A System for Management of Computational Fluid Dynamics Simulations for Civil Engineering

Peter Sempolinski and Douglas Thain
Computer Science and Engineering
University of Notre Dame
Email: psempoli@nd.edu and dthain@nd.edu

Daniel Wei and Ahsan Kareem
Civil & Environmental Engineering & Earth Sciences
University of Notre Dame
Email: zwei1@nd.edu and kareem@nd.edu

Abstract—**We introduce a web-based system for management of Computational Fluid Dynamics(CFD) simulations. This system provides an interface for users, on a web-browser, to have an intuitive, user-friendly means of dispatching and controlling long-running simulations. CFD presents a challenge to its users due to the complexity of its internal mathematics, the high computational demands of its simulations and the complexity of inputs to its simulations and related tasks. We designed this system to be as extensible as possible in order to be suitable for many different civil engineering applications. The front-end of this system is a webserver, which provides the user interface. The back-end is responsible for starting and stopping jobs as requested. There are also numerous components specifically for facilitating CFD computation. We discuss our experience with presenting this system to real users and the future ambitions for this project.**

I. Introduction

Civil Engineering, among its many aspects, requires attention to be payed to the flow of wind over structures. One way to gain a greater understanding of the interaction of wind and structures is to use Computational Fluid Dynamics (also known as CFD). However, CFD is one of the most challenging types of scientific computing. Typically, a civil engineer who wishes to use CFD must immediately consult with a CFD professional to perform even a basic simulation. One of our goals for this project is to develop tools that can make CFD more widely useable for various types of civil engineering. In particular, a long-term motivating goal of this project is to eventually produce a "virtual wind-tunnel" that would allow users in non-CFD engineering fields to upload and analyze designs for CFD analysis, without having to re-train themselves in CFD. Also, we are interested in providing resources for beginning users of CFD, to help such beginners understand the nature of CFD work. These motivations serve as long-term goals for this project, providing direction for both present and future work.

In this paper we introduce a task management system for creating and managing CFD simulations. Our design has two major pieces, a web-based front-end which controls interaction with the user and a back-end task manager which controls the dispatch of CFD tasks to various back-ends. One of our key design interests in building these two pieces was to make them extensible, so that the front-end can present many different options for CFD simulations to the user and the back-end can interface with many different execution environments. In addition to these two major components, we developed a number of smaller tools, integrated into our system, specifically intended for the management of CFD tasks.

In the following sections, we discuss the problems related to performing CFD, the overall architecture of our system and the specific tools which we built to help us in managing CFD. We also describe our current progress and the experiences which we have had presenting this system to groups of users. We close with our expectations for further development.

II. Challenges From CFD

CFD is the science of using numerical schemes to solve and analyze fluid-related problems. The basis of CFD is the Navier-Stokes Equation which is a precise description of all kinds of continuous flow. The Navier-Stokes Equation is a complex transport equation system (second order, coupled nonlinear, chaotic system). Therefore, it is extremely difficult to find an analytical solution. Furthering the understanding of these equations remains an unsolved problem for which there is great interest. A problem regarding solutions to these equations is even labeled by the Clay Mathematics Institute as one of their famous "the Millennium Prize" problems [1].

Starting from the late 1970s, solutions to the Navier-Stokes equation were pursued by the discretization method. This method, because of gradually increasing computing power, has met with tremendous success. By the end of the 1980s, together with traditional statistical analysis and wind tunnel testing, CFD has been shown to be a critical and reliable research tool in fluid mechanics. However, it remains that one of the key bottlenecks of CFD analysis is limitations on computing power [2].

In order to increase the accuracy and speed of CFD, various approximations and models have to be used. All of them are difficult both in terms of physics and programming. For example, which discretization scheme should be used, central differencing or upwind schemes? How should one model the near wall turbulence? In addition, challenges remain in parallel computing and multiphase flow along with many other considerations. Things become more complicated in engineering applications of CFD because of boundary conditions and flow patterns. For example, in wind engineering, the flow is mostly turbulent, with a high Reynolds number, separated but also accompanied with reattachment, and the bluff body could be moving.

The work of CFD, however, is not confined to the simulation. A compete CFD analysis includes Pre-processing, Solving and Post-processing. Each of these presents additional challenges. In the Pre-Processing stage, mesh generation is required. A mesh is the spatial discretization of the computational domain. At each simulation time-step, values such as fluid velocity and pressure are computed for each cell in the mesh. The properties of the mesh often play a huge role in determining simulation time and accuracy. Furthermore, a poorly constructed mesh can render a simulation useless. Mesh construction is sufficiently complex that special tools are needed to perform it.

With regard to computer science, there are several key facts about CFD analysis and its application to engineering problems:

- CFD code is usually very large and difficult to maintain, port, install, verify and validate.
- The learning curve of CFD is very steep and requires knowledge of fluid mechanics, mathematics and computer science.
- Because of the massive computational resources needed for CFD simulations, it is usually necessary to dispatch jobs to back-end computing resources.
- Because of the difficulty of configuring "good" CFD, is useful to be able to monitor simulations in progress and stop and correct simulations that are wrong, rather than wait for completion.
- The CFD simulation is actually only a small part of a CFD workflow, increasing the practical difficulties involved.
- The results of CFD often make more sense with some visualization. This includes both graphs and pictures of properties of the flow fields.

We are building a platform, specifically interested in the civil engineering audience, to address these challenges listed above. To do this, we have constructed a system which comprises a front-end for user interaction and a back-end for dispatching tasks. By providing the front-end as a web-based portal, we can tune the user difficulty with regard to the complexity of the simulations the user runs. To do this we implemented an extensible, modular system, in which CFD cases, (a single CFD simulation, and accompanying data and tasks, is called a case) each has a type. This type is both a template and a link to a module in the user interface. In this way, we can create templates for beginner users or advanced users, depending on need. Moreover, by tuning the modules which we make available to users, we can, by varying degrees, conceal the underlying complexity of the CFD code. Of course, by devising templates and case types which correspond to various CFD situations, we can give users the option of various kinds of simulations. Our ambition is to include settings such as wind tunnel testing, 2D sectional model, or 3D wall-mounted model. We can also tune, (or allow users to tune, depending on the template) the turbulence modeling, which is one of the most prominent factors affecting a simulation.

The back-end for our system is designed to take CFD tasks and dispatch them to appropriate execution platforms. Since a true CFD workflow involves more than the simulation, each CFD case requires several different types of tasks, which might be better suited to different execution platforms. For example, the actual simulation for a large 3D simulation would require an execution platform that can perform massive computation, preferably in parallel. However, a post-processing task for a smaller simulation might run faster if simply run locally on the server hosting our system. Therefore, we designed the back-end to have extensibility as well, allowing us to incorporate multiple execution platforms to our back-end and run tasks where they are best suited. Finally, in order to insure the credibility of our platform, we primarily use OpenFOAM as our back-end CFD code which has been well-verified and extensively validated with regard to mesh handling, turbulence modeling, numerical schemes, and unsteady and steady solvers [3]. Moreover, OpenFOAM is open-source code, which makes it particularly suited to our needs. As of right now, we are using the default mesh generator of OpenFOAM.

III. Previous Work

Placing scientific computing resources on the web, is, of course, not new. One example is the Astrophysics Simulation Collaboratory which makes use of the Cactus computational toolkit [4]. Perhaps one of the quintessential examples is the HubZero platform, which is designed for creating websites to facilitate collaboration and research [5]. One popular example deployment of this is nanoHub, which is an online resource hub for nano-electronics [6].

Web portal technology is also employed for monitoring grid resources. For example, GridSphere is specifically designed to host a variety of different applications for the purpose of giving users easy access to grid resources [7]. Also, the JGrid infrastructure augments web technology with Java interfaces, to further enhance user control of grid resources [8]. Also, for visualizing data that results from log-running computation, VizLitG is a framework for accessing and viewing remotely stored data sets [9].

Of course, in addition to platforms providing computation resources, there are many purely informative websites in scientific computing. In CFD, a good example is CFD-online, which provides a wiki and other information of general interest to CFD professionals [10]. Our contribution is to develop an execution platform specifically tuned to CFD, intended to a civil engineering audience.

IV. System Architecture

Our system is composed of two main pieces each of which can be configured with a high degree of extensibility. These components are the front-end and the back-end core. The front-end will take user input to create files for the tasks to be run. The front-end deposits task descriptions into a database,

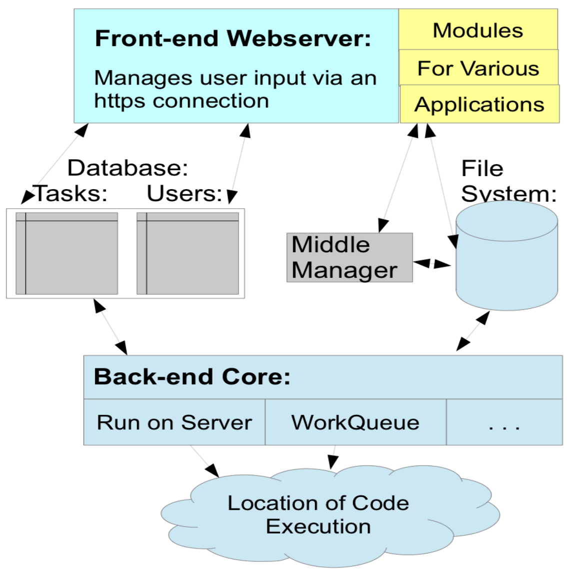


Fig. 1. System Diagram

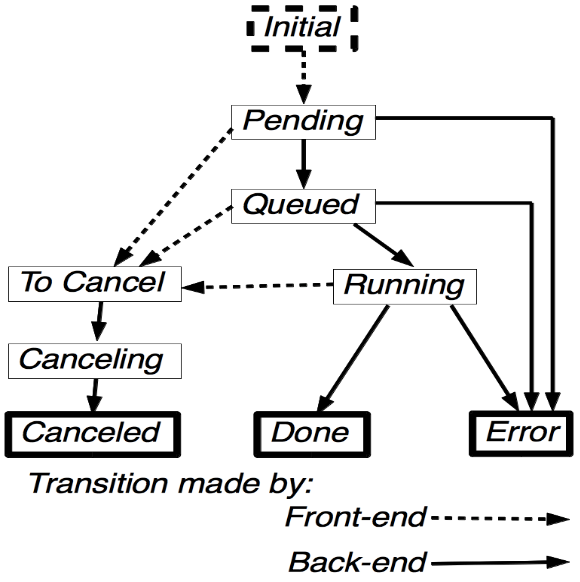


Fig. 2. Task State Transitions: Tasks start in the *Initial* state at the top and end in one of the terminal states at the bottom. Transitions, with rare exceptions, are designated to either the front-end or the back-end, to prevent errors.

which is read by the back-end core. The back-end core will dispatch tasks to various execution environments, depending on configuration and the specification of the front-end. See Figure 1 on how the major parts fit together. In addition to these main pieces, there are several smaller parts (not in the figure) specific for supporting CFD work.

One of the main tensions for this design was to balance the need for extensibility with the need for the system to be effectively tailored to CFD. That is, we wanted our system to be extensible, in both the front-end, so we can provide a user interface for many different kinds of CFD tasks that our users might want, and the back-end, which can communicate equally with several different computational systems, in order to send different tasks to the appropriate execution system. However, we also had to limit this extensibility, in order to provide an an environment with sufficient support for CFD. We discuss the implementation details below.

A. *Life-cycle of a Task*

The foundation of our system is not a particular software component. Rather it is a specific understanding of the states of tasks as they are run. In Figure 2, we have the state diagram for all our tasks. The reason why the state diagram is so important is that it precisely defines the relative duties of the front-end and the back-end. Each valid state transition can either be performed by the front-end or the back-end. The front-end may start tasks, by transitioning to the *Pending* state, and cancel tasks by transitioning to the *To Cancel* state. The back-end attends to running the tasks by transitioning through the *Running* state into the *Done* state. (Or the *Error* state if some thing goes wrong.) By being clear about what each state transition is supposed to do to a task, we can also be precise about what each action on a task is, allowing us to define the interface for our computing backend. Also, by being very precise about what is and is not a valid state transition, it is easier to detect errors and to deal with situations where both the front-end and the back-end are modifying a task's state.

In brief, the nine states are:

- *Initial*: This is the starting state of all tasks.
- *Pending*: The front-end will transition a task into this state to signal that the user wants the task to run. The task will stay in this state until the back-end has confirmed the successful execution of all pre-requisite tasks.
- *Queued*: When the task is ready to run, the back-end core will check if the actual execution back-end for this task is willing to accept more tasks. The task will remain in this state until the task can be run.
- *Running*: When the back-end starts a task, the task is moved to this state to signal that the task is running.
- *To Cancel*: The front-end will transition a task to this state to signal that the user has requested that the task be canceled.
- *Canceling*: The back-end will transition a task to this state to acknowledge that it is attempting to cancel the task.
- *Canceled*: The back-end will transition a task to this state when it has confirmed that the canceled task is no longer running.
- *Done*: This state means that the task has successfully completed.
- *Error*: If something goes wrong, the task is moved to the error state. For example, if a program for a task exits with a non-zero exit code, the error state will be used. Or, if the server crashes, tasks falsely marked as running will be set to error when the server restarts.

When a task is created, it is first placed in an *Initial* state. When a task reaches one of the final states, *Canceled*, *Done* or *Error* it can be 'reset'. When a task is 'reset', the results of that task are rolled back. That is, outputs for that task are deleted. After that, the record for the task in the database is replaced by a new task record, set in the *Initial* state. The old, previous record is retained in the database but marked as 'defunct'. This feature allows users to retry failed tasks without destroying or overwriting records of previously run tasks in the database. Also, each individually run task has a unique task id.

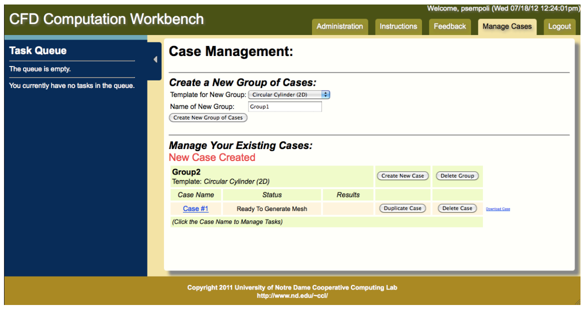
One particularly useful thing about this state system is that it helps prevent users from accidentally starting extra tasks due to their impatience or confusion. The command to start a task is not a command to create a new task. Rather, it is a command to transition from *Initial* to *Pending*. So, for example, if a user, in frustration, clicks the start button three times he will not start three tasks. The first command will perform the state transition. The second and third commands will try to perform the same transition on the same task, but will not be able to since the task is no longer in the *Initial* state.

V. Front-End

The front-end is programmed in PHP [11], running on an Apache webserver [12], and communicates with a mysql database [13]. This aspect of the front-end is fairly conventional. The front-end has several responsibilities in our system. It manages users, security and authentication, similar to a typical web application. It also performs file management by creating, deleting and otherwise controlling user files, which are organized into groups of cases. (In this system, a "case" is a collection of files for one simulation setup, but might involve several tasks needed to run and analyze that simulation.) The front-end also gives commands to the back-end, by way of a task database. The front-end creates records of tasks to be run, specifying the task to be run, how the task is to be run, and any command line parameters.

In order to make this system as extensible as possible, each group of cases for a user has a particular type, which is shared by all cases in that group. For a particular case type, a set of functions must be implemented as a PHP module to handle that type of case. This PHP code serves as the specific user web-interface for a particular type of case. Whenever a case is created, accessed or otherwise touched, the front-end will pass execution to the appropriate function. For most of these functions, a specific data structure containing most data about the current case is available to the function when it executes. Also, a number of helpful utility functions are implemented to make it more simple to place tasks in the database (and other common commands) with reference to the current case.

To demonstrate, in Figure 3, we have the case management screen just after a case has been created. The case is part of a group of cases using the Circular Cylinder template. The status line for the case indicates that it is ready to generate a mesh. (With the default values for the template.) Each case template module is required to have a function that gives this status. We click on the hypertext that says "Case #1" to see the screenshot in Figure 4. Each case template module is required to have


CFD Computation Workbench
Welcome, psempoli (Wed 07/18/12 12:24:01pm)
Administration
Instructions
Feedback
Manage Cases
Logout
Task Queue
Create New Group of Cases
Create New Case
Delete Group
Duplicate Case
Delete Case
Download Case
Copyright 2011 University of Notre Dame Cooperative Computing Lab
http://www.nd.edu/~ccl/

The queue is empty.

You currently have no tasks in the queue.

Case Management:

Create a New Group of Cases:

Template for New Group: Circular Cylinder (2D)

Name of New Group: Group1

Manage Your Existing Cases:

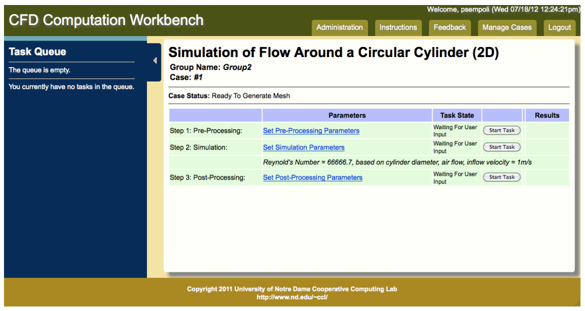
New Case Created

Group2
Template: *Circular Cylinder (2D)*

Case Name	*Status*	*Results*
Case #1	Ready To Generate Mesh	

(Click the Case Name to Manage Tasks)

Fig. 3. Case Management Screen just after a case is created


CFD Computation Workbench
Welcome, psempoli (Wed 07/18/12 12:24:21pm)
Administration
Instructions
Feedback
Manage Cases
Logout
Task Queue
Start Task
Copyright 2011 University of Notre Dame Cooperative Computing Lab
http://www.nd.edu/~ccl/

The queue is empty.

You currently have no tasks in the queue.

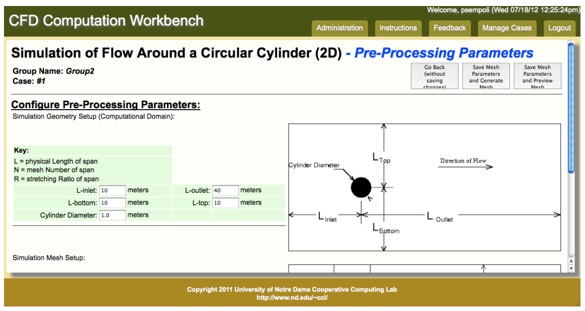
Simulation of Flow Around a Circular Cylinder (2D)

Group Name: ***Group2***
Case: ***#1***

Case Status: Ready To Generate Mesh

	Parameters	**Task State**		**Results**
Step 1: Pre-Processing:	Set Pre-Processing Parameters	Waiting For User Input	Start Task	
Step 2: Simulation:	Set Simulation Parameters	Waiting For User Input	Start Task	
	Reynold's Number = 66666.7, based on cylinder diameter, air flow, inflow velocity = 1m/s			
Step 3: Post-Processing:	Set Post-Processing Parameters	Waiting For User Input	Start Task	

Fig. 4. A Circular Cylinder Case, before anything is run


CFD Computation Workbench
Welcome, psempoli (Wed 07/18/12 12:25:24pm)
Administration
Instructions
Feedback
Manage Cases
Logout
Go Back (without saving changes)
Save Mesh Parameters and Generate Mesh
Save Mesh Parameters and Preview Mesh
Cylinder Diameter
L Top
Direction of Flow
L Inlet
L Outlet
L Bottom
Copyright 2011 University of Notre Dame Cooperative Computing Lab
http://www.nd.edu/~ccl/

Simulation of Flow Around a Circular Cylinder (2D) ***- Pre-Processing Parameters***

Group Name: ***Group2***
Case: ***#1***

Configure Pre-Processing Parameters:

Simulation Geometry Setup (Computational Domain):

Key:
L = physical Length of span
N = mesh Number of span
R = stretching Ratio of span

L-inlet:	10	meters	L-outlet:	40	meters
L-bottom:	10	meters	L-top:	10	meters
Cylinder Diameter:	1.0	meters			

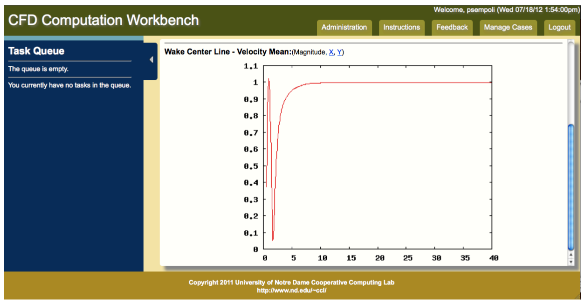
Simulation Mesh Setup:

Fig. 5. Mesh Parameters

a "display" function. Figure 4 shows this for the Circular Cylinder case type. In Figure 5, you can see part of the screen for adjusting parameters for the mesh. (The rest scrolls down.) From this screen we can set the mesh parameters and also request a mesh preview. Once we are satisfied with all of our parameters, in the main management screen of the case, (see Figure 4) we click the buttons to start all the tasks.

After a while, the various steps run and finish. If we were dissatisfied with the results of a step, we could click a "Reset Task" button to roll back the case to before that step. Then, we change our parameters and try again. We can view the results of all of the computation, in visual form. Figures 6 and 7 show a graph and an image from among the many available results for this kind of case.

This, however, is only a demonstration of one very simple type of case. The main advantage to the extensible, modular design for the front-end, especially with regard to our application in CFD for Civil Engineering is that we can create many different kinds of CFD templates with specific user interfaces

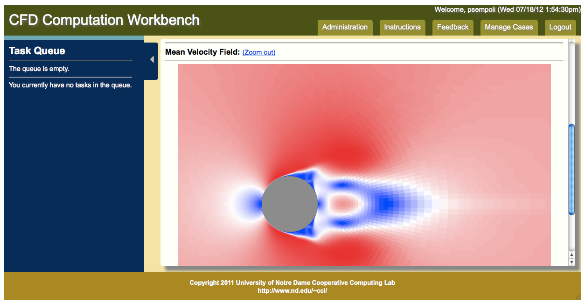

CFD Computation Workbench
Welcome, psempoli (Wed 07/18/12 1:54:00pm)
Administration
Instructions
Feedback
Manage Cases
Logout
Task Queue
Wake Center Line - Velocity Mean:(Magnitude, X, Y)
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
5
10
15
20
25
30
35
40

The queue is empty.

You currently have no tasks in the queue.

Copyright 2011 University of Notre Dame Cooperative Computing Lab
http://www.nd.edu/~ccl/

Fig. 6. A graph of some results. The data for the graph can also be downloaded


CFD Computation Workbench
Welcome, psempoli (Wed 07/18/12 1:54:30pm)
Administration
Instructions
Feedback
Manage Cases
Logout
Task Queue
Mean Velocity Field: (Zoom out)

The queue is empty.

You currently have no tasks in the queue.

Copyright 2011 University of Notre Dame Cooperative Computing Lab
http://www.nd.edu/~ccl/

Fig. 7. An image of a velocity field result. Note: This was a short simulation, so the flow only went a short distance.

for each kind of case. This allows for cases that hide most details from the user, or cases that show more details to the user. This is more helpful since we are interested in making CFD more user-friendly for applications in civil engineering.

The front-end also has a number of administrative tools implemented, which allow administrative users to monitor or change the tasks being dispatched and the overall state of the system, as well as manage users.

VI. CFD SUPPORT COMPONENTS

The role of the front-end is to produce files and database entires for various tasks. In order to do this, a number of extra components, in addition to the two main components, are used to facilitate the use of CFD in our system

A. Middle Manager

The middle manager is primarily designed for instances in which the case in question is controlled by a more user-friendly module in the front-end, which is hiding the more tricky details from the user. For some kinds of cases, this means that a large number of files must be managed. In particular, when a case is created, a number of files must be placed with default values. Similarly, sometimes, if a file set is complex, a single changed parameter results in changes to many files. Finally, if a task fails, then a user would want to roll back to before the task is run by deleting specific files. We found that placing this file management code in the front-end template modules (described above) was often unwieldy.

The middle-manager is a command-line program (written in python), with a number of functions in the front-end that can invoke it. This program works by reading from a set of directories of template files which guide its actions. Each template is comprised of a configuration and a set of guiding files. The middle-manager has two main kinds of functions. First, it allows variables to be set for a specific case. When a variable is set (for example, flow velocity), according to the configuration of the corresponding template, the text in the actual case files is changed. Second, if specific files are to be deleted in order to undo the running of a task, this tool can be told to simply undo a task by the name of that task. The middle-manager will use the template configuration files to determine which files to delete and then delete them.

B. Structured Mesh Generation and Preview Assistant

One of the more challenging aspects of working with CFD software, (in our case the OpenFOAM solver) is that the input files that define the mesh are often obscure. Of course, given the mesh complexity, one rarely produces the mesh by hand, but rather uses mesh generation tools based on a mesh dictionary to produce the mesh. Unfortunately, the inputs to this mesh generation can also be difficult to understand. Moreover, the mesh generation can be a long-running task, which means that we need to find ways of giving users a faster "preview" mesh. We solve both these problems by using a custom made library (written in python) which can generate a mesh dictionary for certain kinds of structured mesh generation and estimate the size of the mesh without running the full mesh generation.

In Figure 8 we show the relationship between the input to our structured mesh library, a mesh dictionary and a true mesh. The input to our library is a short python program, about ten lines at most, which gives a high level description of the simulation. This program calls functions in our library to define the object to be simulated, the size of the simulation area, the mesh properties near the object and elsewhere. There are a number of different objects to choose from, each with its own structured mesh configuration.

With all of this information, this mesh dictionary producing library creates data structures for the various blocks in the mesh dictionary. It insures that adjacent blocks are consistent in mesh number and ratio and estimates the number of mesh cells within a block. The sum of these is used to estimate the final mesh size and uniformly rescale the mesh if needed. The usefulness of this is in the desire for a mesh preview. Since the preview should not take too long, we set a maximum tolerable mesh size for a preview. If we want the full mesh, this library will produce the full dictionary. If we want the preview, this library will check the mesh size and rescale down if needed. As such, we can use the same parameters and programs to produce a full mesh or a preview.

The main achievement of this component is that, at least for the structured mesh types it understands, we are translating from more understandable geometric descriptions of the mesh, which are easier to visualize. So, in Figure 8, the high level description on the left gets translated by our mesh library functions to the block mesh dictionary in the middle. (Which

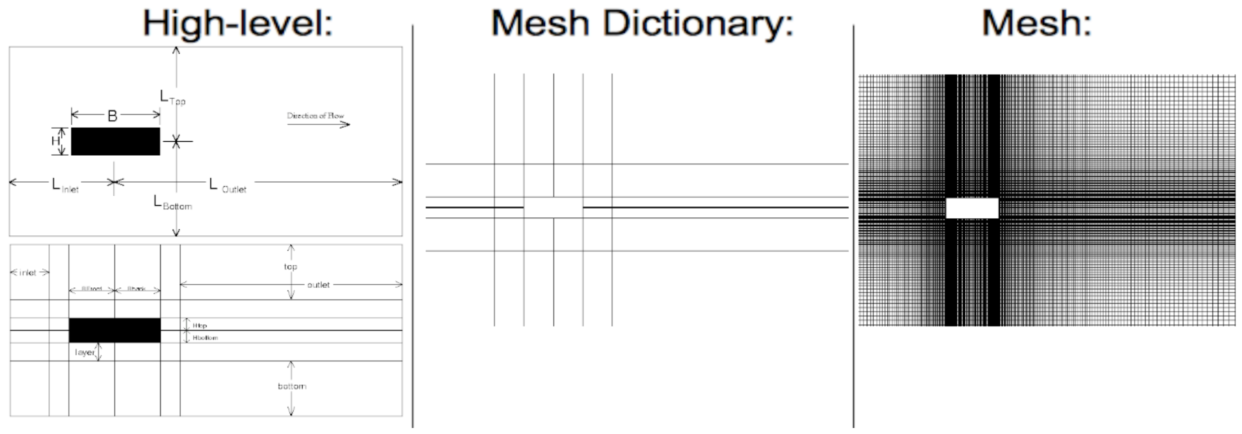


Fig. 8. Progression from a simulation idea to a mesh

we visualize by showing the mesh blocks.) The actual mesh generation software (which is in the CFD software package) performs the meshing to make the dictionary into a real mesh.

Of course, the downsize to this library is that it is limited to the specific structured meshes which it knows how to compute. Right now the program is constrained to a 2D mesh on a specific set of objects. We are researching other meshing tools to allow us more flexibility. (See the section on future work.) However, whatever new tools we use and incorporate, we are sure to preserve the rescale-for-preview idea of this mesh utility and the strength of being able to translate from a high-level description.

C. Execution Monitoring

One of our long-term goals for this project is to build a large degree of task monitoring to determine the correctness of CFD simulations in progress. To facilitate this, we have constructed a number of scripts designed for parsing the simulation output as it proceeds. So far, we have been interested in monitoring the simulated time and the Courant numbers. Monitoring the simulated time is useful because this is the key information needed to estimate simulation progress. Courant numbers quantify the number of mesh points across which flow moves in a single time-step. If the courant numbers are wrong, it means that either the mesh or the simulation time-step length need to be adjusted.

We have constructed some tools which pull the simulation time and the Courant numbers and process them as the simulation runs. We are still working on how to retrieve that information from various remote execution systems. However, if one is using the “local” execution environment, then our web portal reports estimated percentage completion and produces real-time self-updating graphs of the maximum Courant numbers of a simulation as it runs. We hope to expand this concept by retrieving this info from remote execution platforms as well and adding to the data which can be monitored.

D. Result Visualization

The best way to get a quick idea of how a simulation ran is to use some form of visualization. This includes both graphs and images of the various pertinent values in flow fields. While OpenFOAM comes with tools for such visuals, these are for desktop users with direct access to the data files. Since we are routing our visuals over the web, we needed to develop our own scripts for looking at various result files and producing graphs and images for the user. This data includes things like velocity fields, pressure fields, and graphs of force coefficients and graphs along the wake-stream line behind an object. Most importantly, this includes a visualization of the mesh. In order to make our data accessible to users, we developed some tools to produce these visuals, which can then be accessed by the front-end and displayed.

VII. Back-end Core

The back-end core is responsible for reading instructions from the database and starting or canceling jobs accordingly. Like the front-end, the back-end is built with a extensible design. The core of the back-end interacts with the database, procuring data about tasks to be run. The backend is designed to be able to interact with a variety of what we call “execution environments”. For our purposes in this paper, an execution environment is a way to run executable code. For example, our first execution environment simply ran the code locally on the same server as the back-end core. For each kind of execution environment, six functions must be implemented for the back-end core to use it. The six functions to be implemented are: *Initialize*, *Cleanup*, *IsFull?*, *IsDone?*, *Cancel* and *Start*. See the Table 1 for details. In this way, we designed the system to extensibly be able to run code in many different kinds of places. It is the front-end’s responsibility to specify a task type by setting this value in the database entry for that task. The back-end core, based on its configuration and the user permissions, will use that type to decide upon an environment.

We have implemented back-end execution environments for both local execution, (that is, running the process on the server) and dispatching to a system called Work Queue. The Work Queue system is a batch job system that runs on a classic master-worker setup, and has many different applications that have been built on or around it [14] [15] [16]. In our section on future work, we discuss our plan for even more such modules.

The back-end has a configuration file which is read whenever the server is started. This configuration specifies, among other things, the parameters for each desired execution environment. For example, our implementation that sends tasks to the Work Queue system includes a parameter for specifying the port of the Work Queue, and our implementation for running jobs locally includes a parameter for capping the number of tasks run at one time.

VIII. Our Experience

We have had two opportunities to test this system on unfamiliar users. In each case, we were both encouraged by the results and learned some lessons in how to make the system better.

A. First Version

The first version of the project was used to allow users to run a channel flow simulation. This provided the necessary tools for a case-study in 'crowd-sourcing' [17]. This was our first attempt at deploying the system with real users. At the time, many of the features described in this paper were not yet implemented. This was our first step in integrating a task control back-end with the CFD code and provided a great deal of insight into how to robustly design such a system. Our most important finding in this experience was the critical need to contain the number of threads running simultaneously on the server. Out of this experience came the *IsFull?* function in the back-end, the ability to set a task "cap" for the local execution environment and certain front-end enhancements to prevent users from starting too many tasks at once.

B. Second Version

In order to test the most recent version, we enlisted the aid of a group of students in a civil engineering class. These students were asked to perform some simulations of fluid motion over a circular cylinder. We consider this a reasonable proof-of-concept for the following reasons. First, while the environment was simplified, these relative beginners were still able to perform basic CFD simulations and produce real results. Second, this group of total newcomers to both CFD and this system were able to run their simulations without any complaints or crashes in the system.

We have also been improving since this second version. In particular, we note that the middle-manager component was added after this second trial.

IX. Future Work

The single most important avenue for expanding this project is to use the extensibility of the front-end and back-end to produce more modules for more applications. Since we have built up the core of our system and confirmed its reliability, we now intend to branch out into using it as a basis for more and more useful civil engineering applications. In this way, we can move toward our motivating goals of making this a true civil engineering toolkit of CFD. In order to do this, we will emphasize:

- Mesh Generation: Right now, we are confined to the specific set of meshes which we can produce for our mesh generation. In order to broaden our applications for our users, especially for importing custom geometries for analysis, we must incorporate better mesh tools and expand upon the mesh supporting tools which we have developed. In particular, we wish to use more varied mesh generation techniques, including unstructured mesh generation [2]. By gathering more mesh-making tools, we can perform mesh generation on complex or even arbitrary bluff bodies, making our system that much more useful to our intended civil engineering audience.
- Front-end Modules: We need more front-end modules to do more tasks. We are interested in developing tools to vary between beginners and experts, as well as allowing for more varied simulation types. Included in this development are plans to extend the capacity of the system to handle a greater variety of CFD analyses useful for civil engineering.
- Back-end Modules: In the back-end, we seek to develop a larger number of interfaces to execution platforms. In particular, we are interested in interfacing with platforms which will take advantage of the parallelism which is necessary for many CFD tasks. Also, we intend to consider a better mechanism for deciding which execution environment will run a particular task.
- Currently, data for this system is locally stored on the server. We were content with this naive approach since disk storage was not the focus of this project. However, going forward in this project we can improve by, first, exploiting some sort of large distributed file system and, second, incorporating mechanisms to enforce disk quotas upon users.
- Real-Time Visualization: In CFD, there is a great interest in being able to visualize a task as it is being run. It is also useful, absent a full visual, for some part of a task's data to be available for viewing as the task is being run. This data is helpful for estimating task completion time or for checking a partially completed simulation for correctness. (And canceling it if something goes wrong without having to wait for completion.) All these parts require the ability to 'query' a task in progress. We are considering the possibility of configuring batch job systems to allow such a query.
- Executable Dependancies: We find that insuring that the needed files, libraries, executables and environment variables are present in a distant environment is not always trivial across all execution platforms. Since no complex program runs in isolation, typically there are many dependancies. We would like to create an explicit mechanism for specifically defining the files and libraries that a task needs, and researching ways to insure that the same are always available.

TABLE I
FUNCTIONS TO IMPLEMENT FOR AN EXECUTION ENVIRONMENT IN THE BACK-END

	When function is called	Responsibility of function
Initialize	Once for each execution environment when the back-end process is started.	First, if the server failed in an inconsistent state, such as a power failure, tasks may be marked as being in the *Running* state when they are not running. The *Initialize* function must correct these and other inconsistencies for any tasks marked in the database as belonging to it. Second, the *Initialize* function must start any processes or obtain any resources needed to start jobs. Third, one input to this function is all the parameters for the execution environment which were set in the configuration file. This function must confirm that whatever values are required are both set and make sense. It must then record information from those configuration parameters.
Cleanup	Whenever the back-end server is shutting down	Attempt to cancel all running tasks that belong to this environment and gracefully free all resources which it has acquired.
IsFull?	Periodically, the back-end core will ask each back-end environment if it can take more tasks by calling the *IsFull?* function.	If the particular environment cannot take more tasks, (i.e. it is full) this function should return 1. Otherwise, it should return 0.
IsDone?	Periodically, the back-end core will ask each back-end environment if any of its tasks are completed.	This function should check if there are any running tasks which are now done or canceling tasks which are now canceled and, if so, return information about one of them. Also, it should retrieve files and the terminal output from remote locations to the correct place on the server.
Cancel	If a task is marked in the database for cancellation, this function will be called for the environment of that task and supply that task's information.	The task should be made to stop as soon as possible. If the task cannot be canceled immediately, then the *IsDone?* function should be able to recognize when the cancel is complete.
Start	When a task is ready to begin, this function is called, with the task data supplied.	Attempt to start a task. Insure that files are moved to the right remote location, if necessary.

X. CONCLUSION

The goal for this system was to devise ways to manage Computational Fluid Dynamics tasks so that we have a platform upon which we can prepare for users in civil engineering useful analysis tools. In order to do this, we developed a set of interlocking components for interfacing between users and back-end computation systems. Our system is designed to be as extensible as possible, while still being focused on CFD. This is with respect to both user interfaces for various types of CFD tasks in the front-end and flexibility regarding what kinds of systems can launch tasks in the back-end. In addition to these main components, we have created numerous supporting components for attending to various aspects of CFD workflows.

Our initial experiences have shown the core of our system to be stable and able to dispatch and control CFD tasks in a user-friendly way. From this, we hope to extend our system to incorporate a greater variety of components useful for analysis in civil engineering and attach to more powerful computational back-ends in order to run these tasks.

ACKNOWLEDGMENTS

We would like to thank John Quinn and Taylor Seale, who worked with us to test/break our code, and improve the overall appearance of our front-end.

This work was supported in part by National Science Foundation Grants OCI-1148330, CBET-0941565, CNS-0855047, and CNS-0643229. Their support is gratefully acknowledged.

REFERENCES

[1] C. M. Institute, "Navier-stokes equation," 2012. [Online]. Available: http://www.claymath.org/millennium/Navier-Stokes_Equations/

[2] P. Moin and J. Kim, "Tackling turbulence with supercomputer," *Scientific American*, vol. 276, no. 1, pp. 62–68, 1997.

[3] H. G. Weller, H. Jasak, and G. Tabor, "A tensorial approach to computational continuum mechanics using object-oriented techniques," *Computers in Physics*, vol. 12, no. 6, pp. 620–631, 1998.

[4] M. Russell and all, "The astrophysics simulation collaboratory: A science portal enabling community software development," in *10th IEEE International Symposium, High Performance Distributed Computing*, 2001, pp. 207–215.

[5] M. McLennan and R. Kennell, "Hubzero: A platform for dissemination and collaboration in computational science and engineering," *Computing in Science and Engineering*, vol. 12, no. 2, pp. 48–52, March/April 2010.

[6] M. Lundstrom and G. Klimeck, "The ncn: Science, simulation, and cyber services," in *IEEE Conference on Emerging Technologies - Nanoelectronics*, January 2006, pp. 496–500.

[7] Novotny and all, "Gridsphere: a portal framework for building collaborations," *Concurrency And Computation: Practice And Experience*, no. 16, pp. 503–513, 2004.

[8] S. Pota and Z. Juhasz, *Lecture Notes In Computer Science Computational Science – ICCS 2006*, 2006, no. 3991, ch. Supporting Interactive Computational Science Applications Within the JGrid Infrastructure, pp. 830–833.

[9] A. Kačeniauskas and R. Pacevič, "Vizlitg: Grid visualization e-service enabling partial dataset transfer from storage elements of glite-based grid infrastructure," *Journal Of Grid Computing*, pp. 573–589, 2011.

[10] "Cfd online." [Online]. Available: http://www.cfd-online.com/

[11] Php: Hypertext preprocessor. [Online]. Available: http://www.php.net/

[12] Apache http server project. [Online]. Available: http://httpd.apache.org/

[13] Mysql.com. [Online]. Available: http://www.mysql.com/

[14] A. Thrasher, Z. Musgrave, D. Thain, and S. Emrich, "Shifting the Bioinformatics Computing Paradigm: A Case Study in Parallelizing Genome Annotation Using Maker and Work Queue," in *IEEE International Conference on Computational Advances in Bio and Medical Sciences*, 2012.

[15] M. Albrecht, P. Donnelly, P. Bui, and D. Thain, "Makeflow: A Portable Abstraction for Data Intensive Computing on Clusters, Clouds, and Grids," in *Workshop on Scalable Workflow Enactment Engines and Technologies (SWEET) at ACM SIGMOD*, 2012.

[16] R. Carmichael, P. Braga-Henebry, D. Thain, and S. Emrich, "Biocompute 2.0: An Improved Collaborative Workspace for Data Intensive Bio-Science." *Concurrency and Computation: Practice and Experience*, vol. 23, no. 17, pp. 2305–2314, 2011.

[17] Z. Zhai, P. Sempolinski, D. Thain, G. Madey, D. Wei, and A. Kareem, "Expert-citizen engineering: "crowdsourcing" skilled citizens," *Dependable, Autonomic and Secure Computing, IEEE International Symposium on*, pp. 879–886, 2011.