hence, the second level enables human users to conduct higher-level steering on applications of DDDAS according to the results of the dynamic data-driven computational steering.

Based on this aim, the objectives of this research are as follows:

O1. To review state-of-the-art applications of computational steering and existing approaches that incorporate physical data in DDDAS.
O2. To apply the functions of conventional computational steering to DDDAS applications.
O3. To develop an architecture that can support both dynamic data-driven computational steering and human-driven computational steering based on DDDAS requirements.
O4. To implement the above architecture on HPC resources.

O5. To demonstrate the functionality and usability of the components developed above by utilizing a Water Distribution System (WDS) as a use case.

1.4 Research Methodology

The research method of this thesis involves three components: 1) a comprehensive review of computational steering and DDDAS, in order to achieve objective O1 and to support O2 and hypothesis H1; 2) theoretical work related to utilising concepts of conventional computational steering in the context of DDDAS, to achieve objective O2; 3) architecture development to support objectives O3 and hypothesis H2; and 4) architecture framework and evaluation in order to achieve objectives O4 and O5, and support hypotheses H3 and H4.

Figure 1-2 provides a guide to the structure of this thesis. This research is primarily an experiment-driven project showing that the integration of computational steering with DDDAS can facilitate running applications of DDDAS on HPCs. In addition, this thesis is a build-driven project, since 1) existing computational steering architectures are not feasible for conducting the experiment, and 2) it is necessary to demonstrate that new applications of computational steering can be deployed on real HPCs and can be applied in real-life DDDAS environments. From Chapter 3 to Chapter 5, therefore, this work develops computational steering architecture, developing computational steering components in the new DDDAS context (Objective O2), and developing computational steering architecture to realise the functions and workflow of components in the architecture (Objective O3) and developing a framework for the implementation of the architecture on supercomputer and cloud computing systems (Objective O4).

Furthermore, the contents of this experiment-driven research are partially incorporated in this structure:

- To achieve objectives O1 and O2, we have conducted a comprehensive statistical review of the historical development of computational steering. The data is collected using a program written to collect related references from popular academic online search tools. In this review, the concepts of computational steering are examined, and an analysis is carried out of the factors hindering further development of computational steering and showing promising advancing direction. This study is included in the background chapter.

- To support H3, time management methods designed in Workflow Management System (WMS) are studied and time is considered as the primary factor of the quality of service. More specifically, this project draws on the experience of the deadline assignment, workflow scheduling, planning scheme and runtime prediction methods of the WMS. This study is conducted and the architecture is developed in Chapter 4.
- As the proposed system evolves from computational steering, it is important to select an existing computational steering architecture as a prototype to achieve O3 and support H3. However, most current steering architectures are dedicated to the areas of infrastructure and application. Rather than modifying an existing architecture, the core steering concepts are adopted from the RealityGrid steering library[2], and its main features are conversed; in addition, it is customised to be compatible with applications of DDDAS.
- To further support H3 and O4, a collaboration was carried out with IBM to implement human-driven computational steering on a IBM Blue Gene/Q system. This collaboration provided a prototype for implementation of our architecture on other HPC infrastructures. This collaboration is described in Chapter 5.
- At last, to support hypotheses H1 and H4 and to achieve objective O5, a set of experiments were conducted using applications of Water Distribution System (WDS) as a use case; these experiments were run on frameworks developed for IBM Blue Gene/Q system and the Amazon Cloud computing system. The IBM Blue Gene/Q system is used as the example of supercomputers since two of its implementations are ranked as the 4th and 6th most powerful supercomputer [2]. Besides, this PhD project has a deep collaboration with IBM since their scientists are interested in applying computational steering to their system. Using Amazon Cloud as the example of cloud computing system is because they provide flexible and affordable computing resources. Based on the chosen computing platforms, we compared the performance of steerable application and the un-steerable control group. In order to further demonstrate the benefits of this work, new functions were developed that can only be achieved using the steerable method to address some of the research problems in WDS. The results of this use case were incorporated into a knowledge

[2] The RealityGrid steering library is part of the RealityGrid project and aims to provide a highly flexible and robust computing infrastructure to support the modelling of complex condensed matter systems.

exchange project between University of Manchester and water companies. These contents are included in Chapter 6, which presents an evaluation of this work.

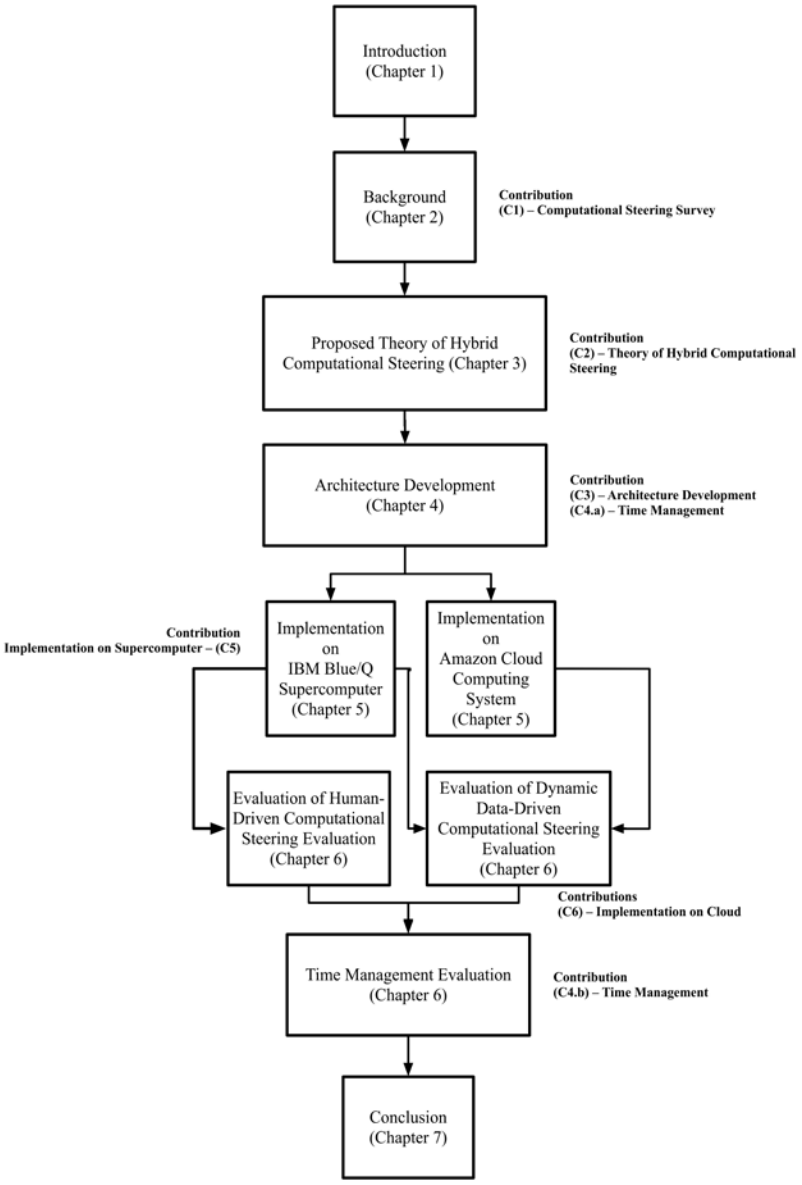


Figure 1-2 A Guide for readers on the Structure and Dependencies of this Thesis: The head node, Introduction, generally describes the problem arising in areas of computational steering and dynamic data-driven application systems. It is followed by Chapter 2, the background, in which we have a comprehensive review on features and development of computational steering and its proposed new application area. Since no comprehensive review of computational steering was conducted in recent years, this review is claimed as contribution C1. Based on the review, we introduced our new understanding of concepts of computational steering in Chapter 3, namely hybrid computational steering. Since this new understanding can realise the integration of computational steering with dynamic data-driven application systems, it is considered as contribution C2. Afterwards, Chapter 4 describes a proposed architecture for development of conceptual components raised in Chapter 3, and this architecture is considered as contribution C3. Additionally, to realise coordination among components in the architecture, it also reviewed and proposed a time management method. To demonstrate the hybrid computational steering, the general architecture must be implemented on computing infrastructures. Hence, IBM Blue Gene/Q supercomputer and Amazon Cloud computing system are selected as two representatives of HPC infrastructures. Due to the difference between them, two frameworks are implemented separately based on the general architecture in Chapter 5. The two implementations are considered as contributions C5 and C6. The implementations are used to evaluated human-driven steering and dynamic data-

driven computational steering in Chapter 6. The framework designed for Blue Gene/Q is used to evaluate both types of steering. However, since the cost of using Amazon Cloud is great, only the dynamic data-driven computational steering is evaluated on the cloud. Moreover, the time management method proposed in Chapter 3 is also evaluated using water distribution system as an example in Chapter 6. The method and evaluation constitutes the contribution on time management C4.

1.5 Research Contributions

The novel contributions of this thesis are as follows:

C1. An exhaustive review and analysis of computational steering.

C2. A new understanding of advantages of computational steering and the use of these advantages in DDDAS applications.

C3. The provision of an architecture to support C2.

C4. The introduction of time management in computational steering.

C5. The provision of conventional computational steering as a web service on IBM Blue Gene/Q supercomputers.

C6. The introduction of using cloud to dynamically assign computing resources to support the hybrid computational steering.

It is necessary to clarify that contributions of this PhD project are achieved by collaborating with other researchers, scientists and companies. The collaboration includes:

1. Working with scientists in IBM to design and develop the computational steering web service. This collaboration is discussed in Section 5.2.
2. Simulating communications with a computing resource broker that is designed by a PhD student, Zeqian Meng. The broker is used to support C6 and it will be discussed in Section 4.3.
3. Using an online water modelling tool designed by Dr. Kashif Khan as the use case. This toolkit provides reference to code of algorithms and water simulations packages that are modified to meet requirements of experiments discussed in Chapter 6.

1.6 Publications and Research Activities

1.6.1 Publications

1. Han, Junyi, Robert Haines, Adel Salhli, John Martin Brooke, Bruce D'Amora, and Bob Danani. "Virtual Science on the move: interactive access to simulations on

supercomputers." In IEEE 25th International Conference on Application-specific Systems, Architectures and Processors (ASAP), 2014, pp. 178-179.

2. Han, Junyi. "Steering simulations on high performance computing resources." In IEEE International Conference on High Performance Computing & Simulation (HPCS), 2014, pp. 1008-1010.
3. Han, Junyi, and John Brooke. "Hybrid Computational Steering for Dynamic Data-driven Application Systems." Procedia Computer Science 80 (2016): 407-417.

1.6.2 Activities

1. Internship at the IBM Thomas J. Watson Research Centre in New York, USA collaborating with IBM engineers on a prototype for dynamic computational steering architecture, September 2013 to December 2013.
2. Study of data visualisation and high-performance computing at the Hartree Centre Summer School, June 2014 to July 2014.
3. Assisting Dr. John Brooke of the Daresbury Laboratory in setting up a computational steering environment on IBM Blue Gene/Q systems, August 2014.
4. Meeting with users and developers of computational steering at the Daresbury Laboratory, April 2013.
5. Presentation and discussion with water engineers at Thames Water Utilities, June 2013.
6. Presentation and software demo at Thames Water Utilities, August 2013.
7. Presentation and software demo at Northumbrian Water Company, September 2015.
8. Poster presentation in a poster symposium in the school of Computer Science at the University of Manchester, November 2014.
9. Oral presentation in a research symposium in the School of Computer Science in the University of Manchester, October 2015.

1.7 Thesis Structure

The structure of the remainder of this thesis is presented as follows.

Chapter 2 presents an overview of the development of computational steering, introduces some background knowledge of other related areas and reviews related works.

Based on the understanding of the conventional computational steering described in Chapter 2, Chapter 3 introduces the integration of computational steering concepts with dynamic data-driven application systems in terms of the types of simulations, the driver of the steering, the aim of steering, benefit of the integration and time issues raised by the integration.

Chapter 4 describes a proposed architecture which supports the integration discussed in Chapter 3. The framework describes the workflow of the new computational steering process driven by dynamic data, and in addition, and also integrates conventional computational steering driven by human users. Moreover, it explicitly introduces the development of three primary components: the Steerer, Time Manager and Computing Resource Manager.

Following this, Chapter 5 introduces an implementation of the conventional computational steering on the IBM Blue Gene/Q supercomputer. Based on this implementation, the implementation of components of the proposed dynamic data-driven computational steering on both supercomputers and cloud computing systems is then presented.

In Chapter 6, the implementations discussed in Chapter 5 are evaluated by using simulations of a water distribution system as an example.

At last, the ability and performance of the new computational steering application are summarised, and an agenda for future works is suggested in Chapter 7.

Chapter 2 Background and Literature Review

This PhD project is initially driven by the need to improve the usability of computational steering. As the research continues, we argue that, instead of reducing the cost of using conventional computational steering, it is an opportunity to integrate computational steering with Dynamic Data-Driven Application System (DDDAS) that requires the ability to interact with simulations to improve the efficiency of utilising HPCs. As a result, we need to have a comprehensive review of existing works in computational steering to understand its concepts, developments and to introduce methods to meet requirements of DDDAS. Additionally, we need to study the requirements of DDDAS as a new application area that provides new challenges for existing computational steering projects.

Thus, at the beginning of Section 2.1, we have a background study of existing projects of computational steering, and afterwards, we conduct a statistical study to analyse the development, impacts and challenges of computational steering. In Section 2.2, we have a background study of DDDAS so that its application areas and data assimilation algorithms can be understood to facilitate its integration with computational steering. Finally, in Section 2.3, we discuss modern HPCs since we need to implement our project based on HPC infrastructures which can help applications of DDDAS to keep pace with physical systems.

2.1 Computational steering

As claimed by Mulder et al., in a broad perspective, all interactive computational processes belong to the scope of computational steering [106]. In existing computational steering works, this scope is confined into scientific and engineering modelling and simulations. The value of such computational steering has been emphasised for more than two decades. However, the impact of steering has been much less than originally predicted. The aim of this section is to understand the development process of computational steering and to analyse fundamental concepts of computational steering to form the background knowledge. Additionally, according to its development history, the author discusses reasons that form the recent stagnation in uptake of computational steering. This analysis includes the original

motivations, research interests in different periods, the research climate around computational steering and other factors leading to the current situation.

2.1.1 The Origin of Computational Steering

Computational steering has been applied in many areas in the computational science. A broadly accepted computational steering architecture comprises a steering Application Programming Interface (API) enabling users to manoeuvre simulations. Additionally, it contains a visualisation component which provides users with real-time visual results of running simulations. However, except for this common understanding, there is a paucity of a common taxonomy of computational steering. Since this project aims at introducing computational steering into DDDAS, we have to research existing steering paradigms and understand their conceptual components. The basic concepts of computational steering have been discussed in the Section 1.1.2, and this section further discusses computational steering in terms of functions, architectures and application areas.

The first concept of computational steering emerged around 1970s. Works such as OPS-4 [74], SIMPLE [38], Interactive GASP IV [45] and NGPSS [10] appeared during that time. Most of these works were driven by the trend that High Performance Computing (HPC) resources were available for more general users. Thus, users needed to find an efficient way to utilize computing resources. One method to improve the efficiency is to understand the working process of running simulations. By using the early steering tools, users could build, test and validate simulations in an interactive way. Although those works showed the value of interactive simulations, the visualization of early computational steering was only able to present simulation results as complex and tedious data. It was difficult for users to build concrete pictures of simulations only according to this information. To tackle this issue, Hurrion [64], in 1978, introduced the “animation” into interactive simulations. The original aim of his work was to reduce the confidence gap between users and developers by enabling users to understand the processes of simulations. Hurrion’s work was realised in the project named SEE-WHY and is well-known in the area of “animation” which aims at portraying running simulations. In 1979, Hurrion integrated functions in SEE-WHY with a dedicated simulation programming language to produce the Visual Interactive Simulation (VISON) [66]. To the author’s knowledge, VISON is considered as the first generation of computational steering (even if it did not use computational steering as its name), and it established two fundamental functions of computational steering, 1) interaction and 2)

visualization. After VISION, works such as [65, 137] established the fundamental environment of the so called visual interactive simulation around the 1980s. Consequently, the emphasis of the interaction was on getting visual feedback from running simulations.

Afterwards, Mc-Cormick et al. 1987 [98] in the ViSC workshop (Visualization in Scientific Computing) pointed out that scientists and engineers wanted to have multiple ways of interacting with their running computations on an integrated platform. This influential report considered steering as an indispensable function in visualization, listed steering tools as the long term needs in visualization and even pictured the future scenario in which scientists use computational steering to solve problems. These ideas were transformed from a requirements specification to an implementation vision rapidly. This report turned Scientific Visualization and computational steering into a major research area [144].

The importance of the facility of communication between users and their simulations were re-emphasized by Brooks [20] and Upson [140] in the literature. Afterwards, computational tools, which assimilate those ideas, emerged. Works such as Application Visualisation System (AVS) [141], FAST [12], APE [30], etc. were the first to introduce steering and visualization tools into specific application areas. Robert [94], in 1990, mentioned this transition phase in visualization. Researchers began to design their toolkits into three categories: post-processing, tracking and steering, which were considered as the ultimate goal in visualisation technique during that period. Although those tools were not fully fledged as computational steering environments but as early prototypes, they presented the common acknowledgement that a computational steering tool could efficiently help users to understand simulations and reduce time intervals between modifications and analysis. More advanced environments, such as the Application Visualisation System (AVS), provided ease of use, low cost, completeness, extensibility and portability with computational steering. Those requirements provided principles for the emergence of computational steering environments during the rest of the 1990s.

Therefore, during the rest of the 1990s, many teams dedicated themselves to the area of computational tools that helped users to interact with running simulations. Benefiting from the increasing number of computational toolkits and environments, which were driven by the increasing power of HPC resources, researchers began to consider further interaction abilities such as altering parameters of running simulations and switching algorithms used in running computations. The period from 1990 to early 2000s is considered as a booming

era for computational steering. Different teams began to develop prototypes of computational steering based on requirements from users. This tendency led to the situation that there was no unified standard for computational steering. Different systems had different understandings of the steering ability and developments of steering environments. In 1995, Progress [149] and Scirun [110] were first announced as steering frameworks which focussed on these steering enhancements. In 1996 and 1997, CSE [88], CUMULVS [51] and Magellan [147] joined computational steering family. Later, in 2003 and 2004, gViz [18] and RealityGrid [19] were published as the latest generation of computational steering. These works, by integrating the experience from other scientific projects, developed the embryonic theoretical basis of computational steering. This thesis now analyses these typical systems to focus attention on 1) what are original interests that attracted researchers to be involved in computational steering, 2) what are the significant progresses since computational steering was introduced, 3) what are the issues in current works and 4) why has the impact of computational steering decreased.

2.1.2 An Overview of Computational Steering Environments

This section introduces steering environments that have made significant contributions on the development of computational steering.

Progress

According to Jeffrey [149], if High-Performance Computing (HPC) resources continue to remain non-interactive, end-users and program developers will not capitalize on new techniques for interactive data visualization and program animation, remote and collaborative work, interactive debugging and monitoring. The development team of Progress explored requirements and challenges for computational steering by designing a practical toolkit. The applications, which can use Progress, should be able to run in parallel on supercomputers. The steering is defined as the runtime manipulation of an application program and its execution environments, and the objectives of steering are performance and functionality improvement. They distinguish their work from debugging and visualization by pointing out that only computational steering affects simulations results. In terms of the understanding of computational steering, Jeffrey agrees with Mukherjee [104] and Bihari [16] that steering can keep users and the targeted progress “in the loop” when the simulation is running. The loop consists of manipulations on parameters and real-time responses from running simulations.

In terms of the architecture of Progress, it consists of a server and a client. The server is a separate thread running in the same memory space of the simulation. The client is running on another terminal which provides users with interfaces to steer the running simulations. However, Progress did not provide an adequate visualization interface for users to monitor simulations and get visual feedback. The developers mainly focused on the design of steering functions. In order to make the server communicate with the simulation, programs are annotated with the Progress library. Operations such as reading, writing, getting states of an object can be realized as function calls in Progress library.

Scirun

Developed at the same time as Progress, Scirun [110], which was proposed by Parker, treated itself as a general purpose environment which should be suitable for most software requirements and hardware environments. According to the concept of Scirun, steering can clarify the cause-effect relationships within simulations. It can also facilitate the scientific research process by finding answers of What-If questions, namely what happens if an alteration was made on running programs and its executing environments. Parker thought steering can close the loop between modifying and getting feedback. However, in Scirun, the focus is not only on how to realise steering, it focused more on how to build an environment which can "*integrate a wide range of computing disciplines with a wide range of equally disparate application disciplines*" [111]. Therefore, Scirun is intended to provide a computational workbench for scientists.

The workbench is based on the data-flow programming model in which designers use modules to represent their algorithms and use connections to build relationships among those modules. The provided C++ class indicates users how to declare steerable parameters in modules. Furthermore, as a workbench, Scirun provides a 3D visualization component in the user interface which makes it more like a mature software engineering product.

CSE

The Computational Steering Environment (CSE) was presented first by Jarke et al. in 1994. However, this work can only be found in references of later works Thus, we discuss CSE by using [144] and [143] instead. As they consider computational steering as the ultimate goal of interactive simulations, their work is motivated by the fact that computational steering can "*greatly reduce the time between changes to model parameters and the viewing of the results*". Furthermore, as a work in the rising phase of computational steering, it took

two important factors into consideration during its development. The first one was that the impact of computational steering can be determined by the steering speed. It means that simulations running on HPC resources must run at an interactive speed. Making simultaneous interactions with steered simulations is considered as an important feature of computational steering. Secondly, Jarke realised that even if computational steering is an ideal goal for interactive simulation, it is important to find an operable and realistic way to enable general users to use it.

The architecture of CSE is in the server-client format. The server is a data manager which is responsible for recording steerable variables of different simulations. Annotations should be made on the simulation, which is the client here, to build connections to the server. The most important feature of CSE is that users can steer parameters by manipulating graphical units in the visualisation interface [105].

CUMULVS

After several computational steering environments emerged, CUMULVS [51], which was proposed by Geist, was more like a commercial steering software package. However, as well as contributing to making computational steering more convenient for normal users, it also expanded functions of previous computational steering toolkits. Geist et al. have a clear recognition of the application area of computational steering, they proposed that computational steering can not only facilitate the debugging process, reveal algorithm dynamics and identify subtle errors, but can also help to explore physical models. They believe that "*computational steering has the potential to revolutionize computer simulation experiments by allowing scientists to interactively explore (steer) a simulation on time or space dimensions and concentrate more on the science than on the computer*". The two main contributions of CUMULVS are, 1) it brought fault tolerance into computational steering which allows users to roll back the running simulation to playback what is interesting or correct mistakes, and 2) CUMULVS enables multiple users to steer and collaborate on the same simulation.

The developers positioned CUMULVS as a middleware which bridges the gap between application codes and commercial visualization packages. This positioning, on the one hand, tightened the relation between the steering and visualisation, on the other hand, it clarified the boundary between steering and visualisation.

RealityGrid Steering Library

Developed after 2000, RealityGrid steering library has become one of most well-known Computational steering environments. It received strong support from the UK eScience programme and has targeted simulations in the atomistic and meso-scales applications of material science. As a project starting in the grid era of the early 2000's, the RealityGrid steering team did not only focus on the steering functionality, but also paid great attention on the infrastructure on which computational steering is applied. It made a major contribution to integrating computational steering architectures with the grid computing infrastructures. As a toolkit, which was motivated by the RealityGrid project, it provided a computational steering environment which was realised as a service running on the grid. After experiencing the rapid change in proposals for grid standards, such as Open Grid Services Infrastructure (OGSI) [138] and Web Services Resource Framework (WSRF) [41], the RealityGrid steering environment evolved into a flexible and modularised design to reduce the interference made by changes in web service standardization and also to be compatible with different HPC architectures. This thesis considers that this adaptability has been a key factor in ensuring that the RealityGrid steering library is still being used in current software projects.

gViz

Unlike the projects introduced above, the gViz project focused on integrating an existing visualisation system (Iris Explorer) with grid infrastructures. To overcome challenges that scientists experienced when connecting to remote visualisation system from their own devices, gViz intended to provide a system which can address issues caused by the use of visualisation in grid computing. However, since gViz is driven by issues in the grid computing, the project paid less attentions to steering functions compared with other projects mentioned above. Computational steering is treated as an add-on in this system.

gViz went a further step to hide complex architecture from normal users. It built two separate systems, a backend system which provided grid resources for simulations, and a front-end system which provided visualisation functions for users.

2.1.3 Taxonomy of Computational Steering

Based on the emergence of computational steering environments, understandings of computational steering gradually converged. A generally accepted definition of

computational steering was first introduced by Parker et al. [111]. Application steering, algorithm refinement and performance tuning were considered as the three predominant functions of computational steering. The application steering refers to the ability to direct computational processes by altering parameters of simulations at runtime. This is also named as model exploration in later works that falls into this steering category. The algorithm refinement provides the steering ability to modify configurations of algorithms. The performance tuning is to steer configurations of the computing resources that are utilised by running simulations. In a summary, computational steering provides a tool to explore the parameter space of applications. The term, "parameter space" is used in this literature to describe different aspects of a steerable object. It can indicate parameters of simulations. Additionally, it can represent parameters of algorithms and configurations of computing resources. To realise the above functions, 1) integrating computational steering with applications, 2) making steering decisions and 3) conducting steering operations are considered as the essential components in computational steering workflow.

1) Implementing Steering

In the development of computational steering environments, many projects incorporated computational steering at the development phase by inserting library calls into the code. However, in many areas, comprehensive and well-tested simulation packages have no support for computational steering. It is not cost-effective to re-develop these projects by including calls to computational steering libraries in their code. Hence, the question, how to integrate computational steering with an existing software package**,** is of great importance.

SCIRun [110] and SCIRun2 [159] are two examples of computational steering environments that only require users to re-develop existing projects. They are based on the dataflow programming paradigm and they model a program by abstracting instructions as nodes that operate on the data flow. Instruction nodes are realised as modules that are written in C++. Specific steering C++ functions are provided for users to integrate customised modules with computational steering. However, this environment limits the programing paradigm for general users, and it is expensive for an existing project to be integrated into SCIRun.

Annotation of code is used by other computational steering environments, e.g. RealityGrid, CSE and CUMULVS. To implement the code annotation, users must have a comprehensive understanding of the projects and programs. Parameters of interests are registered by using the programing interface provided by the steering environment, and the communication

between the visualisation, steering client and simulations adds further complexity for regular users. Therefore, this approach is more suitable for researchers having strong computer and programming knowledge.

Above steering environments require users to identify the position to make the change on steerable parameters. In general, the steering location is at the beginning or end of the main loop of programs so that states of simulations that regard to the steered parameters can be embodied at the next simulation step. If any variables are related to steered parameters, it is also important to re-compute such variables. Either steering modules or steering APIs are inserted at the steering locations chosen by users. They are responsible for communicating with the steering client and altering parameter values from the memory.

2) Making Steering Decisions

The majority of existing projects consider human users as the decision-makers in computational steering. Two sources of information are considered to inspire the decision-making. The first source is the knowledge of human users, which includes the understanding, experience and intuition of particular application areas. Hence, the effect of computational steering is extensively affected by the individual behaviour of the users. Additionally, the steering decisions are dynamically affected by the feedback from simulations. This feedback is typically represented in the visualisation component that graphically displays the states of simulations. In addition to graphical feedback, values representing the states of simulations, such as real-time values of parameters, can also be embodied in the steering client for the purpose of a precise understanding.

3) Conducting Steering Operation

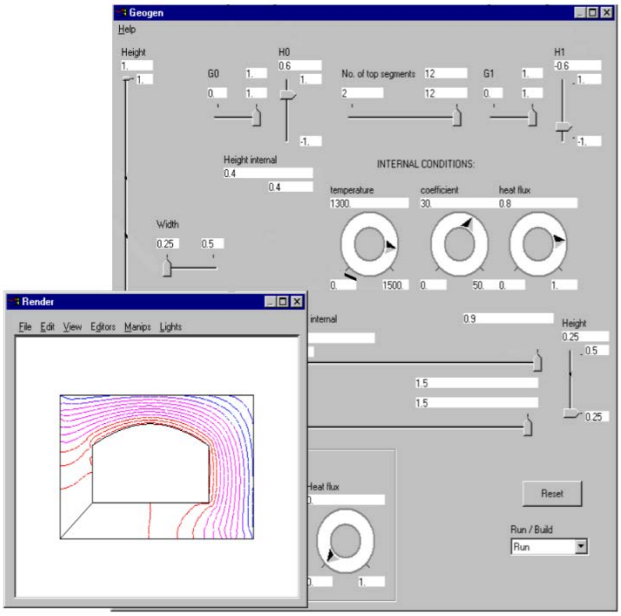


Figure 2-1 A Typical Computational Steering Interface [153]: The graphic window presents real-time results of running simulations. The interface behind it shows parameter values and steering controls that can be used by users to modify steerable parameters.

As depicted in the typical computational steering paradigm in Figure 1-1, human users alter the parameters of a simulation through a steering client. The steering client provides a user interface as the example presented in Figure 2-1. The steering decisions generated by changing the values and moving controls are transferred to the running simulations via this steering interface. In a computational steering session, without significant steering lag, users can have a direct visual feedback relating to their steering decisions via the visualisation component. This type of steering interface is adopted by a great number of environments such as RealityGrid [19], CUMULVS [51], CSE [88] and many other specifically designed steering environments [94]. Additionally, some works [25, 152] integrate the visualisation and steering client components. This integration provides users a more immersive steering experience by avoiding switching between the steering interface and the visualisation. In some works, direct manipulating the visualisation can increase the steering lag experienced by human users, and it is believed by some users that a continuous visualisation context can help to inspire users intuition [153]. Furthermore, McAdam [97] combined the steering client with a hand-held controller, and users can steer the parameters of the simulation by controlling two multi-axis joysticks.

2.1.4 A Statistical Analysis of the Impact of Computational Steering

To analyse current situations of computational steering, only reading papers is not sufficient to evaluate the impact of computational steering. The author made a statistical evaluation by leveraging several popular on-line research tools to analyse the impact of computational steering from 1990 to 2013. Furthermore, as no survey of computational steering has been conducted in recent years, this section also compares the contributions of recent computational steering works with earlier works. In this thesis, we analyse the period from 1990 to 2013 in terms of the impact of computational steering.

The Analyzing Tool

In terms of current searching technology, there are different kinds of academic citation indexing and search services such as Google Scholar[3] (GS), Web of Science[4] (WoS), Citeseer[5] and Scopus[6]. On the other hand, many publishers also have on-line search tools. For example, Elsevier operates the ScienceDirect[7] and Springer[8] has a search tool on its website. Additionally, both Association for Computing Machinery (ACM) and IEEE Computer Society (the primary US umbrella organization for academic and scholarly interests in computing) have their own web search applications such as ACM[9] and IEEEXplore[10]. These tools dramatically improve the efficiency in doing searching, citing, analysing in a particular scientific area. However, despite advantages such search services bring to researchers, there are a number of arguments about whether those tools are suitable and reliable for bibliometrics. Therefore, we need to have an overview of search tools before we can assure that our gauges work correctly or at least with transparent limitations [69].

After analysing recent researches on academic literature collections [5, 85, 87, 99], we found Scopus and WoS have higher quality than GS. Since GS collects information from other websites, it may have the largest coverage related to searched keywords. However, for the same reason, it may include same works that have different details on different websites. Therefore, this thesis uses GS as the first tool to get a gross collection of relevant works and

[3] Google Scholar Website, http://scholar.google.co.uk
[4] Web of Science Website, http://apps.webofknowledge.com
[5] Citeseer Website, http://citeseerx.ist.psu.edu/index
[6] Scopus Website, http://www.scopus.com
[7] ScienceDirect Website, http://www.sciencedirect.com
[8] Springer Website, http://www.springer.com
[9] ACM Digital Library Website, http://dl.acm.org
[10] IEEEXplore Website, http://ieeexplore.ieee.org/Xplore/home.jsp

then utilises WoS and Scopus to filter the collection. In case of any missing references from the list supplied by GS, IEEEXplore, ACM DL, Springer and ScienceDirect are used as additional search tools to make supplementary.

As the comparison work is tedious and there are hundreds of thousands papers needed to be analysed, the author built an application to communicate with the above search services. The principle of this application is simple. First of all, all papers that are relevant to computational steering are recorded with their publishing years and citation numbers by using GS. The citation number is the times a paper is cited by other works. These papers are searched again by using other web search engines. If there are any mismatches, the result from GS is modified. Finally, the number of papers that are relevant to computational steering is counted, and their citation numbers are also noted to have a general picture of the impact and development. The code and user interface are listed in Appendix A.

The Trend of Computational steering

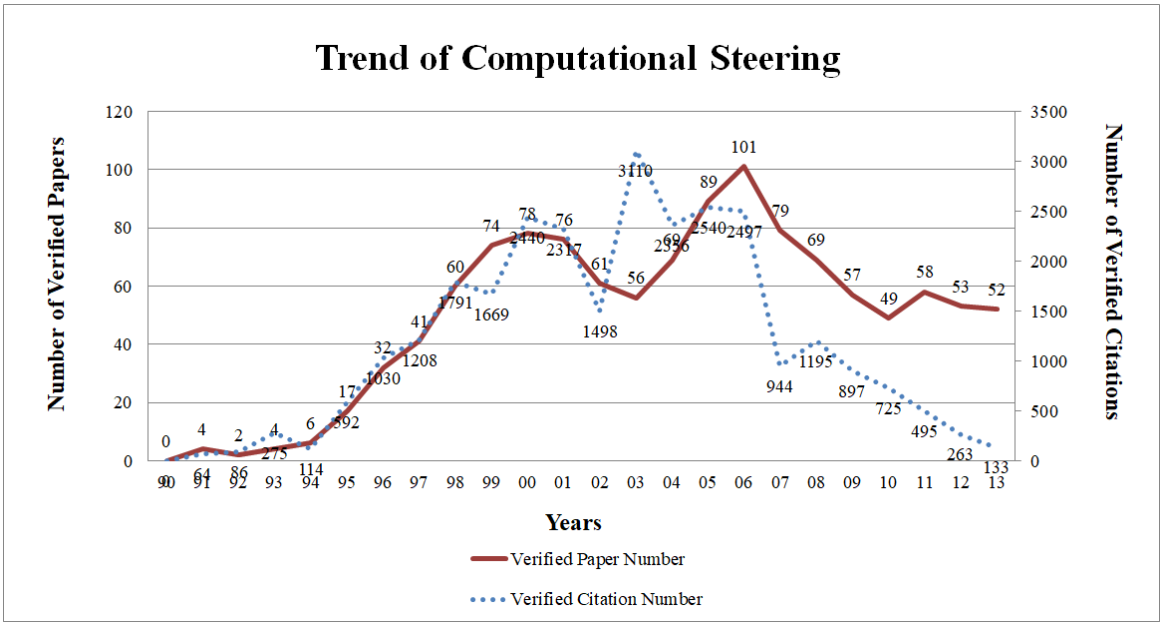


Figure 2-2 Trend of Computational steering: The red and solid line indicates the number of papers of computational steering published and verified in our application in each year. Blue and dashed line indicates number of citations obtained by papers in each year.

As indicated in Figure 2-2, the red solid line stands for the number of papers related to computational steering in each year, and the blue dashed line indicates the number of citations obtained by papers published in each year.

Based on the number of papers, it is evident that from 1990 to 2004, it was a booming period of computational steering. However, this increasing trend stopped at 2006, and both numbers

of papers and citations dropped drastically afterwards and continued on this low-level to the present. Especially between 2006 and 2007, the number of citations decreased more than fifty percent. Additionally, there are two precursors to indicate this drop. The first one is the negative correlation between the citation number and paper number from 2003 to 2004. Another precursor is shown between 2004 and 2006 in which citation numbers remain around 240 for three years, but numbers of papers keeps growing in the same time window.

To understand the development pattern of computational steering, we did another general review to have an overview of modern development of computational steering. In this review, we only select papers that have representativeness or outstanding features. Since we have reviewed the early stage of computational steering in Section 2.1.1, this review will start from 2004.

From 2004 to 2006, many works were interested in integrating computational steering with the grid computing in order to enhance the visualisation capability using the grid computing architecture. Johnson in his paper [72] proposed issues for the scientific visualisation in terms of both computational steering and grid computing. He indicates that "*powerful graphics facilities are becoming available as part of grid based computing systems. Missing are effective ways to tap into the available graphics capabilities of these distributed graphics systems to create scalable visualizations*", and the effective ways turn out to be a key motivation in utilizing grid computing with computational steering. With the same acknowledgement of challenges on the grid, Brodlie et al. [18], Zhang et al. [159], Pickles et al. [114] and Peter et al. [29] are dedicated to tackling problems that exist between grid computing and computational steering and to building environments and tools such as RealityGrid and Scirun2. Other works focus on data management issues in grid computing such as data collecting [90], data storage [54], data movement [48] and communication [119]. Additionally, Darema [31] proposes a new way to utilize concept of computational steering. In her work, she presented a new perspective on steering which, for her, means to "*entails the ability to incorporate additional data into an executing application*". The additional data stated by Darema means observations from the physical world, and the executing applications are related to application systems that interact with physical systems.

However, the turning point appeared in 2007 with a rapid drop on the citation number. According to our review, number of papers that discuss integrating computational steering with grid computing systems decreased as well. Instead, works that have specific

contributions on computational steering took advantage of web technology to provide new clients with computational steering [145], and to decrease the cost in using computational steering. Additionally, Junyi et al. [57] integrated computational steering with IBM Blue Gene/Q supercomputer to provide a consumable HPC web service. Denham, in his work [34], considered the Genetic Algorithm (GA) as a kind of computational steering. In 2010, Wright [154] noticed challenges brought by the increasing hardware and software complexity. She discussed models that use split architectures in which computing resources, visualisation and local devices of users are geographically divided. Moreover, the pipeline of visualisation can be divided as well in order to facilitate steering process. In terms of visualisation, Hernando et al. [60] integrate the immersive visualisation with computational steering. Similarly, Kuckuk et al. [84] use Xbox Kinect as an interface to interact with the running program. As a result, although researchers attempted to make contributions to computational steering, without a strong development on the architecture context, such as grid computing, contributions are diverse and few.

However, only analysing works that aim at enhancing computational steering is not enough. In order to explore motivations behind the increase and decrease of the research trend, it is necessary to compare different factors that attracted interest in steering in each year from 2004 to 2014. Since the amount of papers relevant to computational steering is too large to read one by one, only works of which citation numbers are greater than the average citation number at the same year are analysed. Additionally, selected works are categorized based on the area of their contributions and application areas. The division is based on the idea that computational steering has been involved into many research areas and integrated with technologies in various areas (such as visualisation, big data movement and grid computing). Following this analysis, the proportions of each category is shown in Figure 2-3.

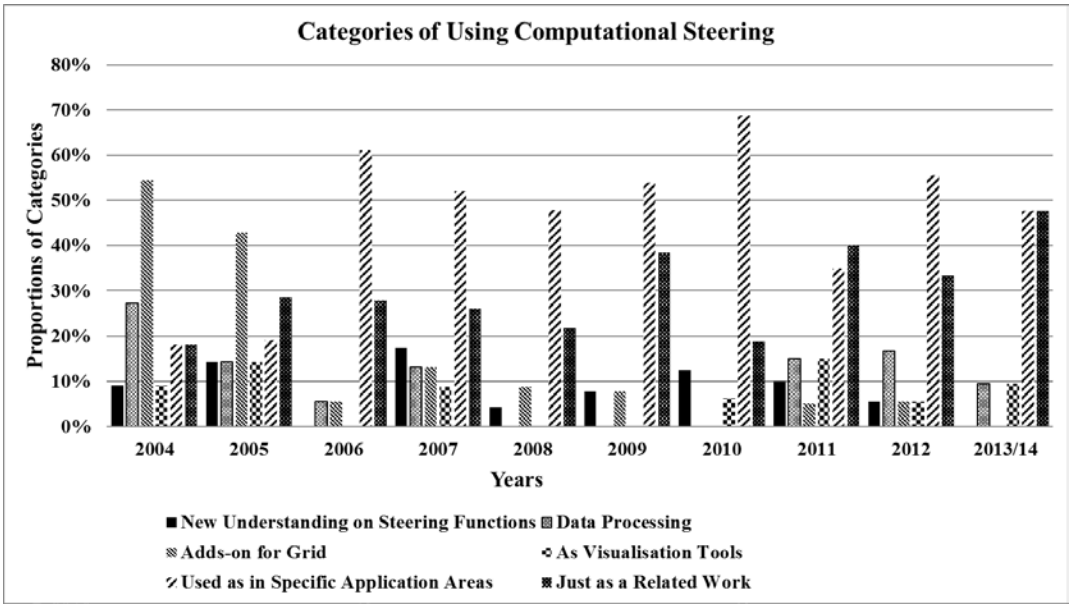


Figure 2-3 Categories of Using Computational steering: Contributions of works from 2004 to 2014 are divided into six categories. The height of columns indicates the proportion accounted by each category in each year. The sum of proportions can be greater than 100 since one work may cover more than one contribution areas.

As shown in Figure 2-3, works during the declining period are divided into six categories. The "New Understanding of Steering Function" means works that focus on improving the steering ability and propose new requirements and issues on computational steering. Works that tackle issues on big data are categorized as "Data Processing". Those works are interested in solving problems such as the transport and storage of large datasets which are produced as the visualisation results. The works of "Adds-on for Grid" are interested in enabling users to steer their applications on the grid. Since it is possible for one work belong to different categories at same time, the sum of proportions accounted by each category is greater than 100%. The "As Visualisation Tools" is also an attracting area, many works are committed to providing new user interfaces to interact with visualisation components. Additionally, such works also addressed on the performance of visualisations. Works belonging to the "Used in Specific Application Area" category only integrates existing computational steering environments or tools with specific applications. They primarily focus on their own scientific areas such as medicine, social science, engineer, etc. The last category is the "Just as a Related Work" group. This group contains works that only use computational steering as an example, a part of the future work and an interesting function of their related works. Although these works do not use computational steering at all, the number can, to some extent, indicate the influence of computational steering.

Figure 2-3 depicts the proportion that different categories account for in each year. It indicates the difference of efforts researchers invested on the quantity level. An interesting time point shown on Figure 2-3 is 2006. It is a turning point of two categories, “Adds-on for Grid” and “Used in Specific Application Area”. In 2004 and 2005, most works attempted to deploy computational steering on grids, and it created an increasing trend during that time. However, since 2006, efforts in this area greatly decreased. By 2013/14, there are few works that still mention integrating grid with computational steering. In contrast, more works begin to integrate existing computational steering environments into their project to facilitate the research in various application areas. At the beginning, the amount of projects considered using computational steering stayed at a low-level during 2004 and 2005. But the situation altered dramatically since 2006. The ratio of papers, which only use but do not further develop existing computational steering tools, rises by about 50 percentages and stays at that level from 2006 to 2014. Moreover, another category which occupies a large scale is the “Just as a Related Work” and its trend is stable. The evidence shows that the sum of “Just as a Related Work” and “Used in Specific Application Area” takes up about 80 percentage of steering works after 2006. Conversely, works that attempt to improve the core functions of computational steering, such as visualisation, steering functions and data movement, remain steady at a low ratio level except that works related to grid computing were once at a high ratio at the beginning but fell down rapidly.

We can gain further insight by combining Figure 2-2 and Figure 2-3. The significant drop on citation number between 2003 and 2004 could be explained by the observation that the cost of learning and implementing computational steering remained too high for general users, especially users who have not studied computer science, to utilise computational steering in practise. This falling trend was temporarily alleviated by the rise of grid computing when researchers attempted to integrate distributed computing resources in a more integrated and usable structure.

However, this promising perspective lasted only for three years. Afterwards, another major decrease in the interest of computational steering happened between 2006 and 2007. From Figure 2-3, it is noticeable that works relevant to grid computing suddenly have a considerable drop. Therefore, the author assumes the drop of citations referring to the computational steering is related to the drop of applying computational steering on grid computing. We assume two reasons for the loss of interest in integrating computational steering with grid computing. Firstly, research challenges such as big data movement and

real-time visualisation of grid computing cannot guarantee a stable user-experience required by computational steering. Additionally, a significant application direction of computational steering on grid is to provide computing resources and steering as an integrated web service. However, the future of the standardization of web services on grid is uncertain. The initially proposed Open Grid Services Infrastructure (OGSI) [138] was replaced by Web Services Resource Framework (WSRF) [41]. However, with an uncertain future of WSRF, it is not convincing for researchers to build a service on unstable fundamental. By communicating with scientists and developers of RealityGrid computational steering, we obtained a confirmation on the influence caused by varying standard of grid computing.

On the other hand, a large amount of works from other scientific areas still attempts to take up the steering technology. This explains the existence of the largely dominant category, “Used in Specific Application Area”. However, limited by the fact that the integration requires dedicated knowledge and efforts to work on intricate HPC structures, grid service standards and steering code annotation, recent works appear to have less impact compared with previous works.

Finally, our analysis can explain the curb between 1999 to 2002 shown by the number of published papers seen in Figure 2-2. We argue that the increasing trend of computational steering motivated by the original requirement of facilitating interactions between users and running simulations became a spent force after 2000. The followed-up major decline in the number of citations at this time supports this argument. However, this decline coincides with the rise of grid computing. Few of works that are dedicated on implementing computational steering on grid environments obtained a great amount of citations. Hence, this argument is also a possible conjecture to explain the largest difference between number of citations and number of papers in 2003.

2.1.5 Discussion

This section summaries concepts of computational steering based on its development and presents a discussion on the future direction of computational steering.

Historical Discussion

In the early phase, the concept of computational steering is blurry. As introduced in section 2.1.1, works at that time were trying to explore the interactivity between running simulations and users. Even though the so-called interactivity is more like a deep monitoring ability,

these primary researches present the original motivation of computational steering which is to get more intensive understanding of simulations and achieve higher work efficiency. However, this initial motivation cannot take the premature computational steering further without addressing the issues of visualisation. Evolved from the animation, visualisation in the early 1990 enhanced the effect of feedback in the interactive simulation and brought new requirements on computational steering. Consequently, computational steering went into a transitional period during which researchers defined the previous interactivity as the tracking ability and coined the term, steering. However, a rose has its thorns. Since basic requirements of computational steering were born within the context of visualisation, researchers saw computational steering as an accessory of visualisation. Moreover, the application area of computational steering was tightly affixed to simulations running on the HPC. As a result, both developments of the visualisation and HPC restrict the scope of computational steering.

At the beginning period, scientists and engineers developed computational steering toolkits and environments based on the assumptions and descriptions of requirements from users. In spite of being applied in various areas, these tools presented the basic design patterns, conceptual and architectural components of computational steering. Through this process, the fundamental nature of computational steering was forming. Although these concepts are still confined in the relation between HPC and visualisation, many researchers began to take high-level views of computational steering and to give it a more general description in which the users and simulations form a feedback loop, and interactions between users and running simulation in the feedback loop can be facilitated by using computational steering. Based on such concepts, researchers defined model exploration, algorithm refinement and performance tuning as three functions of computational steering. In order to realise the steering functions, computational steering needs to cover issues such as data movement in the distributed computing architecture, in-time data movement, in-time feedback and visualisation of simulation results. Furthermore, it also needs to provide users an easy-to-use environment. As a result, the term “computational steering” also indicates the interaction capability between users and simulations.

Consequently, we argue that the development of computational steering always depends on the development of other computer science areas, such as visualisation, supercomputers and grid computing. Although the development of such areas motivates and accelerates the growth of computational steering, their fluctuations and uncertainty can also have negative affections on computational steering. As shown in Figure 2-2 and Figure 2-3, the high

dependency of computational steering on the grid computing makes challenges of grid computing impede the development of computational steering. Although we believe the development of related areas, such as grid, can ultimately facilitate the growth of computational steering, it is still necessary to protect the core concepts of computational steering from technological turbulence in such areas during their ferment period.

The Future and Extrapolation

By understanding and acknowledging the historical pattern of the development of computational steering, the author proposes a new direction for using computational steering.

Based on the development of computational steering, the author acknowledges that instead of making contribution by implementing computational steering on new computing infrastructures and integrating with existing simulation models, another way of making contribution is to find a new application area for computational steering. Utilizing the review of computational steering, the author perceives opportunities to integrate computational steering with computations that have interactions with physical systems.

Since physical systems can include a great amount of uncertainties and dynamics, it is necessary to build interactions between applications and the physical processes being simulated. On the software level, the interaction can delivery significant information that can be leveraged by algorithms to adapt models to represent real states of physical systems. On the hardware level, the interaction can enable dynamical assigning computing resources to support applications on software level. Such interaction requirements have been raised by other research fields, such as Cyber-Physical System (CPS) and Dynamic Data-Driven Application System (DDDAS), as dynamical reorganization and reconfiguration [31, 128]. The required innovative interaction ability in the field of CPS and DDDAS fits exactly into the three functions that can be facilitated by using computational steering: model exploration, algorithm refinement and performance tuning. The rest of this thesis uses DDDAS to illustrate applications that require such interaction ability.

The most significant conceptual difference between the interaction of computational steering and applications of DDDAS is that in computational steering, most interactions are built between computation process and human users instead of the physical world. However, for simulations applied in DDDAS that have enormous parameter space and are required to run constantly, it is not feasible for human users to interact with such simulations. Based on existing works in DDDAS, algorithms have been utilized to realize such interactions.

Darema has claimed that DDDAS requires computational steering to dynamically steer simulations at their execution time by using data collected from the physical world [31]. Under this paradigm, Denham and co-workers utilize a genetic algorithm to steer simulation of forest fires based on dynamic data [34, 35]. Gabrielle proposes the requirement of automatic steering ability for hurricane forecasting [6]. Additionally, a great number of works in the Water Distribution Systems utilize genetic algorithm, Kalman filter, etc. to steer simulations based on sensor data [56, 67]. However, even though steering simulations at the execution time has been raised in applications of DDDAS, in practice, it can be more easily realized as feeding simulations with new inputs, which is similar to the batch mode of running simulations on HPC resources.

On the other hand, as claimed in the CSE, computational steering should "*greatly reduce the time between changes to model parameters and the viewing of the result*" [144]. For an algorithmic steering, it should reduce the time consumed by interacting between algorithms and models. This understanding is also supported by existing works in the field of computational steering. To the author's knowledge, the first person that discussed such steering ability is Vetter [147] who proposed the algorithmic steering to replace human users to make steering decisions. In addition to make automatic steering decisions, steering algorithms take advantage of steering interfaces deployed on the program side to have an efficient communication with running simulations. Similarly, Smith proposed goal-oriented computational steering [130] based on AVS steering environment and Swan used the term "inverse steering" to describe the algorithmic steering [134] based on CUMULVS.

As a result, current works in DDDAS has started to notice the importance of computational steering. Their concepts of steering only focused on realizing interactions between two objects. Nevertheless, they neglected the consideration that the steering interaction can also be considered as a process of facilitating the original batch interaction. The author believes that by proposing applications of DDDAS as a new application area, the achievements made by computational steering can be applied to required interactions between dynamic data, algorithms and simulations in DDDAS.

2.2 Dynamic Data-Driven Application Systems

Based on observations from the physical system, a system that utilises models to simulate, predict and analyse a large-scale real-time physical system is described as Dynamic Data-

Driven Application System (DDDAS) in the Chapter 1. The research area indicated by DDDAS has arisen in the recent decade. As initially proposed by Darema [31], "*Dynamic Data Driven Application Systems entails the ability to incorporate additional data into an executing application - this data can be archival or collected on-line; and in reverse, the ability of applications to dynamically steer the measurement process*", it covers a broader application area than what is studied in this thesis. This chapter introduces the specific sub-area of DDDAS that is addressed in this thesis by having an overview of the broad definition of DDDAS. After this section, the term DDDAS will only indicate the sub-area focused by this thesis.

2.2.1 Three Components Areas of Dynamic Data-Driven Application System

This section uses the taxonomy defined in the work [31] and divides the application area of DDDAS into applications, mathematical algorithms and platforms and infrastructures. By discussing these areas, we locate those areas to which computational steering can make contribution.

Applications

DDDAS requires its applications to incorporate dynamic data at execution time and be steered by such data. Applications typically correspond to functions such as monitoring, prediction and other methods that can help to analyse physical systems. Considering the scale and stochastic behaviour of physical systems, it is not feasible to physically monitor without uncertainties. Hence, monitoring on this level means conducting real-time simulations by interpolating dynamic information collected from physical systems. In terms of prediction, researchers utilise results of simulations that represent current states to estimate future states of the physical systems. Specific analysis such as precautions of anomalies and disasters, what-if simulations and decisions support tools can be conducted based on the monitoring and prediction as well.

Since this project specifically takes Water Distribution System (WDS) as an example of the physical systems, we pay a special attention on simulations of the WDS. Khan [81] has discussed using on-line hydraulic data collected from sensor networks to provide a simulation that can track and predict states of water networks. Haines developed a decision support tools in which dynamic information is incorporated into simulations [56]. Hutton

utilised dynamic information to estimate current and predict future rainfall-runoff situations [117]. Preis also takes sensor data to present current and future states of a real water network in Singapore [118]. In addition to works in the WDS, applications of DDDAS are also widely used in large-scale systems such as forest fire spread prediction [35] and hurricane prediction [6].

Mathematical Algorithms

Mathematical algorithms are critical in terms of incorporating dynamic information into applications of DDDAS. Algorithms need to interpolate incomplete data, estimate uncertainties of physical systems and even handle emergencies and anomalies happening in physical systems [31]. In practise, many works develop such algorithms as calibration processes that utilise dynamic data to rectify models used by simulations. Additionally, the development of applications is also integrated with the development of such algorithms so that the calibration results can be directly used by real-time applications, and the calibration algorithms are considered as data assimilation algorithms. The result of such integration was first raised in the area of weather forecast as the data assimilation. In the application areas discussed above, algorithms such as genetic algorithm [35, 81, 118], ensemble Kalman filter [158] and 4DVAR [121] are used to realise the calibration in the data assimilation.

Platforms and Infrastructures

To support applications and algorithms, computing platforms and infrastructures are needed by DDDAS. They must meet the real-time requirements in dynamic data storage and transport, dynamic computing resource management, etc. For a specific application area, frameworks and architectures that contain communication protocols and service standards also need to be developed. One significant feature of such platforms and infrastructures is that they must have the flexibility to adapt according to dynamic changes of physical processes.

2.2.2 Algorithms of Data Assimilation

Since the main hypothesis of this thesis is that computational steering can facilitate the Data Assimilation (DA) algorithms, and other interaction abilities required by DDDAS are derived from the interaction between algorithms and dynamic data, the this section specifically discuss the general workflow of such algorithms.

Numerical modelling has been largely deployed to describe states of physical systems. However, the numerical model is typically confronted with uncertainties that are raised by dynamic states of the physical system. To derive high-level applications, such as monitoring and prediction, from a numerical model, it is crucial to calibrate parameters and initial states of models to match with the varying states of the physical system. DA describes a collection of algorithms that incorporates observation data into models to derive uncertain initial states so as to improve the prediction [150] and monitoring of physical systems. It was first used in the area of meteorology [40], and as advances in data collection methods, it has been widely applied in hydrology, hurricane and forest fire prediction [95, 142].

In a general applications of data assimilation, the numerical model ($\widehat{H}$), which represents the relationship between input model parameters ($\hat{X}$) and model states ($\hat{Y}$), is

$$\hat{Y} = \widehat{H}(\hat{X}). \tag{1}$$

However, one main issue of using a numerical model to represent the real system is the uncertainty. Hence, we use the symbol $\wedge$ to indicate that the mode, its parameters, and its states do not equal to their counterparts in the real systems. In terms of the model $\widehat{H}$, it is not possible to perfectly represent a physical system by using a numerical model. In general, the numerical model is limited by the large scale of DDDAS system. For example, details of physical systems are always generalised out in numerical models, and structures of the real world are normally skeletonised and abstracted to facilitate the model design and reduce the computational expense. Additionally, even if model $\widehat{H}$ can manage to signify physical systems precisely, the initial parameters, $\hat{X}$, also introduce uncertainties into the model due to its corresponding states of physical systems can change as time passes.

Besides the Equation (1) which maps initial parameters to states of physical systems, another basic function of the data assimilation is to estimate initial parameters at the next simulation time step $\hat{X}_t^e$ based on known model states $\hat{X}_{t-1}^a$, that is

$$\hat{X}_t^e = \hat{P}(\hat{X}_{t-1}^a). \tag{2}$$

$\hat{P}$ is the method used to predict initial parameters. Exactly as for the $\widehat{H}$, it cannot flawlessly predict the future initial states. In $\hat{X}_t^e$, the symbol e indicates the initial parameters at time step t is estimated from calibrated parameters, $\hat{X}^a$, at the previous time step, $t-1$.

By using $\hat{H}$ and $\hat{P}$ as tools, the data assimilation incorporates observation data, $\ddot{Y}$, to calibrate the $\hat{X}^e$, and the calibration result is $\hat{X}^a$. If we use F to denote the algorithm used to calibrate the $\hat{X}^e$, then calibrated states can be presented as

$$\hat{X}_t^a = \mathrm{F}(\mathrm{H}(\hat{X}_t^e), \ddot{Y}_t). \quad (3)$$

in which the calibration algorithm compares the difference between $\ddot{Y}$ and the predicted system states $\hat{Y}^e$. Typically, the $\ddot{Y}$ is limited by methods of collecting the observation data. Hence, the $\ddot{Y}$ can only represent observational states in the subset of the entire and computed states of the system, $\hat{Y}^e$. The realization of $F(\mathrm{H}(\hat{X}_t^e), \ddot{Y})$ depends on particular algorithms used in the calibration, and it will be discussed in the section: Genetic Algorithm – as an Example of Calibration . As a result, the three basic equations of the data assimilation form an iteration, one example of the iteration is the Prediction and Calibration Scheme discussed in the following subsection.

Prediction and Calibration Scheme

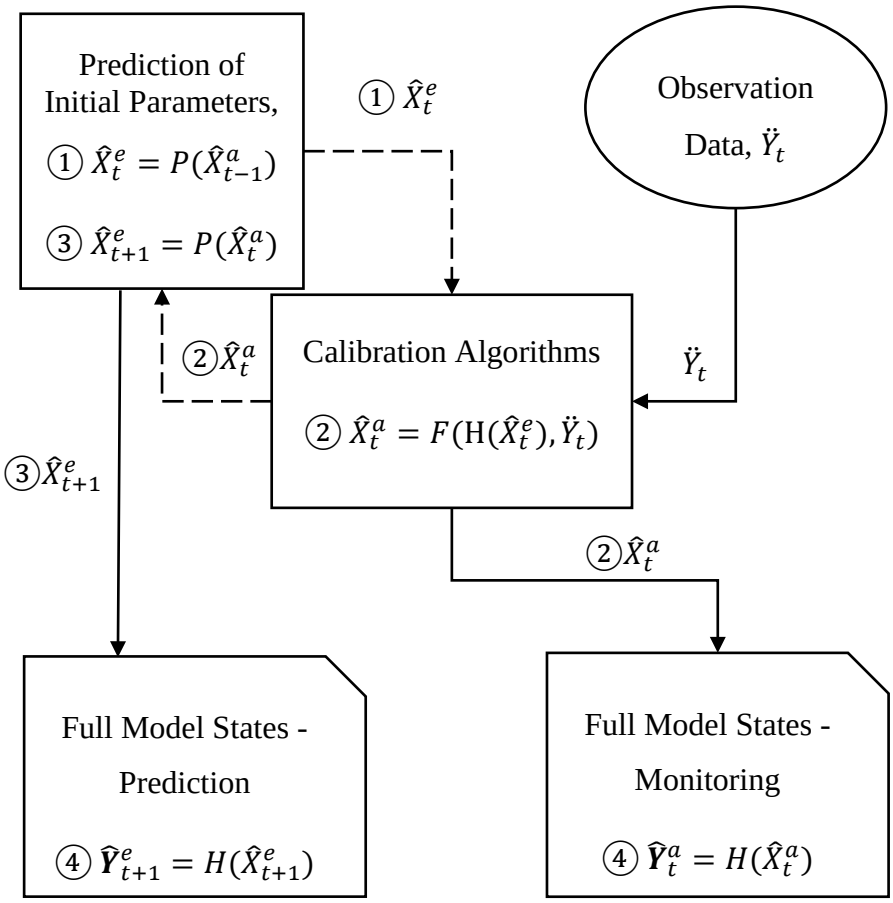


Figure 2-4 The Prediction and Correction Scheme of the Data Assimilation: $\hat{X}$ denotes initial states that are used to drive models. $\hat{X}_t^e$ and $\hat{X}_t^a$ indicate the predicted and calibrated states at time step t. H() is the model

that is driven by $\hat{X}$ to generate observations Y. $\ddot{Y}_t, \hat{\boldsymbol{Y}}_t^e$ *and* $\hat{\boldsymbol{Y}}_t^a$ indicate observations collected from the physical world, predicted observations and calibrated observations generated by using H() and $\hat{X}$. Due to observations collected from real world cannot cover an entire physical system, $\hat{\boldsymbol{Y}}_t^e$ *and* $\hat{\boldsymbol{Y}}_t^a$ covers more states than $\ddot{Y}_t$. This figure uses circled numbers to illustrate one iteration of the loop. This iteration needs to calibrate states at time step t based on calibrated states at time step t-1. In the iteration, calibration algorithm $F()$ creates ensembles of multiple sets of initial states derived from predicted states $\hat{X}_t^e$ and compares results of H() driven by such initial states with $\ddot{Y}_t$. The initial states set that leads to a minimum difference between H() results and $\ddot{Y}_t$ is selected as the calibration result. Then it is used by monitoring to present current states of systems and also leveraged by prediction component to estimate future initial states.

Figure 2-4 depicts the prediction and calibration scheme which is widely utilised in the area of data assimilation area [67, 91]. The core iteration is shown by dashed lines between Prediction of Initial Parameters and Calibration Algorithms. When the observations $\ddot{Y}_t$ arrive, the Calibration Algorithm adapts the predicted states $\hat{X}_t^e$ to represent the physical world. As the result of the calibration, $\hat{X}_t^a$ is utilised as the input to predict the $\hat{X}_{t+1}^e$. According to the definition of the control theory, it can be defined as a feedback loop.

Since the calibration results $\hat{X}_t^a$ increases the confidence on the initial parameters, the model driven by the calibrated parameters is utilised to realise the monitoring. But its accuracy highly depends on the coverage of the observation data $\ddot{Y}$ and the implementation of the calibration algorithm $F()$. In terms of the observation data coverage, the calibration results can only reflect the system attributes carried by $\ddot{Y}$. Hence, the bias introduced by partially calibrating the model is embodied by model, H, when it maps the calibration results to the entire DDDAS. In addition, numerous algorithms have been implemented as the calibration method. Their performance differs according to application area and evaluation metrics. Even though for the same algorithm and problem domain, the accuracy is also contingent upon the configuration of algorithms.

Furthermore, the term monitoring may refer to the data extracted from the sensor reading directly. This type of monitoring has a small delay from obtaining data and displaying them on simulations. However, as limited by the amount of applied sensors, the sensor reading cannot often cover the monitored area sufficiently. Hence, as discussed in the application area of DDDAS, DA is always required by high-level applications of DDDAS. Consequently, since this work focuses on the calibration algorithm, only the monitoring that requires DA is addressed as an example of applications of DDDAS in the rest of this thesis.

Genetic Algorithm – as an Example of Calibration Algorithms

There is a great family of algorithms that can be utilised to conduct calibration. For example, there is a continuing argument about the choice between 4D-VAR and ensemble Kalman

filter in the area of weather forecast [75, 92], and recently there is a requirement to use the two algorithms together [27]. Moreover, in the WDS, which is the use case application area of this thesis, ensemble Kalman filter [26], genetic algorithm [117], particle filter[125], weighted least squares [76], etc. have been applied as calibration algorithms. The selection of algorithms typically depends on model linearity, computing cost and difficulty of implementation.

This work has received help and support from water companies. Thus, we use the Water Distribution System (WDS) as the use case of the Dynamic-Data Driven Application System (DDDAS). We build our project based on a Knowledge Transfer Partnership project between the Water Research Centre and the University of Manchester [82]. In previous software implementations of the collaborative project, the Genetic Algorithm (GA) has been applied as the calibration algorithm. Thus, we need to use GA as an example of calibration algorithms. Additionally, although this work does not claim contributions on calibration algorithms, demonstrating the project built with GA provides sufficient features to test the concepts and implementation of our innovations in the steering process. To demonstrate the development based on GA can be applied in projects using other calibration algorithms, we have an analysis of common features of calibration algorithms.

As indicated by Kalnay, ensemble methods and variational methods provide an option for scientists to realise data assimilation [75]. Additionally stated by Hutton that ensemble methods, especially ensemble Kalman filters and its variations, account for the majority of implementations of data assimilation algorithms in this developing application area, WDS [67]. Furthermore, GAs are also widely utilised as calibration algorithms to adapt water models to dynamic information obtained from the physical world [47, 83, 120, 124, 127]. Therefore, we argue that the method used to integrate computational steering with GA can also be applied to other calibration algorithms.

```
// using the predicted initial states vector X̂^e as the first individual
// to fill up a population by adding perturbations; each state in
// the states vector is considered as a gene segment.
while (current individual number < desired population size){
        initializing new individual;
        inserting new individual into population
}

generation = 0;
while (generation < max generations){
        // evaluating a generation of individuals by using them as
        // dynamic parameters of the model H() and run simulations based
        // on such model; at last, simulations results are compared
        // with observations Ÿ, and the difference indicate the accuracy
        // of estimated initial states represented by individuals.
        evaluating new population;

        if (best incumbent individual is better than requirement) exit;

        // using competitive individuals to generate the next generation
        // of individuals.
        selecting competitive individuals;

        // randomly mutating states in individuals
        mutation operations;

        // randomly crossover states between individuals
        crossover operations;

        generation = generation + 1;
}
```

Figure 2-5 Pseudocode of the Genetic Algorithm: It is better to read the pseudocode with Figure 2-4. Population size and maximum generation number are parameters that can be adjusted by developers based on specific requirements. On the premises of simplifying implementation, this pseudocode utilises elitism selection, Gaussian mutation and simulated binary crossover [32].

We begin our argument by introducing the GA. Compared with other calibration algorithms, GA has less requirements on model and problems. However, the reason for scientists to think GA is too computational expensive for real-time calibration [67] is the large computational workload required by the evaluation function. This is because that, during the evaluation, GA needs to run a great number of simulations. The number depends on the population size and max generation number. As a result, the product of population size multiplied by max generation number can always lead to hundreds of thousands of simulation executions.

Since computational steering can facilitate the process of altering parameters of models that are used by simulations, the author hypothesizes that the evaluation and its related

interactions between simulations and GA can be improved by using computational steering. Consequently, to present the project developed for GA so that it can also be applied to other calibration algorithms, the process of large-scale parameter alteration needs to be identified in other algorithms

Initially, the largely utilised ensemble Kalman filter is based on Monte Carlo methods that acts similarly to GA except that the perturbations are added by using values from a probability distribution rather than mutation and crossover methods. Additionally, similar algorithms that are derived from ensemble Kalman filter such as ensemble adjusted Kalman filter [7] fit into our project as well. Consequently, we argue that a big calibration algorithm family based on ensemble Kalman filter can be integrated with our work in the future. Furthermore, as indicated by the term ensemble, ensemble assimilation methods typically require a number of ensembles to estimate model errors. As the error estimation is based on adding perturbations on $\hat{X}^e$, computational steering is, theoretically, able to make contributions on all ensemble algorithms that have the perturbation addition process, although the practical effect depends on the size of the ensemble.

Returning to GA, as it is simple to be implemented, has less limitations on models, shares common features with a large portion of other assimilation algorithms and is also the method used in the collaboration project with water companies, we assume that it is reasonable to use the GA as an example of the calibration algorithm.

2.3 High-Performance Computing Resources

Both applications of DDDAS and simulations steered by using computational steering require High-Performance Computing resources (HPCs). Therefore, to establish the hypothesis, it is important to ground our work on real HPC architectures. As a result, this section discusses architectures of different types of HPCs and we select the IBM Blue Gene/Q, Amazon Cloud and CNGrid as examples of supercomputers, cloud and grid.

2.3.1 IBM Blue Gene/Q System

Blue Gene series systems are designed to reach operating speeds in the PFLOPS (petaFLOPS) range and are specially optimized for energy efficiency. There are three generations of Blue Gene systems, the Blue Gene/L, Blue Gene/P and Blue Gene/Q. This

section only introduces the Blue Gene/Q system of which two implementations are ranked at 4th and 6th on the TOP500 [2] in June 2016.

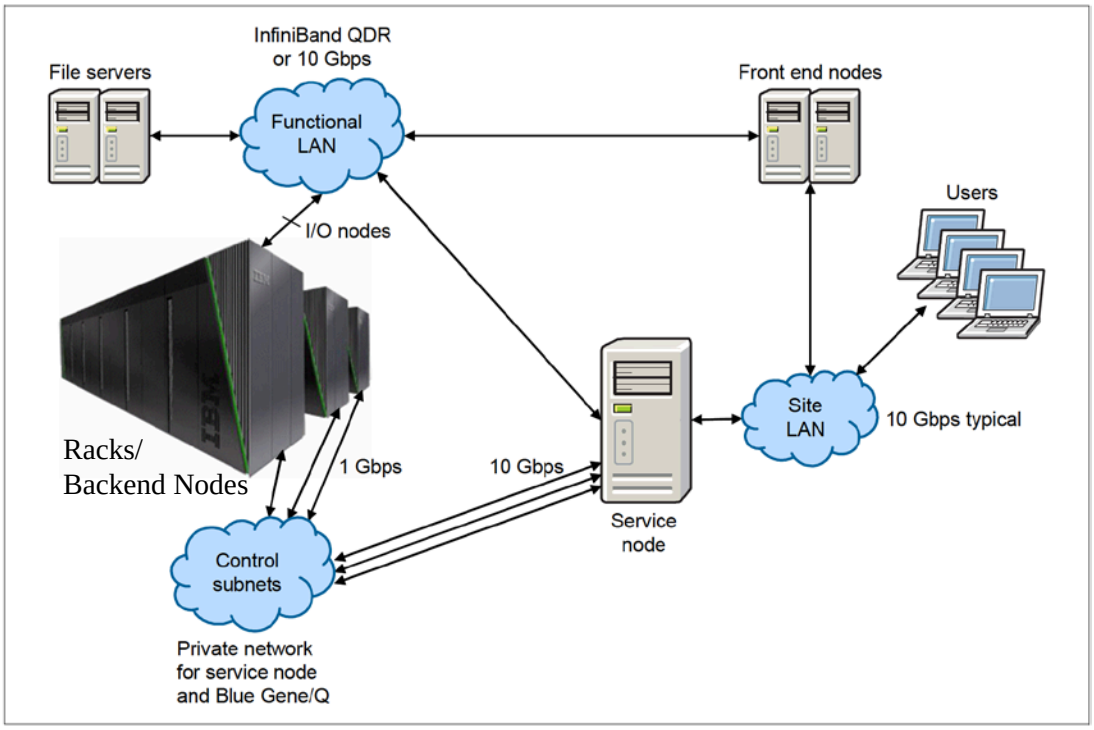


Figure 2-6 Blue Gene/Q Architecture [53]: Arrows among components indicate their communications. The racks shown on the left are computing resources, it is also named as Backend nodes. The File servers provide storages for other components. The Service node checks administration of Users. The FNs provide users the interface to interact with File servers and Back end nodes.

Figure 2-6 depicts the architecture of the IBM Blue Gene/Q system. The pictures of physical racks shown on the left present the essential component to provide the computing resources. It is also named as the Backend Node (BN) since users cannot directly interact with it. A rack is densely assembled by one or two compute midplanes and one I/O drawer. 16 node boards constitute one compute midplane and each node board contains 32 compute cards. Finally, one compute card is made of one 16 cores chip and 16 GB DDR3 memory. At a maximum, the system can be scaled to 512 racks.

In addition to the primary component, the system comprises: service node, Frontend Node (FN), storage system and the communication subsystem. The service node is used for administration and management of the Blue Gene/Q system. The FNs, which are also referred as the login nodes, are the interface for external users to access the computing and storage resources.

In a regular workflow, users login to the FN to prepare their programs with the software provided by the system. Since the Blue Gene/Q is a diskless [53] system, the File server is

used as the storage to keep users' programs and data. After being edited, compiled and debugged, programs of users can be submitted to a job scheduler, such as the LoadLeveler scheduler, in which programs are queued based on the their priority. Finally, results are transferred from the backend compute nodes to the file server so that users can post-process results via the FN.

2.3.2 Amazon Cloud

Rather than purchasing and maintaining a dedicated supercomputer, cloud computing provides software and hardware of HPCs as services to end users. On the hardware level, cloud computing offers users great flexibility so that they are able to tailor computing infrastructures in minutes or even in seconds. Hence, this flexibility eliminates an up-front commitment by cloud users, and moreover, it provides the ability of paying for usage on demand [8, 9]. This service is named as Infrastructure as a Service (IaaS). On the software level, cloud computing divides the service into Platform as a Service (PaaS) and Software as a Service (SaaS). The PaaS sheds complexities of infrastructure and provides tools and environments for users to conveniently deploy applications. In addition, the SaaS tends to separate the possession of a software from its use [139]. By providing software as a service, end users can focus more on using the software rather than installing and maintaining it. Moreover, users can also access the software service via thin clients. For example, depending on the development of the WEB technology, the software services on the cloud can be based on Web services, and on the client end, users only need a web browser to utilize the SaaS [21].

Cloud computing has been provided as a service by a number of companies such as Amazon Elastic Compute Cloud, Google AppEngine, Microsoft Azure, etc.. Comparing details of different cloud services is beyond the topic of this thesis. According to the market research done by Magic Quadrant, Amazon Cloud service is ranked the first cloud service provider in Q2 2016 [136]. Hence, this work uses the Amazon Cloud as an example of large-scale computing environment on the cloud.

Figure 2-7 displays a guidance provided by the Amazon Web Service (AWS) Architecture Centre which aims at helping general users to build highly scalable and reliable systems in the AWS cloud. Hence, we use this architecture as an example of general architectures of large scale clusters implemented on the AWS cloud.

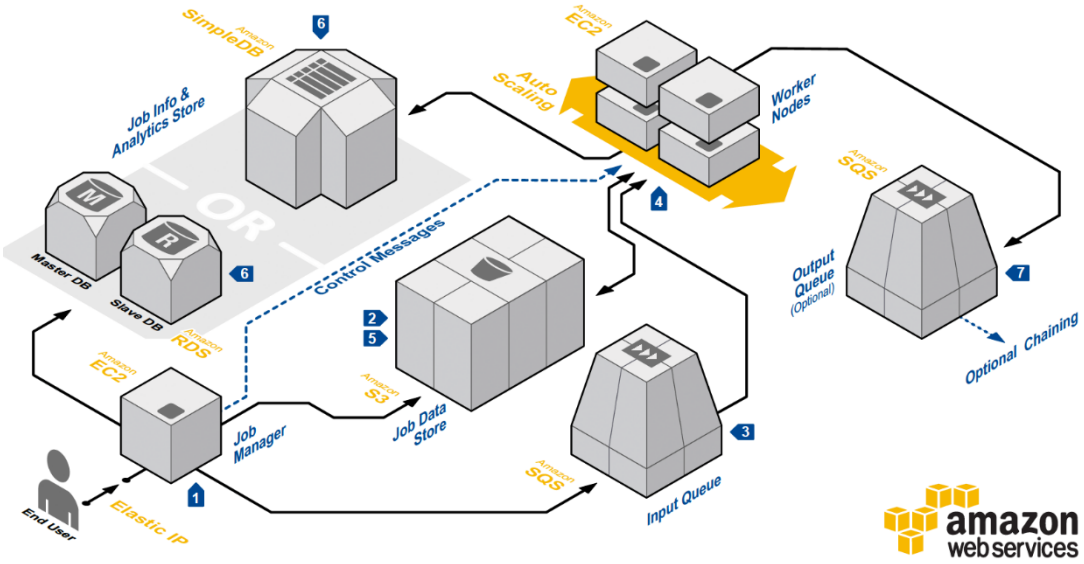


Figure 2-7 Amazon Cloud Batch Processing [1]: This architecture is similar as that of IBM Blue Gene/Q system. Arrows indicate communication between components. The EC2s on the up right corner are computing resources users can dynamically apply or release based on their demands. It can be considered as the Compute/ Back end Nodes. Users also use the EC2 to realize the FN on which users manage their jobs. The Amazon SimpleDB, Amazon RDS and Amazon S3 can store results obtained from computing EC2s. The difference is on read/write speed and data format. Additionally, queue services can be applied to job and results management if this architecture is shared with multiple users.

The Amazon EC2 stands for Amazon Elastic Computing node which is also referred as the instance. Instance is similar to a virtual computer, and users can apply a number of instances to meet their requirements on the computing power. In Figure 2-7, one EC2, on the bottom left corner, is used as the Job Manager for users to install applications and manage their jobs. On the upper right corner, a quantity of EC2s are used as Worker Nodes to execute the computing task. The amount of worker EC2 instances can be scaled automatically according to requirements of users. Furthermore, the program and input data of users are stored in the Amazon Simple Storage Service (S3) which can also be used to store the output from Worker Nodes. Alternatively, results of tasks can be transferred to storages that are shown as the Amazon SimpleDB or Amazon Relational Database Service (RDS), and users can manage their tasks by using Amazon Simple Queue Service (SQS) so that tasks can be queued up and executed sequentially.

2.3.3 China National Grid

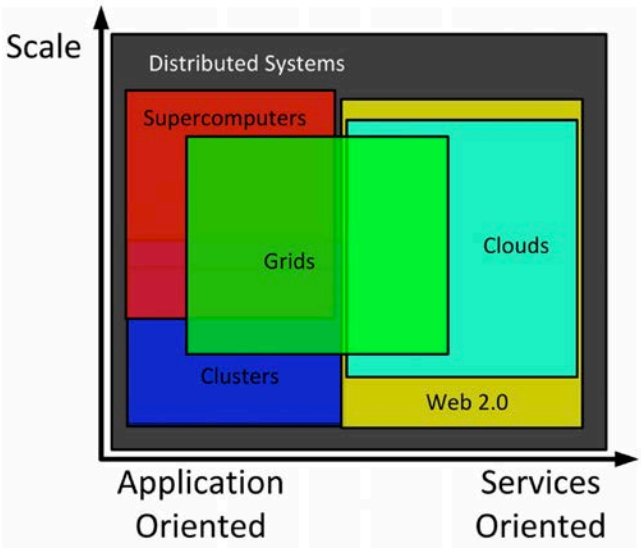


Figure 2-8 Grids and Clouds Overview: Grid computing system has overlaps with supercomputers, clusters and clouds. This work does not differentiate clusters and supercomputers.

Grid computing is considered as a branch of the distributed computing [49]. There is little consensus on how to define the difference among grid computing, cloud computing and supercomputers. This thesis agrees with Ian Foster who believes grid computing has overlapping features with supercomputers, cloud computing and even clusters as shown in Figure 2-8 [43]. The main reason for the grid computing shares common features with the others is that grid computing aims at "*enabling resource sharing and coordinated problem solving in dynamic, multi-institutional virtual organizations*" [42]. The virtual organizations are defined as flexible, secure, coordinated computing resource in a grid. Therefore, a supercomputer, a cloud computing resource, or a cluster which is able to be coordinated under a specific grid can all be used in a virtual organization. As a result, the coordination is the primary concept differs the grid computing from others.

Grid computing environment has several names in different areas. In China, it is called High-Performance Computing Environment (HPCE), but in most European areas, it is named as the e-Science Environment. Alternatively, a synonymous term, cyberinfrastructure, is used by the US National Science Foundation (NSF). Since the HPCE is designed by referring to the UK's e-Science projects [61], and collaborated with US Cyberinfrastructure projects such as Globus [50], this project uses the HPCE as an example of the grid computing.

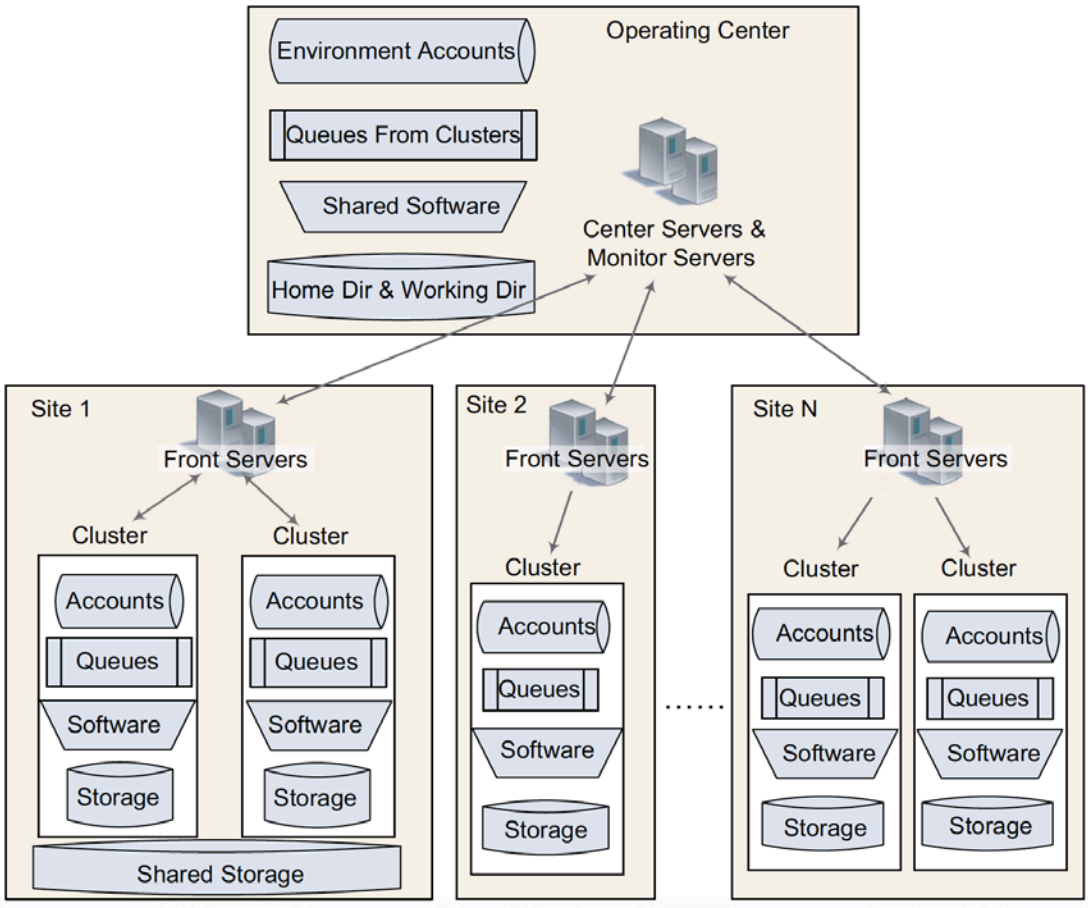


Figure 2-9 The CNGrid architecture [155]: The Center Servers and Monitor Servers bridge users and HPC sites. One user holds two accounts, one for the Grid service and one for their accessible sites. Each site is a computing resource provider who either registers supercomputer or cloud computing resources to the Operating Center. Thus, the structure of each site is similar to that of supercomputers and cloud. The difference is, instead of login on Front End/ Front Servers, users only need to submit jobs on the Operating Center which selects optimal site to execute their jobs.

The China National Grid is a realisation of the HPCE and its architecture is shown in Figure 2-9 [43]. On the top box of this figure, the environment account is issued to external users to login to the operating center. The center hides computing resources details from external users, thus, users can manage their programs and tasks as they are run on clusters and supercomputers. The difference is that their jobs are distributed to the site selected by the Center/Monitor Servers. A Front Server is applied to a Site to coordinate with the Center Server. In order to be a Site in the Grid, the Front Server must update its real-time usage status for the Center Servers. Moreover, the architecture and configuration of a Site must match with the coordination protocol required by the specific grid. The design of a common coordination protocol is still a challenging research area since it is related to the collaboration of different utilities of which the HPC systems are designed dedicatedly. Consequently, the grid computing transfers complexities and difficulties of utilizing HPCs from end users to system engineers and designers.

2.3.4 Discussion

Different types of HPCs provide various options for users. Supercomputers, cloud and grid computing are three types of HPCs that are widely in use. Thereafter, this project should have the compatibility to be installed on these three types of HPCs.

Since computational steering focuses on interactions between users and simulations running on computing resources, it is crucial to discuss the method of building interaction to the computing resources on different types of HPCs. From the above introduction, supercomputers, cloud and grid have similarities in their architectures in terms of communicating with computing resources. In IBM Blue/G supercomputer, the computing resources, BNs are connected to FNs and Service nodes, in which the FNs provide the interface for the supercomputer to connect to the outside network. Additionally, on the cloud, the Job Manager hides the computing resources, Worker Nodes, from the external network. Users can only access the cloud through the Job Manager. Finally, on the grid, each site has an interface, namely Front Servers, that bridges outside communications to the internal computing resources. One additional function of the Front Server, compared with FNs and Job Manager, is the negotiation with the grid Operating Center.

As a result, communications with all computing resources on the three types of HPCs must be relayed by a front node/manager/server. For grid computing, it may also need to be relayed by an additional Center Server on the Operating Center. Consequently, this similarity provides a template to develop a computational steering project that has the compatibility to be applied to general HPC architectures.

However, even though HPC architectures have such similarity, the complexity raised by above architectures still hinder regular users to utilize their great computing power. As pointed out by Xu [155], an important open question is why technology for global networked supercomputing has not yet become as widespread as the Internet or Web. One common complaint is that modern HPCE is too complex and heavy to learn and use.

In terms of providing computational steering for HPC users. The implementation of computational steering on supercomputers and cloud maybe impeded by the security requirements on interactions between the external network and the internal network of the computing resources. Especially for a dedicated computing resource, users do not have the authority to implement the steering communication, and the cost of applying a steering

implementation outweighs the potential benefits. Additionally, the issue of realizing the steering interaction is even worse on the grid computing systems. As discussed in 2.3.3, grid does not allow users to interact with FNs on a resource provider. The Center Servers build one extra wall between users and the BNs.

This situation can be improved by providing computational steering as a SaaS. SaaS is typically based on PaaS in which the software service depends on a reliable platform service. However, the development of computational steering service always binds to specific grid. As a rapid developing area, grid cannot guarantee a stable platform service for computational steering. One example is the unstable standardization of web service popularized by big companies such as the WSRF. It initially replaced the original Open Grid Services Infrastructure (OGSI). However, it required a plethora of Web Service standards and it is now argued that the Web Service style of distributed computing should be replaced by using the Representational State Transfer (REST) architectural style. Therefore, since computational steering has to keep adapting to different architecture styles, it cannot be provided as a stable service.

Moreover, the coming version of HPCE 3.0 will be a human-cyber-physical ternary computing [155]. This thesis believes computational steering will play an important part in future utilization of HPCs since it facilitates the interaction with running programs. This interacting ability is crucial to the human-cyber-physical ternary computing since both human and physical systems can raise dynamic and uncertain requirements on tasks running on HPCs.

In a conclusion, different types of HPC architectures provide common features that can be utilized to realize communication with computing resources. However, the development of computational steering is restricted by the complexity and uncertain environments of HPC infrastructures as well. Based on requirements of future HPC applications, this thesis develops a computational steering architecture that can be developed to tackle existing issues by being specific to common features of HPC architectures, instead of being dedicated on specific HPC infrastructure. Existing HPC providers, such as Amazon cloud and Microsoft Azure, can provide the required HPC as a service to evaluate the computational steering architecture.

Chapter 3 Real-Time Hybrid Data-Driven Computational Steering

Section 1.1 has discussed the gap between the large scale Dynamic Data-Driven Application System (DDDAS) and computational steering. To fill this gap, this chapter proposes a new concept, namely Hybrid Computational Steering, which consists of the Dynamic Data-Driven Computational Steering (DDDCS) and Human Driven Computational Steering (HDCS).

Long running simulations of DDDAS are based on models that encompass a large amount of dynamics and uncertainties. Calibrating such models includes exploring a large parameter space and requires the calibration to run constantly. Because of the size of parameter space and the required continuity, algorithms are widely used to conduct the calibration. The interaction between algorithms and models forms a feedback loop that can be facilitated using computational steering. However, many existing projects have considered human users as the sole engine of computational steering for decades. Hence, the first problem raised in integrating computational steering with DDDAS is to utilise an algorithm to drive the steering.

Additionally, even though algorithms can handle the calibration of DDDAS, users can have dynamic requirements on many high-level functions, such as monitoring[11] and prediction, in which algorithms have not yet been proved to be reliable and efficient enough to meet users' requirements. As a result, human intelligence is still required in such systems. However, most existing works in DDDAS only consider human users in the offline functions such as algorithm tuning. Hence, the second problem that needs to be tackled by this chapter is to find what interactions between human users and computations can be facilitated by using computational steering in the context of steering a DDDAS.

In the rest of this chapter, DDDCS and HDCS are discussed based on our review of computational steering (in Chapter 2) to tackle the two research problems. Section 3.1 discusses the aim of the hybrid computational steering and usage of both DDDCS and HDCS. Afterwards, based on steering usages, algorithms and human users are discussed as steerers

[11] The monitoring function can be realised by real-time simulations that keep representing states of physical systems. Such monitoring has been introduced in Chapter 2.

of DDDCS and HDCS in section 3.2. This discussion is followed by steering targets introduced in section 3.3, and steering loops depicted in section 3.4 and 3.5.

3.1 The Aim and Usage of Hybrid Computational Steering

This thesis assumes that computational steering can be seen as a tool that increases interaction speed of two objects. The objects can be pairings of simulations and data-streams or simulations and human users. Consequently, even though this thesis addresses the hybrid computational steering in the context of Dynamic Data-Driven Application Systems (DDDAS), its aim is same as the conventional computational steering.

Parker et al. [111] summaries that the computational steering can be used in model exploration, algorithm refinement and performance tuning. Since this summary has been widely cited in works in the area of computational steering, the rest of this section discusses usages of the proposed hybrid computational steering in the context of DDDAS.

3.1.1 Using Computational Steering in the Calibration of Dynamic Data-Driven Application Systems

As this thesis is addressed in the context of DDDAS, it is necessary to discuss what processes of DDDAS can utilise computational steering. While the concept of DDDAS is suitable for a great number of application areas which can have significantly diverse functions, to present the steering ability in such a sizeable and complex system, a common and crucial steering goal is required. Based on the background study discussed in Chapter 2, this thesis believes that studying, analysing and predicting physical systems are the ultimate objectives of DDDAS applications. However, uncertainties of physical systems and uncertain reliability of numerical models impede the achievement of these objectives. Therefore, a significant and appealing challenge related to the common steering aim is exploring uncertain parameters of simulation models to search parameter spaces for what can represent corresponding states of physical systems. Typically, the evaluation of such processes is based on dynamic observations on physical systems. The exploration process is generally termed as calibration and using the results of calibration to conduct high-level functions such as monitoring and predictions, are named as Data Assimilation (DA). We have discussed that the calibration of DA is considered as the interface to apply computational steering on

simulations that represent physical systems. Hence, this thesis takes the model exploration as the first usage of computational steering in DDDAS.

The model exploration in conventional computational steering can be further divided into goal-oriented exploration and variation-oriented exploration. Goal-oriented exploration is similar to driving a car with specific directions and destinations in mind. The steering operation is determined based on traffic conditions. In contrast, the variation-oriented steering is more similar to exploring a city by a driver and being curious about the view around the next corner. The steering operation is determined by the area of interests, namely what-if scenarios. For the goal-oriented exploration, each steering operation can be evaluated by measuring the distance between steered simulation states and expected simulation states. However, for the variation-oriented exploration, the steering results are evaluated subjectively by human users.

Therefore, by taking DDDAS application context into consideration, the calibration can be taken as analogous to the goal-oriented exploration in which calibration algorithms use the dynamic observations of physical world as the goal and explores model parameters to simulate similar states to the observations. Additionally, the objective functions of the calibration process can be used to evaluate the exploration results. On the other hand, it is challenging to use algorithms to automatically define the areas of interests for researchers and engineers. Hence, it is difficult to determine values of the variation-oriented exploration by algorithms. Consequently, a calibration process of DDDAS belongs to goal-oriented model exploration giving the summarised application areas in the context computational steering, and the real-time computations of a calibration process raises a problem that can be tackled by using computational steering to increase the speed of interactions between calibration algorithms and calibrated simulations.

3.1.2 Using Computational Steering on the Calibration of Dynamic Data-Driven Application Systems

Another challenge in the calibration process of DDDAS is to provide the ability to configure algorithms to adapt to dynamic events happening in physical systems. Even though existing calibration algorithms, such as Genetic Algorithm (GA), have the self-adaption abilities to change specific methods based on dynamic observations, it cannot meet all requirements of users. For example, for situations such as emergencies and anomalies, it is crucial to adapt algorithms with particular targeted methods. However, automatic algorithms adaption is still

a challenging research area. As discussed in Chapter 2, many existing applications of DDDAS only consider general calibration algorithms, and challenges in applying automatic algorithm adaption create the research gap in adapting calibration algorithms to cope with uncertain situations in physical world.

This work believes an alternative method is to build interactions between algorithms and human users. The argument of the importance of human users has been discussed in Chapter 1. Researchers, such as Salhi [126], believe that the ultimate goal of computational steering and autonomic computing system is to exclude humans in the control loop. However, by using the current technologies, human intelligence is crucial to analyse complex systems. As stated by Lin et al., "*the self-regulation and separation of concerns provide human beings with the ability to concentrate on high-level objectives without having to micro-manage the specific details involved* [89]." This thesis believes that the first use of computational steering has raised a promising goal-oriented model exploration method to tackle the "specific details" in DDDAS by algorithms. Hence, the human users can play another role by focusing on high-level steering, and the aim of HDCS is to incorporate human users into the real-time workflow of DDDAS, and facilitate interactions between users and DDDAS applications.

If we assume that real-time functions that utilise the calibration results can provide human users high-level perspectives of DDDAS applications, it is reasonable for us to assume that human users need to alter parameters of calibrations to adjust performance of the high-level functions. However, because the interaction between calibration algorithms and human users is an underdeveloped area, most calibration algorithms used in DDDAS are static and pre-tuned in offline mode. This thesis proposes that computational steering can be used to facilitate the interaction between human users and algorithms; hence, it can enable human users to alter parameters of algorithms in on-line mode. As a result, the usage of HDCS in this thesis is in altering parameters of algorithms. Corresponding to the three categorisations, it is similar to the algorithm refinement. However, the difference is that instead of searching an optimal configuration of algorithm, the HDCS enable users to steer algorithms based on their requirements on the high-level functions. Thus, we name the usage of HDCS as algorithm adaption.

For an example of the algorithm adaption, the calibration algorithm requires a large amount of time to conduct a fine-grained calibration. However, in regular situations, users may not

need such accuracy on simulations; hence, they can steer the calibration algorithm to generate a less accurate calibration result with less time consumption. On the other hand, users can also increase the calibration accuracy on a specific area of simulations by steering the calibration algorithm to take more dynamic information and wait for a longer time.

Moreover, the usage of this type of computational steering also specifically indicates the aim in this thesis of HDCS is to facilitate interactions between human users and calibration algorithms.

3.2 The Steerer of Hybrid Computational Steering

The term “steerer” is defined as the object that makes the steering decisions. Based on the two usages discussed in Section 3.1, this section introduces algorithms and human users as two steerers of the hybrid computational steering. Before discussing the two steerers, we firstly review current utilisations of algorithms and human users in computational steering and related areas.

Conventionally, human users dominate the role of steerer. However, significant challenges have been raised in the integration of computational steering and DDDAS. As a real-time system, the required deadlines attached to steering tasks cannot be guaranteed by human users since they require flexible time to analyse the current status. As simulations that reflect behaviours of physical systems contain enormous amounts of dynamics and uncertainties, it is infeasible for human users to have a comprehensive understanding and rapid steering reaction on dynamic data. Furthermore, human users are more suitable for simulations that have short running times. It is not realistic for human users to supervise a real-time simulation that simulates physical processes constantly. Even in steering scripts, particular steering behaviours can be triggered by pre-defined conditions, and the reliability of this method cannot endow users enough confidence with regards to a physical system that has a large number of uncertainties. Therefore, human users are not able to fulfil the steering requirements in most DDDAS applications.

Rather than driving computational steering by human users, algorithms which are used as the alternative steerers [134, 147, 148] have been discussed before 2000. In a summary of existing works that use algorithms as the steerer, they indicate that computational steering will finally converge into goal-oriented computation when human users, the intrinsic component of computational steering, are replaced by algorithms. They regarded the

algorithm, which replaces the original human users, as the new steerer, namely the algorithmic steering agent. However, restricted by applicable areas and algorithms, complex steering objectives required by human users cannot be realised by automatic steering agents proposed in early works. Moreover, steering is linked with software development in the early stage and is required to be highly reactive, which requires a human rather than algorithmic steerer. Nevertheless, in some existing works, only the ability of changing model parameters either by algorithms are addressed [13, 14, 34, 93]. The feature of improving the interaction speed between the steerer and steered objects, which was raised with the emergency of computational steering (see Section 2.1.2), is neglected.

We argue that computational steering is also addressed by the necessity of facilitating interactions between human users and computations. However, a large number of existing works only focus on developing components to realise such interaction. This thesis sees computational steering from a different aspect in which computational steering can facilitate the feedback loop between different processes such as simulations, human users and algorithms by increase their interaction speed. Therefore, we claim that the contribution of computational steering is not only to provide interaction ability, but also to provide a mechanism whereby steerers and steered simulations can interact more efficiently compared with interactions without computational steering. Hence, the steerer of this project is a container that should be compatible with components, e.g. algorithms and humans, that repeatedly conduct steering-enhanced interactions with complex and time-consuming computations. Based on this understanding, the two steerers implemented in the hybrid computational steering are introduced as follows.

Algorithmic Steerer

We need to have an algorithmic steerer that will take care of lower levels of the steering, that are highly data intensive and hence are not suitable for a human steerer. Consequently, this thesis looks for algorithms that explore parameter spaces in DDDAS. As a result, the data assimilation is selected since it is the main compute-intensive component of which calibration requires a significant amount of running time to explore parameter spaces. Moreover, the calibration algorithms in the assimilation process require frequent interactions with simulations. Hence, this thesis believes that calibration algorithm is a potential component that can be used as the automatic steerer of the DDDCS, and the improvement

brought by computational steering can be represented as the reduced time consumption of calibrations.

In considering the type of steering decisions made by the algorithm is initially driven by the variation of the physical system in a DDDAS, quality of the data is the key determiner of the steering decisions. Therefore, we use the term, Dynamic Data-Driven Computational Steering (DDDCS), and we also use it to match "Dynamic Data-Driven Application System (DDDAS)", which is the name of the application context of this thesis, in order to show how those two acronyms are closely linked.

Human Steerer

In addition to the algorithmic steerer, this thesis keeps steering interfaces for human users to realise the second usage of the hybrid computational steering. The reason is that current algorithmic steerers can only replace human users to conduct function-simple but workload-cumbersome steering tasks. For fuzzy and complex situations in a physical system, human instinct and intelligence are indispensable. Therefore, in this thesis, the steering tasks driven by human users are lifted from the low-level parameter exploration to high-level functions analysis. For example, the steering results generated by the DDDCS must be utilised by functions such as monitoring and predictions. Instead of focusing on the calibration process, human users can analyse the monitoring and prediction results, and steer the calibration algorithm, which is the steerer of the DDDCS. Since the results of calibration process are input for high-level functions, this human-driven computational steering can indirectly steer the high-level functions in terms of accuracy and updating frequency. Compared with conventional computational steering, this human steerer requires more dedicated visualisation components to analyse its steering results. For steering interfaces, a graphical user interface that shows values of steerable parameters and provides controls for users to alter these values can be utilised.

Since both conventional computational steering and the steering raised in this thesis can be driven by human users, in the rest of this thesis, we use Human-Driven Computational Steering (HDCS) to indicate the human-driving part of the proposed hybrid computational steering, and use conventional computational steering to indicate the "legacy" human-driven steering.

In conclusion, this thesis applies the algorithmic steerer in a new application area, DDDAS. Additionally, this thesis keeps the original human users as another type of steerer to steer

the algorithmic steerer. Finally, DDDCS and HDCS are used to indicate the two compositions of the hybrid computational steering.

3.3 Steering Targets

Steering targets are the objects that interact with steerers. For algorithmic steerer in the Dynamic Data-Driven Computational Steering (DDDCS), its steering targets are simulations that can reflect the effects of steered models. In the calibration process, calibration algorithms alter parameters of models. The models are used in simulations that reflect steering effects on the difference between simulated states and observed states. In this thesis, models are considered as static descriptions of physical processes, and simulations are considered as processes of running the model. Hence, even though the calibration algorithm alters parameters of a model, we consider algorithms that actually interact with simulations. Therefore, simulations in the calibration process are steering targets of the algorithmic steerer.

For human steerers in the Human-Driven Computational Steering (HDCS), its final steering target is the output of high-level functions. However, as these functions are driven by results of the calibration in DDDAS, users can steer parameters in the calibration process to indirectly steer simulations. Additionally, endowing human users the ability of steering the calibration can help users to adapt the algorithmic steerer to steer the simulation to areas of interest. By introducing the aim and usage of HDCS, we have explicitly discussed the reason to steer the algorithms by users. Moreover, it is necessary to discuss that this thesis attempt to provide an interface to include human users into the workflow of DDDAS. Hence, by introducing the conventional computational steering into DDDAS, using the calibration algorithm as an example of the steering target of HDCS, other parameters that are interested by users may also be steerable in the future.

A significant difference between the hybrid computational steering and conventional computational steering is that this project applies computational steering on a real-time system. Hence, understanding real-time features of simulations in DDDAS is significant. Therefore, the rest of this section only further discusses real-time simulations that are used as the steering targets in the DDDCS.

3.3.1 Three types of Steerable Simulations

Previously, computational steering was rarely integrated with real-time simulations. The on-line feature is usually discussed in terms of obtaining visual results of the steered simulations in real-time. However, to integrate computational steering with simulations of physical systems, it is essential to differentiate simulations of the Dynamic Data-Driven Application Systems (DDDAS) from simulations steered by conventional computational steering. Consequently, this thesis summarises existing simulations that have been integrated with computational steering into 1) static simulations, 2) offline dynamic simulations and discusses 3) real-time simulations targeted by this thesis. To describe features of theses simulations, simulation time and wall-clock time are two types of time considered in this section. The simulation time is a measurement of duration of simulated events and gaps between them. It provides a dimension to compare states of different stages of a simulation process. The wall-clock time is the human perception of the passage of time in the physical world. It measures the actual amount of time a computing task needs in seconds, minutes and hours.

1) Static Simulations

Firstly, the static simulations are further divided into two types. The first type includes simulations that only reflect states of systems at a specific time point. Additionally, the second type contains simulations that include time passages. But as static simulations, instead of keeping changing as time passes, these simulations enter into a stable status after several iterations of computations. For example, in a wind tunnel experiment, users want to explore the parameter space for an optimal aerodynamic design of a sport car. After an initialization process, environmental parameters such as the strength and direction of the airflow do not fluctuate as simulation time passes. Both types of the static simulations are based on static inputs and they both generate static results at the end.

After being integrated with computational steering, core computations of such simulations continue iterating and checking values of steerable parameters. Hence, without re-initialising, when users steer parameters, simulation results can be emitted faster. In practise, simulations of computational molecular dynamics and fluid dynamics, which have widely been integrated with computational steering, belong to this type.

2) Offline Dynamic Simulations

In offline dynamic simulations, the behaviour of the simulation can evolve in time. The offline dynamic simulations can be considered as running static simulations with a series of static inputs. Hence, it can represent a process in a physical system. The examples can be given from offline simulations that attempt to reproduce a process that evolves as time passes. By using a driving simulation as an example, the aim of an experiment is to explore the fastest route by driving a car from location A to location B through N crossroads. In front of each crossroad, users need to steer the direction parameter which changes the direction of the vehicle. Simulation results include the traffic map and conditions and the location of the vehicle. Since the simulated environment keeps changing, users need to make the steering decisions in an appropriate amount of time according to the information they can collect from the simulated map and the traffic condition. Hence, steering offline dynamic simulations have their timeliness, which means that the steering decision has no value or gradually less value when it arrives after its best time point (deadline) to take effect. In this example, a delayed turning command may lead the vehicle to a remote route.

However, since states of these simulations only change with the simulation time instead of wall-clock time, users can pause the simulations to leave enough time to make steering decisions. If any steering decision is delayed by a mistake, a steering checkpoint can be retrieved to make the steering decision again. If no steering decision is made, the simulations change in a static pattern and wait for steering decisions. Therefore, by including the time saved from re-initialisations, computational steering can be used to steer simulations that evolve to areas that are not of interest back to the areas of interest in the investigation and, accordingly, reduce time wasted on the overall amount of computation for the whole investigation.

3) Real-time Simulations

The previous wo types of simulations are offline simulations that execute based on simulation time. However, to monitor and predict behaviours of physical systems, simulations are required to be synchronised with physical systems. Hence, we use the term real-time simulations to describe dynamic simulations that are based on the wall-clock time. As a result, such simulations need a proper updating frequency to avoid missing important changes in the real world. By using the driving simulation as the example, but this time, human users steer a real vehicle remotely with the help of a simulation. The simulation provides the simulated environment including traffic conditions, the road map and the real-

time states of the vehicle. The pilot needs to explore the simulation to find the optimised route based on the real-time and predicted information. However, different from the second simulation type, if the pilot cannot finish the steering in time, we assume that the vehicle either waits in front of a green traffic light or keeps the current driving direction, which may lead to a detour. Since the simulation is synchronised with the physical system, a delayed or improper steering decision cannot be retrieved, and its influence can only be made up by following steering decisions. Consequently, it is essential to constrain the time for the Steerer to analyse simulation results and make steering decisions. However, most current computational steering projects only focus on offline dynamic and static simulations. To the author's knowledge, no existing works focus on steering real-time simulations.

Because this thesis focuses on simulations of DDDAS, the simulations must reflect the information represented by the dynamic real-time data. Thus, only real-time simulations fit into our context. In addition, this thesis defines the class of real-time simulations as an ensemble of static simulations in which the real-time states of the physical system at each time-step are obtained by using a real-time input to calibrate new static simulations. Hence, it is important to notice that the real-time simulations discussed in the remainder of this thesis includes the workflow of the two static simulations types as well.

3.3.2 The Steerable Real-Time Simulation

The author agrees with the definition that simulations are the imitation of the operation of a real-world process or system over time [71], and a real-time simulations can reflect significant changes in the physical world in time. Practically, the updating rate can be different from one microsecond per simulation time step to one day or one week per simulation time step. Hence, it is necessary to discuss the frequency of simulation updates.

The continuous simulations and discrete-event simulations are two terms tightly connected with the real-time simulations in the field of modelling and simulation. As discussed by Birta et al. [17], continuous simulations are based on a set of differential equations that represent deductive processes which can be governed by the specific physical laws. In contrast, the time of discrete-event simulations jumps between events and may advance faster or slower than the physical time. Hence, it is argued that discrete-event simulations are not real-time simulations since it can omit subtleties such as state changes and unknown events [157]. However, recent works such as Belanger et al. [15] consider the discrete simulations as the real-time simulations provided that the condition of triggering events is sufficiently subtle

to reflect important changes of the physical systems. Moreover, recent research integrates continuous dynamics with the discrete-event models by approximating a superdense model of time [86]. Simulations of such superdense model outputs the status between two discrete events by calculating the effects of a set of casual-related actions. The actions are executed strictly in order but without the concept of time passage. Hence, as long as the time restrictions between two discrete events are met, the simulations behave similarly to the physical system.

The author considers that both continuous and discrete-event simulations to be real-time simulations in this thesis. Regarding the real-time discrete-event simulations, they must comply with the deadline of each event, and only the actions between two events can neglect the time passage. As a result, the steerable real-time simulations are divided into steerable continuous simulations and steerable discrete-event simulations. For the steerable continuous simulations, they are considered as continuous simulations with an imposed discrete time step[12], which is the steering time step. The size of the time step is determined by particular applications and one steering operation must be finished in the interval between two discrete steering time steps. Alternatively, the time requirement of the steerable discrete-event based simulations is determined by the interval between two events. Different from continuous simulations, which can have a static steering time step, discrete-event simulations must predict the arrival time of the next event to set the deadline for conducting the steering. In conclusion, the continuous simulations require static deadlines. On the other hand, the discrete-even simulations require dynamic deadlines.

Because this thesis uses the simulation of the water distribution network as the use case, a widely used water network simulation package, EPANET is used as an example to introduce real-time simulations. EPANET is a software package that simulates the status of large-scale water distribution system, and it has a function named Extended Period Simulation (EPS) that aims for simulating a continuous period. The basic simulation function is static and only calculates the state of the network at one time step, and this time step can only reflect the state of the physical world at a specific time point. By combining these single time steps together, the EPS can simulate the continuous states of a network during a period by presenting the system states at a serial of time points. Additionally, it assumes states that one time point can represent states of the interval between two time points. On the other hand,

[12]Continuous simulations with discrete time step are a time based simulations. It is different from the discrete-event simulations, which are event based.

the continuous simulations can also be integrated with discrete events. By assimilating real-time information from the physical world, this offline dynamic simulation can be used as a real-time simulation.

3.4 The Steering Loop of Dynamic Data-Driven Computational Steering

In light of the analysis of computational steering in Chapter 2, this thesis believes that implementing computational steering includes three phases, namely, identifying steerer and steering target, selecting steering instrument and facilitating the control loop. Hence, this section follows these phases to discuss the implementation of the Dynamic Data-Driven Computational Steering (DDDCS).

As the steerer of DDDCS is a calibration algorithm and the steering targets are simulations used in the calibration process, to apply computational steering, it is significant to study the calibration process so as to target the feedback loop that can be facilitated by computational steering. Firstly, the term "feedback loop" used in computational steering needs to be explained. Vetter [148] first raised the analogy between computational steering and the closed-loop feedback system based on feedback control systems described by Philips [112], and it follows the definitions of the control theory. In the control theory, the control loop can be categorised as an open or closed control loop. As defined by Hellerstein [58], an open control loop, also referred as feedforward control, is a technique that avoids adjusting input by referring to its output. In contrast, outputs of a closed control loop can be fed back to affect its future inputs. Hence, the closed control loop is also named as the feedback loop.

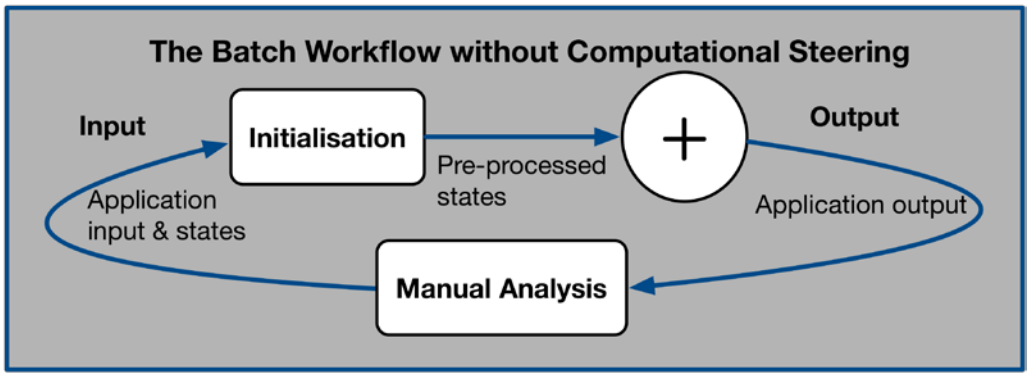


Figure 3-1 The Original Workflow without Computational Steering: It depicts the batch mode of running simulations on HPCs in a form of a closed control loop and indicates a closed control loop has already existed before introducing computational steering.

According to the survey discussed in Section 2.1.4, many works only noticed that computational steering can close a control loop by providing the ability of interactions between human users and simulations [73, 93, 102, 105, 106, 110]. This thesis argues that definitions referred from the control theory are not sufficient to capture the nature of the conventional computational steering. A problem domain targeted by computational steering is the batch process of analysing compute-intensive and time-consuming models. Figure 3-1 depicts the workflow of such batch process. The output of the computation shown, as symbol ⊕, is used by human users, and the following modifications made to the input are based on the output. Therefore, in the context of the control theory, the previous workflow has already been a closed control loop or a feedback loop. Hence, it is not comprehensive to claim that computational steering can only close the loop.

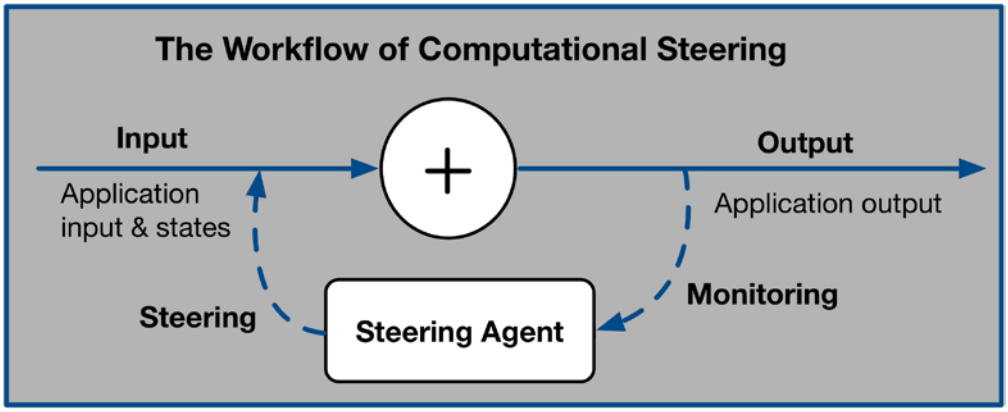


Figure 3-2 The Workflow of Computational Steering [148]: This figure is used to be compared with Figure 3-1to support our argument that it is necessary to emphasize another feature of computational steering is to reduce the length of the feedback loop.

Figure 3-2 presents the workflow of computational steering introduced by Vetter. By comparing with Figure 3-1, Figure 3-1 shows that human users can only get results at the end of the simulation process and change the input parameters at the beginning of a simulation process. However, in Figure 3-2, the steering agent can make steering decisions and obtain computation status when the simulations are still running. Therefore, this thesis reasons that computational steering does not only provide a new control loop. In addition, it improves the efficiency of the original control loop by reducing the length of the closed control loop.

Based on our understanding of computational steering, the first task of implementing computational steering is to target control loops in DDDAS. As stated by Kephart et al. [79], the underlying concept of an autonomic component is a control loop. Hence, since DDDAS

can contain a complex hierarchy of autonomic components to achieve high-level functions, we assume it is reasonable to find control loops in applications of DDDAS. Consequently, targeting the closed control loop becomes the new task.

The realisation of a DDDAS can be diverse among specific projects. Since the calibration is a critically time-consuming component of DDDAS, we attempt to deploy computational steering on components of the calibration process by analysing its workflow as shown in Figure 3-3.

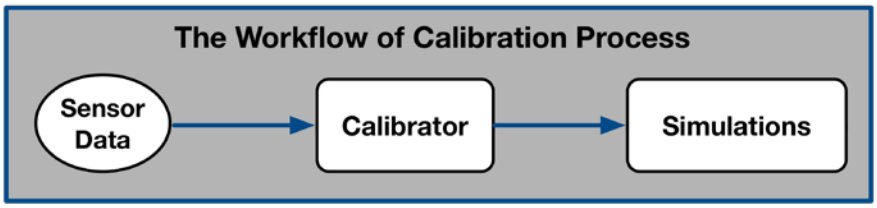


Figure 3-3 The Workflow of the Calibration Process: The arrows indicate dataflow between components. The Calibrator calibrates models being used by Simulations based on information represented by Sensor Data.

The steerer in this workflow is the Calibrator, which is a calibration algorithm that can be used to adjust model parameters. The sensor data is considered as an example of the dynamic data of physical systems. On the right side, the Simulations component alters parameters of its models based on input received from the Calibrator and outputs simulation results based on such models. As a result, the Sensor Data is taken by the Calibrator as references to evaluate its calibration results represented by simulated states. To present the control loops as depicted in Figure 3-1 and Figure 3-2, Figure 3-4 extends the general workflow of DDDAS shown in Figure 3-3.

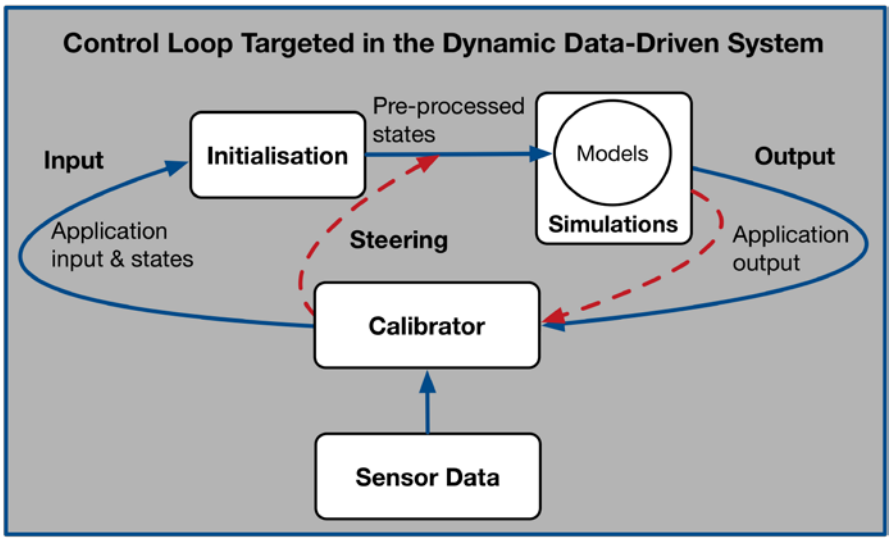


Figure 3-4 Control Loop Targeted in the Dynamic Data-Driven System: This figure is derived from Figure 3-4, and is used to expose the control loop in the calibration process in DDDAS. The black solid lines indicate the

workflow which is not integrated with computational steering. The red dashed lines indicate the workflow which is integrated with computational steering.

The solid arrows between Calibrator and the input and output indicate the previous control loop, and the dashed arrows indicate the workflow after being integrated with computational steering. In the steering workflow, Simulations are kept running. However, rather than looping from the beginning to the end, Simulations do no re-execute initialisations. The Initialisation component includes processes of writing input parameters into memory and recalculating variables that are related to input parameters. Hence, only re-running core computations avoids re-initialising unchanged parameters and their related variables. This mechanism can improve the efficiency of programs of DDDAS since its primary application areas are the large-scale physical systems which typically have a significant amount of input parameters. Moreover, as indicated in Section 3.3.1, simulations can require a pre-processing phase to reach a stable or a specific state, namely the "transient" or as "spin-up" time, and only then are the input parameters utilised by the model. This thesis includes this phase into the Initialisation component. For such simulations, computational steering improves the speed of iterations by restarting the model with environmental parameters of previous executions. Furthermore, even though the improvement represented on one re-execution is slight; however, as the Calibrator has the potential to issue a tremendous amount of re-executions as discussed in Section 2.2.2, the overall effect of improvement can be considerable. Conventionally, it is solved by using parallel programming to tackle massive reruns of the simulations. This thesis proposes that by combining computational steering with the parallel programming, the running time of calibration process can be further reduced.

However, as well as benefits, applying computational steering to calibration process also introduces issues that have not been addressed by existing works in computational steering. Since real-time systems having a time requirement on the calibration, it is necessary to guarantee the time consumed by interactions between the calibration algorithm and steered simulations. The dynamic quality of the data can have a major effect on the workloads of the calibration process. Therefore, it is necessary to predict the time required by the calibration process with respect to available dynamic computing resources. However, existing time management on computational steering only focuses on reducing update latency between the steerer and steering targets. To the author’s knowledge, no work has studied the time issues raised by dynamic steering workload. As a result, this thesis considers time management and computing resources assignment methods in scientific workflow systems to tackle this problem. Because these methods are not conventional computational

steering methods and require to be explained within an architecture, we will discuss them in the design of the steering architecture in Chapter 4.

3.5 The Steering Loop of Human-Driven Computational Steering

Because the aim of HDCS is to facilitate interactions between human users and calibration algorithms, and the calibration algorithm is the steerer of the DDDCS, this thesis considers that the realisation of HDCS is also an integration between HDCS and DDDCS. In terms of three steering development phases introduced in Section 3.4, we have identified the human users as the steerer and calibration algorithm as the steering target. Hence, this section discusses the feedback loop in HDCS.

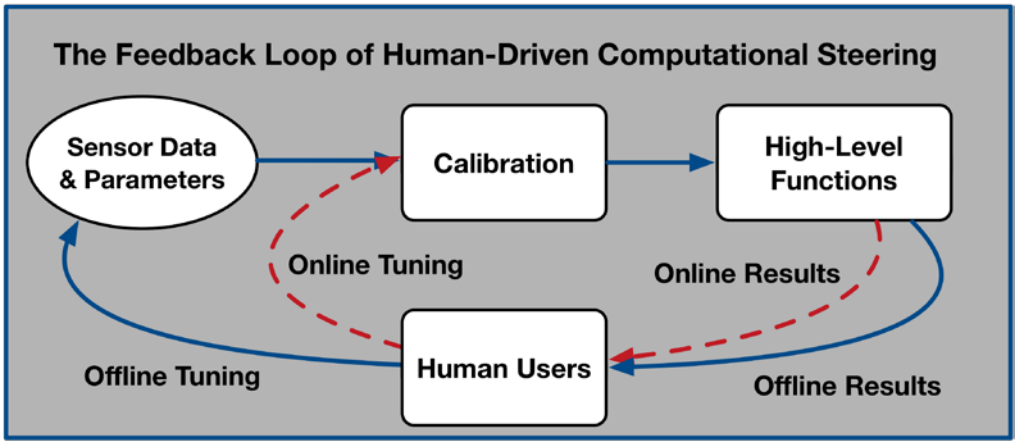


Figure 3-5 The Feedback Loop of Human-Driven Computational Steering: The solid arrows present the feedback loop before using computational steering, and the dashed arrows indicate the feedback loop after being integrated with computational steering. The steerable workflow enables users to alter the calibration online. However, in the original workflow, users can only make changes on the calibration in offline mode. Since the DDDCS only affects communication inside the calibration process, this figure can also indicate the composite steering loop of the hybrid computational steering.

Figure 3-5 shows the proposed feedback loop in HDCS. The Calibration component includes the calibration algorithm. The solid arrows present the feedback loop before using computational steering, and the dashed arrows indicate the feedback loop after being integrated with computational steering.

Since without integrating computational steering with this workflow, calibration algorithms are typically tuned in the offline mode by users. Only historical sensor data can be used to tune the calibration algorithm. Based on the results of high-level functions, users modify parameters of the calibration process in order to obtain expected results from high-level

functions. This workflow is same as the batch mode described in Section 1.1.2, and it is used to search an optimal configuration for the calibration algorithm to cope with situations indicated by the historical sensor data.

On the other hand, integrating computational steering with original workflow enables human users to have direct interactions with calibration algorithms. It reduces the length of the original feedback loop to enable users to refine algorithms more efficiently. Moreover, as the computational steering also incorporate human users into the workflow, human users can alter parameters of calibration algorithms without break the real-time nature of applications of DDDAS. As a result, the calibration can use dynamic sensor data to support high-level functions to present the real-time states of the physical systems.

In addition to refining calibration algorithm, this online interaction ability can be used to adapt the calibration algorithm based on requirements of users on high-level function, since the high-level functions uses results of the calibration algorithm as input. A typical example of an algorithm parameter that can be steered by human users is the amount of sensor data. For instance, during regular situations, the variation of physical system is not great; hence, users can reduce the amount of sensor data to reduce the calibration workload and obtain a faster calibration speed. On the other hand, when emergencies occur, users can increase the amount of sensor data assimilated by the calibration and require a more accurate understanding of emergencies. Consequently, high-level functions are used as the visualisation component of the conventional computational steering to provide steering feedback to users, and the visualised information is the core knowledge that users can analyse for their steering operations The implementation of the visualisation component depends on requirements of specific applications. However, the visualisation components, such as those require heavy rendering tasks, also have the potential to be a barrier for computational steering. As the use case of this project does not have a time-consuming rendering task, only computing resources for the visualisation will be considered in the future work.

One issue raised with integrating HDCS with DDDCS is the potential conflicts between human users and the calibrator. At the current stage, this project assumes modifications made by human users and sensor data are at two different levels. Human users only handle tasks that are not reliable to be executed by algorithms, and the steering in simple and tedious tasks are handed over to algorithms. However, in future works, it is possible and interesting

to enable human users to steer some interesting parameters currently handled by algorithms and vice versa. Hence, it would be necessary for developers to carefully arrange the steering code annotations to guarantee steering priorities required in different projects. Moreover, a dedicated function to introduce race conditions would be interesting to be developed in the future work.

Moreover, even though the HDCS is a high-level computational steering, it can influence the low-level components. By using monitoring as an example of the high-level function, a higher updating frequency of the monitoring requires a higher convergence speed of calibration algorithm. When users do not want to increase speed by losing accuracy, it is still necessary to dynamically assign computing resources to meet user requirements. Hence, the dynamic workload of the steerable calibration is an issue introduced by the HDCS, and this project proposes to allocate dynamic computing resources as a solution to address this issue. For example, the monitoring can be updated every 30 minutes by running calibration on eight-cores machines. However, when users require the system to be updated every 15 minutes, 20 cores need to be assigned to the Calibrator (16 cores may not be enough due to the parallel overhead). However, to realise such funtions, a computing infrastructure that includes time management and a computing resource assignment must be developed. Since such functions are beyond scope of conventional computational steering, and this chapter mainly introduces the hybrid computational steering inspired by the conventional computational steering, this thesis discusses them as one function of the architecture which will be discussed in Chapter 4.

3.6 Chapter Summary

This chapter discusses the theory of hybrid computational steering. It consists of both Dynamic Data-Driven Computational Steering (DDDCS) and Human-Driven Computational Steering (HDCS).

We emphasise that computational steering does not only create a feedback loop; instead, it improves the speed of interactions between components in the feedback loop. This understanding is used to describe the aim of the hybrid computational steering and aims of its subsystems, DDDCS and HDCS.

For DDDCS, model exploration is defined as its usage, calibration algorithms are its steerers and real-time simulations of physical processes are its steering targets. The hypothesized

advantage of using DDDCS is that it reduces the length of the original feedback loop in the calibration process and increases the speed of interactions between calibration algorithm and calibrated simulations. Finally, its specific aim is facilitating interactions between calibration algorithms and simulations.

Since using algorithms to conduct all steering tasks has not yet been proved to be reliable enough to meet users' requirements, the calibrator in DDDCS may need to have a higher level input for supporting interpolation. As a result, algorithm adaption is defined as the usage of HDCS in which human users are steerers and calibration algorithms are its steering targets. The hypothesized advantage of using HDCS is that it facilitates interactions between human users and calibration algorithms. Hence, human users can adapt high-level real-time functions by altering parameters of calibration algorithms without interrupting real-time applications of DDDAS. Finally, its specific aim is facilitating interactions between human users and calibration algorithms.

Chapter 4 Hybrid Computational Steering Architecture and Time Management

Based on the concepts of the hybrid computational steering discussed in Chapter 3, it is inevitable for this thesis to design an architecture to establish concepts of the hybrid computational steering. Hence, this architecture must be designed with respect to three main problems raised in integrating computational steering with Dynamic Data-Driven Application System (DDDAS). Firstly, conventional computational steering is typically driven by human users, and to our knowledge, this thesis is the first to integrate computational steering with the calibration process in DDDAS. Hence, the architecture must include a new design of workflow and communication methods among components of the Dynamic Data-Driven Computational Steering (DDDCS). Additionally, the DDDCS must be integrated with the HDCS to form a composite computational steering architecture. Lastly, since time is a critical factor in integrating the in-silico process with the physical world, using dynamic information to drive computational steering must meet the time constraint required in the physical world. Hence, this architecture must comprise components to manage time and computing resources for the DDDCS and HDCS.

Consequently, the aim of this chapter is to introduce the development of the hybrid computational steering architecture. The general organisation of components is introduced by grouping components into three layers, 1) HDCS, 2) DDDCS and 3) Time and Computing Resource Management. The time and computing resource management layer is also named as the DDDCS on the Hardware Level (DDDCS-HL), and the reason will be discussed in Section 4.1. To differentiate the time and computing resource management with the DDDCS. We change the DDDCS to Dynamic Data-Driven Computational Steering on Software Level (DDDCS-SL). This general architecture defines relations among layers and general functions they need to realise. Afterwards, the general architecture is separated into three sub-architectures designed specifically for each layer. Based on the target function, the sub-architecture defines relations among components and introduces methods utilised to realise such components.

Additionally, even though the development of the architecture is based on general applications in the context of DDDAS. Since introducing a specific use case can make the explanation more clearly, a Water Distribution System (WDS) is used as the example of

DDDAS to create an exemplary application for readers. As the simulation and visualisation occupies a much less execution time compared with the calibration process, the key aspect being considered is time for running the calibration process. Moreover, a number of projects have provided toolkits to support the development of computational steering (See Section 2.1). However, many of them must be used under specific development environments. Thus, this thesis explicitly uses the RealityGrid computational steering to support the development of steering related components, since the RealityGrid steering has a high flexibility to be integrated with other projects. We also hope to motivate the usage of RealityGrid after this thesis is published.

4.1 Architecture of the Hybrid Computational Steering

By introducing simulations of DDDAS as the new application area of computational steering, we generally propose two types of computational steering: Dynamic Data-Driven Computational Steering (DDDCS) and Human-Driven Computational Steering (HDCS). Additionally, the DDDCS is further divided into two levels. The first level is the software level on which the algorithm steers models based on real-time observation data. The second level is the hardware level on which the amount of the computing resources assigned to software is steered to meet the requirement of the time constraint. The integration of the DDDCS and the HDCS is named as the hybrid computational steering, and its theory has been discussed in Chapter 3. To support this theory, this section introduces the architecture of the hybrid computational steering developed in the context of the Dynamic Data-Driven Application System (DDDAS).

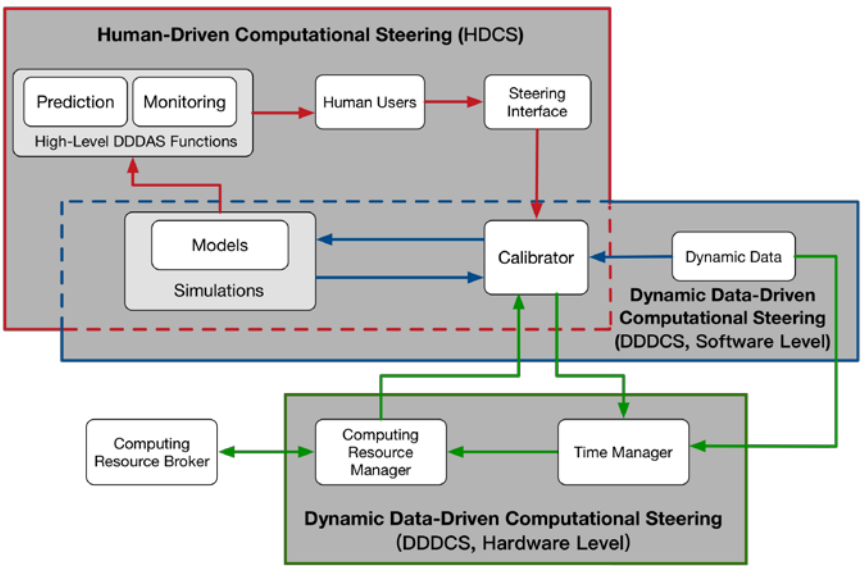


Figure 4-1 Hybrid Computational Steering Architecture: The red lines outline the HDCS on the highest level. The blue lines outline the DDDCS-SL in the middle. The green lines outline the DDDCS-HL. This figure illustrates conceptual components of the hybrid computational steering on three layers. It indicates the communications among these conceptual components to fulfil the aims explained in concepts of the hybrid computational steering.

Figure 4-1 illustrates the general architecture of the hybrid computational steering. The HDCS, DDDCS on the Software Level (DDDCS-SL) and DDDCS on the Hardware Level (DDDCS-HL) are three groups that constitute the general architecture of the hybrid computational steering, and they are organised as three layers on top of each other. The HDCS is on the highest layer on which human users make steering decisions based on the steering results of the DDDCS-SL. The DDDCS-SL is located on the middle layer. It explores parameter spaces of simulation models by utilising computational steering and provides input for the high-level functions on the HDCS layer. The DDDCS-HL is at the bottom layer. It manages the time and computing resources utilised in the middle layer. The core layer is the DDDCS-SL. Firstly, as the main component of the DDDCS-SL, the Calibrator component is a link between the preceding and the following steering layers. Secondly, our main hypothesis is to use computational steering to increase the speed of the calibration in DDDAS; hence, steering on the other two layers are derivatives used to support the steering at the middle layer. Since each of the three steering layers has a steering loop, this section describes this architecture by introducing the steering loop and their surrounding components, in which the core layer, the DDDCS on the software level, is the first component discussed.

This work considers a model as the mathematical representation of a physical world, which can be driven by parameters to generate states of physical systems. Additionally, a simulation is a process of using those parameters to drive a model and to present generated results. Based on this understanding, we discuss components inside the DDDCS-SL. Firstly, the Models component is used by Simulations to estimate states of the physical world. However, in practise, some of the parameters that drive the Models cannot be directly observed from the physical world, and observations such as sensor data can need pre-process before they can be used by the simulation. Thus, the Calibrator is applied to estimate the driving parameters of the Models. During the calibration, the Calibrator drives Models with different sets of parameters and comparing the simulation results with the Dynamic Data. The model is well-calibrated when one set of estimated parameters can drive the model to generate simulated results that are close enough to the Dynamic Data. Otherwise, the Calibrator is stopped after reaching its deadline. The measure of whether a simulated is close

enough to sensor data is determined by specific stopping criteria configured in the calibration algorithm. After the calibration finishes, the calibration results are taken as input by the High-Level DDDAS Functions such as Prediction and Monitoring. They can provide Human Users with insight of the states of physical systems. The Calibrator is the Steerer in the DDDCS-SL. It interacts with Simulations and steers parameters of Models used by Simulations. The communication loop formed by the interaction between Calibrator and Simulations is considered as the steering loop on the middle steering layer.

Above the DDDCS-SL is the HDCS in which Human Users analyse results from High-Level DDDAS Functions. Based on the analysis, Human Users, as the Steerer of the HDCS, steer configurations of the Calibrator. Hence, we claim that the HDCS is on a higher layer than the software-level DDDCS. The Steering Interface is the same as what is defined in conventional computational steering. It provides users with the real-time information from the Calibrator and provides users with controls to alter steerable configurations of the Calibrator. As a result of using this architecture, two limitations that prevent Human Users from interacting with applications of DDDAS are addressed. Firstly, since results of High-Level DDDAS Functions are based on the calibration results from the middle layer, the steering decisions made by Human Users can indirectly affect results of the High-Level DDDAS Functions. Hence, Human Users do not need to manage an enormous amount of steerable parameters in the parameter space. Secondly, by separating HDCS from the DDDCS, this architecture does not require Human Users to make steering decisions under a tight time constraint. One difference between the HDCS shown in this architecture and conventional computational steering is that this thesis does not have a specific visualisation component. Instead, the indirect steering results are presented in the High-Level DDDAS Functions, and visualisation components could be built in these functions. Figure 4-1lists Monitoring and Prediction as two examples of such functions in DDDAS. Finally, the second steering loop is formed by the interaction between Human Users, the Calibrator and High-Level DDDAS Functions.

The last steering loop includes the Time Manager, Computing Resource Manager and the Calibrator. This loop is defined as the DDDC-HL. Since meeting time requirements is crucial in DDDAS, this thesis proposes a method to manage the required execution time of the dominantly time-consuming components, the Calibrator and Simulations, by assigning them dynamic computing resources based on the time constraint. To achieve this function, the Time Manager maintains relations among the real-time Dynamic Data, Calibrator

parameters steered by HDCS, amount of computing resources and the required running time of the Calibrator. Hence, the Time Manager sends the amount of required computing resources to the Computing Resource Manager which further applies for computing resources from the Computing Resource Broker. The Computing Resource Broker maintains a service with computing resource providers which provide computing resources such as grid, cloud and supercomputers. The Computing Resource Manger also indicates the amount of applied computing resources to the Calibrator which adapts its parallelism scale to available computing resources. The feedback sent to the Time Manager is the actual running time consumed by the Calibrator with values of steered parameters. Together with dynamic data and requested computing resources, the Time Manager stores this information as a historical relation to estimate the execution time in the future. It may seem as a contradiction to claim computational steering on the hardware level. This thesis argues that, even though objects steered by the Time Manager and Computing Resource Manager is the amount of computing resources, additionally, they also affect the program of the Calibrator in terms of running in parallel. We further argue that steering computing resources ultimately affect the results users obtained from the High-Level DDDAS Functions. Hence, this thesis describes altering the amount of computing resources as a type of steering driven by dynamic data. Additionally, as this ability of assigning dynamic computing resources is a specific research area which has been addressed by works in grid computing, this work does not include it in the discussion of the hybrid computational steering in Chapter 3.

After introducing the overall architecture of the hybrid computational steering, the rest of this chapter introduces our methods to implement components on the three steering layers explicitly. Since the implementation of HDCS has no additions on the conventional computational steering, its description is interwoven into the description of other steering layers.

4.2 Architecture of Dynamic Data-Driven Computational Steering on Software Level

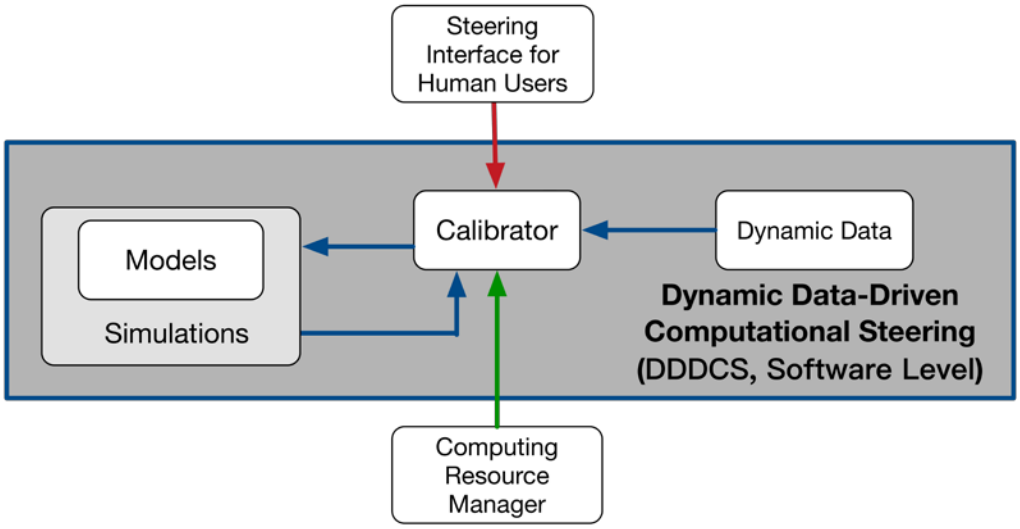


Figure 4-2 The Architecture of the Dynamic Data-Driven Computational Steering on the Software Level: This figure is extracted from Figure 4-1. Arrows indicate the dataflow between components. The Computing Resource Manager informs available computing resources for the Calibrator in order to run Simulations in parallel. The Steering Interface for Human Users can interact with the steering interface inside the Calibrator to change alter configurations of the calibration process.

Figure 4-2 illustrates the architecture of the Software-Level Dynamic Data-Driven Computational Steering (DDDCS-SL). It is extracted from the architecture of the hybrid computational steering shown in Figure 4-1. The four components include: Calibrator, Dynamic Data, Models and Simulations. In existing works of DDDAS, the calibration process also consists of these components. However, the DDDCS-SL aims for optimising the interactions between the Calibrator and Simulations. Therefore, this section focuses on introducing computational steering into the existing calibration process. Other research problems in the calibration such as storage and transfer of dynamic data are not studied in this thesis. Additionally, since this section only discusses the implementation of DDDCS-SL, the remainder of this section uses the term DDDCS without indicating the software level.

4.2.1 Calibrator – The Steerer of Software-Level Dynamic Data-Driven Computational Steering

The key process in utilising computational steering is to realise the steering interaction between the Steerer and the steered simulations. Based on the DDDCS discussed in Chapter 3, calibration algorithms are used as the Steerer of the DDDCS to explore parameter spaces

of models. Thus, instead of describing in the abstract, this thesis first uses a calibration algorithm as our example in the explanation.

This work has received help and support from water companies. Thus, we use the Water Distribution System (WDS) as the use case of the Dynamic-Data Driven Application System (DDDAS). We built our project based on a Knowledge Transfer Partnership project between the Water Research Centre [13], Northumbrian Water company and the University of Manchester [82]. In previous software implementations of the collaborative project, the Genetic Algorithm (GA) has been applied as the calibration algorithm. Moreover, GA is also a widely utilised calibration algorithm to adapt water models to dynamic information obtained from the physical world [47, 83, 120, 124, 127]. Therefore, we argue that it is reasonable to use the GA as an example of the calibration algorithm. As a result, the calibration algorithm mentioned in the rest of this thesis indicates the GA. The background of GA can be found in Section 2.2.2.

Since GA is selected as the algorithm utilized by the Calibrator, and the Calibrator is the Steerer in the DDDCS, we then discuss our method to realize the interaction between the GA and steered Simulations. By applying the steering interaction, the increased speed of executing the calibration process is considered as the contribution of the DDDCS.

This thesis uses the term, steering interaction, to indicate the interaction improved by using DDDCS, and to describe the original interaction as the batch interaction. Based on the theory of the hybrid computational steering, the improvement taken by steering is based on opening the simulation model, which acts as a black box in the view of the GA, and on establishing an interaction between the GA and the internal workflow of Simulations. Therefore, to develop the steering interaction, we firstly introduce the batch interaction to understand what is in the black box.

[13]WRc is an Independent Centre of Excellence for Innovation and Growth. They operate across different sectors including Water, Environment, Gas and Resource Management. http://www.wrcplc.co.uk/

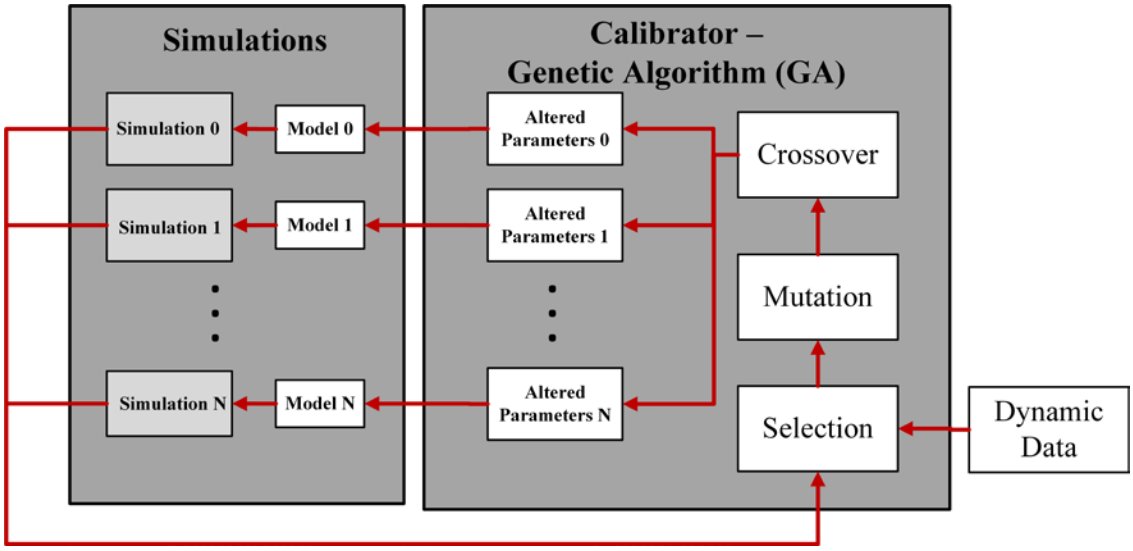


Figure 4-3 The Batch Interaction between Genetic Algorithm and Simulations: This figure indicates the workflow between GA and Simulations without steering ability. It can be compared with the Figure 3-1 to indicate the calibration is similar to a parallel batch workflow. It also helps to understand the calibration process in detail.

The batch interaction between the GA and Simulations is depicted in Figure 4-3. Simulations are executed based on the Models. The Models are initialised based on prepared inputs, and afterwards, it uses parameters obtained from the Calibrator as inputs. Results of Simulations are compared with the real-time Dynamic Data in the Selection component. The comparing results are represented as fitness values which indicate the measure on the correctness of input parameters which are used as input by Models. Afterwards, input parameters that lead to better Simulation results are selected to generate a new ensemble of input parameter sets. The new input parameters are created by altering selected parameters through Mutation and Crossover functions, and drive Models to generate a new batch of simulation results. One such loop in GA is named as one generation, and each parameter set is termed as one individual. A parameter in the parameter set is considered as a gene segment of an individual. The calibration process is terminated upon reaching a specific fitness value, e.g. a generation number or an execution time. Since GA is a widely utilised algorithm, a great number of studies have focused on studies of selection, mutation and crossover methods. As the purpose of this thesis is not contribution to GA, the tournament selection, Gaussian mutation and simple crossover [32], are used in the implementation of GA for the purpose of easy to code.

After introducing the batch interaction in the non-steerable calibration, this thesis summaries its difference from the batch interactions targeted by most conventional computational steering. As a result, three difference are indicated as follows. The first difference is that the

Calibrator/GA can take advantage of parallel computing to accelerate the exploration speed. The parallel steering can be similar with the scenario in which human users steer multiple simulations at the same time based on the same model. However, the difference is that GA can steer tens and hundreds of simulations simultaneously. Secondly, the GA does not require a visualisation component since the Selection can directly process simulation results in raw data type. Finally, unlike human users, the Calibrator itself is a steering target in the view of its higher and lower level Steerers such as Computing Resource Manager and Human Users. Hence, it must be developed with the ability of being steered.

By discussing the batch interaction between GA and Simulations, we conclude that using GA as the algorithm of the Steerer does not make the interaction between Steerer and Simulations significantly different from the interaction between Human Users and Simulations in the conventional computational steering. With respect to three differences discussed above, existing Computational Steering programming toolkits can be used to realise the interaction between GA and Simulations. Among toolkits reviewed in the background, this thesis takes the RealityGrid Computational Steering toolkit [19] to develop the steering interaction.

Because the RealityGrid is developed with flexibility in mind, it can be modified to fit into a new project conveniently. Based on its loosely coupled feature, its visualisation component is removed, and functions that are responsible for interactions between Steerers and Simulations are kept. In terms of the parallel steering, since the conventional RealityGrid does not support the Steerer to acknowledge multiple steered Simulations, this thesis abstracts multiple Simulations as one steering target, and sends the steered parameters for all simulations to the Simulations component. This solution can be realised in two ways: 1) The steered parameters for all simulations can be sent as an entire data set to a master process. The master process assigns the steered parameters to the slave processes based on the meta information received with the steered parameters. 2) Another method is to develop a relay steering service that can divide the entire data set into slices, and each slice corresponds to a process in the parallel programming. Both methods will be further discussed in the Chapter 5 since they are related to the implementation on the computing infrastructure.

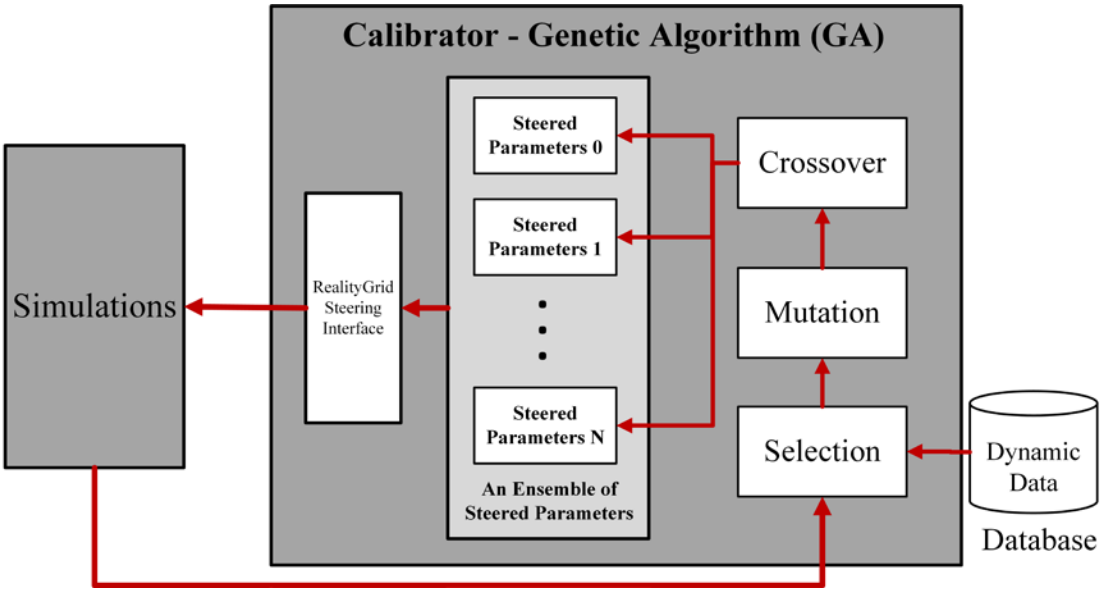


Figure 4-4 Steering-Enabled Calibrator: This figure can be compared with Figure 4-3. It indicates the implementation of the computational steering on the GA side. The steered parameters are ensemble and transferred through the RealityGrid Steering Interface to Simulations.

Consequently, Figure 4-4 indicates the steering-enabled Calibrator. By being compared with the non-steerable Calibrator shown in Figure 4-3, the difference is that it packs Steered Parameters for all simulations into an ensemble and sends them through the RealityGrid Steering Interface. This steering interface interacts with the steering interface inside Simulations on the left. The next section discusses the development of the Simulations component.

4.2.2 Simulations

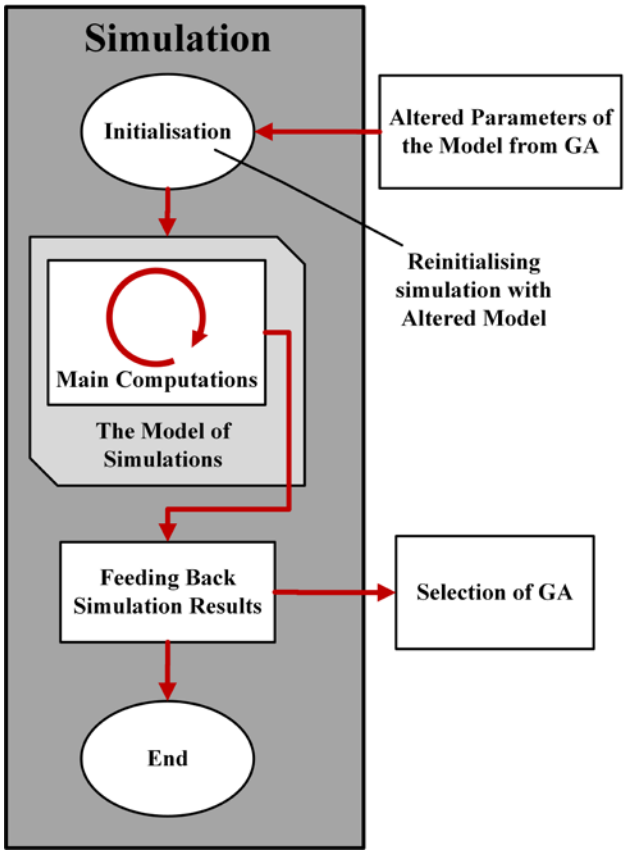


Figure 4-5 Ordinary Interaction in Calibrator: This figure presents the workflow inside the simulation being used in the calibration process. As a non-steerable workflow, it is used to find the position to integrate the computational steering. In indicates, without computational steering, the altered parameters need to re-initialise the simulation to enter the Main Computations. The circle shown in the Main Computations indicates the iterations required to finish the computation.

Similar to the study on the Calibrator, this section introduces the development of the steerable Simulations based on studying a non-steerable simulation. Figure 4-5 indicates the batch workflow of one non-steerable simulation and its corresponding new input. At the initialisation stage, parameters are stored into memory, and in addition, variables that are based on such parameters are computed. After the initialisation stage, parameters and variables are utilised by the Main Computations.

The Main Computations are based on the model which describes numerical relations between parameters and states of simulated systems. Based on the three steerable simulation types discussed in the Section 3.3.1, most simulations targeted by computational steering have an inner loop in the Main Computations. Depending on specific simulation type, the inner loop can be a number of computations that search for better solutions. In addition, it can also represent the progress of changing states based on one set of input parameters. For

the first type of inner loop, Main Computations can be stopped after a specific number of iterations, to be stopped when there is no improvement after a number of iterations or to be stopped after reaching a specific fitness value. On the other hand, the computation iteration for the second inner loop type finishes after reaching the final states.

The results of Main Computations do not require to be sent to the calibrator before the loop is terminated since the Calibrator implemented in this thesis does not support the steering function in which the Steerer can evaluate the running direction of the Main Computations based on the intermediate states and make corresponding steering before the current computation loop terminates.

By taking the large number of elements in WDS into consideration, both initialisation and destruction of these elements can have an enormous contribution on the unnecessary time consumption. Since the GA may require hundreds and thousands times of interactions, the time wasted on re-initialisation can also be magnified by the large-scale interactions between GA and Simulations.

After studying the workflow of non-steerable simulations, we introduce three factors that can be improved by introducing computational steering. Firstly, the new model parameters modified by GA may account for only a small portion of the total steerable parameters. However, the re-execution of the initialisation re-initialises unchanged parameters and reruns computations related to these unchanged parameters. The time wasted is presented as T_{ucp}. Secondly, simulations, such as those based on the model of DL_MESO[14], require a large number of computations from the initial state, S_0, to reach a stable state, S_1. This time at the start of running a model is known as "transient" or as "spin-up" time. If we denote $T(S_i, S_j)$ to the execution time for a simulation to run from state S_i to S_j, then the original calibration method creates a large amount of unnecessary transient time on simulations that have a relation: $T(S_0, S_2) > T(S_1, S_2)$. Moreover, the execution of all other extra works required at the initial stage, such as network negotiation, is presented as T_{rt}. Combing the previous concepts, the total time wasted in a single batch interaction is $T_{wb} = T_{ucp} + (T(S_0, S_2) - T(S_1, S_2)) + T_{rt}$. Thirdly, by taking GA and the parallel programing into

[14] DL_MESO is a general purpose mesoscopic simulation package developed at Daresbury Laboratory by Dr Michael Seaton under the auspices of the Engineering and Physical Sciences Research Council (EPSRC) for the EPSRC's Collaborative Computational Project for the Computer Simulation of Condensed Phases (CCP5). The package is the property of the Science and Technology Facilities Council (STFC).

consideration, the wasted time also depends on parameters such as the generation number, population size and the scale of the parallelism. If we denote the generation number, population size and the scale of the parallelism as Gen, Pop and Pro respectively (For example, if MPI is used, the Pro is the number of available of processors). Therefore, the number of re-executions is $\frac{Gen \times Pop}{Pro}$ and the total wasted time is $T_{tw} = T_{wb} \times \frac{Gen \times Pop}{Pro}$. Without increasing the converging rate of GA, one method to reduce this time waste is to increase the parallelism scale. However, this method is limited by hardware cost and the decreasing parallel advantage obtained from the increasing number of cores.

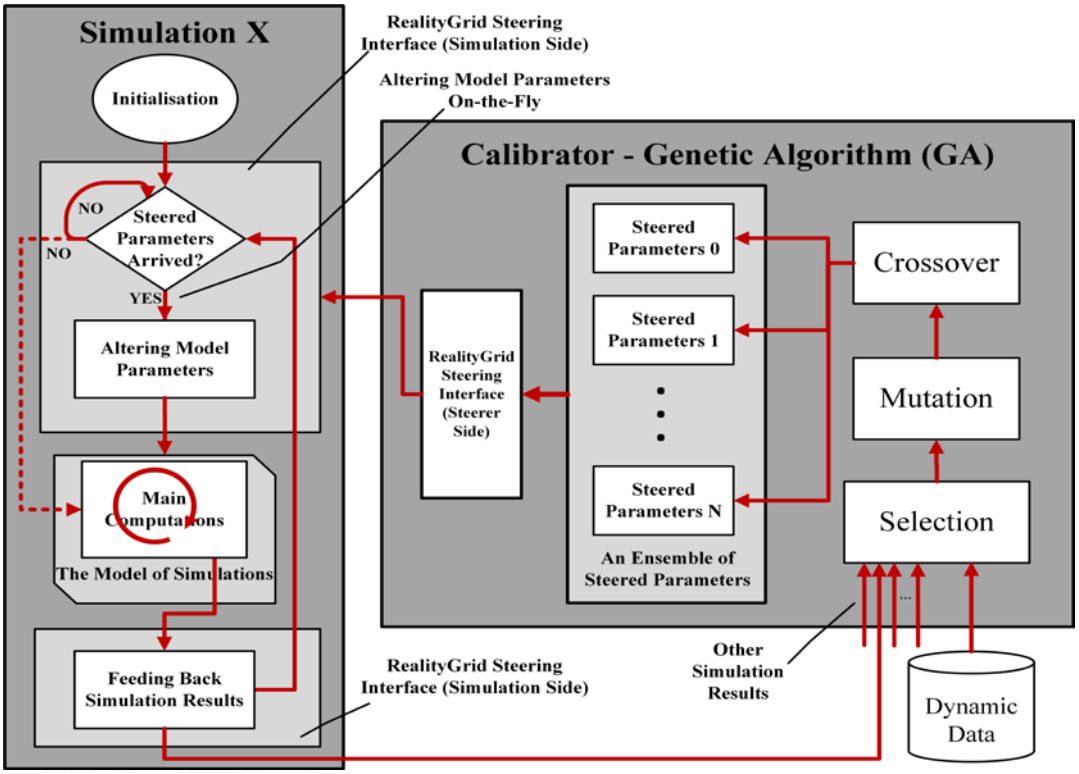


Figure 4-6 Steerable Interaction between Simulations and the Calibrator: The difference between the simulation shown in this figure and Figure 4-5 is that two RealityGrid Computational Steering Interfaces are inserted in the simulation. The steering interface below the Initialisation alters parameters of the model used by the Main Computations and variables related to the steered parameters. The Interface at the end of the simulation is used to send computation results to the GA. Only one simulation is shown as an example, in practice, multiple simulations can use such steering interface to communicate with the GA.

Since computational steering has been utilized to tackle such wasted time, this thesis integrates computational steering by opening the non-steerable simulations to create an interaction between the post-initialisation stage of simulations and GA. Figure 4-6 shows the steerable interaction between the Calibrator and Simulations. Two RealityGrid Interfaces are inserted. The first steering interface below the Initialization component is used to interact with the steering interface on the Calibrator side. It receives steered parameters from the

Calibrator and modifies local parameters based on the steered values. If it is the master process, it may also need to assign steered parameters to other processes. The steering interface inserted at the bottom is used to send the simulation results to the Calibrator. Because the GA does not require visualization from the simulation results, the visualization component which is one critical component in conventional computational steering is removed.

As the result, instead of starting a new execution after each time a new set of steered parameters is received, the main computations can start from the last state, which is the results by using last set of parameters. The new start is realised by re-initializing and computing states only related to steered parameters. Additionally, the Steered Parameters Arrived? component checks if there is a new steering input at the start of a new round of Main Computations. If no new steered parameters arrive, it has the following options, 1) continue waiting for new steered parameters or 2) continue to execute main computations based on current parameters. In the conventional computational steering, the second option is typically taken, and the Main Computations component constantly runs the loop during the steering. It emits intermediate results and checks if new steered parameters have arrived after a specific number of iterations. Hence, this method is appropriate for Human Users to steer the direction of a simulation even before it reaches the stopping criteria. However, in the DDDCS implemented in this thesis, the Steerer is the GA that cannot make such complex analysis on the running computations, and only analyses simulation results after the computations stop after reaching stopping criteria. Hence, it is not meaningful to continue running computations without a new set of steered parameters.

As a result, this method can reduce the time wasted on repeating initialisation. However, as we introduce two steering interfaces in the workflow, the time consumed by transferring steering information and processing steering decision can increase the execution time in calibrations. Therefore, for a calibration task, a specific number of interactions must be reached to ensure the benefit obtained from integrating computational steering is greater than the overhead. Additionally, the Simulation shown on the left is named as Simulation X since the GA needs to steer multiple simulations concurrently. As this section focuses on introducing the steering interactions between the GA and simulations, the parallel workflow will be further discussed in Section 5.3.1.

4.2.3 Dynamic Data

The Dynamic Data component is the interface between the physical systems and cyber system in the context of DDDAS. It can be for example the satellite data, manual observations, data from observation stations, sensor data, and data from other sources. This data needs to be collected in real-time, pre-processed in the format of the Calibrator inputs. Finally, it also needs to be transferred to the Calibrator in time. Since this thesis focuses on the high-level application of DDDAS, we do not study methods to handle challenges on real-time data transport and storage. Instead, based on our collaboration with water companies and water research utilities, this thesis can utilise historical sensor data that was collected every 15 minutes by the sensors developed in the real water network. Additionally, we also generate artificial sensor data based on the model that has been calibrated by water engineers. Thus, the calibration is assumed to take real sensor data collected from the physical world. These sensor data is our source of Dynamic Data and it is stored into a database that can be connected to the Calibrator. During the calibration, the Calibrator extracts the historical sensor readings as the real-time sensor data obtained from the physical world. By doing so, we abstract the data transport layer and only focus on using such dynamic data. The following figure presents an example of the sensor data stored in the MySQL[15] Database.

Name	Date	Time	Duration	Value	ForP
47295461	07/07/2016	00.00.00	15	48.0581	P
47295461	07/07/2016	00.15.00	15	48.2102	P
47295461	07/07/2016	00.30.00	15	48.2709	P
47295461	07/07/2016	00.45.00	15	48.341	P
47295461	07/07/2016	01.00.00	15	48.4274	P
47295461	07/07/2016	01.15.00	15	48.5144	P
47295461	07/07/2016	01.30.00	15	48.5649	P
47295461	07/07/2016	01.45.00	15	48.6401	P
47295461	07/07/2016	02.00.00	15	48.6954	P
47295461	07/07/2016	02.15.00	15	50.1333	P
47295461	07/07/2016	02.30.00	15	50.267	P
47295461	07/07/2016	02.45.00	15	50.3889	P
47295461	07/07/2016	03.00.00	15	50.5673	P
47295461	07/07/2016	03.15.00	15	50.7002	P

Figure 4-7 An Example of Sensor Data Stored in the Database: Sensor ID: 47295461. ForP indicates it is a Flow reading or a Pressure reading. Value indicates the pressure value in the unit of meter (height is the proxy for pressure).

[15] MySQL is an open-source relational database management system (RDBMS).

4.3 Architecture of Dynamic Data-Driven Computational Steering on Hardware Level

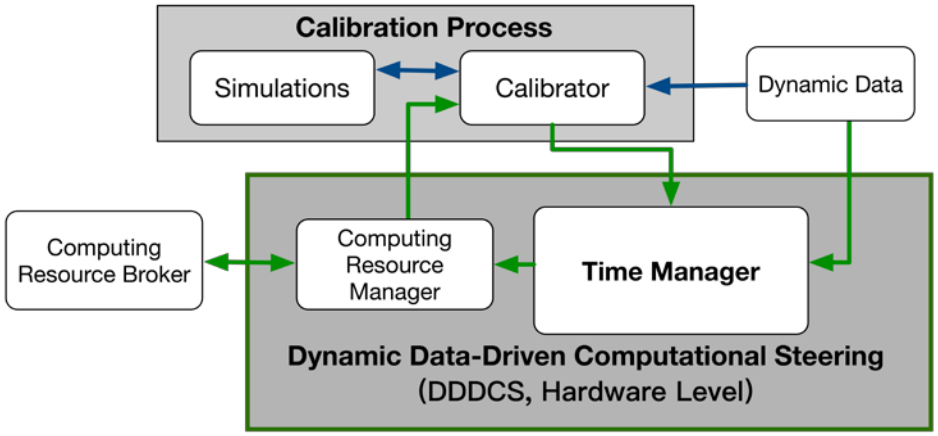


Figure 4-8 The Architecture of the Dynamic Data-Driven Computational Steering on Hardware Level: The Simulations and Calibrator belongs to the software-level DDDCS. The Calibrator, Time Manager and Computing Resources Manager form a feedback loop in which the computing resources required by Time Manager is based on estimating the execution time of the Calibrator. The estimation is based on historical execution time consumed by the Calibrator. As no interaction is created or facilitated by using computational steering techniques, we do not consider it as an individual steering loop. Thus, it is categorised as DDDCS on the Hardware Level (DDDCS-HL).

Based on the architecture of the hybrid computational steering, Figure 4-8 illustrates the architecture of the Hardware-Level Dynamic Data-Driven Computational Steering (DDDCS-HL). As this thesis is developed in context of Dynamic Data-Driven Application System (DDDAS), it is crucial to be able to restrict the execution time of programs and to provide sufficient computing resources to guarantee that tasks finished in the restricted time window can be finished with reliable quality. Therefore, two components, the Time Manager and Computing Resource Manager are included in this architecture. The Time Manager provides principles for assigning deadlines to components working on the DDDCS-SL and estimates the required amount of computing resources to make the calibration finish in time. The Computing Resource Manager applies for the required amount of computing resources from Computing Resource Broker, and provides the information of the applied computing resources to the Calibrator. Based on this information, the Calibrator coordinates with Simulations to adapt the parallelisation scale of the calibration process to assigned computing resources.

To meet the time requirement, the amount of computing resources is significant for DDDAS. However, the communication with the Computing Resource Broker is based on specific Application Programming Interfaces (APIs) and communication protocols provided by the

Computing Resource Broker. If we take the example of cloud computing, Amazon Cloud provides its particular APIs to apply for instances based on HTTP and REpresentational State Transfer (REST). Additionally, gird organizations also provide particular APIs for applying computing resources based on diverse standards such as Web Services Resource Framework (WSRF) and Open Grid Services Infrastructure (OGSI). This thesis intends to provide flexibility to integrate with different computing resource providers. Hence, a Computing Resource Broker is assumed to be implemented in the architecture. It is responsible for communicating with different resource providers and handle diverse interfaces. Since the development of the broker is beyond the scope of this thesis, we collaborate with another PhD student, Zeqian Meng, who focuses on the study of the communication between brokers and various resources providers. Hence, the Computing Resource Broker is benefited by Meng's work [100]. Based on interfaces provided by the Computing Resource Broker, the Computing Resource Manager of this thesis can apply for computing resources. However, in practice, it is not feasible to test the requests based on real grid and supercomputer computing resources. Hence, by managing to communicate with a Computing Resource Broker provided by Meng, this thesis demonstrates that it is applicable to implement the Computing Resource Manager in practice in the future work. For the evaluation, all resources are assigned manually. Since the details of requesting computing resources are not the focus of this thesis, the Computing Resource Manager will not be discussed explicitly. The rest of this section focuses on the design of the Time Manager.

The Time Manager is the key component in this architecture since it guarantees the value of steering results by taking the time of the physical world into in-silico simulations. This section reviews the time management developed in another research area, the scientific Workflow Management System (WMS), and introduces its time management methods for the hybrid computational steering system. Therefore, a review of time management methods is presented in Section 4.3.1. Based on methods discussed in the review, Section 4.3.2 discusses the architecture of the Time Manager including Deadline Assignment and Estimation of Execution Time. Since the implementation of the function Estimation of Execution Time relates to a complex hierarchy of knowledge, an individual section, Section 4.3.3, is dedicated to discussing it.

4.3.1 A Literature Review of Time Management in the Scientific Workflow

This section reviews the Time Management from four perspectives: 1) system structure, 2) quality of service, 3) deadline assignment and 4) estimation of execution time.

Workflow management has been a major research interest for around 20 years [156]. It was defined in the business domain in 1996 by the Workflow Management Coalition as: "*The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules*" [135]. With the advent of grid and application technologies, the workflow systems were studied to manage the grid-based scientific workflows in which distributed scientists can collaborate on conducting large-scale scientific experiments and knowledge discovery applications.

Since we refer to the time management methods of the workflow system to manage the time in DDDCS-SL, components in the DDDCS-SL architecture are considered as tasks of the workflow system. Generally, tasks of the workflow system have four types of structures: sequence, parallelism, choice and iteration. The sequence structure consists of a series of ordered tasks while the parallelism structures enable tasks to be concurrently executed. Moreover, in the choice structure, the priority of a task can be changed based on specific conditions. Finally, a set of tasks may be executed repeatedly in the iteration structure [156]. These types of structures are not mutually exclusive, and can constitute a composite structure. In the hybrid computational steering proposed in this thesis, steering loops on each layer are considered as the iteration structure. The condition component that checks whether new steered parameters have arrived and the condition component checks whether the new stopping criteria of GA has been reached fits into the choice structure. Additionally, the parallel steering fit into the parallel structure. Finally, since some of components in the DDDCS architecture need to be executed in a specific sequence, for instance, the execution sequence of Calibrator, Simulation, High-Level DDDAS Functions and Human Users fits into the sequence structure. Since no further structure types are discovered in the architecture of the hybrid computational steering, we suppose methods designed in the WMS structure can be used in the hybrid computational steering.

In terms of the quality of service in the workflow system, there are five main requirements, which are shown in Figure 4-9. However, only Time and Cost are considered in this thesis

since we are building a "just-in-time" system where the remaining three components are not of primary importance, although this could change in future work.

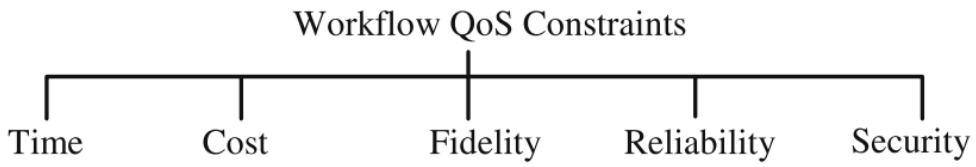


Figure 4-9 QoS Requirements of Workflow System[156]: Time and Cost are two factors considered as the Quality of Service in this thesis.

Time is a basic measure of performance, and it is also a critical measure of timeliness of system results. Generally, it represents the total time required to finish all tasks in the workflow, namely the time requirement of the global task. Specifically, it can also refer to the time demanded by each task individually. In order to keep the workflow working in a timely manner, it is essential to leverage deadline assignment strategies to manage the running time of both the global task and individual tasks.

In real-time systems, deadline is used to restrict the execution time of tasks so that a cyber system can keep pace with the system in physical world. Existing research has provided sophisticated protocols for the deadline assignment problem [77, 122] which can be adopted by the Time Manager of this thesis. Since this thesis does not aim at exploring the most optimal scheduling strategy but only a sufficiently good one, we introduce four deadline assignments methods which are widely cited, namely ultimate deadline, effective deadline, equal slack, and equal flexibility [77]. Before we continue to discuss these methods, we need to generally divide the deadline assignment problem into a serial task problem and a parallel task problem since this division determines the realisation of deadline assignment methods, and both types exist in our architecture.

For serial tasks, we can only apply the ultimate deadline if no knowledge of subtasks is available, and it can be expressed as the following:

$$dl(T_i) = dl(T) \tag{4}$$

The $dl(T_i)$ denotes the deadline of a subtask T_i, and $dl(T)$represents the deadline of global task. However, this simple method cannot indicate accurate information such as how much time the next task can be delayed. While, if an estimation of the subtask execution time is available, the effective deadline strategy can be applied as the following:

$$dl(T_i) = dl(T) - \sum_{j=i+1}^{m} pex(T_j) \quad (5)$$

pex denotes the predicted execution time (a duration of time and not a time point) and m is the number of rest tasks. However, this method has a problem named "little slack for final-stage subtasks", which means that tasks at early stages consume most of available time resources and leave a low probability for the rest of subtasks and even the global task to finish on time. A simple strategy to conquer this issue is to divide the total remaining slack equally among the remaining subtasks, and it can be referred to as equal slack strategy, formally,

$$dl(T_i) = ar(T_i) + pex(T_i) + \frac{[dl(T) - ar(T_i) - \sum_{j=i+1}^{m} pex(T_j)]}{m - i + 1} \quad (6)$$

The ar represents the arrival time point of a task. The $dl(T) - ar(T_i)$ calculates the amount of time available for tasks T_i to T_m. Hence, the distance between the deadline and arrival time of task T_i is calculated using the sum of predicted execution time of T_i and calculated equal slack.

Moreover, the total remaining slack can also be divided in proportion to their execution time. Thus, we obtain

$$dl(T_i) = ar(T_i) + pex(T_i) + [dl(T) - ar(T_i) - \sum_{j=i}^{m} pex(T_j)] * [\frac{pex(T_j)}{\sum_{j=i}^{m} pex(T_j)}] \quad (7)$$

Above deadlines are for the serial structure, for the parallel structure, the ultimate deadline can also be used since subtask are executed in parallel. Additionally, a global task always consists of both the serial and parallel subtasks, and strategies discussed in this literature review can be integrated to tackle problem together. Furthermore, in order to achieve the deadline assignment strategies discussed above, another crucial task is to measure the execution time of subtasks, $pex(T_j)$.

Estimating the execution time is a critical and challenging task [68]. Generally, there are three major approaches to estimate the execution time: code analysis, analytical benchmarking/code profiling and statistical prediction. In the code analysis, the estimation is based on the analysis of the number of instructions of a task. Since it is a low-level method, only specific tasks can utilize it [146]. Analytic benchmarking/code profiling was first

presented by Freund [46]. This method first defines a number of primitive code types. Afterwards, each type of code is executed on different machines to generate corresponding benchmarks. At the prediction stage, the code type and benchmarks are considered together to estimate the runtime. Lastly, the statistical prediction method estimates the runtime by analyzing historical observations. In this method, tasks are categorised into different groups, and a quantity of runtimes is recorded for specific combinations of task group and machine type. At the prediction stage, prediction results are obtained by referring to the task configurations, such as input parameters of tasks, and specifications of computing infrastructure. One advantage of this method is that the accuracy of the estimation can increase with the accumulation of the historical observations.

4.3.2 Design of the Time Manager

After reviewing methods utilized in the time management of the workflow management system, this section introduces how to implement the time management in the architecture of the hybrid computational steering.

This work belongs to the soft real-time system since a delayed result, can still reflect states of physical systems to some extent. Hence, the time management focuses on the deadline assignment only for those components that predominantly occupy the time resources. As discussed, the core layer in the general architecture of the hybrid computational steering shown in Figure 4-1 is DDDCS-SL, and the Simulations and Calibrator are the two primarily time-consuming processes in the workflow. Therefore, this thesis only considers the time management on the Calibrator and Simulations. Additionally, as the Calibrator and Simulations components are tightly bound with each other, this thesis only considers assigning deadlines to the process combined by these two components, namely the Calibration Process. As a result, the Estimation of Execution Time, which is the component used to support the Deadline Assignment, only predicts the execution time required by the calibration process as well. In the future, based on the complexity of applications users build on the calibration results, the Deadline Assignment component may need to constrain the execution time of the High-Level DDDAS Functions component.

It is argued that the time taken by the Estimation of Execution Time component on the hardware level DDDCS and Human Users on HDCS layer are significant as well. Hence, it is necessary to explain why this thesis does not consider them in the Time Management and deadline assignment. Firstly, the Estimation of Execution Time component, which will be

introduced in the next section, is based on a learning process and a large number of historical relations among running time, task configurations and machine specifications. Since estimating the execution time involves a large amount of historical processing that can be done offline and then used online, we assume that estimating the execution time is relatively stable compared to the time scale of calibration. Secondly, HDCS in this thesis is separated from DDDCS as two layers. This separation is based on the aim of providing human users sufficient time to conduct the steering without being concerned with the time limitation raised in the physical system. The separation also raises HDCS on a higher layer of which the steering decisions are made based on results of DDDCS on the lower layer. Hence, steering conducted by human users is considered as a "slow" real-time system in which Time Management is a subjective task operated by human instinct. Moreover, the time management of conventional computational steering focuses on minimizing the duration from issuing a steering command to observing the corresponding steering feedback. However, in the HDCS, the feedback of steering can only be shown to users at the speed of calibration process, and the time required by a calibration task in the water use case is at least in minutes. As this required time is much greater than the duration from issuing a steering command to getting the corresponding feedback, the issue considered in the time management of the conventional computational steering is not covered in this thesis.

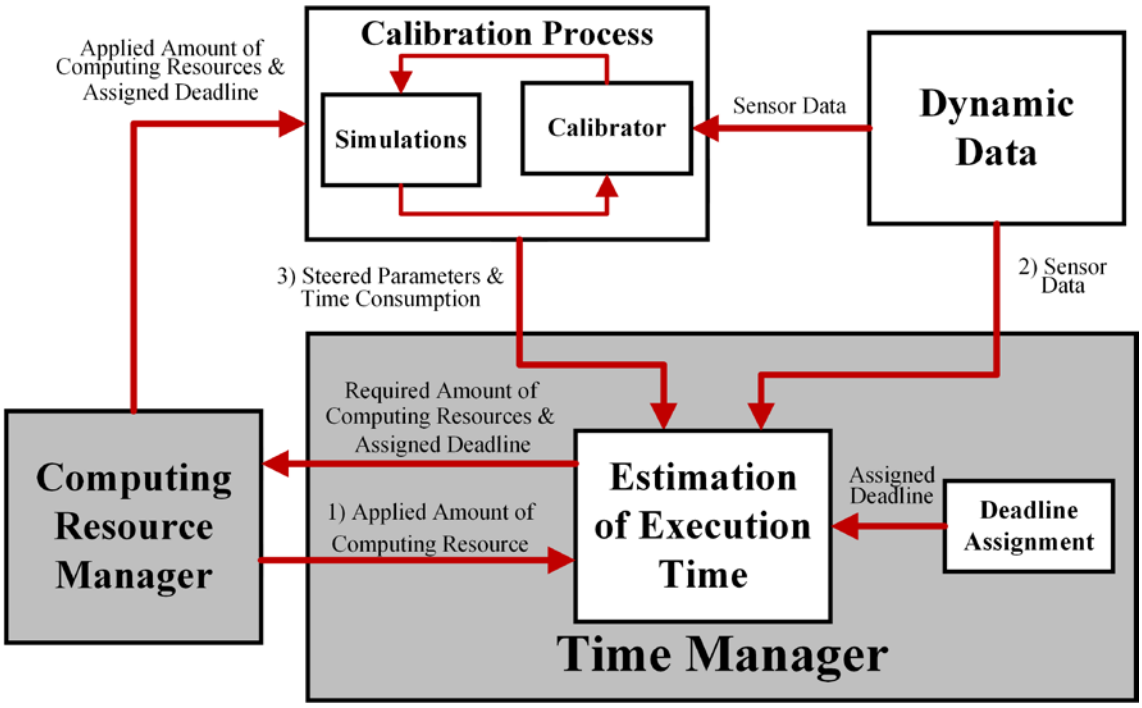


Figure 4-10 The Architecture of the Time Manager: Time Manager learns the relations among the amount of the computing resources, the sensor data, execution time of the calibration process and the Calibrator parameters, which are steered by the human users based on historical information. According to the studied

relations and the deadline required in applications of DDDAS, it request computing resources from the Computing Resource Manager by estimating the dynamic workload of the calibration based on the real-time sensor data.

Based on the architecture of DDDCS-HL, the architecture of the Time Manager is shown in Figure 4-10. There are two components inside the Time Manager, the Deadline Assignment and Estimation of Execution Time. The Estimation of Execution Time component utilises the Dynamic Data as the indicator of the workload of the calibration process. Based on this indication, it estimates the running time required by the calibration process. Since the deadline has been acknowledged by the Deadline Assignment component, the Estimation of Execution Time component needs to find out how much computing resources are enough for the calibration to be finished in time based on the estimated workload. Afterwards, it sends the required amount of computing resources to the Computing Resource Manager which further requests the computing resources from resource providers. The relation between amount of computing resources and time consumption based on a particular workload is maintained and learnt from a large number of such historical relations.

As we analysed, the architecture of the hybrid computational steering has a compound structure that includes all four structure types of a workflow management system. However, in this thesis, the time management only manages the time on the level of the Calibration Process. Hence, the parallelism and choice structures are not considered by the Time Manager. In addition, the loops on the hardware DDDCS and HDCS layers, which fit into the iteration structure, are considered as the offline workflow and “slow” real-time workflow, and they do not need to be manged by the Time Manager. Therefore, the iteration structure is not considered in our implementation as well. Finally, we conclude that the structure of this implementation only fits into the sequence structure in which the Calibration Process must be finished in time to guarantee that the following functions can benefit from the real-time Dynamic Data. As presented in Figure 4-1, the timeliness of the High-Level DDDAS Functions directly depends on the Calibration Process, the sequence structure that needs to be managed by the Time Manager consists of two components, the High-Level DDDAS Functions and the Calibration Process.

However, since the execution time of the High-Level DDDAS Functions can significantly vary based on specific implementations, its time management needs to be developed in particular projects. Therefore, this thesis assumes that the Calibration Process is the only component that needs to be managed. Subsequently, the Ultimate Deadline is selected as the deadline assignment method in this thesis, and the other three deadline assignment methods

are suitable for a more complex structure. As a result, the deadline of the Calibration Process, $dl(T_{cp})$, can be represented as $dl(T_{cp}) = dl(T)$ in which T denotes the global task. As the hybrid computational steering is driven by the Dynamic Data, the interval between two updates of the Dynamic Data is considered as the deadline of the global task, $dl(T)$. Finally, since this thesis uses a water distribution system as the use case, and a probable updating frequency advised by water engineers is 15 minutes, we finally set $dl(T_{cp}) = dl(T) = 15\ minutes$ in the implementation.

In terms of the QoS, this thesis focuses only on the time and cost. Thus, the strategy of applying for computing resources is divided into market-driven and performance-driven strategies. The main purpose of the use of dynamic computing resources is to provide sufficient computing power to guarantee the time requirements. Under the premise of assuring the time requirement, dynamic computing resource can provide us the flexibility of saving cost. Hence, the performance-driven strategy is first considered, and only when computing power is sufficient, the market-driven strategy can be applied.

The only component left without discussion in Figure 4-10 is the Run-time Estimation. Since its implementation relates to a complex hierarchy of knowledge, an individual section, Section 4.3.3, is dedicated to discussing it.

4.3.3 Estimation of Execution Time

Estimating the execution time of tasks is challenging [68]. In the literature review at the beginning of this section, Code Analysis, Analytical Benchmarking/Code Profiling and Statistical Prediction are discussed as general methods to realize the estimation of execution time. Both the code analysis and code profiling methods focus on analyzing structures, functions and algorithms of programs. However, by demonstrating the calibration process based only on the GA with specific implementation methods, the uncertainties of time estimation are mainly raised by dynamic information such as dynamic data, dynamic computing resources and GA parameters. Hence, this thesis focuses on the influence of factors that are not addressed by the Code Analysis and Code Profiling methods. On the other hand, the statistical prediction method provides the relation between the required execution time and influential factors, which can affect the execution time as a black box. Thus, it does not require specific knowledge on information such as details of computing infrastructure and algorithms used in computations, and it does not have limitations on types of influential factors. Therefore, this thesis selects the statistical prediction as the

fundamental method to develop the Estimation of Execution time component. As a result, the estimated running time of the Calibration Process, $Est.Time$, is modeled as a function:

$$Est.Time = pex(X) \quad (8)$$

where X denotes the vector of factors that affect the execution time, and the $pex()$ represents the algorithm to implement the black box which maps the factors to $Est.Time$. The factors that constitute the vector X will be denoted as Influential Factors. This section discusses a) the four factors that constitute the vector X and b) the algorithm used to implement the $pex()$.

Four Influential Factors

Four types of factors are considered in this thesis: 1) The first one is the sensor data which represents the dynamic information observed in the physical system, and it can be used to indicate the calibration workload. Using the term Sensor Data is because this thesis only takes sensor readings as the dynamic information in the use case. 2) The second type is the amount of computing resources that can be applied from resource providers. Since the parallel programming is utilised by the Calibration Process, the amount of computing resources can determine the parallelism scale of the program. 3) The third information is the values of steerable parameters of the GA. 4) Finally, the inner loop of Main Computations can also contribute to the flexibility of execution time of the Calibration Process. Before discussing the method of mapping the sensor data and amount of computing resources to the running time, it is necessary to explicitly explain the usage of these factors.

The required execution time of the Calibration Process depends on the amount of computing resources and the workload. In terms of the computing resources, the Time Management method reviewed in WMS is based on the grid computing infrastructure. Though grid is one possible option for this thesis, we only implement the hybrid computational steering on local computers, supercomputers and commercial cloud computing infrastructures since we do not have access to available grid computing resources. Since specifications of machines provided by different providers can vary significantly, the amount of computing resources is indicated by two types of information: the computing resource provider and the core number. In our use case, a Macbook Pro, an IBM Blue Gene/Q supercomputer and multiple Amazon Cloud computing instances are utilized as examples of different types of machines provided by resource providers.

Since this thesis focuses on scalability of computing resources, only the number of cores offered by these providers are considered as the scalability to support parallel computing. It is necessary to clarify that Amazon Cloud computing system can provide different type of instances. However, as a demonstration, this thesis does not comprehensively evaluate all specifications of instances. For the implementation on the Amazon Cloud, the same type of instances, which contains only one core and same cache, memory and network configurations, are applied. Hence, the number of instances indicate the number of cores.

Additionally, in the context of large-scale physical systems, the sensor readings can include a great amount of data, and using all of them can greatly increase the space size of vector X. Hence, it is necessary to extract the workload represented by sensor data in a simpler format. To discuss the method of simplifying the information, we first make four assumptions. The calibration is a process of exploring the parameter space of a model that converts the simulation from state S_n to S_{n+1} in which n denotes the simulation time step; hence, 1) we assume parameters that are used to obtain state S_n and S_{n+1} are pv_n and pv_{n+1}, and the calibration starts to explore the parameter space from pv_n. 2) Additionally, we assume the distance between pv_n and pv_{n+1} indicates the workload for the Calibration Process. 3) Finally, we assume the distance between pv_n and pv_{n+1} has a positive correlation with the difference between S_n and S_{n+1}. Since S_n and S_{n+1} can be directly represented on sensor readings SR_n and SR_{n+1}, the difference between S_n and S_{n+1} can be indicated as the difference between sensor readings SR_n and SR_{n+1}. Hence, the difference between SR_n and SR_{n+1} can have a positive correlation with the distance between pv_n and pv_{n+1}, which finally indicates the amount of the workload of the Calibration Process. Consequently, we transfer "using sensor data to directly indicate the workload of Calibration Process" to "using the difference between sensor data at time step n and $n + 1$" to indicate the workload. By doing so, it is possible to utilize measures to indicate the difference between values of a large amount of sensor readings in a single number. For instance, Root-Mean-Square Error (RMSE) and Mean Absolute Error (MAE) are such measures that have been largely applied to compare states of large-scale systems in areas such as meteorology, hydrogeology, etc. There are a number of arguments of selecting the "right measure" to signify the difference [24], and this thesis leaves the choice to future users and does not further compare them.

Then, we discuss the steerable parameters of GA. It is significant to differentiate them from the steerable parameters of simulations. The steerable parameters of GA are steered by users on the higher steering layer, HDCS. For example, users may need to increase the updating

frequency of a monitoring application. The monitoring application is driven by parameters calibrated in the Calibration Process. Hence, it is possible for application developers to register the stopping criteria of GA as a steerable parameter. Since such parameters can significantly affect the execution time of GA, this thesis takes it as intervention raised by HDCS and its effect must be taken into consideration in Estimation of Execution Time.

Finally, the fourth factor is number of iterations of Main Computations as discussed in the Simulations component of the architecture of DDDCS-SL. Since simulations targeted by this thesis need to be executed on HPCs, we assume they have complex computations and may have an uncertain number of iterations. For simulations that have a stable number of computations, they do not account for a large proportion of the uncertainty. Hence, their execution time can be directly estimated. In terms of simulations have a great uncertain number of iterations, we have categorized them based on four stopping criteria: 1) a specific number of iterations, 2) a number of iterations after stopping improves, 3) reaching a specific value that indicates the results quality and 4) the final simulation state is reached. Except for the first stopping criteria, the others raise significant uncertainty on the execution time of simulations. Since the time management on simulations are use case specific, this thesis takes the first stopping criteria as an example in the architecture design. For future studies, this factor can be added to the vector without changing the estimation method discussed in the next section. For example, in the model, EPANET, which is used to model the water distribution system in our use case, its main computation loops to search for the best hydraulic solution. This thesis uses the default iteration number, 40, which is provided by the EPANET as the static number of iterations. Consequently, in the remainder of this thesis, the iterations of Main Computations are static and not considered as an influential factor, and the execution time of simulations are assumed to be stable.

Consequently, by using the $Cores$ and $Provider$ to indicate the number of cores and name of computing resource providers, $Sensor\_Diff$ to indicate the difference between sensor readings at two time steps and $H(h_1, h_2, \dots, h_n)$ to signify steered parameters. The vector X of Equation (8) consists of three influential factors and can be presented as $X = (Cores, Provider, Sensor\_Diff, H(h_1, h_2, \dots, h_n))$.

The Estimation Algorithm

After discussing the vector of influential factors that affect the running time of the Calibration Process, this section further discusses the algorithm used to map the influential

factors to the running time. Classification tree is a widely-used type of algorithm in the prediction of required running time of programs [96, 113, 115]. Compared with Linear Regression and Support Vector Machine [39, 129], classification trees can ignore limitations raised by linear correlation between running time and influential factors and consume less computing resources than Vector Machine. Thus, this thesis utilizes the Classification Tree as the method to manage relations between execution time and influential factors. If the training sample ensemble of the tree is $S = (s_1, s_2 \dots s_n)$, then each sample s_i can be represented as $s_i = (T_{exe,i}, X)$ where T_{exe} denotes the execution time, and X is the vector of the three influential factors we discussed above. Consequently, a simple example tree is shown in Figure 4-11.

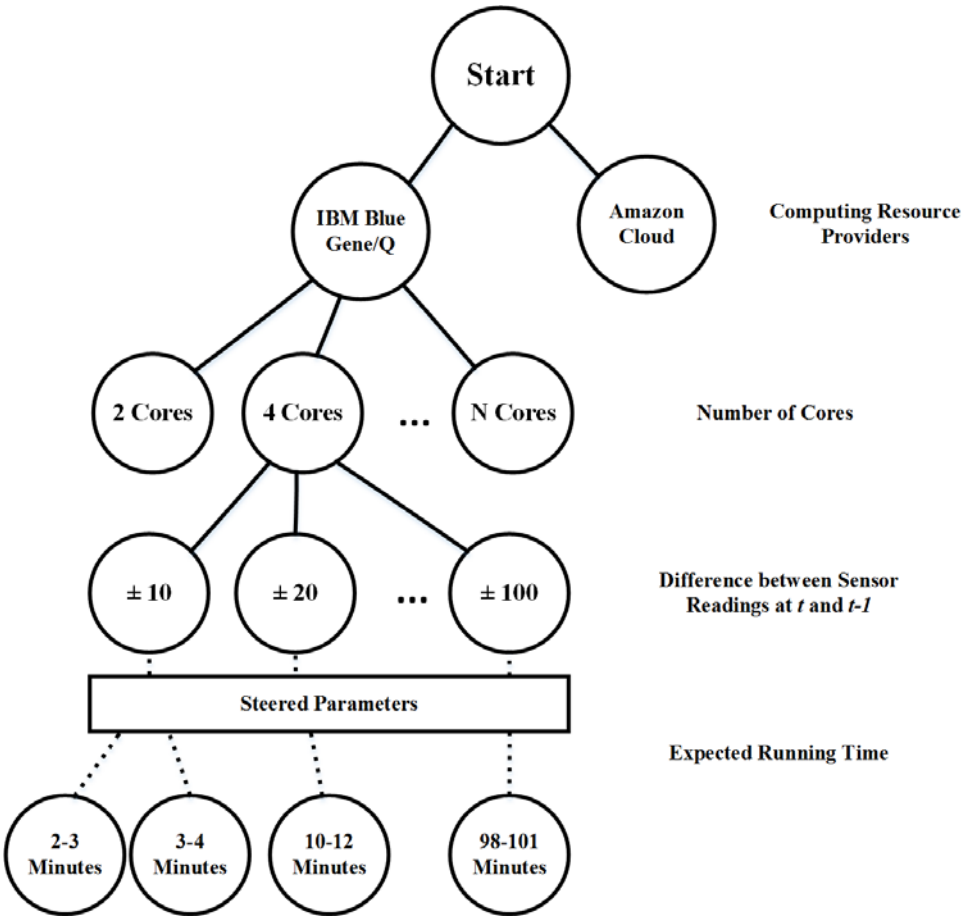


Figure 4-11 A Simple Example of the Classification Tree: All influential factors that affects the execution time are considered in nodes shown in the middle. Since Steered Parameters will be discussed in the evaluation chapter, this figure does not depict specific example for parameters that can be steered. The leaves of the tree are the estimated running time.

The tree shown in Figure 4-11 is only an example. In practice, conditions on nodes can be composite. The leaves of the tree are the estimated execution time; however, the estimation results are presented as a time range instead of a specific number. This is because this thesis

takes uncertainties raised by GA into consideration. In practise, the execution time of the Calibration Process fluctuates even with the same configuration and task since the execution time also depends on the probability of evolving towards the goal of convergence. Hence, the relation between a set of influential factors and the corresponding execution time can be one-to-many instead of one-to-one.

Consequently, this thesis further studies the relation between influential factors and execution time of GA. Since stopping criteria of GA implemented in this thesis is the fitness value, for a particular configuration of GA, its execution time can be represented by the number of generations that it consumes to reach the stopping criterion. As a result, a new relation can be built between Gen_j and $(Sensor\_Diff, H(h_{1,}h_{2,}\dots,h_n))$. The Gen_j denotes the required number of generations for the GA to converge at the j_{th} execution, and the specifications of hardware are not included in to the relation because it only affects the execution time per generation but not the number of generations. Multiple executions, $j = (1, 2, \dots J)$, of GA based on the same configuration and task can generate an ensemble of such relations that can be utilized to study the distribution of required generation numbers. In this thesis, we assume this distribution fit into the Gaussian distribution since operations such as the mutation and crossover of GA are based on random number selected from Gaussian distribution. Based on this assumption, we further establish the estimated number of generations, $E_{e,}$ equals to $\sigma + \mu$ where σ represents the standard deviation, and μ denotes the mean of the distribution. The reason to use the $\sigma + \mu$ is that in a Gaussian distribution, 84.2% of the distribution of execution times are less than E_e. Hence, based on this execution time estimation, 84.2% of calibration can has enough time to finish. This thesis believes that 84.2%, as the estimation accuracy, is acceptable in our system and the extra 15.8% of results, although do not have enough time to be finished, are assumed to be close to the final results.

Additionally, to be compatible with the deadline which is in the form of Wall-Clock Time (WCT), the number of generations needs to be mapped with the WCT. This requirement connects the distribution of generations with one influential factor, which is the amount of computing resources. Since the GA implemented in this thesis does not have functions to make self-tuning, and the simulations it communicates with have stable execution time, it is assumed that unless being affected by the $Sensor\_Diff$ and $H(h_{1,}h_{2,}\dots,h_n)$, configurations of the GA do not change during its execution time. Hence, with a particular

simulation model and configuration of GA, we assume the number of generations and the WCT are in a linear relation. Thus, the linear regression is utilized to generate coefficients of linear functions that represents their relations. Since the slope of the linear functions are determined by the $Core$ and $Provider$. Therefore, two decision trees are utilized together to estimate the execution time as shown in Figure 4-12.

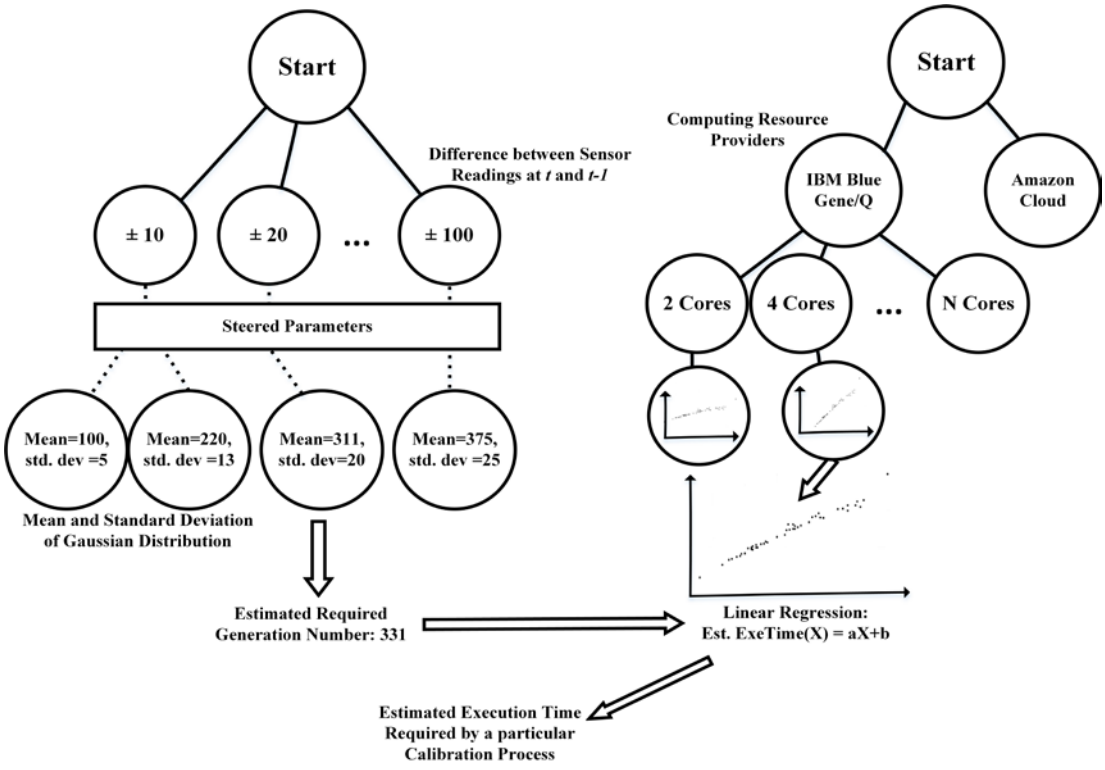


Figure 4-12 Two Classification Trees Used to Make the Execution Time Estimation: Nodes of the left tree are conditions that affect a required number of generations. Since Genetic Algorithm has a variable required generation number even using a static configuration, we assume the distribution of the required generation number, after following a path on the tree, fit into Gaussian distribution. Hence, each path leads to a leaf that indicates mean and standard deviation of the Gaussian distribution instead of a specific generation number. By assuming the sum of mean and standard deviation is greater than the required generations numbers in 84.2% of GA, a specific estimated generation number is generated. On the right tree, conditions are Computing Resource Providers and number of cores. Based on the assumption that the relation between the number of generations is in a linear relation with a particular computing resource, such as 4 Cores on Blue Gene/Q. Hence, based on historical execution information, coefficients of the linear relation function of the number of generations *X* that represents the execution wall-clock time can be obtained. By combining two leaves of each tree together, an estimated execution time is generated.

4.4 Chapter Summary

This chapter introduces the general architecture developed for functions and components raised in the theory of the hybrid computational steering. To realise the hybrid steering and time management, the general architecture is divided into three layers: Human Driven Computational Steering (HDCS), Dynamic Data Driven Computational Steering (DDDCS)

and Time and Computing Resource Management. Additionally, this chapter discusses methods used to develop components on sub-architectures of each layer.

In the development of HDCS, RealityGrid Computational Steering is utilised to realise the steering interactions between human users and computations. Different from conventional computational steering, this high-level HDCS steers parameters of the Steerer of the DDDCS. Additionally, instead of using visualisation to directly present results of steered simulations, a High-Level DDDAS Functions component is utilised to represent high-level functions based on the results of DDDCS.

On the layer of DDDCS, a Genetic Algorithm (GA) is used as an example of calibration algorithms to compare the batch and steering interaction of conventional and steerable calibration process. Then, RealityGrid Computational Steering is used as the steering toolkit to develop the steering interaction between the Calibrator and Simulations components. Different from conventional computational steering, the Calibrator does not require a visualisation component since it can directly analyse output emitted by the Simulations. Additionally, we restrict types of simulations that can be steered in this architecture and use the workflow of targeted simulations to present reasons that computational steering is applicable to improve the efficiency of interactions between Simulations and the Calibrator. Furthermore, a steering-integrated simulation workflow is implemented in the architecture of the DDDCS. It interacts with GA which has the ability to run in parallel. Additionally, it is modified based on the steerable simulation workflows of conventional computational steering. In this workflow, the Steerer only analyses results of simulations after the loop of main computations of simulations finish, and computations only check if the steered parameters have arrived when it starts a new round of iterations. Finally, the sensor data provided by water companies is provided as an example of dynamic data.

Since time management has rarely been studied in DDDAS, this thesis introduces time management methods of the scientific Workflow Management Systems (WMS). By projecting structures of developed architectures onto structures of WMS, this thesis choses ultimate deadline assignment and historical classification trees to constrain execution time and estimate required computing resources to meet the constrained execution time. The calibration process constituted by Calibrator and Simulations are assumed to be the predominant time-consuming process in the entire architecture, and factors that can affect the execution time of the calibration process are studied.

Even though the architecture is developed in the modularised structure and aims at being used generally, this chapter still discusses it under several assumptions and specific use cases. This is because, although introducing the development of the architecture, this chapter also needs to introduce methods this thesis utilises to design the infrastructure-level frameworks and evaluate the use case, a Water Distribution System (WDS). Therefore, this chapter demonstrates that it is feasible to build a compatible architecture based on the theory of the hybrid computational steering for the specific WDS use case. Future applications of this architecture can be built based on its flexibility.

Chapter 5 Computational Steering on High-Performance Computing Resources

The aim of this chapter is to introduce the development of hybrid computational steering framework. The framework is used to support implementing the architecture of the hybrid computational steering on the High-Performance Computing (HPC) infrastructures, such as supercomputers and cloud systems. As the hybrid computational steering consists of Human-Driven Computational Steering (HDCS) and Dynamic Data-Driven Computational Steering (DDDCS), this chapter is divided into implementing HDCS on HPC infrastructure (Section 5.2) and implementing DDDCS on HPC infrastructure (Section 5.3). This thesis believes that the best way to develop the HDCS framework is to learn from existing computational steering frameworks. Thus, we worked in close collaboration with IBM to integrate computational steering with their Blue Gene/Q supercomputers. However, repeating implementations of existing steering frameworks is not sufficient for a PhD study. Thus, based on a gap we found between conventional computational steering and HPC resources, we proposed to provide computational steering as a dynamic web service. Hence, Section 5.1 introduces our thoughts on the computational steering service. Additionally, Section 5.2 implements such thoughts in the project collaborated with IBM, and the collaboration result is used as an example framework of implementing HDCS on HPC infrastructures. Moreover, Section 5.3 introduces the framework of DDDCS that is developed and based on both the framework of HDCS and the computational steering web service. Finally, Section 5.4 integrates the frameworks of HDCS and DDDCS to generate the ultimate framework of the hybrid computational steering.

5.1 Beyond the Steering Toolkit to the Steering Service

One efficient way to maintain the increase of the computing speed is to increase parallelism scale by developing multi-core processors. This leads to non-uniform communication patterns in HPC infrastructures (within processors and between processors) and hybrid models of parallel programming (shared-memory and distributed-memory paradigms). The drive for higher performance has made writing efficient programs for such top-end HPC infrastructures a highly specialised task. It is further challenging to communicate with supercomputers and interact with their running programs. Additionally, the specialised

architectures and programming environments of HPC resources make it difficult for users to analyse the results of their programs especially when the volume of results is considerable. Specifically, users need to either manually transfer results into visualisation component or program dedicated components that can only fit into specific HPC architectures.

Computational steering has been studied explicitly to deal with this problem. Although computational steering can increase the efficiency of running simulations on HPC infrastructures, the process of implementing computational steering is a burden for regular users. One reason many of the early systems were not sustained can be that they did not give enough return to users. As discussed in Section 2.1.2, of the successful systems, COVISE [151] and SCIRun2 [159] have moved beyond providing a steering toolkit to providing rich computational environments, which incorporate computational steering. To scientists and engineers, such computational steering environments provide specific computational models that are specifically integrated with computational steering. Additionally, components of computational steering, such as visualisation and user interfaces, are developed on dedicated computing infrastructures so that regular users only need to focus on their simulations instead of on steering environments. Therefore, these steering environments reduce the costs of utilising computational steering and provide platforms for model developers to contribute more “steering-ready” models to the steering family.

As a result, an interesting future work can be generalizing the use of computational steering. This future work requires contributions from model developers who can provide more steerable computational models for regular users. It also requires model developers and HPC owners to create a general understanding of the importance of utilising computational steering among HPC users.

However, as indicated in our survey introduced in Section 2.1, the development of computational steering has always been related to the development of its application environments. As its fundamental computing infrastructures, dedicated supercomputers are expensive and difficult to be maintained by scientists. Hence, this limitation impedes developers from integrating computational steering with more computational models. As a result, computational steering environments were largely integrated with grid computing environments around 2005.

The grid provides collaborative computing environments that share distributed computing infrastructures. Hence, it makes HPC infrastructures more accessible by providing them as

grid services. Based on the grid service, peripheral services are provided as the middleware between HPC infrastructures and end-users. The computational steering service is one of them. However, based on this understanding of computational steering service, the implementation, such as communication protocols and frameworks, of computational steering service depends on specifications of different grids. Considering that grid is developing at a fast speed and its implementation is largely diverse and rapidly altering, it is challenging for the steering service to be updated with a compatible grid developments and it is further challenging to convince model developers to keep updating their models with an unstable steering environment or to restrict their development on specific grid environment. One example of the disadvantage of considering computational steering to be a peripheral service can be seen in the development of the RealityGrid steering environment [55].

Based on this situation, this work proposes to treat the service of computational steering as being on the same level as the grid service. Thus, computational steering service can act as a client of a computing resource providers, but not an accessory service that is bound to the grid. Additionally, an interface can be built on the steering service to specifically deal with the communication with computing resource providers. Hence, the interface can isolate the development of steerable models from the development of communication with computing resource providers. As a result, model developers can have a stable steering environment to contribute to develop steering-enabled models. Finally, since a computational steering service treats grid just as a computing resource provider, we can extend the scope of computing resource providers to commercial clouds and supercomputers, abstract functions of resource providing and focus on developing the framework for HPC infrastructures.

To implement such a computational steering service, we need to choose a steering toolkit that 1) has the flexibility of being implemented on different infrastructures, 2) is still being used, and 3) can be easily integrated with general models used on HPC infrastructures. The RealityGrid steering library, as the latest “alive” steering toolkit, was explicitly designed to deal with the problems of steering on high-end supercomputers. Additionally, it adopted a loosely coupled architecture where the computational components could run independently from the visualisation and the graphic user interface. This design of the RealityGrid can be adapted to several different HPC structures that may have limitations on visualisation and communication with external users. Moreover, not wishing to be tied to any standards that might prove to be ephemeral, RealityGrid steering is also able to adapt to other service-based approaches or even to low-level communication libraries.

Benefitting from the loosely coupled structure of the RealityGrid steering library, this project proposes a solution in which computational steering can be offered as a “standalone” web service. As a standalone service, it can separate the efforts made by model developers from that of the steering infrastructures developers. The word *standalone* means that our proposed computational steering service is different from some existing steering services, which are highly dependent on the grid environment. In addition, this work takes the IBM Blue Gene/Q as an example of an HPC infrastructure to represent how to implement the steering service with a specific HPC infrastructure.

However, to realize such CSWS requires massive support from computing resource providers including negotiation protocols and communication frameworks. Such functions have been dedicatedly studied in the area of grid computing. Thus, the CSWS can act as a client of the grid to request computing resources that support computational steering. Nevertheless, as stated in Section 4.3, the development of communication and protocols between the Blue Gene/Q system and the Computing Resource Broker Service is a dedicated research area that is beyond the scope of the collaboration project and this thesis. As we only aim at providing a promising direction of the development of computational steering, this thesis does not explicitly develop components to communicate with providers such as grid in the framework developed for the collaboration. Instead, we use the Blue Gene/Q system as an example to present the effect of computational steering that can be provided if the expected support can be obtained from computing resource providers.

Moreover, the development of the CSWS server is also based on the requirement of providing the IBM Blue Gene/Q as a service for users of computational steering. Hence, instead of the expected use of the CSWS in the future, the development of CSWS also needs to address requirements such as flexibility, security, user-friendliness and communications. Such requirements are addressed in the collaboration project and considered as objectives of the current CSWS. Finally, the development of the CSWS is introduced in the following section. It presents a general framework for the so-called “mobile supercomputing” conception proposed by IBM *[virtual science on the move].

5.2 Implementing Human-Driven Computational Steering on IBM Blue Gene/Q

Referring to the three-phases theory raised by Xu[155], the modern HPC era is from 1976 to 1994 in which the performance of computing is considered to be the most important concern. Afterwards, the scalability of HPC infrastructures became the most significant feature in the second phase, which spans from 1994 to the present. Currently, we are entering the third phase in which the efficiency will become the priority of HPC systems. Besides improving the energy efficiency, we believe increasing the efficiency of using HPC systems is also a crucial challenge.

According to our collaboration with the IBM Thomas J. Watson Research Center, scientists and engineers in IBM believe integrating computational steering with supercomputers is an promising approach to tackle such crucial challenge and improve the efficiency of running simulations on supercomputers. To facilitate the use of computational steering on supercomputers, this section discusses a flexible framework developed for the Computational Steering Web Service (CSWS) that provides steering-enabled software models as a software service for users. The feedback obtained from IBM engineers is attached in Appendix C.

Moreover, it is necessary to emphasise that the computational steering service is the outcome from the collaboration, and initially it is only a contribution to conventional computational steering. Except for the realization of the visualization component, the computational steering interactions of the proposed Human-Driven Computational Steering (HDCS) are not different from the conventional computational steering based on our analysis. Thus, we assume that the framework developed for the collaboration project can be further used as the framework of the HDCS.

5.2.1 The Computational Steering Web Service

The Overview of the New Framework

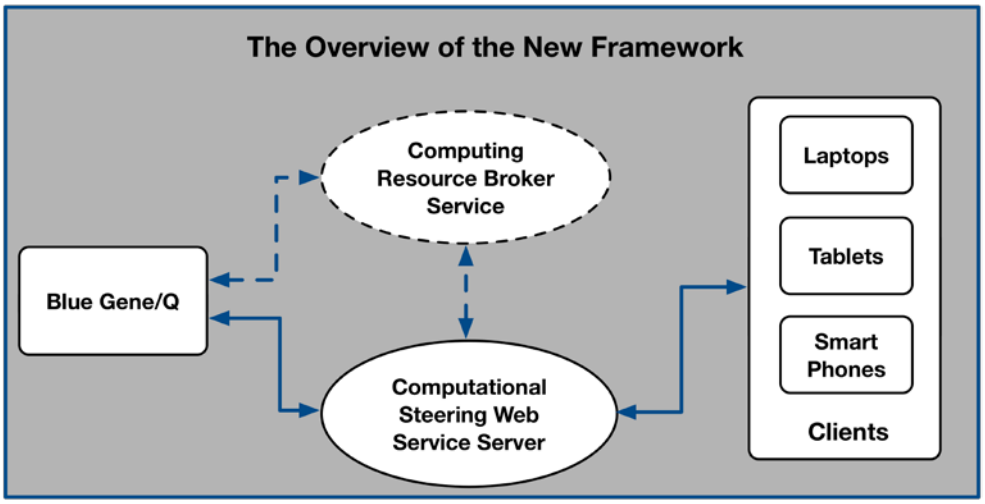


Figure 5-1 The Overview of the New Framework: The Computing Resource Broker Service on the top provides computing resources as a service. The Computational Steering Web Service Server hides computing infrastructure services, such as grid interfaces, and make users focus on their simulations. The Blue Gene/Q stands for HPC infrastructures which needs to register itself as a steering-enabled computing resource provider on the Computing Resource Broker Service. It also needs to indicate the steering-enabled models it supports. The dotted arrows and components indicate communications and functions proposed in this thesis but are not realised in current implementation.

An overview of the CSWS framework is shown in Figure 5-1. It includes the Clients, Computational Steering Web Service Server (CSWS server), Computing Resource Broker Service and Blue Gene/Q supercomputers components. To make this design compatible with more light-weight client devices, web browsers are used as the platform for running the client-side steering application. This application acts as a user interface for steering parameters, monitoring states of running simulations, and visualising real-time simulation results. Since this client-side steering application is built as a web application, it can be easily integrated into native Apple, Android and Windows operating systems.

Additionally, the web application is hosted by the CSWS server. The CSWS server keeps connections with the clients open in order to stream steering information and visualization results in real-time. This is a critical requirement to realize the real-time steering in the CSWS. After clients submit a request to the CSWS server, computing resources that meet the requirement of users are requested through the Computing Resource Broker Service. Afterwards, the Computing Resource Broker Service returns information of the Blue Gene/Q system to the CSWS server. According to the returned information, the CSWS server submits tasks to the resource provider and sets up several TCP connections to the Blue

Gene/Q. Because the identity of CSWS server is checked by the Computing Resource Broker Service, the Blue Gene/Q does not need to manage computational steering users.

As the Computing Resource Broker Service is not developed in the collaboration project, we only demonstrate the workflow after the Blue Gene/Q has been successfully assigned to the CSWS. The remainder of this section introduces our design of the CSWS Server and development on the Blue Gene/Q system.

Computational Steering Web Service Server

Figure 5-2 shows the design of CSWS. By referring to Figure 5-1, all communications on the left side are to/from the Blue Gene/Q system, and all communications on the right side are to/from users clients. Different colours are only used to differentiate components and communications.

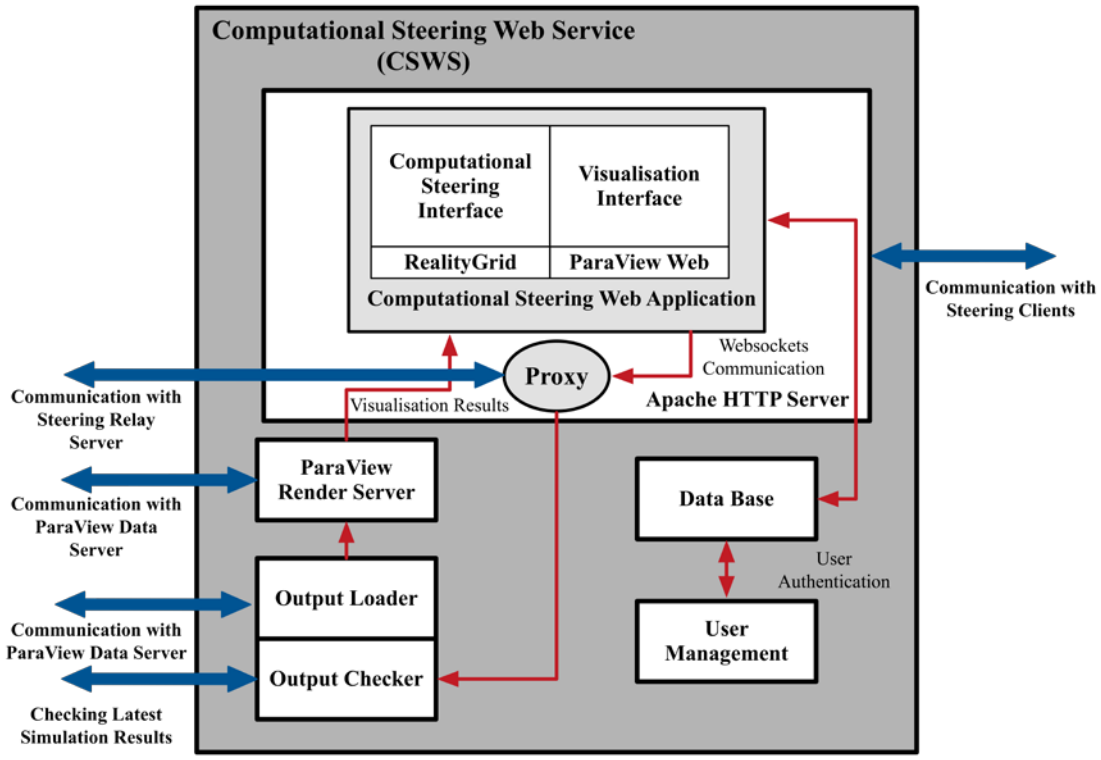


Figure 5-2 Computational Steering Web Service Server: This figure shows the internal framework of the CSWS server. The Blue Gene/Q system is on its left and the users are on its right. The Apache HTTP Server hosts a web application for users to steer parameters and view results of simulations. The visualisation of the web application is developed using the ParaView library. It also uses a proxy to transfer steering commands to the Blue Gene/Q system. The Output Loader and Checker are used to check states of simulation results and coordinates other components to work based on status of simulation results.

To serve the web application to users, an Apache HTTP server is applied to the CSWS server. The web application contains three main parts. The first part is the Computational Steering Interface. It is a set of web interfaces that are built based on the RealityGrid Computational

Steering toolkit. It obtains real-time steering information from the simulations running on the Blue Gene/Q system and transfers this information to end users. The Visualization Interface is realised using the ParaViewWeb library[16]. This library supplies Javascript APIs for users to integrate a viewport into the web application. The viewport has a global ID, which will be passed to the ParaView Render Server. The render server builds connections to the visualization interfaces and to render the viewport when it receives requests from the Output Loader. The advantage of using the ParaViewWeb library is that it can process inputs obtained from the end-user to generate new images. Thus, users interact with the view since the ParaViewWeb can generate new images by processing user operations. For example, users can rotate the image, zoom in and out, and change which portion of the data can be displayed.

The visualization data and status of steering programs are transferred to the web application in real-time via WebSocket. The reason for using WebSocket is that users need to obtain real-time feedback from running simulations. Originally, users needed to send requests to the CSWS server to check the status and outputs of their simulations but with WebSocket, servers can push information and data to the web application without requests from clients Hence, a lot of handshaking is eliminated. Since the Apache HTTP server does not support WebSocket during the development time, a Proxy is designed to handle WebSocket requests from users. There are three servers behind the proxy: the Output Loader, Output Checker, and Relay Server. The Relay Server is deployed on the Blue Gene/Q system to relay steering information between the CSWS server and Blue Gene/Q system, and it will be introduced in the following section. The Output Checker connects to the Blue Gene/Q system to check whether a new output has been generated. If there is one, it sends a signal to the Output Loader, which asks the ParaView Data Server to open the newly generated output file and sets up initial attributes about how to display the output data. Finally, the Output Loader commands the data server to pass the processed data to the ParaView Render Server which renders the data on the viewport of web applications.

[16] ParaView is an open-source, multi-platform data analysis and visualization application. http://www.paraview.org/

Since protocol frameworks of WebSocket such as Web Application Messaging Protocol (WAMP) [17] are not as mature as implementations of HTTP, Autobahn Python [18] and Autobahn JS are used as open-source implementations of the WebSocket protocols in this thesis. Many traditional server engines are based on the requests and responses mode. In this project, a server engine called Twisted is used as prototype for Output Loader and Output Checker. It is written in Python and is an event-driven networking engine that supports WebSocket technology. The Output Loader utilizes the WAMP that enables a web application to call remote procedures. The Output Checker uses the basic WebSocket protocol to guarantee that Output Loader can obtain a real-time signal.

Finally, the User Management component is used to check the access permissions, and the information of users and their applications are stored in the Data Base.

Configuration on the Blue Gene/Q System

To be a computing resource that supports computational steering, specific components must be implemented on the Blue Gene/Q system. This section presents the Blue Gene/Q systems extracted from Figure 5-1 in Figure 5-3. All communications on the right side go to the CSWS server as shown in Figure 5-1. There are four components shown in Figure 5-3: the Backend Nodes (BNs), the General Parallel File System (GPFS), the Internal Network and the Frontend Nodes (FNs) and the Local Area Network (LAN). Since other components in the Blue Gene/Q system do not have a direct effect on this project, they are omitted from this figure.

[17] WAMP is an open standard WebSocket protocol that provides two application messaging patterns in one unified protocol.
[18] Autobahn Python is a WebSocket WAMP library for Python 2 and 3 and Autobahn JS is a WAMP client library that works with any WAMP server. http://autobahn.ws

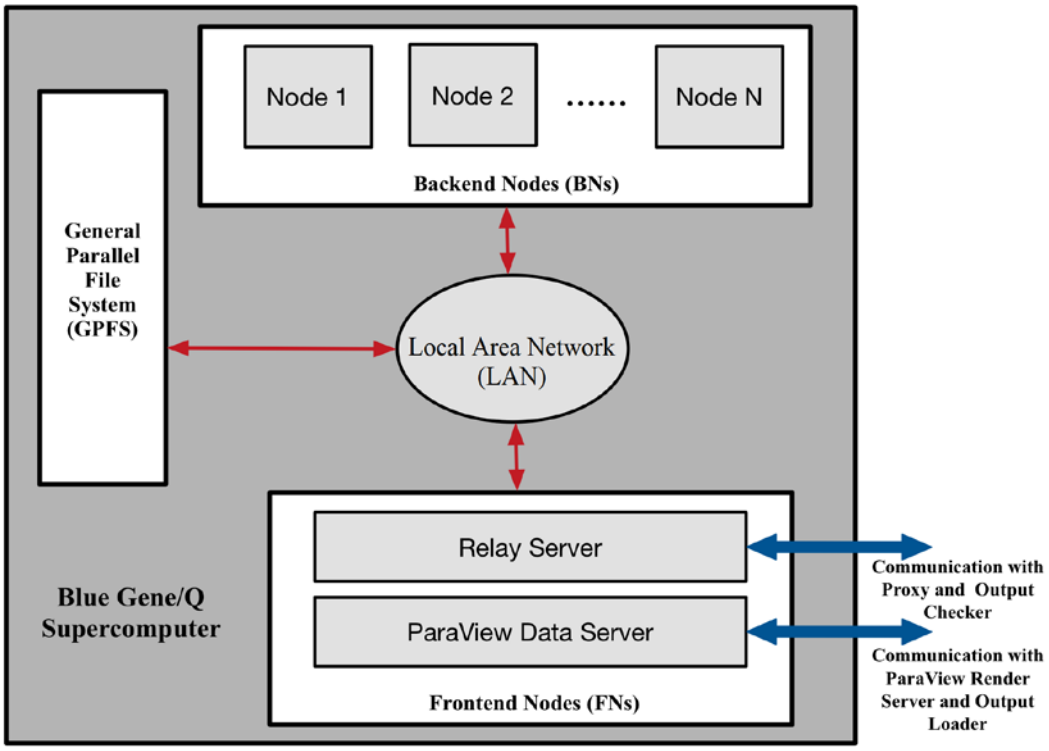


Figure 5-3 Implementation on the Blue Gene/Q: The FNs interact with the Proxy to accept and transfer steering commands. The ParaView Data Server is used to process results of simulations for the visualisation. The BNs are computing resources to run the simulations. The GPFS are storage of FNs and BNs.

The FNs act as the interface on the BGQ system for building and launching applications. Typically, users cannot access the BNs directly. Instead, they need to access a FN to upload their programs and submit jobs. The FNs and BNs connect to the functional LAN to communicate with the GPFS. The GPFS is used as the shared storage of the FNs and BNs. The computational models that have been integrated with computational steering are stored on the GPFS.

For the system security, this private LAN is typically inaccessible from outside, so communications between simulations running on BNs and remote client devices cannot be created directly. In this case, we have created a Relay Server to act as a transfer agent between the private computer network and CSWS server. The Relay Server is developed based on the RealityGrid steering library and is implemented on the FNs. Thus, it can communicate with BNs and utilizes the RealityGrid steering library to alter parameters of simulations running on BNs. Additionally, it receives real-time states of simulations running on BNs, pushes information to the CSWS server, and receives steering commands and steered parameters from the CSWS server. Additionally, commands and modified parameters retrieved from the CSWS server are passed to the running simulations through the Relay Server. The communications between the Relay Server and simulations can be

based on either files or sockets, but currently we only leverage sockets to avoid pressure on the file system.

The reason for separating data and render servers in this project is that the specific Blue Gene/Q system has no GPUs to support rendering tasks. As the CSWS server runs on a separate machine that has GPUs, we implement the render server to the CSWS server to speed up the image rendering. Another reason is that the data server has the ability to transfer data to the render server in parallel. This means separating the render server and the data server can save a lot of development time in realising parallel data transport. It may be possible to leverage a shared file system, such as the GPFS, between the Blue Gene/Q and CSWS server as a more dedicated data transport method in the future.

5.2.2 Discussion of the CSWS Framework

Deploying the conventional computational steering on HPC systems as a web service has many uncertainties. Performance and feasibility of the CSWS can be limited by the support for the visualization, the structure of the HPC systems, and the overall network performance. Therefore, components of this framework are loosely coupled into different servers. The performance of this framework can be affected by the locations of these servers, and the implementations can be flexibly based on the requirements of users and the structures of hardware systems. To demonstrate that this framework can cope with the uncertain requirements, this section presents our experience as two examples in which the loosely coupled design helps to complete the collaboration project.

Generally, the implementation should limit data movement, especially for simulations that generate large amounts of visualisation data. However, if under certain circumstances the data movement is inevitable, it can be accomplished either via network transfer or shared storage solutions. The reason for the data movement in our collaboration project was the lack of support for GPUs. For example, users may require a volume visualization in the web application. The volume visualization requires a volume rendering that is computationally expensive and typically requires GPU acceleration to achieve interactive performance required by computational steering. However, the visualisation data is stored on the Blue Gene/Q system, which does not have GPU support[19]. To cope with this issue, the ParaView

[19] ParaView also provides the Catalyst that allows CPUs to render images. Since the speed of rendering using Catalyst on Blue Gene/Q is slow, this project provides another method to introduce GPUs. To the author's knowledge, some implementations of Blue Gene/Q have began to have high end GPUs installed while the thesis was writing.

server can be separated as the ParaView Data Server and ParaView Render Server. The ParaView Data server processes visualisation data into a specific format and transfers the data to the ParaView Render Server which can be installed on machines that have specific support from GPUs.

In another case, we assume that the FNs do not have the computing power to support the ParaView Data Server at the beginning of this project. This assumption is based on the fact that FNs are only suitable for light-weight tasks, such as launching applications, because of the light computing resource assigned to it. Hence, in the original plan, the ParaView Data and Render Servers are both installed on the CSWS server to avoid introducing compute-intensive tasks on FNs. The data transfer will be implemented between the Output Checker and Output Loader. However, because of the tight development time during the collaboration, the author did not have enough time to implement the original plan. Hence, benefit from the loosely coupled data and render servers, the current implementation is a comprise solution based on the loosely coupled data and render server.

Consequently, scenarios described above illustrate that the loosely coupled structure of the framework enables components to be flexibly implemented on the BG/Q system and CSWS server. The loosely coupled structure also provides the ability for the CSWS framework to deal with uncertain specifications and structures of general supercomputer structures. Thus, configurability is shown as an important feature in the framework of the CSWS. Users can deploy this framework by re-organising components based on their requirements.

Finally, this collaboration project succeeded in enabling users to steer a DL_MESO[14] simulation based on the CSWS framework. We demonstrate the steering interface as seen by the users using a web browser to steer the simulation and obtain the visualization feedback in the Appendix B.

5.3 Implementing Dynamic Data-Driven Computational Steering on High-Performance Computing Resources

To guarantee the Dynamic Data-Driven Computational Steering (DDDCS) can finish tasks in time, it is important to dynamically assign computing resources to the DDDCS to meet the requirement of computing power. The author considers the calibration process to be the predominant time-consuming component of the DDDCS. Hence, this section primarily

introduces implementing the calibration component on dynamic computing resources. The required amount of computing resources is analysed by the Time Manager introduced in Section 4.3. Hence, this section introduces how to link the calibration component to the Time Manager and the Computing Resource Manager in terms of working on the specific amount of computing resources required by the Time Manager and assigned by the Computing Resource Manager.

5.3.1 The Workflow of Parallel DDDCS

As discussed in the Section 3.2, the Dynamic Data-Driven Computational Steering (DDDCS) is represented by the calibration process. Additionally, the workflow of the calibration process, which consists of the Genetic Algorithm (GA) and the simulations, has been depicted in Section 4.2.2. However, the workflow described in Section 4.2.2 focuses on steering interactions between GA and simulations, conversely, this section outstands functions which are required to be implemented DDDCS on HPC resources. Moreover, as DDDCS uses GA as the steerer, this section needs to analyse the specific requirement raised by the GA, in order to benefit from the template framework developed in the collaboration project with IBM.

Consequently, three features of DDDCS are discussed as follows. Firstly, the GA Steerer can trigger massive parallel steering on multiple computing resources. The parallel DDDCS is an equivalent to a scenario in which a human steerer uses two or three copies of one simulation simultaneously with different groups of steered parameters. The difference is that the automatic Steerer, GA, has a higher scalability in terms of parallelism which means it can steer a large number of simulations concurrently. Secondly, since DDDCS requires dynamic computing resources, the communication between the Application Side Steering Interface and the Steerer can only be built dynamically. Moreover, since the DDDCS is conducted in parallel, the Steerer needs to communicate with multiple Application Side Steering Interfaces and coordinate steering tasks. Thus, parallel workflow of DDDCS is shown in Figure 5-4.

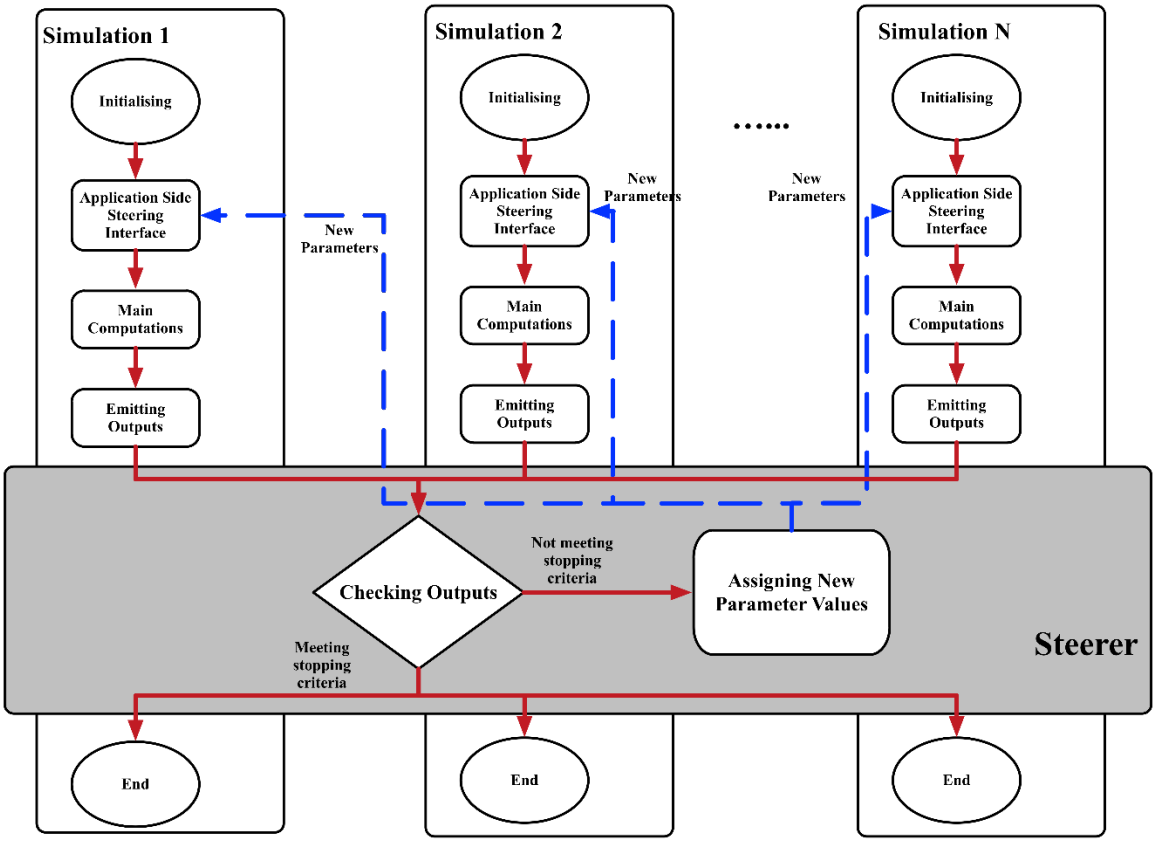


Figure 5-4 Parallel Workflow of the DDDCS:Dashed arrows are used to differentiate between overlapped arrows. It emphasizes the Checking Outputs and Assigning New Parameter Values components that are designed to handle the parallel programming on the HPC resources.

Figure 5-4 depicts the design of the parallel workflow of DDDCS based on the workflow introduced in Figure 4-6 of Section 4.2.2. The description of each workflow can be found in Section 4.2.2 as well. This section only discusses the functions shown in the Steerer. At first, the Assigning New Parameter Values function acts as the steering interface on the steerer side. It manages a task pool which consists of parameter groups generated by the GA. As the calibration is running in parallel, the Assigning New Parameter Values function also needs to assign each group of parameters to the corresponding simulation. On computing resources, all processes are initialised simultaneously and wait for the steered parameters. Thus, based on the parallel scale N, N sets of parameters are explored concurrently. The Main Computations components are identical in all parallel processes, and their outputs are collected by the Checking Outputs component. If any of them meet the stopping criteria, the workflow reaches the End status. Otherwise, N sets of new values from the task pool are generated and used to trigger another round of steering on remote simulations.

5.3.2 The Framework of Dynamic Data-Driven Computational Steering

The parallel workflow discussed in the last section introduces functions required by interactions between the steerer and simulations. However, since the DDDCS also needs time management and computing resource management, the parallel steering workflow needs to be further extended to include functions that have been introduced in Section 4.3. Additionally, we propose to extend the CSWS to support the DDDCS as well.

However, the framework developed for the HDCS and CSWS only provides abstract description of components such as Computing Resource Broker and Computing Resource Manager. As we collaborate with Zeqian Meng who focuses on the study of the communication between brokers, resources providers and resource requesters, the developed framework is only tested based on a set of fundamental interfaces provided by Meng's work [100]. The test can show that the designed framework can communicate with one of modern computing resource broker.

By doing so, new functions that are created to tackle challenges of DDDAS, such as time management and computing resource management, are included in the CSWS server.

By adding new DDDCS functions to the CSWS server, it is necessary to adapt the workflow among Steerer, CSWS server and HPC infrastructures that are developed for HDCS to the DDDCS. Additionally, since this section focuses on integrating DDDCS functions with CSWS, the details of HDCS on the CSWS will not be discussed.

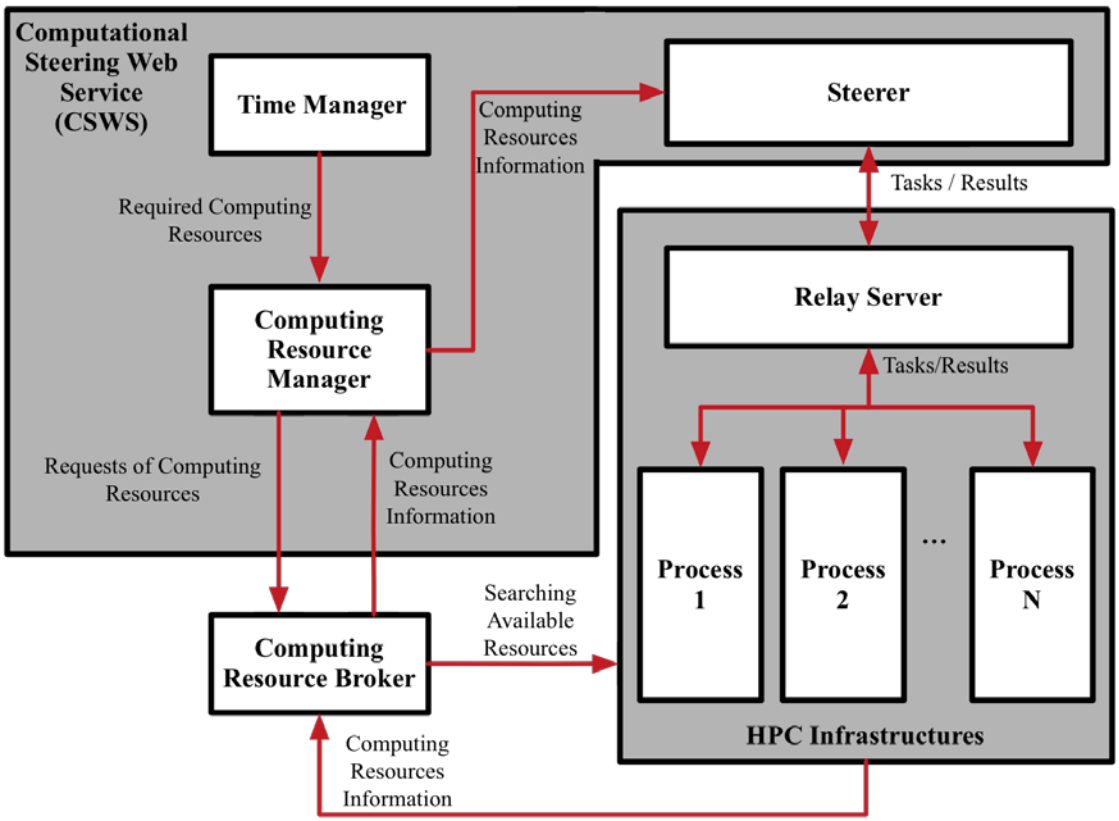


Figure 5-5 The Workflow of the Dynamic Data-Driven Computational Steering: This figure indicates Steerer, Time Manager and Computing Resource Manager are integrated with CSWS developed for the conventional computational steering. The Steerer, the GA, steers parameters of simulations through the Relay Server.

Figure 5-5 shows a general CSWS workflow of the DDDCS which is inspired by the CSWS framework developed for the HDCS. It is flexibly designed to be implemented on HPC infrastructures. The HPC Infrastructures shown on the right represents a general HPC structure, and Blue Gene/Q and Amazon Cloud will be used as two specific HPC structures to demonstrate the applicability of the general framework. The component shown on the left is the extended CSWS, and it includes Time Manger and Computing Resource Manger that are used to support real-time steering functions required by the DDDCS. Additionally, the steerer of the DDDCS is also included in the CSWS since the CSWS can provid secure communications with the HPC infrastructures. The Time Manager estimates the running time of the Steerer based on the incoming sensor reading and further estimates the least required computing resources to ensure the Steerer to finish in time and sends the amount of required computing resource to the Computing Resource Manager. The Computing Resource Manager is the interface of the CSWS to the Computing Resource Broker. It wraps the required computing resources into the request to the Computing Resource Broker, which is responsible for searching available computing resources from resource providers, and initialising a computing environment on these resources to make them ready for users. The

Computing Resource Broker provided by Meng uses REpresentational State Transfer (RESTful) interfaces for the Computing Resource Manager to send the wrapped request. The requested amount of resources is in the form of core numbers, and the feedback from the Computing Resource Broker includes IP addresses and port numbers of Relay Server on the HPC Infrastructure.

In practice, the specification of CPUs and the network configuration are also important for computing performance. However, this work is limited by functions that can be provided by the simple Computing Resource Broker. Hence, to demonstrate the ability of using different amount of computing resources, it is reasonable to use the number of a specific type of CPUs to measure the amount of computing resources. In addition, as testing the performance of different computing infrastructures is not in the scope of our work, we do not have a comprehensive study on comparison between different types of computing resources. As studied in Section 4.3, this thesis only provides a general method to demonstrate existing time management method can be used to support the hybrid computational steering by learning the relation between the required amount of computing resources, the workload of calibration process and the deadline. Studying the exact amount of computing resources required by a steering job is left as a future work in the development of a specific project.

The Relay Server is also a concept used in the template framework developed with IBM. By referring to the parallel steering workflow, it can be considered as one part of the Application Side Steering Interface. Its basic function is to relay steered parameters to simulations by communicating with the steering interface on the simulations. This steering interface is considered as the other part of the Application Side Steering Interface. Additionally, in this work, one process is assumed to solely occupy one basic computing unit in the HPC infrastructures. Hence, we do not consider the situation of two threads running on the same CPU based on a mechanism such as the round robin. To reduce the computing burden of the Relay Server, the Computing Resource Manager indicates the number of cores applied from the resource provider to the Steerer. Based on this information, the Steerer groups steered parameter values into steering tasks, and the number of tasks in a group equals to applied core numbers. Thus, the Relay Server only needs to relay each steering task to the corresponding task process.

The general framework acts as a template of implementing DDDCS on HPC infrastructures. However, the specific implementation can be different. In order to support our general

DDDCS framework, we discuss the derived framework implemented on two popular HPC infrastructures. In this thesis, frameworks of the IBM Blue Gene/Q supercomputer and Amazon Elastic Computing Cloud (Amazon EC2) are used as representatives of computing resources that can be offered by the Computing Resource Broker.

Implementing Dynamic Data-Driven Computational Steering on Blue Gene/Q supercomputer

The Blue Gene/Q supercomputer is a representative of local clusters and supercomputers. Supercomputers and local clusters have geographically closed processors. Since in the Blue Gene/Q system, processors do not share memory, this implementation only aims at distributed-memory supercomputers. Thus, each simulation process in this work maintains its own memory and the steering process needs to alter parameters in memories for each steering task assigned to the processes. By taking this general feature of supercomputers into consideration, we transform the general framework designed in Figure 5-5 to the framework designed for the Blue Gene/Q supercomputer.

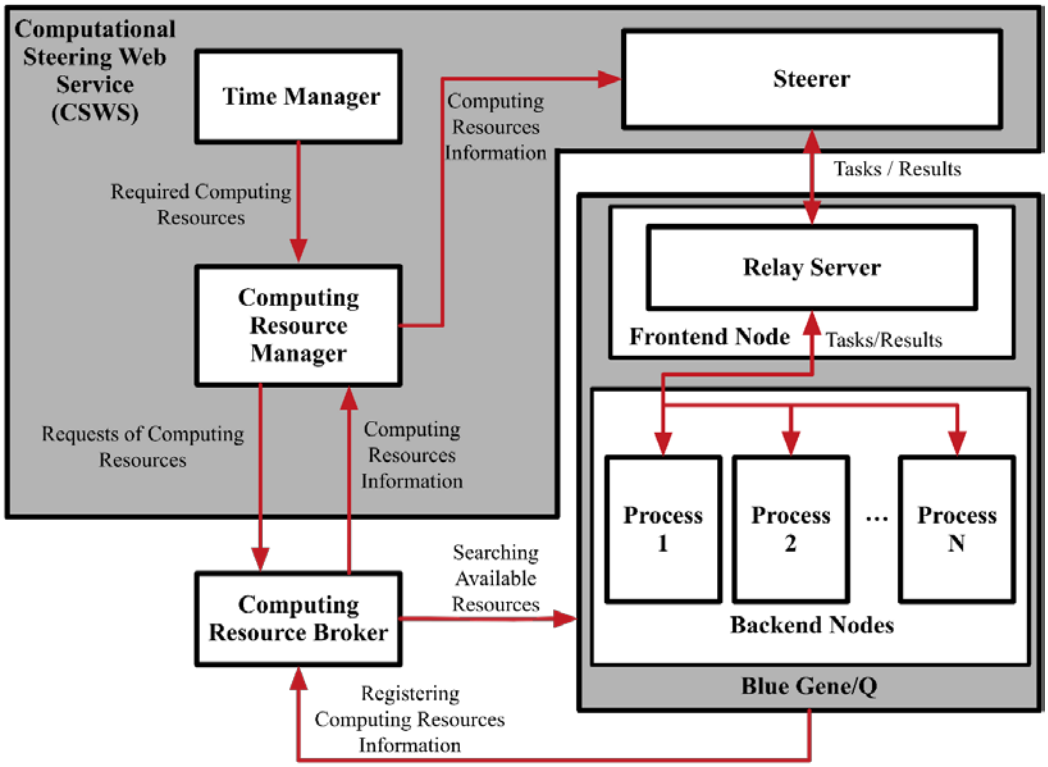


Figure 5-6 The DDDCS framework implemented on the Blue Gene/Q supercomputer: Since the general framework is developed based on the framework of developed for conventional computational steering, this figure is similar to Figure 5-5. The difference is the Relay Server only communicates with one process on the Backend Nodes to exchange the steering information.

As shown in Figure 5-6, communications among the Blue Gene/Q supercomputer, Computing Resource Broker, Computing Resource Manager, Time Manager and the Steerer remain the same as that of the general framework. Additionally, based on the design of HDCS framework, the interface component, Relay Server, is implemented on a Frontend Node (FN) of the Blue Gene/Q. As part of the negotiation protocol, the information of computing resources, which is used by the Steerer to send tasks, are the Internet Protocol (IP) address and TCP port of the Relay Server.

In this framework, the Relay Server has three functions: 1) the first one is to separate the outside network from the computing node; 2) it transfers all tasks received from the Steerer to tasks running on the BNs, and 3) it is also responsible for initialising steering environments for accepted steering tasks. The first two tasks have been discussed in previous sections, and the third function means that the Relay Server, after confirming to the Computing Resource Broker to provide computing resources, must start several steerable simulations on its computing resources. Additionally, the TCP ports that have been provided as resource information for the broker must be reserved by the Relay Server for communications from the Steerer. The pre-started simulation waits for the Steerer by stalling at the Steering Interface as shown in Figure 4-6. Afterwards, the Computing Resource Manager can obtain the information of the ready-to-use computing resources from the Computing Resource Broker and pass the information to the Steerer, which starts the DDDCS by transferring steering tasks to the Relay Server through the reserved TCP ports.

As the Blue Gene/Q is a distributed-memory system, each sub-process of the parallel simulation has one of its own copies of parameters. Hence, the steering process must be conducted on each of the sub-process to alter the steered parameters. However, the RealityGrid Computing Steering toolkits used in this thesis do not provide support to directly alter parameters on all processes. Hence, to enable the parallel steering, we either need to create steering interfaces for each processes or need to realise the parameter alternation by modifying the original parallel simulation package to pass the steered parameters from the master process to slave processes.

The method used for the Blue Gene/Q supercomputers is modifying the original parallel simulation package. The reason is that the communications among BNs are specifically optimised in the Blue Gene/Q system. Instead of customise the communication between the Steerer and all processes, interfaces such as Message Passing Interface (MPI) can be directly

used to take advantage of the optimised communication among BNs. Since in the distributed-memory system, MPI is a predominant communication protocol to realise the communication, we assume all parallel simulations discussed in this thesis running on the Blue Gene/Q system utilise the MPI.

To achieve this method, all information of steered parameters received from the Relay Server is transferred to one process, the master. Before all processes enter the Main Computations component as shown in Figure 5-6, the master process, by using Process 1 as an example, sends steered parameters to slave processes using MPI, and at the end of the Main Computations, results of each processes are collected by the Process 1 which sends them back through the computational steering interface.

Implementing Dynamic Data-Driven Computational Steering on Amazon Cloud

Supercomputers always have an absence of computing resources, especially when the resources are shared among several institutions and organisations, instead, on-demand cloud computing resources can be applied in time without consideration on lacking resources. Additionally, Amazon Cloud provides different instance types. An instance is a virtual machine on the Amazon Cloud, which can be customised with different configurations on the CPU, memory, storage, and network resources. It is a type of virtual computing resource since the physical computing resources are dynamically assigned to instances on demand. For example, in the Amazon Cloud, the smallest instance has one CPU, 0.613 gigabyte of memory and a low network performance. On the other hand, the largest instance, which is optimised for compute-intensive tasks, has 32 CPUs, 60.5 of gigabyte memory and a 10-gigabit network bandwidth. Therefore, after the Computing Resource Broker accepts the application from the Computing Resource Manager, it starts instances that meet the requirement of the Time Manager.

Based on the types of instances of Amazon Cloud accessed in October 2016, three solutions can be used to deploy the parallel steering tasks: 1) utilising OpenMP to parallelise steering tasks on compute-intensive instances, such as the “c3.8xlarge” instance that has 32 CPU cores, 2) utilising the one-core instances, such as the “t2.nano”, to assemble a distributed computing system, and 3) combining the first and second solutions by utilising the multi-core instances to construct a distributed computing system. In terms of choosing the optimal solution for a specific project, the decision can depend on the particular requirements of computing power, project budget, and specific model of the simulation. A “fat” instance that

has up to 128 CPUs has advantages of fast communication among parallelised tasks and a short application time. However, it has a high price compared with smaller instances. For example, the price of the "c3.8xlarge" instance is $1.68 per hour, while, the price of the "t2.nano" instance costs $0.0065 per hour. Although the "c3.8xlarge" has 32 times the number of CPU cores as the "t2.nano", it costs 258 times more than the small instance. Since the hybrid solution includes the structure of the first and second solutions, this section solely introduces the framework designed for the third solution as an example.

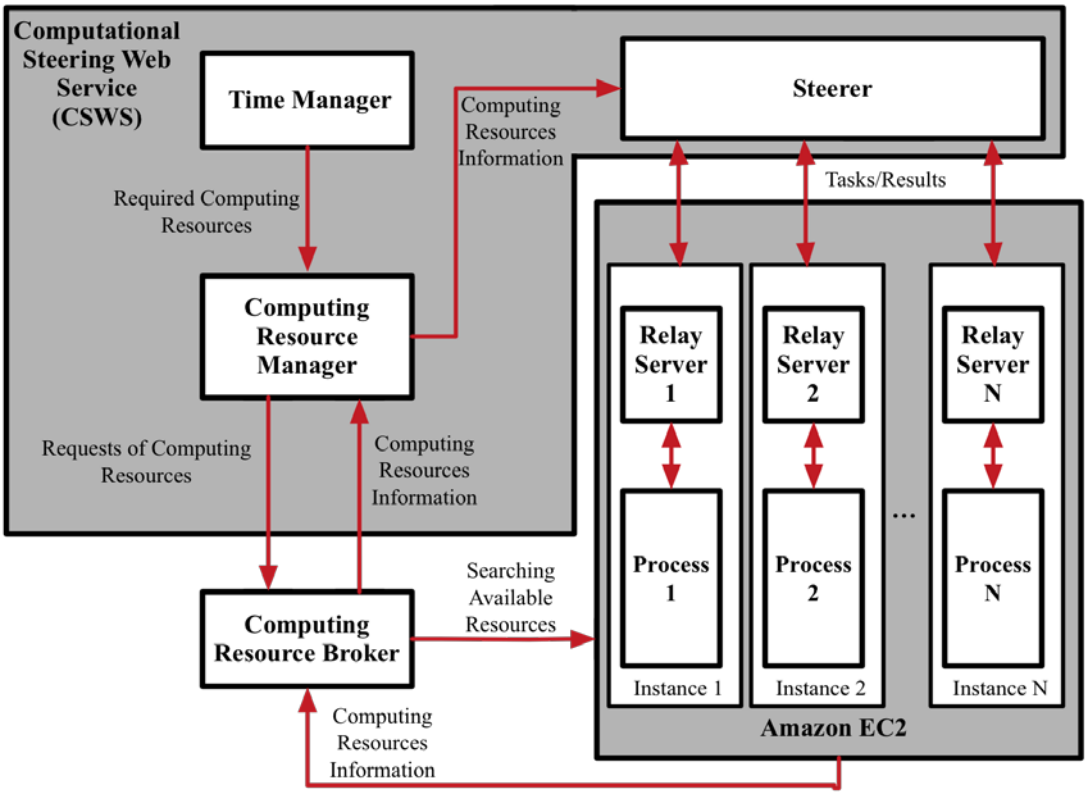


Figure 5-7 The DDDCS framework implemented on the Amazon EC2: This figure emphasizes the Relay Servers are installed on all instances to handle their communications with the Steerer.

Figure 5-7 depicts the framework designed for the cloud computing resources. Since the basic workflow is the same as that of frameworks shown in Figure 5-5 and Figure 5-6, we only emphasize its difference.

In the development of the DDDCS framework for the cloud structure, each instance has its *per se* Relay Server. After Computing Resource Broker manages to find requested computing resources, it returns IP addresses and port numbers of Relay Servers on all instances to the Computing Resource Manager. This information is transferred to the Steerer which dynamically divides tasks into slices and assigns these to corresponding Relay Servers.

Creating multiple Relay Servers for each instance is necessary. Different from the centralised computing resources in the Blue Gene/Q system, the processor provided by the cloud providers can be geographically distributed. In this example, the computing resources acquired from the Amazon EC2 cloud are in the form of instances. Compared with supercomputers that have a high-speed interconnect to connect the processors, the communication among Amazon instances may need to go crosses countries and continents. In the implementation on the IBM Blue Gene/Q, the steered parameters are assigned to slave processes by a master process using MPI. However, the conventional MPI did not support the cross-sites communication required in the Amazon Cloud structure. Due to the advance of grid computing and meta-computing, variations derived from the original MPI, such as OpenMPI, PACX-MPI and MPIg, have started to handle communication among distributed processes. However, as we do not want to bind our project with specific MPI standards, we move the job of separating and assigning steering tasks from the master process to the Steerer. To achieve a concurrent task assignment effect, the Steerer is run in parallel in this implementation. As a result, each process of the Steerer corresponds to an instance requested from the Amazon Cloud. The MPI is still used, but instead of being used among Processes, it is applied to the Steerer to distribute steering tasks. The Relay Server simply relays steering tasks to computing processes and sending simulation results back to the Steerer.

A challenge in this implementation is to deploy the steering-enabled models on an instance which is requested dynamically and set up the software environment of such instance to make it ready for the DDDCS tasks. As for the framework designed for the Blue Gene/Q system, this thesis assumes that the toolkit, simulation models and other required libraries have been installed on the instance, and this setup is a requirement of being a provider of the DDDCS service. To realize this assumption, the instance image mechanism provided by the Amazon EC2 is utilized to store the configuration of a pre-configured instance. By requesting instances with an image identity, the required software environment and steerable models are ready to use.

5.4 The Framework of the Hybrid Computational Steering

After describing the frameworks of Human-Driven Computational Steering (HDCS) and Dynamic Data-Driven Computational Steering (DDDCS), this section integrates them to achieve the framework of the hybrid computational steering. The result of the integration incorporates workflows introduced in both frameworks.

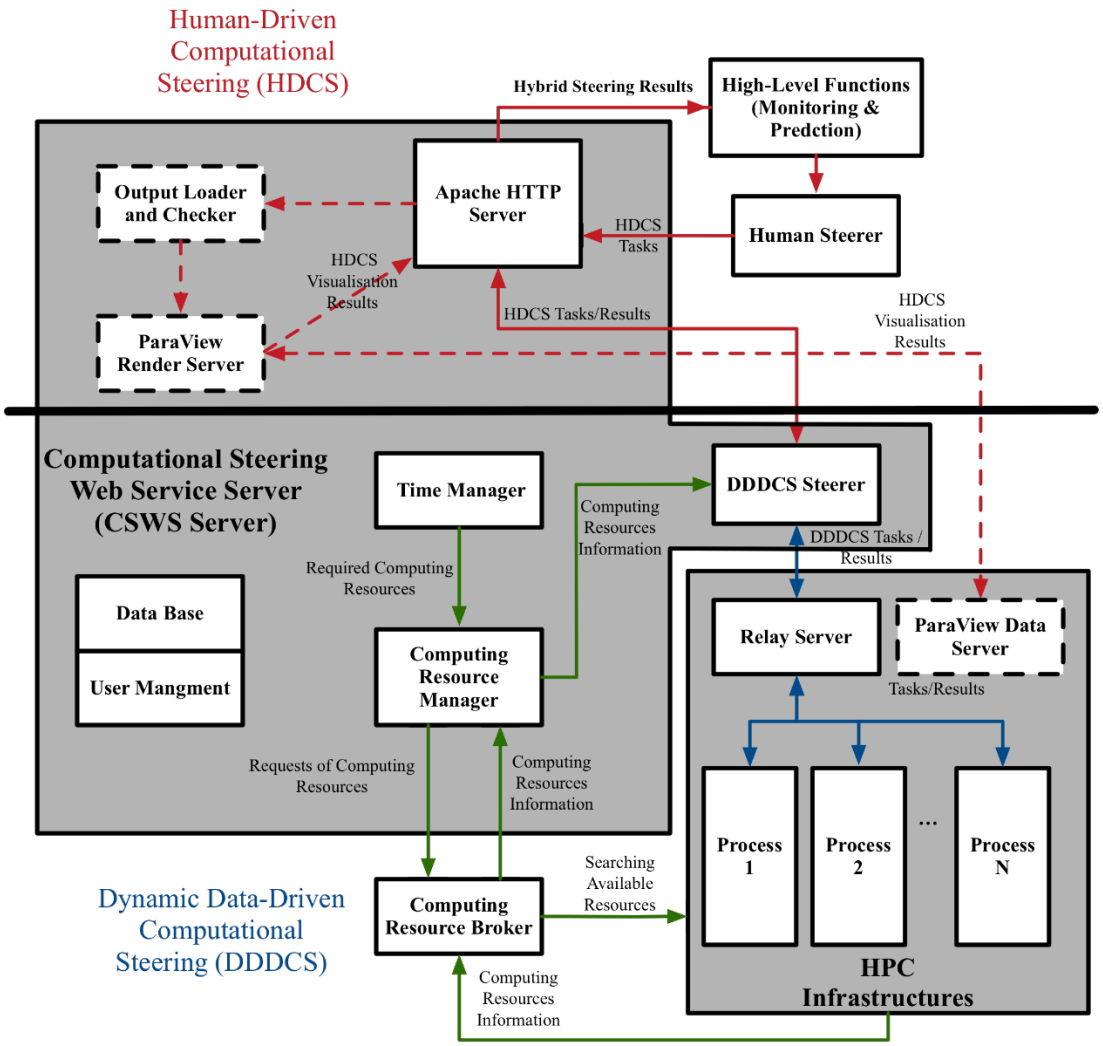


Figure 5-8 Framework of the Hybrid Computational Steering: The red, blue and green colours are used to indicate the HDCS, DDDCS-SL and DDDCS-HL layers introduced in architecture design in Figure 4-1. The black bar in the middle separates the frameworks of HDCS and DDDCS. The components with dashed outlines indicate the functions used to support the conventional computational steering. Such functions can be removed in the development of the hybrid computational steering.

Figure 5-8 illustrates the framework of the hybrid computational steering. Although this framework has a complex workflow, this figure separates the HDCS on the upper side from the DDDCS on the lower side. Details of each steering framework are not repeated in this section. As introduced in Chapter 4, the HDCS is considered as a higher steering layer compared with the DDDCS, this section focuses on introducing the interaction between the HDCS and DDDCS.

In the hybrid computational steering, human users are enabled to alter parameters of the calibration algorithm at its execution time. The web steering application developed in the framework of the HDCS is kept to provide users an interface to steer parameters of the calibration algorithm. This steering application is served by the Apache HTTP Server and

acts as a Relay Server between the DDDCS Steerer and users since it needs to pass the steering commands assigned by users to the DDDCS Steerer. As the function of relaying steering command is the same as its function used in the project collaborated with IBM, the implementation of Apache HTTP Server is kept the same, and the DDDCS Steerer is integrated with RealityGrid Computational Steering to interact with the Apache HTTP Server. Moreover, as the HDCS provide high-level functions for users to analyse the behaviour of the physical world, the visualisation data processing and rendering are not critical in the context of DDDAS. Such components that are designed to facilitate the conventional computational steering are outlined in dashed lines. It means that these components can be removed in the framework of the hybrid computational steering.

5.5 Chapter Summary

This chapter presents the frameworks that are developed to implement architectures of the hybrid computational steering on HPC infrastructures. As the infrastructure of the hybrid computational steering can be divided into Human-Driven Computational Steering (HDCS) and Dynamic Data-Driven Computational Steering (DDDDCS), its framework is also divided into HDCS and DDDCS frameworks.

Since the hybrid computational steering is inspired by conventional computational steering which are largely driven by human users, the development of HDCS framework is based on a collaboration with IBM which intends to improve the usability of supercomputers using conventional computational steering. Based on the survey conducted in Section 2.1, we analysed that the diverse and rapidly changing communication protocols and standards of the grid may prevent model developers from contributing to computational steering. Thus, the developed framework sees computational steering as the Computational Steering Web Service (CSWS) that can isolate the steering middleware from the HPC resource service. Additionally, the developed framework manages to support the web service to provide both conventional computational steering and HPC infrastructures as services for users. We managed to steer the DL_MESO simulation through the web application and managed to allow the CSWS server to handle communications between the steerer and simulations running on HPC resources. At the end of the collaboration, IBM takes the steering service as a solution to provide an app that enable consumer devices (e.g. tablets) to access an IBM Blue Gene/Q system. Since there is no significant difference between the HDCS and conventional computational steering in terms of implementing on HPCs, the framework

developed in the collaboration project is considered as the framework developed for the HDCS. Moreover, it also provides us experience and a template in the development of the DDDCS framework.

Based on the framework developed for the HDCS, we further developed frameworks for the DDDCS that can be implemented on both supercomputers and clouds. Inspired by the framework of HDCS, the Steerer of the DDDCS is incorporated by the CSWS to enable secure communications between the steerer and Relay Server. The Relay Server is also a product of the collaboration project. It transfers the steered parameters from the DDDCS Steerer to the simulations running on the backend computing resources. To cope with the parallel simulations running on the HPCs, dividing steered parameter groups for different processes are introduced as an additional function of the DDDCS Steerer. Additionally, to communicate with the Steerer, the Time Management and Computing Resource Manager are also included into the CSWS. Based on features of IBM Blue Gene/Q and Amazon Cloud computing architecture, the general DDDCS framework is modified to demonstrate that it can be applied to specific computing resources.

Finally, the frameworks developed for HDCS and DDDCS are integrated as the framework of the hybrid computational steering. Our aim in developing above frameworks is to implement architectures of the hybrid computational steering in practice. Hence, it is crucial to develop components in these frameworks as programs. Since the main contribution of this thesis is not program development and also because of the limitation of the content size, we do not introduce the progress of programming. All programs designed in the collaborative project must not be disclosed in the public according to the requirements of IBM. Programs for Steerer, Relay Server, Computing Resource Manager in both DDDCS and the hybrid computational steering are stored in Bitbucket. The web link to access these programs are attached in Appendix A. Since we collaborated with another PhD research which provided the program of the Computing Resource Broker, we do not include it as the work of this thesis. For the Time Manager, the time management principles are evaluated manually; hence, we do not realize the Time Manger as a program.

Chapter 6 Evaluation

The concept of the hybrid computational steering has been introduced in Chapter 3. Chapter 4 and Chapter 5 have presented its design of the architecture and framework levels. This chapter evaluates the combination of functions, architectures and frameworks introduced above and demonstrates that the hybrid computational steering is feasible to be implemented in practice and make contributions to solving problems of a real large-scale physical system.

This chapter uses an application of water distribution systems as a use case because of the collaboration with WRc and Northumbrian Water company. The background and implementation of the use case are introduced in Section 6.1 and 6.2. Section 6.3 evaluates the functions and performance of Human-Driven Computational Steering (HDCS), which constitutes the first half of the evaluation of the hybrid computational steering. For the second half, namely the Dynamic Data-Driven Computational Steering (DDDCS), Section 6.4 compares the performance of the calibration process developed in Section 4.2 with the calibration of existing projects in the field of the water distribution system. Additionally, to achieve the time requirement in a DDDAS, the Time Manager and the ability of using dynamic computing resources, which are introduced in Section 4.3, are evaluated in Section 6.3, 6.5 and 6.6. A local cluster, a local multi-core computer and a cloud computing system are used as two platforms to conduct the evaluation.

6.1 Use Case: The Simulation of Water Distribution System

As we propose to integrate computational steering with applications of DDDAS, it is essential to select a physical system as a use case that needs to utilise a numerical model to implement a real-time and compute-intensive simulation. This section first discusses the reasons for selecting a Water Distribution System (WDS). In addition, we introduce the software used to model the WDS. Finally, this section introduces a genetic algorithm that is used as the specific calibration algorithm in this evaluation.

6.1.1 Computational Challenges of Modelling a Water Distribution System

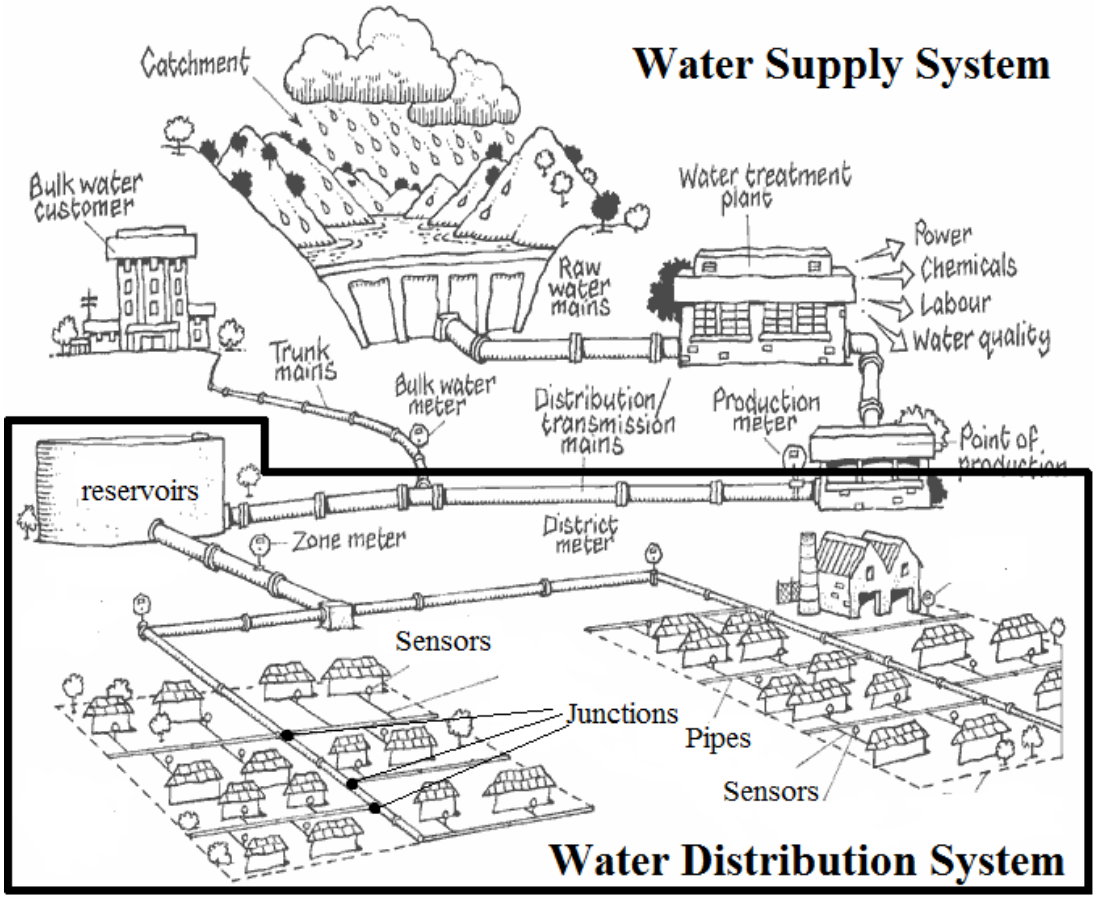


Figure 6-1 The Water Supply System [109]: The water supply system includes the water distribution system that is framed into the lower-half box. A numerical model of the water distribution system can be implemented to simulate states of junctions, pipes and reservoirs. These states are important for daily works of water engineers. Sensors are used to collect information to rectify the numerical model, and they are abstracted as being implemented on junctions. This figure gives readers a general idea of a water distribution system.

A Water Distribution System (WDS) is a sub-system of the Water Supply System as shown in Figure 6-1. It is mainly responsible for clean water storage and transmission. In a physical WDS system, the main components of a WDS include pipes, valves, pumps, junction, tanks and reservoirs. In modelling a WDS, junctions, reservoirs and tanks are always abstracted as nodes and pipes, pumps and valves are categorised as links [82].

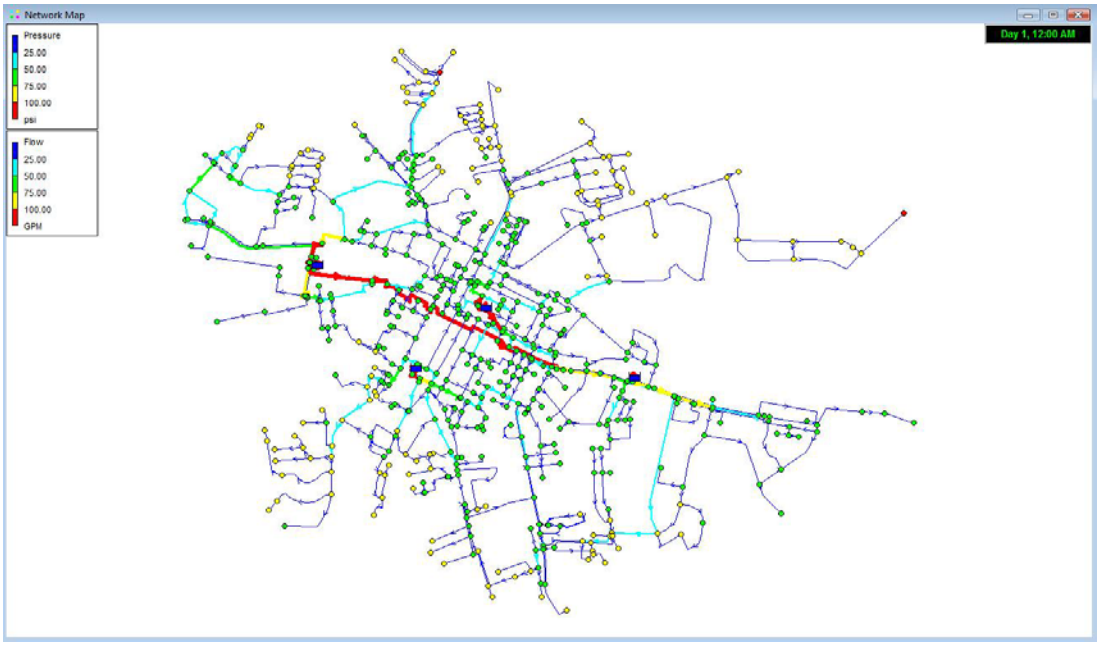


Figure 6-2 The Simulation of the WDS: This is a graphical result of modelling a water network. Two scales are shown on the upper left corner to map colours to values of pressures and flow speeds. Rounds denote junctions and lines denote pipes. By taking the size of a real water network into consideration, this figure indicates the complexity of simulating a water network when

Figure 6-2 shows a graphical result of a simulation whose links and nodes are shown as lines and circles. Its colours correspond to particular pressure and flow values. However, Figure 6-2 only shows a portion of the entire water distribution network that can include thousands of junctions and pipes. Modelling a WDS is challenging for our evaluation methods since a WDS is a complex physical large-scale network system. In the United States, the distance of the distribution WDS network has reached 400,000 miles in 2005 [132], which is more than 16 times longer than the perimeter of the earth. The network operated by our project partner Northumbrian Water company operates 326 water service reservoirs, 25678.3 km mains, 29923.1 km sewers and serves more than 1104 megalitres water to their customers every day [3]. Additionally, these distributed networks are always largely interconnected and buried under the ground. Thus, there is a high degree of uncertainty about the state of the WDS, and both the simulation and calibration of the WDS turn to be a compute-intensive task.

Furthermore, as identified by Shannon and Jeanne in 2005 [70], system wide sensor networks coverage is very rarely available and exists only in selected testbeds. Hence, conventional WDS simulations typically run in offline mode. However, as the development of sensor networks, using sensor data to simulate and calibrate WDS is becoming a goal for water companies, and it raises interests in running real-time simulations and calibrations so

that anomalies such as pipe leakage and bursts can be identified, measured and fixed in time. Hence, a real-time and compute-intensive problem has arisen in WDS, and it fits the problem domain defined in Chapter 1.

6.1.2 Modelling Water Network

We chose a widely available open source software package, EPANET, as the simulation component for modelling a WDS. EPANET was developed by the Environmental Protection Agency for the water modelling community, and it is widely utilised for public and commercial water network software development. This work uses its latest version, EPANET 2, which was released in 2008.

EPANET requires an input file that describes the geographical and engineering details of the network to be simulated. SZ01.inp is an input file that describes the network information provided by the Northumbrian Water company. The suffix, INP, is the abbreviation of the word *input* and indicates the format of the file. SZ01 is a code that denotes the particular area in the water network. Figure 6-3 presents the graphical result of simulating the SZ01 area.

Figure 6-3 Overview of SZ01: Dots are junctions and lines are pipes. It presents the structure and element positions of the water network.

This project divides the information stored in the SZ01.inp into static and dynamic information. The static information includes locations of elements, length and diameter of pipes, types of water demand, etc. The static information describes fundamental parameters and structure of the network as shown in Figure 6-3. Hence, the next step is to calculate the states of elements in the water network and get the result as shown in Figure 6-2. The dynamics of the EPANET model are driven by the water demand pattern contained in the INP file. It means the model in the EPANET converts the water consumption states such as pressure and flow rate. In a real-time simulation, the water demand is obtained from the calibration component dynamically. However, in order to introduce the water demand, we use static water demand supplied by Northumbrian water in the INP file as an example.

In an INP file, the water demand is represented by an array of values that are attached to components in the model. The elements of this array are called Demand Multiplication Factors (DMFs) and are defined in the [PATTERN] section. EPANET calculates the demand of a junction by multiplying the DMF with the base demand (recorded in [DEMANDS] section) of a junction. Therefore, when the DMF is greater than 1, the water demand of the corresponding junction increases, and conversely, its demand decreases. Additionally, in SZ01, DMFs are categorised into 510 types according to the types and locations of consumers. For a specific type, the number of multiplying factors indicates the duration covered by the INP file.

```
DO_BW005_S5PDT    0.143540    0.110130    0.081760    0.058430    0.045550    0.043110
DO_BW005_S5PDT    0.042450    0.043560    0.053550    0.072430    0.106390    0.155450
DO_BW005_S5PDT    0.303860    0.551610    0.783580    0.999740    1.060010    0.964390
DO_BW005_S5PDT    0.895640    0.853740    0.822940    0.803220    0.734950    0.618140
DO_BW005_S5PDT    0.528640    0.466450    0.421030    0.392370    0.407470    0.466350
DO_BW005_S5PDT    0.546820    0.648870    0.701600    0.705030    0.759750    0.865770
DO_BW005_S5PDT    0.953680    1.023500    1.081760    1.128470    1.210830    1.328860
DO_BW005_S5PDT    1.302040    1.130390    0.935480    0.717320    0.520070    0.343740
```

Figure 6-4 An Example of DMFs: The DO_BW005_S5PDT is the name of a water demand multiplier factor. It indicates the fluctuation of one type of water demand. The numbers indicate the value of the DMF at a specific time step. The duration a factor can represent depends on the time information noted in the [TIMES] section in the INP file.

```
[TIMES]
Duration             24:00
Hydraulic Timestep   00:15
Quality Timestep     00:05
Pattern Timestep   00:30
```

Figure 6-5 [TIMES] Section of the INP File: Duration is the total time can be covered by the DMFs. The Hydraulic and Quality Timestep indicate time steps in two types of simulations. The Pattern Timestep is the duration one DMF can represent.

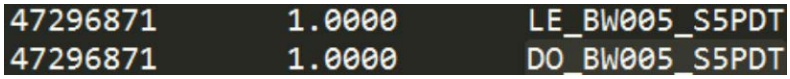


Figure 6-6 [DEMANDS] Section of the INP File: The first column is the ID of a junction. The column on the right shows the type of water demand, and their initial water demand values are listed in the middle. For one junction, its water demand equals to the sum of all types of water demands shown in this section.

For instance, Figure 6-4 shows the DMFs of a particular water demand pattern, namely DO_BW005_S5PDT. According to the information shown in Figure 6-5, the 48 factors of the DMF covers the water demand fluctuation in 24 hours and the step of the DMF is 30 minutes. Moreover, Figure 6-6 presents an example of assignment of the water demand pattern. The node 47296871 contains two types of water demand, LE_BW005_S5PDT and DO_BW005_S5PDT, and the base demand for both types are 1. Hence, for node 47296871, the water demand of DO_BW005_S5PDT at the beginning of the hydraulic model is 0.143540 which equals to the product of base demand multiplied by the first factor of the DO_BW005_S5PDT. As a result, by using the SZ01.inp as the input file, a 24 hour offline simulation of the water network can be executed with a 30 minutes updating frequency. As a result, the parameters that drive the simulation are the DMFs, and the results of the simulation are states on pipes and junctions such as the water pressures and speed.

However, in order to monitor a WDS in real-time, rather than using the static DMFs stored in an INP file, DMFs must be updated to reflect the progress of a real WDS. Such knowledge is important for understanding the implementation of the use case discussed in Section 6.2.

	Junctions	Junctions with Demand Patterns	Links	Sensors	Types of the Demand Multiplication Factors (DMFs)
Amount	5000	1853	5226	39	510

Table 6-1 Summary of SZ01: This table indicates the size of the water network area SZ01. The states of Junctions that have Demand Patterns fluctuate as time passes. Other junctions have static states. DMFs are utilised to calculate water demands on junctions. Its number indicates the amount of types of water demand. Since the number of sensors is much less than the types of demand, modelling cannot avoid errors and uncertainty.

As a result, Table 6-1 summaries elements and DMFs stored in the SZ01.inp, and it also indicates the number of sensors implemented in the SZ01 area by the Northumbrian Water company. The labels of junctions, links and DMFs are collected by using functions provided

by the EPANET. The labels of junctions with demand patterns are extracted from the [DEMANDS] section of the input file. It is used for defining multiple water demands at junction nodes. Nevertheless, only 1853 out of 5000 junctions are assigned with demand patterns. The assumption is that the water demands on other junctions are constant [123]. Thus variations of the water network states are mainly represented by the 1853 junctions that are assigned demand patterns. In order to monitor the SZ01 region, 39 sensors are deployed by the Northumbrian Company, and sensor readings are used to signify the states of 5000 junctions and 5226 links. Hence, modelling SZ01 based on such dynamic information cannot avoid errors and uncertainty.

6.1.3 Calibration

There are multiple sources of uncertainties that have significant influence on the accuracy of modelling of a WDS[67]. Preis [117] indicates that the general method is to calibrate model parameters such as pipe roughness in offline mode. However, values of DMFs change much more dynamically than pipe roughness as model parameters and thus require to be calibrated in online mode. In a modelling of a water network, model states such as water pressures and flow rates can be directly obtained from sensors. Since EPANET is driven by un-measurable DMFs to output measurable states, such as pressure, the calibration is to assimilate measurable states to rectify the un-measurable DMFs.

As discussed in Chapter 3, this thesis considers the existing calibration process as a batch utilisation of HPC that can be improved by computational steering. Therefore, model parameters can be considered as the steerable parameters. However, since we focus on the online calibration, only DMFs, that are steered based on the dynamic sensor data, are considered as our steerable parameters.

After determining the steerable parameter, we need to discuss the calibration method that can be integrated with computational steering. The review of potential calibration algorithms has been discussed in Section 2.2.2. For this evaluation, we build on a previous decision support tool that has been shown to work well for WDS modelling [82]. In the previous collaboration, a decision support tool [82] has been developed based on the GA, hence, as part of the continued collaboration, the GA is selected as the calibration algorithm.

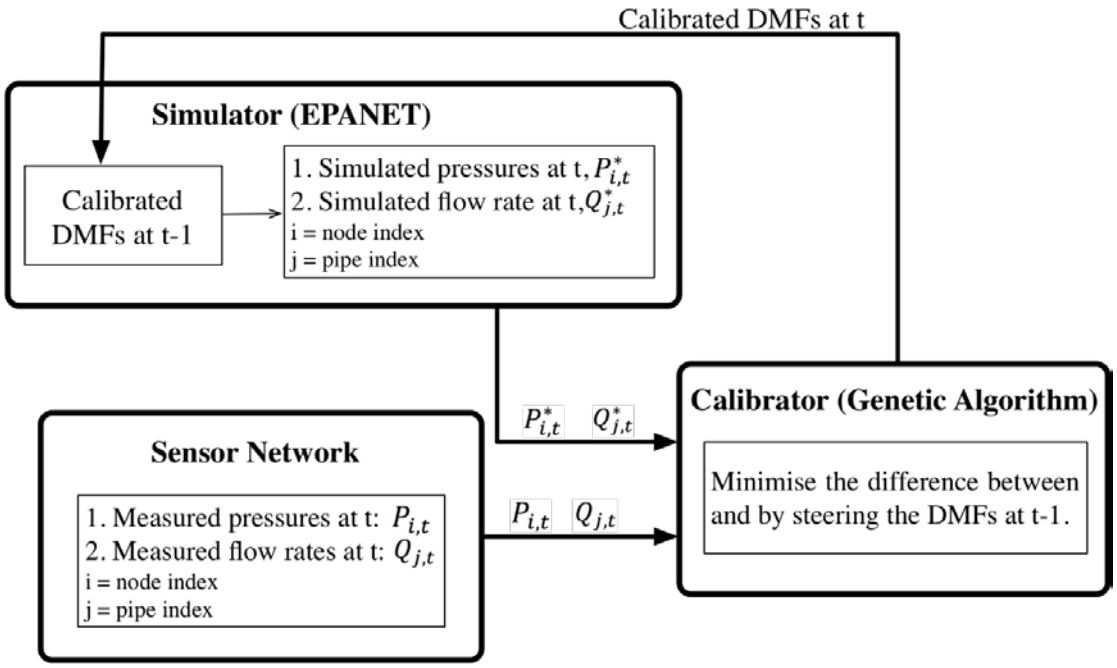


Figure 6-7 The Calibrator Designed for the Use Case: The Calibrator is implemented by using a genetic algorithm. It has an ensemble that consists of different sets of DMFs estimations. The DMF is used to drive the EPANET water model to simulate states of the water network. The simulation results are compared with sensor readings. The set of estimated DMFs that generate the closet simulation results to the sensor readings is selected as the best estimation, and simulated states based on this set of DMFs are considered as real-time states in the WDS.

The Figure 6-7 depicts the workflow of the Calibrator realised by the GA. At time t, the Calibrator steers the DMFs based on the calibrated DMFs at time t-1. By using the steered DMFs as input parameters, the Simulator generates the estimated pressures and flow rates at time t, namely $P^*_{i,t}$ and $Q^*_{j,t}$. By comparing the ($P^*_{i,t}$, $Q^*_{j,t}$) with the sensor readings ($P_{i,t}$, $Q_{j,t}$), the Calibrator steers the DMFs for another round to further minimise the fitness function shown in Expression (9).

$$\sum_{i=1}^{NP} W_P(P_i^* - P_i)^2 + \sum_{j=1}^{NQ} W_Q(Q_j^* - Q_j)^2 \qquad (9)$$

Based on existing works [82, 117], the Equation (9) is the weighted sum of difference between squares of estimated states and the sensor reading at time t. NP and NQ denote the number of sensors installed to measure water pressures and flow rates. The normalised weighting factors W_P and W_Q are shown in Equation (10) and (11), where n indicates a particular sensor.

$$W_{P,n} = NP \frac{P_n}{\sum_i^{NP} P_i} \tag{10}$$

$$W_{Q,n} = NQ \frac{Q_n}{\sum_j^{NQ} Q_j} \tag{11}$$

Furthermore, in order to evaluate the performance of the calibration, this chapter uses three metrics, Correlation Coefficient (CC), Root-Mean-Square Error (RMSE) and Mean Absolute Error (MAE) to compare estimated states (e) and sensor readings (a).

The CC defines the correlation between the estimated states (e) and sensor readings (a):

$$CC = \frac{Cov(e,a)}{\sigma_e \sigma_a} \tag{12}$$

where the $Cov()$ denotes the covariance between e and a, and the σ stands for the standard deviation. It scales from -1 to 1, and 1 signifies the perfect estimation.

The RMSE of estimated states (e) with respect to sensor readings (a) is defined as Root-Mean-Square Error:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(e_i - a_i)^2}{N_s}} \tag{13}$$

where N_s is the number of sensors. The e_i and a_i denotes the estimated states and sensor readings. The RMSE ranges from 0 to infinity, and 0 represents the estimation fits ideally with the actual parameter.

The MAE is similar to the RMSE except that it uses the absolute error rather than the squared errors.

$$MAE = \frac{\sum_{i=1}^{N} |e_i - a_i|}{N_c} \tag{14}$$

As with the RMSE, the e_i and a_i stand for a specific type of the calibrated state and sensor reading, and N_c is the amount of DMF categories. The value of MAE ranges from 0 to infinity and 0 means the best fit between calibrated states and sensor readings.

6.2 Implementation of the Use Case

6.2.1 Sensor Selection

Since water companies currently lack software to make a productive use of sensor data, they have not deployed large-scale sensor networks that can cover full details of the WDS. As shown in Table 6-1, only 39 sensors are used to calibrate 510 DMFs in SZ01. In addition, the implementation of the 39 sensors are not deployed optimally to assist the model calibration. By anticipating the future development of the sensor network, this thesis assumes that more sensors can be used to realise a more comprehensive monitoring of the WDS. Therefore, this evaluation selects 1853 junctions from the INP file as artificial sensors to augment the physical sensors provided by the water company. Since the selected sensors include all types of water demand described in the SZ01.inp, we postulate that the calibration has a better accuracy by using our artificial sensors. In order to support this assumption, we test this postulate by comparing the calibration using 28 original sensors and 1853 artificial sensors.

Artificial Sensor Readings

This experiment needs sensor data corresponding to the 1853 selected artificial sensors. Using artificial sensor readings to evaluate projects of the WDS has been applied in many previous works such as [81, 117], hence, based on related works, this experiment utilises the EPANET and SZ01.inp to generate a group of artificial states. Consequently, we remove known DMFs from the SZ01.inp when we conducted the calibration, and only evaluated the calibration results by comparing it with artificial sensor readings.

As discussed in the Chapter 1, one significant challenge in DDDAS is to keep applications on the cyber side up with physical processes. However, the INP file provided by the water company only contains DMFs every 30 minutes, and it may not represent changes of the water network in time. By reviewing recent projects in model calibration in the area of WDS, the “Hydraulic Time Pattern” can be required to be less than 15 minutes to reach the actual real-time monitoring requirement. For example, in January 2012, in a survey conducted by the Smart Water Network (SWAN) Forum, they claimed that “*a monitoring of WDS is real-time only if data transmission is every 15 minutes*” [133]. Hence, to evaluate the ability of our steering components in keeping up with this rate of data flow, more DMFs are required

to be interpolated to reach the 15-minutes updating frequency. To achieve this requirement, a software, Pattern Splitter, was developed as part of this thesis.

The Pattern Splitter utilises a customised linear interpolation method to insert an artificial DMF between two existing DMFs. Since DMFs can fluctuate randomly between two time points, instead of using the mid-value of two known DMFs as the interpolant, the Pattern Splitter uses standard normal distribution to generate the interpolant by using two adjacent DMFs as lower and upper bounds. Furthermore, by analysing the original DMF patterns, there is a small possibility for DMFs to fluctuate non-gradually. Hence, the position of a newly generated DMF is switched with its previous or next DMF with a possibility of 5%. Finally, the created data file is attached in Appendix A and the file name is SZ01_15.inp.

Figure 6-8 depicts the workflow of generating and utilizing the artificial sensor readings with the calibration process. Based on the SZ01_15.inp, the EPANET is used to build a dedicated program, which we call EPANET Engine, and it takes DMFs at a specific time point as input and outputs states of the entire water network at the corresponding time point.

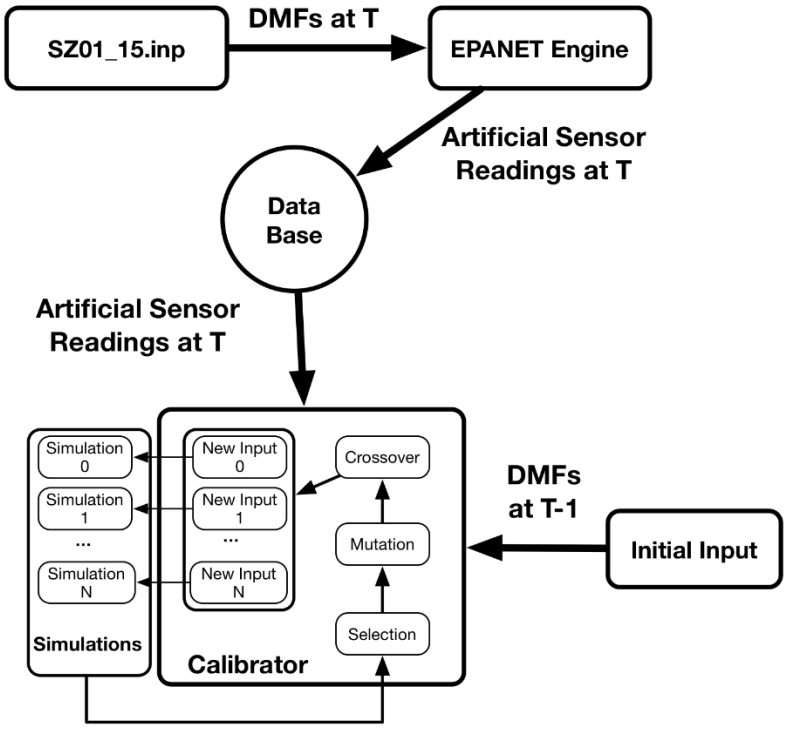


Figure 6-8 Generating and Utilizing Artificial Sensor Readings: The SZ01.inp is the input file for the EPANET software. It describes the information of the water network. The *T* is the time point at which the EPANET Engine generates artificial sensor readings. The DMFs at *T-1* are previous calibration results. They are used as the input for the Calibrator to estimate DMFs at *T*. The estimation is implemented by mutation and crossover functions. Thus, a number of different estimations are generated as one generation. It is used as input for simulations to generate the estimated state of the water network. The elite estimation is selected and sent to mutation and crossover components for another round of estimation. The calibration terminates when the elite set of DMFs can generate simulation states that are close enough to the artificial sensor reading.

As shown in Figure 6-8, DMFs at time *T-1* is used as the input for the EPANET Engine. Its results represent the artificial sensor readings at *T-1* and are stored in the Data Base. Afterwards, the Calibrator calibrates DMFs at *T-1* and compares the artificial sensor readings provided by the database with simulation results from $Simulation_0$ to $Simulations_n$. Simulations are driven by a number of different sets of DMFs, and the simulation whose results are closest to the artificial sensor readings at *T* is selected. Its DMFs are used to represent the actual DMFs abstracted from the real water network. The advantage of using the artificial sensor readings is that, since the believed "real" sensor readings are generated by the EPANET model, complications arising from noises in the sensor data are avoided. In the future work, the noises generated by real sensor data need to be handled by water engineers.

Comparing Calibration Results by Using 28 Sensors and 1853 Sensors

In the SZ01.inp, 1853 sensors are just able to cover all types of water demands that are crucial for the calibration. Hence, in this section, the calibration results obtained using 28 real sensors are compared with the calibration results obtained using 1853 artificial sensors. We argue that by using both real and artificial sensor readings, the overall accuracy of the model can be improved. The balance between the number of real and artificial sensors will be a topic for future research.

In the calibration which is based on 28 sensors, the GA is stopped when there is no fitness improvement after 40 consecutive generations. On the other hand, the stopping criteria for calibration based 1853 sensors is stopping when the fitness reaches a specific value. Consequently, five fitness values, 3000, 2500, 2000, 1500 and 1000 are used as the stopping criteria. The reason to use different stopping criteria is that for 28 sensors, the fitness value is always smaller than using 1853 sensor since the fitness is the sum of difference between sensor readings and calibration results. Hence, it is not a valid comparison to stop calibration based on 28 sensors on the same fitness value as calibration based on 1853 sensors. Additionally, as the calibration based on 1853 sensors can always have a small improvement on the fitness value, it is difficult for the calibration to stop using the same stopping criteria. Thus, the chosen stopping criteria enables the calibration based on 28 sensors to have a longer running time and greater amount of generations. Therefore, if the calibration results based on 28 sensors are still more inaccurate than the calibration based on 1853 sensors, we

can conclude that more sensors can lead to a more accurate calibration. Additionally, other methods of GA such as selection and crossover are identical in both experiments.

Number of Reference Sensors	Stopping Criteria	CC	RMSE	MAE
1853 Sensors	Fitness 1000	0.999997	0.694639	0.134121
	Fitness 1500	0.999997	0.701646	0.136441
	Fitness 2000	0.999997	0.721422	0.137312
	Fitness 2500	0.999996	0.746155	0.139456
	Fitness 3000	0.999995	0.838509	0.146798
28 Sensors	No Improvement after 40 Generations	0.954	3.604	1.628

Table 6-2 Comparing Calibration Results by Using 28 Sensors and 1853 Sensors: This table uses three criteria to evaluate the calibration results under different sensor numbers. CC indicates the correlation between calibration results and the sensor readings. It scales from -1 to 1, and 1 signifies the perfect estimation. Root Mean Square Error (RMSE) ranges from 0 to infinity, and 0 represents the estimation fits ideally with the actual parameter. Mean Average Error ranges from 0 to infinity and 0 means the best fit between calibrated and actual DMFs.

To achieve the goal of this experiment, the calibration process was run 50 times for each stopping criterion. Based on the calibration results, the CC, MAE and RMSE introduced in Section 6.1.3 are used as metrics for this evaluation, and Table 6-2 shows their mean values of 50 runs. As a result, CC, RMSE and MAE all indicate that the calibration results are much better when using 1853 sensors. The most noticeable improvement is on RMSE whose value decreases around 77% to 81%. Consequently, we conclude that the results indicate it is necessary to deploy more sensors to monitor the WDS if a higher accuracy is required by current and future functions of the WDS.

It can be argued that a smaller amount of sensors can achieve the same or close calibration results since for metric CC the improvement is not significant. To response this argument, we need to state that it is difficult for the author of this thesis to select suitable positions for sensors since we do not have specialised knowledge in WDS. Additionally, by calibrating areas with a large number of sensors engineers can determine the best locations for a smaller number of sensors that can be kept permanently.

Thus, based on the discussions and meetings with professionals in water industrials, all sensors are selected based on the INP file we obtained from Northumbrian Water Company.

Since the INP file emphasizes the junctions that have significant influence on the water demand specifically, these junctions are selected as the sensor locations for this experiment.

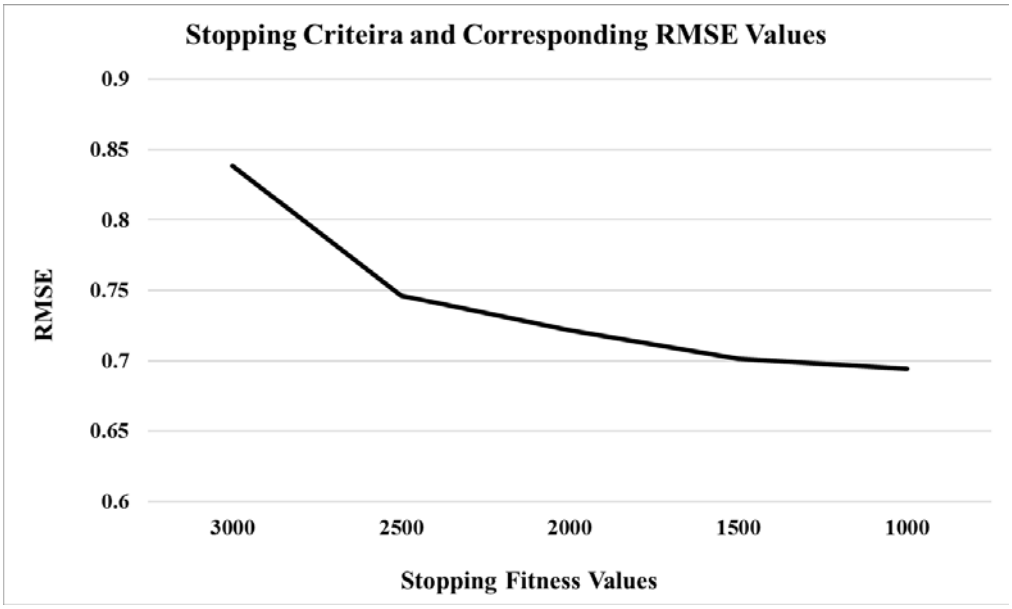


Figure 6-9 Stopping Criteria and Corresponding RMSE Values: This figure compares values of RMSE by taking 3000, 2500, 2000, 1500 and 1000 as stopping criteria. The percentages represent the improvement in the calibration by using a smaller fitness value as the stopping criteria. This figure indicates an improvement can be achieved by using a restricted stopping criteria.

Furthermore, Figure 6-9 presents the RMSE of the calibration based on 1853 sensors. The number below the line indicates the decrease in value of the RMSE. As values required by stopping criteria decreases, the increase of the improvement represented by RMSE slows down. As a result, the stopping criteria can also be configured as a steerable parameter which can alter the monitoring accuracy of the system. However, exploring a suitable stopping criteria for this specific WDS is beyond the scope of this thesis, and this thesis only demonstrates that we provide more dynamic routines for water engineers to explore the WDS by using computational steering.

Increasing the number of reference sensors greatly increases the workload of the Calibrator because more DMFs need to be calibrated in order to lower the fitness value. Hence, there are two more reasons for this thesis to select a more considerable number of reference sensors, 1) to evaluate the Computing Resource Manager which is supposed to assign sufficient computing resources to meet the time requirement and 2) to embody the value of Dynamic Data-Driven Computational Steering (DDDCS) which can improve the efficiency of the calibration.

6.2.2 Regular and Abnormal Situations

To achieve the objectives of this thesis, it is necessary to extend steering to anomalous situations, namely abnormal situations, as well as more predictable ones, namely normal situations. In evaluation of DDDCS, time management and computing resource assignment, we need different workload to understand the performance of our projects under different situations. However, utilizing sensor readings to assist daily management tasks of WDS still requires considerable collaboration between computer scientists and water engineers. Hence, this thesis lacks water engineering knowledge to provide a comprehensive evaluation on particular variation patterns of the WDS that can interest water engineers. As a result, we only evaluate the smallest and biggest variation that are obtainable in the data provided by the Northumbrian water company. Moreover, we assume that if the time management function proposed by this thesis can manage to handle the smallest and biggest variations of the WDS, then the variation in between can be handled as well. Consequently, we choose 01:00 am as the time point for the regular situation which is supposed to have the smallest water demand variation of a day. Additionally, the abnormal situation is defined as a pipe burst event in which the water demand has significant variation.

The first task in separating the regular and pipe burst situations is to generate corresponding artificial sensor data. For the regular situation, the sensor data can be created directly by using the SZ01_15.inp with the EPANET engine. However, the regular SZ01_15.inp file does not contain the abnormal information of a pipe burst. Thus, an artificial pipe burst event must be created in the input file. Additionally, to guarantee the artificial pipe burst data is reliable, it must be based on a real pipe burst.

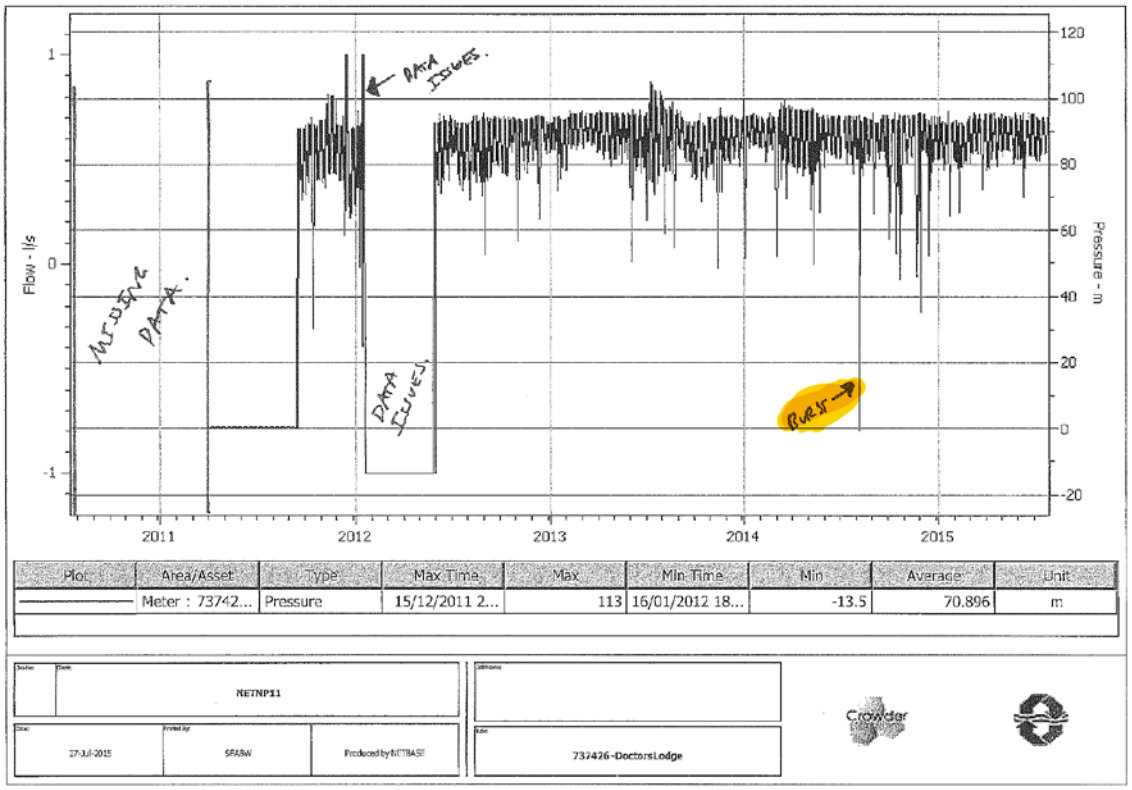


Figure 6-10 Burst Sensor Reading: This figure is the burst sensor report provided by the Northumbrian Water company. We use it to find the pressure drop scale in the pipe burst. Since it is not feasible to get the pressure reading from this figure, we take the burst junction ID, burst time and date to find the burst in corresponding sensor readings.

Thanks to the collaboration with the Northumbrian Water company, they provided us the information of a real pipe burst. As shown in Figure 6-10, this pipe burst happened between 2014 and 2015. The ID of the sensor that monitored this pipe burst is 737426. According to the meter-to-junction relations provided by the water company, the corresponding junction ID is 47298618. Hence, the junction 47298618 must be located in the burst area. But since the water company is only using 28 sensors to monitor the entire area, it is not possible to tell if the burst happened exactly on the position of the junction 47292618. However, since this burst information is the only information provided by the water company, in this thesis, it is assumed that the burst happened on the same location as the 47298618.

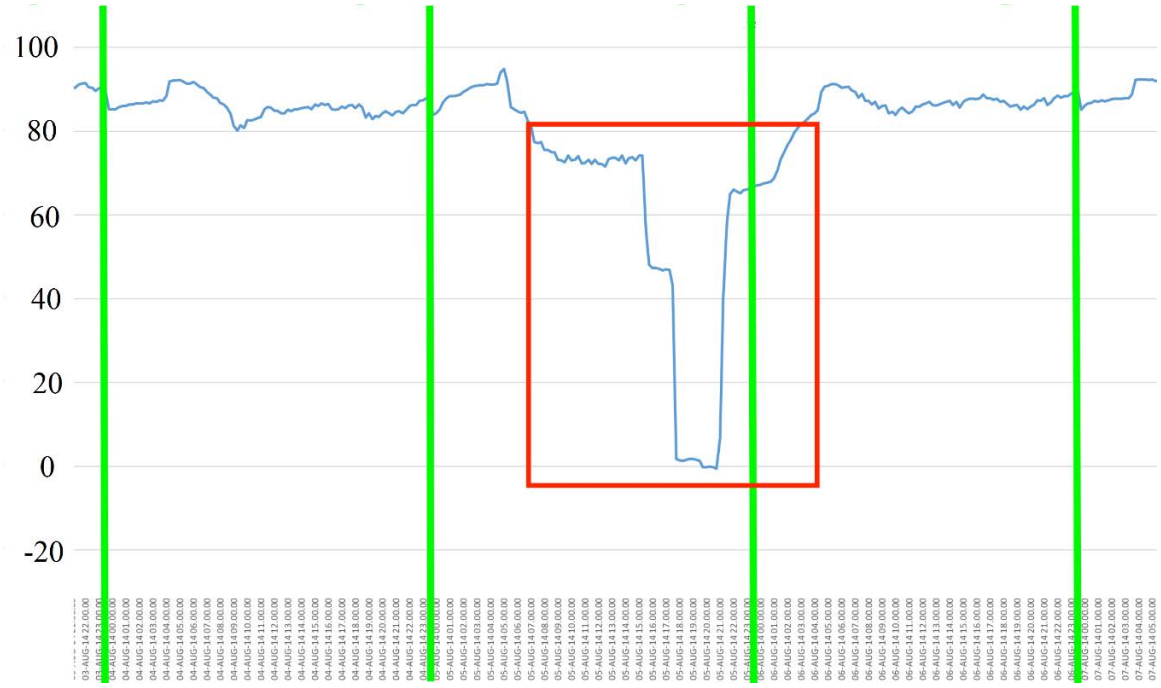


Figure 6-11 Pressure of 47298618 during the Burst Period: Figure 6-10 provides us the range of the date on which the burst happened. Hence, this figure shows the plot of real sensor reading during that time to locate the pipe burst. The x axis indicates the data and time, and the y axis indicates the water pressure. The vertical lines separate sensor readings in days. The rectangle indicates the pressure drop caused by the pipe burst we found.

Depending only on Figure 6-10, it is difficult to recognize the burst sensor reading clearly. Thus, we attempted to further explore the sensor readings via the date indicated in Figure 6-10. Therefore, by checking the sensor records between 1st August and 10th August in 2014, Figure 6-11 reproduces the pressure of component 47298618 in three days around the burst time. The interval between two vertical lines stands for the time of a day. As a result, by comparing the pressure reading of a day before and after, we found a great pressure drop which ranges between 0-80 meters of pressure head (height is the proxy for pressure).

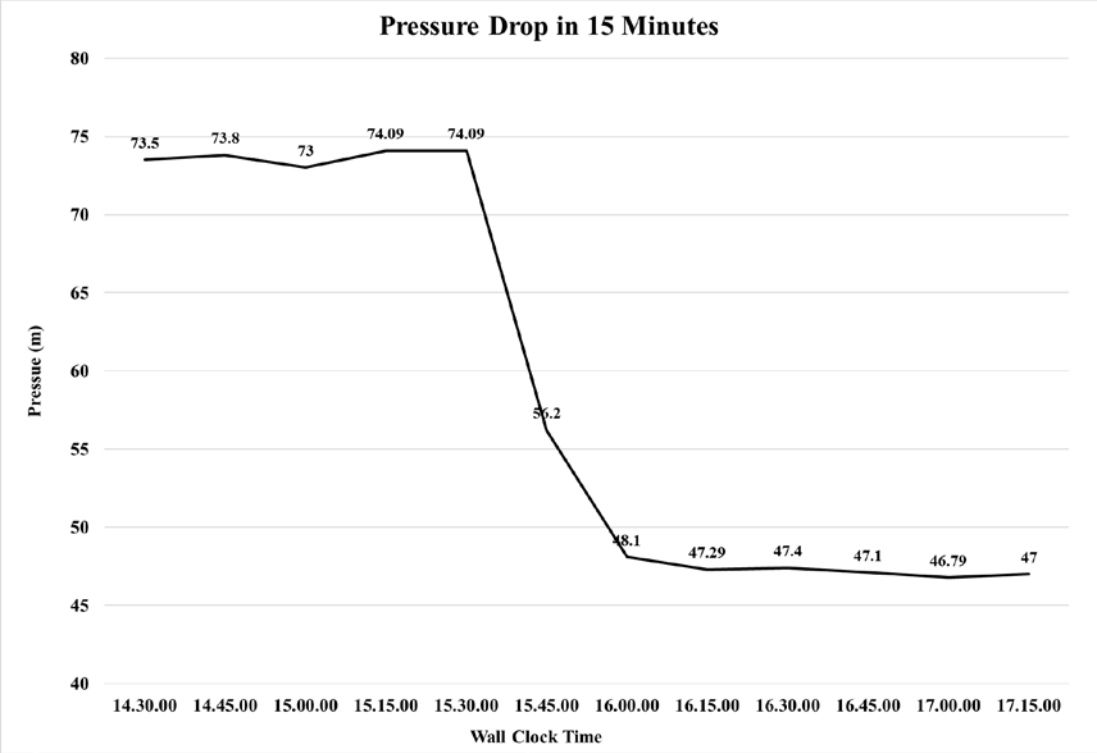


Figure 6-12 Pressure Drop in 15 Minutes: This figure zooms into the pipe burst found in last figure and indicates the pressure drop in a pipe burst can be from 74.09 to 48.1 in 15 minutes.

To zoom into the interval of the burst event, Figure 6-12 shows the drop details in every 15 minutes between 14:30 and 17:15. The most noticeable decrease happened between 15:30 to 15:45, during which the pressure decreased more than 20% and fell from 74.09 to 56.2. As a result, we take the 20% fell on pressure as the biggest variation of the WDS.

Even after understanding the scale of the pressure decrease during a burst, it is not appropriate to directly alter artificial pressure values of the burst junction in the database which is used as the sensor reading in the calibration workflow shown in Figure 6-8. In fact, a burst can not only affect one junction in the network, and it also has an area of influence in which other junctions are affected as well. Hence, to generate the artificial sensor reading for the burst event, the challenge in directly altering the artificial sensor reading is that we cannot estimate the recession of the burst influence on adjacent sensors.

To tackle this challenge, this thesis provides an original method to generate artificial burst sensor readings in WDS. Rather than altering one or several pressure sensor readings, the artificial burst sensor readings are generated by adding a new type of water demand to the burst junction in the INP file and we name the file as SZ01_15_burst.inp. This water demand simulates the burst as a consumer who suddenly has a great water consumption. As a result, by running the modified INP file with the EPANET engine, the artificial burst sensor

readings are generated with its effects on adjacent sensors automatically. Since one type of water demand corresponds to one DMF, hence, we then need to find a suitable value of the new type of DMF to manage a given fall on pressure. The burst simulation is conducted several times with different burst DMFs. Finally, a suitable DMF value, which causes a 22% pressure decrease on the burst junction, is used to represent the DMF for the artificial burst.

Afterwards, the challenge is how to calibrate the pipe burst. Junctions located in nearby areas can share same water demand patterns. If the artificial pressure of one junction is manually increased in the database, the Calibrator must alter the DMFs, which are owned by the burst junction and are also shared with other junctions, in order to match the calibration results with the burst sensor reading. However, by doing so, the calibration results of other junctions cannot match their sensor readings.

As a result, this thesis provides a novel method in calibrating burst event in WDS. The first step of this method is to locate sensors that have the highest probability of observing the pipe burst. To achieve the first step, we conduct a simulation based on calibration results at *T-1*. The simulation results are compared with the artificial sensor readings at *T*. The difference should be in a small range in regular situations. However, during a burst event, several big differences are supposed to emerge on a number of sensors, and these sensors are assumed to be around the burst site.

Sensor ID	Sensor Readings	Simulation Results	Difference
47295565	29.8972	39.6749	0.246445486
47295566	29.9176	39.6749	0.245931307
47295932	29.8179	39.4987	0.245091611
47295564	37.5886	48.2876	0.22156827
47295461	37.7635	48.4274	0.220203852
47300261	37.6856	48.3072	0.219876126
47295484	34.4052	42.8062	0.196256617
47295483	34.4921	42.8959	0.195911497
47297274	34.65	42.8579	0.191514283
47297273	37.2391	45.7988	0.18689791
47300260	38.4054	47.1192	0.184930984
47297272	38.4432	47.109	0.183952111
47300270	36.9304	45.1994	0.182944906
47298426	36.8159	44.9201	0.180413668
47298443	36.8358	44.9376	0.180290002
47300263	40.443	48.6028	0.167887447
47300257	40.8118	48.9678	0.166558432
47297271	40.576	48.6599	0.166130633
47297270	40.6936	48.7504	0.165266336
47300256	41.0449	49.144	0.164803435

Table 6-3 Sorted Difference between Sensor Readings and Predictions: The sensor readings are pressures at time *T*. Their junctions IDs are shown in the first column. The Simulation Results are obtained from the states of the water network simulated using DMFs at *T-1*. All junctions are sorted by the difference between prediction and sensor readings. If the difference is more than 20% of the prediction, the junction is highlighted on the top, and considered as junctions linked to the burst pipe.

Table 6-3 presents the sorted difference between sensor readings and predictions. The burst demand is added to junction 47295564 which is where the burst happened artificially. In this evaluation, sensors that have a difference more than 20% between sensor readings (at *T*) and simulation results (at *T-1*) are selected as the sensors with abnormal readings. Consequently, all of junctions shown at the top of the Table 6-3 are close to the burst. As shown in Figure 6-13, the selected sensors shown in Table 6-3 are marked as stars on the map and the red star, 47295564, is the burst site. As can be seen, all sensors that have significant difference from the predictions are around the burst site.

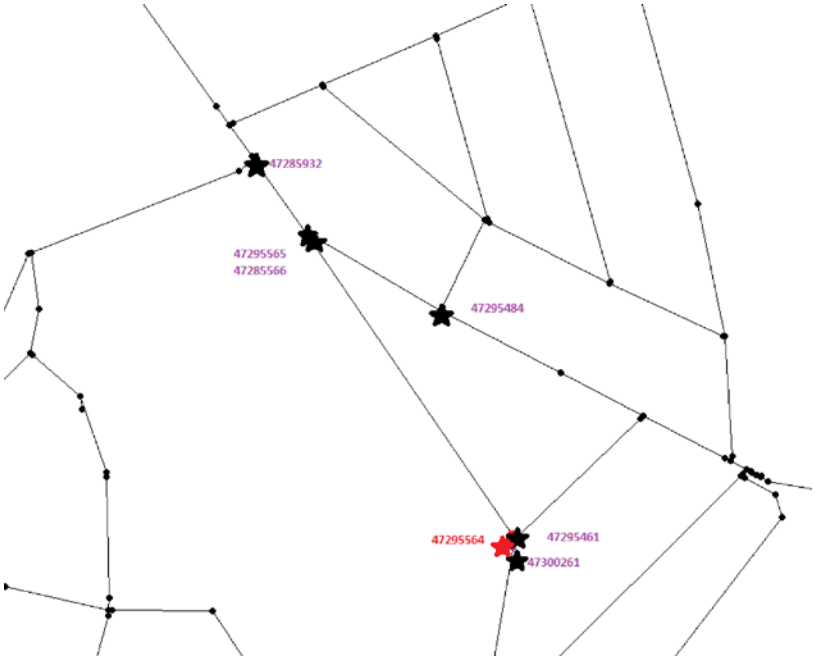


Figure 6-13 Burst Shown on Map: The highlighted junctions shown in Table 6-3 are shown in this figure as stars. It indicates all highlighted junctions are surround the burst one which is highlighted as the red star.

After locating the burst areas, the Calibrator needs to calibrate the DMF to indicate the burst scale. However, the selected sensors shown in Figure 6-13 do not share exactly the same DMFs, and even worse, they all share DMFs with other junctions that are not affected by the burst. Hence, in order to calibrate the selected "problem" sensors without affecting other "innocent" sensors, a new water demand pattern named ANOMALY is added into the INP file and applied to the "problem" sensors in the [DEMADNS] section. Together with the new pattern, an initial water demand is assigned separately to suspected burst sensors by taking the mean of their original water demands in different categories. All initial DMFs of

the new pattern are set to 0. As a result, the GA is able to calibrate ANOMALY DMFs without interfering other sensors.

6.3 Evaluation of Human-Driven Computational Steering

As stated by Munir [107], human-in-the-loop feedback control system provides interesting opportunities in DDDAS. However, most existing projects use only algorithms in the calibration and simulation of WDS and the influence of human uses is neglected. This thesis believes that human users have limitations in conducting steering when parameter space is large, but we also believe their intelligence and experience can be used on a higher level steering based on Dynamic Data-Driven Computational Steering (DDDCS). Inspired by existing computational steering projects, which are generally driven by human users, this thesis proposes a hybrid computational steering architecture to extend the usability of computational steering in the context of DDDAS on multiple levels. Hence, this section needs to 1) specify functions that can be facilitated by using Human-Driven Computational Steering (HDCS) in applications of WDS and 2) manage the calibration steered by human users to finish in time.

6.3.1 Selection of Steerable Parameter

In order to evaluate the HDCS, we firstly need to decide which parameter needs to be registered as steerable parameters. The steerable parameter must 1) affect high-level functions such as monitoring and prediction described in Chapter 4, 2) fit into a real application scenario in which water engineers need to manually steer the system and 3) not require taking effect under a tight time constraint since it is difficult to control the time consumed by human users. However, because of the integration of HDCS and DDDCS has rarely been studied in the existing projects, it is challenging for a PhD student in computer science to propose a suitable application scenario in the WDS, which requires dedicated knowledge in water engineering. Hence, we worked with engineers in Northumbrian water company to propose an application scenario which they are interested and can also be applicable in the future.

Consequently, parameters of the calibration component are selected as steerable parameters. Firstly, as the calibration is the most time-consuming function compared with the monitoring and prediction, this evaluation selects the parameters that can affect the calibration as the steering target. Hence, users can control the progress speed of the application by steering the

calibration component. Secondly, the calibration is an intermediate function that supports monitoring and prediction. Since results of the calibration component have a direct effect on the prediction and monitoring components, parameters of the calibration component can also affect result of ultimate purpose of the system. Subsequently, by considering the trade-off between the time consumption and accuracy of monitoring and prediction, water engineers are interested in steer calibration parameters to explore suitable values for particular scenarios.

Afterwards, we need to decide on what application scenarios, parameters of the calibration need to be steered. As a new emerging area in the WDS, most current works focus on extracting high-level information from sensor networks to human users. But the requirement of integrating human influence is always neglected by existing works. On the other hand, WDS modelling used by many water companies is still based on offline calibration and analysis. Even though sensors implemented by water companies have the ability to upload data in minutes, engineers still analyse water networks in a daily or weekly manner. This situation limits water engineers in exploring additional functions in using the real-time sensor data. As a result, there is a critical gap between the functionality that can be provided by computational methods and the ability of water engineers to use it to make decisions.

After discussing with engineers in the Northumbrian Water company, we decide to use the number of sensors as the steerable parameters. The corresponding scenario is built under the promising real-time monitoring function in which the fidelity of the monitored network mainly depends on the coverage of sensors. To have a more accurate understanding of the area of interests, engineers may introduce more sensors in a specific district. On the other hand, a big sensor number can bring a large amount of computational burden to a WDS system when engineers only require a general but real-time overview of the monitored system. Consequently, the great computing workload may reduce the update frequency of the system and increase the consumption on computing infrastructures and sensor networks. Therefore, water engineers have the potential requirement to steer the number of sensors.

We need to emphasize that in this experiment we do not have direct control of physical sensors. As this thesis only focuses on the high-level functions of DDDAS, steering sensor number means steering the number of sensor readings the calibration process takes, and we do not steer the number of sensors implemented in the physical network. Moreover, obtaining real-time sensor readings from physical sensors implemented by Northumbrian

Water company requires complex business and security negotiations. Hence, instead of using actual real-time sensor readings, we store them into a database and assume the information extracted from the database is in real-time. Furthermore, since methods and benefits of utilising sensor networks are still under exploration, their implementation in real-world is limited. Thus, Northumbrian only applied 40 district pressure sensors in its covered area. The benefit of using a large amount of sensors to cover the entire water network has been presented in Section 6.2.1, but we also need to demonstrate that the steering ability can help water engineers based on the amount of sensors they are currently using. Hence, in this section, we only use a small number of sensors to demonstrate that the HDCS can be implemented and the time management method can handle the fluctuation raised by steering operations of HDCS.

Historical logs of these sensor reading are incomplete, from the sensor reading we obtained from Northumbrian, some sensors do not have readings during some periods due to sensor failures. Therefore, we choose 28 out of 40 sensors that have available sensor readings at 12pm on 27th of January in 2012 as the maximum available sensor number. Finally, for the purpose of clarification, we emphasize that this experiment is conducted based on real sensor readings and artificial sensor readings introduced in 6.2 is not generated for this experiment.

6.3.2 Results of Human-Driven Computational Steering and its Time Management

This section evaluates the HDCS from two aspects: steering function and time management. Since steerable parameters belong to the calibration process, effects on both aspects are presented by steering the calibration process. In terms of the steering function, this section demonstrates that the steering affects accuracy and time consumption of the calibration. In addition, the time management is evaluated by examining whether time manager can estimate the required running time of the calibration based on the number of sensors steered by human users.

Since the Genetic Algorithm (GA) is selected as the calibration algorithm in this thesis, we focus our estimation on the execution time of the GA. The running time of the GA varies with its implementation methods such as fitness function, population size, crossover and mutation methods and the model which describes the problem area. Based on fitness functions in existing works introduced in Chapter 5, we use a simple GA implementation with the fitness function,

$$F(x) = \sum_{ns=1}^{NS} W_{ns} \times [OS_{ns} - CS_{ns}]^2 \quad (15)$$

where *ns* designates the ns-th sensor of *NS* sensors, W_{ns} is the weight of the ns-th sensor and OS_{ns} and CS_{ns} denotes the ns-th observational sensor reading and its corresponding simulation result after the calibration. Thus, the time taken by the GA to converge can be determined by the sensor data OS_{ns}. The tournament selection and Gaussian mutation method are used in this simple GA implementation. Based on the previous study [82], the population size is set to be 80. Since no steerable parameters are registered in the selection, mutation and crossover methods, the number of sensors steered by human users is the only factor that affects the converging rate of GA given by equation (15). For example, when the number of reference sensors increases, the GA must consume more time to match more simulation results with more real sensor readings. On the other hand, for an extreme instance, when there is only one pressure sensor used as the reference, the calibrator only needs to match the corresponding node pressure calculated from the simulation without considering other parts of the network. Hence, the calibration with a small number of sensors can be finished much quicker than the calibration with a large number of sensors.

Results of Human-Driven Computational Steering

In order to further support the assumption that using more sensors can provide engineers more accurate states of the water network and to demonstrate the capability of HDCS, we steer the number of sensors from 7 to 14 and to 28 in the calibration. To ensure the number of sensors are the only variables in this experiment, the same sensor readings and previous calibration results are used for each calibration. The three commonly used metrics: Correlation Coefficient (CC), Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE), which have been introduced in Section 6.2, are used to evaluate calibration results. The outcome is shown in Table 6-4.

	CC	RMSE	MAE	Time(seconds)
Before Calibration	0.877	11.779	8.134	——
28 Reference Sensors	0.954	3.604	1.628	2028.42
14 Reference Sensors	0.936	6.399	3.877	1237.84
7 Reference Sensors	0.921	7.867	4.960	422.507

Table 6-4 Comparing Calibration with Different Number of Sensors: This table uses three criteria to evaluate the calibration results under different sensor numbers. CC indicates the correlation between calibration results and the sensor readings. It scales from -1 to 1, and 1 signifies the perfect estimation. Root Mean Square Error (RMSE) ranges from 0 to infinity, and 0 represents the estimation fits ideally with the actual parameter. Mean Average Error ranges from 0 to infinity and 0 means the best fit between calibrated and actual DMFs. This table presents the relation between sensor numbers and calibration accuracy. It also indicates the trade-off between accuracy and time consumption.

As we can see, when users steer the calibration process to use more sensors, the results of calibration become better, but the calculation time also increases as the price. On the other hand, when users need to reach a tight time window instead of a more accurate calibration, they can steer to reduce the number of reference sensors. The results indicate that we provide a promising use case to connect the development of sensor network to the practical engineering in WDS. Finally, this result also demonstrates the design and the development of HDCS, which is based on conventional computational steering, managing to enrol human users into the control loop of DDDAS. Consequently, the steering method proposed by this thesis can support users to steer more parameters such as the stopping criteria and population size of the GA.

Running Time Estimation of Human-Driven Computational Steering

However, merely achieving the steering capability does not satisfy the requirement of a real-time DDDAS. In order to guarantee the critical real-time nature of the system, we need to evaluate the time management component introduced in Section 4.3. Furthermore, as stated in Section 4.3, this thesis cannot demonstrate a comprehensive time management method that can handle all human steering interferences. Hence, the time management method we present is based on a training and learning process so that different and unknown steering interferences can be learnt by the time management component. As a result, this section only examines the accuracy of estimating the time consumption by using the steerable number of sensor as an example.

In our experiment, we select 8 problem sizes in which respectively contains 7, 10, 13, 16 ,19, 22, 25 and 28 pressure sensors, then we execute the calibration process 20 times for each problem size to calibrate the water model by using real sensor information obtained from

Northumbrian Water Company at 12pm on 27th of Jan in 2012. The HPC infrastructure used in this experiment is a HPC cluster named RedQueen[20]. The GA is stopped after 40 consecutive generations that cannot reduce the result of the fitness function (15), and the total generations taken by the GA is considered as the representation of the convergence speed.

As a result, the required generation numbers to meet the stop condition for each problem size is shown in Table 6-5, and accordingly, Figure 6-14 depicts the distribution of required generation numbers.

Trials	7 Sensors	10 Sensors	13 Sensors	16 Sensors	19 Sensors	23 Sensors	25 Sensors	28 Sensors
1	196	274	167	139	196	220	220	136
2	173	146	294	191	173	82	136	138
3	238	239	206	150	238	192	190	273
4	330	205	174	140	330	263	159	219
5	223	222	275	106	223	248	191	119
6	152	137	226	140	152	224	167	204
7	184	172	90	177	184	248	191	267
8	137	215	113	452	137	305	273	399
9	251	219	166	183	251	246	198	239
10	366	193	184	210	366	181	185	237
11	155	162	179	257	155	268	288	268
12	228	178	139	146	228	210	234	221
13	153	166	118	136	153	185	253	219
14	161	184	275	284	161	210	214	177
15	190	100	248	149	190	316	293	237
16	141	176	216	147	141	130	252	199
17	262	220	271	200	262	301	119	276
18	165	208	277	107	165	109	363	446
19	276	148	122	275	276	135	326	297
20	237	153	247	299	237	223	205	140

Table 6-5 Required Generation Numbers: This table lists the number of generations reuqired by the genetic algorithm to converge when using 7, 10, 13, 16, 19, 23, 25 and 28 sensors to calibrate DMFs, and the experiment is conducted 20 times for each number of sensors.

[20] The introduction of RedQueen http://ri.itservices.manchester.ac.uk/redqueen

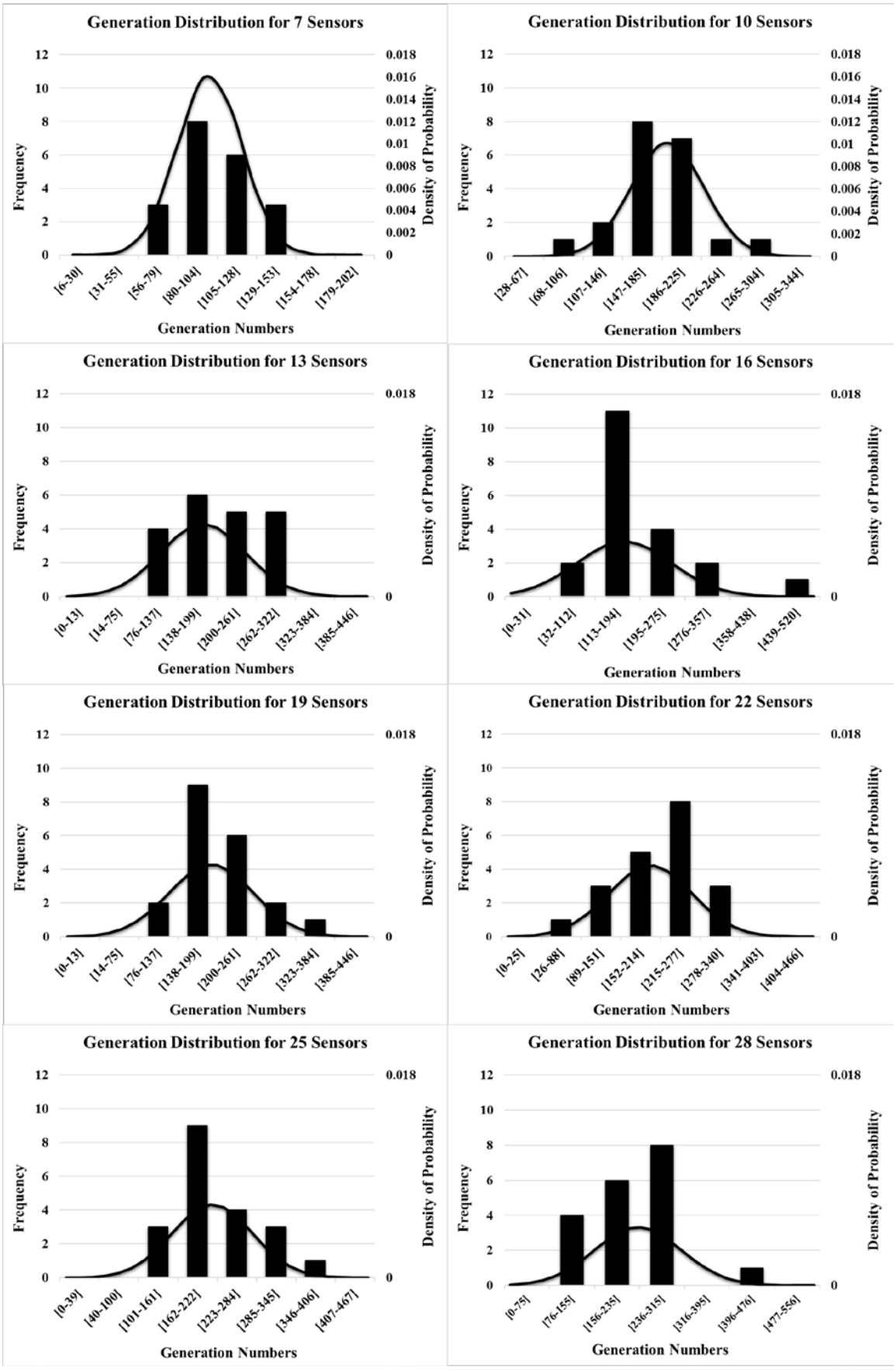


Figure 6-14 Distribution of Generation Numbers Using HDCS: The standard deviation (σ) and mean (μ) generation numbers required by the GA based on 7, 10, 13, 16, 19, 22, 25, 28 sensors are calculated. Each

calibration was executed 20 times. Points on the x axis are calculated in the form of $[\mu-4\sigma, \mu-3\sigma)$, $[\mu-3\sigma, \mu-2\sigma)$, $[\mu-2\sigma, \mu-\sigma)$, $[\mu, \mu+\sigma)$, $[\mu+\sigma, \mu+2\sigma)$, $[\mu+2\sigma, \mu+3\sigma)$. The y axis on the left indicates the number of calibrations whose required generations number belongs to the interval indicated on the x axis. The sub-figures give a general recognition on the distribution of the required generation numbers. Based on this figure, we have some visual evidence that we are justified in the assumption that we can use the Gaussian distribution to estimate the required generations. A normality test was conducted to further confirm the assumption of using Gaussian distribution. The result of the test is used to depict a linear curve that represents the density of probability of a Gaussian distribution in each subfigure.

Based on Figure 6-14, we have some visual evidence that we are justified in the assumption that we can use the Gaussian distribution to estimate the required generations. Generally, visual inspection and numerical inspection are two methods that can be used to identify the Gaussian distribution. As our experiment has a small sample size, it is not convincing by only using Figure 6-14 to conclude that the Gaussian is a justifiable approximation for the data distribution. Hence, the numerical methods are used to further test our assumption. Since Shapiro-Wilk is widely used method to test normality, in this experiment, we only take Shapiro-Wilk as the numerical method [52]. A null-hypothesis used by the Shapiro-Wilk is that the tested distribution is a Gaussian distribution. It tests the normality by calculating the probability that a tested sample was from a normal population, and the probability is indicated by the p-value. Typically, if the probability of finding such a sample is greater than 5%, it claims that we cannot reject the null-hypothesis. As a result, many scientists claim when a p-value is greater than 0.05, the distribution of the tested sample can be assumed as normal distribution.

Tests of Normality (Shapiro-Wilk)	
Number of Sensors Used in Calibrations	p-values
7 Sensors	0.41
10 Sensors	0.23
13 Sensors	0.23
16 Sensors	0.61
19 Sensors	0.13
22 Sensors	0.69
25 Sensors	0.60
28 Sensors	0.19

Table 6-6 Normality Test of Human Driven Computational Steering: If p-values are greater than 0.05, then we claim the distribution of generations can be assumed as Gaussian distribution.

Table 6-6 shows the results obtained from Shapiro-Wilk method. Since the significance values for all sensor numbers are bigger than 0.05, it means there is no significant difference between the observed distributions of generations and the normal distribution. The shape of each Gaussian distribution is shown as a linear curve in Figure 6-14. Consequently, the

estimated generation number, E_g, for a particular problem size can be estimated based on the features of the Gaussian distribution and represented as:

$$E_g = \lfloor n * \sigma + \mu \rfloor \quad (16)$$

which is the sum of the standard deviation, σ, and mean generations, μ. n is factor that can be set depending on the requirement of the system. In this experiment, we use $n = 1$ as a standard value since when $E_g = \sigma + \mu$, the estimation can cover situations in which the number of generations required by the GA can be from 1 to $\sigma + \mu$. This range covers about 84% of the distribution possibility. This thesis proposes the use of this n as a steerable parameter in the future. This is because it raises an interesting trade-off in which, although increasing n can improve the estimation accuracy, a large amount of computing resources can be wasted since more calibrations can converge faster than the estimation. As a result, the estimated generation numbers required by the GA for each steered sensor numbers are shown in Figure 6-15.

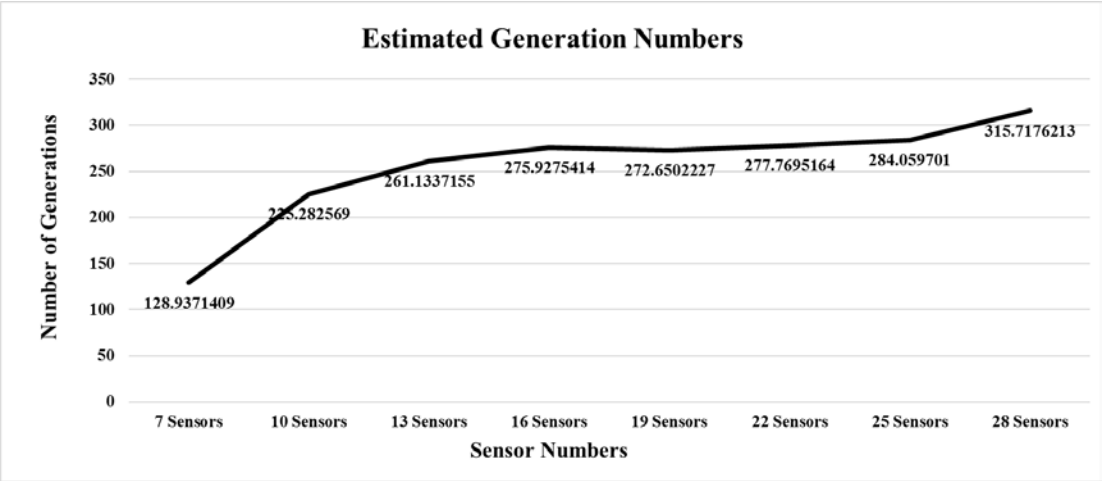


Figure 6-15 Estimated Generation Number in Human-Driven Computational Steering: The x axis indicates the number of sensors which is the steerable parameter in this experiment. The figure indicates the required running time increases as the number of sensors used in the calibration is steered to increase.

Since the running time of generations are close, the total running time of the calibration depends on the number of generations. Consequently, if users require a more accurate result, they can steer the calibration to increase the sensor number, and this improvement is limited by the time constraint in DDDAS since increasing sensor number also rises the required number of generations.

For testing the accuracy of the estimation, we run another 50 rounds of calibrations for each problem sizes, and the results are shown in Table 6-7. Above 85.8% of the results fall into

our estimation among which 31.8% of them have a difference from the estimation that is less than 1σ, and 72.5% of results have a difference that is less than 2σ. However, 14.3% calibrations still ran out of time before their convergence. As the calibration is terminated when it runs out of time, our method may affect the accuracy of calibrations. Since execution time estimation is not the main aim of this thesis, and the evaluation of time management is only used to support our hybrid computational steering, we consider 14.3% is an affordable estimation error.

Result of Execution Time Estimation in Human-Driven Computational Steering				
Steered Sensor Number	**Fall into Estimation**	**Difference between Real Requirement and Estimation**		
		Less than 1σ	Less than 2σ	Less than 3σ
7	41	41	15	41
10	39	39	14	39
13	41	41	15	41
16	46	46	19	46
19	44	44	15	44
22	41	41	20	41
25	45	45	16	45
28	46	46	13	46

Table 6-7 Result of Execution Time Estimation in Human-Driven Computational Steering: The first column is the steered number of sensors for each experiment. The second column is the number of experiments in which the required number of generations fall into the range of estimation. Less than 1σ, 2σ and 3σ mean the difference between the actual required generation number and the estimated generation number is less than $n\sigma$. It indicates our method may cause a waste of computing resources.

In a conclusion, the proposed time management method, although has a margin of errors that can cause computing resource waste, manage to estimates the required running time of GA after it is steered by human users. Hence, it supports our design of HDCS. Additionally, the range of steerable sensor numbers are restricted by the number of sensors practically deployed by water companies. Therefore, it demonstrates its value based on current sensor networks.

This section solely describes the estimation of generation numbers required by GA. In addition to this estimation, the time management also needs to map the number of generations to the wall-clock time. Additionally, we also need to discuss the relation between

the amount of computing resources and the estimated running time. As a result, the rest time management methods will be discussed in Section 6.5 together with the time management of DDDCS.

6.4 Evaluation of Dynamic Data-Driven Computational Steering

One contribution of this thesis is to integrate computational steering with the calibration process of DDDAS. Since we hypothesise computational steering optimises the interaction between the calibration and simulations, an improved time efficiency of the calibration is expected as the result of this evaluation. In this use case, the time efficiency of the calibration has a correlation with the converging speed of the Genetic Algorithm (GA) which is the predominant time consuming component in the calibration as introduced in Section 2.2.2. When the number of individuals in one generation is fixed, the converging speed depends on two factors: 1) the number of generations the GA requires, and 2) the execution time required by each individual of one generation. This thesis focuses on the second factor and the population size is increased to 240 since 1853 sensors provide a more complex calibration task compared with 28 sensors.

This evaluation is designed in 4 groups, and in each group the calibration is executed 10 times. The group is divided under 2 mutual conditions that are with or without computational steering and using sensor readings of normal or abnormal situations. The calibration with or without computational steering is represented by the terms steerable calibration and non-steerable calibration. In addition, the abnormal sensor readings are generated in the artificial pipe burst event discussed in Section 6.2.2. Furthermore, to simplify the experiment, the running time of the first 10 generations is measured instead of waiting for the calibration to converge. Hence, this section uses the time consumed by 10 generations to represent the converging speed that can be increased by using Dynamic Data-Driven Computational Steering (DDDCS). Finally, Figure 6-16 shows the average running time of the 10 executions in each group.

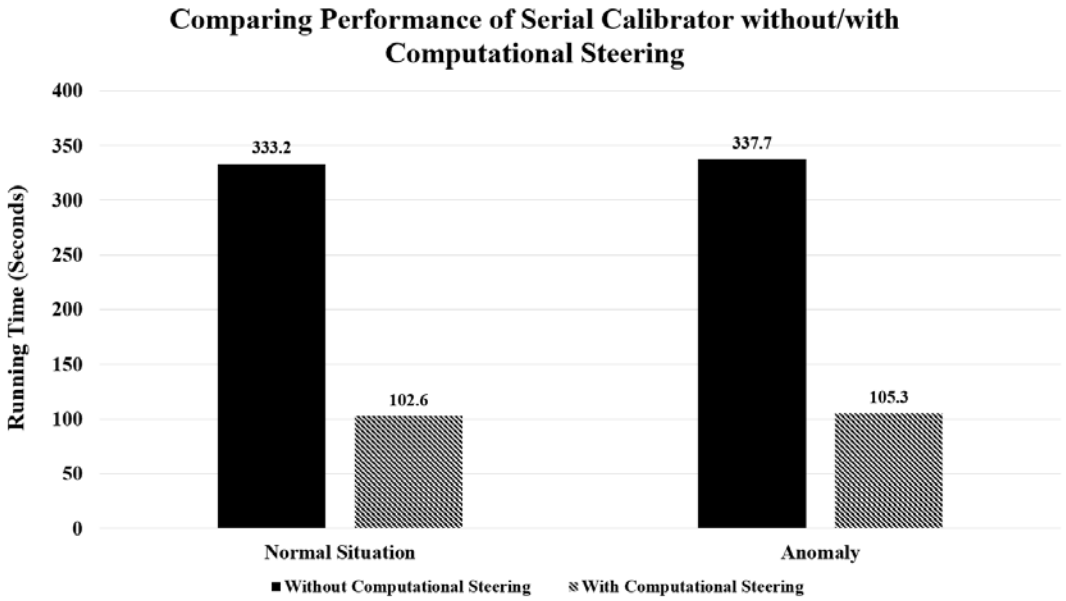


Figure 6-16 Comparing Performance of Serial Calibrations with and without Computational Steering: No matter based on what kind of situations, computational steering can efficiently reduce the running time of the calibration.

As a result, after 10 generations, the running time taken by the steerable calibration is three times less compared with non-steerable calibration in both the normal and pipe burst situations. Additionally, it is necessary to further compare the execution time by using the calibration in parallel. To facilitate the experiment, only the normal situation is utilised.

To collect the running time of the parallel Calibration, two sets of experiments have been conducted by using two and four cores independently. Figure 6-17 depicts the results by comparing the parallel calibration with the serial calibration experiment.

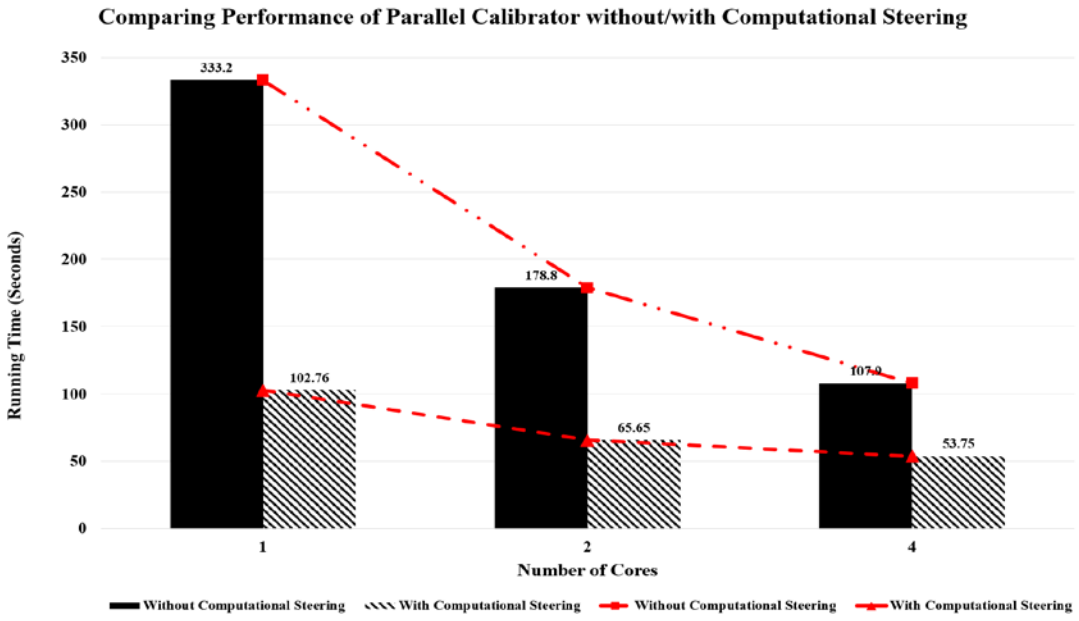


Figure 6-17 Comparing Performance of Parallel Steerable and Non-Steerable Calibration: The number of cores indicate the parallelism scale. For instance, by using two cores, we parallelise the calibration task on two worker processes by using MPI. This figure indicates, although computational steering can reduce the running time of the calibration, its improvement extent reduces as the parallelism scale increases.

The number of cores indicate the parallelism scale. For instance, by using two cores, we parallelise the calibration task on two worker processes by using MPI. In a conclusion, computational steering improves the time efficiency of the calibration dramatically in all cases. For parallelising the calibration in 2 cores, its running time is reduced from 178.8 seconds to 65.65 seconds, leading to a reduction of 63%. Additionally, by running the calibration on 4 cores, its running time drops from 107.9 to 53.75, leading to a reduction of 50%. However, in terms of comparing the advantage of parallel programming, it is noticeable that the steerable calibration is not improved on the same scale as the non-steerable calibration in parallel executes. Table 6-8 quantifies this discovery.

Improvement made by parallelizing Steerable and Non-Steerable Calibrations				
Core Number	**Steerable Calibrations**		**Non-Steerable Calibrations**	
	Running Time	Improvement Extent	Running Time	Improvement Extent
1	102.76	0%	333.02	0%
2	65.65	36%	178.78	46%
4	53.75	48%	107.99	68%

Table 6-8 Improvement made by parallelizing Steerable and Non-Steerable Calibrations: This table indicates the difference on the improvement extents between using and not using computational steering as the parallelism scale increases.

As indicated in Table 6-8, the steerable calibration has a 36% improvement compared with the 46% improvement of the non-steerable calibration by using 2 cores. Moreover, the improvement difference increases when running them on 4 cores in which the steerable calibration is increased by 48%. Instead, the non-steerable calibration has a 68% improvement on the running time.

The reason is that the parallel programming reduces the times a simulation needs to interact with the Calibrator by distributing simulation tasks to multiple simulations. As a result, the workload of one computing resource is reduced by the parallel programming. Since the execution time saved by using computational steering has a positive correlation with the workload of one computing resource, reducing the workload using parallel programming decreases the execution time that can be saved using computational steering. If we denote the time costs using computational steering as T_{ec} and the time saved by computational steering as T_{sc}, then, as T_{sc} decreases as the increase of N, it is reasonable to assume that computational steering can lose effectiveness when N reaches a specific value.

We then attempt to locate the cross-point at which the non-steerable calibration overpasses the steerable calibration. Equation (17) denotes the time function, $T(N)$, which suggests the running time of the calibration. The T_{serial} and $T_{parallel}$ are serial and parallel running time of the non-steerable and steerable calibrations collected in Table 6-8. In parallel programming, the T_e is the overhead that is raised by the non-parallel code and imperfect schedule. Since this experiment is conducted on a machine which only has 4 physical cores, the calibration parallelised on 4 cores must share the CPU time with other processes. Hence

the 2 cores parallel running time is used to calculate the T_e for both steerable and non-steerable calibration based on Equation (18).

$$T(N) = \frac{T_{serial}}{N} + T_e \tag{17}$$

$$T_e = T_{parallel} - \frac{T_{serial}}{N} \tag{18}$$

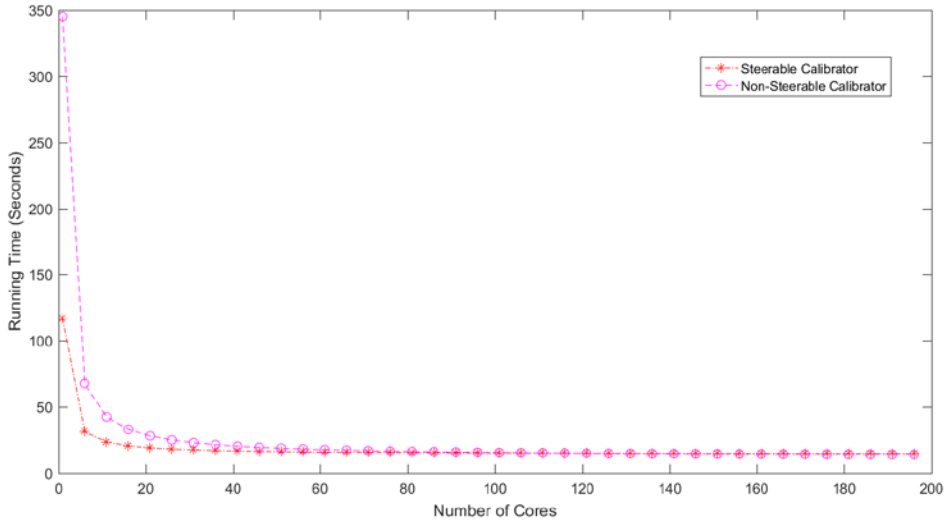


Figure 6-18 Comparing the Running Time between the Steerable Calibration and the Non-Steerable Calibration: As the parallelism scale increases, the speed of non-steerable calibration equals to that of steerable calibration when 37 core numbers are used for parallel programming.

As a result, Figure 6-18 shows the running time of steerable and non-steerable calibrations based on the Equation (17). Two lines cross around the point where *N* equals to 115.13. Since the core number is an integer, 116 is taken as the particular core number from which the non-steerable calibration is faster than the steerable calibration. However, the difference between two calibrations turns to be very small after 60 cores. If 20% is considered as a significant improvement, then based on Equation (17), the Inequality (19) must be met for the steerable calibration has a significant advantage compared with the non-steerable calibration. The T_{ens} and T_{es} denote parallel overhead of the non-steerable and steerable calibrations separately.

$$\frac{\frac{T_s}{N} + T_{es}}{\frac{T_{ns}}{N} + T_{ens}} < \frac{4}{5} \tag{19}$$

As a conclusion, the inequality is broken when N equals to 37. While the improvement made by computational steering is reduced by the increase of parallelism scale, it has a significant contribution until the parallel program scales up to 37 cores. At last, we use the ratio of task size to parallelism scale to indicate whether computational steering can make a significant improvement. As the population size is 240, the generation number is 10, and 37 cores are considered as the maximum parallelism scale, the ratio of this specific use case is 64.8. Thus, when the ratio is smaller than 64.8, we claim it is not feasible to use computational steering on our implementation of parallel calibration of the WDS. This ratio indicates if the interactions happened between the Calibrator and one simulation is less than 65, the computational steering may not provide sufficient values.

In a conclusion, this section indicates the DDDCS can improve the time efficiency of the calibration process. However, the parallelism scale can reduce the improvement made by computational steering. Whether computational steering can make a significant improvement depends on the problem size of the calibration, the parallelism scale, the implementation of the parallel program, and the computing infrastructures, etc. Thus, this section provides a method to determine whether DDDCS can make sufficient contribution to a system by calculating the ratio of parallelism scale and problem size.

The time management has not been evaluated on DDDCS since its evaluation relates to a complex hierarchy of knowledge. Hence, an individual section 6.5, is dedicated to discussing it. In order to demonstrate the time management of the hybrid computational steering, the time management of DDDCS is also combined with the time management of HDCS.

6.5 Evaluation of the Estimation of Execution Time

Section 6.4 has demonstrated that computational steering can improve the performance of the calibration of DDDAS. However, whether the improvement is significant enough to handle the dynamic workload so as to finish the calibration within the required time window needs to be further examined. This thesis proposes to utilize the Time Manager which estimates the running time of the calibration based on the dynamic data obtained from the physical world, and then according to the results of the estimation, the Computing Resources Manager assigns sufficient computing hardware to guarantee the execution time of the calibration. This section is the first part of the time management method which evaluates the function of estimating the required running time of the calibration.

Since the calibration in this use case is implemented by using the GA, this section takes the required execution time of the GA as that of the calibration process. Section 4.3.2 has discussed the reason for this thesis to only estimate the execution time of a Genetic Algorithm (GA), and the estimation method has been discussed in Section 4.3.3. Since the distribution of generation number is one main factor that indicates the execution time of a GA, this section evaluates our method that estimates the required generation numbers for the GA to calibrate the water model. The configurations of GA remain same as the GA used in Section 6.4, and the problem size of GA is divided into regular and abnormal situations as discussed in Section 6.2.2. Additionally, to demonstrate that the dynamic data collected from different situations can affect the time management, the artificial regular and abnormal situations are used in this evaluation.

At last, Section 6.5.1 examines the time management under the regular situation and Section 6.5.2 examines whether computational steering can help the Calibrator to handle the extra workload raised in an abnormal situation based on limited computing resources. Additionally, the relation between the number of generations and the Wall-Clock Time (WCT) are studied for each situation.

6.5.1 Estimating the Running time of the Calibrator in the Regular Situation

By fixing the configuration of the GA, the required execution time of GA depends on the number of generations it takes to converge. Hence, the estimation is divided into 1) estimation of the required number of generations and 2) mapping the generation number to the WCT. In this thesis, we assume that the number of generations fits into the Gaussian distribution and WCT has a linear relation with the number of generations. As a result, the first part of this evaluation is to support the use of Gaussian distributions to estimate the required number of generations. Additionally, we also consider the stopping criteria as a type of steerable parameters that can be steered by users. Thus, in regular situation, we also demonstrate the capability of estimating the execution time which is affected by both HDCS and DDDCS. The second part is to evaluate whether the relation between the number of generations and WCT can be obtained by using the linear regression.

Distribution of Generations

In the regular situation, the problem is divided into 5 groups in which the GA stops when the fitness value reaches 1000, 1500, 2000, 2500 and 3000. For each stopping criteria, the experiment is conducted 50 times, and Figure 6-19 depicts the distribution of required generation numbers.

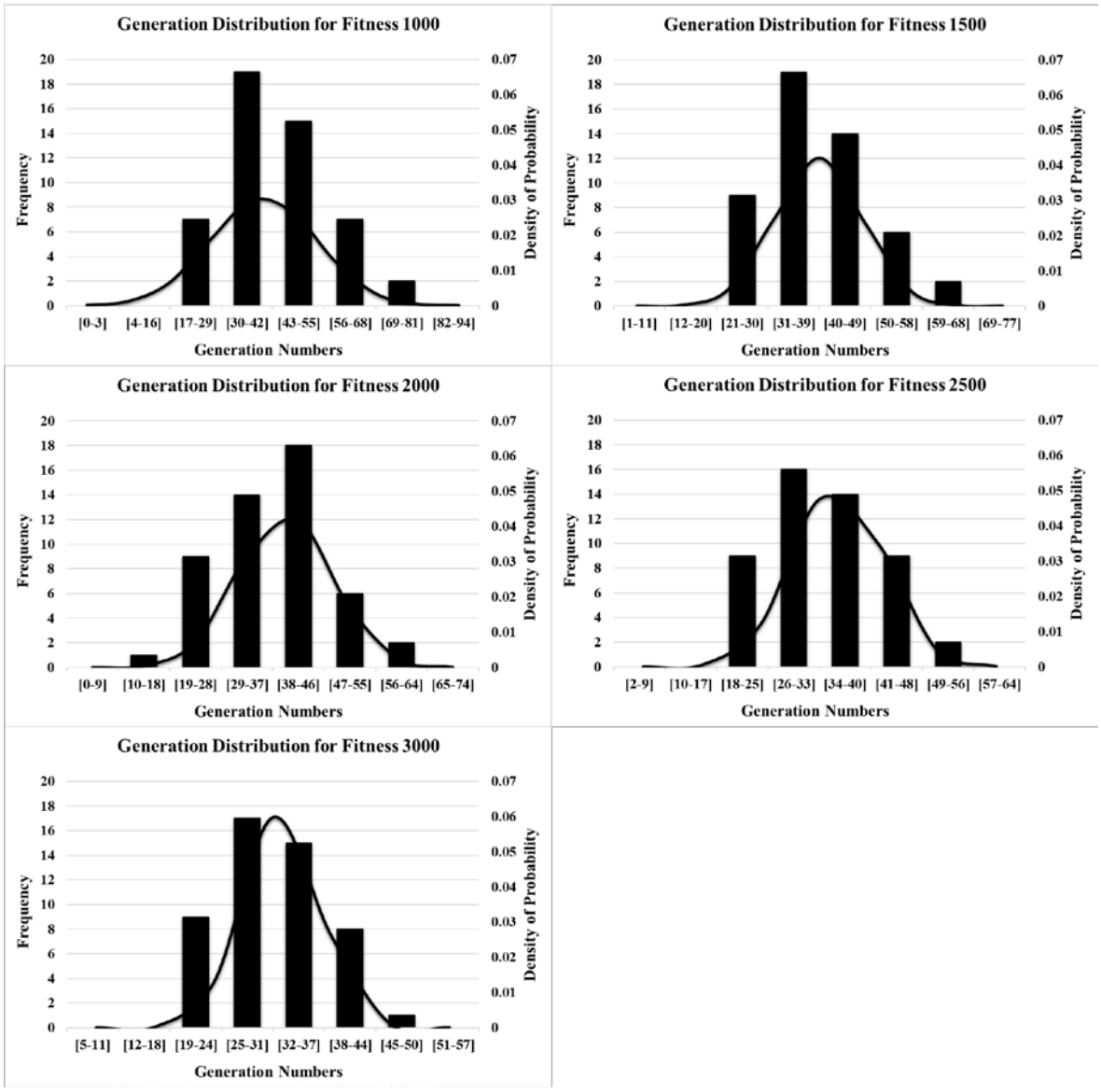


Figure 6-19 Distribution of Generation Numbers Using DDDCS These figures provide a visual inspection on the distribution of required number of generations for experiments running based on different stopping criteria. Reaching a specific fitness value (1000, 1500, 2000, 2500 and 3000) is set as the stopping criteria for each set of experiments. A normality test was conducted to further confirm the assumption of using Gaussian distribution. The result of the test is used to depict a linear curve that represents the density of probability of a Gaussian distribution in each subfigure.

Tests of Normality (Shapiro-Wilk)	
Stopping Criteria (Fitness Value)	p-values
1000	.62
1500	.19
2000	.42
2500	.10
3000	.85

Table 6-9 Normality Tests in Normal Situation: If p-values are greater than 0.05, then we claim the distribution of generations can be assumed as Gaussian distribution.

Based on Figure 6-19, we have some visual evidence that we are justified in the assumption that we can use the Gaussian distribution to estimate the required generations. As discussed in Section 6.3, visual inspection and numerical inspection are two methods can be used to identify the Gaussian distribution. Hence, the results are further examined by using Shapiro-Wilk as the numerical method. The IBM Statistical Package for the Social Sciences (SPSS) is utilised as a statistical tool to conduct this test, and the outcome is shown in Table 6-9. As all the significance values are greater than 0.05, we conclude that we can use Gaussian distribution to estimate the distribution of required number of generation. The shape of each Gaussian distribution is shown as a linear curve in Figure 6-19.

Information of the Gaussian Distributions		
Stopping Criteria	Standard Deviation	Mean
Fitness 1000	13.09	42.76
Fitness 1500	9.57	39.78
Fitness 2000	9.29	37.38
Fitness 2500	7.80	33.14
Fitness 3000	6.57	31.46

Table 6-10 Information of Gaussian Distributions: The decrease of the standard deviation means a strict stopping criteria increases the uncertainty in estimating the running time. The decrease of the mean indicates a strict stopping criteria increase the average running time.

The information of the Gaussian distribution is shown in Table 6-10. Both the standard deviation and mean decrease as the stopping fitness value increases. In terms of the mean values, its decrease suggests that the GA can converge faster by using a high fitness value as the stopping criteria. Since a high fitness value indicates a more inaccurate calibration, this result presents the trade-off between running time and calibration accuracy. Moreover, the decrease of the standard deviation reveals that a more restrictive stopping criteria or a high requirement of the calibration accuracy can lead to a more unstable distribution of the required generations numbers. Thus, another trade-off is between the estimation accuracy and the calibration accuracy. The estimation accuracy indicates the capability of the time management component to estimate the required running time of the calibration process. Due to the time constraint required in a DDDAS, this project applies dynamic computing resources to speed-up the calibration if the estimation predicts the calibration will exceed the deadline. However, a lower estimation accuracy can lead to a waste or deficit of computing resources. Subsequently, the estimation accuracy turns to be the cost efficiency of utilising the purchased dynamic computing resources. Finally, according to the information of Gaussian distributions shown in Table 6-10 and the Equation (16) discussed in Section 6.3.2, the estimation functions are generated in Table 6-11.

Functions of the Running Time Estimation		
Stopping Criteria	*Generations Estimation*	*Estimated Generation Number when* $n = 1$
Fitness 1000	$E_g = \lfloor n * 13.09 + 42.76 \rfloor$	56
Fitness 1500	$E_g = \lfloor n * 9.57 + 39.78 \rfloor$	50
Fitness 2000	$E_g = \lfloor n * 9.29 + 37.38 \rfloor$	47
Fitness 2500	$E_g = \lfloor n * 7.80 + 33.14 \rfloor$	41
Fitness 3000	$E_g = \lfloor n * 6.57 + 31.46 \rfloor$	38

Table 6-11 Functions of the Estimating Required Number of Generations: It provides the result of estimating number of generations in the form of estimation functions. The variable in the function is n, it indicates the accuracy level of an estimation. A high value of n can increase the estimation accuracy but cause great

computing resource waste. The column on the right shows the estimated number of generations required by a GA when n=1.

Since the generation number cannot be a float number, the next integer is used to represent the required generation number. This thesis takes $n = 1$ in this evaluation since it covers 84.2% possible situations. As a result, the estimated required generation numbers based on the tested stopping criteria are shown on the right column of Table 6-11.

However, in practical applications of a WDS, the time constraint is defined by using the WCT instead of the generation number. Hence, relating the WCT to the required generation of GA is essential.

Relation Between Generation Number and Wall-Clock Time

By keeping the problem size and GA configurations consistent, the required WCT of the GA depends only on the specifications of computing resources. In this experiment a MacBook Pro Late 2013 version is used to execute the calibration. It has a 2GHz Intel Core i7-4750HQ Crystal Well processor with 6 MB on-chip L3 and 128MB L4 cache. The Intel Core i7-4750HQ has 4 cores physically and 8 threads so that it at most supports 32 threads in parallel programming with the limitation that the performance improvement is restricted by the physical core number. The main purpose of this section is to explore the method that can define the relation between the generation number and the wall-clock time. Hence, it is reasonable to use the local laptop to conduct the experiment. Afterwards, Section 6.6 will take this verified method to define such relation on Amazon Cloud, which is used as the example of the external computing resource provider in this project.

As discussed in Section 4.2.2, the variation of the execution time of a simulation can depend on the number of iterations of its main computation. In the EPANET, this iteration is to search for the best hydraulic solution. This thesis uses the default iteration number, 40, which is provided by the EPANET software as the static number of iterations. Therefore, the execution time of simulations are assumed to be stable. Under this assumption, the relation between the WCT and the number of generations required by the GA should be in the form of a linear equation in which the generation number should be the only variable. Additionally, except for stopping criteria, the GA configurations of experiments are same. Thus, all experiments are supposed to share the same relations between generation numbers and the WCT. Finally, 20 pairs of these relations are selected from above experiments. The running time is collected by using C++ time() function that calculates the difference between the

beginning and ending calendar time of the calibration, and the results are plotted on Figure 6-20.

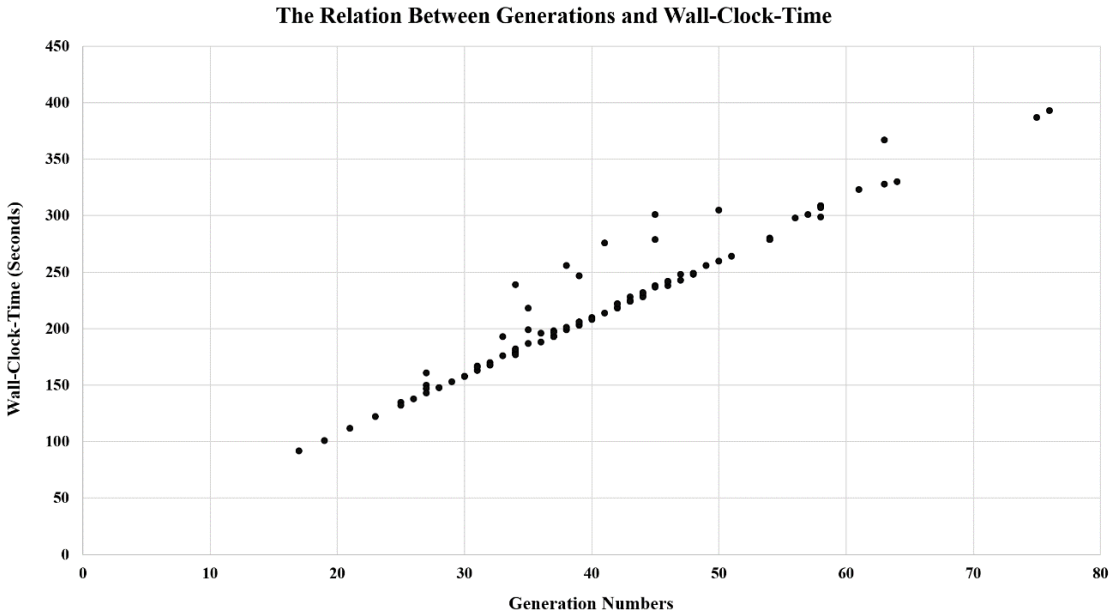


Figure 6-20 Relation between Generations and WCT in Regular Situations: This figure provides a visual inspection on the linear relation between the generation number and the WCT.

Based on Figure 6-20, this section further examines the mathematical relation between generations and the WCT by using the linearity test function provided by IBM SPSS. The results are shown in Table 6-12. It predicts a linear regression equation and test how well the equation fits the data. The null-hypothesis is that the predicted model does not fit the data, and the p-value indicates the probability of finding a data that cannot be fitted into the predicted equation. If the p-value is less than 0.05, we claim that the null-hypothesis is rejected and we can assume it is a linear relation.

Tests of Linearity		
p-value	**Constant**	**Coefficient**
0.00	9.32	5.14

Table 6-12 Test of Linearity in Regular Situation on Local Machine: If the p-value is less than 0.05, we can assume the relation between number of generations and WCT is linear.

Moreover, Table 6-12 shows the coefficients of the result linear function. Based on the values of Constant and Coefficient, the relation between generation numbers and the WCT in can be represented as:

$$WCT = 5.140 * Gen + 9.322\ (seconds) \quad (20)$$

The WCT denotes the required Wall-Clock Time and the Gen signifies the required generation number. Finally, the required WCT is shown in Table 6-13.

Required Wall Clock Time			
Stopping Criteria	Generations Estimation	$n = 1$	WCT(seconds)
Fitness 1000	$E_g = \lfloor n * 13.09 + 42.76 \rfloor$	56	297.16
Fitness 1500	$E_g = \lfloor n * 9.57 + 39.78 \rfloor$	50	266.32
Fitness 2000	$E_g = \lfloor n * 9.29 + 37.38 \rfloor$	47	250.90
Fitness 2500	$E_g = \lfloor n * 7.80 + 33.14 \rfloor$	41	220.06
Fitness 3000	$E_g = \lfloor n * 6.57 + 31.46 \rfloor$	38	204.64

Table 6-13 Required Wall Clock Time: It provides the result of estimating running time based on the estimated number of generations. According to the linear relation between the generation number and the wall-clock time, the required running time is shown in the last column in seconds.

By considering 15 minutes as the time constraint, the calibrations under examined stopping criteria can be finished in time, specifically, in 5 minutes. In a conclusion, by running the calibration in parallel with 4 processes, the GA can manage to converge in 5 minutes in the normal situation. However, as the system is considered to be steered by the dynamic sensor data, we further evaluate the running time estimation method in the abnormal situation.

6.5.2 Estimating the Running time of the Calibrator on an Abnormal Situation

Section 6.2.2 has introduced the abnormal situation which is artificially generated based on a real pipe burst sensor readings. The significant variation of sensor readings that raised by the pipe burst considerably increases the task size of the calibration. Hence, by demonstrating that this project is able to estimate the running time of the Calibrator in an anomalous situation, it is important to evaluate whether implementing steering on the calibration component is useful enough to finish extreme tasks in the WDS use case.

The experiment is divided into two parts based on the methods evaluated in Section 6.5.1. Firstly, we estimated the required generation numbers. Afterwards, we estimated the relation

between the generation number and the WCT. Based on this relation, we calculated the required WCT for the calibration to finish on a specific computing environment and in the abnormal situation.

Distribution of Generations

The first step of estimating the running time is to analyse the distribution of required generation numbers. Hence, the calibration is executed 50 times to calibrate the water model in the artificial pipe burst. Since this section does not focus on demonstrating the hybrid steering ability, the calibration is stopped when its fitness value reaches 1000. As the feasibility of using Gaussian distribution to estimate the distribution of generation numbers has been discussed in Section 6.3 and 6.5, this section does not repeat the normality test. The information of the estimated Gaussian distribution are shown in Table 6-14, and the function used to estimate the required generations is shown in Table 6-15.

Information of the Gaussian Distribution	
Mean	Standard Deviation
105.60	27.20

Table 6-14 Information of the Gaussian Distribution in Pipe Burst

Stopping Criteria	***Generations Estimation***	$\boldsymbol{n = 1}$ ***(generations)***
Fitness 1000	$E_g = \lfloor n * 27.2 + 105.6 \rfloor$	133

Table 6-15 Estimation of the Required Generations: When taking n as 1, the required generation number is estimated as 133.

Compared with the distribution of generation numbers in the regular situation, the mean value of the Gaussian distribution increases from 42.8 to 105.6. It indicates the generation amount required by the Calibrator increased to more than twice as compared with regular situation. Additionally, the standard deviation in abnormal situation is greater than normal situation. It suggests when the problem size increases, distribution of required generations become dispersed, and the time management component may apply for more redundant computing resources.

Relation Between Generation Number and Wall-Clock Time

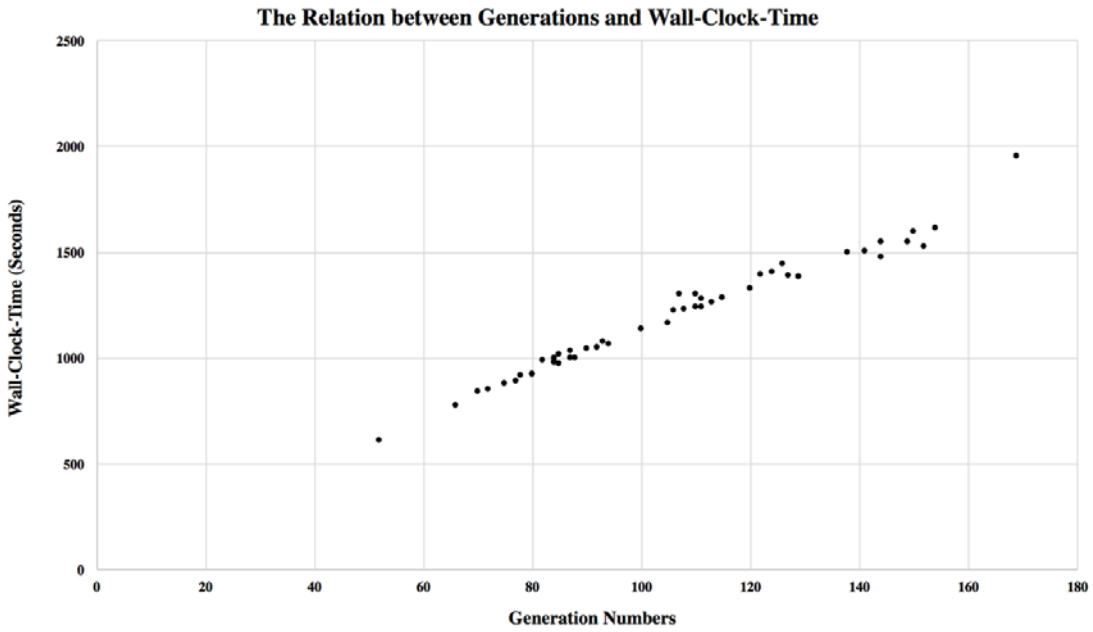


Figure 6-21 Relation between Generations and Wall-Clock Time in a Pipe Burst: This figure provides a visual inspection on the linear relation between the generation number and the WCT.

After understanding the distribution of required generations, we further study the correlation between the generation number and its required WCT. The correlation is depicted in Figure 6-21.

Tests of Linearity		
p-value	**Constant**	**Coefficient**
0.00	161.99	9.71

Table 6-16 Linearity Test of Relation between Generations and Wall-Clock Time in a Pipe Burst: If the p-value is less than 0.05, we can assume the relation between number of generations and WCT is linear.

The linearity test method introduced in Section 6.5.1 is utilised to test the linearity in this abnormal situation, and the results are shown in Table 6-16. The Significance of ANOVA is closed to 0.00 which means all data sets fit well with a linear function as described by Equation (21).

$$WCT = 9.719 * Gen + 161.99\ (seconds) \qquad (21)$$

By using 133 as the generation number, the estimated WCT is about 1454.617 seconds (about 24 minutes). Since the time constraint is 15 minutes, we can conclude that without a

dedicated study of the required running time of the calibration, it is possible for an existing monitoring system to have a delayed update when an abnormal situation happens. Moreover, our time management method manages to estimate the required execution time of the calibration process in a pipe burst event.

6.6 Evaluation of Using Dynamic Computing Resources

In Section 6.5, it demonstrates that, with static computing power, the calibration can run out of time even its running time has been reduced using computational steering. Therefore, this section evaluates the capability of using dynamic computing resources. Additionally, the cloud based computational steering architecture introduced in Section 5.3 is utilised and the Amazon EC2 is used as the cloud service provider.

To demonstrate that the cloud computing can provide enough computing resources for the calibration, the distribution of generation numbers and calibration speed of the GA need to be studied specifically for this new infrastructure. Moreover, the efficiency of the parallel programming also needs to be reconsidered since in this evaluation, the network communication is between remote instances distributed in the Europe. The instance used in this experiment runs with 64bit Ubuntu Server 14.04 LTS, and has one CPU and one gigabyte memory. The number of instances are considered as the amount of computing resources in this evaluation.

Additionally, the artificial pipe burst is used since it raises a more complex calibration task than the normal situation. This section assumes that an artificial pipe burst situation can generate the upper limit of the calibration workload. Hence, as long as the calibration can be finished in time under this abnormal situation, it supports that the method proposed by this thesis manages to meet the time requirement in the WDS. Finally, the configurations of the Genetic Algorithm (GA) is kept same as the experiment discussed in Section6.5.

6.6.1 Generation Distribution

The number of generations required by the GA to tackle a problem depends on selection, mutation and crossover, and it also depends on parameters such as the population size and stopping criteria. Therefore, when the configurations of the GA and the problem size are identical, the distribution of required generations should be similar. However, in realisation, specific functions, such as selection of random numbers, can make influence on the GA.

Therefore, this section aims at analysing the generation distribution on instances provided by Amazon Cloud.

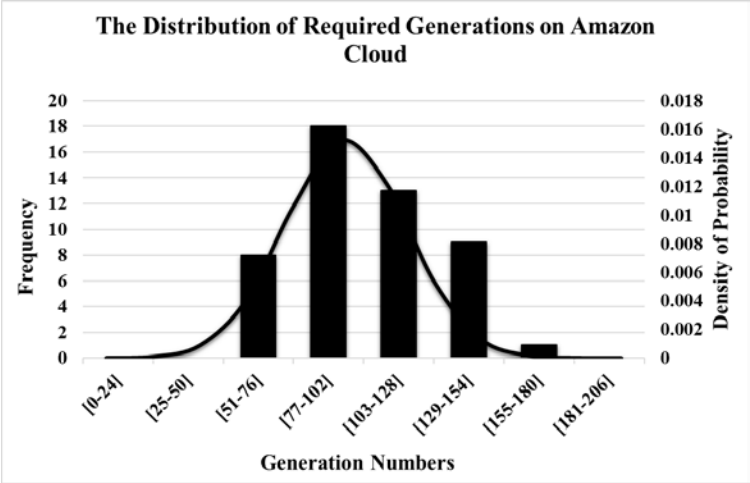


Figure 6-22 The Distribution of Required Generations on Amazon Cloud: This figure provides a visual inspection on the distribution of required number of generations. A normality test was conducted to further confirm the assumption of using Gaussian distribution. The result of the test is used to depict a linear curve that represents the density of probability of a Gaussian distribution in each subfigure.

The calibration process is executed 50 times in this experiment, and their required generation numbers are analysed by the IBM SPSS. Figure 6-22 depicts the distribution histogram of the generation number, and Table 6-17 shows the results of the normality test. Moreover, the normality test conducted for the calibration on the local machine is also listed in Table 6-17 to make comparison.

Tests of Normality			
Platforms	p-values	Mean	Std. Deviation
Amazon EC2	.205	102.24	26.33
Local MacBook	.108	105.66	27.23

Table 6-17 Normality Test of Generation Distribution on Amazon EC2: If p-values are greater than 0.05, then we claim the distribution of generations can be assumed as Gaussian distribution.

Since the significance of Shapiro-Wilk test is 0.22, which is greater than 0.05. Hence we claim that the distribution of generation number can fit into the normal distribution. In the comparison with the distribution on the local machine, the difference between mean values is less than 3 and the difference between the standard deviation is around 1. Hence, the difference of generation distribution caused by the different hardware is not significant.

Moreover, by using Equation (16) with $n = 1$, the estimated generation number required to calibrate this pipe burst anomaly is 129. In order to obtain the required Wall-Clock Time

(WCT), we still need to obtain the relation between generation number and WCT. Since this relation depends on the hardware specifications, the next section estimates the running time of the calibration with different instance numbers.

6.6.2 Relation Between Generation Number and the Wall-Clock Running Time

Four sets of experiment have been conducted to collect the WCT. Each set consists of 10 executions of the calibration and they are executed separately on 1, 4, 8, 12 and 16 instances. As a result, the consumed WCT and the corresponding generation numbers are plotted from Figure 6-23 to Figure 6-27.

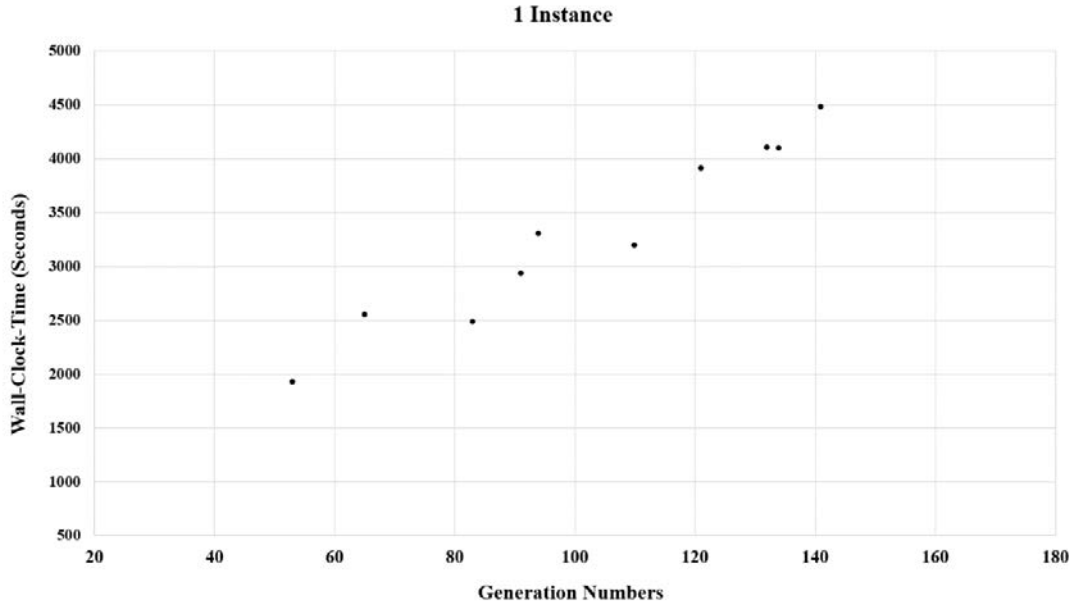


Figure 6-23 The Relation between WCT and Number of Generations – 1 Instance: This figure provides a visual inspection on the linear relation between the generation number and the WCT.

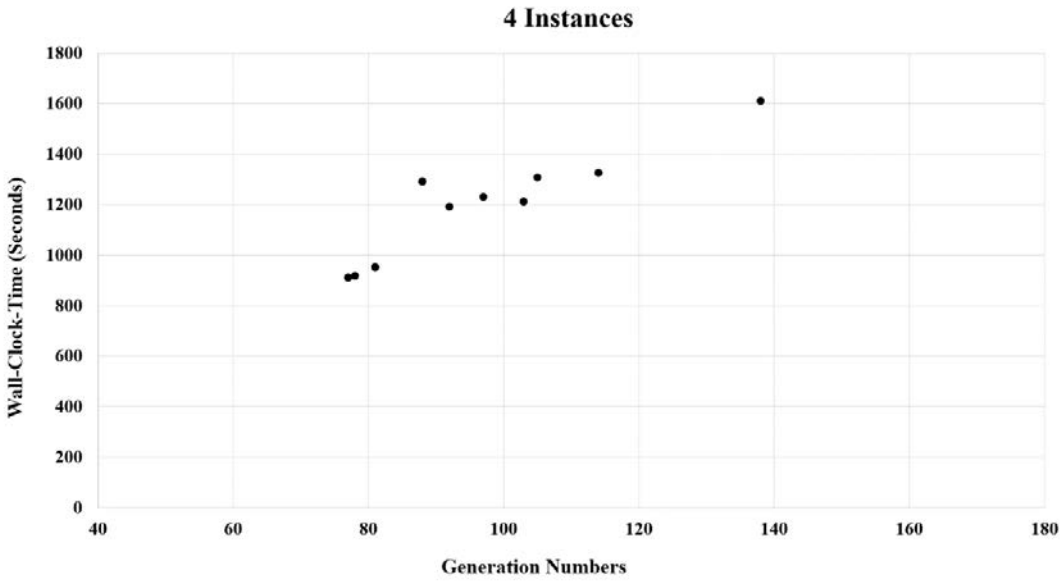


Figure 6-24 The Relation between WCT and Number of Generations – 4 Instances: This figure provides a visual inspection on the linear relation between the generation number and the WCT.

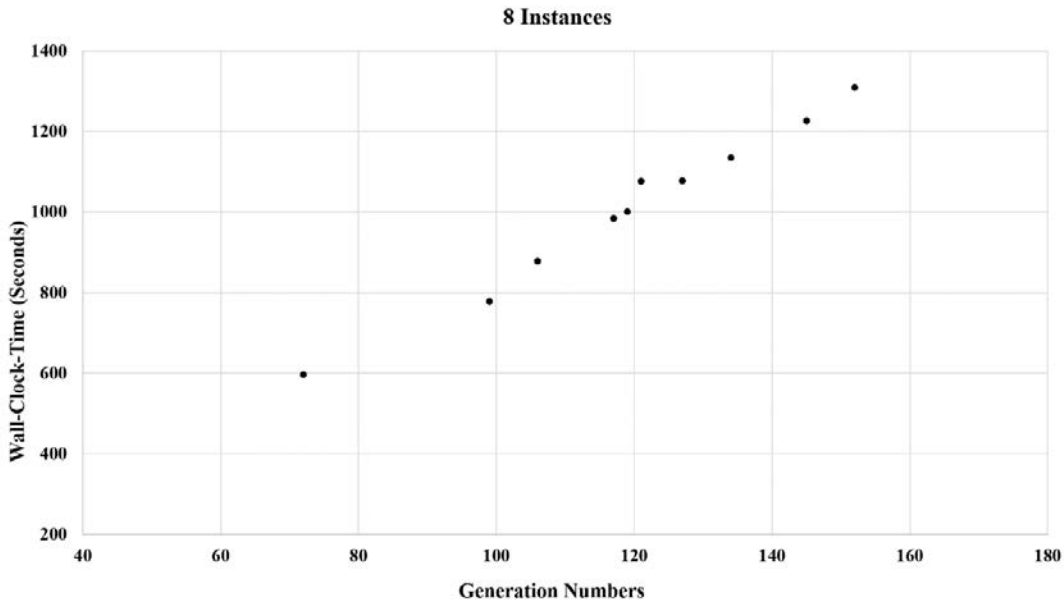


Figure 6-25 The Relation between WCT and Number of Generations – 8 Instances: This figure provides a visual inspection on the linear relation between the generation number and the WCT.

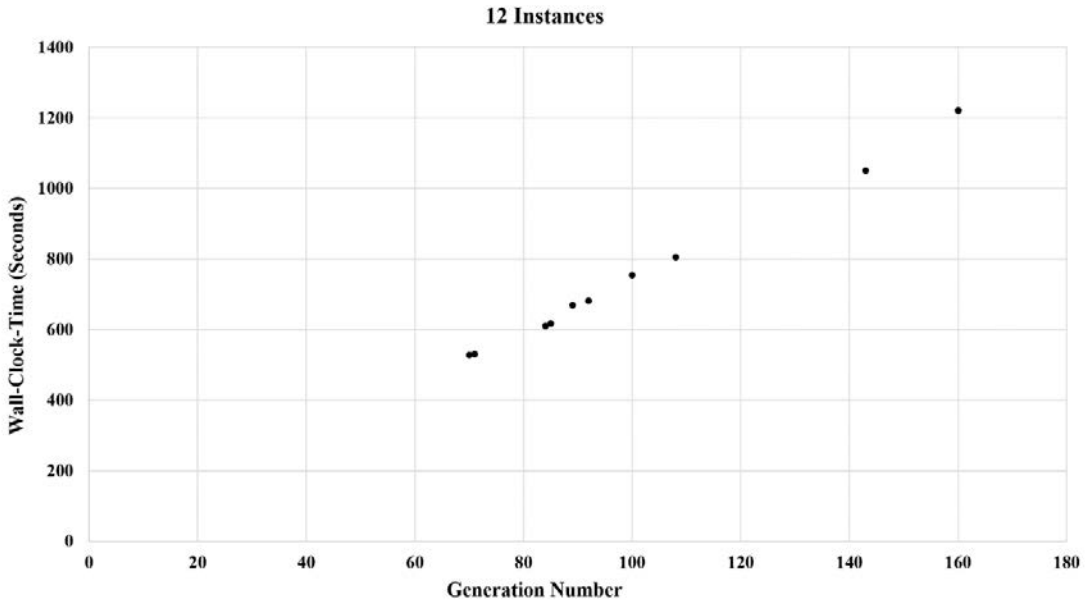


Figure 6-26 The Relation between WCT and Number of Generations – 12 Instances: This figure provides a visual inspection on the linear relation between the generation number and the WCT.

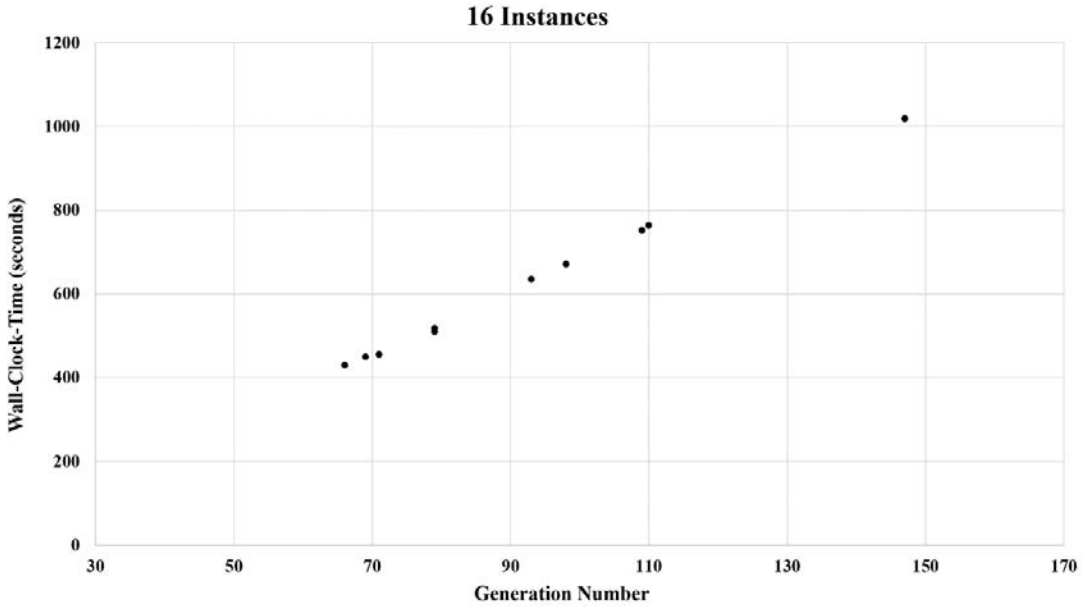


Figure 6-27 The Relation between WCT and Number of Generations – 16 Instances: This figure provides a visual inspection on the linear relation between the generation number and the WCT.

By observing the figures, we assume the relation between the WCT and the number of generations can still be linear. Hence, the IBM SPSS is used to conduct linearity test, and the results are shown in Table 6-18.

Tests of Linearity			
Number of Instances	**Sig. of ANOVA**	**Constants and Coefficients**	
		Constants	Coefficients
1	≈0.00	529.19	27.03
4	≈0.00	134.87	10.87
8	≈0.00	-75.17	9.07
12	≈0.00	-16.51	7.61
16	≈0.00	-65.43	7.45

Table 6-18 Linearity Tests of Relation between WCT and Generation Number on Amazon Cloud: If the p-value is less than 0.05, we can assume the relation between number of generations and WCT is linear.

Because all significance value of ANOVA equal to 0.00, it indicates that every relation shown in Figure 6-27 fits well in the linear functions with the coefficients shown on under "Coefficients" column of Table 6-18. Afterwards, the linear functions that represent these relations are shown in Table 6-19.

Relations of WCT and Generation Number			
Number of Instances	**Functions**	$Gen_{Num} = 129$	
		Seconds	**Minutes**
1	$WCT = 27.037 * Gen + 529.196$ (22)	4016.97	66.95
4	$WCT = 10.873 * Gen + 134.875$ (23)	1537.49	25.62
8	$WCT = 9.075 * Gen - 75.175$ (24)	1095.50	18.26
12	$WCT = 7.618 * Gen - 16.51$ (25)	966.21	16.10
16	$WCT = 7.450 * Gen - 65.437$ (26)	895.61	14.93

Table 6-19 Relations of Wall-Clock Time and Generation Number on Amazon Cloud Computing Environment: A function indicates the relation between the WCT and the required number of generations when a specific number of instances are used for the parallel calibration. By taking 129 as the estimated required number of generations, the estimated required running time is shown on the last two columns in seconds and minutes.

As a result, by using 16 instances, the running time is reduced from 67 minutes to less than 15 minutes. As the time constraint in this use case is set as 15 minutes, we conclude that our time management component can successfully guarantee the time constraint by dynamically assign cloud computing resources to the calibration process even when an abnormal situation creates great workload for it.

6.6.3 Calibration Speed – Seconds per Generation

To evaluate the efficiency of parallel programming, functions shown in Table 6-19 are plotted in Figure 6-28.

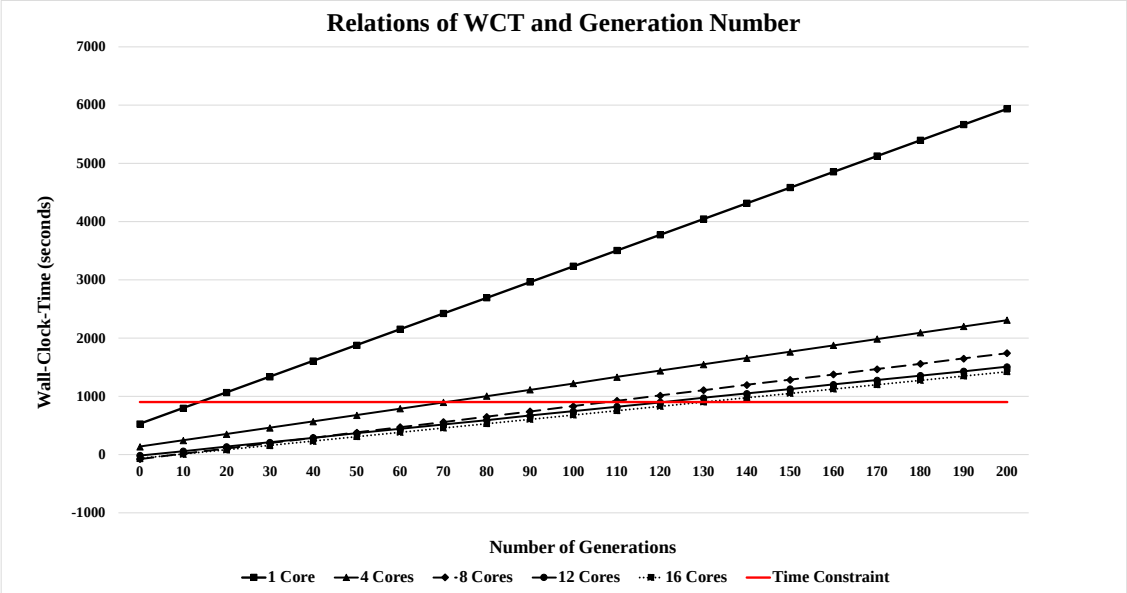


Figure 6-28 Relations of WCT and Generation Number: The x axis is the number of generations required to finish the calibration. The y axis is the time for running the amount of generations in seconds. This figure indicates the linear relation between the WCT and the number of generations when the calibration is run with 1 , 4, 8, 12 and 16 instances(s). It shows when more instances are used, the linear relation has a lower slope. Since the problem size studied in this thesis generally require more than 60 generations, this figure indicates the calibration running on a greater number of instances can always be faster. However, by taking the cost of computing resources into consideration, as long as the time constraint is met, a smaller number of instances can be used to save cost. The red line indicates the 15 minutes time constraint which is used in this use case.

Each line shown in Figure 6-28 corresponds to a function shown in Table 6-19. As instance numbers increase, the difference between adjacent slopes decreases. After the instance number reaches 8, the relation functions begin to have crossings. Specifically, the function of 8 instances crosses with the function of 12 instances and 16 instances when the number of generations equal to 40.3 and 5.9. It indicates that when the calibration task is simple and requires less than 41 generations, using 8 instances is better than using 12 instances. Nevertheless, for the specific problem size discussed in this section, the generation number required by the GA for the abnormal situation can reach 129. Therefore, increasing the

instance of numbers can manage to handle the time requirement in this pipe burst situation. For regular situations whose estimated generation numbers are less than 41, 2 and 4 instances can be an efficient choice.

Due to the 15 minutes time constraint, the Figure 6-28 also indicates the suitable instance numbers for different problem sizes that may have different generation distributions. The results of suitable core number and its corresponding generation number is listed in Table 6-20.

Intervals of Required Number of Generations	**Instance Numbers**
[0,13]	1
[14,70]	4
[71,107]	8
[107,120]	12
[120,129]	16

Table 6-20 Generation Distribution Intervals and Suitable Instance Numbers to Tackle Its Corresponding Problem: This table provides an example of a training result of the computing resources manager. It indicates the number of instances needs to be assigned to the calibration process when the estimated number of required generations fall into the intervals indicated on the left column.

As the improvement made by the parallel programming decreases as the increase of the instance number, the interval size that a particular instance number can cope with drops as the increase of instance number as well. Since the artificial pipe burst used in this experiment is considered to be an extreme problem size in the WDS and a restricted stopping criteria (when fitness reaches 1000) is applied to the GA, the estimated generation number required by the Calibrator is assumed to be the maximum in the WDS use case. Therefore, we conclude that by using 16 instances, most problems can be guaranteed to finish in the required time window. Moreover, based on requirements of users, a smaller problem sizes in an abnormal situation can be created by steering parameters such as the stopping criteria. According to the required generation number estimated by analyzing the problem size, a suitable core number can be selected based on Table 6-20.

6.7 Chapter Summary

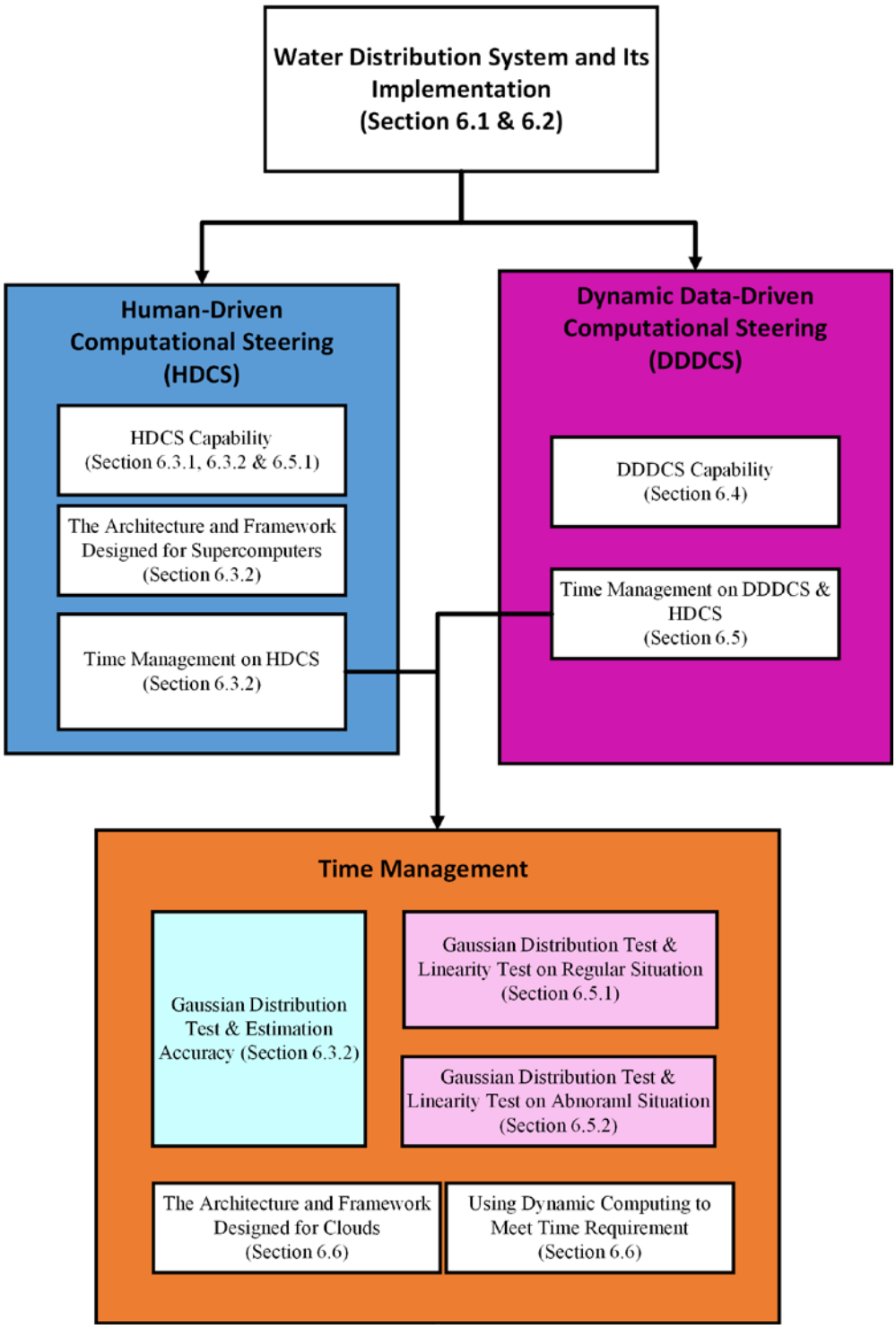


Figure 6-29 Structure of the Evaluation Chapter: This chapter begins with an introduction of the Water Distribution System which is used as the use case in this evaluation. To evaluate functions, architectures and frameworks of the hybrid computational steering, this chapter is mainly divided as evaluations on HDCS and DDDCS. Additionally, as the time management is a complex function for both HDCS and DDDCS, it is emphasised in this figure as a separate component.

Since this chapter has a complex structure, Figure 6-29 summaries its structure. This section is divided into four components. The first component is the Water Distribution System and Its Implementation. It introduces a water distribution system which is used as the use case in this evaluation. Based on the calibration process discussed in 4.2, it further introduces our methods to implement this use case for the purpose of demonstrating functions and time management methods of the hybrid computational steering.

Afterwards, the evaluation is separated as evaluation on Human-Driven Computational Steering (HDCS) and evaluation on Dynamic Data-Driven Computational Steering. For the HDCS, we evaluate its steering capability by enabling human users to steer number of sensors that are used in the calibration process, which is realised by using a genetic algorithm. The results presented that the accuracy and execution time of the calibration process can be steered by users successfully. Since other high-level functions can be based on the speed and accuracy of the calibration, we claim that HDCS can incorporate human users in the workflow of DDDAS as introduced in Section 3.1.1. Additionally, as the machine that are used to support this evaluation is a local cluster, the architecture and framework designed for supercomputers are used to conduct the experiment. Thus, this evaluation also supports our design in Section 4.2 and 5.2. Moreover, the generation number required by a GA, which can be affected by HDCS, is estimated by using time management methods discussed in Section 4.3. It supports our assumption that the distribution of required generation number can be assumed as a Gaussian distribution. Finally, it presents that, based on the estimation results, about 85% of calibrations can have enough time to be finished. Since this thesis considers this estimation method as an approach to demonstrate functions of a hybrid computational steering, even though this estimation method might wastes some computing resources, we think it can be used as an example of time management method that can be improved in future work.

For the DDDCS, its steering capability is presented on the reduced execution time of the calibration in Section 6.4. By comparing with a non-steerable calibration, the DDDCS can greatly increase the execution speed of the calibration process, which is realised by using a genetic algorithm. However, the parallel programming can weaken the improvement effect made by DDDCS since it reduces interactions between calibrators and simulations. Thus, we provide a method for users to measure whether it is effective to use DDDCS. As the result of the measurement, we claim the for this specific use case and its problem sizes, DDDCS can always improve the efficiency of the calibration process. The time management

of DDDCS is introduced in Section 6.5 with the time management of HDCS. In section 6.5.1, the stopping criteria is considered as steerable parameter of HDCS, and its effects on the time management is considered. In addition to estimate the required number of generations, This time management also supports our assumption in which the relation between the required number of generations and wall-clock time is linear. Based on the equation obtained from a linear regression method, the time management can estimate the required running time of the calibration process based on a regular situation in the WDS. To further evaluate the time management, the variation of the dynamic data is represented by using an abnormal situation. The results of the time management conducted for the abnormal situation is compared with the estimated execution time obtained from the regular situation. The results show that the time management method can manage to reflect the effects taken by the dynamic data of DDDCS.

At last, the Amazon cloud is used as the computing infrastructure to evaluate whether using dynamic computing resources can meet the time requirement raised by DDDAS in Section 6.6. By using the abnormal situation with a restricted stopping criteria, the estimated execution time of the calibration is successfully reduced under 15 minutes. Hence, we conclude that assigning dynamic computing resources to the calibration, which is steered by HDCS and DDDCS, can help to meet the time requirement. As the Amazon cloud is used in this evaluation, it also demonstrates the value of the architecture and framework designed for cloud computing. Part of the results shown in this chapter has been demonstrated and discussed with water engineers and scientists, their feedback has been recorded and listed in Appendix C.

Chapter 7 Conclusions

We began this thesis by proposing a central hypothesis that computational steering can be integrated with Dynamic Data-Driven Application System (DDDAS) to improve the speed of data assimilation algorithms and to help incorporating human intelligence into workflow of DDDAS applications. To establish this hypothesis, we presented the theory, architecture and frameworks of hybrid computational steering in the course of this thesis. The achieved hybrid computational steering is the result of a set of objectives we set up. This section concludes our contributions by following the sequence of set-up objectives. To help readers to have an overview of this thesis, Figure 1-2 presented in Section 1.4 was re-presented as Figure 7-1 in this section.

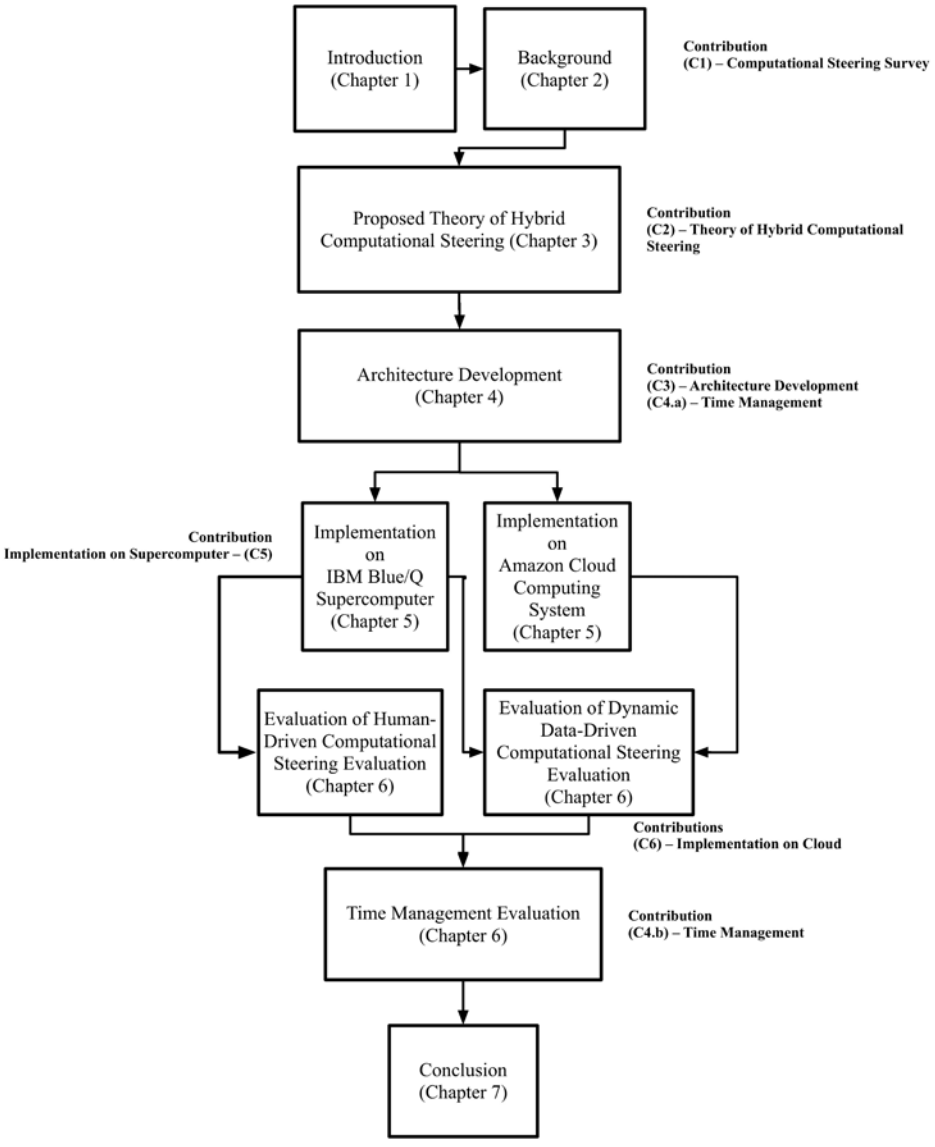


Figure 7-1 Structure of the Thesis Presented

7.1 Summary of Research Contributions

First, a statistical analysis of the impact of computational steering is conducted; based on this analysis, the reasons for utilizing applications of DDDAS as the future application area of computational steering is raised (Objective O1). The impact of computational steering is analysed based on the number of published papers relating to computational steering and the citation numbers of these papers. In addition, the published papers are categorized to further examine relations between computational steering and technologies arising together with computational steering; see Chapter 2.

Contribution C1: A statistical analysis of impact of computational steering. Computational steering has been studied for more than two decades. Several reviews were conducted in order to understand its concepts, taxonomy and involved projects. However, few reviews were conducted in the past ten years, and to the author's knowledge, no existing review aims for a historical and statistical analysis. Based on our statistical analysis, we conclude that the development of computational steering has always been bound with other popular topics such as High-Performance Computing (HPC), visualizations and grid computing. As a coin has two sides, development of computational steering is motivated but also limited by these related areas. In the case of the development of grid computing, the negative correlation between number of published papers and number of citations these paper obtained indicates the arising grid motivates the development of computational steering. However, the motivation did not turn to widely-accepted utilization in practice. Through our analysis, uncertain future of alternating protocols and standardization utilized by the grid can impede its integration with computational steering. Additionally, since development of computational steering has always been related to developments of other areas, we assume that, instead of introducing new computing and visualization technologies to the computational steering, finding a new and arising application area for the computational steering can facilitate its development as well.

Depending on the review in Chapter 2, we proposed the hybrid computational steering as an approach to facilitate the development of DDDAS. Hence, DDDAS is considered as a new application area for the computational steering. Additionally, the hybrid computational steering is divided into Human-Driven Computational Steering (HDCS) and Dynamic Data-Driven Computational Steering (DDDCS). Each sub-steering is proposed by discussing its aims and usages with respect to DDDAS. Following the discussions, their conceptual

components are defined in terms of the steerer, steering targets; and workflows are also designed to indicate relations and communications among conceptual components. Consequently, by forming conceptual steering components and workflows, we defined the proposed hybrid computational steering for DDDAS in Chapter 3 (Objective O2).

Contribution C2: A definition of hybrid computational steering and an understanding of advantage of computational steering from a new aspect. As applications of DDDAS are selected by this thesis as a new application area of computational steering, we contribute a hybrid computational steering that can meet requirements of DDDAS applications. In the HDCS, instead of exploring parameter spaces conducted in conventional computational steering, users steer parameters of algorithms utilized in DDDAS in order to indirectly affect simulations. In addition, we introduce our understanding of computational steering from a new aspect in which, the advantage of computational steering can be seen as reducing the length of feedback loop between components. This advantage was mentioned by computational steering developed at the early stage, but was neglected in the following development. Based on this understanding, we propose the DDDCS in which interactions between Genetic Algorithm (GA) and simulations used in calibrations are facilitated. The genetic algorithm is used as an example of calibration algorithms. Even calibration algorithms have a wide scope, we argue that the genetic algorithm can represent a number of popular calibration algorithms that were used in recent works, and we conclude that the hybrid computational steering proposed for DDDAS has a certain extent universality.

To demonstrate the hybrid computational steering, we must design a new architecture to support it (Objective O3). Based on the conceptual components developed in Chapter 3, the architecture is introduced in Chapter 4, and it is divided into two layers, namely HDCS layer and DDDCS layer. The design of HDCS architecture is based on architecture of conventional computational steering. Additionally, DDDCS layer is further divided into software level and hardware level. The architecture designed for the software level is an integration of new steering components proposed for DDDAS applications with the workflow of conventional computational steering. The hardware level architecture is designed as the result of taking time management, which is required by DDDAS, into consideration.

Contribution C3 and C4.a: The architecture designed for the hybrid computational steering and the introduction of time management methods of Workflow Management System (WMS). The design of architectures of HDCS and DDDCS takes the difference between the

hybrid computational steering and conventional computational steering into consideration. For HDCS, computational steering interfaces are introduced in calibration algorithms, which is the steering target of HDCS. The visualization components of conventional computational steering are changed as interfaces for users to realise their high-level functions, such as the monitoring and prediction. For DDDCS, steering interfaces are introduced in the algorithmic steerer and simulations used in calibration. The visualization component of conventional computational steering is replaced by evaluation functions in calibration algorithms. The algorithmic steerer is also enabled to steer simulations in parallel. Additionally, time management methods such as deadline assignment and running time estimation of WMS are introduced to assure components of designed architectures can meet time requirements in DDDAS. The time management methods are provided to handle flexible requirements of users in the future. However, in our current implementation, the calibration process consumes predominant time among components designed in the hybrid computational steering architecture. Hence, only its execution time needs to be estimated. As GA is used to realize the calibration algorithm, we then provide a classification tree to estimate required execution time of GA. The prediction tree builds a relation between amount of computing resources and required execution time based on the task size indicated by the dynamic data. Hence, it can help to ensure the calibration can be finished in time. As for GA has uncertainties on required execution time for the same problem size, the required execution time is further indicated by distributions of required number of generations, and we propose to fit such distributions into Gaussian distribution. Consequently, we claim that we proposed a method to assign dynamic computing resources to the steerable calibration process, in order to meet time requirements in DDDAS.

Frameworks are developed in order to implement the designed architectures on computing infrastructures in Chapter 5 (Objective O4). Based on our statistical review of computational steering in Chapter 2, providing computational steering as a flexible web service is put at the heart of the design process. Afterwards, a framework developed for the conventional computational steering is presented. This dedicated framework is based on a collaboration project with IBM to extend the usability of the Blue Gene/Q supercomputers. This specifically designed framework is used as an template for the development of both HDCS and DDDCS frameworks. Finally, frameworks of HDCS and DDDAS are integrated as a general framework of the hybrid computational steering.

Contribution C5: The Computational Steering Web Service (CSWS) and its framework which are developed for the IBM Blue Gene/Q system**.** During the collaboration with IBM, we developed a flexible framework that provides the computational steering as a web service based on the Blue Gene/Q supercomputers. A CSWS server is developed as the middleware that hides the complex architecture of supercomputers from users and provides a steering web application to enable users take advantage of HPC resources on light-weight clients. Additionally, a relay server is designed to securely deal with communications between simulations running on the backend nodes and steering commands received from CSWS server. Moreover, this framework provides us the knowledge and experience of developing computational steering workflow on HPC infrastructures. Hence, even the framework is developed for conventional computational steering, the frameworks of hybrid computational steering is derived from it.

In Chapter 6, calibration in Water Distribution Systems (WDS) is utilized as an example of data assimilation in DDDAS to evaluate our proposed concepts, architectures and frameworks of the hybrid computational steering (**objective O5**). HDCS function was evaluated by steering amount of sensor data which are used by the calibration algorithm. Genetic algorithm (GA) is taken as an example of the calibration algorithm to evaluate DDDAS functions. In the evaluation, the execution speed of GA, which was not integrated with computational steering, is compared with the execution speed of steering-enabled GA. Additionally, the time management method is evaluated by estimating required running time of calibration processes under two variables. First variable is the GA stopping criteria which are steered by human users, and the second variable is the varying situation in WDS. Finally, the computing resources assignment method is evaluated by developing the framework of the hybrid computational steering on Amazon Cloud. The number of instances provided by Amazon Cloud are used to indicate the amount of computing resources. By executing the calibration with different number of instances, a relation among the amount of computing resources, calibration workload and the calibration execution time is created. Such relations are used to train our time manager which can request dynamic computing resources to meet the deadline. Moreover, as the function such as computing resource broker is beyond the scope of thesis, we collaborated with another PhD project, which focuses on the provision of dynamic computing resources.

Contribution C4.b and C6: Implementing designed architectures and frameworks of the hybrid computational steering on Amazon Cloud computing system, and using dynamic

computing resources of Amazon Cloud to manage execution time of the hybrid computational steering system.

The implementation and designed time management methods are demonstrated using a WDS as an example of DDDAS. We have claimed that HDCS can assimilate dynamic steering commands from user into calibration algorithms. To demonstrate this ability in applications of a WDS, the number of sensors used by the genetic algorithm are steered by users in this evaluation. The results indicate HDCS enable users to adjust the genetic algorithm according to their requirements based on the trade-off between execution time and accuracy of calibration. In DDDCS, we presented that GA can steer simulations in the calibration process based on proposed architectures and frameworks for the supercomputer and cloud computing. Additionally, by comparing speeds of conventional calibration and the steering-enabled calibration, we show that DDDCS can greatly increase speed of steerable calibration. Moreover, time management methods can estimate required execution time of the calibration process by studying historical distributions of required generation numbers of GA, and the estimation results can reflect different situations happening in the WDS and dynamic steering commands assigned by users. Since the execution time estimation can indicate a specific number of instances to execute the calibration in parallel, and the calibration can be finished in time in our artificial pipe burst scenario, we claim that based on the estimation, required execution time can be met by assigning dynamic computing resources to DDDCS.

Furthermore, we also conclude that it is difficult to have a perfect estimation on the execution time of calibrations. Thus, the trade-off between estimation accuracy and computing resources cost is required to be considered. This work further introduces a parameter for users to steer in order to control such trade-off.

7.2 Future Work

The hybrid computational steering opens the door to a number of future work directions, including but not limited to the following:

Future Publications

The results of using Amazon cloud computing resources to support calibration process has not been published. As sensor reading collected from different scenarios is another

significant factor that affects the performance of calibration process, it is interesting to publish the results in addition to the results of HDCS in the future.

Introducing Computational Steering to General Calibrations

This thesis only used the Genetic Algorithm as an example of the calibration algorithm to demonstrate the benefits of integrating computational steering with the calibration process. As discussed in Section 2.2, other calibration algorithms can be also expected to take advantage of computational steering. Therefore, we look forward to evaluating integrating computational steering with other calibration algorithms. Moreover, as the estimation of the execution time of calibration algorithm is a dedicated research area, we expect to improve the performance of the time management by researching other methods to estimate the execution time.

Time Management in a Complex System

The time management of workflow management system has been introduced in our project. However, as the use case did not provide complex system, this thesis only selected parts of the methods to demonstrate the capability of the introduced time management method. Therefore, we expect to have a more complex applications of DDDAS to evaluate rest time management methods.

Hybrid Framework of Dynamic Computing Resources

This thesis evaluates computing resources on the frameworks developed for supercomputers and clouds separately. In the future work, it is interesting to conduct evaluations on the computing resources that are from both types of frameworks at the same time.

7.3 Final Thoughts

Computational steering has been considered as a valuable computation tool for decades, and interaction issues between steerer and steered simulations running on HPCs need to be tackled in many research areas. However, its development does not go smooth, and targeted issues are still challenging. By introducing DDDAS as a new application area of computational steering and showing computational steering can facilitate interactions between components in DDDAS, we hope to attract attentions of researchers in the area of DDDAS and motivate the development of computational steering.

In addition, we present an architecture and its frameworks for providing computational steering as a web service on HPCs. By keeping the flexibility of developed frameworks, we hope our work can motivate development of such service in practise in order to provide a stable platform for scientists and software developers to build steerable models and systems.

Finally, I hope this research thesis will contribute to a deeper understanding of advantages of computational steering and designs of computational steering architectures on high-performance computing resources.

Appendix A The Programs Repository

A.1 Statistical Analysis of Computational Steering

In Section 2.1.4 we have introduced that we conducted a statistical analysis of the impact of computational steering by developing a program to search through popular online search tools, such as the Google Scholar and IEEEXplore. It is developed on MacOS using Objective-C as the program language. The program can be found in Bitbucket with the link: https://demonmerlin@bitbucket.org/demonmerlin/gproofread.git. The Figure A-1 demonstrates the user interface we developed for this application.

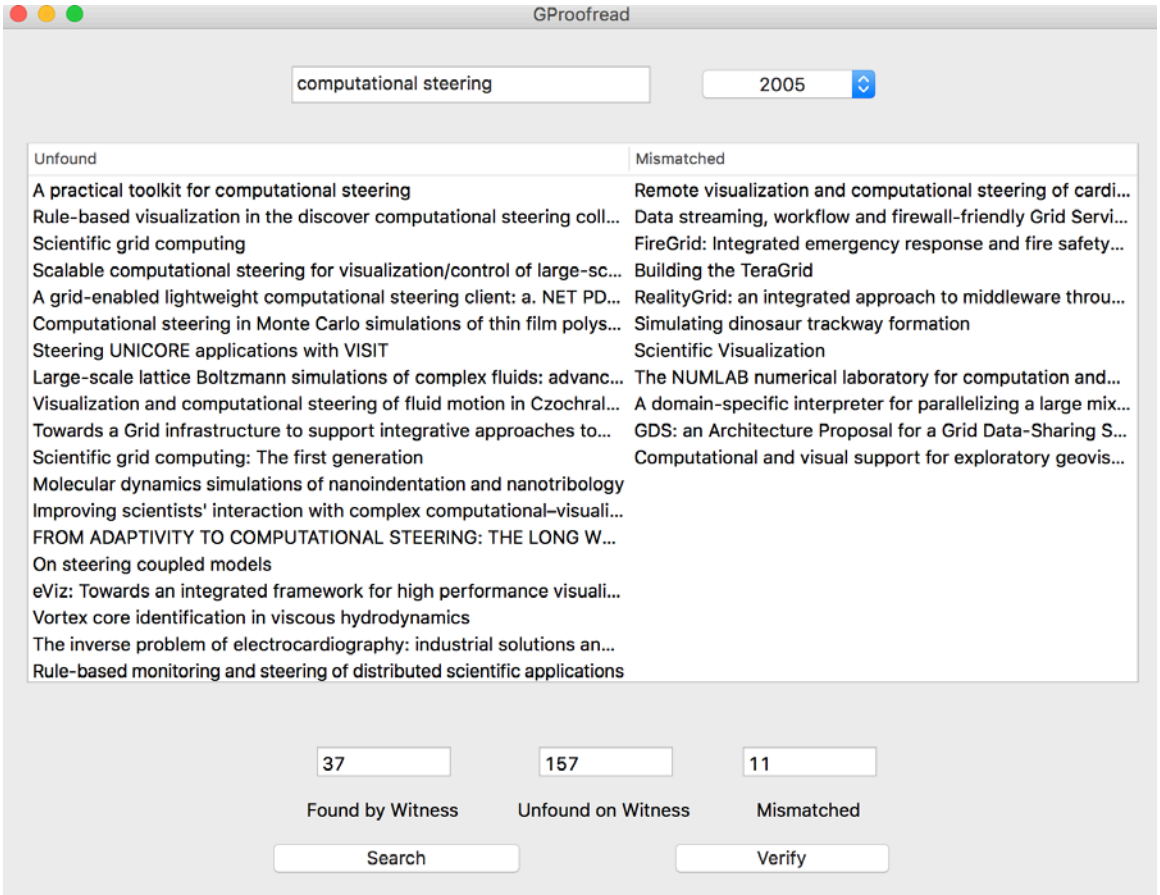


Figure A-1 The User Interface of the Online Research Tool – GProofread: The “computational steering” is the keyword used to search on other online search tools. The list gives options to select years. Users first press “Search” button to search the key word on Google Scholar. Afterwards, users can press “verify” button to filter found papers on IEEE Xplore, ACM DL, Springer and ScienceDirect, WoS and Scopus. The “Found by Witness” indicates the number of papers found on Google Scholar and also on other search tool, and the information on both side are matched “Unfound on Witness” indicates the number of papers are only found on Google Scholar. The “Mismatched” indicates papers found on Google Scholar and also on other search tool, but the information on both side are not matched. We choose to trust other search tools.

A.2 Relay Server

This relay server is derived from the “Relay Server” developed for the collaboration project. Since all programs developed in the collaboration project with IBM cannot be published in public, we only present our relay server developed for the framework on Amazon Cloud. The program was developed on MacOS using C++. It can be found using the link: https://demonmerlin@bitbucket.org/demonmerlin/relay-server.git.

A.3 Calibrator – The Steerer of the DDDCS

The Calibrator is mainly developed by the Genetic Algorithm using C++. The program has been uploaded on Bitbucket and can be found using the link:

- For Amazon Cloud:
  https://demonmerlin@bitbucket.org/demonmerlin/calibratorec2.git.
- For Supercomputers:
  https://demonmerlin@bitbucket.org/demonmerlin/calibratorbgq.git.

A.4 Simulations

The simulation program is developed using the EPANET model provided by the Environmental Protection Agency. The program has been uploaded on Bitbucket and can be found using the link:

- For Amazon Cloud:
  https://demonmerlin@bitbucket.org/demonmerlin/simulationsec2.git.
- For Supercomputers:
  https://demonmerlin@bitbucket.org/demonmerlin/simulationsbgq.git.

A.5 Computing Resource Manager

This Computing Resource Manager is used to demonstrate that our project can communicate existing works to request dynamic computing resources. It presents the progress of requesting dynamic computing resources by communicating with a Computing Resource Broker. The Computing Resource Broker provides RESTful API for the Computing Resource Manager. The Computing Resource Manager uses TCP sockets to communicate

with the Steerer using C++, and it connects with Computing Resource Broker based on Restlet, which is an open source RESTful web API framework for the Java platform. As a result, Java Native Interface (JNI) is used to call Java functions in a C++ program. The program has been uploaded on Bitbucket and can be found using the link:

- For the C++ part:
  https://demonmerlin@bitbucket.org/demonmerlin/timemanagercpp.git.
- For the Java part:
  https://demonmerlin@bitbucket.org/demonmerlin/timemanagerjava.git.

A.6 Pattern Splitter and EPANET Engine

A pattern splitter has been developed to generate artificial water demand patterns which can be used as input to obtain artificial sensor readings. The program can be found with the link: https://demonmerlin@bitbucket.org/demonmerlin/patternspliter.git.

The generated water demand patterns were integrated with the EPANET input file which describes the water network. The link for the generated input file can be found using the link: https://www.dropbox.com/s/zplinuw5u0ggew3/SZ01_nc_15M_ex.inp?dl=0.

An EPANET Engine is also developed to generate required artificial sensor readings as described in Section 6.2.1. The program of the EPANET Engine can be accessed using link: https://demonmerlin@bitbucket.org/demonmerlin/epanetengine.git.

Appendix B The Computational Steering Web Application

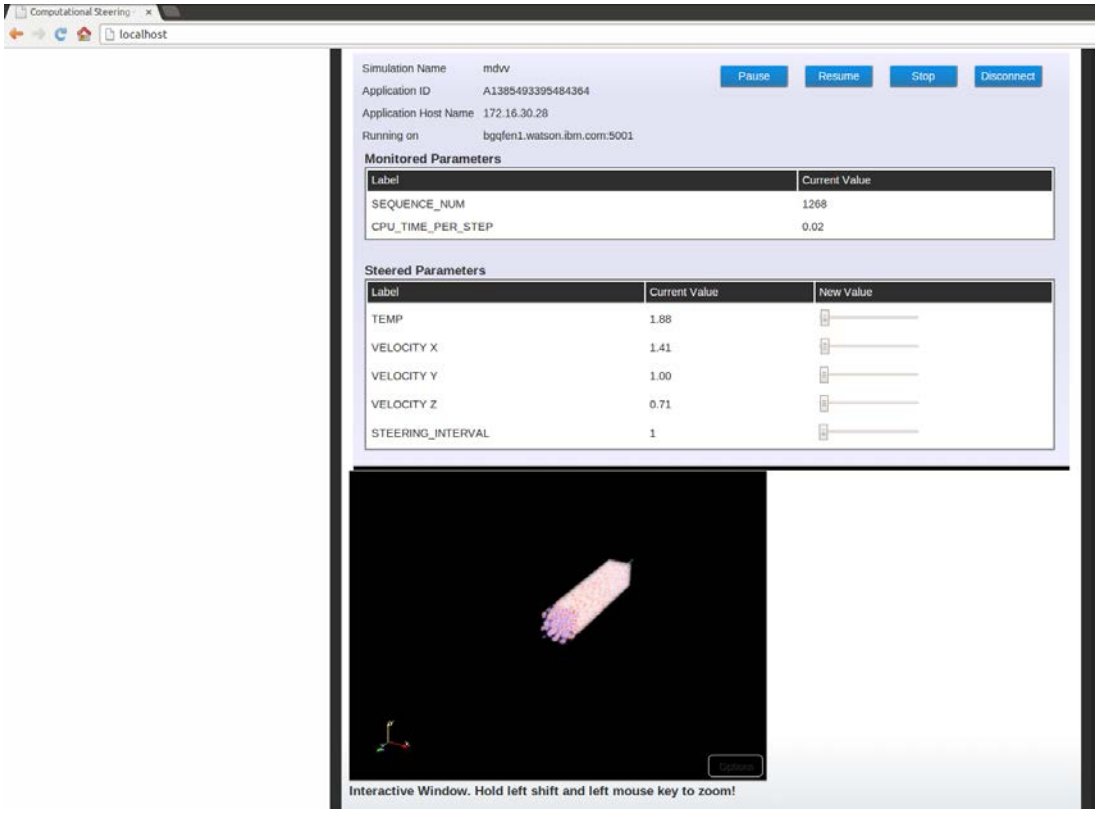


Figure B-1 Web Application Screenshot 1: The Browser based steering interface running on mobile device. It presents the real-time results of a simulation running on the Blue Gene/Q supercomputer.

Figure B-1 shows that users use the Google Web Browser to open the web application and the rea- time visualization data is shown on the web page. The visualization results extracted from running simulations are shown in the interactive window on the bottom. Users can pause the simulation to check the visualization data and manipulate the object in real-time shown in the viewport. The steerable parameters are shown in the Steered Parameters window. Users can slide bars under the “New Value” column to modify parameters on the fly. As a result of modifying parameters, users obtain feedback from the visualization data in real-time.

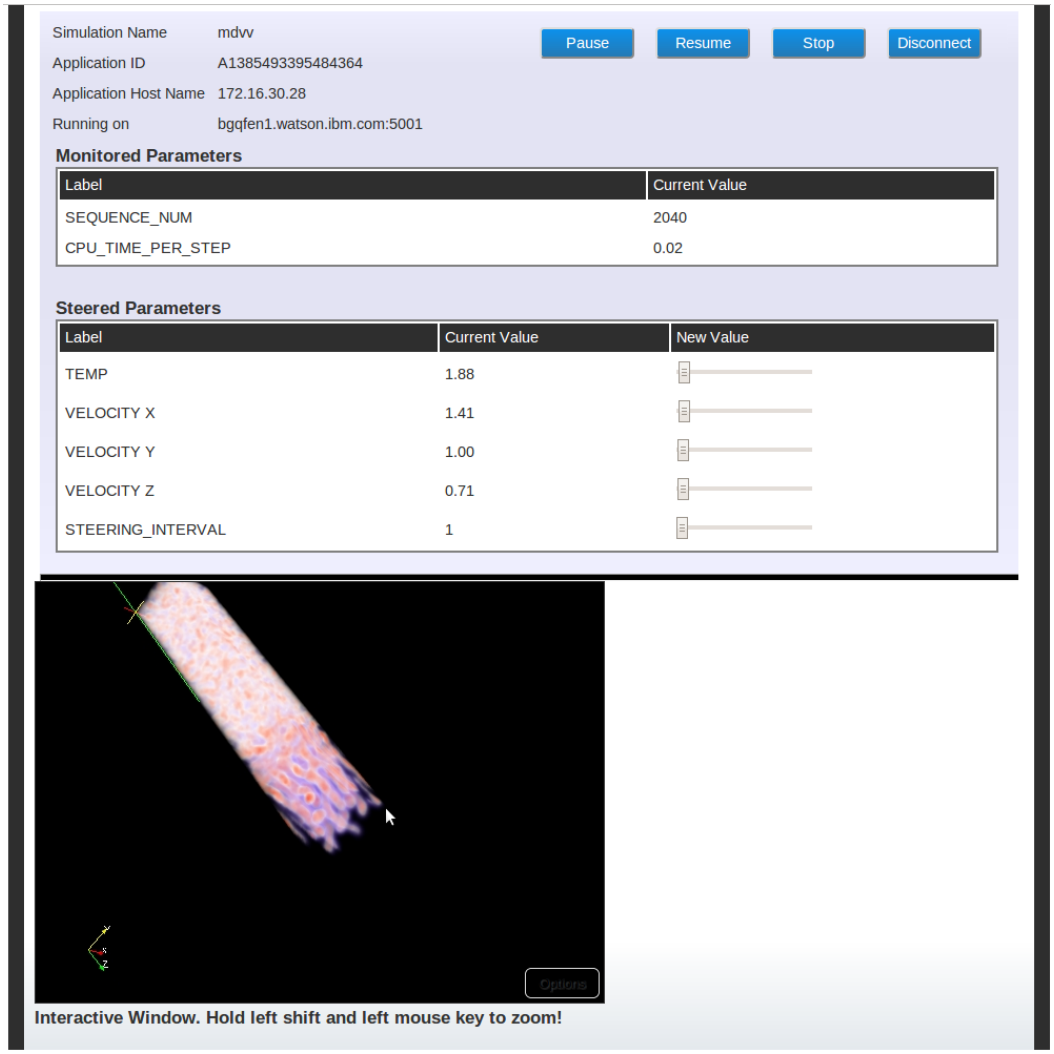
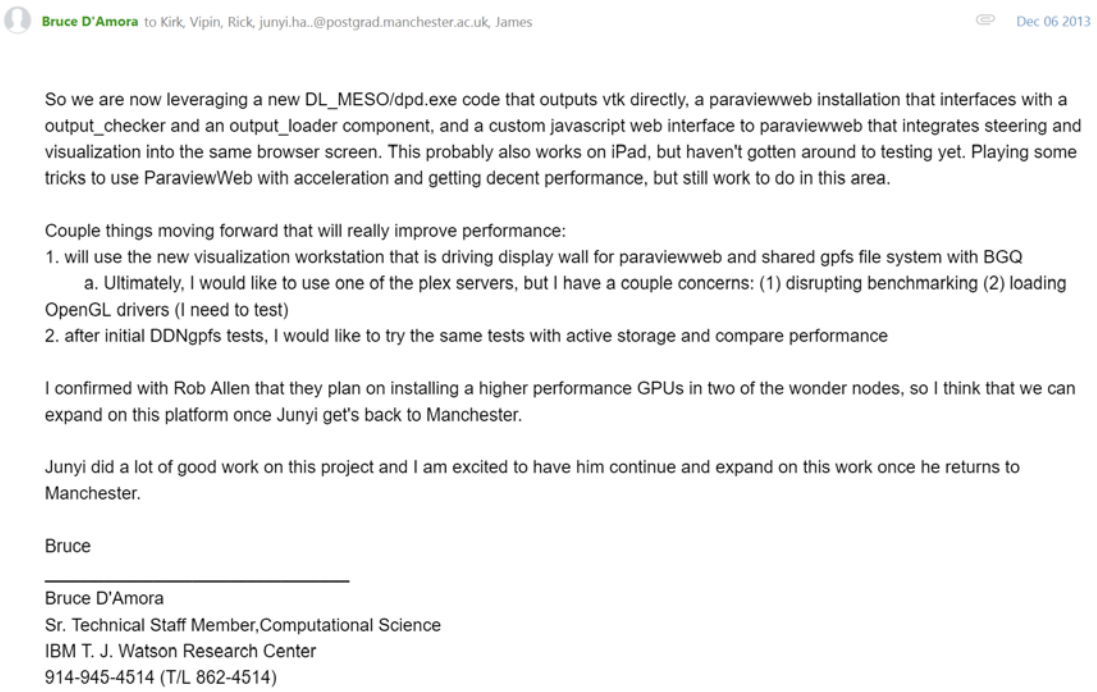


Figure B-2 Web Application Screen Shot 2: The Browser based steering interface running on mobile device. It presents the real-time results of a simulation running on the Blue Gene/Q supercomputer. It demonstrates that users can change angles view the simulation results.

By comparing Figure B-2 with Figure B-1, we demonstrate that instead of displaying a picture of the visualisation result, this web application provides users with a 3D and interactive visualisation interface in which users can manipulate visualization data such as zooming in and changing the visualisation angle.

Appendix C Feedbacks



Bruce D'Amora to Kirk, Vipin, Rick, junyi.ha..@postgrad.manchester.ac.uk, James Dec 06 2013

So we are now leveraging a new DL_MESO/dpd.exe code that outputs vtk directly, a paraviewweb installation that interfaces with a output_checker and an output_loader component, and a custom javascript web interface to paraviewweb that integrates steering and visualization into the same browser screen. This probably also works on iPad, but haven't gotten around to testing yet. Playing some tricks to use ParaviewWeb with acceleration and getting decent performance, but still work to do in this area.

Couple things moving forward that will really improve performance:
1. will use the new visualization workstation that is driving display wall for paraviewweb and shared gpfs file system with BGQ
   a. Ultimately, I would like to use one of the plex servers, but I have a couple concerns: (1) disrupting benchmarking (2) loading OpenGL drivers (I need to test)
2. after initial DDNgpfs tests, I would like to try the same tests with active storage and compare performance

I confirmed with Rob Allen that they plan on installing a higher performance GPUs in two of the wonder nodes, so I think that we can expand on this platform once Junyi get's back to Manchester.

Junyi did a lot of good work on this project and I am excited to have him continue and expand on this work once he returns to Manchester.

Bruce

Bruce D'Amora
Sr. Technical Staff Member,Computational Science
IBM T. J. Watson Research Center
914-945-4514 (T/L 862-4514)

Figure C-1 Feedback from IBM Engineers

The feedback from WRc and water engineers was recorded in a voice memo. As I do not have the right to publish the entire meeting contents, I can only gave the link of the recording that relevant to my PhD project. The recording has been uploaded on Dropbox and can be found using the link: https://www.dropbox.com/s/upwx87rutbz6tku/Mine.wav?dl=0.

References

[1] (07-08-2016). *AWS Architecture Center*. Available: https://aws.amazon.com/architecture/

[2] (07-08-2016). *Lists on June 2016*. Available: https://www.top500.org/lists/2016/06/

[3] (02-03-2016). *Our operating area*. Available: https://www.nwl.co.uk/your-home/our-operating-area.aspx

[4] S. Abrishami, M. Naghibzadeh, and D. H. Epema, "Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds," *Future Generation Computer Systems*, vol. 29, pp. 158-169, 2013.

[5] A. Aghaei Chadegani, H. Salehi, M. M. Yunus, H. Farhadi, M. Fooladi, M. Farhadi, "A comparison between two main academic literature collections: Web of Science and Scopus databases," *Asian Social Science*, vol. 9, pp. 18-26, 2013.

[6] G. Allen, "Building a dynamic data driven application system for hurricane forecasting," in *International Conference on Computational Science*, 2007, pp. 1034-1041.

[7] J. L. Anderson, "An ensemble adjustment Kalman filter for data assimilation," *Monthly weather review*, vol. 129, pp. 2884-2903, 2001.

[8] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, "A view of cloud computing," *Communications of the ACM*, vol. 53, pp. 50-58, 2010.

[9] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, "Above the clouds: A berkeley view of cloud computing," 2009.

[10] K. Ast, J. Katzke, J. Nickerson, J. R. Norden, and L. Rogin, "NGPSS/6000: A new implementation of GPSS," in *Proceedings of the 6th conference on Winter simulation*, 1973, pp. 804-813.

[11] R. Baheti and H. Gill, "Cyber-physical systems," *The impact of control technology*, vol. 12, pp. 161-166, 2011.

[12] G. Bancroft, T. Plessel, F. Merritt, and V. Watson, "Tools for 3d scientific visualization in computational aerodynamics at NASA Ames Research Center," in *OE/LASE'89*, 1989, pp. 161-172.

[13] Y. Bazilevs, M.-C. Hsu, and M. Bement, "Adjoint-based control of fluid-structure interaction for computational steering applications," *Procedia Computer Science*, vol. 18, pp. 1989-1998, 2013.

[14] Y. Bazilevs, A. L. Marsden, F. L. di Scalea, A. Majumdar, and M. Tatineni, "Toward a computational steering framework for large-scale composite structures based on continually and dynamically injected sensor data," *Procedia Computer Science*, vol. 9, pp. 1149-1158, 2012.

[15] J. Belanger, P. Venne, and J. Paquin, "The what, where and why of real-time simulation," *Planet RT*, vol. 1, 2010.

[16] T. E. Bihari and K. Schwan, "Dynamic adaptation of real-time software," *ACM Transactions on Computer Systems (TOCS)*, vol. 9, pp. 143-174, 1991.

[17] L. G. Birta and G. Arbez, *Modelling and simulation*: Springer, 2007.

[18] K. Brodlie, J. Duce, J. Gallop, M. Sagar, J. Walton, and J. Wood, "Visualization in grid computing environments," in *Visualization, 2004. IEEE*, 2004, pp. 155-162.

[19] J. Brooke, P. Coveney, J. Harting, S. Jha, S. Pickles, R. Pinning, "Computational steering in RealityGrid," in *Proceedings of the UK e-Science All Hands Meeting*, 2003.

[20] F. P. Brooks, "Grasping reality through illusion—interactive graphics serving science," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 1988, pp. 1-11.

[21] P. Buxmann, T. Hess, and S. Lehmann, "Software as a Service," *Wirtschaftsinformatik*, vol. 50, pp. 500-503, 2008.

[22] J. Cao, S. A. Jarvis, S. Saini, and G. R. Nudd, "Gridflow: Workflow management for grid computing," in *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium*, 2003, pp. 198-205.

[23] S. Center, "Computing Services to Accelerate Research and Innovation," ed, 2015.

[24] T. Chai and R. R. Draxler, "Root mean square error (RMSE) or mean absolute error (MAE)?–Arguments against avoiding RMSE in the literature," *Geoscientific Model Development*, vol. 7, pp. 1247-1250, 2014.

[25] F. Chatzinikos and H. Wright, "Computational Steering by Direct Image Manipulation," in *VMV*, 2001, pp. 455-462.

[26] M. P. Clark, D. E. Rupp, R. A. Woods, X. Zheng, R. P. Ibbitt, A. G. Slater, "Hydrological data assimilation with the ensemble Kalman filter: Use of streamflow observations to update states in a distributed hydrological model," *Advances in water resources*, vol. 31, pp. 1309-1324, 2008.

[27] A. Clayton, A. C. Lorenc, and D. M. Barker, "Operational implementation of a hybrid ensemble/4D‐Var global data assimilation system at the Met Office," *Quarterly Journal of the Royal Meteorological Society*, vol. 139, pp. 1445-1461, 2013.

[28] P. Courtier, J. N. Thepaut, and A. Hollingsworth, "A strategy for operational implementation of 4D‐Var, using an incremental approach," *Quarterly Journal of the Royal Meteorological Society*, vol. 120, pp. 1367-1387, 1994.

[29] P. V. Coveney, "Scientific grid computing," *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 363, pp. 1707-1713, 2005.

[30] C. Csuri, S. Dyer, J. Faust, and R. Marshall, "A Flexible Integrated Graphics Environment for Supercomputers and Workstations," *Science and Engineering on Cray Supercomputers, Cray Research, Inc*, pp. 533-548, 1987.

[31] F. Darema, "Dynamic data driven applications systems: A new paradigm for application simulations and measurements," in *Computational Science-ICCS 2004*, ed: Springer, 2004, pp. 662-669.

[32] K. Deb, "An efficient constraint handling method for genetic algorithms," *Computer methods in applied mechanics and engineering*, vol. 186, pp. 311-338, 2000.

[33] D. P. Dee, "Bias and data assimilation," *Quarterly Journal of the Royal Meteorological Society*, vol. 131, pp. 3323-3343 %@ 1477-870X, 2005.

[34] M. Denham, A. Cortés, and T. Margalef, "Computational steering strategy to calibrate input variables in a dynamic data driven genetic algorithm for forest fire spread prediction," in *International Conference on Computational Science*, 2009, pp. 479-488.

[35] M. Denham, K. Wendt, G. Bianchini, A. Cortés, and T. Margalef, "Dynamic data-driven genetic algorithm for forest fire spread prediction," *Journal of Computational Science*, vol. 3, pp. 398-404, 2012.

[36] P. Derler, E. A. Lee, and A. S. Vincentelli, "Modeling cyber–physical systems," *Proceedings of the IEEE*, vol. 100, pp. 13-28 %@ 0018-9219, 2012.

[37] P. A. Dinda, "Online prediction of the running time of tasks," *Cluster Computing*, vol. 5, pp. 225-236, 2002.

[38] J. J. Donovan, M. Jones, and J. Alsop, "A graphical facility for an interactive simulation system," in *IFIP Congress (1)*, 1968, pp. 593-596.

[39] R. Duan, F. Nadeem, J. Wang, Y. Zhang, R. Prodan, and T. Fahringer, "A hybrid intelligent method for performance modeling and prediction of workflow activities in grids," in *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2009, pp. 339-347.

[40] G. Evensen, "The ensemble Kalman filter: Theoretical formulation and practical implementation," *Ocean dynamics*, vol. 53, pp. 343-367, 2003.

[41] I. Foster, K. Czajkowski, D. Ferguson, J. Frey, S. Graham, T. Maguire, "Modeling and managing state in distributed systems: The role of OGSI and WSRF," *Proceedings of the IEEE*, vol. 93, pp. 604-612, 2005.

[42] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations," *International journal of high performance computing applications*, vol. 15, pp. 200-222, 2001.

[43] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *2008 Grid Computing Environments Workshop*, 2008, pp. 1-10.

[44] National Science Foundation. (2016, 21-09-2016). *Cyber-Physical Systems (CPS)*. Available: https://www.nsf.gov/pubs/2016/nsf16549/nsf16549.htm#pgm_intr_txt

[45] M. A. Fox and A. A. B. Pritsker, "Interactive simulation with GASP IV on a minicomputer," *ACM SIGSIM Simulation Digest*, vol. 6, pp. 55-60, 1975.

[46] R. F. Freund, "Optimal selection theory for superconcurrency," in *Proceedings of the 1989 ACM/IEEE conference on Supercomputing*, 1989, pp. 699-703.

[47] G. Fu and Z. Kapelan, "Fuzzy probabilistic design of water distribution networks," *Water Resources Research*, vol. 47, 2011.

[48] J. Gao, J. Huang, C. R. Johnson, and S. Atchley, "Distributed data management for large volume visualization," in *VIS 05. IEEE Visualization, 2005.*, 2005, pp. 183-189.

[49] D. Garlasu, V. Sandulescu, I. Halcu, G. Neculoiu, O. Grigoriu, M. Marinescu, "A big data implementation based on Grid computing," in *Roedunet International Conference (RoEduNet), 2013 11th*, 2013, pp. 1-4.

[50] N. Geddes, "The Large Hadron Collider and Grid computing," *Philosophical Transactions of the Royal Society of London*, vol. 370, pp. 965-977, 2012.

[51] G. Geist, J. A. Kohl, and P. M. Papadopoulos, "Cumulvs: Providing Fault Toler. Ance, Visualization, and Steer Ing of Parallel Applications," *International Journal of High Performance Computing Applications*, vol. 11, pp. 224-235, 1997.

[52] A. Ghasemi and S. Zahediasl, "Normality tests for statistical analysis: a guide for non-statisticians," *International journal of endocrinology and metabolism*, vol. 10, pp. 486-489, 2012.

[53] M. Gilge. (2013, 08-08-2016). IBM System Blue Gene Solution Blue Gene/Q Application Development. 2. Available: http://www.redbooks.ibm.com/abstracts/sg247948.html

[54] J. Gray, D. T. Liu, M. Nieto-Santisteban, A. Szalay, D. J. DeWitt, and G. Heber, "Scientific data management in the coming decade," *ACM SIGMOD Record*, vol. 34, pp. 34-41, 2005.

[55] R. Haines. (11-10-2016). *The RealityGrid Computational Steering Library*. Available: http://www.cs.man.ac.uk/~jbrooke/ReGLecture.pdf

[56] R. Haines, K. Khan, and J. Brooke, "Bringing simulation to engineers in the field: a Web 2.0 approach," *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 367, pp. 2635-2644, 2009.

[57] J. Han, R. Haines, A. Salhli, J. M. Brooke, B. D'Amora, and B. Danani, "Virtual science on the move: interactive access to simulations on supercomputers," in *2014 IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors*, 2014, pp. 178-179.

[58] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback control of computing systems*: John Wiley & Sons, 2004.

[59] R. L. Henderson, "Job scheduling under the portable batch system," in *Workshop on Job Scheduling Strategies for Parallel Processing*, 1995, pp. 279-294.

[60] J. Hernando, J. Martinez, V. Martin, M. López, and J. Martin-Gago, "Svis: a computational steering visualization environment for surface structure determination," in *Visualisation, 2009. VIZ'09. Second International Conference*, 2009, pp. 36-39.

[61] A. J. Hey and A. E. Trefethen, "The data deluge: An e-science perspective," 2003.

[62] P. L. Houtekamer and H. L. Mitchell, "Data assimilation using an ensemble Kalman filter technique," *Monthly Weather Review*, vol. 126, pp. 796-811, 1998.

[63] D. Hughes, P. Greenwood, G. Blair, G. Coulson, P. Grace, F. Pappenberger, "An experiment with reflective middleware to support grid‐based flood monitoring," *Concurrency and Computation: Practice and Experience,* vol. 20, pp. 1303-1316 %@ 1532-0634, 2008.

[64] R. Hurrion and R. Secker, "Visual interactive simulation an aid to decision making," *Omega,* vol. 6, pp. 419-426, 1978.

[65] R. D. Hurrion, "Implementation of a visual interactive consensus decision support system," *European Journal of Operational Research,* vol. 20, pp. 138-144, 1985.

[66] R. D. Hurrion, "An interactive visual simulation system for industrial management," *European Journal of Operational Research,* vol. 5, pp. 86-93, 1980.

[67] C. J. Hutton, Z. Kapelan, L. Vamvakeridou-Lyroudia, and D. A. Savić, "Dealing with uncertainty in water distribution system models: a framework for real-time modeling and data assimilation," *Journal of Water Resources Planning and Management,* vol. 140, pp. 169-183, 2012.

[68] M. A. Iverson, F. Özguner, and L. C. Potter, "Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment," in *Heterogeneous Computing Workshop, 1999.(HCW'99) Proceedings. Eighth*, 1999, pp. 99-111.

[69] P. Jacso, "As we may search-Comparison of major features of the Web of Science, Scopus, and Google Scholar citation-based and citation-enhanced databases," *Current Science-Bangalore,* vol. 89, p. 1537, 2005.

[70] L. S. a. M. Jeanne, "SCADA and GIS for Drinking Water Distribution System Monitoring and Response: Critical Gaps," in *Beyond SCADA: Networked Embedded Control for Cyber Physical Systems*, US, 2003.

[71] B. Jerry, *Discrete-event system simulation*: Pearson Education India, 1984.

[72] C. Johnson, "Top scientific visualization research problems," *IEEE Computer Graphics and Applications,* vol. 24, pp. 13-17, 2004.

[73] C. Johnson, S. Parker, D. Weinstein, and S. Heffernan, "Component‐based, problem‐solving environments for large‐scale scientific computing," *Concurrency and Computation: Practice and Experience,* vol. 14, pp. 1337-1349, 2002.

[74] M. M. Jones, "On—line simulation," in *Proceedings of the 1967 22nd national conference*, 1967, pp. 591-599.

[75] E. Kalnay, H. Li, T. Miyoshi, S. C. YANG, and J. BALLABRERA‐POY, "4‐D‐Var or ensemble Kalman filter?," *Tellus A,* vol. 59, pp. 758-773, 2007.

[76] D. Kang and K. Lansey, "Demand and roughness estimation in water distribution systems," *Journal of Water Resources Planning and Management,* vol. 137, pp. 20-30, 2010.

[77] B. Kao and H. Garcia-Molina, "Deadline assignment in a distributed soft real-time system," *Parallel and Distributed Systems, IEEE Transactions,* vol. 8, pp. 1268-1274, 1997.

[78] H. Kao and H. Garcia-Molina, "Deadline assignment in a distributed soft real-time system," in *Distributed Computing Systems, Proceedings the 13th International Conference*, pp. 428-437, 1993.

[79] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, pp. 41-50, 2003.

[80] S. K. Khaitan and J. D. McCalley, "Design techniques and applications of cyberphysical systems: A survey," *IEEE Systems Journal*, vol. 9, pp. 350-365, 2015.

[81] K. Khan, R. Haines and J. M. Brooke, "*A Distributed Computing Architecture to Support Field Engineering in Networked Systems*," 2010 International Conference on Complex, Intelligent and Software Intensive Systems, pp. 54-61, 2010.

[82] K. Khan, "A Distributed Computing Architecture to Enable Advances in Field Operations and Management of Distributed Infrastructure," *Doctor of Philosophy Administered thesis*, Faculty of Engineering and Physical Sciences, The University of Manchester, The University of Manchester, 2012

[83] S. T. Khu, S. Y. Liong, V. Babovic, H. Madsen, and N. Muttil, "Genetic programming and its application in real‐time runoff forecasting1," *JAWRA Journal of the American Water Resources Association*, vol. 37, pp. 439-451, 2001.

[84] S. Kuckuk, T. Preclik, and H. Köstler, "Interactive particle dynamics using OpenCL and Kinect," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 28, pp. 519-536, 2013.

[85] A. V. Kulkarni, B. Aziz, I. Shams, and J. W. Busse, "Comparisons of citations in Web of Science, Scopus, and Google Scholar for articles published in general medical journals," *Jama*, vol. 302, pp. 1092-1096, 2009.

[86] E. A. Lee, S. Matic, S. A. Seshia, and J. Zou, "The Case for Timing-Centric Distributed Software Invited Paper," in *Distributed Computing Systems Workshops, 2009. ICDCS Workshops' 09. 29th IEEE International Conference*, 2009, pp. 57-64.

[87] J. Li, J. F. Burnham, T. Lemley, and R. M. Britton, "Citation analysis: Comparison of web of science®, scopus™, SciFinder®, and google scholar," *Journal of electronic resources in medical libraries*, vol. 7, pp. 196-217, 2010.

[88] R. Liere and J. J. Wijk, "CSE: a modular architecture for computational steering," 1996.

[89] P. Lin, A. MacArthur, and J. Leaney, "Defining autonomic computing: a software engineering perspective," in *2005 Australian Software Engineering Conference*, 2005, pp. 88-97.

[90] D. T. Liu and M. J. Franklin, "GridDB: a data-centric overlay for scientific grids," in *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, 2004, pp. 600-611.

[91] A. Lorenc, S. Ballard, R. Bell, N. Ingleby, P. Andrews, D. Barker, "The Met. Office global three‐dimensional variational data assimilation scheme," *Quarterly Journal of the Royal Meteorological Society*, vol. 126, pp. 2991-3012, 2000.

[92] A. C. Lorenc, "The potential of the ensemble Kalman filter for NWP—a comparison with 4D ‐ Var," *Quarterly Journal of the Royal Meteorological Society*, vol. 129, pp. 3183-3203, 2003.

[93] P. Malakar, V. Natarajan, and S. S. Vadhiyar, "InSt: an integrated steering framework for critical weather applications," *Procedia Computer Science*, vol. 4, pp. 116-125, 2011.

[94] R. Marshall, J. Kempf, S. Dyer, and C.-C. Yen, "Visualization methods and simulation steering for a 3D turbulence model of Lake Erie," in *ACM SIGGRAPH Computer Graphics*, 1990, pp. 89-97.

[95] P. Matgen, M. Montanari, R. Hostache, L. Pfister, L. Hoffmann, D. Plaza, "Towards the sequential assimilation of SAR-derived water stages into hydraulic models using the Particle Filter: proof of concept," *Hydrology and Earth System Sciences*, vol. 14, pp. 1773-1785, 2010.

[96] A. Matsunaga and J. A. Fortes, "On the use of machine learning to predict the time and resources consumed by applications," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, 2010, pp. 495-504.

[97] R. J. McAdam, "Continuous interactive simulation: Engaging the human sensory-motor system in understanding dynamical systems," *Procedia Computer Science*, vol. 1, pp. 1691-1698, 2010.

[98] B. H. McCormick, T. A. DeFanti, and M. D. Brown, "Visualization in scientific computing," vol. 7, ed: IEEE Computer SOC 10662 Los Vaqueros Circle, PO Box 3014, Los Alamitos, CA 90720-1264, 1987, pp. 69-69.

[99] L. I. Meho and K. Yang, "Impact of data sources on citation counts and rankings of LIS faculty: Web of Science versus Scopus and Google Scholar," *Journal of the american society for information science and technology*, vol. 58, pp. 2105-2125, 2007.

[100] Z. Meng and J. Brooke, "Negotiation Protocol for Agile and Reliable E-Science Collaboration," in *e-Science (e-Science), 2015 IEEE 11th International Conference*, 2015, pp. 292-295.

[101] F. R. Met Office, Exeter, Devon, EX1 3PB, United Kingdom. (2009). *Optimising the Use of Observations in Data Assimilation*. Available: http://www.metoffice.gov.uk/research/areas/data-assimilation-and-ensembles/use-of-observations

[102] M. Miller, C. D. Hansen, and C. R. Johnson, "Simulation steering with SCIRun in a distributed environment," in *International Workshop on Applied Parallel Computing*, 1998, pp. 366-376.

[103] Y. Mo, T. H.-J. Kim, K. Brancik, D. Dickinson, H. Lee, A. Perrig, "Cyber–physical security of a smart grid infrastructure," *Proceedings of the IEEE*, vol. 100, pp. 195-209, 2012.

[104] B. C. Mukherjee and K. Schwan, "Improving performance by use of adaptive objects: Experimentation with a configurable multiprocessor thread package," in *High*

Performance Distributed Computing, 1993., Proceedings the 2nd International Symposium, 1993, pp. 59-66.

[105] J. D. Mulder and J. J. Van Wijk, "3D computational steering with parametrized geometric objects," in *Proceedings of the 6th conference on Visualization'95*, 1995, p. 304.

[106] J. D. Mulder, J. J. van Wijk, and R. van Liere, "A survey of computational steering environments," *Future generation computer systems*, vol. 15, pp. 119-129, 1999.

[107] S. Munir, J. A. Stankovic, C.-J. M. Liang, and S. Lin, "Cyber physical system challenges for human-in-the-loop control," in *Presented as part of the 8th International Workshop on Feedback Computing*, 2013.

[108] I. M. Navon, "Data assimilation for numerical weather prediction: a review," in *Data assimilation for atmospheric, oceanic and hydrologic applications*, ed: Springer, 2009, pp. 21-65.

[109] M. Palve. (2015, 03-04-2016). *Water supply systems in Architecture Services*. Available: http://www.slideshare.net/MinalPalve/water-supply-system-minal-palve

[110] S. G. Parker and C. R. Johnson, "SCIRun: a scientific programming environment for computational steering," in *Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM)*, 1995, pp. 52.

[111] S. G. Parker, M. Miller, C. D. Hansen, and C. R. Johnson, "An integrated problem solving environment: The SCIRun computational steering system," in *Proceedings of the Hawaii International Conference on System Sciences*, 1998, pp. 147-156.

[112] C. L. Phillips and R. D. Habor, *Feedback control systems*: Simon & Schuster, 1995.

[113] J. T. Piao and J. Yan, "Computing resource prediction for MapReduce applications using decision tree," in *Asia-Pacific Web Conference*, 2012, pp. 570-577.

[114] S. Pickles, R. Haines, R. Pinning, and A. Porter, "A practical toolkit for computational steering," *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 363, pp. 1843-1853, 2005.

[115] I. Pietri, G. Juve, E. Deelman, and R. Sakellariou, "A performance model to estimate execution time of scientific workflows on the cloud," in *Workflows in Support of Large-Scale Science (WORKS), 2014 9th Workshop*, 2014, pp. 11-19.

[116] S. Prabhakaran, M. Iqbal, S. Rinke, C. Windisch, and F. Wolf, "A batch system with fair scheduling for evolving applications," in *2014 43rd International Conference on Parallel Processing*, 2014, pp. 351-360.

[117] A. Preis, A. Whittle, and A. Ostfeld, "On-line hydraulic state prediction for water distribution systems," in *Proc. WDSA09 (Water Distribution Systems Analysis Symposium), World Environmental & Water Resources Congress, Kansas City, Mo*, 2009.

[118] A. Preis, A. J. Whittle, A. Ostfeld, and L. Perelman, "Efficient hydraulic state estimation technique using reduced models of urban water networks," *Journal of Water Resources Planning and Management*, vol. 137, pp. 343-351, 2010.

[119] P. Ranjan, R. J. La, and E. H. Abed, "Global stability conditions for rate control with arbitrary communication delays," *IEEE/ACM Transactions on Networking (TON)*, vol. 14, pp. 94-107, 2006.

[120] Z. Rao and E. Salomons, "Development of a real-time, near-optimal control process for water-distribution networks," *Journal of Hydroinformatics*, vol. 9, pp. 25-37, 2007.

[121] F. Rawlins, S. Ballard, K. Bovis, A. Clayton, D. Li, G. Inverarity, "The Met Office global four ‐ dimensional variational data assimilation scheme," *Quarterly Journal of the Royal Meteorological Society*, vol. 133, pp. 347-362, 2007.

[122] J. M. Rivas, J. J. Gutiérrez, J. C. Palencia, and M. G. Harbour, "Optimized deadline assignment for tasks and messages in distributed real-time systems," in *Proceedings of the 8th International Conference on Embedded Systems and Applications, ESA*, 2010.

[123] L. A. Rossman. (20/09/2016). *EPANET USERS MANUAL*. Available: https://nepis.epa.gov/Adobe/PDF/P1007WWU.pdf

[124] K. Roushangar, M. Zarghaami, and M. Tarlaniazar-Azar, "Forecasting Daily Urban Water Consumption using Conjunctive Evolutionary Algorithm and Wavelet Transform Analysis, A Case Study of Hamedan City, Iran," 2015.

[125] P. Salamon and L. Feyen, "Assessing parameter, precipitation, and predictive uncertainty in a distributed hydrological model using sequential data assimilation with the particle filter," *Journal of Hydrology*, vol. 376, pp. 428-442, 2009.

[126] A. Salhi, "Visual and numerical methods for unfolding bifurcations," *Doctor of Philosophy Administered thesis*, Faculty of Engineering and Physical Sciences, The University of Manchester, The University of Manchester, 2012.

[127] D. A. Savic, Z. S. Kapelan, and P. M. Jonkergouw, "Quo vadis water distribution model calibration?," *Urban Water Journal*, vol. 6, pp. 3-22, 2009.

[128] J. Shi, J. Wan, H. Yan, and H. Suo, "A survey of cyber-physical systems," in *Wireless Communications and Signal Processing (WCSP), 2011 International Conference*, 2011, pp. 1-6.

[129] W. Smith, I. Foster, and V. Taylor, "Predicting application run times using historical information," in *Workshop on Job Scheduling Strategies for Parallel Processing*, 1998, pp. 122-142.

[130] W. R. Smith, W. L. Anderson, M. I. Haftel, E. Kuo, M. Rosen, and J. K. Uhlmann, "Goal-orientated computational steering," *in SPIE Conference on Enabling Technology for Simulation Science*, 1999, pp. 250-261.

[131] J. A. Stankovic, "Real-time computing," In *Tutorial: hard real-time systems* (pp. 14-37). IEEE Computer Society Press. 1992.

[132] I. Stoianov. (2005 17-09). *Sensor Networks for Monitroing Water Supply Systems*. Available: http://db.csail.mit.edu/dcnui/applications.htm

[133] SWAN-Forum, "The Value of Online Water Network Monitoring," SWAN Forum UK20122012.

[134] J. E. Swan II, M. Lanzagorta, D. Maxwell, E. Kuo, J. Uhlmann, W. Anderson, "A computational steering system for studying microwave interactions with missile bodies," in *Proceedings of the conference on Visualization'00*, 2000, pp. 441-444.

[135] Taverna Team. (2014, 21-05). *Why use workflows?* Available: https://taverna.incubator.apache.org/introduction/why-use-workflows

[136] Trefis Team. (2016, 09-09-2016). *Amazon Continues To Gain Share In Cloud Infrastructure Services Market.* Available: http://www.forbes.com/sites/greatspeculations/2016/08/17/amazon-continues-to-gain-share-in-cloud-infrastructure-services-market/#18951d4e323e

[137] R. Tomlinson, "Intervention—the interface between reality and thought," in *DGOR*, ed: Springer, 1981, pp. 25-40.

[138] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, "Open grid services infrastructure (OGSI)," Open Grid Services Infrastructure (OGSI) Version, 1(0), 2009.

[139] M. Turner, D. Budgen, and P. Brereton, "Turning software into a service," *Computer.*, vol. 36, pp. 38-44, 2003.

[140] C. Upson and S. Fangmeier, "The Role of Visualization and Parallelism in a Heterogeneous Supercomputing Environment," *Parallel Processing and Image Display, R. Earnshaw, ed., Springer-Verlag. New York*, 1989.

[141] C. Upson, T. Faulhaber, D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, "The application visualization system: A computational environment for scientific visualization," *IEEE Computer Graphics and Applications*, vol. 9, pp. 30-42, 1989.

[142] P. J. Van Leeuwen, "Particle filtering in geophysical systems," *Monthly Weather Review*, vol. 137, pp. 4089-4114, 2009.

[143] R. Van Liere, J. D. Mulder, and J. J. Van Wijk, "Computational steering," *Future Generation Computer Systems*, vol. 12, pp. 441-450, 1997.

[144] J. J. van Wijk and R. Van Liere, "An environment for computational steering," *Scientific Visualization: Overviews, Methodologies, and Techniques*, pp. 89-110, 1997.

[145] P. Vasev, "Web based computational steering system," in *Proceedings of IADIS Multi Conference on Computer Science and Information Systems, Computer Graphics and Visualization*, 2008, pp. 324-326.

[146] S. Verboven, P. Hellinckx, F. Arickx, and J. Broeckhove, "Runtime prediction based grid scheduling of parameter sweep jobs," in *Asia-Pacific Services Computing Conference, 2008. APSCC'08. IEEE*, 2008, pp. 33-38.

[147] J. Vetter and K. Schwan, "High performance computational steering of physical simulations," in *Parallel Processing Symposium, 11th International*, 1997, pp. 128-132.

[148] J. Vetter and K. Schwan, "Models for computational steering," in *Configurable Distributed Systems, Third International Conference*, 1996, pp. 100-107.

[149] J. S. Vetter and K. Schwan, "Progress: A toolkit for interactive program steering," in *24th International Conference on Parallel Processing*, 1995.

[150] J. A. Vrugt, H. V. Gupta, B. Nualláin, and W. Bouten, "Real-time data assimilation for operational ensemble streamflow forecasting," *Journal of Hydrometeorology,* vol. 7, pp. 548-565, 2006.

[151] A. Wierse, "Performance of the collaborative visualization environment (COVISE) visualization system under different conditions," in *IS&T/SPIE's Symposium on Electronic Imaging: Science & Technology*, 1995, pp. 218-229.

[152] J. Wood and H. Wright, "Steering via the image in local, distributed and collaborative settings," *Concurrency and Computation: Practice and Experience,* vol. 20, pp. 265-276, 2008.

[153] H. Wright, K. Brodlie, and T. David, "Navigating high-dimensional spaces to support design steering," in *Visualization 2000. Proceedings*, 2000, pp. 291-296.

[154] H. Wright, R. H. Crompton, S. Kharche, and P. Wenisch, "Steering and visualization: Enabling technologies for computational science," *Future Generation Computer Systems,* vol. 26, pp. 506-513, 2010.

[155] Z. Xu, X. Chi, and N. Xiao, "High-performance computing environment: a review of twenty years of experiments in China," *National Science Review,* vol. 3, pp. 36-48, 2016.

[156] J. Yu and R. Buyya, "A taxonomy of workflow management systems for grid computing," *Journal of Grid Computing,* vol. 3, pp. 171-200, 2005.

[157] B. P. Zeigler, H. Praehofer, and T. G. Kim, *Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems*: Academic press, 2000.

[158] H. Zhang and Z. Pu, "Beating the uncertainties: ensemble forecasting and ensemble-based data assimilation in modern numerical weather prediction," *Advances in Meteorology,* vol. 2, 2010.

[159] K. Zhang, K. Damevski, V. Venkatachalapathy, and S. G. Parker, "SCIRun2: A CCA framework for high performance computing," in *High-Level Parallel Programming Models and Supportive Environments, 2004. Proceedings. Ninth International Workshop*, 2004, pp. 72-79.

[160] X.J. Z. a. Q. M. Zixu Liu, "Integrating Demand Response into Electricity Market," presented at the World Congress on Computational Intelligence, Vancouver, 2016.