2013-11-14

Developing an Architecture Framework for Cloud-Based, Multi-User, Finite Element Pre-Processing

Jared Calvin Briggs
Brigham Young University - Provo

Developing an Architecture Framework for Cloud-Based, Multi-User, Finite Element Pre-Processing

Jared C. Briggs

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

C. Greg Jensen, Chair
Chia-Chi Teng
Steven E. Gorrell

Department of Mechanical Engineering

Brigham Young University

November 2013

ABSTRACT

Developing an Architecture Framework for Cloud-Based,
Multi-User, Finite Element Pre-Processing

Jared C. Briggs
Department of Mechanical Engineering, BYU
Master of Science

This research proposes an architecture for a cloud-based, multi-user FEA pre-processing system, where multiple engineers can access and operate on the same model in a parallel environment. A prototype is discussed and tested, the results of which show that a multi-user pre-processor, where all computing is done on a central server that is hosted on a high performance system, provides significant benefits to the analysis team. These benefits include a shortened pre-processing time, and potentially higher-quality models.

Keywords: finite element analysis, fea, cloud computing, multi-user, collaboration, collaborative engineering, pre-processing, cae, cax, byu

ACKNOWLEDGMENTS

I would like to thank my advisory committee for their input and support throughout this research and my wonderful family and friends for their continued support and encouragement. Special thanks to the National Science Foundation and industry sponsors participating in the IU-CRC Center for eDesign at BYU, including Boeing, Pratt & Whitney, Belcan, PCC Airfoils, Spirit AeroSystems, and CD-adapco. I would also like to thank James Wu for his work on the C# networking code, Prasad Weerakoon, Tim Bright, and Ammon Hepworth for many stimulating conversations, and Brett Stone, Josh Coburn, and Alden Yellowhorse for their help creating and running tests.

TABLE OF CONTENTS

LIST OF TABLES . . . . . **vi**

LIST OF FIGURES . . . . . **viii**

NOMENCLATURE . . . . . **x**

Chapter 1 Introduction **1**
1.1 An Introduction to Analysis Pre-Processing 2
1.2 Research Motivation 3
1.3 Research Objective 4
1.4 Thesis Organization 5

Chapter 2 Background **6**
2.1 Current Commercial FEA Pre-Processors 6
2.1.1 Basic Pre-Processing Functions 7
2.2 Current Multi-User Methods and Implementations 8
2.2.1 Multi-User CAD 9
2.2.2 Multi-User FEA Pre-Processing 10
2.3 Current Utilization of Cloud-style Architectures in Engineering 12
2.3.1 HPC Use in Industry 13
2.3.2 Cloud-Based FEA Pre-Processors 13
2.3.3 Research into Cloud-Based, Multi-User Software 14

Chapter 3 Methods **16**
3.1 Overall Architecture 16
3.1.1 Client 18
3.1.2 External Connectivity 19
3.1.3 Queue 21
3.1.4 Server Processing Instance (SPI) 21
3.1.5 Controller and Multiple Available SPIs 23
3.1.6 Central Data Storage 24
3.2 Basic Pre-Processing Function in a Collaborative, Cloud Environment 24
3.3 Multiple Concurrent Projects 26
3.4 Simultaneous Command Execution 27
3.4.1 Parallel Meshing: Multi-Object, Multi-Thread 27
3.4.2 Other Parallel Commands 28

Chapter 4 Implementation **29**
4.1 MUFE (Multi-User Finite Elements) 29
4.1.1 Client 30
4.1.2 Server 32
4.1.3 Command Structure 32

4.1.4 Network Communication Methods . . . 37
4.1.5 Meshing and Other Functionality . . . 37
4.2 Small Server Implementation . . . 38
4.2.1 Client . . . 38
4.2.2 External Connection . . . 38
4.2.3 Queue, Controller, and Central Data Storage . . . 39
4.3 Cluster Implementation . . . 39
4.3.1 Client . . . 41
4.3.2 External Connection . . . 41
4.3.3 Queue and Controller . . . 41
4.3.4 Multiple Available Instances . . . 41
4.3.5 Central Data Storage . . . 42

Chapter 5 Results . . . 43
5.1 Multiple Simultaneous Projects . . . 43
5.1.1 Developing the Test . . . 43
5.1.2 Test Results . . . 46
5.2 Wing Section . . . 46
5.2.1 Developing the Wing Section Test . . . 46
5.2.2 Wing Section Test Results . . . 48
5.3 Symmetric Pressure Vessel . . . 49
5.3.1 Developing the Pressure Vessel Test . . . 49
5.3.2 Pressure Vessel Test Results . . . 49
5.4 Results Summary . . . 51

Chapter 6 Conclusions and Recommendations for Future Work . . . 53
6.1 Conclusions . . . 53
6.2 Future Work . . . 54
6.2.1 Parallel Algorithms . . . 54
6.2.2 Distributed Cloud Implementation . . . 54
6.2.3 Multi-User Server Performance Under Heavy Load . . . 54
6.2.4 Hybrid Architectures for Distributing Computation to Both Server and Clients . . . 55
6.2.5 Economic Costs and Benefits of Multi-User Collaboration in Engineering . . . 55
6.2.6 Decomposition Methods and User Editing Rights . . . 55
6.2.7 Conflict Resolution and General Multi-User Project Management . . . 56
6.2.8 Undo/Redo Commands . . . 56
6.2.9 Integration with Product Lifecycle Management (PLM) . . . 56
6.2.10 Integrating Multi-User Design with Multi-User Analysis . . . 56

REFERENCES . . . 57

Appendix A An Introduction to the Analysis Process . . . 60
A.1 Pre-Processing . . . 61
A.2 Solving . . . 63

A.3 Post-Processing 64
A.4 Other References for FEA and Related Methods 64

Appendix B MUFE User's Manual 66
B.1 Starting MUFE 67
B.1.1 Single-User Mode (No Server) 67
B.1.2 Utilizing a Remote Server 67
B.2 MUFE Control Overview 68
B.3 Model Import and Export 69
B.3.1 Importing a .STL File 69
B.3.2 Importing NURBS from an IGES File 69
B.3.3 Saving a MUFE Log File 70
B.3.4 Opening a MUFE Log File 70
B.4 Creating and Editing Geometry 70
B.5 Materials 71
B.5.1 Creating a Material 71
B.6 Meshes 71
B.6.1 Supported Mesh Types 71
B.6.2 Creating a Mesh 72
B.7 Load Collectors 72
B.8 Loads 73
B.8.1 Creating Loads 73
B.9 Constraints 73
B.9.1 Creating Boundary Conditions (Constraints) 74
B.10 Loadcases 74
B.11 Creating a Loadcase 74
B.12 Rigid 1D Elements 75
B.12.1 Creating a Rigid Element 75
B.13 Solving 76

LIST OF TABLES

4.1 Comparison of String Lengths Using JSON and XML Formats 37

5.1 Time Results from the Wing Section Test 49
5.2 Time Results from the Pressure Vessel Test 50

LIST OF FIGURES

2.1 Time Required to Collaboratively Complete a CAD Task 9
2.2 Time Required to Collaboratively Pre-Process a FEA Model 11

3.1 A Simplified, High-Level Framework 17
3.2 An Example Client Dataflow 18
3.3 An Example Validation Dataflow in the External Connection 19
3.4 An Example External Connection Dataflow (After Validation) 20
3.5 An Example SPI Dataflow 22
3.6 An Example Controller Dataflow 24
3.7 An Example High-Level Dataflow with Multiple Concurrent Projects 27

4.1 Client User Interface 31
4.2 Client Dataflow in Single-User Mode 32
4.3 Client Dataflow in Multi-User Mode 33
4.4 Dataflow in Server 34
4.5 JSON String Defining a New Material 35
4.6 JSON String Describing a Mesh Command 35
4.7 Part of a JSON String Defining a Newly-Created Mesh 36
4.8 High-Level Dataflow of the Entire System 40

5.1 Screenshot of the Server from the Multi-Project Test 44
5.2 Screenshot of a Client from the Multi-Project Test 45
5.3 Beginning Flat Sheet Geometry for the Wing Section Test 47
5.4 Final Shaped Geometry for the Wing Section Test 48
5.5 Final Model for the Wing Section Test 48
5.6 Average Time Results for Each Test Type Performed for the Wing Section Test . . 50
5.7 Final Shaped Geometry for Pressure Vessel Test 51
5.8 Final Model for Pressure Vessel Test 51
5.9 Average Time Results for Each Test Type Performed for the Pressure Vessel Test . 52

A.1 An Asymetrical Collision Analysis in FEA 60
A.2 Example CAD Geometry of the Liftfan for the USAF F-35B 62
A.3 Example of a Car Body that has been Discretized into a Mesh 62
A.4 A Simple Pre-Processed Part 63
A.5 Image of Analysis Results Showing Fluid Vorticity due to Jet Engine Blades 65

B.1 MUFE User Interface 66

NOMENCLATURE

$2D$	Two-dimensional
$3D$	Three-dimensional
A	An example stiffeness matrix
API	Advanced Programming Interface
b	An example force vector, or a matrix composed of several independent force vectors
BYU	Brigham Young University
CAD	Computer-Aided Design, including methods and tools
CAE	Computer-Aided Engineering (analysis), including methods and tools
CAM	Computer-Aided Manufacturing, including methods and tools
CAx	Computer-Aided methods, including CAD, CAM, CAE, etc.
CC	Collaborative Command
CFD	Computational Fluid Dynamics
FDM	Finite Difference Method
FEA	Finite Element Analysis
FEM	Finite Element Method
FVM	Finite Volume Method
GB	Gigabyte
GHz	Gigahertz
GUI	Graphical User Interface
HPC	High Performance Computing, including architectures such as clusters, server farms, supercomputers, cloud infrastructures, etc.
$HTTP$	Hypertext Transfer Protocol
$HTTPS$	Hypertext Transfer Protocol Secure
IP	Internet Protocol
$ITAR$	International Traffic in Arms Regulations
IGA	Isogeometric Analysis
$IGES$	Initial Graphics Exchange Specification
$JSON$	JavaScript Object Notation
KB	Kilobyte
LAN	Local Area Network
MB	Megabyte
$MUFE$	Multi-User Finite Element pre-processor
n	Number of an entity applied to a task (may include users, processors, threads, etc.)
NAS	Network Attached Storage
$NURBS$	Non-Uniform Rational B-Spline
PD	Product Design, or the overall process of Product Design
PLM	Product Lifecycle Management
RAM	Random Access Memory
SPI	Server Processing Instance
$STEP$	Standard for the Exchange of Product model data
STL	Stereolithography file format
t	Time required to complete a task (status quo if no subscript)
TCP	Transmission Control Protocol

UI	User Interface
VM	Virtual Machine
x	An example displacement vector, or a matrix composed of several indepenedent displacement vectors
XML	Extensible Markup Language

Subscripts and superscripts

$[\,]_1$	Original amount (status quo)
$[\,]_n$	Amount after n number of entities have been applied to the particular task

CHAPTER 1. INTRODUCTION

As part of any modern product design method, it is common practice to analyze a product before it is produced in order to predict whether it will perform according to the required design criteria. Products are commonly analyzed for conditions such as stress, strain, deflection, failure, heat and mass transfer conditions, etc. Unfortunately, the vast majority of products have geometries, materials, loads, environments, or a combination of these that are too complex to accurately model using simple, closed-form solutions. Therefore, methods have been developed to facilitate the numerical approximation of these products, thus enabling the computational simulation of their real-world performance. These include the Finite Difference, Finite Volume, and Finite Element methods, the uses of which are often very time-intensive—yet critical—in the product design process. Throughout this paper, these methods will be referred to as Finite Element Analysis (FEA) tools and processes, or simply, as the analysis process.

Modern engineering applications that facilitate the use of FEA methods are strictly single-user [1]. This means that only a single engineer can work on an analysis model at a time, even in cases where the model is extremely large or consisting of many parts. For example, when a product is designed, a Computer-Aided Design (CAD) geometry model is delivered to a single analyst who defeatures it, defines an appropriate mesh, refines the mesh, applies realistic boundary conditions, and then submits the simulation model to be solved—often to a High Performance Computing (HPC) cluster or other cloud-type environment. This work introduces an architecture framework that allows multiple engineers to efficiently work on the same FEA model at the same time, thereby shortening design cycle times.

1.1 An Introduction to Analysis Pre-Processing

The vast majority of the time in the analysis phase is spent building and refining the FEA model (i.e., "pre-processing")[1]. According to a survey performed at Sandia National Labs, about 73% of the time spent developing an analysis model is consumed in this stage [2]. Regardless of the software used, the main steps involved in pre-processing include: importing geometry, discretizing the geometry ("domain"), defining material properties, and applying loads and boundary conditions [3, 4].

1. Importing geometry may include reading a kernel-neutral file-format (such as IGES or STEP), or a kernel- or CAD-specific format (such as Parasolid®[2], ACIS[3], NX™[4] part file, etc.). It is often necessery to simplify (or "clean up") this geometry by removing small or insignificant artifacts such as fillets, chamfers, small holes, non-structural components, and any other artifact that does not contribute to the intent of the analysis.

2. Discretizing the domain (also known as "meshing") is done in order to break up the complex geometry into smaller shapes ("elements") whose closed-form solutions are known, thus rendering the problem easier to solve by summing the solutions of all of the smaller problems. Discretization creates both nodes and elements, where elements are made up of neighboring groups of nodes. If an automatic meshing algorithm is used, it is often necessary for the user to manually fix elements that are either too small, too large, or whose geometry may render them numerically unstable. In a large model, this process can take a considerable amount of time.

3. Material properties are applied to the elements (be it individually, to a subset of elements, or to all elements in the mesh). These properties may include Young's Moduli, Poisson's Ratios, Density, etc. of the material with which each element is associated. Section properties (beam, shell, solid) are also defined.

[1] For those unfamiliar with Solving or Post-Processing in FEA, or for more information about the overall FEA process, see Appendix A.

[2] Parasolid is a registered trademark of Siemens Product Lifecycle Management Software, Inc.

[3] ACIS is owned by Spatial Corporation.

[4] NX is a trademark of Siemens Product Lifecycle Management Software, Inc.

4. Loads and boundary conditions are applied. Loads may represent instances of structural loads, displacements, temperatures, pressures, heat/mass flows, etc., and boundary conditions are used to signify phenomena such as constraints (places of zero displacement), and energy/heat sinks. This process can also be quite time-consuming if the analysis requires a complex loading scenario, or a large number of loading scenarios.

This very serial workflow causes the single-user preparation of analysis models to frequently become a substantial bottleneck in the product design process [5]. Even though much research has been done to expedite the overall analysis process, little attention has been paid to enabling multiple concurrent users in the same pre-processing environment. It is the preprocessing phase that holds the greatest potential for benefit gains from the research presented in this thesis; therefore, this document does not propose any additions or modifications to the solver or post-processor display phases.

1.2 Research Motivation

Modern computing power allows researchers and engineers to analyze much larger models than in the past. For example, many recent simulations have in excess of tens of millions of nodes, something that was much less common only a few years ago. Analyses this large are frequently very difficult to set up, on account of the substantial number of manual corrections that need to be made before the simulation is ready to solve, as well as other complications that are inherent with complex geometries, materials, and loading conditions. A significant bottleneck in the analysis process is the fact that today's FEA software allows only a single user in a model at a time. This inhibits the analysis team from working cooperatively on the same model. Parallelizing this process so that multiple engineers can work together would expedite this time-sink, thus shortening the overall product development cycle. Current research into collaborative engineering methods focuses on CAD software, and although many of the same principles can be applied to analysis modeling, the architectural differences of these two multi-user environments are significant.

In order to effectively utilize a multi-user pre-processor, the analysis team must standardize and follow accepted modeling practices and techniques. For example, modeling conflicts can be minimized if each team member works in a pre-defined section of the model (domain), or fills

an assigned functional role. Doing so in a multi-user environment, these considerations will help teams reduce modeling time by a significant amount. Also, due to the fact that the modeling load is shared among users, it is expected that the final simulation quality will be greater than what is currently achieved.

Multi-user CAx systems generally require significantly more resources in a more reliable architecture than single-user systems. This is due to the linear increase in command frequency resulting from the increased number of users per project, and is especially noticeable in FEA pre-processing. Because of the high command-execution load associated with multi-user applications (FEA processes in particular), a multi-user FEA pre-processor would be ideally implemented on a cloud-type environment, enabling a high degree of reliability and accessibility for geographically distributed teams. This type of architecture would allow for dynamic scalability of system resources as the number of users on the project changes, and would reduce the amount of resources required on client machines.

1.3 Research Objective

FEA pre-processing is the most time- and labor-intensive step in the entire analysis process. The duration of this step can be significantly shortened if the workflow is parallelized in a way that allows multiple users to work collaboratively. No commercial FEA pre-processing software currently has this capability.

This research proposes a finite element pre-processing architecture that includes a centralized, cloud-based server where all commands are executed in a dynamically-scalable system. Aspects of this framework will be demonstrated by creating and implementing a simple, multi-user pre-processor prototype on a networked system, where all commands are executed in parallel on the server. The author originally intended to utilize the Microsoft®[5] Azure Cloud as the host for the server; however, due to the fact that many engineering companies already host their own High Performance Computing (HPC) data centers, and may be hesitant to trust their data to cloud services, the architecture presented in this research will be prototyped and tested using an in-house cluster-style environment. It is important to note that this architecture could also be implemented

[5]Microsoft is a registered trademark of Microsoft Corporation.

in a public or private cloud with little or no modification, which would make it available to companies who don't have the in-house resources to host such a system. For the purposes of this paper, the term "cloud" is used to include implementations in public and private clouds, HPC systems (including clusters), and other distributed architectures.

The main objectives for this thesis are:

1. Design a framework to facilitate cloud-based, multi-user pre-processing.
2. Create a simple, multi-user FEA pre-processor to test the framework.
3. Implement and test this software on a local, networked system.

1.4 Thesis Organization

This research couples new methods in multi-user applications with existing concepts for cloud-based computing. The prior research and utilization of these technologies as well as a discussion regarding their relation to the current work is presented in Chapter 2. Chapter 3 gives a general overview of the methods and architecture used for this research. A detailed description of the prototype implementation and design rationale are included in Chapter 4. The results of a demonstration of the implementation are presented in Chapter 5. Chapter 6 discusses conclusions drawn from the research, as well as suggests possible future efforts to further this work.

CHAPTER 2. BACKGROUND

Before the advent of engineering software such as computer-aided design (CAD), computer-aided engineering (CAE), and other computer-aided technologies (CAx) the standard modes of engineering design allowed for a great deal of collaboration. However, from the inception of CAx software, this collaboration has been greatly limited by the single-user nature of the software. Interest and research into applying collaborative methods to modern CAx software has recently increased.

Also, the use of High Performance Computing (HPC) and other similar cloud-type architectures in engineering is increasing in popularity due to their speed, high reliability, easy access, and relatively low cost. These systems are already being used to increase workflow for resource-intensive operations (such as solving FEA models), and could be leveraged for other engineering purposes.

2.1 Current Commercial FEA Pre-Processors

Many commercially-available FEA pre-processors exist and are in wide use in industry. These include ANSYS®[1] Mechanical™/CFD™/Fluent®/Structural™, Altair®HyperMesh®[2], Siemens®[3] Femap™[4], CUBIT™[5], and Simulia®[6] Abaqus/CAE™[7], as well as several that are integrated

[1] ANSYS is a registered trademark of ANSYS, Inc.
[2] Altair and HyperMesh are trademarks of Altair Engineering, Inc.
[3] Siemens is a registered trademark of Siemens AG.
[4] Femap is a trademark of Siemens Product Lifecycle Management Software, Inc.
[5] CUBIT is a trademark of Sandia National Laboratories.
[6] Simulia is a registered trademark of Dassault Systémes.
[7] Abaqus/CAE is a trademark of Dassault Systémes.

within CAD systems such as Siemens NX™[8], CATIA®[9] SIMULIA™[10], and Autodesk® Inventor®[11].

All commercial FEA pre-processors have similar basic functionality (discussed in Section 2.1.1), although levels of accuracy and granular control can vary widely among platforms. Some systems are designed toward seasoned analysts, allowing more low-level manipulation and refinement. Others are intended for more high-level use by designers, automating much of the usually tedious pre-processing operations in the background at the sacrifice of direct fine control and, potentially, some quality. Many pre-processors also support a variety of solvers, whereas some are specific to a particular analysis code.

2.1.1 Basic Pre-Processing Functions

These systems all perform the same general pre-processing operations including the following:

- Import geometry: this may be in the form of a generic file-format (IGES, STEP, or STL), or direct import from either a CAD system or kernel database file.
- Edit geometry: this includes the ability to remove fillets, chamfers, small holes, and other extraneous geometry.
- Apply materials: including linear, nonlinear, and temperature-dependent isotropic, anisotropic, and orthotropic materials. Depending on the pre-processor, these may be applied to either the geometry or directly to the mesh.
- Discretize geometry: many mesh and element types are available. Some common element types are 1D rigids, bars, beams, and springs; 2D triangles and quadrilaterals; and 3D tetrahedra, hexahedra, and pentahedra (prisms or pyramids). Most modern systems support at least first- and second-order meshes (i.e., linear and quadratic elements).

[8] See Chapter 1, footnote 4.
[9] CATIA is a registered trademark of Dassault Systémes.
[10] See footnote 6.
[11] Autodesk and Inventor are registered trademarks of Autodesk, Inc.

- Edit mesh: meshes can have various operations performed on them (either globally or locally) including refining or simplifying the mesh, changing element type or order, adding/editing/deleting nodes and elements, and moving and connecting/disconnecting nodes.

- Apply loads and boundary conditions: loads and constraints (boundary conditions) are usually applied to nodes; however, most software allows these entities to also be applied to geometry, which then distributes the load or constraint to the associated finite element nodes.

2.2 Current Multi-User Methods and Implementations

Many types of commercial collaborative technologies have been developed, such as screen or application "sharing" software like WebEx®[12], Product Data Management (PDM) and Product Life-cycle Management (PLM) software including Siemens Teamcenter™[13], shared document editing software like Google Docs™[14], and others. Much research has been done in an attempt to reduce the amount of time required for user-intensive CAx operations by parallelizing the workflow and allowing multiple users to work concurrently on the same model. However, no known commercial engineering software exists that allows multiple users to simultaneously edit the same file. Past research into various multi-user CAx systems has shown that increasing the number of engineers in a particular model can greatly decrease production time [1, 6–8]. Several prototypes have been made that successfully demonstrate this in CAD, and even a few in the CAE/FEA pre-processing realm.

The need to create a collaborative FEA pre-processing environment has been prevalent for some time. A paper from Sandia National Laboratories stated in 2001 that FEA pre-processing "dominates the analysis time, accounting for over 90 per cent of that time in some cases" [9]. This issue can be traced back many more years. For example, in 1995 a joint paper between researchers at Brigham Young University and Sandia pointed to the fact that pre-processing is "a time consuming and often difficult manual task" [10]. Now, almost two decades later, this need is still very real, as stated by Rashed [11], Weerakoon, et al. [6, 7], E. Red, C. G. Jensen, et al. [8],

[12] WebEx is a registered trademark of Cisco Systems, Inc.
[13] Teamcenter is a trademark of Siemens Product Lifecycle Management Software, Inc.
[14] Google Docs is a trademark of Google, Inc.

Roensch [12], Yang [13], and others. It is the pre-processing phase that holds the greatest potential for benefit gains from the research presented in this thesis.

2.2.1 Multi-User CAD

Recent multi-user prototypes (including Brigham Young University's NX-Connect and CATIA-Connect) have been developed that show the reduction in modeling time attained by multiple, concurrent users. For example, if a CAD task takes one engineer t_1 time to complete, and n number of engineers work collaboratively to accomplish that task, then the amount of time t_n ideally required to finish that task could be expressed with the following equation:

$$t_n = t_1/n \tag{2.1}$$

Let's imagine that a particular project takes t_1 hours for one engineer to complete. If we have n engineers working to complete this same task, then the time ideally required to complete this task can be modeled as shown in the figure below.

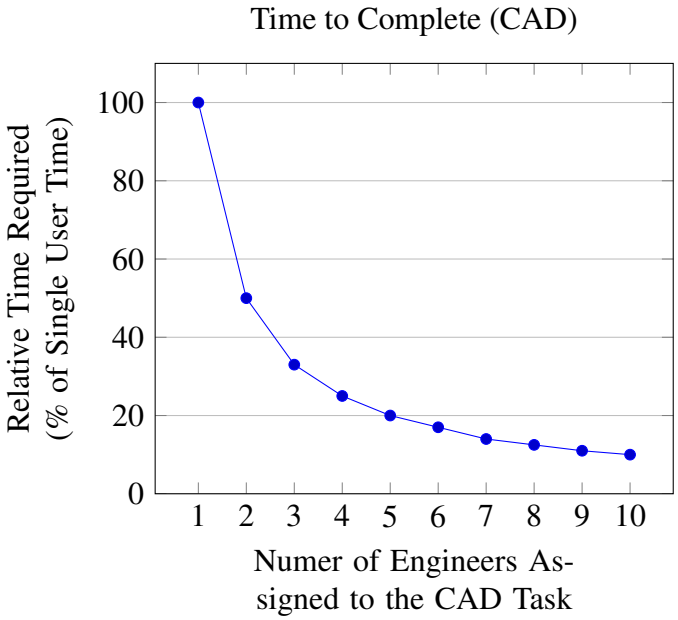

Figure 2.1: Time Required to Collaboratively Complete a CAD Task

It is important to note that even though Figure 2.1 illustrates a significant reduction in modeling time per engineer added to a collaborative CAD task, due to the law of diminishing returns, there will come a point where adding more engineers to the task will not further reduce the modeling time. The number of engineers at which this point comes into effect can be influenced by the size and complexity of the model, and the collaborative techniques utilized by the design team. This research does not consider the cause or effects associated with this phenomenon.

The current multi-user CAD prototypes require each command to be executed on every user's workstation [1, 9]. This means that as more engineers participate in the multi-user session, more commands must be executed on each machine per unit of time. Since CAD operations often require little time to execute, this method may be acceptable for multi-user CAD prototypes.

2.2.2 Multi-User FEA Pre-Processing

As mentioned earlier, a vast majority of the time spent in the analysis phase of product design is spent merely building the model (i.e., "pre-processing"). According to a survey performed at Sandia National Laboratories, about 73% of the time spent on an analysis is consumed in this stage [2]. The remaining time is used solving the model and interpreting the results (i.e., "post-processing").

FEA pre-processing algorithms can take considerably longer to execute than most CAD methods. With that in mind, let's assume that FEA pre-processing consumes 73% of the overall time t_1 required for a single user to complete a finite element analysis, and n engineers are engaged in collaboratively pre-processing the model. Then, the ideal time t_n needed to complete a full analysis for a project can be modeled.

$$t_n = t_1 \left(\frac{0.73}{n} + 0.27 \right) \tag{2.2}$$

Let's recall the example project from Figure 2.1, and assume that 73% of the project time is now spent pre-processing the model. Then, when we assign n engineers to the model-preparation stage, we again see a significant decrease in the overall time required to complete the project, as is shown in Figure 2.2.

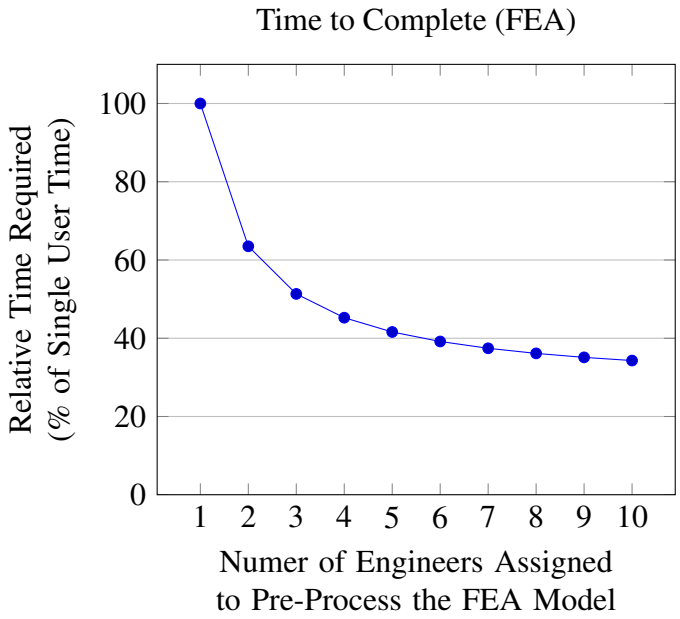

Figure 2.2: Time Required to Collaboratively Pre-Process a FEA Model

However, we note that for each engineer that we have pre-processing the FEA model, the relative time-savings is less drastic than it would be with the same number of engineers assigned to a CAD task. For example with seven engineers assigned to pre-processing, the ideal overall time required to complete the entire process (solving included) is 37% of the original, single-user pre-processing time. However, if seven engineers work together on a CAD task, than the task could ideally be accomplished in approximately 14% of the single-user CAD time. This difference is due to the fact that multi-user processes can only speed up the portion of the overall time that is required to create the FEA model, not to solve it. This phenomenon is known as Amdahl's law [10]. However, it is important to note that a cloud-based multi-user pre-processing server can access and utilize a distributed solver, which would then have the potential to shorten the solving time as well. Although the benefits of such a system are logical, this research will focus on parallelizing only the pre-processing phase of analysis.

Again, it is necessary to note that the law of diminishing returns will ultimately limit the effect of each additional engineer in relation to the complexity of the model and the collaborative techniques employed by the modeling team. Therefore, the trend shown in Figure 2.2 might not be fully realistic after a certain point, even in ideal cases. This issue will not be examined in this research.

Procedures in FEA pre-processing often require substantially longer amounts of time to execute than those in CAD. These processes include commands such as geometry cleanup, meshing, re-meshing, mesh morphing, etc., and can often take from a few seconds to many minutes to complete. Similar to the current multi-user CAD architectures, existing multi-user FEA pre-processing prototypes (like Brigham Young University's CUBIT-Connect) require all commands to be executed on each machine [6]. Therefore, as more engineers participate in a multi-user pre-processing session, more of these often-time-consuming commands must be executed on each user's computer. This can cause the clients' modeling GUIs to become unresponsive for extensive durations of time. For example, if a group of engineers were using BYU's CUBIT-Connect prototype, and several of them simultaneously meshed complex parts, then the corresponding mesh commands would be propagated to all of the other computers, making the client machines unresponsive until they completed executing all of the commands.

Furthermore, it was found from the author's HyperMesh-Connect prototype that even though a mesh command is sent with the same parameters to all clients in a multi-user project, the resulting meshes can be slightly different on each machine [14]. This may be due to the fact that some meshing algorithms use a random seed to determine where to start meshing or where nodes should be placed [15, 16]. Some problems resulting from this anomaly may include differing numbering sequences among the clients, loads that may sum to zero on one client but not on another, and other issues. These can become severe problems when dealing with tasks such as free-body modeling, topology optimization, etc.

2.3 Current Utilization of Cloud-style Architectures in Engineering

High Performance Computing (HPC) and other cloud-type architectures are commonly used today in engineering for solving large numerical problems. Many companies currently have in-house servers, server farms, or clusters on which they may run proprietary software and simulations. Historically, these machines have been used mainly for solving numerical problems and data-mining. More recently, small sections of these computers may have been allocated to perform other roles. For example, the advent of systems such as HP's Remote Graphics Software (RGS) allows a user to remotely connect and utilize the resources of a server as if it was a desktop

workstation [17, 18]. This setup emulates the terminal-mainframe configuration of past computing systems.

2.3.1 HPC Use in Industry

The most common uses of HPC-type architectures in industry include anything that requires highly calculation-intensive algorithms, such as computing the numerical solutions to large models, data-mining large data sets, gene sequence matching, molecular modeling, weather forecasting, oil and gas exploration, etc. These systems are often accessed remotely via the local network or internet, with the user submitting a job, logging off, and periodically checking until the job is finished. Some jobs can last anywhere from a few minutes to days to weeks, depending on the job type, the system running the job, and the system resources allocated to the job.

These architectures often allow for little user interaction. In the case of Finite Element modeling, all pre-processing is typically done on the user's local machine before the model is submitted to be solved on one of these systems. Then, after the solution is found, the user must download the results files (which can total in excess of terabytes) to his or her local machine for post-processing.

2.3.2 Cloud-Based FEA Pre-Processors

Several software programs have been developed recently that allow a user to pre-process FEA models on the cloud. However, these are often limited in model size and geometry type. One example of this is NEi™[15] Stratus, which allows someone to use a menu-based interface on an iPad® or iPhone®[16] to specify primitive shapes, meshing parameters, material properties, and basic loading and constraint conditions for an FEA model. These parameters are then submitted to the NEi cloud, where the model is created and solved. Static images of the resulting contour plots are then sent back to the user's device for viewing [19, 20].

Japanese software company Fujitsu™[17] developed the Engineering Cloud, an environment where a user can connect to a remote server, and perform FEA pre-processing on the server [21,22].

[15] NEi is a trademark of NEi Software, Inc.
[16] iPad and iPhone are registered trademarks of Apple Computer, Inc.
[17] Fujitsu is a trademark of Fujitsu Limited.

Fujitsu originally developed proprietary CAD, FEA, and Product Data Management (PDM) software. Then, the company integrated their software into their Engineering Cloud, thus enabling manufacturers to access design and analysis software from anywhere with an internet connection, even on low-resource machines. Fujitsu's Engineering Cloud utilizes Remote Virtual Environment Computing (RVEC), a proprietary high-speed image compression technology, to pass model viewing data to the user [23]. This means all data is retained on the server machines, allowing the client to function without the risk of overloading its local resources.

The computational fluid dynamics (CFD) software by CD-adapco, STAR-CCM+™[18], was built on a client-server architecture, which means that the pre-processing computation can be done remotely on a server, with the visualization done locally on the client [24]. This requires fewer resources on the client than would be needed otherwise.

While these systems are a step in the right direction, they still only allow a single user to execute one pre-processing command at a time on the server, thus limiting the systems' collaborative effectiveness. The research presented here proposes utilizing a dynamic system to host the server, allowing it to execute incoming pre-processing commands in parallel, thus enabling an efficient multi-user pre-processor.

2.3.3 Research into Cloud-Based, Multi-User Software

Over twenty years ago, it was stated that the ideal mode of engineering collaboration would be "a network of computers/users" that share and pass information to one another through a centralized server [25]. There have been architectures proposed for multi-user CAx that utilize a single server as the system's main compute point. However, the failure of this server would disrupt work for all users who utilize it [26]. This would mean that if something happened to the server which inhibits its proper function, all collaborative progress would stop, and past data could potentially be lost. Therefore, a single centralized server that is based on a standard client-server architecture, upon which resides the entire load of the multi-user projects, would not be an ideal solution for this type of application.

Cloud-based systems have access to a larger amount of more integrated hardware than most other architectures, thereby allowing higher redundancy accross many aspects of the system. The

[18] STAR-CCM+ is a registered trademark of CD-adapco.

availability and accessibility of a multi-user FEA pre-processing environment could be enhanced by placing it in a cloud-type architecture, thus allowing for high up-time and easy accessibility from potentially anywhere with an Internet connection. This would require that all commands be executed only on the server, which would then propagate the resulting data to all of the clients. Weerakoon, et al. [6] stated that in addition to the innate advantages of multi-user CAx, the benefits of a cloud-based architecture for a finite element pre-processor would include the following:

1. Consistent numbering system (all clients are working with dependably identical models)
2. Thinner clients (low client-side resource requirement)
3. Scalability (enabling practically an unlimited number of engineers to work in the same multi-user session)

No prototype is known to actually utilize the cloud for such a purpose. One U.S. patent has been granted for a cloud-based chat system that enables users to interact in a virtual space using avatars [27].

The author believes that this research will advance the state of the art in such a way that companies who develop the leading FEA pre-/post-processors will take notice. Furthermore, it has been communicated to the author by several industry professionals that this type of technology would be revolutionary and is in great need. These professionals include engineers and engineering managers from Altair Engineering, LSF Design Engineering, Alliant Techsystems Inc. (ATK), General Atomics Aeronautical Systems, Pratt & Whitney, and others.

CHAPTER 3. METHODS

In addition to standard FEA pre-processing operations, a cloud-based, multi-user pre-processor requires several capabilities that are not commonly found in today's commercial packages. These include the ability to support several concurrent multi-user sessions (completely separate and independent projects) residing and operating in the same hardware; the capability to simultaneously execute any number of commands without freezing the clients or their User Interfaces (UIs); as well as the inherent high availability, accessibility, and scalability provided by the cloud architecture.

This chapter describes an architecture that enables multi-user pre-processing. The proposed system utilizess a dynamic number of processing servers, thus allowing execution of commands in parallel without freeziing the clients' user interfaces.

3.1 Overall Architecture

The ideal multi-user FEA pre-processor utilizes a strong, central, cloud-based server where all commands are executed, and which sends all resulting data to the clients. The server's residing on the cloud allows it to have dynamic scalability and ready access for its users.

In order to most fully utilize the cloud-architecture, the pre-processing framework includes (shown in Figure 3.1):

- A thin-client, where the user views the model and inputs commands
- A network connection through which the clients connect to the cloud server
- A method for validating users and their project assignments and rights
- A queue that transfers received commands to the Server Processing Instances (SPIs)
- Another queue that collects resulting data from the SPIs for distribution to the clients

- A controller, which creates and destroys compute instances when necessary
- A database, or some other form of central, shared data repository

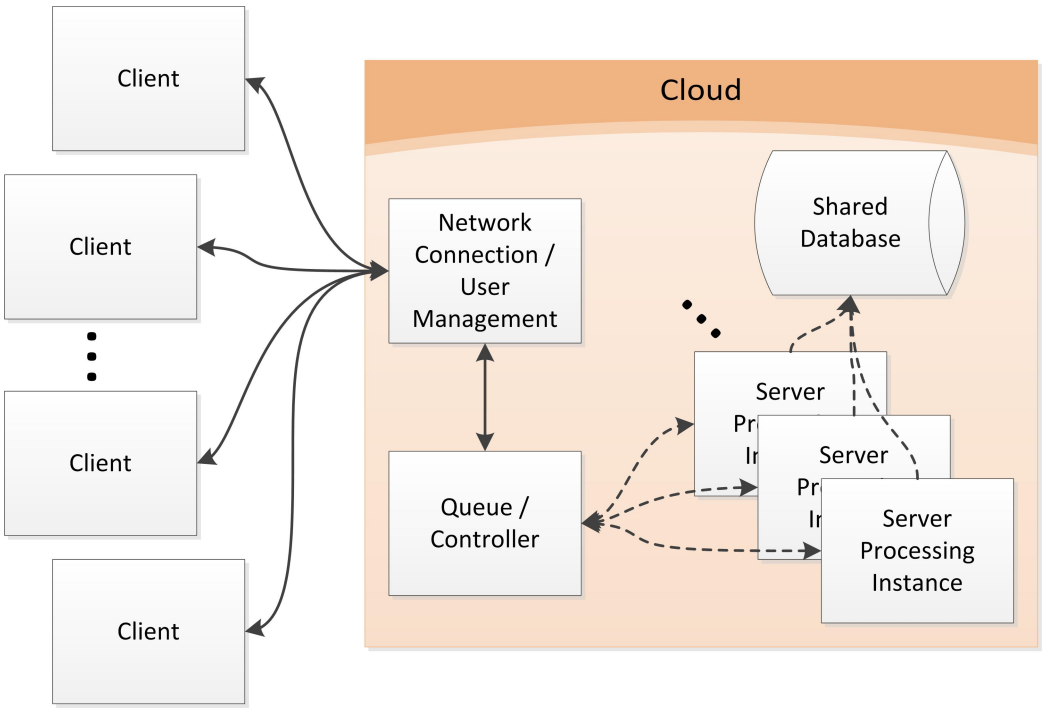

Figure 3.1: A Simplified, High-Level Framework

The cloud consists of a number of machines (either virtual or real), each dedicated and specialized to a particular task. These machines (sometimes referred to as ”nodes”) contain all of the software (libraries and applications) necessary to complete their assignments, communicating with each other as necessary to accomplish the assigned task.

When demand increases to the point that there are fewer than a defined number of “waiting” SPIs at any one time, the system will create new SPIs to accommodate. Furthermore, when demand decreases, resulting in more non-productive SPIs than is preferred, the system will destroy the SPIs that are no longer needed. This process ensures a more optimal balance of performance with resource use.

3.1.1 Client

The client exists locally in the form of a thin-client, which is a piece of software that relies on another computer (in this case the server) to function [28]. This client receives and displays the data given by the server, collects inputs from the user and packages them as commands, and sends these commands to the server. It also listens for data coming from the Server, and updates the local database with the new information as appropriate. An example of dataflow in the Client is shown in Figure 3.2.

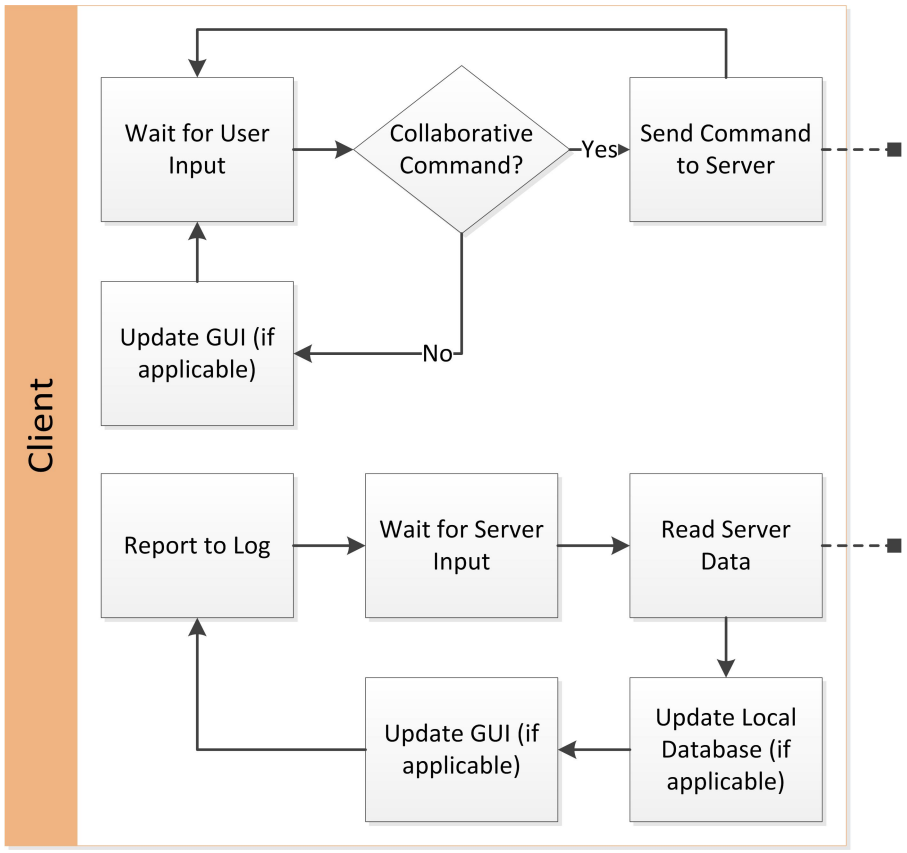

Figure 3.2: An Example Client Dataflow

As can be see in Figure 3.2, no computation is done on the Client other than what is sometimes necessary to create a command. All processing is done for all Clients on the Server. This allows for very light resource requirements on the Client, thus enabling a more cost-effective distribution of resources even for large analysis models [29].

3.1.2 External Connectivity

The cloud-based server has a networking method for communicating with the clients. This is done as a secure TCP network connection which, if not hosted in-house, allows for protected and monitored external access by the users. Clients use this to connect to the server and check user credentials, validating that the user has the appropriate rights to access the server. Figure 3.3 shows an example of this process.

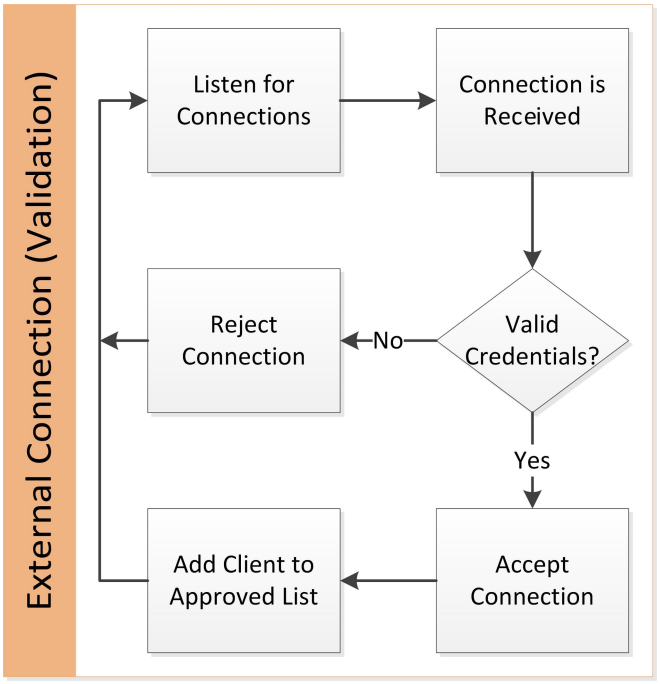

Figure 3.3: An Example Validation Dataflow in the External Connection

According to the example shown in Figure 3.3, clients connect to the system using the following process:

1. Each individual client connects to the cloud using the server's exposed network connection
2. The user's credentials are then checked to ensure that the he or she has sufficient rights to access the requested project data:
   (a) Upon successful validation, the associated client is marked as acceptable, and any appropriate information is then both accepted from and shared with the client

(b) Upon either logging out or disconnecting, the client loses this mark and will need to re-validate at the next connection

Once user rights are established and the connection is secured, the connection is then used for receiving commands from the clients as well as distributing resulting data, as shown in Figure 3.4.

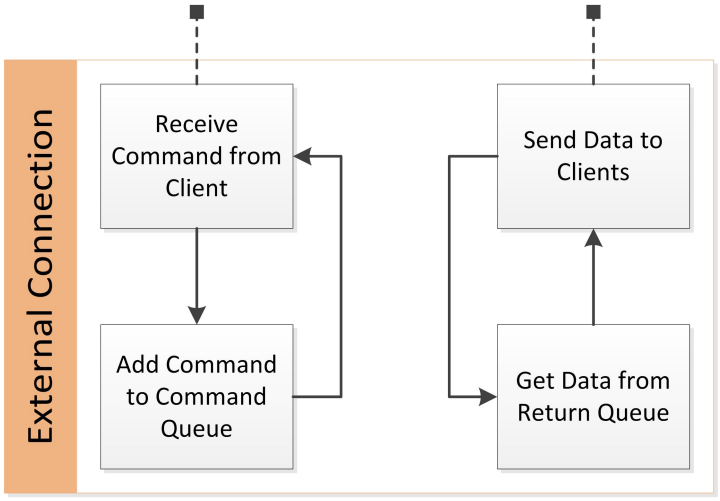

Figure 3.4: An Example External Connection Dataflow (After Validation)

Once connected, users can operate on the model:

1. The user creates a command using the client's user interface
2. The client sends this command to the server
3. The server checks to make sure that the client has already been validated
4. The client's command is added to the incoming queue
5. The next available SPI removes the command from the queue, queries the central database for appropriate model information, and starts processing the command
6. The queue waits for new commands and presents the next command to the next available SPI

3.1.3 Queue

The server prioritizes incoming commands, executing them in the order in which they arrive. At least one Server Processing Instance (SPI) is assigned to each command that is received (some processes allow the assigned SPI to utilize other slave SPIs during execution). The system queues each command until an SPI is available to execute it. Once an SPI retrieves a command from the queue, the next command is presented for processing on the next available SPI. This allows the system to execute commands in parallel.

After a command is successfully executed, the resulting data is distributed to the clients. Since these data sets can be large, it may take a significant amount of time to transmit across the network. Therefore, if multiple commands are completed at the same time, the resulting data is queued and transmitted in a consistent sequence in order to avoid message overlap or other significant communication errors.

3.1.4 Server Processing Instance (SPI)

The central processing unit of the system is the Server Processing Instance (SPI), where all commands are executed. The SPI is self-contained, needing only the incoming command and the related model data in order to execute the command. Any number of SPIs can be running and available at a given time.

When a command is presented to the SPI, it removes the command from the queue, queries the central data storage for the necessary model information, and executes the command. Upon successful completion of the command, the SPI updates the model data on the central data storage database and puts the resulting data on the return queue. See Figure 3.5 for an example of this process.

When an SPI successfully finishes execution, it:

1. Updates the central database with the new model data
2. Places the resulting data on the return queue
3. Clears its memory, returning to its original state
4. Waits for the next command

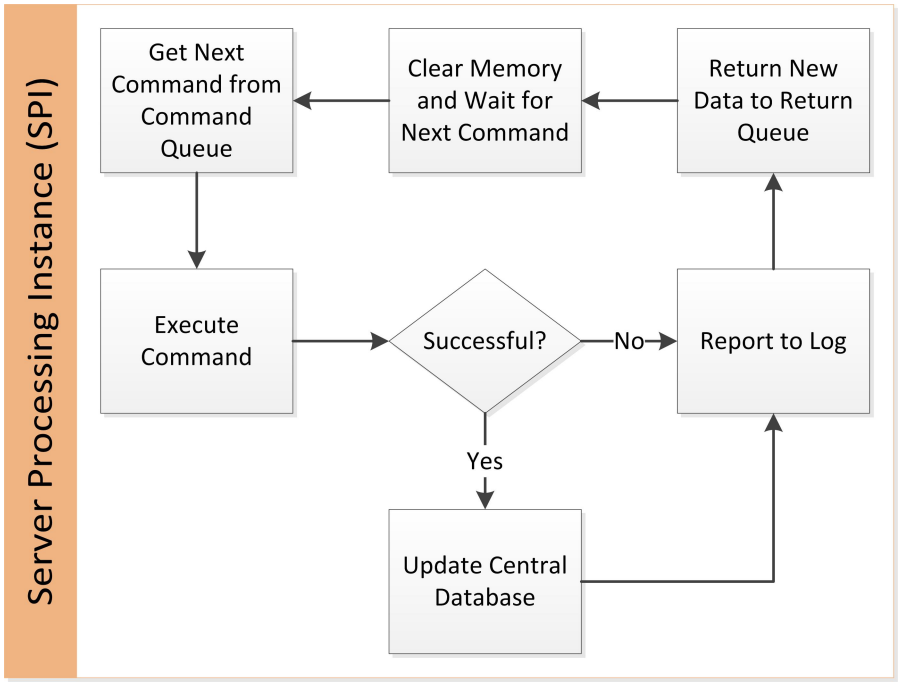

Figure 3.5: An Example SPI Dataflow

5. If the return queue is not empty, its contents are chronologically removed (oldest to latest) and distributed to the connected and approved clients

3.1.4.1 Canceling Commands and Undo

In the event that a user decides to cancel an executing command that he or she instigated, each SPI has a background thread that periodically checks the controller to see if a command has been sent that cancels the SPI's current command. Such a cancel command would specify the issuing user and the time at which the original command was sent. These data are tracked by the controller, which can then direct the command to the appropriate SPI.

If an undo command is submitted, various scenarios can take place. Two possible examples are:

- The last command that was submitted by any user is undone
- The last command that was submitted by the issuing user is undone

For the first case, the system would merely have to save a state prior to executing any command. If a subsequent undo command from any user is received, the system merely reverts

to the previous state. This would be difficult to use when many operations are happening quickly, because there is no guarantee that sending the undo command will undo the intended operation.

The second case, although seemingly more intuitive, is more complicated. This is due to the fact that a command by one user may be dependent on the data resulting from another user's command. For example, if User A meshes an object, User B creates loads on the resulting mesh nodes, and then User A sends an undo command, the nodes on which User B's command is dependent would cease to exist because User A's mesh was deleted.

This shows that in a real-time multi-user collaboration, the mode of correcting mistakes is inherently more complex than in single-user environments. Although not included in this research, further investigation is required in order to find feasible and reliable solutions to these problems (see Section 6.2.9).

3.1.5 Controller and Multiple Available SPIs

In order to take advantage of the scalability of the cloud, a number of SPIs are always "on call" for processing. As long as the system has available SPIs, each command is executed immediately upon receipt by the cloud (barring modeling conflicts, which will not be discussed here). The system monitors and dynamically changes its allocated resources according to several criteria, which may include processing demand (number and types of incoming commands), model size, number of active users, and the general availability of existing SPIs. As the demand increases, the system allocates more resources. As demand decreases, those resources are then freed for other uses. An example of this is shown in Figure 3.6.

When processing demand increases, the system spins up new virtual machines (VMs), each with a single SPI due to the fact that each SPI uses multiple threads. This allows all of the threads on the VM to be dedicated to the SPI. Operating costs could be reduced per SPI if multiple instances are run per VM; however, this could potentially result in performance degradation. The total number of running SPIs, as well as the number of SPIs per VM, can be tailored to optimal performance depending on usage scenarios and desired performance characteristics of the system.

The SPI does not store any data. Once computation has finished and the central data storage system is updated with any new data, the local memory on the SPI is cleared. This allows the

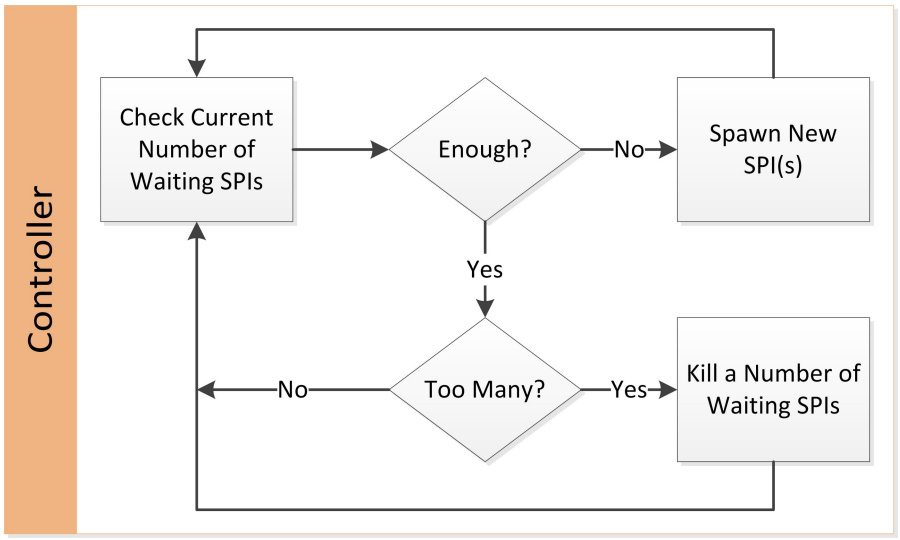

Figure 3.6: An Example Controller Dataflow

same SPI to operate on other projects, utilizing other data sets, without intermixing project data or carrying over old data to future computations.

3.1.6 Central Data Storage

Since a number of SPIs can be running at any time, a central storage system exists for managing shared data. This allows multiple SPIs to access the model data. Data-locking schemes are employed in order to keep the data from being written to by more than one SPI at a time.

3.2 Basic Pre-Processing Function in a Collaborative, Cloud Environment

The general pre-processing operations discussed in Chapter 1 are included. Some of these functions are modified from their original form (see Section 2.1.1) in order to operate more efficiently when used in a cloud-based or multi-user environment.

- Import geometry: when a client originates a command to import geometry, the command is executed by the server. If the geometry file is local to the client that issues the command, the client is able to read and send the file to the server along with the command for execution. Otherwise, the geometry file must be accessible to the server, such as on a shared drive.

- Edit geometry: the user can pick geometry that is displayed on the client for editing (in accordance with any applicable decomposition methods). The subsequent command is sent to and executed on the server, which then transmits the resulting changes to the clients. In the case that a geometry-edit command requires a large amount of resources to compute, the cloud can supply those according to demand.

- Apply materials: to define a new material, a user selects the material type (isotropic, orthotropic, linear, etc.) as well as the appropriate material properties (Young's Moduli, Poisson's Ratios, strengths, etc.). A material library is implemented, allowing a user to query and see if the desired material is already included. If a new material needs to be created, the corresponding parameters are sent to the server, and then distributed to the clients. The application of the material to geometry/mesh is also packaged and sent to the server for execution and dissemination.

- Discretize geometry: depending on the desired mesh type, the user selects the geometry to be meshed and enters the desired parameters into the client. The command is then sent to the server, where the geometry is meshed. Once the mesh is complete, the server then sends the resulting mesh to all of the clients, including node and element definitions and information describing the geometry to which the mesh is applied. In a cloud-based environment, this operation is executed on the server in its own thread or group of threads, thus allowing the server to continue executing other commands while the meshing is processed. Once the mesher signals that it is successfully finished, the server thread then incorporates the mesh into the server's database (including the newly-created nodes and elements), and then distributes the mesh to the clients. This process is further discussed in Section 3.4.1.

- Edit mesh: again, depending on the edit command being used, the user selects the nodes/elements to be edited, and enters the appropriate parameters in to the client. This information is then passed to the server. Due to the fact that many operations that edit meshes often take a substantial amount of time, editing meshes is accomplished on the server on a new thread, thus allowing the server to continue executing other commands. This operates much in the same way as creating a mesh (see above).

- Apply loads and boundary conditions: the application of nodal loads and boundary conditions doesn't require any real computational effort, so this operation is processed serially on the server. Furthermore, when calculating forces from pressure loads across a surface or a line, minimal computation is usually required in comparison to meshing. However, if the time to compute these is large, a new thread is created in order to allow the server to execute other commands.

Mutual exclusion (mutex) is utilized in order to protect against race conditions when operating with multiple threads on the same data. Race conditions occur when separate computation threads attempt to modify shared data at the same time, often causing unpredictable results [30]. Mutex protection is applied to all shared databases, as well as to all shared network communication methods.

3.3 Multiple Concurrent Projects

It is rare that a design company has only one active project. Therefore, the server has the capability to support multiple simultaneous sessions of collaborative projects (see Figure 3.7). This includes maintaining data separation and integrity, as well as ensuring that each client gets consistently updated with the data pertaining to its project. For example, many companies that operate under the United States' International Traffic in Arms Regulations (ITAR) are required to keep ITAR-restricted data inaccessible to non-approved parties. This may include isolating data from some of their own engineers. If a company has one ITAR-restricted multi-user project and another non-restricted multi-user project, both projects are able to utilize the same hardware without inappropriately exposing sensitive data [31].

Furthermore, the workload applied to the hardware due to executing a command for one project doesn't interfere with the operation or performance of another project. Means are established that will detect, limit, and scale the load distribution in the event of high command traffic and execution (see Section 3.1).

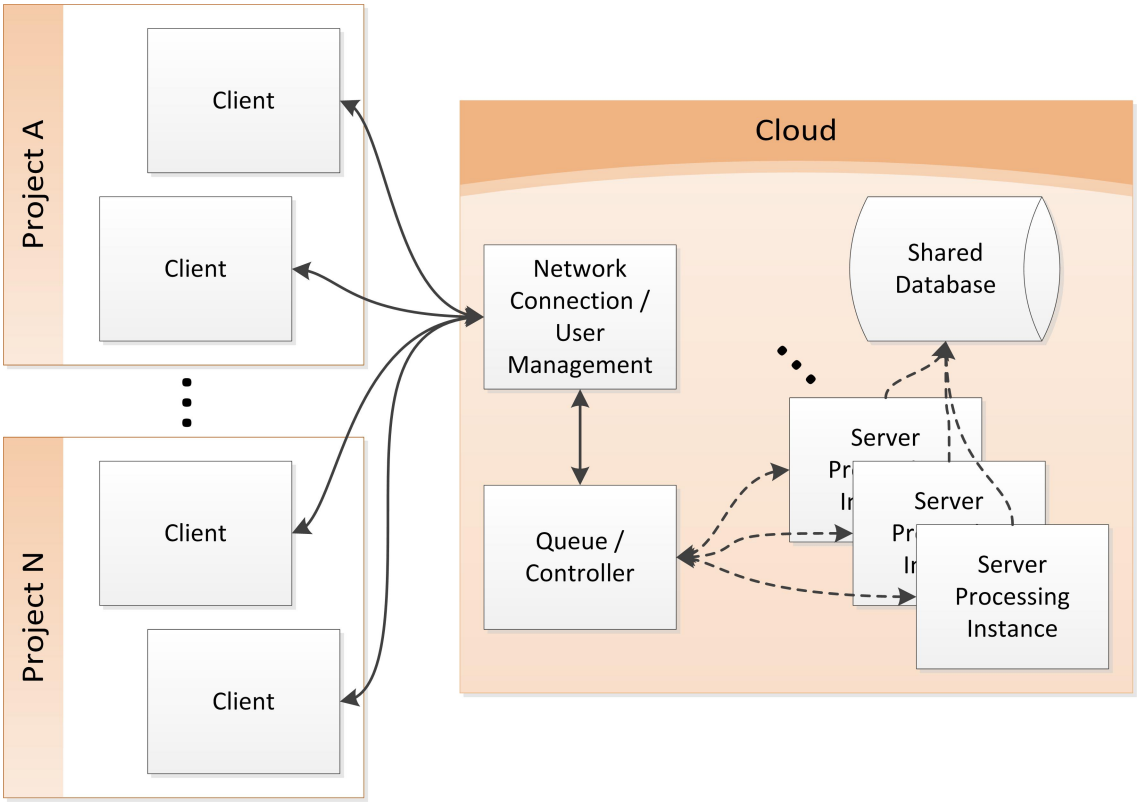

Figure 3.7: An Example High-Level Dataflow with Multiple Concurrent Projects

3.4 Simultaneous Command Execution

In FEA pre-processing, some commands are dependent on data created by other commands. For example, many operations including mesh clean-up and applying nodal loads and constraints are impossible until the mesh is created. However, once meshed, an object can undergo many parallel, simultaneous operations.

3.4.1 Parallel Meshing: Multi-Object, Multi-Thread

If an assembly consists of multiple discreet parts or objects, each part is independently meshed on its own computation thread.

1. Meshes in adjacent parts have a surface mesh created at each interface
2. The solid volume of each part (if applicable) is meshed, using any existing interface meshes as starting seeds

3. The meshing thread signals the SPI's main thread that it is finished, and returns the newly-created mesh

This architecture allows the pre-processor to initiate meshing instances at the same time and to utilize a continuous numbering sequence for all entities created (nodes, elements, etc.), thereby retaining numbering consistency among all clients. This n_p number of parts can be meshed in approximately one-$n_p{}^{th}$ ($1/n_p$) the time required using a conventional pre-processor if n_p number of threads are applied toward meshing (one thread for each part).

Some research has been done elsewhere on multi-threaded meshing algorithms, but these are not studied in this research [32]. However, the author does not see any technical barrier to their use in this architecture.

3.4.2 Other Parallel Commands

Once a model is meshed, it can be decomposed into individual workspaces according to the needs of the design team [33]. Each user can then clean up the mesh and otherwise operate in his or her assigned workspace. Potential issues and appropriate methods of resolving conflicts along the workspace boundaries will not be studied as part of this research.

Applying loads and boundary conditions to the model can also occur simultaneously due to the small amount of information and processing required to execute such commands. For example, the information needed to create a new load includes the load type, an identifier for the node upon which the load is placed, a magnitude, and vector components of the load application.

CHAPTER 4. IMPLEMENTATION

This chapter presents two prototype implementations of the architecture proposed in Chapter 3. Since the focus of this research involves cloud-based, multi-user FEA pre-processors, much attention was placed on the multi-user server, and less on implementing a large variety of pre-processing functionality. This is due to the fact that many pre-processing algorithms have already been extensively researched and would not require significant modification (if any) when implemented in this new architecture. Furthermore, due to time and energy constraints, several aspects of the prescribed architecture were not included in these implementations, including a Controller, Shared Database, and extensive User Verification. Since these technologies have already been used and proven in various aspects of cloud-computing, this allowed more resources to be spent developing the more interesting aspects of the system.

4.1 MUFE (Multi-User Finite Elements)

The author began developing a Multi-User Finite Element pre-processor (MUFE) in early Fall of 2011 as a class project. It was intended to be a simple test-bed for new collaborative technologies, in an effort to avoid the often cumbersome and inadequate Advanced Programming Interfaces (APIs) found in currently-existing software [1]. Although MUFE has only basic functionality[1], it can be modified with relative ease in order to implement new collaborative prototypes (see Section 4.1.5 for more description about MUFE's current functionality). It was designed and built with a strong-server/weak-client architecture. This allows all commands to be sent to the server, where they are executed, and all resulting data to be sent back to the clients, thus guaranteeing data consistency among all clients.

Originally, MUFE was intended to be used either in stand-alone mode (single-user, no server required), or in multi-user mode (where all computing is done on the server). This design

[1] See Appendix B for a description of MUFE's current functionality and limitations.

required that largely the same code be implemented on both the client and the server. This was done by allowing both the client and server to share the same source tree, with macros defining the differences at compile time.

The MUFE program was written in C++, using Microsoft Visual Studio®[2] and Qt™[3] Creator. Multi-threading was done in C++ by utilizing the open-source Boost libraries. The C# networking software was written by James Wu in the Fall of 2011.

4.1.1 Client

The User Interface (UI) and the Graphical User Interface (GUI) were written using the cross-platform C++ integrated development environment Qt Creator (which is included in the free Qt SDK). Its main component is the Graphic Area, which draws the objects for viewing and manipulation using OpenGL®[4]. An image of the UI is shown in Figure 4.1, where the Graphic Area is the large blue rectangle on the left of the image.

The Graphic Area acts as the main viewer for the model, as well as accepting input for picking entities upon which to operate. Other components in the UI include a series of buttons (along the top-right of the UI) that allow the user to modify display attributes of the GUI, as well as a model tree (right side of the UI) that displays all of the entities currently in the model, and a small text widget (bottom-right) that is used to output information.

Some user inputs are commands that are not necessary to share with other users (such as view transformations, display changes, etc.). However, many commands define the creation or editing of modeling entities, which are necessary for the successful completion of the model. A command of this latter type is defined as a Collaborative Command (CC).

In single-user mode, the MUFE program waits for input from the user, then packages that input and sends all collaborative commands to the MUFE Core (all non-collaborative commands are sent straight to the GUI for processing). The Core then executes the necessary commands, creates or edits the appropriate entities (if applicable), and signals the GUI to update its display. Figure 4.2 shows how the MUFE program uses input data in single-user mode.

[2]Visual Studio is a registered trademark of Microsoft Corporation.
[3]Qt is a trademark of Nokia Corporation.
[4]OpenGL is a registered trademark of Silicon Graphics, Inc.

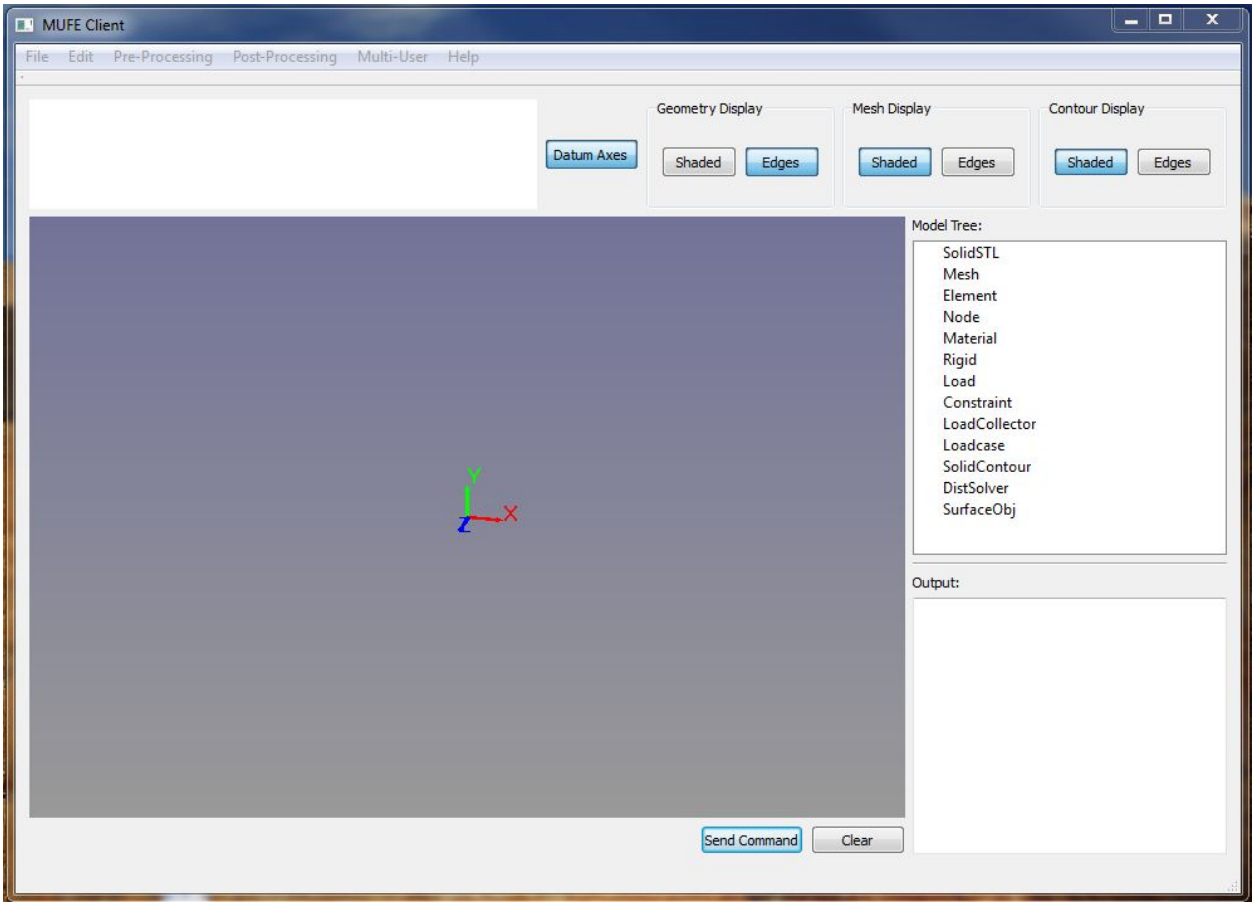

Figure 4.1: Client User Interface

In multi-user mode, the Client waits for input from either the user or the Server. When input is received from the user, it checks to see if the input is a CC. If so, it is packaged into a command and sent to the networking client via a Named-Pipe; otherwise, the command is passed directly to the appropriate local module (usually the rendering module). The Client then returns to a waiting state.

When data is received from the server (via another Named-Pipe), the Client identifies the data type represented by the command (geometry, mesh, load, etc.), then locally creates the appropriate entity using the provided data. Once that is complete, the GUI is updated with the new information. This process is shown in Figure 4.3.

Since all entity types are created on the Server before their data are distributed to the Clients, the entities share the same integer identifier on all machines. This means that object identifiers are uniform and unique for all users in the project. Therefore, when a user picks an en-

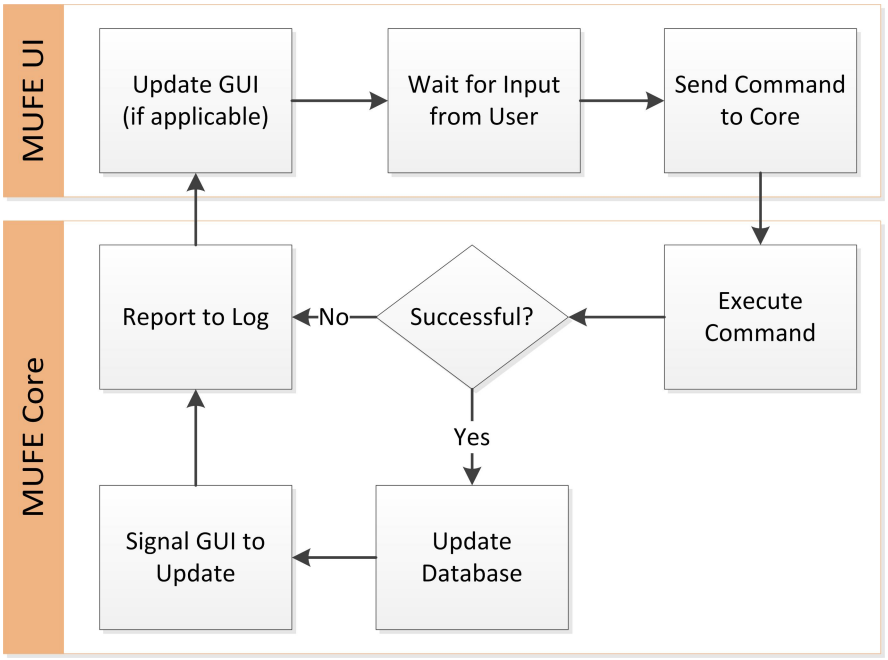

Figure 4.2: Client Dataflow in Single-User Mode

tity upon which to operate, the entity is identified using its identifier, which is passed to the Server as part of the command string.

The latest version of the Client consists of over 34,000 lines of code.

4.1.2 Server

The Server is a headless (without GUI) version of the MUFE Core. It maintains network connections with the Clients, periodically checking to see if there are any new commands. When new commands are received, it processes them in the order in which they were received. Figure 4.4 illustrates the flow of data in the MUFE server.

4.1.3 Command Structure

In order to maintain transparency and ease of modification, commands were encoded as ASCII text for transmission between the Client and Server. The JSON text format was chosen to represent the data structures in the commands. JSON (JavaScript Object Notation) is a data-interchange format similar to, but slightly lighter than Extensible Markup Language (XML). JSON

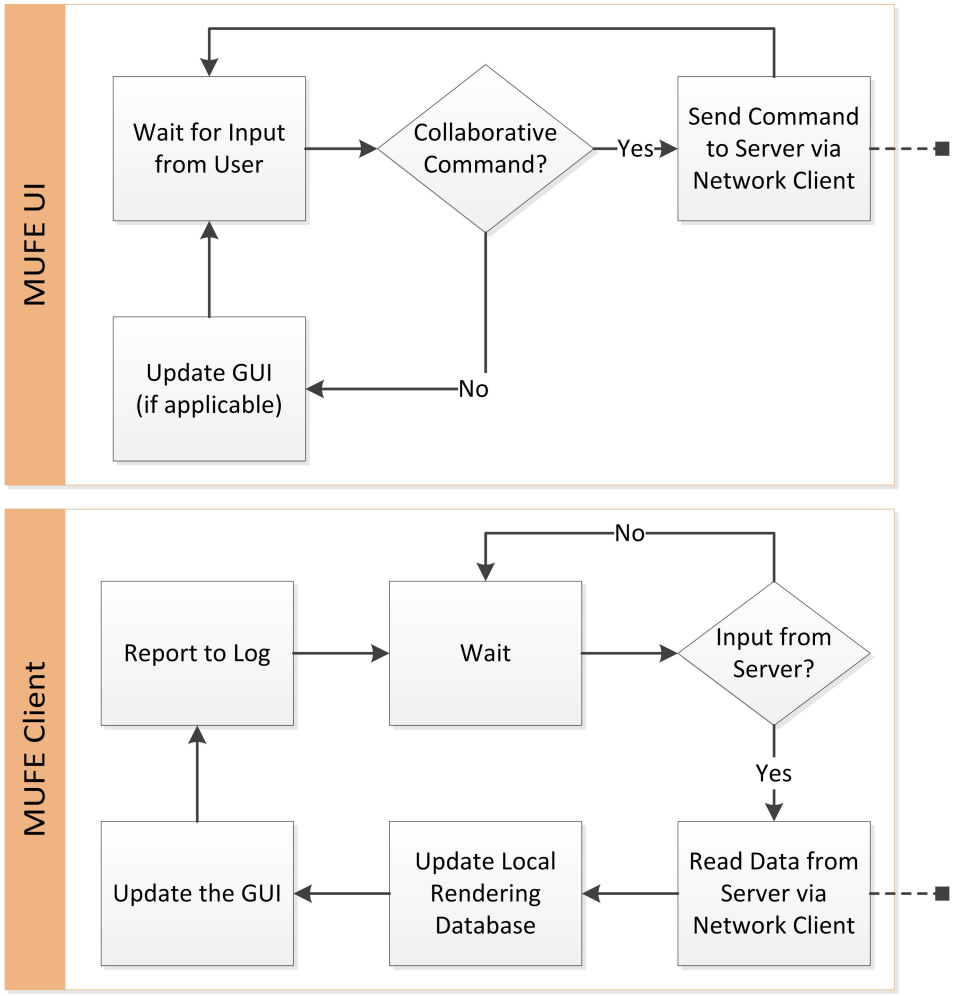

Figure 4.3: Client Dataflow in Multi-User Mode

allows for a command with somewhat shorter length than XML due to the lack of explicit close tags. A sample JSON command specifying a material creation is shown in Figure 4.5.

Future implementations may encode the command and data strings differently. For example, they may be further encrypted, or even kept in binary for transmission. The overall architecture presented in this research is not dependent on JSON as the only method of data transmission.

Some commands and their resulting data are very similar. For example, the command in Figure 4.5 doesn't require any real computation, merely a material definition. This is because in this case, the command string encapsulates the entire definition of the new material, which means that the resulting data string that the Server sends back to all of the Clients would be the same. Other commands that perform similarly are those that define nodal constraints and loads.

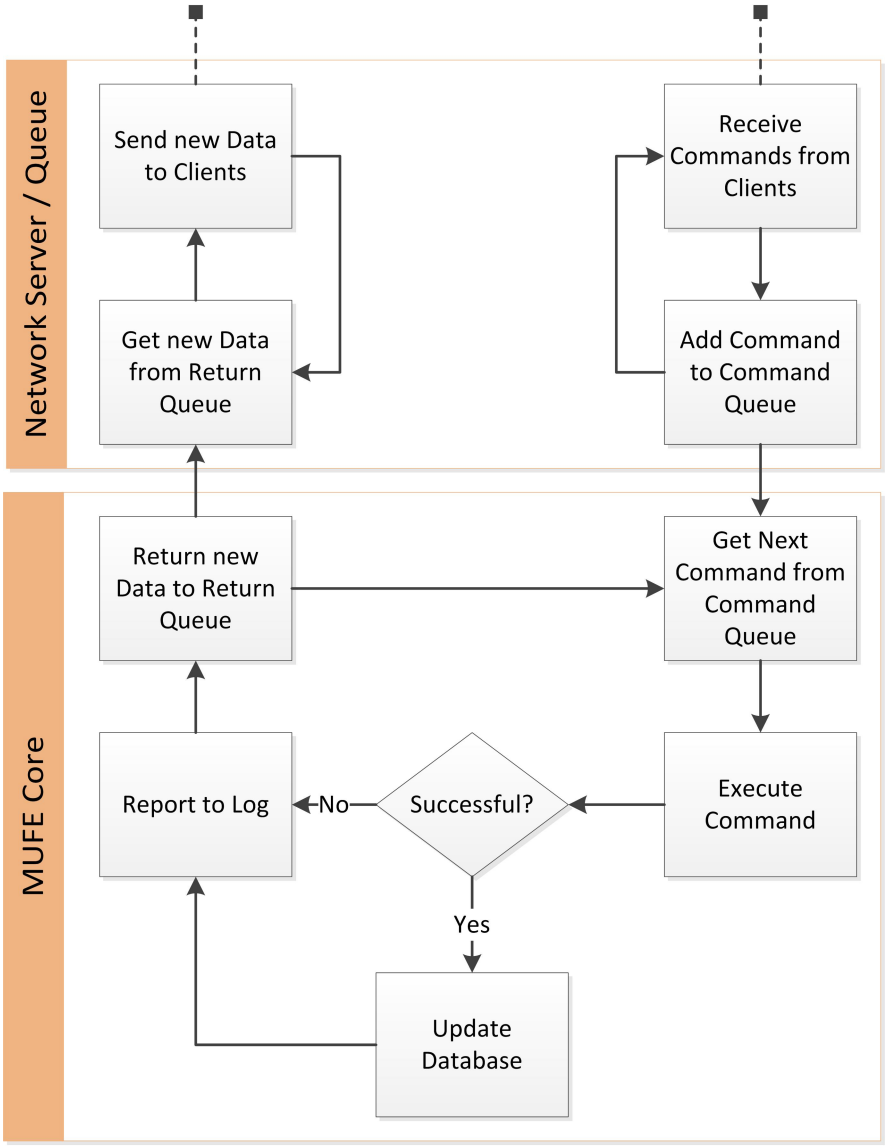

Figure 4.4: Dataflow in Server

All command and definition strings have two main parts: a string specifying the command type, which explains if this string is declaring a new command or defining an object that has been created on the server; and a parameters section, which is where data specific to a particular command or object type are stored. A command string that operates on a number of existing objects will include in the parameters section a number *selobjcount* describing the number of objects upon which the command operates, and a list *selobjs* giving the integer identifier for each of the selected objects. Each command operates on a specific object type, so the integer identifiers in *selobjs* relate to a specific object. For example, commands that call for the creation of a mesh operate on

```
{
"commandtype":"newmaterial",
"parameters":{
      "name":"Aluminum",
      "E":"1e+007",
      "Nu":"0.33",
      "Rho":"0.0975"
      }
}
```

Figure 4.5: JSON String Defining a New Material

geometric objects. The command shown in Figure 4.6 creates a mesh on geometric object number zero (indexing starts at zero for stored objects). If more objects were to be meshed in the same command, *selobjs* would include a line for each one (as demonstrated in Figure 4.6).

```
{
      "commandtype": "newmesh",
      "parameters": {
            "meshtype": "2",
            "material": "0",
            "params": "p",
            "meshsize": "0.05",
            "selobjcount": "1",
            "selobjs": {
                  "i0": "0"
            }
      }
}
```

Figure 4.6: JSON String Describing a Mesh Command

Some commands, such as the mesh command in Figure 4.6, are relatively simple, including all of the necessary parameters that the Server needs to create the mesh. However, the resulting data from such a command that the Server sends out to the Clients involves a considerably larger amount of data, since it includes the definition for each newly-created Node in the mesh, as well as the nodal connectivity for each Element (see Figure 4.7).

```
{
      "commandtype": "definemesh",
      "parameters": {
            "material": "0",
            "params": "p",
            "meshtype": "2",
            "selobjcount": "1",
            "selobjs": "0",
            "numnodes": "131",
            "numelements": "380",
            "nodes": {
                  "i0": {
                        "p0": "-2.02254",
                        "p1": "4",
                        "p2": "-1.46946"
                  },
                  "i1": {
                        "p0": "0.772542",
                        "p1": "4",
                        "p2": "-2.37764"
                  },
                  …
            },
            "elements": {
                  "i0": {
                        "t": "2",
                        "rn": "4",
                        "nl": {
                              "n0": "70",
                              "n1": "4",
                              "n2": "72",
                              "n3": "118"
                        }
                  },
                  …
            }
      }
}
```


Figure 4.7: Part of a JSON String Defining a Newly-Created Mesh

JSON-formatted strings are typically shorter than XML-formatted strings defining the same object. The author compared the lengths of JSON and XML strings defining a mesh of various sizes. Table 4.1 summarizes the results. As can be seen, these command strings may easily reach lengths on the order of megabytes (MB). For example, a small mesh containing 131 Nodes and 380 4-Node Tetrahedral Elements is represented in JSON by a string with a length of 34.6 kilobytes

Table 4.1: Comparison of String Lengths Using JSON and XML Formats

Element Type	Number of Nodes	Number of Elements	Length of JSON String (MB)	Length of XML String (MB)
4-Node Tetrahedra	131	380	0.0346	0.0396
4-Node Tetrahedra	673	2,988	0.267	0.306
4-Node Tetrahedra	2,914	14,975	1.38	1.58
4-Node Tetrahedra	5,512	29,619	2.72	3.14
4-Node Tetrahedra	25,329	146,730	14.1	16.2

(KB). An XML string representing the same mesh would have a length of 39.6 KB. A considerably larger mesh (but still relatively small in comparison to many industrial models) with 5,512 Nodes and 29,619 Elements utilizes a JSON string of 2.72 MB, while the same mesh defined in XML requires 3.14 MB of memory. This shows that JSON consistently produces a shorter string than the comparable XML string storing the same data. For this reason, JSON was selected as the format used for transferring data over the network.

The open-source C++ Boost[5] libraries were used for parsing and creating JSON and XML strings. Using existing and well-tested libraries such as Boost allowed code for other, more significant, parts of the software to be developed.

4.1.4 Network Communication Methods

In single-user mode, MUFE does not utilize any networking methods. However, in multi-user mode, a command that originates in the UI in the form of a JSON-formatted string is sent via Named-Pipe to an external C# process that was developed by James Wu for BYU's Cubit-Connect project.

4.1.5 Meshing and Other Functionality

As was stated in Section 4.1, the MUFE pre-processor was developed with only very basic functionality, including operations such as read stereo-lithography-faceted (.stl) geometry, define isotropic materials, create tetrahedral solid meshes, specify nodal forces and constraints, and set

[5] See http://www.boost.org/ for more information.

up linear-static loadcases. It utilizes the open-source meshing program TetGen[6] (compiled as a separate executable) to mesh solid volumes with tetrahedral elements. MUFE can be easily extended to utilize various solvers; for example, Altair Optistruct®[7] and the open-source solver CalculiX[8] can currently be used with limited functionality.

For this basic implementation, no methods for mesh cleanup or decomposition were implemented. This is not due to any limitation of the proposed architecture, but merely constrained by resources available for exploring and writing the necessary code.

4.2 Small Server Implementation

The MUFE Server was originally designed to work only over the Local Area Network (LAN), utilizing a local workstation as the server. This setup is not scalable, and only parallelizable to the extent of the Server's hosting hardware. This configuration was tested on a small scale, the results of which are presented in Chapter 5.

4.2.1 Client

Other than connecting directly to the server instead of an external connection, the Client for this implementation functions as described in Section 4.1.1. This includes creating command strings and sending them to the Server, accepting model changes from the Server, and allowing the user to view and select portions of the model as necessary to create commands.

4.2.2 External Connection

Since this server was tested only on the local network and all connections originated from within the network, an externally-facing connection was not necessary on the server. However, an external connection was temporarily simulated with a VPN connection, which allowed for secure access from remote networks.

[6] See http://wias-berlin.de/software/tetgen/ for more information.

[7] OptiStruct is a registered trademark of Altair Engineering, Inc.

[8] CalculiX is a free three-dimensional structural finite element solver. See http://www.calculix.de/ for more information.

4.2.3 Queue, Controller, and Central Data Storage

Commands that are recieved by the server's networker are placed in a queue for processing. That queue is frequently checked by the MUFE Core (SPI). When a command is found by the SPI, it executes the command (spawning a new thread if necessary), updates the data storage, and returns the resulting changes to the networker for transmission to the clients.

Due to the fact that this particular implementation was not scalable, a Controller was not necessary. Also, MUFE's original central data storage system was sufficient for this implementation. Scalability would require significant changes to the way the SPI stores and uses data. Note that even though the iplementation discussed here is not scalable, the architecture presented by this research (as described in Chapter 3) is scalable.

4.3 Cluster Implementation

With a few architectural changes, the system was modified to work on a local cluster. Figure 4.8 shows the high-level process flow for a cloud-based implementation. Note that due to the inherent scalability of the cloud, a large number of Server Processing Instances (SPIs) could potentially be spun up in order to support processing demand; however, only one SPI was used in this implementation. A more thorough redefinition of the MUFE database storage system would allow for multiple concurrent SPIs, but this was not attempted due to time constraints.

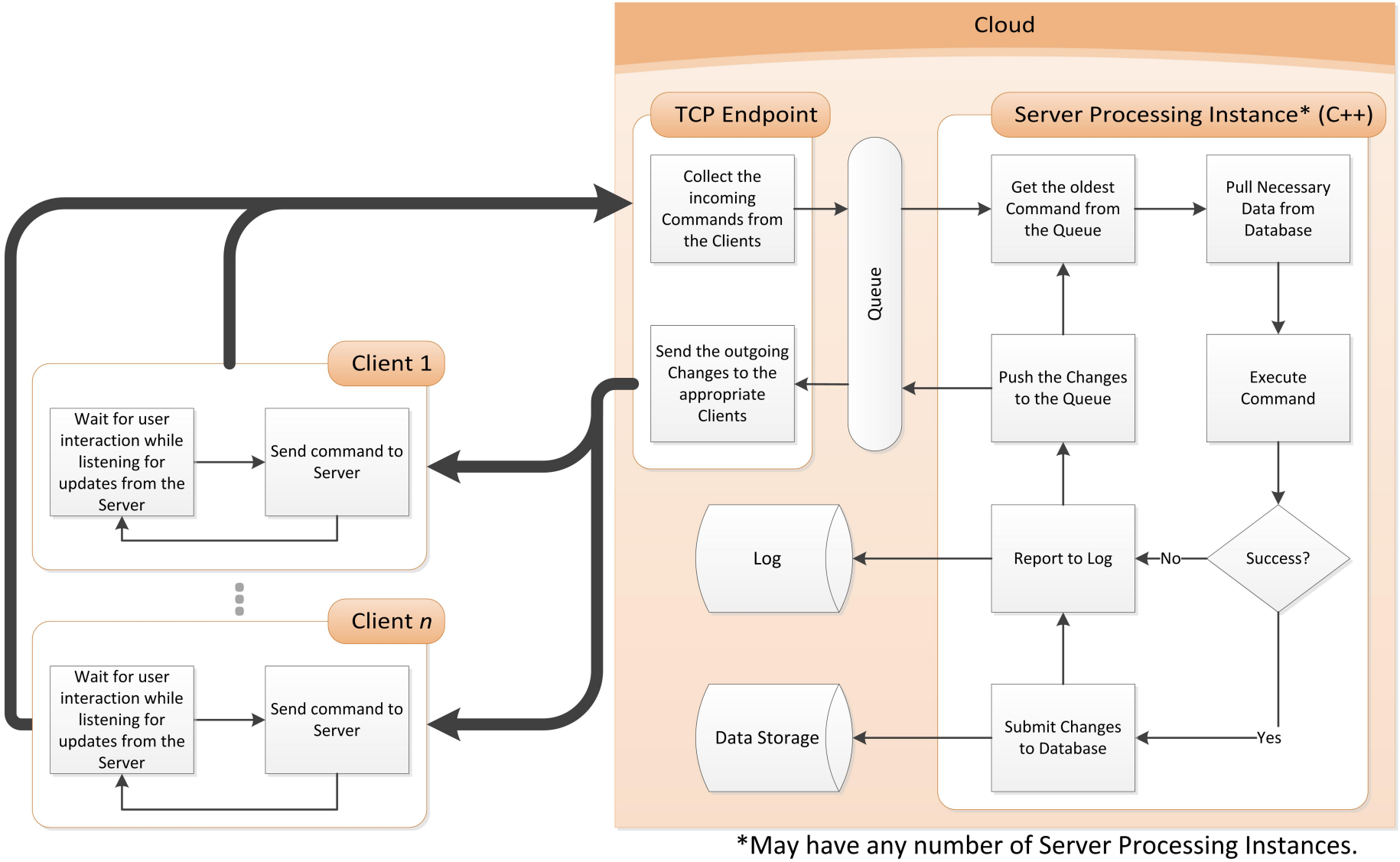

Figure 4.8: High-Level Dataflow of the Entire System

4.3.1 Client

The Client for this implementation functions exactly the same as for the local implementation (see Section 4.2.1), only it directs its network connection to the server's externally-facing connection on the cluster. See Section 4.1.1 for a full description of the Client.

4.3.2 External Connection

The cluster exposes an externally-facing connection to the hosted server. This allows users to remotely access the server via standard network protocols such as TCP/IP. The TCP protocol also supports the common communication methods HTTP and HTTPS [34].

All Clients connect directly to the exposed TCP connection on the Frontend, which is dedicated to managing communication between the Clients (external to the cluster) and the server processes (which are hosted in the cluster). This machine communicates with the external Clients, as well as with the processes that are hosted on the compute machines, by receiving and distributing information to and from them.

4.3.3 Queue and Controller

Once the Frontend receives a command from a Client, the command is entered into a queue. The SPI then pulls the command from the queue for processing. Once the SPI finishes a computation, the resulting data are placed in another queue and sent back to the Clients via the Frontend. If the size of the resulting data is large, the data set is placed on network-attached storage (NAS), and a link to the data is entered into the return queue. The Frontend can then retrieve the data from the NAS for transmission to the Clients.

4.3.4 Multiple Available Instances

Due to the fact that the use of an external database would require a drastic re-write of the MUFE code, only one SPI was used for this implementation. This allowed MUFE to be more immediately utilized in the cluster. However, this approach limited the prototype's ability to distribute and scale beyond a single machine.

4.3.5 Central Data Storage

As stated in the previous section, this implementation did not utilize a central database. Instead, the single SPI uses a custom data storage system that stores each data type (mesh, node, element, geometry, etc.) in its own table. This structure uses a basic singleton class to protect against multiple simultaneous writes to the database, thus enabling data consistency.

CHAPTER 5. RESULTS

The prototype was developed to study the change in time required for a collaborative team to pre-process an analysis model when all computation for the model is done on a remote server. Three tests were accomplished: multiple simultaneous projects, modeling a wing section, and modeling a pressure vessel with a single plane of symmetry. The goal of the first test was to merely demonstrate that it was possible to maintain data integrity and consistency, even when multiple projects share the same processing server. The next two tests studied the time benefits gained when a multi-user pre-processor uses a central server for all computation.

5.1 Multiple Simultaneous Projects

The first test was performed on a very early version of the prototype. It consisted of one user modeling a solid cylinder under compressive and tensile forces, and two other users collaboratively modeling a solid cantilever beam, attached rigidly to a solid frustum. All three users were logged into the server, which was running all of the computation for both projects (see Figures 5.1 and 5.2). This test was intended to find out if it was possible to utilize the same server for both projects and maintain data separation and consistency while not creating any noticeable amount of lag at the clients.

5.1.1 Developing the Test

The Server was hosted on a HP Z400 workstation with an Intel®[1] Xeon®[2] W3520 CPU (8 threads at 2.67 GHz) and 12 GB of RAM. The three Clients used various hardware configurations for this test, including:

[1] Intel is a registered trademark of Intel Corporation.
[2] Xeon is a registered trademark of Intel Corporation.

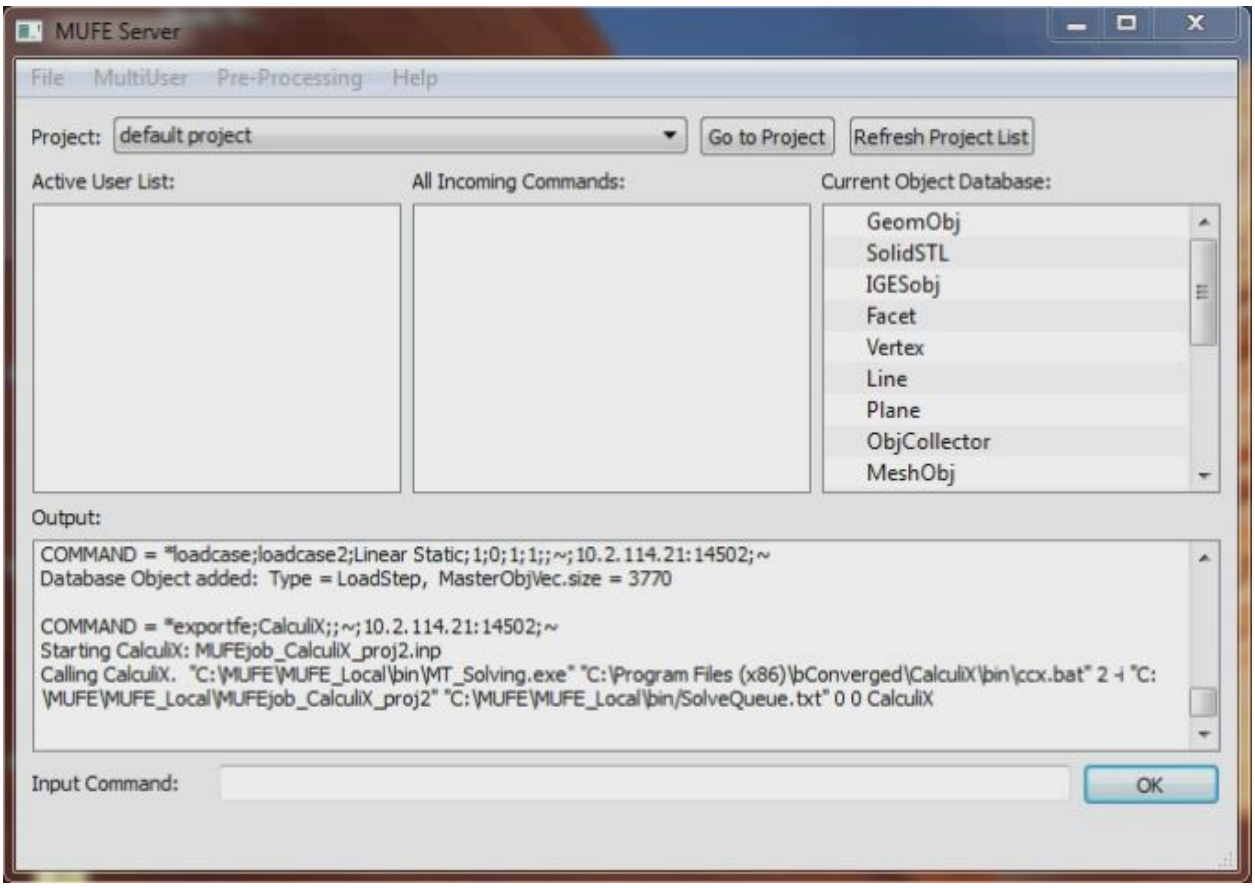

Figure 5.1: Screenshot of the Server from the Multi-Project Test

- Gateway®[3] GT5432 desktop with an AMD Athlon™[4] 64 X2 5000+ CPU (2 threads at 2.6 GHz) and 3 GB of RAM
- HP xw4600 workstation with an Intel®[5] Core™[6] 2 Quad CPU (4 threads at 2.5 GHz) and 6 GB of RAM
- HP xw4300 workstation with an Intel®[7] Pentium®[8] D 945 CPU (2 threads at 3.4 GHz) and 4 GB of RAM

Both models were simultaneously built. The cylinder model was created using the following process:

[3]Gateway is a registered trademark of Gateway, Inc.
[4]AMD Athlon is a trademark of Advanced Micro Devices, Inc.
[5]See footnote 1.
[6]Core is a trademark of Intel Corporation.
[7]See footnote 1.
[8]Pentium is a registered trademark of Intel Corporation.

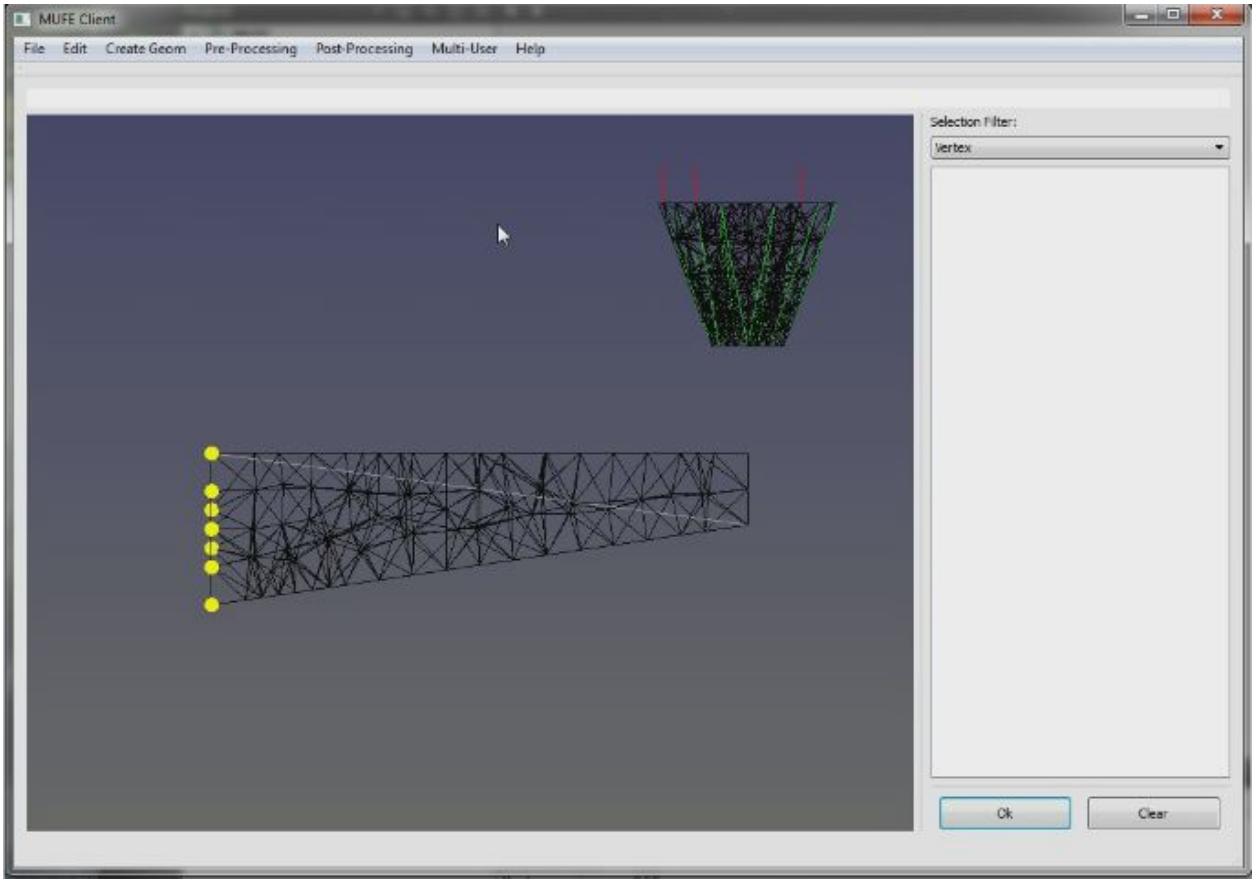

Figure 5.2: Screenshot of a Client from the Multi-Project Test

1. Cylinder geometry was imported
2. A material was defined
3. The cylinder was meshed with tetrahedral elements
4. Loads and boundary conditions were applied

The team building the cantilever beam and frustum model performed the following procedure:

1. Beam and frustum geometries were imported
2. Materials were defined
3. The geometries were independently meshed with tetrahedral elements

4. Boundary conditions were applied to the base of the cantilever beam

5. Loads were applied to the top of the frustum

6. The tip of the cantilever beam was rigidly connected to the bottom of the frustum

This resulted in two separate and independent models, one single-user and one multi-user. These were both simultaneously built using the same central server.

5.1.2 Test Results

The two simple models used in this test showed that even though the computation was done for both models on the same central server hardware, data from one project were neither mixed with nor available to the other project. This demonstrates that simultaneous multiple projects can reliably be run on the same hardware without sacrificing data integrity. It may be necessary to further study the effect on projects that share a server with a project that undergoes long computations or transfers large data sets (see Section 6.2.3).

5.2 Wing Section

This project consisted of editing the geometry of a cantilevered wing section, then modeling the wing for FEA. Since the ability to edit geometry inside of the prototype is quite limited, the control points that define the wing geometry were manipulated, thereby modifying the geometry itself. This study measured the time benefit gained by parallelizing the modeling process in a multi-user environment.

5.2.1 Developing the Wing Section Test

The same Server that was described in Section 5.1.1 was used. The following hardware configurations were used as Clients:

- HP Z400 workstation with an Intel Xeon W3520 CPU (8 threads at 2.67 GHz) and 12 GB of RAM (two of these were used as clients)

- HP xw4300 workstation with an Intel Pentium D 945 CPU (2 threads at 3.40 GHz) and 4 GB of RAM
- HP xw4300 workstation with an Intel Pentium 4 CPU (2 threads at 3.40 GHz) and 3.25 GB of RAM

The wing was formed out of a 2^{nd}-degree NURBS surface consisting of nine Bézier patches (three in both directions), with a total of twenty-five control points.

The following process was used for the test:

1. Starting geometry was imported in the form of a flat sheet (see Figure 5.3)

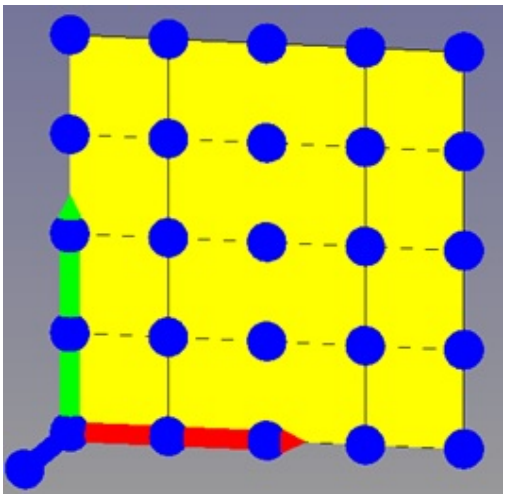

Figure 5.3: Beginning Flat Sheet Geometry for the Wing Section Test

2. The geometry's control points were manipulated to curve around the specified shape of the wing (see Figure 5.4)
3. A material was created
4. The geometry was meshed
5. Pressure loads and boundary conditions were applied (see Figure 5.5, where the loads are represented by red lines and the boundary conditions by yellow spheres)

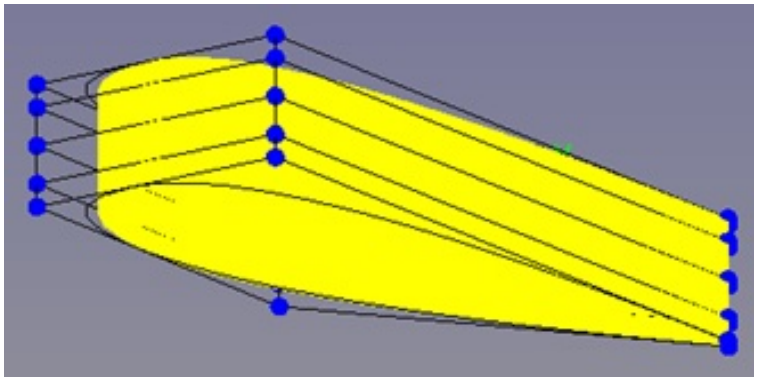

Figure 5.4: Final Shaped Geometry for the Wing Section Test

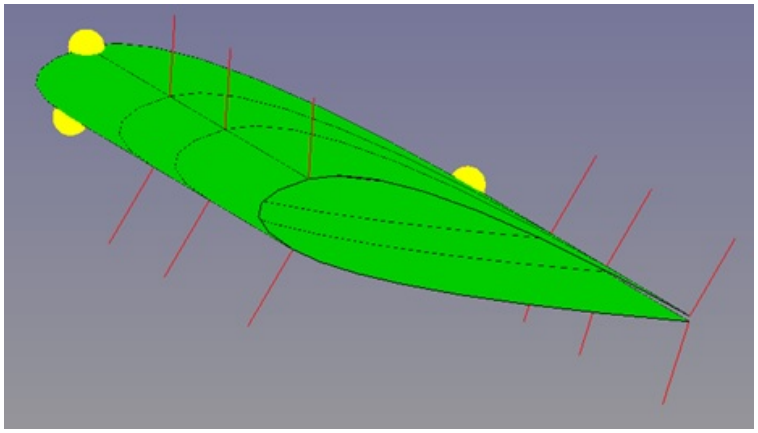

Figure 5.5: Final Model for the Wing Section Test

5.2.2 Wing Section Test Results

This test was executed in three ways:

1. An individual user performed all of the operations sequentially
2. A group of two users collaborated to perform the operations
3. A group of four users collaboratively performed all parallelizeable operations

On average, the two-person teams completed the project in about 51% of the time required for the average individual user, where the four-person team finished it in about 20% of the same time (or 40% of the two-person team average). Results from the wing test are recorded in Table 5.1 and can be seen in Figure 5.6.

Table 5.1: Time Results from the Wing Section Test

Run #	Individual User	2-User Team	4-User Team
1	31:48	16:08	5:17
2	31:44	10:04	–
3	27:25	–	–
4	28:48	–	–
5	22:36	–	–
6	18:52	–	–
7	22:31	–	–
8	22:52	–	–
Average:	25:49	13:06	5:17
Standard Deviation:	4:47	4:17	n/a

5.3 Symmetric Pressure Vessel

The third test was similar to the Wing Section experiment, only somewhat more complicated. The finished model was one half of a vase-shaped prssure vessel with an outwardly-directed pressure load.

5.3.1 Developing the Pressure Vessel Test

The same hardware was used for this test as was described in Section 5.2.1. The geometry was made up of a 2^{nd}-degree NURBS surface with twenty-five Bézier patches (five in each direction), with a total of forty-nine control points. A process similar to that described in Section 5.2.1 was followed here (see Figures 5.7 and 5.8). Also, the same types of tests as before were again used to study the time benefit for this technology.

5.3.2 Pressure Vessel Test Results

The same three random tests as were described in Section 5.2.2 were executed here. However, for this experiment the tests were performed in random order, with three total repititions for each test type. Also, the users for each test run were randomly selected from a pool of users. On average, the two-person teams completed the project in about 65% of the time required by individual users, and the four-person teams completed it in approximately 61% of the two-person time.

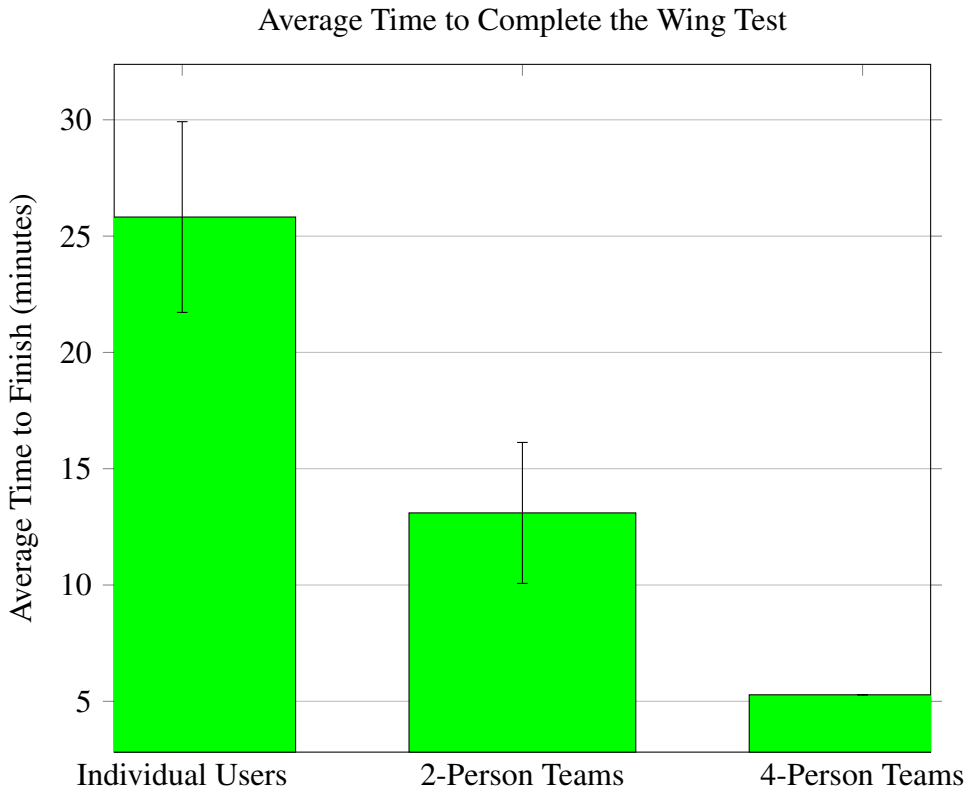

Figure 5.6: Average Time Results for Each Test Type Performed for the Wing Section Test

This means that the four-person teams completed the project in about 39% of the time required for an average individual user. Results from the pressure vessel test are summarized in Table 5.2 and shown graphically in Figure 5.9.

Table 5.2: Time Results from the Pressure Vessel Test

Run #	**Individual User**	**2-User Team**	**4-User Team**
1	53:26	36:17	18:44
2	30:30	22:18	15:57
3	25:17	16:21	10:43
Average:	36:24	24:58	15:08
Standard Deviation:	14:58	10:13	4:04

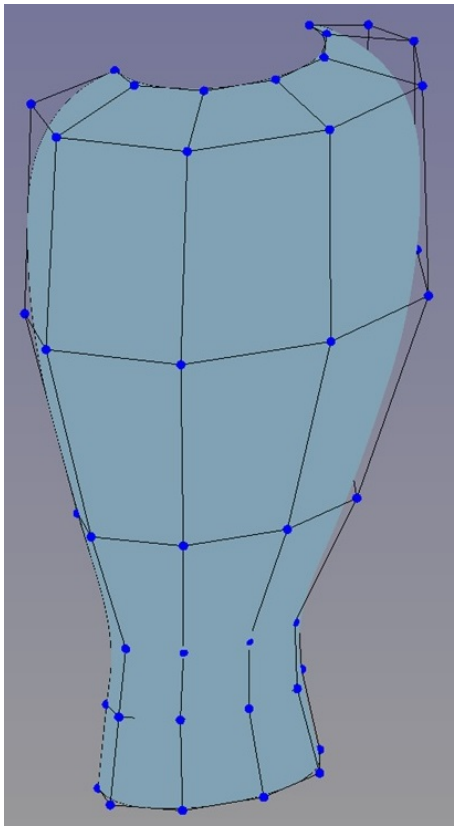

Figure 5.7: Final Shaped Geometry for Pressure Vessel Test

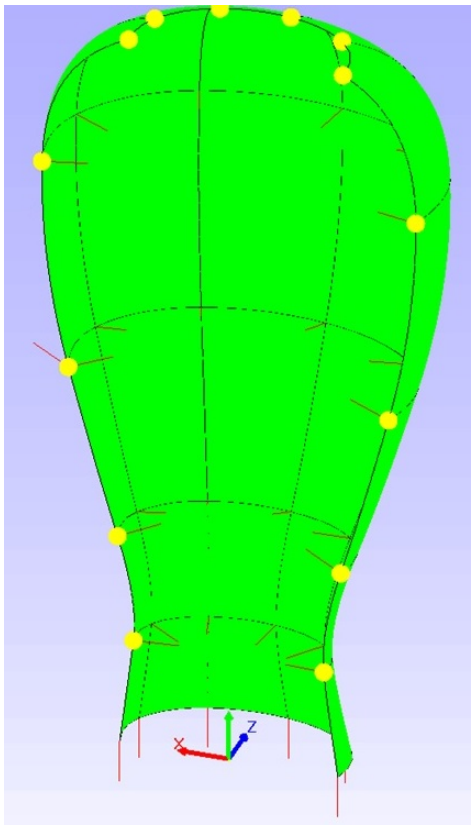

Figure 5.8: Final Model for Pressure Vessel Test

5.4 Results Summary

These tests demonstrated a substantial decrease in pre-processing time as additional users are added to the collaborative environment. For simpler models, the time difference approaches $1/n^{\text{th}}$ of the original, single-user pre-processing time, where n is the number of users collaborating on the model (see Section 2.2.1). Collaboratively pre-processing more complex models will likely yield varied results, based on factors such as team size and experience, as well as equipment, implementation, network connectivity, and distribution of workload. With the appropriate management and forethought, a collaborative team could significantly decrease the time required to pre-process any practical engineering model. This benefit (excluding the time required for the solving process and post-processing) could theoretically approach that found in multi-user CAD processes (see Section 2.2.1).

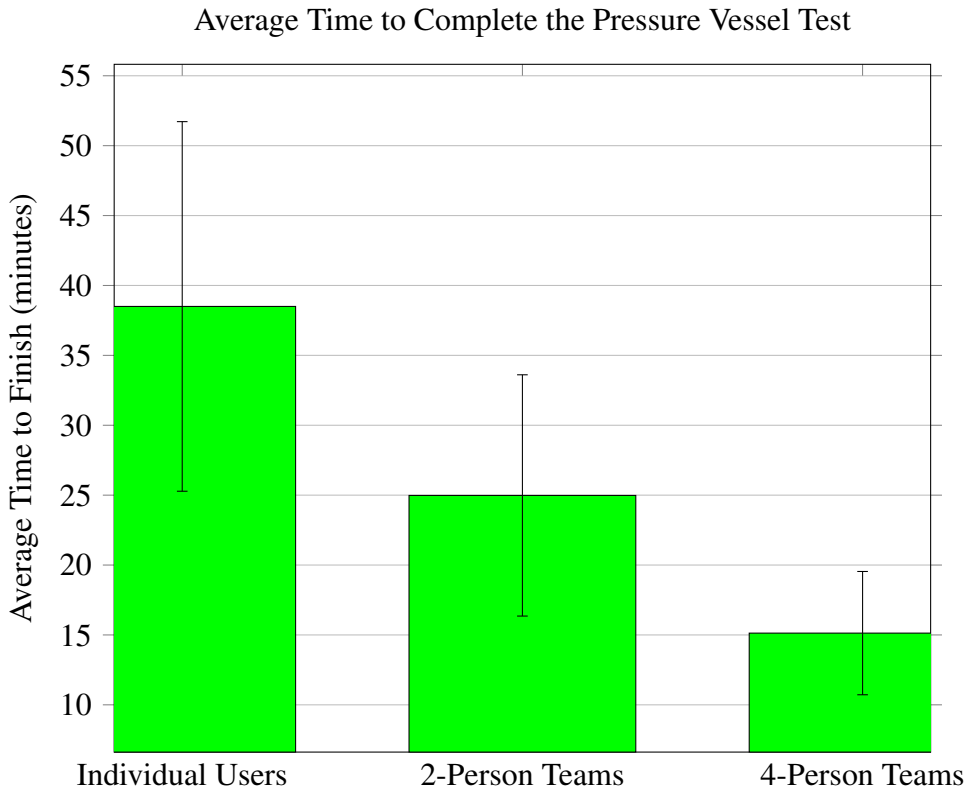

Figure 5.9: Average Time Results for Each Test Type Performed for the Pressure Vessel Test

CHAPTER 6. CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE WORK

6.1 Conclusions

As stated in Section 1.3, the main objectives for this research were:

1. Design a framework to facilitate cloud-based, multi-user pre-processing.
2. Create a simple, multi-user FEA pre-processor to test the framework.
3. Implement and test this software on a local, networked system.

The methods and architecture presented in Chapter 3 fulfill the first objective by detailing a framework that supports and facilitates cloud-based, multi-user FEA pre-processing. The design and implementation of such a system can be accomplished without substantially greater difficulty than that which would be encountered when creating a standard FEA pre-processor. Please note that the focus and deliverable of this research is the framework design, not necessarily any particular implementation.

Section 4.1 introduces a simple FEA pre-processor prototype that was designed and built with the sole purpose of testing a server-based, multi-user FEA pre-processing system. However, because this implementation merely prototypes the framework that has been previously presented (see Chapter 3), several aspects of the architecture were not implemented due to constraints on the author's time and coding abilities. Those parts that were not included in the prototype discussed in Chapter 4 include the Controller, Shared Database, and a complete User Verification system. The author chose not to focus the implementation on these components because they have already been proven and used widely in various other related cloud-computing and distributed architectures. As such, this preliminary implementation meets the second research objective.

Sections 4.2 and 4.3 present various network implementations of the software. Furthermore, Chapter 5 discusses several tests that were performed with the intent to evaluate the im-

plemented system with a variety of users and modeling scenarios. The data for these tests are presented in Sections 5.1.2, 5.2.2, and 5.3.2, thereby satisfying the last objective.

6.2 Future Work

Through the course of this research, the author has noticed several potential areas of further study, including parallel algorithms applied to cloud-based FEA pre-processors, implementing this technology using other cloud providers, and studying the economic effect of using this and other related technology in real design situations.

6.2.1 Parallel Algorithms

It would be beneficial to consider algorithms for meshing, geometry importing/exporting, data transfer, etc., that are more distributable. Doing so would more fully take advantage of the scalable benefits of the cloud, thus potentially increasing overall performance of the system.

6.2.2 Distributed Cloud Implementation

The implementation tested as part of this research consisted of a non-distributed system. The ideal method proposed in Chapter 3 includes multiple SPIs that can operate on the same central database. This architecture should be implemented and tested in order to more fully explore and demonstrate the benefits and challenges of cloud-based, multi-user FEA pre-processing.

In addition to local cluster implementations, research should be done considering Microsoft Azure Cloud and other cloud providers such as Amazon EC2 and Google Cloud since these are popular entry points into cloud computing.

6.2.3 Multi-User Server Performance Under Heavy Load

Study the effect that one heavily-loaded multi-user session would have on other sessions that share the same server. This might consider the sensitivity of measures such as network lag, client graphical interface responsiveness, server responsiveness, etc., to variables such as dataset

size, computation type and length, Server Processing Instance (SPI) availability, server hardware, user distribution, etc.

6.2.4 Hybrid Architectures for Distributing Computation to Both Server and Clients

Other potential architectures should be studied in an attempt to balance computation load on the server with distributed computing on the clients when client resources permit. Systems like this are already implemented in industry, which utililizes idle client hardware as a remote-computing node. Extending the client's capability to act as an extra Server Processing Instance when idle would allow for a greater, system-wide ability to distribute resources.

6.2.5 Economic Costs and Benefits of Multi-User Collaboration in Engineering

It would be interesting to study the monetary effect of using multi-user CAx engineering tools and paradigms in a typical product design environment. The potential costs and benefits of the use of this type of software in a design company have never been established. The author believes that this knowledge would be beneficial in garnering more industrial attention for these new methods and practices.

6.2.6 Decomposition Methods and User Editing Rights

Methods for decomposing the modeling tasks in order to avoid user conflicts should be studied. Possible methods include:

- Workspace decomposition: each user is assigned a finite workspace, beyond which they have limited or no ability to edit the model
- Role decomposition: each user is assigned a task or specialty (for example, one user only deals with materials, another with boundary conditions, another with non-linear loads, another with optimization responses, etc.), and the user is responsible for that role throughout the entire model
- Hybrid Workspace and Role decomposition: each user a specific role or task to perform as part of a team that is assigned to a finite workspace in the model

6.2.7 Conflict Resolution and General Multi-User Project Management

This research would include investigating methods to avoid user conflict through proper management procedures (some methods of which are suggested in Section 6.2.6), as well as methods to successfully resolve modeling conflicts when they occur. Furthermore, research should look into ways to automate conflict management and resolution, as well as to investigate possible modification to existing technology (geometry kernels, databases, etc.) that would more readily support multi-user environments, and thereby a more conlict-tolerant model.

6.2.8 Undo/Redo Commands

Methods for reliably and predictably handling Undo and Redo commands in a multi-user environment are imperative for successfull operation of any multi-user CAx tool. Further reserach would include developing algorithms for gracefully handling such cases, as well as developing architecture and database structures for efficiently manipulating the affected data.

6.2.9 Integration with Product Lifecycle Management (PLM)

Branching methods should be considered that will allow users to create a copy of the model (associative or otherwise), in which they could try different operations or parameters without breaking the model for all of the other users. This methodology would help Producte Lifecycle Management (PLM) systems manage collaborative data, as well as handle design changes and other similar modification scenarios.

6.2.10 Integrating Multi-User Design with Multi-User Analysis

Research into cross-discipline systems where multi-user CAD and multi-user analysis can share the same central model, where all users are always working on the most up-to-date model. This type of environment would greatly increase the amount of collaboration among disciplines, thereby shortening the overall product development process by helping catch design errors and minimizing potentially costly turn-backs later on. Furthermore, this environment could also be extended to include other disciplines such as manufacturing, marketing, etc.

REFERENCES

[1] Red, E., Jensen, C. G., French, D., and Weerakoon, P., 2011. "Multi-user architectures for computer-aided engineering collaboration." In *ICE Conference*. 1, 8, 10, 29

[2] Owen, S. J., Clark, B. W., Melander, D. J., Brewer, M., Shepherd, J. F., Merkley, K., Ernst, C., and Morris, R., 2007. "An immersive topology environment for meshing." In *16th International Meshing Roundtable*. 2, 10, 61

[3] Balling, R. J., 2010. *Finite Elements*. BYU Academic Publishing, Provo, Utah. 2, 61

[4] Pelosi, G., 2007. "The finite-element method, part i: R. L. Courant [historical corner]." *Antennas and Propogation Magazine,* **49**(2), April, pp. 180–182. 2, 60, 61

[5] Lee, H., Kim, J., and Banerjee, A., 2010. "Collaborative intelligent CAD framework incorporating design history and tracking algorithm." *Computer-Aided Design,* **42**(12), December, pp. 1125–1142. 3

[6] Weerakoon, P., Wu, J., Bright, T., Teng, C.-C., Red, E., Jensen, C. G., and Merkley, K., 2012. "A networking architecture for multi-user FEA pre-processor." *Computer-Aided Design & Applications,* **9**(a). 8, 12, 15, 63

[7] Weerakoon, P., Wu, J., Bright, T., Red, E., Teng, C.-C., Jensen, C. G., and Merkley, K., 2012. "Multi-user FEA pre-processing & workspace assignment." In *2012 ASME Early Career Technical Conference Proceedings*. 8

[8] Red, E., Jensen, C. G., Weerakoon, P., French, D. J., Benzeley, S. E., and Merkley, K. G., 2012. "Multi-user computer aided prototypes." In *TMCE*. 8

[9] Tautges, T. J., 2001. "The generation of hexahedral meshes for assembly geometry: survey and progress." *International Journal for Numerical Methods in Engineering*(50), pp. 2617–2642. 8, 10

[10] Hill, M. D., and Marty, M. R., 2008. "Amdahl's Law in the multicore era." *Computer*, July, pp. 33–38. 8, 11

[11] Rashed, G., 2012. "Development and evaluation of an open source finite element analysis framework." In *1st International Conference on Civil Engineering for Sustainable Developement*. 8

[12] Roensch, S., 2008. Finite element analysis: Pre-processing [Online; accessed 15 August 2012]. 9

[13] Yang, X.-S., 2007. *A First Course in Finite Element Analysis*. Luniver Press, Frome. 9, 61, 64

[14] Briggs, J. C., 2011. HyperMeshMU [Class Project]. 12

[15] Ebeida, M. S., and Mitchell, S. A., 2011. "Uniform random voronoi meshes." In *20th International Meshing Roundtable*. 12

[16] Quey, R., Dawson, P. R., and Barbe, F., 2011. "Large-scale 2d random polycrystals for the finite element method: Generation, meshing, and remeshing." *Computer Methods in Applied Mechanisms and Engineering,* **200**, pp. 1729–1745. 12

[17] Hewlett-Packard Development Company, L. P., 2012. HP remote graphics software [Online; accessed 23 January 2013]. 13

[18] Hewlett-Packard Development Company, L. P., 2010. *Remote Graphics Software: Secure, collaborative access*. [Online; accessed 23 January 2013]. 13

[19] NEi Software, 2013. NEi Stratus: An introductory mobile nastran FEA app [Online; accessed 4 February 2013]. 13

[20] Stackpole, B., 2011. "NEi Stratus meshes nastran solvers with the cloud." *Design News*, February. 13

[21] Arita, Y., Nozaki, N., and Demizu, K., 2012. "Mechanical design platform on Engineering Cloud." *Fujitsu Scientific & Technical Journal*, pp. 422–427. 13

[22] Fujitsu Limited, 2011. Fujitsu announces "Engineering Cloud" for new era in manufacturing. 13

[23] Yasuda, M., 2012. "Fujitsu's Engineering Cloud." *Fujitsu Scientific & Technical Journal*, pp. 404–412. 14

[24] CD-adapco, 2013. A flexible approach to hardware deployment [Online; accessed 4 February 2013]. 14

[25] Sriram, D., Logscher, R., and Fukuda, S., 1991. "Computer-aided cooperative product development." In *MIT-JSME Workshop, Lecture Notes in Computer Science*. 14

[26] Chen, H.-M., and Tien, H.-C., 2007. "Application of peer-to-peer network for real-time online collaborative computer-aided design." *Journal of Computing in Civil Engineering*, March/April, pp. 112–121. 14

[27] Leahy, D., Challinger, J., Adler, B. T., and Ardon, S. M., 2009. System and method for enabling users to interact in a virtual space [United States Patent 8145998]. 15

[28] Wikipedia, 2013. Thin client, January [Online; accessed 23 January 2013]. 18

[29] Nieh, J., Yang, S. J., and Novik, N., 2000. "A comparison of thin-client computing architectures.". 18

[30] Wikipedia, 2013. Race condition, January [Online; accessed 4 February 2013]. 26

[31] U.S. Department of State, 2013. International Traffic in Arms Regulations (ITAR). 26

[32] Antonopoulos, C. D., Blagojevic, F., Chernikov, A. N., Chrisochoides, N. P., and Nikolopoulos, D., 2009. "Algorithm, software, and hardware optimizations for delaunay mesh generations on simultaneous multithreaded architectures." *Journal of Parallel and Distributed Computing*, July, pp. 601–612. 28

[33] Weerakoon, P., 2012. "Multi-user methods for fea pre-processing." Master's thesis, Brigham Young University, Provo, Utah. 28

[34] Microsft, 2013. How to set up communication with a virtual machine [Online; accessed 14 February 2013]. 41

[35] Fish, J., and Belytschko, T., 2007. *A First Course in Finite Elements*. Wiley Press, June. 64

[36] Logan, D. L., 2011. *A First Course in the Finite Element Method*., 5th ed. CL Engineering, January. 64

[37] MacDonald, B. J., 2011. *Practical Stress Analysis with Finite Elements*., 2nd ed. Glasnevin Publishing, May. 64

[38] Zeid, I., 2005. *Mastering CAD/CAM*. McGraw-Hill, New York, ch. 17: Finite Element Method. 64

[39] Hughes, T. J. R., 2000. *The Finite Element Method: Linear Static and Dynamic Finite Elment Analysis*. Dover Publications, August. 64

[40] LeVeque, R., 2007. *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems (Classics in Applied Mathematics)*. Society for Industrial and Applied Mathematics, July. 64

[41] Demmel, J. W., 2007. *Applied Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, September. 64

[42] Bazilevs, Y., ao da Veiga, L. B., Cottrell, J. A., Hughes, T. J. R., and Sangalli, G. "Isogeometric analysis: Approximation, stability and error estimates for h-refined meshes.". 65

[43] Cottrell, J. A., Hughes, T. J. R., and Bazilevs, Y., 2009. *Isogeometric Analysis: Toward Integration of CAD and FEA*. Wiley. 65

[44] Borden, M. J., Scott, M. A., Evans, J. A., and Hughes, T. J. R., 2011. "Isogeometric finite element data structures based on Bézier extraction of NURBS." *International Journal for Numerical Methods in Engineering*, pp. 15–47. 65

APPENDIX A. AN INTRODUCTION TO THE ANALYSIS PROCESS

Methods have been produced and refined over the last seventy years in efforts to facilitate the numerical approximation of real-world problems [4]. These methods, including Finite Element Analysis (FEA), are used to solve various types of complex problems in many areas including structural, thermal, fluid, magnetic, electronic, nuclear, and chemical. In industry, FEA is used in various stages of product design and development, ranging from preliminary design to design optimization to post-release failure analysis. It is widely used to solve structural and mass/heat transfer problems in aerospace, automotive, architecture, manufacturing, civil engineering, city planning, and many other industries. Figure A.1 shows an example where FEA was used to simulate a vehicle crash. The FEA process consists of three main stages: pre-processing, where

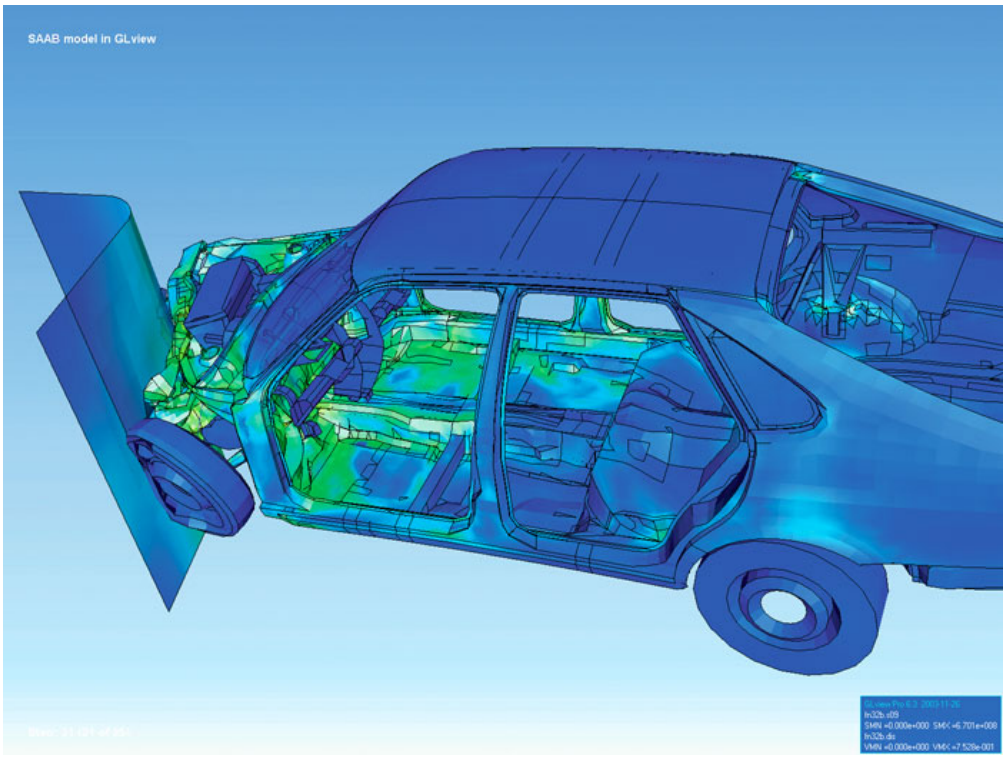

Figure A.1: An Asymetrical Collision Analysis in FEA

the model is setup and defined; solving, where the results of the problem are approximated; and post-processing, where the results are examined [13].

A.1 Pre-Processing

A vast majority of the time spent in the Analysis phase is spent building the model (i.e., "pre-processing"). According to a survey performed at Sandia National Laboratories, about 73% of the time spent developing an analysis model is consumed in this stage [2]. Regardless of the software used, the main steps involved in pre-processing include: importing geometry, discretizing the geometry ("domain"), defining material properties, and applying loads and boundary conditions [3, 4].

1. Importing geometry may include reading a kernel-neutral file-format (such as IGES or STEP), or a kernel- or CAD-specific format (such as Parasolid®[1], ACIS™[2], NX™[3] Part, etc.). It is often necessary to simplify (or "clean up") this geometry by removing small or insignificant artifacts such as fillets, chamfers, small holes, non-structural components, and any other artifact that does not contribute to the intent of the model. Figure A.2 shows an example CAD model.

2. Discretizing the domain (also known as "meshing"), is done in order to break up the complex geometry into smaller shapes ("elements") whose closed-form solutions are known, thus rendering the problem easier to solve by summing the solutions of all of the smaller problems. Discretization creates both nodes and elements, where elements are made up of neighboring groups of nodes. If an automatic meshing algorithm is used, it is often necessary for the user to manually fix elements that are either too small, too large, or whose geometry may render them numerically unstable. In a large model, this process can take a considerable amount of time. Figure A.3 shows an example sports car body that has been discretized into many smaller finite domains, or elements.

[1] Parasolid is a registered trademark of Siemens Product Lifecycle Management Software Inc.
[2] ACIS is a registered trademark of Spatial Technology, Inc.
[3] See Chapter 1, footnote 4.

Figure A.2: Example CAD Geometry of the Liftfan for the USAF F-35B

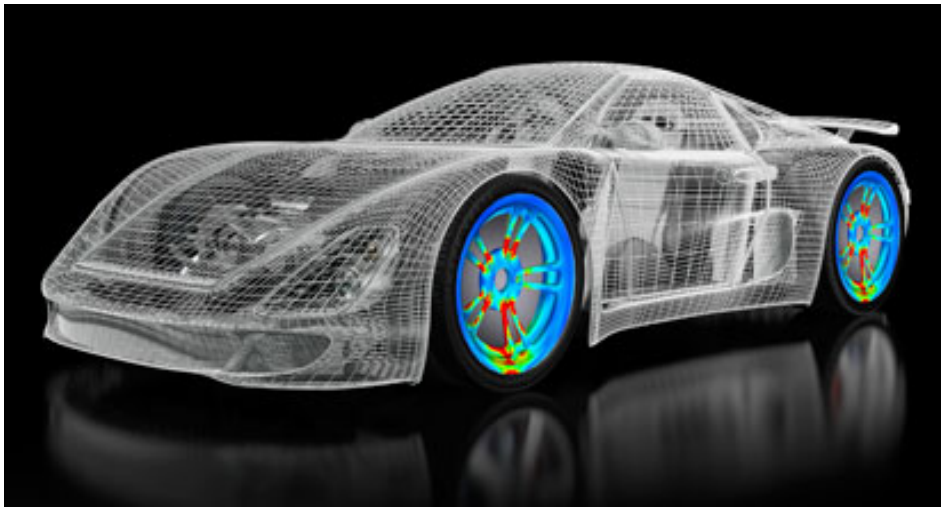

Figure A.3: Example of a Car Body that has been Discretized into a Mesh

3. Material properties are applied to the elements (be it individually, to a subset of elements, or to all elements in the model). These properties may include the Young's Moduli, Poisson's Ratios, Density, etc. of the material with which each element is associated.

4. Loads and boundary conditions are applied to the model. Loads may represent instances of structural loads, displacements, temperatures, pressures, heat/mass flows, etc., and boundary conditions are used to signify phenomena such as constraints (places of zero displacement), and energy/heat sinks. This process can also be quite time-consuming if the model requires

a complex loading scenario, or a large number of loading scenarios. Figure A.4 shows an example of a simple pre-processed part.

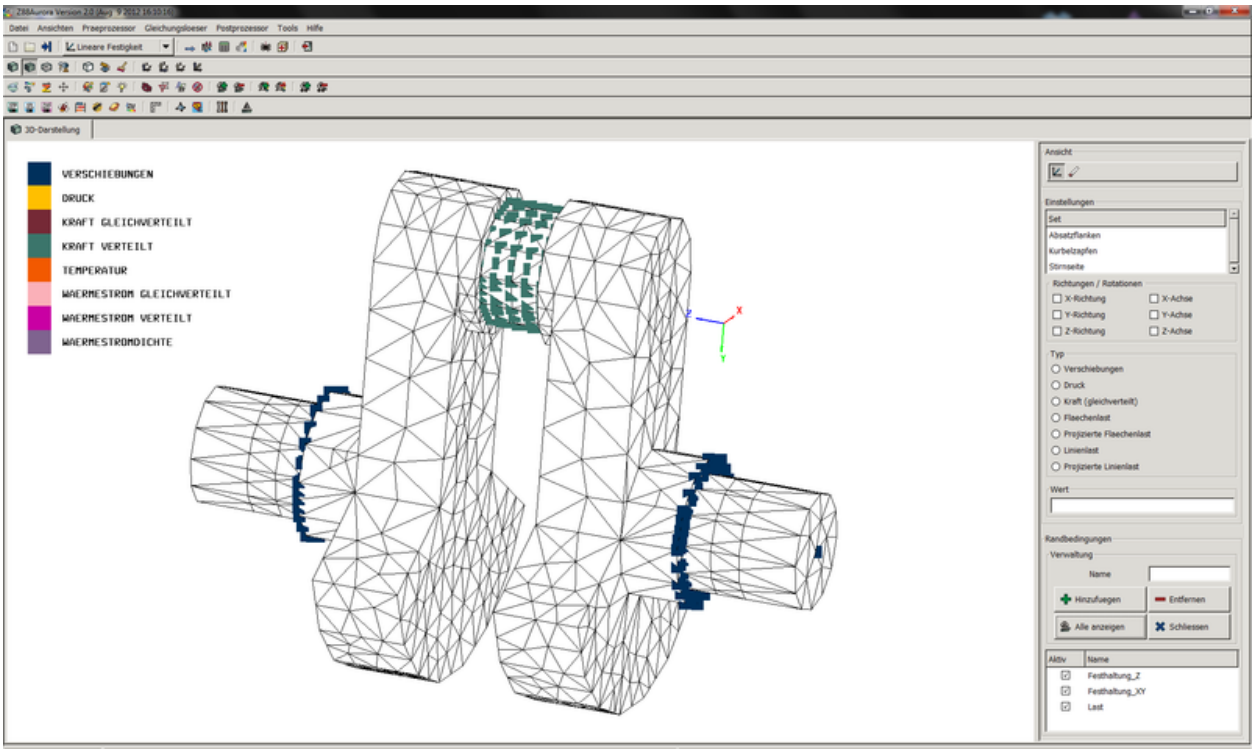

Figure A.4: A Simple Pre-Processed Part

This very serial workflow causes the single-user preparation of analysis models to frequently become a substantial bottleneck in the product design process [6]. Once the model is built (i.e., "pre-processed"), it is often exported from the pre-processor as a file which contains definitions for all of the nodes, elements, materials, loads, boundary conditions, and any other information pertinent for the model. This file is then given to the solving routine.

A.2 Solving

In a simple explanation, the solver calculates the desired results (i.e., displacement, stress, temperature, etc.) for each node and element by simultaneously solving many linear equations. Other results such as temperature, fluid flow, energy, etc., can be also obtained from appropriate models.

In a little more detail, the input deck is read by the solver, which then calculates a stiffness matrix for each element, based on the element's geometry and material properties. All of these elemental stiffness matrices are accumulated into a global stiffness matrix A, which is sparse and symmetric positive definite. All of the loads are accumulated into a vector b. The rows and columns of A and the rows of b that correspond to the constrained degrees of freedom (where the nodal displacements, temperatures, etc. have been constrained to zero) are removed, and the vector of nodal values (for example, nodal displacements in structural mechanics) x is found by solving Equation A.1:

$$Ax = b \tag{A.1}$$

Various numerical methods have been created to solve Equation A.1, including both direct and iterative approaches, for both linear and nonlinear models. For the sake of simplicity, we will assume that magic happens and x is found. If x represents displacements, then they are applied to the corresponding elements to find the elemental stresses. The results are then exported from the solver (generally as a file or a sequence of files) for post-processing.

A.3 Post-Processing

After the solver is finished, the results are read by the post-processor. Contour plots can be generated and displayed on the model showing the selected results. These may be interpolated across each element to allow for a smooth contour plot. The user can switch among loading scenarios and associated results sets, and interrogate the contour plot at locations of interest. Figure A.5 shows an example post-processing image which displays the results of a vorticity analysis in a jet engine.

A.4 Other References for FEA and Related Methods

There are many good references on FEA and other related methods. These include books, papers, journal articles, and various internet websites. More information about the overall FEA process and other analysis methods can be obtained from [13,35–37]. For more information on the mathematics of the Solving phase of FEA, [38–41] may be helpful. Other interesting alternatives

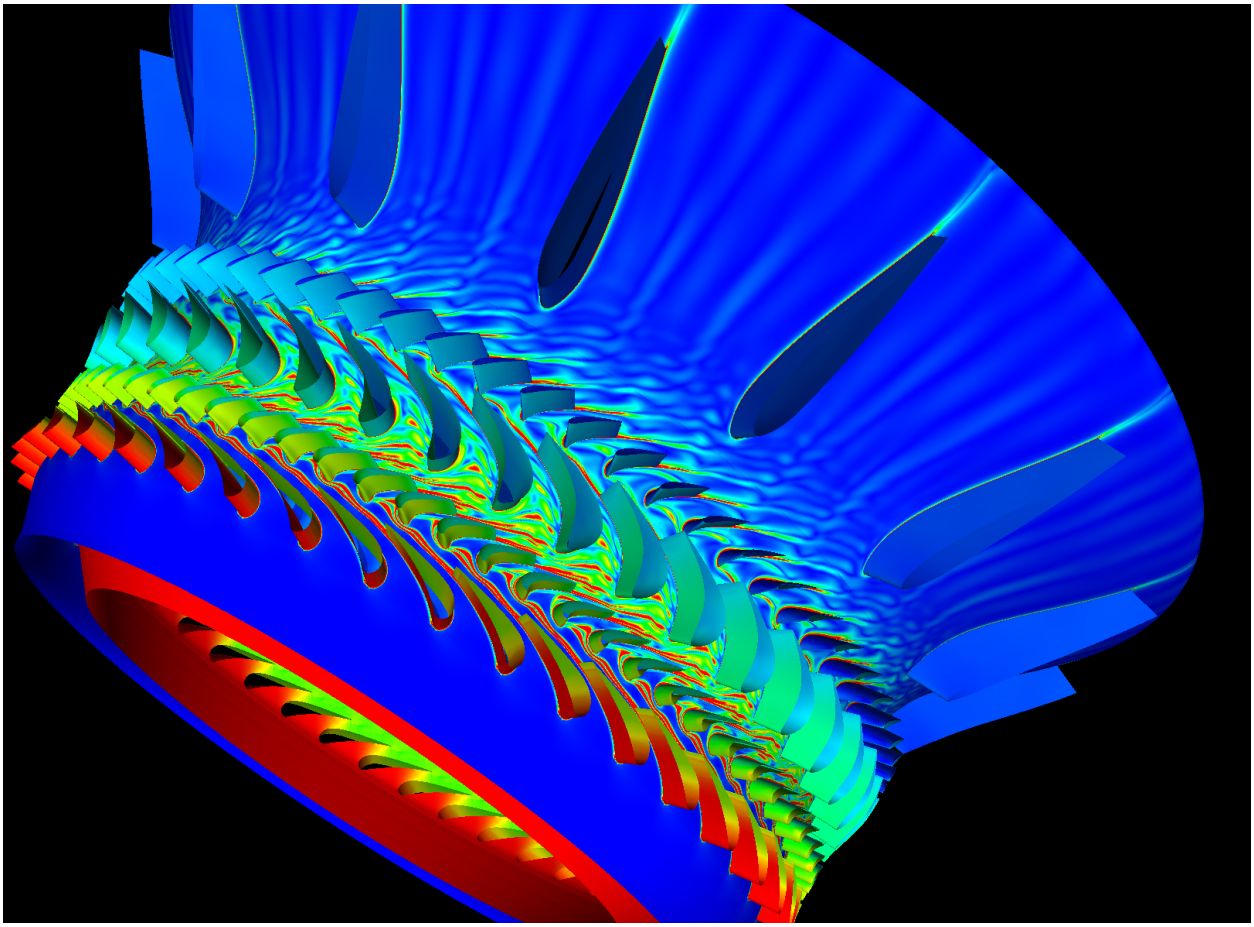

Figure A.5: Image of Analysis Results Showing Fluid Vorticity due to Jet Engine Blades

to traditional FEA include methods in Isogeometric Analysis (IGA), more information for which can be found in [42–44].

APPENDIX B. MUFE USER'S MANUAL

This chapter introduces the basic controls and commands for Multi-User Finite Elements (MUFE). The User Interface (UI) is shown in Figure B.1, where the Graphical User Interface (GUI) consists of the large blue rectangle on the left side of the UI. Buttons for controlling entity display are located along the top of the UI, a tree displaying information about various entities is on the right-center of the UI, and a screen where information and errors are displayed is located at the bottom-right.

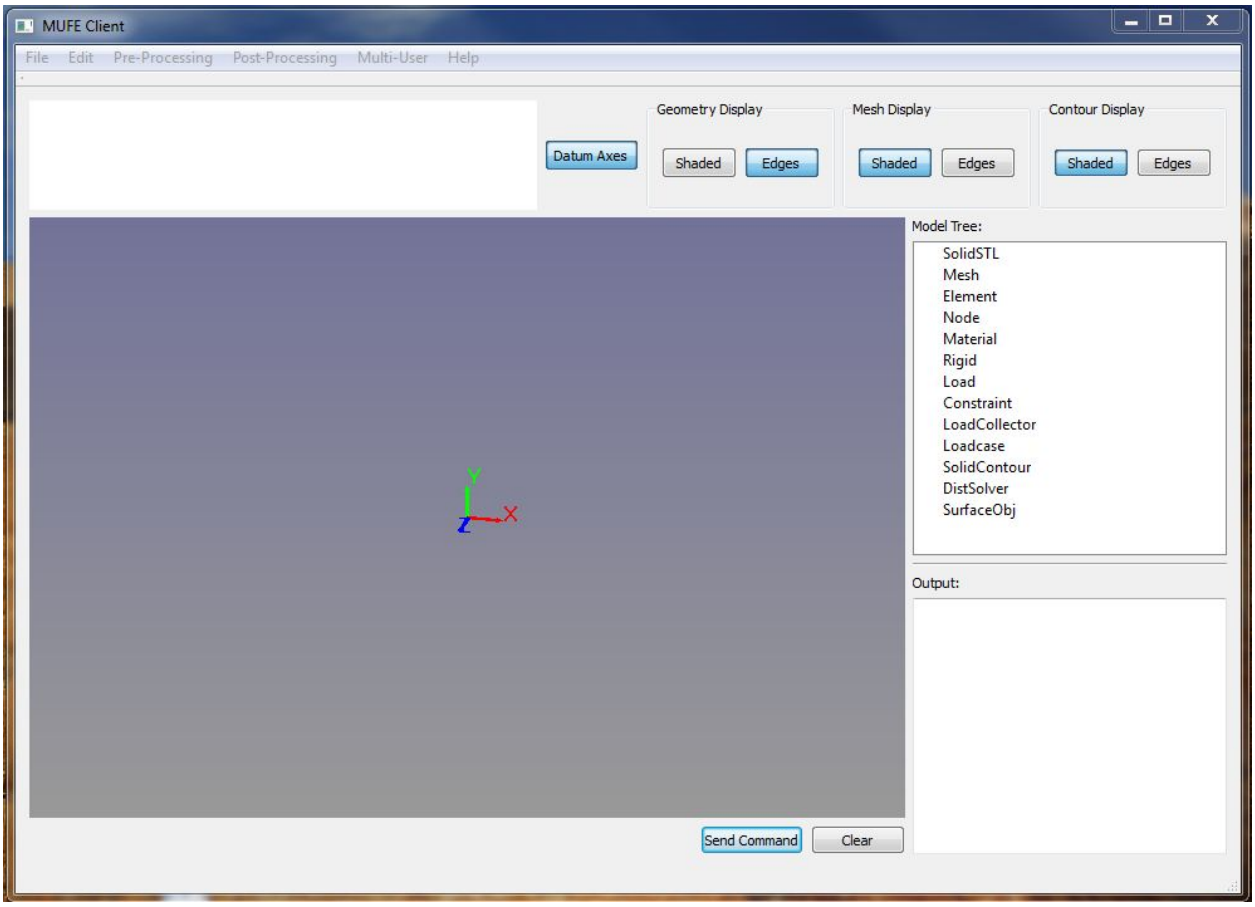

Figure B.1: MUFE User Interface

B.1 Starting MUFE

B.1.1 Single-User Mode (No Server)

To use MUFE in stand-alone configuration, where all processing is done on the local machine, the user can start MUFE using either of the following ways:

- Double-click directly on the MUFE client executable *MUFE_client_VS2.exe*.
- Double-click on the MUFE client batch file *MUFE_client.bat*.

B.1.2 Utilizing a Remote Server

In order to have command execution performed on a remote machine, or to work in multi-user mode, MUFE will need to be started on the server (the remote machine) as well as on each client. In both cases, MUFE will have to be started along with the networking agent. **PLEASE NOTE:** the MUFE server networking agent expects to have the IP address *10.2.114.65*. To change this, the agent will have to be recompiled (C# project *WuNetworkServer*) with a new address entered on line 81 of file *CubitConnectServer.cs*. Also, the MUFE client networker (C# project *ModefiedClient*) will need to be recompiled with the same address inserted at line 135 of file *Client.cs*.

B.1.2.1 Starting MUFE on the Server

The server version of MUFE does not have a UI. Instead, it has two command-style windows: one displaying information about the CubitConnect Server (which acts as a networking agent for MUFE), the other displays information about the MUFE processing server.

The MUFE server must be started and running before the MUFE clients can access it. To do so, double-click on the MUFE server batch file *MUFE_server.bat*. The user should see the two windows appear, displaying no errors. If not, check to make sure the batch file is pointing to both the *WuNetworkServer.exe* and the *MUFE_server.exe* (in that order), and that they have executable permissions.

B.1.2.2 Starting MUFE on the Client

Once the MUFE server is up and running, start each MUFE client by double-clicking on the MUFE client batch file *MUFE_client.bat*. The MUFE UI should appear. Under the *Multi-User* menu at the top of the screen, click on *Login*. This instructs the client networking agent to make a connection with the server's networking agent. If the connection is successful, the user will see notification of such in the output display at the bottom-right of the MUFE UI.

B.2 MUFE Control Overview

- **To Rotate:** Hold *Ctrl* + *Left Mouse Button* while moving the mouse within the blue graphics window.
- **To Pan:** Hold *Ctrl* + *Right Mouse Button* while moving the mouse within the blue graphics window.
- **To Zoom:** Hold *Ctrl* while scrolling with the *Middle Mouse Wheel* in the blue graphics window.
- **To Select an Entity:** Click with the *Left Mouse Button* on the desired entitiy when selection is necessary (you may need to rotate the model slightly first to force the GUI to redraw). Most entities will highlight once selected (nodes might not, depending on the version of MUFE).
- **To Select a Group of Entities:** While holding *Shift*, define the selection box by clicking with the *Left Mouse Button* once where you want the two opposing corners (again, you may need to first rotate the model slightly to force the GUI to redraw). Most entities will highlight once selected (nodes might not, depending on the version of MUFE).
- **To Toggle Datum Axes View:** Click on the *Datum Axes* button to turn on and off the viewable datum axes. Note that these are always centered at (0, 0, 0).
- **To Toggle Geometry Shading:** Clicking on the *Geometry Display: Shaded* button will turn on and off geometry shading.

- **To Toggle Geometry Wireframe Edges:** Clicking on the *Geometry Display: Edges* button will turn on and off geometry wireframe edges.
- **To Toggle Mesh Shading:** Click on the *Mesh Display: Shaded* button to turn on and off mesh element shading.
- **To Toggle Mesh Edges:** Click on the *Mesh Display: Edges* button to turn on and off mesh edges.
- **To Toggle Contour Shading:** Click on the *Contour Display: Shaded* button to turn on and off contour element shading (if there is no active contour, nothing will happen).
- **To Toggle Contour Edges:** Click on the *Contour Display: Edges* button to turn on and off contour edges (if there is no active contour, nothng will happen).

B.3 Model Import and Export

MUFE supports importing geometry as ASCII-based stereolithography (.STL) format, where the geometry is defined as a tessellated surface. MUFE can also read many NURBS curve and surface definitions from an IGES file, as well as save and read a MUFE log file that can be used to re-create the current model.

B.3.1 Importing a .STL File

To import an ASCII text-based STL file, go to the *File* drop-down menu, go to the *Import* sub-menu, and click on *.STL (text)*. A window will appear where the user can select the appropriate file, then click *OK*. A dialog will appear allowing the user to name the geometry being imported. Enter a name and click *OK*.

B.3.2 Importing NURBS from an IGES File

To import a NURBS definition via IGES file, go to the *File* drop-down menu, go to the *Import* sub-menu, and click on *IGES (v. 6)* (or press *Ctrl + I*). A window will appear where the

user can select the appropriate file, then click *OK*. A dialog will appear allowing the user to name the geometry being imported. Enter a name and click *OK*.

B.3.3 Saving a MUFE Log File

To save a MUFE file, go to the *File* drop-down menu, and click on *Save Model* (or press *Ctrl* + *S*). A window will appear where the user can specify the file to use, then click *OK*.

B.3.4 Opening a MUFE Log File

To open a previously-saved MUFE file, go to the *File* drop-down menu, and click on *Open Model* (or press *Ctrl* + *O*). A window will appear where the user can select the appropriate file, then click *OK*. A dialog might appear asking if it is okay to discard any changes in the current model. If that is acceptable, click *Yes*.

B.4 Creating and Editing Geometry

MUFE currently does not have any methods for creating geometry. Geometry can only be imported via the methods explained in Section B.3.

NURBS geometry can be edited by modifiying individual control points. To do this:

1. Go to the *Design* drop-down menu, and click on *Edit Control Point* (or press *Ctrl* + *E*). A dialog will appear.

2. In the GUI, click on the control point to be moved (depicted as a blue sphere). If the control point doesn't immediately turn white (highlight) upon selection, try rotating the view slightly and then selecting the control point again.

3. Once the control point is highlighted, click on *Refresh* in the *Edit Nurbs* dialog. This will populate the *X*, *Y*, and *Z* fields with the current location of the selected control point.

4. Enter the new location for the control point and click *OK* (click *Cancel* to cancel the operation without any changes).

B.5 Materials

MUFE currently only supports linear, isotropic materials.

B.5.1 Creating a Material

To create a material:

1. Go to the *Analysis* drop-down menu, go to the *Materials* sub-menu, and click on *Create Material* (or press *Ctrl* + *M*).
2. Enter a name for the material.
3. Enter a value for *Young's Modulus (E)*.
4. Enter a value for *Poisson's Ratio* (ν).
5. (Optional) Enter a value for *Density* (ρ).
6. (Optional) Enter a value for *Shear Modulus (G)*. If a value for shear modulus is entered, MUFE will calculate a value for Poisson's ratio to replace the one given earlier.

Note that MUFE currently does not support editing an existing material.

B.6 Meshes

A mesh is a discritization of geometry, for the purpose of approximating the behavior of the geometry under certain loading environments.

B.6.1 Supported Mesh Types

MUFE currently supports the following mesh types:

- **Quad4Node:** This is a structured surface mesh made by evenly dividing the parameter space of a NURBS surface into 4-node squares, where the smallest edge in each *u* and *v* paremeter-space determines the number of elements in that direction. This is done by dividing the shortest edge by the given mesh size and rounding to the nearest integer value.

- **Tri3Node:** This is a structured surface mesh made in a similar manner to Quad4Node, except each square is divided once more into 3-node triangles.
- **Tet4Node:** This is the only solid mesh that MUFE currently supports. It is an unstructured mesh consisting of 4-node tetrahedra. This mesh is created by calling the open-source mesh generator TetGen (as a separate executable), and giving it the solid's .STL geometry.
- **BezierPatch:** This is a surface mesh created specifically for Isogeometric Analysis (IGA). It consists of subdividing a NURBS surface into its respective Beziér patches, each of which corresponds to an element. The order of the element depends on the order of the underlying Beziér patches.

B.6.2 Creating a Mesh

To create a mesh:

1. Go to the *Analysis* drop-down menu, go to the *Meshes* sub-menu and select *Create Mesh* (or press *Ctrl* + *T*). A dialog will appear.
2. Select the material to use for the mesh from the *Materials to Use* drop-down menu.
3. Select the mesh type to create from the *Mesh Type* drop-down menu (see Section B.6.1 for information about the mesh types that MUFE supports).
4. If a surface mesh was selected, enter a desired thickness to be represented by the mesh in the *Thickness* field. If a the selected mesh type is Quad4Node, Tri3Node, or Tet4Node, then enter a desired mesh size in the *Mesh Size* field.
5. Enter other options as appropriate: *max Rad/Edge ratio* and *2nd-Order Elements* only apply to Tet4Node meshes, and are only partially supported.

MUFE does not currently support editing existing meshes.

B.7 Load Collectors

Load collectors bunch multiple and distinct loads into groups for applying to a particular loading scenario or boundary condition set.

To create a load collector, go to the *Pre-Processing* drop-down menu, enter the *Load Collectors* sub-menu, and select *Create Load Collector*. Enter a name for the collector and click *OK*.

MUFE does not currently support editing existing load collectors.

B.8 Loads

MUFE represents each load as a red line, one end of which is connected to the affected node. The magnitude and direction of the load are represented by the length and orientation of the line, respectively.

B.8.1 Creating Loads

MUFE only currently supports nodal loads. To create a load on a node or a number of nodes:

1. Go to the *Pre-Processing* drop-down menu, enter the *Loads* sub-menu, and select *Create Loads* (or press *Ctrl* + *L*).
2. Choose the load collector in which you want this new load to be included from the *Collector* drop-down box.
3. Set the *Magnitude* and *X*, *Y*, and *Z* direction components to the desired values.
4. Click *OK*.
5. In the blue graphics window, select the desired nodes (you may need to rotate the model slightly to force the GUI to redraw).
6. Click the *Send Command* button (near the bottom of the GUI).

MUFE does not currently support editing existing loads.

B.9 Constraints

MUFE represents each constraint as a yellow sphere, centered on the affected node.

B.9.1 Creating Boundary Conditions (Constraints)

To create a displacement constraint on a node or a number of nodes:

1. Go to the *Pre-Processing* drop-down menu, enter the *Constraints* sub-menu, and select *Create Constraints*.
2. Choose the load collector in which you want this new constraint to be included from the *Collector* drop-down box.
3. Check the degrees of freedom (DOFs) to which you want to enforce zero displacement.
4. Click *OK*.
5. In the blue graphics window, select the desired nodes (you may need to rotate the model slightly to force the GUI to redraw).
6. Click the *Send Command* button (near the bottom of the GUI).

MUFE does not currently support editing existing constraints.

B.10 Loadcases

A loadcase is a specific loading scenario, consisting of exacly two load collectors:

1. A load collector with loads.
2. A load collector with constraints.

This pairing tells the solver how to load and constrain the model, in order to achieve a useful solution.

There can exist any number of loadcases, each with a different pairing of loads and constraints, as grouped in load collectors.

B.11 Creating a Loadcase

To create a loadcase:

1. Go to the *Pre-Processing* drop-down menu, enter the *Loadcases* sub-menu, and click on *Create Loadcase*.

2. Enter a name for the loadcase (note that MUFE does not currently check if the name is valid or already in use).

3. Select the analysis type from the *Type* drop-down list (currently only ”Linear” is supported, so there are no other options).

4. Select the load collector that contains the desired boundary conditions in the *Constraint(s)* box.

5. Select the load collector that contains the desired loads in the *Load(s)* box.

6. Click *OK*.

MUFE does not currently support editing existing loadcases.

B.12 Rigid 1D Elements

A rigid element is used to rigedly connect distinct meshes, or to simulate welds, fasteners, or other rigid bodies. The user can specify what degrees of freedom are considered rigid, leaving the others free. MUFE represents each of these elements as a set of blue lines, where one end of each is connected to the associated dependent nodes, and the other ends all join at the independent node (which is either designated by the user or calculated as the average position of the dependent nodes). This is commonly referred to as a rigid ”spider” element, due to the radiating effect of the element’s ”legs”.

B.12.1 Creating a Rigid Element

To create a rigid element:

1. Go to the *Pre-Processing* drop-down menu, enter the *Rigids* sub-menu, and select *Create Rigid*.

2. If you want MUFE to place the independent node at the average location of all of the dependent nodes, click on the *Calculate Ind. Node* tab. If you wish to designate an independent node, click on the *Select Ind. Node* tab.

3. (If selecting the independent node) In the blue graphics window, select the independent node (you may need to rotate the model slightly to force the GUI to redraw).

4. (If selecting the independent node) In the *Define Rigid* dialog box, select the first *Selected* button.

5. In the blue graphics window, select the dependent nodes (you may need to rotate the model slightly to force the GUI to redraw).

6. In the *Define Rigid* dialog box, select the *Selected* button.

7. Click *OK*.

MUFE does not currently support editing existing rigid elements.

B.13 Solving

MUFE orignially supported exporting models with a limited set of functionality to, and reading results from, both Altair Engineering's OptiStruct and the open-source, Abaqus-style solver CalculiX. However, due to various changes in the code without proper attention being paid to the associated methods, the solving and post-processing capability of MUFE is currently broken (as of September, 2013).