APP CHAINING SOFTWARE-AS-A-SERVICE FOR AN ADVANCED MANUFACTURING MARKETPLACE

A Thesis
presented to
the Faculty of the Graduate School
at the University of Missouri

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

by
AMIT KUMAR RAMA AKULA

Dr. Prasad Calyam, Advisor

MAY 2015
The undersigned, appointed by the dean of the Graduate School, have examined the thesis entitled

APP CHAINING SOFTWARE-AS-A-SERVICE FOR AN ADVANCED MANUFACTURING MARKETPLACE

Presented by Amit Kumar Rama Akula

A candidate for the degree of

Master of Science

And hereby certify that, in their opinion, it is worthy of acceptance.

____________________________________
Dr. Prasad Calyam

____________________________________
Dr. Rohit Chadha

____________________________________
Dr. Praveen Edara
ACKNOWLEDGEMENTS

I would like to thank Prof. Prasad Calyam for mentoring me and being my advisor during my thesis research. I would also like to thank him for providing me an opportunity to be part of Virtualization Multimedia and Networking Lab. I am extremely grateful for his constant guidance, insight and encouragement through the entire course of my research. I would like to express my gratitude to Prof. Rohit Chadha and Prof. Praveen Edara for their interest in my research and consenting to be part of my thesis committee.

Special thanks to Ronny Bazan for his invaluable help in the test bed setup for my project. I would also like to thank Shravya Ramisetty as the work done by her was a stepping stone to my thesis research.

I would also like to thanks to the Department of Computer Science, University of Missouri for providing the opportunity and resources to pursue and complete my Master’s.
TABLE OF CONTENTS

Acknowledgements..............................................................................................................ii

List of Figures......................................................................................................................v

List of Tables........................................................................................................................vi

Abstract................................................................................................................................vii

1. INTRODUCTION .............................................................................................................. 1
   1.1 Advanced Manufacturing and Cloud ............................................................................ 1
   1.2 Advanced Manufacturing App Marketplaces ............................................................... 3
   1.3 App Runtime for the Marketplace Environment .......................................................... 3
   1.4 Thesis Outline .............................................................................................................. 4

2. BACKGROUND AND LITERATURE SURVEY .......................................................... 5
   2.1 Case study: ‘WheelSim’ ............................................................................................... 5
   2.2 Literature Survey ......................................................................................................... 11
      2.2.1 Advanced Manufacturing using cloud ................................................................. 11
      2.2.2 Cost Model for cloud based services ................................................................. 12

3. APP RUNTIME FRAMEWORK .................................................................................... 14
   3.1 Architecture ................................................................................................................. 14
   3.2 App Runtime framework ........................................................................................... 15
      3.2.1 Core Applications in the App Runtime ............................................................... 15
      3.2.2 Functions in the App runtime .............................................................................. 17

4. ACCOUNTING SERVICE AND COST MODEL FOR APP MARKETPLACE .......... 20
   4.1 Accounting Service .................................................................................................... 20
   4.2 Cost model and Pricing Strategies .............................................................................. 21
      4.2.1 Service Provider Cost & Pricing model ............................................................... 22
      4.2.2 Manufacturing Enterprise Cost & Pricing model .............................................. 24

5. APP RUNTIME WEB FRAMEWORK .......................................................................... 26
   5.1 App creation ............................................................................................................... 27
   5.2 App Publishing ........................................................................................................... 28
   5.3 App Addition .............................................................................................................. 29
5.4 App execution ................................................................. 29
6. COST MODEL CASE STUDIES ............................................ 31
   6.1 Cost Performance Model .............................................. 31
   6.2 Cost Pricing Model ..................................................... 35
7. FUTURE WORK ............................................................... 42
8. CONCLUSION ................................................................. 44
9. BIBLIOGRAPHY ............................................................... 46
LIST OF FIGURES

Figure 1.1. Illustration of the SMaaS workflow…………………………………………2
Figure 2.1 Cloud architecture for WheelSim architecture………………………………5
Figure 2.2 Workflow of the ‘WheelSim ’web based Advanced Manufacturing App……6
Figure 2.3 Workflow of modularized Manufacturing Apps that are chained together to
create complex workflows……………………………………………………………………7
Figure 2.4 Modularized flow of different phases of the Wheel simulation…………10
Figure 3.1 Architecture of the App Marketplace……………………………………….15
Figure 3.2 Workflow of App Runtime Manager………………………………………….19
Figure 5.1 Testbed on which App Runtime is deployed……………………………….27
Figure 5.2 A workflow for App creation using the web framework……………………28
Figure 5.3 A workflow of App publishing using web framework………………………28
Figure 5.4 A workflow for App addition using the workflow…………………………29
Figure 5.5 A workflow for App execution using the web framework………………….30
Figure 6.1 Plot indicating the time taken to complete a job for different processors…..34
Figure 6.2 Plot indicating the time taken to complete a job for different processors…..35
Figure 6.3 Plot indicating the cost vs. revenue for variable number of App runs……..37
Figure 6.4 Plot indicating the cost vs. revenue for variable number of RUs consumed…………………………………………………………………………………………….39
LIST OF TABLES

Table 1.1 Comparison of Local and cloud based approach to manufacturing...........2

Table 6.1 Cost of acquisition and operational cost.............................................32

Table 6.2 Metrics for the variable cost...............................................................32

Table 6.3 Cost and the execution time for running a simulation with different number of processors..................................................................................................................34

Table 6.4 Cost vs. Revenue based on number of App runs...............................36

Table 6.5 Cost vs. Revenue based on number of RUs consumed.........................39
ABSTRACT

Advances in the field of cloud computing and networking have led to development of Marketplaces (e.g., Awesim) that support Advanced Manufacturing enterprises. These Marketplaces host Apps that perform simulation and modeling on specialized designs (e.g., pipes, automobile parts). However, the salient limitation in these App Marketplaces is the lack of a development environment that supports effective runtime capabilities for ‘Agile Manufacturing’ that efficiently and cost-effectively integrates several Apps when building innovative products.

To address this problem, we propose a new Software-as-a-Service based App Runtime for the Marketplace environment that can be utilized for agile development of ‘Apps’ that involve high-performance modeling and simulation. Our solution approach features a web framework for the App runtime that: (a) builds upon the abstractions for most common workflows to support management of a generic set of ‘Apps’, (b) provides a chaining mechanism to create complex workflows for creation of new Apps based on customer requirements, and (c) runs complex simulation jobs on remote supercomputer resources and publishes customer-facing results for specific sets of inputs. We demonstrate how multiple Apps can be chained using our web framework for a product case study viz., ‘WheelSim’ deployed in the NSF GENI Cloud platform. Our results show improved App development convenience via rich UI elements interacting with RESTful web services and through dynamic chaining of workflows. Our study also provides App developers with insights pertaining to estimation of resource cost, App pricing issues relevant to a manufacturing Marketplace, and the corresponding performance achievable with common App configurations in a cloud environment.
1. INTRODUCTION

1.1 Advanced Manufacturing and Cloud

Advances in the field of cloud computing and networking have led to cloud adoption and migration in several fields like Healthcare, Gaming, Online Retail, Gaming etc. These advancements have helped in creating economic benefits by transforming investment in infrastructure. Enterprises can now rent infrastructure on-demand without the hassle of frequent maintenance or upgrades. They can also access high performance computing and elastic resources to collaborate with their peers and improve service delivery to customers [1].

In the field of Advanced Manufacturing, however the adoption of cloud technologies is still in its early stages. Advanced Manufacturing today require platforms that are capable of fostering community engagement for advanced manufacturing design (e.g., fluid/thermal analyses), which typically requires collaborative work among multi-site engineering experts. To leverage cloud services and minimize cost, there is also a need to transform traditional workflows that feature data-intensive computation and networking during design and development of new domain applications. This adoption had led to a new type of service offered through the cloud called Simulation-as-a-Service (SMaaS). The typical workflow of the SMaaS is illustrated in Figure 1.1. It is essentially an iterative process where modeling and simulation phases essentially result in delivery of Apps and physical improvements in products. The cloud transformation of the manufacturing domain provides several benefits.
These benefits are illustrated in Table 1.1. The use of cloud technologies will greatly benefit and transform the Advanced Manufacturing domain. The initial stages of adoption have led to several marketplaces that house Advanced Manufacturing Apps. We now look at some of them.

<table>
<thead>
<tr>
<th>Today: Local/home-grown</th>
<th>Future: Cloud/apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Screen capture/Email</td>
<td>WebEx on virtual desktop</td>
</tr>
<tr>
<td>- Asynchronous collaboration, physical Win7 PC access required</td>
<td>- Synchronous collaboration, Cross-platform/device operation</td>
</tr>
<tr>
<td>Lots of data copies</td>
<td>Cloud storage</td>
</tr>
<tr>
<td>- Mail DVDs, SCP</td>
<td>- Mass storage, Box.net</td>
</tr>
<tr>
<td>Slow data transfer</td>
<td>- Data security and control</td>
</tr>
<tr>
<td>- Public Internet</td>
<td></td>
</tr>
<tr>
<td>Local cluster procurement</td>
<td>Fast data transfer</td>
</tr>
<tr>
<td>- Time consuming, overprovision</td>
<td>- Extended VLANs, GENI</td>
</tr>
<tr>
<td></td>
<td>Elastic compute access</td>
</tr>
<tr>
<td></td>
<td>- Can handle demand bursts</td>
</tr>
</tbody>
</table>

Table 1.1 Comparison of Local and cloud based approach to manufacturing
1.2 Advanced Manufacturing App Marketplaces

The initial adoption of cloud for Advanced Manufacturing domain has led to the development of App Marketplaces that deliver SMaaS Apps to the customers. There are now several App Marketplaces like AweSIM [2], Nimbis [3], and Amazon Marketplaces [4] that deliver these SMaaS Apps. In our previous work [5], we analyzed the architecture of a generic App Marketplace. In this work, we explored the different services at infrastructure, platform and software level to design an App Marketplace. We also developed an App ‘WheelSim’ that performed simulation on the front wheel of a vehicle. These Marketplaces have a lot to offer however they lack a development environment to support effective runtime capabilities for ‘Agile Manufacturing’ to efficiently and cost-effectively integrate several Apps when building innovative products.

1.3 App Runtime for the Marketplace Environment

To overcome the limitations in the existing Marketplaces, we propose a new Software-as-a-Service based App Runtime in this thesis that can be utilized for agile development of ‘Apps’ that involve high-performance modeling and simulation. Our solution features a web framework that abstracts the most common workflows to support management of a generic set ‘Apps’. These Apps are the core applications of the Runtime and can be used to create complex workflows through a chaining mechanism which is also a part of the Runtime. The Runtime also runs complex simulation jobs on the supercomputer resources that publish customer-facing results for specific sets of inputs. Through the use of this Runtime, we show how multiple Apps can be chained together for a product case study viz., ‘WheelSim’ deployed in the NSF GENI Cloud platform. We also discuss a
cost model for the Marketplace environment and present case studies that will help
Manufacturing Enterprises in migrating their complex workflows in the cloud.

1.4 Thesis Outline

The remainder of this thesis is organized as follows: In Chapter 2, we describe the
thesis background that provides context to the problem and the solution approach.
Additionally a detailed literature study is presented. In Chapter 3, we provide a detailed
description to our solution approach with reference architecture. In Chapter 4 we discuss
the Accounting Service and the associated Cost model for the Marketplace. In Chapter 5
we present the web framework for the Runtime. Chapter 6 presents a detailed analysis of
the cost model case studies. Chapter 7 discusses future work and Chapter 8 concludes the
thesis.
2. BACKGROUND AND LITERATURE SURVEY

In this chapter, we first describe a case study for the product ‘WheelSim’ which led to the idea of the App Runtime and its essential components. We also present specific literature work conducted on the cost accounting for different type of cloud architectures.

2.1 Case study: ‘WheelSim’

To design and develop our App Runtime, we look at the workflow of an existing manufacturing App as a case study. This App was a part of a cloud architecture proposed...
in our previous work as shown in Figure 2.1. This work aimed at developing a unique environment for an App Marketplace using existing disconnected services. The App performs simulation of lift and drag forces on the front wheel. The workflow is shown in Figure 2.2. This App obtains user input through a portal and executes some scripts that complete the simulation at the Supercomputer.

**Figure 2.2 Workflow of the ‘WheelSim ‘web based Advanced Manufacturing App**

The results are displayed back to the user in their browser. This kind of workflow has certain limitations however. One of the limitations is that the App works on a fixed geometry i.e. the wheel. This itself limits the capability of the App. The code developed works for a wheel simulation only. Another limitations is that if in the near future, another App that is similar to the existing App has to be developed that performs simulation on a related geometry (e.g. an automobile), the existing code for the wheel application is unused though the process and workflow remain the same. This leads to wastage of time and no code reuse.

Additionally this App also integrates common workflows into a complex workflow without any proper validation. This could lead to resource wastage in case of failure of any intermediate workflow. Hence to overcome these problems, we look at developing a dynamic runtime that addresses these limitations. Our runtime would convert this workflow into a modularized workflow resembling Figure 2.3. In this case, a set of core
applications work together through a chaining mechanism to convert existing complex workflows into a sequence of simple chained workflows. The challenge here is to identify these core applications.

![Image](image.png)

**Figure 2.3 Workflow of modularized Manufacturing Apps that are chained together to create complex workflows**

Identification of the Apps is an important step towards building the runtime and hence to this end, we attempt to dissect the current App that performs wheel simulation and understand its inner workings to determine the core applications. We primarily inspected the scripts that ran the simulation. From analysis of the scripts, we concluded that a complex modeling and simulation workflow can be fundamentally divided into four main phases. They are: 1. Mesh generation phase, 2. Simulation phase, 3.Export phase, 4.Post-processing phase. Each of these phases are described below.

1. The Mesh generation phase is also termed as Pre-processing phase. In this phase, a case that is required to be simulated (for example a wheel) is defined using text files and any text editor can be used for defining this case. To define a case, we are required to describe its attributes like mesh, boundary, physical properties and control data. Mesh data is typically data about the composition of simple polygons that represent an object. In our example the object is the wheel and mesh is a composition of polygons that form this object. Boundary data refers to initial and boundary conditions of the simulation which describes the environment.
of the mesh and its interaction with the mesh. Based on the type of simulation that is to be run on the case, several physical properties may be applicable like viscosity, drag etc. Control data usually defines the control of time like start and stop times. This also defines the sampling at which simulated data is to be captured that can be viewed later in the post-processing phase. Post completion of this phase, there is sufficient data about the mesh and its environment that enable validation of the input provided. The output of this stage is the mesh in its environment that can be viewed and validated for its accuracy, construction of the geometry and resolution. In our case, the output would be a geometry of a wheel and parameters for simulation setup would be the lift and drag forces acting on it for a specific velocity.

2. The Mesh generation phase sets the stage for the Simulation phase by defining all necessary parameters that are essential to perform the simulation. In this phase, a suitable program is selected based on the problem and geometry. There are specific programs called “solvers” that perform different type of simulations and hence based on the geometry an appropriate solver is selected. As simulation is a very computationally intensive task, sequential execution is time consuming. OpenFOAM facilitates parallelization of this process which can greatly reduce the time consumed if access to HPC resources are available. Our infrastructure supports a Supercomputer and hence it is possible to complete this task in a reasonable amount of time. The Simulation phase performs simulation of the conditions on the object specified. In our case, this step would perform simulation on the wheel with specified geometry to obtain the resultant lift and drag forces.
acting on it when the wheel runs at a specified velocity. The raw simulation data is thus generated.

3. The Export phase follows up after the Simulation phase or solve phase where it processes the raw simulation data and converts it into a suitable format that can be visualized in 3-D using applications that support VTK (Visualization Toolkit). Example of such an application is ParaView. Based on the visual data that is to be observed, specific filters can be applied to the data to present different aspects like heat and effects of velocity. Using these files, the User can interact and move the model and observe it from various angles. The output of this phase is the simulation results stored in suitable formats that can be visualized by the user with applications.

4. The Post-processing phase occurs after the simulation phase. This is usually an optional phase but is preferred if different views and angles of the object post simulation are required to be captured and stored. The files generated in Export phase with the help of tools like ParaView can be used to visualize data. The Post-processing phase, based on the configuration generates sets of PNG (Portable Network Graphics) files of different angles and filters to facilitate easy visualization in case of unavailability of appropriate software. They can also be used as references in documentation of the simulation that provide validation to the geometry and the design. The output of this phase is hence a set of images that convey the results of simulation in a universal way without any software overhead as this format is readable on any platform. In our case the output produces a set of PNG files with different angles and different filters.
Figure 2.4 Modularized flow of different phases of the Wheel simulation

These are the four main phases of the simulation and modeling for any kind of simulation. Upon further reflection, we concluded that these phases can be abstracted into functional applications that can perform any kind of simulation tasks based on the inputs they receive. These apps are Mesh & Solve, Export, Post-processing and Status/Chaining.

We combine the Mesh and simulation phase into one App for simplicity but they can be decoupled into two different Apps as well. The workflow of WheelSim with these modularized Apps are shown in the Figure 2.4. The new apps would define a configuration that would reflect the requirement and run those applications that would most fit the needs of the user. Hence the Wheel simulation would actually encompass all the applications. If developed with a generic implementation, these apps as a part of a Runtime can be used to implement applications which can use other geometries based on the requirement which can drastically reduce the design, development and deployment.
cycles of advanced manufacturing applications from several weeks to a few clicks. This Runtime would also facilitate reuse of existing infrastructure for similar applications as determined by the Ontology Service by ensuring that each new app developed would run in its own sandbox environment. Additionally if this Runtime would also allow these Apps to be published in a Marketplace, it would benefit both the developers who build these applications and customers who may not have expertise in the field but have a specific use case for advanced manufacturing which they find already implemented in the Marketplace.

2.2 Literature Survey

In this section, we describe the detailed literature survey that was conducted to understand the previous work done in this domain and the requirements for designing a cost model for cloud based architecture resources.

2.2.1 Advanced Manufacturing using cloud

The cloud use case for Advanced Manufacturing is in the early stages of adoption but extensive research work has been done to explore the application and migration to the cloud. Authors in [11]-[18] have focused on using cloud in the context of manufacturing in areas like Collaborative Engineering, Machine Tool Industry, Prototyping Manufacturing System, Simulation etc. These interesting use cases have pushed the horizon of Advanced Manufacturing into the cloud. Some of these works focus on translation of Manufacturing application requirements into cloud resources mapping using the concept of Ontology like [5],[11]. Ontology essentially is knowledge
representation of data and has been successfully used in area of Software development [9] and Service discovery [10].

Once the resource mapping is done, the next step would be to deploy these resources. Authors in [19]-[23] have explored Cloud Resource Broker;ing to help users procure resources taking into account the user’s needs like location, cost constraints, vendor etc. Research work done in [7],[8],[24] inspired us to design this App Runtime framework in the context of Advanced Manufacturing using cloud resources.

2.2.2 Cost Model for cloud based services

Extensive research work has been done in the derivation of suitable cost models for a wide variety of cloud architectures. For the formulation of our cost model, we mainly refer the work done in [26]-[29].

Authors in [26] have presented a detailed comparison of running an in-house versus running one in Amazon. The Authors set up a local cluster and configured a cluster in the cloud using Amazon Web Services. They then ran MPI applications and analyzed the performance and the cost. They concluded that running a cluster in the cloud while viable causes delays due to latency issues. This led us to choose a local Supercomputer resource but that charges us based on our usage essentially being a pay-as-you-go model.

Authors in [27] compare and contrast Amazon EC2 and a custom local cloud computing infrastructure called ‘Spring’ to determine the pricing fairness of the current cloud computing domain. They assembled a set of workloads and derived a cost model to determine the actual costs of the provider compared to the pricing charged to the
customer. They concluded that there is pricing unfairness to the customers. This led us to explore how pricing can be made fair to the Manufacturing Enterprises that use cloud infrastructure.

Authors in [28] perform an in depth analysis for common cloud computing environments. Costs were categorized as fixed and variable costs. From this paper we began to incorporate the cost model in terms of fixed costs and variable costs. We also gained keen insight into the composition of these costs. Authors in [29] present a recommendation framework that provisions hybrid cloud resources based on user preferences. It presents an interesting cost model for the Virtual Machine (VM) usage, which we directly incorporate into our model as we allow access of VM with specific software to the Manufacturing Enterprises which they can use to develop Apps and view results. Based on these studies, we formulate our cost model which is unique to our environment.
3. APP RUNTIME FRAMEWORK

In this section, we first describe the architecture on which the App runtime framework is built which is essentially the App Marketplace. This Marketplace is composed of infrastructure, platform and software services which cumulatively provide a dynamic environment for App developers and App consumers to exchange expertise in the domain. We then describe the software framework for the App runtime that performs chaining and enables the creation, monitoring, publication and execution of Apps.

3.1 Architecture

The architecture on which the framework is built is composed of services that various levels as shown in Figure 4.1. We first have the services at the Infrastructure level i.e. Infrastructure-as-a-Service (IaaS) which provide users hardware, software, servers, storage and other infrastructure components for their applications. There are several such providers like Amazon, IBM, and GENI which enable users to reserve resources. In our study we use GENI – an NSF funded public cloud infrastructure and the components covered with red border are the primary contributions to this thesis. To convert the resources available from the IaaS into a coherent environment, we need to be able to gather requirements, allocate resources and provide a means for the clients to keep track of resource usage. We identify three core platform services that are essential to perform these tasks, namely: (i) Ontology Service, (ii) Resource Brokering Service, and (iii) Accounting Service. These services can deploy a development environment for engineers to collaborate with their clients, and allow them to use this environment for agile
development of manufacturing Apps. We also have an App Runtime composed of core applications and functions that help in the creation and distribution of these Apps. We now explore this Runtime in detail in the following section.

![Figure 3.1 Architecture of the App Marketplace](image)

**3.2 App Runtime framework**

In this section we explore in detail the various parts of the App Runtime framework.

**3.2.1 Core Applications in the App Runtime**

Core Applications are the basic apps that can be used in combination to create new applications. As discussed earlier we can abstract the phases of the simulation job and each of these phases are now implemented as a core application. As these apps are related
to each other, output from one app is redirected and served as input to the next thereby simplifying the app creation process. Based on the type of application, the monitoring of the jobs scheduled on the Supercomputer and result retrieval would be unique hence a new application was added that checks the status and retrieves results in accordance with the application. Hence the Core Applications comprise of four applications: 1. Mesh & Solve app, 2. Export app, 3. Post-processing app, 4. Status app

1. Mesh & Solve app: This app receives essential information from the user that are necessary to generate a mesh with its specific boundary conditions and also input that determines the simulation parameters. The input is obtained in the form of a web form from the user. User also has the option to choose a default mesh specification (wheel) or upload a custom mesh specification based on their requirement. Based on the input provided, appropriate solvers are selected and the application begins the process of mesh generation. After the mesh generation, the simulation is carried out on the use case scenario i.e. checking the lift and drag forces on the object specified. The output of this application is the mesh and the raw simulation data.

2. Export app: This app receives input from the Mesh \& Solve app, the input being the generated mesh with the simulation data. The input necessary is pulled from an isolated storage location of the Supercomputer and processed. This location is isolated and each user has a unique location. Based on this information, the Export app beings the process of converting the raw simulation data into 3-D models. Upon completion, the app generates data that is compatible with VTK
format and hence can be viewed in specialized software like ParaView which is not commonly distributed and has to be installed.

3. Post-processing app: This app receives input from the Export app. The input in this case are the files that store the simulation data. This input is pulled from the same unique location as in the case of the Export app. After gathering the input data, this app performs post processing which essentially converts the files with simulation data into image data (PNG) which are universally readable from any OS.

4. Status app: This app is the connecting link between the other three applications. This is the App that is responsible for chaining the applications. It monitors the status of the currently running application, checks for its completion and then runs the next app. It also retrieves the appropriate results from the Supercomputer after the execution of an app.

Additionally given the unique combination of the three phases to produce relevant results, there would essentially be three app configurations/workflows to any new application that combines the core application to produce results required by the user. The Apps discussed so far are isolated and require a medium to be used. This requires the App Runtime to have certain functions in order to successfully utilize the potential of these Apps. We discuss these functions next.

### 3.2.2 Functions in the App runtime

Functions in App Runtime help in creation, addition, monitoring, publishing and deletion of the customized applications created by the users. These functions are implemented as
REST web services. There are essentially 5 functions based on five actions described above.

1. Create: The Create function was implemented to facilitate creation of new customized applications. This function would gather input from the user about the application configuration, whether to use custom or default mesh, the application name description etc. and create an application with the required setup, isolation and initial information. The relevant data is added to the database. It also reserves storage at both the web server and the Supercomputer.

2. Add: The Add function was implemented so that customers can add existing Apps to their accounts. This creates a user specific instance of the App through which the customer can run their App in isolation. The users can delete their instance of the App using the Delete function.

3. Delete: The Delete function was implemented to provide the App owners and App users to create and delete their Apps. The user of the application sends a delete request to the Agent. The Agent removes any storage information of the app from the web server and the Supercomputer and also removes entry from the database. App users i.e. the customers can only delete their instance of the App and only the App owner i.e. the Manufacturing Enterprise developer can delete the actual application.

4. Publish: The Publish function was implemented to facilitate Manufacturers to publish their developed applications into the App Marketplace. Based on the input, this function can either publish an unpublished App or hide a published App. The Marketplace would then be updated with the current list of the Apps.
5. **Monitor**: This function is responsible for keeping track of the resource usage of the user. This function monitors changes in the resource utilization. It then invokes the Accounting service for cost computation.

Based on the action performed by the user, the Runtime uses a general workflow to invoke these functions as shown in Figure 4.2. The user essentially executes one of the actions described above through the web portal. The App Runtime Manager invokes the appropriate function from a library based on the input. The code invoked by the manager performs relevant updates to the database and updates the information in the marketplace.

![Figure 3.2 Workflow of App Runtime Manager](image)
4. ACCOUNTING SERVICE AND COST MODEL FOR APP MARKETPLACE

In this section, we first discuss the Accounting Service that is responsible for accounting the costs of the various entities involved. This is a platform level service that works with other services and the App Runtime. The Accounting Service calculates the costs based on a cost model. In this work we propose a cost model for our unique architecture.

4.1 Accounting Service

The Accounting Service is the book keeper of this environment. It tracks the utilization of the environment by the various entities and charges them a fair price based on the cost model. The cost model is discussed in the following section. This service is invoked each time there is a change in the usage of the system by an entity. It monitors usage by both Manufacturers and their customers. Compute resource usage, VM usage etc. are few of the resources monitored by this service.

The service could be invoked by another platform service or it could be invoked by an Agent that updates this service about usage information. The invoker carries essential information about the resource for which the usage is to be accounted for. Based on the resource, the service queries the cost model which stores the information about the various resources and the related cost associated with each of the entities. This information is processed by the service and existing information about the charges to that entity is updated in a database that is under the control of the service. Based on the preferred cycle, the service invokes related services to generate bills for the entities. The cycle could be weekly, bi-weekly, monthly etc. This service ensures the continuity of the
system by ensuring that the SPs revenue goals are met from the usage of this environment by the Manufacturers.

4.2 Cost model and Pricing Strategies

The App marketplace built over our infrastructure is incomplete without the discussion of the cost model to account for the costs of various entities that utilize this infrastructure. We define three main entities that are directly related to our Architecture:

1. Firstly we have the Service Providers (SP) who have setup the App marketplace environment composed of infrastructure and services that bring together other interested parties to engage in sale and purchase of Apps that run on our Architecture.

2. Secondly we have the Manufacturing Enterprises (ME) that utilize this environment to create, publish and sell advanced manufacturing Apps.

3. Additionally we also have the Customers(C) that use the Apps developed by the MEs.

These three entities define the cost model for the system. Utilizing this environment while beneficial must also account for costs associated with building, setting up the infrastructure and the software used. We thus need to develop a cost model to account for the costs incurred by the SP and the MEs and also address the pricing issue for both of them. Upon closer investigation, we find that the discussion in [28] & [29] are more relevant to our architecture but cannot be directly integrated. We hence use this as a starting point for our model and customize it for our requirements.
4.2.1 Service Provider Cost & Pricing model

The SP does all the grunt work in the system and the range of tasks vary from setting up the infrastructure to deploying the software stack that runs the marketplace. Hence there would be an initial cost of acquisition and then monthly recurring operational cost for compute resources like servers, network etc. Depending on the contract and provider, the network cost could be an unlimited usage fixed monthly cost or variable cost based on /Gb/hour. In our case we consider this as a fixed cost i.e. /month recurring monthly. In our environment, there is also a variable cost associated with usage of VMs and usage of Supercomputers in terms of Resource Units (RUs) by the MEs. Hence the total cost incurred would be a combination of fixed and variable costs and can be depicted in the following equation (1)

\[ C_{Total} = C_{acq} + C_{fixed\_monthly} \cdot T + C_{variable\_monthly} \cdot T \]  

(1)

Here \( C_{acq} \) is the cost of acquisition of hardware & software resources, \( C_{fixed\_monthly} \) is the monthly cost incurred by the SP toward operations of the system, \( C_{variable\_monthly} \) would be the summation of supercomputer usage and VM usage, \( T \) is the total months of operation. We further expand \( C_{variable\_monthly} \) as follows:

\[ C_{variable\_monthly} = C_{Supercomputer\_usage} \]  

(1)

\( C_{Supercomputer\_usage} \) is the cost of total usage of Supercomputer resources by all the MEs in terms of Resource Units (RUs) consumed which can further be broken down as:

\[ C_{Supercomputer\_usage} = C_{RU} \cdot Total\_RUs\_consumed \]  

(2)

where a Resource Unit is the metric for the compute capacity used and \( Total\_RUs\_consumed \) is the total of all the RUs consumed by all the MEs. RU is calculated as follows by the Ohio Super Computer:
\[ RU = 0.1 \cdot Tt \]  

Here \( Tt \) is the total CPU time in hours for all CPUs

\[ Tt = wall\_time \cdot \text{max}_\text{ppn} \cdot \text{nodes\_requested} \]  

where \( wall\_time \) is the time taken for a simulation job, \( \text{max}_\text{ppn} \) is the maximum processors per node and \( \text{nodes\_requested} \) are the number of nodes requested by the user to be used in the simulation job. We take these calculations directly from OSC [30].

The service provider essentially invests a lot of capital into this development project and hence we now suggest ways for return of investment. The best approach for this is to recover costs over a period of time from the participating MEs. The following equation represents the cost incurred by a single ME as a fee for using the service:

\[ C_{\text{manf}} = \frac{C_{\text{acq}}}{M \cdot N} + \frac{C_{\text{fixed\_monthly}}}{M} + C_{\text{variable\_monthly}} \]  

where \( C_{\text{manf}} \) represents the price charged to the MEs monthly as a fee, \( C_{\text{acq}} \) is the cost of acquisition of the hardware and software, \( M \) represents the number of participating MEs using the environment, \( N \) is the period in months for return of investment of the cost incurred by the SPs, \( C_{\text{fixed\_monthly}} \) are the recurring monthly charges incurred by the SPs towards maintenance of the environment and \( C_{\text{variable\_monthly}} \) is the variable cost like Supercomputer usage and VM usage by this ME. We suggest the value of \( N \) to be between 24-36 months. At the end of the recovery period, we reach a break-even point and the charges henceforth are profits towards the first term in the equation.
4.2.2 Manufacturing Enterprise Cost & Pricing model

The MEs utilize this rich environment to speed up their App development cycle. They gain benefits in terms of access to compute resources on-demand, tools to create new Apps and an array of services to help them in their endeavor to develop apps. Their cost accounting is simpler compared to SPs and is done monthly. The MEs total costs \( C_{\text{manf, total}} \) would include two fundamental charges: Fixed monthly \( C_{\text{manf, fixed}} \) and variable monthly costs \( C_{\text{manf, variable}} \).

\[
C_{\text{manf, total}} = C_{\text{manf, fixed}} + C_{\text{manf, variable}} \tag{1}
\]

where \( C_{\text{manf, fixed}} \) is the fixed monthly cost determined for this ME by the SP for using the infrastructure. This is determined by the \( C_{\text{acq}} \) & \( C_{\text{fixed, monthly}} \) costs and can be represented as follows:

\[
C_{\text{manf, fixed}} = \frac{C_{\text{acq}}}{M,N} + \frac{C_{\text{fixed, monthly}}}{M} \tag{2}
\]

where the terms on the RHS are the first two terms in equation (6) which provide the fixed monthly cost for using this environment as charged the SPs. The variable cost for manufacturers is the charges for using the Supercomputer and the VMs

\[
C_{\text{manf, variable}} = C_{\text{RU}} \cdot RUs_{\text{consumed}} + \sum_{i=1}^{n} B_i \cdot t \tag{3}
\]

where \( C_{\text{RU}} \) is the cost per RU consumed, \( RUs_{\text{consumed}} \) is the number of RUs used for various computations by this ME in the current month, \( B_i \) is the Base rate charged for the \( i \)th VM used and \( t \) is the number of hours used. This equation accounts the compute resources and VMs used by a ME.

We now look at pricing models for MEs. Due to fixed nature of the App configurations, we can obtain the cost of a single App run developed by the ME as follows:

\[
C_{\text{app, usage}} = C_{\text{RU}} \cdot RUs_{\text{consumed}} \tag{4}
\]
For a single run of the App, the above equation gives the cost incurred. If the ME decides to charge their customers per run, then we can calculate a price $P_{app\_usage}$ per run of an App such that for N app runs they hit the break-even point with their costs:

$$P_{app\_usage} \cdot N = C_{manf\_fixed} + C_{manf\_variable}$$  \hspace{1cm} (5)

where $C_{manf\_variable}$ accounts for the $C_{app\_usage}$ for N runs beyond which the App is profitable and any other VM usage charges applicable and $C_{manf\_fixed}$ is the fixed monthly cost as calculated in equation (8).

Based on the number of customers and period of recovery, the MEs can charge an appropriate price for their Apps from the customers.

An alternative approach would be to charge the customers according to their individual usage for an App i.e. the cost for the customer would depend on the amount of resources i.e. RUs consumed. In this case, the ME can charge the customer a price $P_{app\_usage}$ for the Resource Units consumed such that

$$P_{RU} \cdot RUs\_consumed = C_{manf\_fixed} + C_{manf\_variable}$$  \hspace{1cm} (6)

Beyond these number of RUs based on $P_{RU}$, the MEs obtain profits.
5. APP RUNTIME WEB FRAMEWORK

Based on the high level design presented in Chapter 3, we implemented a web framework to implement the App Runtime. We developed the core applications with an additional Status App that helps in chaining of these Apps based on the configuration. We also developed functions for the Runtime that help in creation, monitoring and maintenance of these distributed Manufacturing Applications. From the user's perspective, this framework appears to be a normal website with no apparent distributed architecture. The user is presented with an easy to use Web UI and interactive components that enable complex workflow orchestrations with few simple drags and clicks. We have developed unique application dashboards for Manufacturers and their Customers. This development was done using PHP, HTML, MySQL, web services and shell scripting. Authentication is performed through an Active Directory and login information of each entity utilizing the environment are added by an Admin. The App Runtime was deployed on an existing testbed which was configured to run the ‘WheelSim’ product. The test bed is shown in figure 5.1. Using this testbed now with the App Runtime, Apps like ‘WheelSim’ can be created in a few clicks and we further improve resource utilization by deploying multiple applications on this testbed. We now demonstrate the various functions of the App Runtime.
5.1 App creation

Using our web framework, the Manufacturing Enterprises (MEs) after authentication, are displayed a dashboard which they can use to easily get started on the App creation process. The workflow is illustrated in Figure 6.1. To create a new Application, the ME can simply click the 'Create' button to initiate the App creation workflow. The ME is then presented with a web page in which the ME can drag and drop core applications discussed in the previous sections into a workflow, the status application is added by default to any application as it is essential in monitoring the progress of the execution of the App. There is also a provision to specify custom geometry for simulation. MEs can upload a template of a geometry file in .STL format and the appropriate Mesh is selected based on the template uploaded. Lastly the MEs are required to provide a name,
description and cost for their App. The App is then created by clicking ‘Create Application button’.

![Figure 5.2 A workflow for App creation using the web framework](image)

5.2 App Publishing

The MEs can publish their Apps after their creation. This is done through the ‘App info’ page. The web framework provides a publishing mechanism that will allow the Apps to be listed in the Marketplace. The ‘App info’ page provides a ‘Publish’ button that invokes the workflow. Once listed the Apps can be viewed in the Marketplace. The MEs can publish or hide their applications from the Marketplace using the same mechanism. The workflow is shown in Figure 6.3. The published Apps can then be viewed by the customer which can then be added to their accounts. Apps can also be deleted through the ‘App info’ page.

![Figure 5.3 A workflow of App publishing using web framework](image)
5.3 App Addition

The Customers can view Apps from the Marketplace and can add them to their accounts. The web framework provides this mechanism that will allow the Apps to be added to their account. This essentially creates an instance of the App for the specific user. Apps that are published can be added by the customers. This is done through the Marketplace. When customers are presented with published app, they are provided with an option to add an App through the ‘Add to my Apps’ button. This will trigger the App addition workflow. This is illustrated in the figure 6.3.

![Figure 5.4 A workflow for App addition using the workflow](image)

5.4 App execution

Both the MEs and customers can execute Apps. Depending on the App configuration, Status App either displays the results or executes another App. The Monitoring function is invoked each time the user runs an App. The App execution varies with the type of mesh and the configuration of the App. A workflow of App execution is shown for a wheel geometry and Post-pro configuration in Figure 6.2. The user first invokes the App through the web interface using the `Run App' sub menu button. The user is then presented with a form to specify the simulation parameters. These parameters generate a configuration file that is sent to OSC for further processing. The web service invokes scripts that execute the simulation as specified in the configuration file. Each core
application has unique script. The progress of the execution can be measured using the Status App through the sub menu button `App Status' provided. Post App execution, the Status App validates the execution and analyzes the App configuration and presents an interface to run the next App in the workflow. The user is provided with a button to either `Run next App' or `View Result' depending on the App configuration and the last App that was executed. The Status App also retrieves the result based on the App configuration. The `View Results' sub menu button displays the current and past results. Using this page, the user can view the results and analyze the simulation results. An expert in the field can judge and make informed decisions about the design based on the simulation results presented. In case of Figure 6.2 we see that the simulation results of the wheel are presented. The user can download the PVTP file or view the images depending on the requirement.

![Figure 5.5 A workflow for App execution using the web framework](image)
6. COST MODEL CASE STUDIES

In this section, we undertake a cost and performance model analysis. We use the cost model discussed in the Chapter 4 and analyze how the various entities incur costs based on some assumptions. We evaluate costs and undertake a break even study. We also undertake a performance study where we observe how the variation in compute resources effect the execution and the cost.

6.1 Cost Performance Model

We can use the proposed cost model for our architecture with some simulated costs to calculate how much it would cost for Manufacturing Enterprises to use this environment. This is an essential requirement and would shed some light on how they can price their Apps. Hence we now explore the various costs incurred to them using the cost model. We make the following assumptions:

1. The number of participating MEs are 10 and period of recovery is 36 months
2. Cost of acquisition is calculated for a single server with 24 logical cores, 48 GB RAM, HDD 1TB
3. Operation and acquisition costs are calculated according to Amazon’s TCO calculator[31]
4. We will use Vmware horizon environment as the virtualization software
5. We do not yet consider the licensing cost for VMs provisioned with Windows OS

Following table provides the breakdown of the acquisition and monthly fixed operational cost for the environment selected.
<table>
<thead>
<tr>
<th>Name</th>
<th>Specifications</th>
<th>Cost($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>24 logical cores, 48 GB RAM, 1 TB HDD</td>
<td>11,879</td>
</tr>
<tr>
<td>Software</td>
<td>VMware, Windows OS</td>
<td>10665</td>
</tr>
<tr>
<td>Monthly operational cost</td>
<td>Includes, rack, network, power consumption etc.</td>
<td>2954</td>
</tr>
</tbody>
</table>

Table 6.1 Cost of acquisition and operational cost

Following table provides the specifications for the variable costs where the RU value is directly taken from OSC and the VM usage cost is obtained from VMware chargeback center.

<table>
<thead>
<tr>
<th>Name</th>
<th>Metric</th>
<th>Cost($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supercomputer usage</td>
<td>RUs</td>
<td>0.4</td>
</tr>
<tr>
<td>VM usage</td>
<td>usage/VM/hour</td>
<td>0.003</td>
</tr>
</tbody>
</table>

Table 6.2 Metrics for the variable cost

For a single ME, the fixed monthly cost would be calculated according to equation (6) and is approximately $358. It is difficult to predict how the variable costs will add to the total cost of the ME but we will make an assumption for a typical usage of the system and then calculate the price accordingly. Let us assume that a Manufacturing Enterprise ME1 uses 100 VMs for 40 hours every week. Let us also assume that total RUs consumed by ME1 as a result of App execution by their customers is 100 RUs then according to the above table the variable cost for a month with this kind of usage would be $88 hence the total Cost would be $445. This price gives access to a Supercomputer for computation, a development environment which provides tools for agile development and an App marketplace for publishing their Apps. If MEs tried to do their own setup of this
infrastructure, we note that it would be several times more expensive as just owning the cluster would cost 10 times the monthly cost being charged currently.

Additionally, based on their requirements, the MEs can run their Apps to complete execution in minimum waiting time or minimum execution time. Since we run the simulation component of these Apps on a Supercomputer, we have the capacity to request required number of processors when submitting jobs and these jobs would be placed in a queue. Based on the order and availability of resources, these jobs get executed at the Supercomputer. The number of processors requested have a direct effect on the cost and the duration of the simulation. We performed a cost vs. performance analysis where we studied the effect of number of processors on these parameters. For each application, we measured the total cost by varying the number of processors. We also measure the total time taken to complete the tasks. With the exception of Post-pro which uses a single processor to execute, the Mesh & Solve and the Export app both use the specified number of processors to execute. Table 6.3 illustrates the tabulated data of the results for a simulation on a wheel geometry.

It can be observed that the Export and the Post-pro RUs increase at a lower rate and are only a fraction of the cost of the Mesh & Solve App. We can observe from the Figure 6.1 that as the number of processors increase, the time for the job reduces, we notice this trend until the number of processors reach a threshold (96), beyond this point, the time actually increases which is an anomaly. Upon further investigation, we found that beyond a threshold, the overhead from inter-process communication causes delays and hence the total time increases. From this experiment we found the optimal number of processors for minimum execution time and this value is 96.
<table>
<thead>
<tr>
<th>No. of Cores</th>
<th>Mesh &amp; Solve (RUs)</th>
<th>Export (RUs)</th>
<th>Post pro (RUs)</th>
<th>Total RUs</th>
<th>Total cost($)</th>
<th>Total Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>0.7753333333</td>
<td>0.098</td>
<td>0.080667</td>
<td>0.954</td>
<td>0.3816</td>
<td>1431</td>
</tr>
<tr>
<td>48</td>
<td>0.996</td>
<td>0.202667</td>
<td>0.150667</td>
<td>1.34933</td>
<td>0.539733</td>
<td>1012</td>
</tr>
<tr>
<td>72</td>
<td>1.31</td>
<td>0.284</td>
<td>0.204</td>
<td>1.798</td>
<td>0.7192</td>
<td>899</td>
</tr>
<tr>
<td>96</td>
<td>1.6906666667</td>
<td>0.386667</td>
<td>0.114667</td>
<td>2.192</td>
<td>0.8768</td>
<td>822</td>
</tr>
<tr>
<td>120</td>
<td>2.3033333333</td>
<td>0.466667</td>
<td>0.34</td>
<td>3.11</td>
<td>1.244</td>
<td>933</td>
</tr>
<tr>
<td>144</td>
<td>3.172</td>
<td>0.564</td>
<td>0.408</td>
<td>4.144</td>
<td>1.6576</td>
<td>1036</td>
</tr>
</tbody>
</table>

Table 6.3 Cost and the execution time for running a simulation with different number of processors

![Figure 6.1](image)

Figure 6.1 Plot indicating the time taken to complete a job for different processors

The cost vs. number of processors plot illustrated in the Figure 6.2, is fairly linear and increases as the number of processors increase. This is very logical as we are being
charged more for the additional compute resources being requested. With this knowledge, we now look at the pricing for the MEs and analyze how they can generate revenue to offset their costs.

![Graph](image)

**Figure 6.2** Plot indicating the time taken to complete a job for different processors

### 6.2 Cost Pricing Model

Based on the discussion about the various considerations of the fixed and variable cost, it becomes easier now to perform a pricing analysis for the MEs so that they can generate revenues by having their customers run the Apps in this environment. The MEs can charge their customers as discussed in the cost model analysis based on the App price per run or vary the price of the RUs charged. We explore both these pricing strategies and undertake a break-even analysis for both.
6.2.1 App run pricing

In this case, the MEs, fix a price for each run of the App. We consider the basic Mesh & Solve App that performs simulation on a wheel geometry as an example for this analysis.

The total monthly cost for the MEs can be computed from equation (7).

<table>
<thead>
<tr>
<th>No of App runs</th>
<th>Cost for # App runs</th>
<th>Total monthly cost</th>
<th>Price for # runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>6.762666667</td>
<td>412.7487778</td>
<td>100</td>
</tr>
<tr>
<td>20</td>
<td>13.52533333</td>
<td>419.5114444</td>
<td>200</td>
</tr>
<tr>
<td>30</td>
<td>20.288</td>
<td>426.2741111</td>
<td>300</td>
</tr>
<tr>
<td>40</td>
<td>27.05066667</td>
<td>433.0367778</td>
<td>400</td>
</tr>
<tr>
<td>50</td>
<td>33.81333333</td>
<td>439.7994444</td>
<td>500</td>
</tr>
<tr>
<td>60</td>
<td>40.576</td>
<td>446.5621111</td>
<td>600</td>
</tr>
<tr>
<td>70</td>
<td>47.33866667</td>
<td>453.3247778</td>
<td>700</td>
</tr>
<tr>
<td>80</td>
<td>54.10133333</td>
<td>460.0874444</td>
<td>800</td>
</tr>
<tr>
<td>90</td>
<td>60.864</td>
<td>466.8501111</td>
<td>900</td>
</tr>
<tr>
<td>80</td>
<td>54.10133333</td>
<td>460.0874444</td>
<td>800</td>
</tr>
<tr>
<td>90</td>
<td>60.864</td>
<td>466.8501111</td>
<td>900</td>
</tr>
<tr>
<td>100</td>
<td>67.62666667</td>
<td>473.6127778</td>
<td>1000</td>
</tr>
</tbody>
</table>

Table 6.4 Cost vs. Revenue based on number of App runs
From earlier calculation, we know that the fixed monthly cost to a ME is $358. We impose a restriction on the VM usage to convert it into a fixed cost and then account for different App runs to check the breakeven point. For this use case we assume that for a month, 100 VMs were used by the ME for 40 hours per week. At $0.003 hourly usage, this amounts to $48 for the whole month. We consider the case of minimum execution time and use the cost for the case of 96 processors. In this case, the cost for 1 App run is $0.67. Let us assume that the ME charges the user $10 for each App run. We now compute the number of App runs required to hit breakeven point between the total cost and the Revenue. Table 6.4 illustrates this data. It can be observed that as the number of runs increase, the variable cost increases along with it and the revenue

![Plot indicating the cost vs. revenue for variable number of App runs](image)

**Figure 6.3** Plot indicating the cost vs. revenue for variable number of App runs
increases accordingly and between 40 and 50 App runs we breakeven. We plot a breakeven curve between the cost and the revenue in the Figure 6.3.

We find that the breakeven point for this case is around 45 App runs. Beyond this, the MEs earn profits over their monthly costs. In a month, 45 App runs is easily achievable and the customer also is paying a very low price for each run hence this pricing is beneficial for both MEs and customers. This case just considered one App, if the ME is able to create popular and reusable Apps, the profit earned is higher.

### 6.2.2 RUs based pricing

In this case, the MEs charge a different price per RU consumed. We follow the same assumptions as in the previous case. The RUs consumed for a typical Mesh & Solve App is 1.69 RUs for 96 processors. The cost per RU to the ME is $0.4. The MEs can mark it up and charge a price of $2 per RU consumed by the customer. With these assumptions, we vary the number of RUs and compute the varying cost and revenues. Table 6.5 illustrates this data.

It can be observed that the variable cost increases with the number of RUs consumed. The revenue also increases and the breakpoint in this case is reached for around 253 RUs. We provide a view of the App runs as well to indicate at what number of App runs the break-even point will be reached. We once again plot the breakeven curve and it is shown in the Figure 6.4.
<table>
<thead>
<tr>
<th>Number of RUs</th>
<th>Cost for # RUs</th>
<th>Total monthly cost</th>
<th>Price for # RUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>33.8</td>
<td>13.5253333333</td>
<td>419.5114444</td>
<td>67.6</td>
</tr>
<tr>
<td>67.6</td>
<td>27.050666667</td>
<td>433.0367778</td>
<td>135.2</td>
</tr>
<tr>
<td>101.4</td>
<td>40.576</td>
<td>446.5621111</td>
<td>202.8</td>
</tr>
<tr>
<td>135.2</td>
<td>54.1013333333</td>
<td>460.0874444</td>
<td>270.4</td>
</tr>
<tr>
<td>169</td>
<td>67.626666667</td>
<td>473.6127778</td>
<td>338</td>
</tr>
<tr>
<td>202.8</td>
<td>81.152</td>
<td>487.1381111</td>
<td>405.6</td>
</tr>
<tr>
<td>236.6</td>
<td>94.6773333333</td>
<td>500.6634444</td>
<td>473.2</td>
</tr>
<tr>
<td>270.4</td>
<td>108.2026667</td>
<td>514.1887778</td>
<td>540.8</td>
</tr>
<tr>
<td>304.2</td>
<td>121.728</td>
<td>527.7141111</td>
<td>608.4</td>
</tr>
<tr>
<td>338</td>
<td>135.25333333</td>
<td>541.2394444</td>
<td>676</td>
</tr>
<tr>
<td>371.8</td>
<td>148.77866667</td>
<td>554.7647778</td>
<td>743.6</td>
</tr>
<tr>
<td>405.6</td>
<td>162.304</td>
<td>568.2901111</td>
<td>811.2</td>
</tr>
</tbody>
</table>

Table 6.5 Cost vs. Revenue based on number of RUs consumed

![Figure 6.4 Plot indicating the cost vs. revenue for variable number of RUs consumed](image)

Figure 6.4 Plot indicating the cost vs. revenue for variable number of RUs consumed
We found that this is equivalent to 150 App runs. Hence in this case, the cost incurred to the customer is less in comparison to the previous case. From the MEs point of view, in the previous case the breakeven was reached at 45 App runs but in this case the same is around 150. Both these pricing methodologies are almost equivalent but may change depending on the geometry being simulated and the integrated workflows present in the App. There are few other pricing approaches like Subscription, Freemium etc. which can be used by the MEs but we will not discuss them in our work.

6.2.3 Qualitative comparison between public and private clouds

In this section we briefly compare and contrast the differences between setting up resources in a private cloud versus a public cloud like Amazon Web Services. Firstly in case of a Private cloud, acquiring resources is mainly done through interaction with Personnel. A resource request is usually presented to the IT Department and their request is processed and resource allocation done manually. There are instances where certain API exist but predominantly it is done by the IT personnel. In case of public cloud however, resource reservation is almost always done through the use of APIs. Users who want to provision and reserve resources do so through APIs. These APIs tie in with the infrastructure of the Service Providers like AWS where their resources are provisioned.

Utilizing the private cloud, there is a limitation to how much the resources can scale and hence there is a queuing system that determines when access is granted to which user. This is typically implemented through an order based queuing system where user’s request is processed and provisioned and after certain amount of time, the resources are released, cleaned and then allocated to the next user that made a request to reserve the
resources. In case of public clouds, the scaling potential is hypothetically unlimited. Users can scale and reserve as many resources they want and can afford. This way, they get access to their resources quickly and experience no wait times. The resources are allocated on-demand.

In case of private cloud, the infrastructure and hardware is mostly static. The same hardware is used with minor improvements and upgrades and do not vary a lot. Replacement and complete upgrades if any are done at the end of the service life of the hardware and is typically expensive. This may ensure some level of homogeneity but cause the user to use old technologies and infrastructure where there are better alternatives which are more efficient. In case of public clouds, there are frequent upgrades to the hardware and software. This maintenance is done by the Service Provider free of cost. Better hardware are provided by the Service Provider frequently based on the needs of the customers and availability of infrastructure.

Hosting in Private clouds is considerably cheaper compared to the Public clouds. The benefits of the Public clouds come with a high price. Resources are charged on a pay-as-you-go basis and extensive usage of resources could result in expensive bills. Private clouds however are considerably cheaper compared to the Public clouds. Based on the cost model and resource costs in Amazon, we estimate that by using our Private cloud, the price per simulation is about $0.87 per hour but the same cost for resources with similar specifications is very expensive in case of AWS and is about $5.04.
7. FUTURE WORK

In this chapter, we discuss future work that will be conducted based on the work covered in this thesis. In the first section we discuss the extension of the Runtime to include diverse simulation tools and tasks. In next part we discuss how including multiple public clouds and HPC resources will be beneficial to the user.

7.1 Runtime extension with diverse simulation tasks

The Runtime developed consisted of core applications that performed lift and drag force simulation. This is done by few specific solvers in the TSFOAM which we have abstracted in the form of Apps. There are several different type of solvers which can be incorporated into the Runtime. This will enable creation of diverse set of Apps that cater to the different requirements of the Manufacturing Enterprises who create Apps for their customers. Diversity may lead to popularity of the Marketplace and attract other players in the Advanced Manufacturing domain who prefer using Apps rather than building their own from scratch.

7.2 Multiple public clouds and HPC resources

In this thesis, our focus was oriented towards using the GENI cloud with Ohio Supercomputer as the HPC component. In future additional public clouds can be added to the environment which allow the Manufacturing Enterprises and customers to choose these clouds based on their requirements. Additionally new HPC resources may also be added to the environment to improve the waiting time when the simulation jobs are in queue. Users could be given real time queue information and they can decide based on
their requirements of cost and time which location to choose to perform their simulation and modeling tasks.
8. CONCLUSION

In this thesis we presented the design and development of a SaaS web framework that enable Manufacturing Enterprises to develop, deploy new Apps in a Marketplace environment through ‘Agile Manufacturing’. This was mainly due to the current scenario in the App Marketplaces which lack development environment that support effective runtime capabilities that efficiently and cost-effectively integrates several Apps when building innovative products. Using our web framework in the form a dynamic App Runtime, we showed how Apps can be created, monitored, published and sold effectively. We observed better resource utilization with low management overhead. Due to the automation of several aspects like automated App creation, monitoring and accounting, there is a reduced management overhead by both the Service Providers and the Manufacturing Enterprises. Thus focus can be on building innovative products rather than worrying about infrastructure management and development from the scratch.

Utilizing the App Runtime also provides a dynamic and agile approach for Advanced Manufacturing App development. The Runtime allows simulation of any kind of geometry and thus can be used to create diverse set of Apps. Additionally since several existing modules are combined to create new Apps, the development time for new Apps is reduced. This also allows faster path to market for innovative products as App developers are liberated from complex tasks such as chaining and rewriting complex code as this is already provided as different modules.

The Cost Pricing model provided Manufacturers a keen insight into the costs and associated revenue that can be generated using this environment. The breakeven analysis
encourages the Manufacturing Enterprises to migrate their complex workflows to the cloud and take advantage of this unique environment to develop, deploy and sell their Apps. The Cost and performance comparison provided interesting insights into the tradeoffs that exist. The users can request resources to complete their simulation in minimum time or minimum cost based on their requirements and we also stumbled on a threshold for number of processors (96) required to complete the job in minimum execution time.
9. BIBLIOGRAPHY


[30] OSC: https://www.osc.edu/supercomputing/software/general#charging