Design and Implementation of a Data-Driven Simulation Service System

Ki Yong Lee YoonJae Shin YeonJeong Choe SeonJeong Kim
Department of Computer Science
Sookmyung Women's University
Seoul, Republic of Korea
{kiyonglee, yoonjaeshin, cyj, sjkim}@sookmyung.ac.kr

Young-Kyoon Suh Jeong Hwan Sa Kum Won Cho
Supercomputing R&D Center
Korea Institute of Science and Technology Information
Daejeon, Republic of Korea
{yksuh, sa_c, ckw}@kisti.re.kr

ABSTRACT
Computer simulations are widely used in various fields of science and engineering, including computational fluid dynamics. As the demand for the accuracy and quality of simulations grows, the cost of executing simulations itself is also rapidly increasing. However, until now, the reuse of previously obtained simulation results to improve the execution of later simulations has not been much investigated yet. In this paper, we design and implement a simulation service system, which executes requested simulations and returns the result back to the user. More importantly, our simulation service system is *data-driven* in the sense that the system utilizes previously obtained simulation results to improve the execution of later simulations. Furthermore, our system provides the ability to predict the result of a requested simulation using statistical machine learning techniques on the previous simulation results. This allows the user to roughly estimate the result of her requested simulation without actually executing it. Using our developed system, scientists and engineers can share simulation results and obtain simulation results very quickly without executing simulations redundantly, resulting in a lower overhead of the simulation service system.

Categories and Subject Descriptors
H.2.8 [**Information Systems**]: Database Applications – *Scientific databases*

General Terms
Management, Performance, Design

Keywords
Data-driven simulation, simulation service, simulation prediction

EDB2016, October 17-19, 2016, Jeju Island, Korea.

1. INTRODUCTION
Computer simulations are widely used in various fields of science and engineering, especially including computational fluid dynamics (CFD). For example, CFD is a branch of fluid mechanics that uses numerical analysis and algorithms to solve and analyze problems that involve fluid flows [1]. Computers are used to perform the calculations required to simulate the interaction of liquids and gases. Similarly, computer simulations play an important role in computational physics, biology, chemistry, and astrophysics. However, as the demand for the accuracy and quality of simulations grows, the cost of executing simulations itself is also rapidly increasing. To make matters worse, simulations are often repeatedly executed by varying the values of input parameters. In this case, the cost of executing simulations may be prohibitively high. To improve the performance of the execution of simulations, there have been largely two approaches: (1) improving the performance of computer hardware and (2) improving the performance of computer software (e.g., numerical algorithms). The first approach includes the use of a supercomputer or a cluster of computers to improve the computing capability of the simulation system. On the other hand, the second approach focuses on optimizing numerical algorithms or simulation software to improve the speed of programs within a computer.

Until now, however, the reuse of previously obtained simulation results to improve the execution of later simulations has not been much investigated yet. As far as we know, while taking little consideration of reusing the previous simulation results, most existing simulation service systems appear to conduct the same (perhaps long-running) simulations from scratch each time it is requested, or they provide only limited ability to search the previous simulation results. If we can avoid redundant computation by storing and utilizing previously obtained simulations results, we can improve the performance of simulations and also reduce the overhead of the simulation service system. In this paper, we design and implement a *data-driven* simulation service system, which executes requested simulations and returns the result back to the user. In general, a data-driven application system (DDAS) is a system where execution flow is governed by data it processed, in the sense that obtained data can be incorporated in to the execution of the application [2]. In this

point of view, our simulation service system is *data-driven*, because the system utilizes the previous simulation results to improve the execution of later simulations. The main functionality of our developed simulation service system is as follows:

- **Loading simulation results**: if the execution of a simulation completed, the user can load the simulation result into the simulation database, which is maintained by the system. Because it is quite often that a number of similar simulations are executed together, we provide a bulk loading feature.
- **Reusing simulation results**: if the execution of a simulation is requested by a user, the system first searches the previous simulation results stored in the database system. If exists, the system returns the result back to the user without executing the requested simulation.
- **Predicting simulation results**: if the result of a requested simulation does not exist in the simulation database, the system predicts the simulation result based on the stored data and returns the predicted data to the user. Of course, the user can execute the simulation without using the predicted result. To predict the simulation result, we employ several popular statistical machine learning techniques: linear regression, support vector machine (SVM) regression, neural networks, and *k*-nearest neighbor interpolation.

Using our simulation service system, scientists and engineers can share simulation results and obtain simulation results very quickly without executing simulations redundantly, resulting in a lower overhead of the simulation service system.

2. RELATED WORK

In this section, we describe two main approaches to improve the performance of the execution of simulations, and then present some of existing simulation service systems.

The first approach to improve the performance of the execution of simulations is to improve the performance of computer hardware. To improve the computational capacity of computer hardware, a few approaches are generally used. In one approach, multi-core processors are used to parallelize the execution of simulations on a computer. Similarly, multi-core GPUs are widely used to improve the parallelism of numerical algorithms on a single computer. In another approach, hundreds or thousands of computers connected to each other (i.e., computer cluster) work together to reduce the execution time of simulations.

The second approach is to optimize simulation software or numerical algorithms used in simulations. For example, by optimizing or parallelizing numerical algorithms used in a simulation, the execution time of the simulation can be reduced by more than an order of magnitude. For many numerical problems such as matrix decomposition, equation solving, regression, optimization, clustering, and interpolation, there exist many alternative algorithms that may vary in speed and performance. Thus, it is very important to use a good algorithm or software to improve the performance of a simulation.

Currently, there are a number of simulation service systems that provide the ability to store and share simulation results, whose goal is similar to ours. For example, DataSpaces [3] provides a shared-space abstraction for simulation data staging. BIGNASim [4] is a NoSQL database portal for nucleic acids simulation data. SciDrive [5] is a free open-source scientific data publishing platform with the simplicity of Dropbox. DCMS [6] stores molecular simulation data in a relational database system so that researchers can query and search the simulation results. iBIOMES [7] is a management system for large biomolecular simulation and computational chemistry datasets. Scibox [8] is a cloud-based simulation data sharing and storage system providing a Dropbox-like interface. However, these systems do not provide the ability to reuse the existing simulation results automatically without user involvement, and more importantly, do not provide the ability to predict the result of a requested simulation based on the previous simulation results.

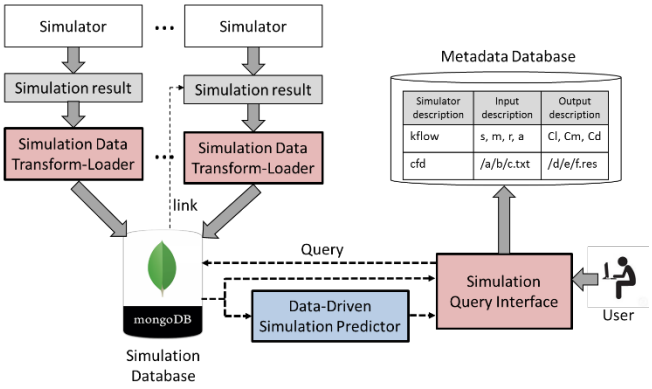

Figure 1. The architecture of our simulation service system

3. OUR SIMULATION SERIVCE SYSTEM

In this section, we describe the architecture and functionality of our data-driven simulation service system.

3.1 Overall Architecture

Figure 1 shows the overall architecture of our developed data-driven simulation service system. Unlike the existing systems, our system utilizes the previous simulation results to improve the execution of later simulations. The developed system mainly consists of five components as follows. (1) Simulation data transform-loader transforms simulation results generated by a simulation program into JSON documents and loads it into the simulation database, which is built on MongoDB [9] in our implementation. (2) Simulation database stores simulation results generated by simulation programs. We adopted the MongoDB as our database system. (3) Simulation query interface receives a user request, searches the simulation database, and returns the corresponding simulation result back to the user. If the simulation database cannot locate the same result, the query interface provides the user with the option to predict the simulation result by calling the data-driven simulation predictor. (4) Data-driven simulation predictor predicts the result of the requested simulation based on the previous simulation results stored in the simulation database. As mentioned earlier, four representative machine learning techniques are considered. (5) Metadata database stores the information about simulation programs (e.g., their input and output parameters). Because the structure of simulation results generated by different simulation programs are quite different from each other, the simulation query interface uses this information to search the result of the requested simulation correctly. In the remainder of this section, we describe each component in more detail.

```
{ simulator: "KFLOW"
  input: {
    airfoil_shape: "NACA0012",
    MACH: "0.1",
    AOA: "2.0",
    RE: "1000000"
  },
  output: {
    aerodynamic: { cl: ...},
    field: ["flo001.dat", ...],
    surface: ["sur002.dat", ...]
}}
```


Figure 2. An example of JSON documents representing a simulation result

3.2 Simulation Data Transform-Loader

If a simulation program finished executing a simulation, it generates the simulation result as files. The user then can load the result into the simulation database by using the simulation data transform-loader. Because different simulation programs have quite different structures of simulation results from each other, we basically need to implement a simulation data transform-loader for each specific simulation program. For example, we have implemented a simulation data transform-loader for the EDISON KFLOW, which is an airfoil simulation program developed by KISTI [10, 11]. To execute a simulation, the KFLOW program needs a number of input parameters such as the name of airfoil shape, Mach number (MACH), Angle of Attack (AOA), Reynolds number (RE), and so on. As a simulation result, it generates aerodynamic data, field data, and surface data. In this case, the simulation data transform-loader reads the simulation output files, extracts the values of input and output parameters from the files, makes a JSON document containing those information, and finally loads the JSON document into the simulation database, which is built on MongoDB. Figure 2 shows an example of a JSON document produced by the simulation data transform-loader. This JSON document is then inserted into the MongoDB, which supports JSON documents.

3.3 Simulation Database

In our implementation, we adopt MongoDB as the simulation database. MongoDB is a free and open-source document-oriented NoSQL database. Unlike the traditional relational database system, MongoDB supports storing JSON documents with dynamic schemas, which makes the integration of various types of data easier. We chose MongoDB because it has two desirable features, which makes it suitable for the needs of our system. First, it supports dynamic schemas, which means we can insert data of any structure without a predefined schema. That makes it easy to store simulations data with various structures. Second, it supports storing large volumes of data using a cluster with a large number of nodes. Because the volume of simulation data are rapidly growing and the number of simulation programs are also increasing, this scalability is quite important for our simulation service system. More specifically, in order to insert and search a simulation result, we use the MongoDB API `db.collection.insert()` and `db.collection.find()`, respectively.

3.4 Simulation Query Interface

A user can request the execution of a simulation and receives the result of the simulation through the simulation query interface. When a user requests an execution of simulation by specifying the values of its input parameters, the simulation query interface first searches the simulation database to find the simulation result with the same values of input parameters. If such simulation result

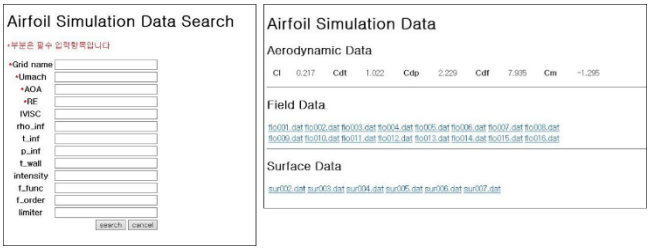

Figure 3. Simulation query interface

exists, then the simulation query interface returns the found result back to the user, without actually executing the requested simulation. Thus, the redundant computation can be avoided and computer hardware resources can be saved. If such simulation result does not exist, the simulation query interface provides the user with two options. The first one is to perform the requested simulation, and the second one is to predict the result of the requested simulation without executing it. If the user chooses the second option, the simulation query interface calls the data-driven simulation predictor, which will be described in the next subsection. Figure 3 is an example of screens showing the simulation query interface, which are used for requesting the execution of an EDISON KFLOW simulation and showing the result found from the simulation database, respectively.

3.5 Data-driven Simulation Predictor

When the amount of simulation results stored in the simulation database becomes large, we can use those data to predict the result of a newly requested simulation, in order to lower the cost of executing a large number of simulations. Of course, because the accuracy of prediction is difficult to estimate in general, this option may be used only for rough estimation. To predict the result of a simulation, we use four representative statistical machine learning techniques: *linear regression*, *support vector machine regression*, *neural networks*, and *k-nearest neighbor interpolation*. All these methods use the simulation results stored in the simulation database as training data. A user can compare the predicted results of those four methods to roughly estimate the result of simulation. Let $X_1, X_2, \ldots, X_n$ be the input parameters of simulation, and Y be the result of simulation. Then the four statistical machine learning techniques used in our system can be briefly described as follows [12]:

- **Linear regression**: This method assumes that the result of simulation has approximately a linear relationship with its input parameters. In other words, Y is modeled as follows, where β_i are unknown parameters:

$$Y = \beta_0 + \sum_{i=1}^{n} X_i \beta_i$$

- **Support vector machine (SVM) regression**: The model is produced by the support vector machine classifier, which maximizes the margin between the training data and the decision boundary. In this model, Y is represented as follows, where $h_m(x)$ ($m = 1, 2, \ldots, M$) is a basis function:

$$Y = \sum_{m=1}^{M} \beta_m h_m(X_1, X_2, \ldots, X_n) + \beta_0$$

- **Neural networks**: In general, a neural network consists of multiple layers, where each layer also consists of multiple neurons. Each neuron in one layer is connected to the

neurons of the subsequent layer. In our implementation, we used a 3-layer neural network with one input layer, one hidden layer, and one output layer. In this model, each neuron is represented as the following function, where x_1, x_2, …, x_m is inputs to the neuron, w_i is a weight, and $K(x)$ is a certain predefined function, called the activation function:

$$f(x_1, x_2, \ldots, x_m) = K(\sum_{i=1}^{m} w_i x_i)$$

- ***k*-nearest neighbor interpolation**: This method uses those observations in the training dataset closest in input space to $X = (X_1, X_2, \ldots, X_n)$ to form Y. Specifically, the k-nearest neighbor fit for Y is defined as follows, where $N_k(X)$ is the neighborhood of X defined by the k closest points in the training sample and y_i is the output of x_i:

$$Y = \frac{1}{k} \sum_{x_i \in N_k(X)} y_i$$

Of course, one size doesn't fit all and each method provides different accuracy depending on the characteristics of the executed simulation and the training data. Thus, we recommend the user to compare the prediction results of the four methods and use them to roughly estimate the simulation result. For example, as a trial, we are trying to predict the EDISON KFLOW simulation results using the four methods, where the number input parameters is 3 (i.e., *MACH*, *AOA*, and *RE*) and the number of output parameters is 5 (i.e., *Cl*, *Cdt*, *Cdp*, *Cdf*, and *Cm*).

3.6 Metadata Database

One important requirement for our simulation service system is to support storing and reusing simulation results produced by multiple simulation programs. To achieve this goal, our system maintains a metadata database storing the following information for each simulation program:

- **Simulation program**: name, version, other information about the simulation program's execution
- **Input parameters:** name, datatype, valid value, required or optional, etc.
- **Output parameters**: name, datatype, display option, etc.

If a user requests the execution of a simulation, the simulation query interface refers to the metadata database to make an appropriate search query and send it to the simulation database. When the simulation query interface displays a simulation result to the user, it refers to the metadata database to build an output screen appropriate for the simulation result. To build the metadata database about various simulation programs, we use two approaches. First, we use the log data maintained in the simulation execution server, which stores the information about execution timestamps, requested users, executed simulation programs, the values of input and output parameters and so on. By analyzing the log data, we can extract the metadata about various simulation programs automatically. Second, we can use a metadata manager to manage the metadata database. The metadata manager allows a user to insert, delete, and modify the metadata stored in the metadata database directly. Using the metadata database, the system can guide users to have integrated and seamless access to simulation results.

4. CONCLUSIONS

In this paper, we designed and implemented a data-driven simulation service system. Unlike the existing systems, the developed system automatically stores and reuses the previously generated simulation results to improve the execution of later simulations. By reusing the previous simulation results, our system can avoid redundant computation, resulting in a reduced response time and saved computing resources. Consequently, the proposed system can serve a greater number of users with less amount of computing resources. Furthermore, the proposed system provides users with the ability to predict simulation results based on statistical machine learning techniques including linear regression, support vector machine regression, neural networks, and *k*-nearest neighbor interpolation. Using this ability to predict a simulation result, a user can quickly estimate the result of her requested simulation without waiting a long time for the execution of simulation. The developed system also can support storing and searching simulation results produced by multiple simulation programs by using the metadata database. We hope that that many CSE researchers and students will efficiently reuse and share simulation results among them using our simulation service system.

5. ACKNOWLEDGMENTS

This research was supported by the EDISON Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT & Future Planning (No. NRF-2011-0020576). Y.-K. Suh is the corresponding author of this paper.

6. REFERENCES

[1] Shiflet, A. B. and Shiflet, G. W. 2014. *Introduction to Computational Science: Modeling and Simulation for the Sciences*. 2nd edition. Princeton University Press.

[2] F. Darema. 2004. Dynamic Data Driven Applications Systems: A New Paradigm for Application Simulations and Measurements. Computational Science. In *Proceedings of Int'l Conf. on Computational Science (ICCS)*. 662–669.

[3] Docan, C., Parashar, M., and Klasky, S. 2012. DataSpaces: an interaction and coordination framework for coupled simulation workflows. *Cluster Computing*, 15, 2 (Jun. 2012), 163-181.

[4] Hospital A. et al. 2016. BIGNASim: a NoSQL database structure and analysis portal for nucleic acids simulation data. *Nucleic Acids Research*, 44 (Jan. 2016), 272-278.

[5] Mishin, D., Medvedev, D., Szalay, A. S., Plante, R., and Graham, M. 2014. Data Sharing and Publication Using the SciDrive Service. In *Proceedings of Astronomical Data Analysis Software and Systems XXIII*, 465.

[6] Kumar, A., Grupcev, V., Berrada, M., Fogarty, J. C., Tu, Y. C., Zhu, X., Pandit, S. A., and Xia, Y. 2014. DCMS: A data analytics and management system for molecular simulation. *Journal of Big Data*, 1, 9 (Nov. 2014).

[7] Thibault, J. C., Facelli, J. C, and Cheatham, T. E. 2013. iBIOMES: Managing and Sharing Biomolecular Simulation Data in a Distributed Environment. *J. Chem. Inf. Model*, 53, 3 (Feb. 2013), 726-736.

[8] Huang, J., Zhang, X., Eisenhauer, G., Schwan, K., Wolf, M., Ethier, S., Klasky, S. 2014. Scibox: Online Sharing of Scientific Data via the Cloud. In *Proceedings of the 28th IEEE International Parallel & Distributed Processing Symposium*. 145-154.

[9] MongoDB, https://www.mongodb.com/.

[10] EDISON-CFD, https://cfd.edison.re.kr/.

[11] Y.-K. Suh, et. al., ``EDISON: A Web-based HPC Simulation Execution Framework for Large-scale Scientific Computing Software," in *Proc. of CCGrid'16*, May 2016, pp. 608–612.

[12] Hastie, T., Tibshirani, R., and Friedman, J. 2008. *The Elements of Statistical Learning*. 2nd Edition, Springer