Computational Complexity and Distributed Execution in Water Quality Management

Maria Chtepen¹, Filip Claeys², Bart Dhoedt¹, Peter Vanrolleghem²,
and Piet Demeester¹

¹ Department of Information Technology (INTEC),
Ghent University, Sint-Pietersnieuwstraat 41, Ghent, Belgium

{maria.chtepen, bart.dhoedt, piet.demeester}@intec.ugent.be

² Department of Applied Mathematics, Biometrics and Process Control (BIOMATH),
Ghent University, Coupure Links 653, Ghent, Belgium

{filip.claeys, peter.vanrolleghem}@biomath.ugent.be

Abstract. Tourist beaches on the southern coast of Turkey are surveyed in order to facilitate a standardised fuzzy approach to be used in litter prediction and to assess the aesthetic state of the coastal environment for monitoring programs. During these surveys the number of litter items on beaches were counted and recorded in different categories. The main source of litter on beaches was determined as “beach users”. A fuzzy system was developed to predict the classification of the beaches, since uncertainty was generally inherent in beach work due to the high variability of beach characteristics and the sources of litter categories. This resulted in effective utilization of “the judgment and knowledge of beach users” in the evaluation of beach gradings.

1 Introduction

The importance of Water Quality Management has drastically increased during the last decades as a consequence of growing environmental awareness. Compared to well-defined systems (e.g. electrical and mechanical) that can entirely be described by classical laws of physics, the behavior of ill-defined water systems is much more difficult to predict due to only partial availability of generally applicable laws. As a result of the complexity of water systems, modeling is regarded an inherent part of design, operation and optimization. The models used in practice are typically based on a combination of general physical laws (typically laws of preservation of energy and mass) and purely empirical laws. These models form an excellent tool to summarize and increase the understanding of complex interactions in biological systems.

As an example of a Water Quality Management application the WEST [1, 2] software system will be considered. WEST¹ is a versatile yet powerful tool, which thus far has mainly been applied to wastewater treatment processes. It consists

¹ World-wide Engine for Simulation and Training.

of clearly separated Modeling and Experimentation Environments. Both environments are self-contained and consist of a graphical front-end, computational back-end and control logic. The Modeling Environment allows for the creation of executable models on the basis of high-level model descriptions, through the application of model compiler techniques. These executable models are subsequently used as a basis for the creation of Virtual Experiments (VE's) in the Experimentation Environment. The reason why the term "VE" was adopted in this case is related to the fact that WEST goes beyond plain simulation. In fact, the types of VE's that are currently supported are: Simulation, Steady-state Analysis, Optimization, Confidence Analysis, Scenario Analysis, Sensitivity Analysis and Uncertainty Analysis. VE's such as Simulation and Steady-state Analysis are said to be *atomic* since they are not hierarchically composed of other VE's. On the other hand, *compound* VE's such as Optimization, Confidence Analysis, Scenario Analysis, Sensitivity Analysis and Uncertainty Analysis all are based on the repeated execution of simulations for different parameter sets.

Modeling and Virtual Experimentation in the domain of Water Quality Management is computationally complex. Total model building times range from a couple of seconds to several hours and typical execution times for compound VE's range from a number of seconds to several weeks on a high-end PC. To reduce this complexity a new system named WDVE [3] was designed that allows for distributed execution of compound VE's. WDVE can be seen as a light-weight Grid system [4] dedicated to the Water Quality Management application area. It was built on top of technologies such as C++, STL, XML, and SOAP. WDVE was initially implemented for the WEST software system, however thanks to generic design it can be used for execution of any computationally intensive task consisting of an unordered set of jobs.

Design and implementation of the WDVE system started from a number of important requirements.

Financial. The new system shouldn't require any additional investments in terms of hardware and staffing.

Functional. The system should evidently be efficient and robust, i.e. it should deliver fast execution of jobs and should account for machine crashes and network drop-outs.

Technical. A strict demand for elegant design and coherent implementation was imposed, together with full OS portability, interoperability and limited dependence on third-party tools.

2 WDVE Architecture

WDVE consists of two major modules: Master and Slave. The Master's main functionalities are to receive VE's (Jobs) from the Application front-end, to store Jobs until they are processed, to match them against available work nodes and to initiate execution. Slaves run (possibly concurrent) Jobs on their computational resources and collect the outputs requested by the user.

Master as well as Slave were further decomposed into a number of sub-modules (see Fig. 1), each implementing a particular, well-defined functionality:

Dispatcher. The Dispatcher's main task is to administer the Jobs from the moment they are submitted by the Application until execution is completed and results are available to the user.

Registry. The goal of the Registry is to allow for work nodes to be statically or dynamically registered as Slaves.

Selector. The Selector implements an algorithm for selecting a Slave for the next Job to be executed.

Checker. The consistency of the state of the system is ensured by the Checker background thread, which constantly monitors the status of various entities and triggers appropriate actions.

Acceptor. This module is at the heart of each Slave. It evaluates requests from the Master for the execution of Jobs. In case a Job is accepted, a new instance of the Application back-end is created and Job execution is started.

Fig. 1. WDVE: Architecture

For the transfer of input, output and status data between Master and Slaves, the gSOAP implementation of the SOAP protocol was used. Reasons for electing SOAP from the wide range of the available middleware technologies are performance, portability and the open source nature of gSOAP. In order for the specifics of the middleware not to be visible by the rest of the code, an abstraction layer was added on top of the gSOAP wrapper classes.

An important architectural question is how to represent Jobs in their various incarnations, starting from the point when a Job is submitted to the system and ending with the completed Job and its output data. It was decided to consistently use XML to this end. In the context of WDVE, Jobs descriptions consist of Executor-dependent and Executor-independent information. The Executor-independent information contains a set of Resources including various types of

input and output files. The Executor-dependent information is made up of various other pieces of information that are relevant to the specific Executor that is to execute the Job.

3 Test Results and Conclusions

Tests were run comparing the efficiency of WDVE with monolithic solutions. The results that were found are in line with the typical performance behavior of other distributed systems. In Fig. 2 are some of the results that were obtained from a test using three identical Slaves that execute Jobs assigned by a Master that has a total of 1 to 100 Jobs to execute. From Fig. 2 (left) one easily concludes that in case the Jobs are short, there is nothing to be gained from distributing Jobs over multiple Slaves. However, in case of Jobs of realistic complexity, one does get a significant reduction of the total execution time (Fig. 2, right). As the number of Slaves increases, this reduction becomes more pronounced, up to a point however where no further improvement is possible. The speed-up factors obtained in practice are also always lower than those that one would theoretically expect. Evidently, this is due to the overhead inflicted by the WDVE software itself and the transmission of data over the network.

Fig. 2. WDVE: Test results for short Jobs (left) and long Jobs (right)

References

1. H. Vanhooren, J. Meirlaen, Y. Amerlinck, F. Claeys, H. Vangheluwe, and P. Vanrolleghem. WEST: Modelling Biological Wastewater Treatment. *Journal of Hydroinformatics*, IWA Publishing, 5(1), 2003.
2. HEMMIS N.V. WEST website. <http://www.hemmis.com>.
3. F. Claeys, M. Chtepen, L. Benedetti, B. Dhoedt, and P.A. Vanrolleghem. Distributed Virtual Experiments in Water Quality Management. In *6th International Symposium on Systems Analysis and Integrated Assessment in Water Management (WATERMATEX)*, pages 485–494, Beijing, China, November 3–5 2004. International Water Association.
4. EGEE. Enabling Grids for E-science in Europe (EGEE) website. <http://egee-intranet.web.cern.ch/egee-intranet/gateway.html>.