Offloading Resource-Intensive Excel Spreadsheet Tasks to Aneka Cloud

COMP90019 - Distributed Computing Project

Software Development Project (25 credit points)

Prakash Chandra Bhandari

682949

Supervisor

Prof. Dr. Rajkumar Buyya

Cloud Computing and Distributed Systems Laboratory

Department of Computer and Information Systems

The University of Melbourne, Australia

ABSTRACT

Spreadsheet applications are widely used in most businesses and scientific research. Microsoft Excel is one of the most used spreadsheet application that provides a beautiful interface to store data, perform calculation and also provides ability to perform analysis on the stored data. Over time, as the data grows so does the necessity to perform compute and resource-intensive analysis tasks. Few applications have been proposed to address this need, however they are mostly complicated, expensive and requires expertise to scale the application on demand.

We propose to offload these tasks using an excel add-in and utilise pay-as-you-go and dynamic scalability nature of cloud computing to help accommodate such tasks as they grow. Also being able to run same task on multiple cloud vendors is a challenging task and we propose to use Aneka to overcome this shortcoming.

I certify that

- This thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person where due reference is not made in the text.
- Where necessary I have received clearance for this research from the University's Ethics Committee and have submitted all required data to the Department

The thesis is 3472 words in length (excluding text in images, table, bibliographies and appendices).

Acknowledgements

I would like to express my gratitude to my supervisor Prof. Dr. Rajkumar Buyya for providing project idea, support and supervision throughout the entire phase of the project. I learned a lot under him and most of what has been achieved goes to him. I would like to thank Raghavendra and Adel Nadjaran Tossi for their guidance and help whenever I was in need. Finally, I would like to thank my family and friends for inspiring me throughout the period and supporting me whenever I had doubts on completing the project in time.

Table of Contents

List of Tables

List of Figures

List of Acronyms

Acronym	Complete Expression
CLR	Common Language Runtime
EOF	End of File
IP	Internet Protocol
MPI	Message Passing Inerface
MS-MPI	Microsoft MPI
OOP	Object Oriented Programming
TCP	Transmission Control Protocol
UML	Unified Modeling Language
VBA	Visual Basic for Applications
VM	Virtual Machine
VSTO	Visual Studio Tools for Office

1. Introduction

Spreadsheet applications have been around for a long time. They have played a vital role in organizations by fulfilling their accounting needs and in large scientific research for storing and analysing data. Microsoft Excel has already been in the market for three decades [3]. As the data grows in these spreadsheet applications it is hard to perform various tasks in reasonable time in a single desktop computer. Some of these tasks performance can be improved by improving the calculation formula and also writing VBA macros [3]. However, for a very large dataset and complex calculations, it takes more time ranging from hours to days and even weeks in some cases. In such context, these intensive tasks can be offloaded to Cloud platforms where resource can be scaled as needed to accommodate faster execution.

Cloud Computing has been in the spotlight for quite some time now, for its model that provides computing services commoditised as utilities [1]. These services/ resources can be purchased cheaply in a pay-as-you-go model and dynamical scalability of resource on demand is possible. This computing model is perfect for our intensive problems as we can easily add more resources (RAM, CPU, VM) whenever needed to achieve good speedup while reducing or even turning off the resources when low use or not in use to keep the cost low.

Cloud Computing offers various services however, running distributed applications on heterogeneous cloud infrastructures from multiple vendors is a daunting task in itself. Aneka is a platform for deploying Clouds and distributed applications that can be scaled seamlessly and provide elastic run-time environment [6][1]. Aneka can be deployed in heterogeneous infrastructures and application developers can leverage various programming models to run distributed application without having to worry about the resources and Cloud Deployment models – Public, Private, Hybrid.

ExcelCloud software is a software package that offloads the resource intensive excel tasks to Cloud. The package consists of two parts, first is a Microsoft Excel add-in (ExcelCloudAddin) that allows defining jobs to be offloaded to Cloud and second is a windows form application (ExcelCloudServer) that receives jobs, executes them and returns the result to the client (add-in). ExcelCloudAddin is installed in client's computer while the ExcelCloudServer is installed in Cloud. It is important to remember that the tasks must be highly intensive and parallelizable to achieve good performance while a trivial task will incur decline in performance.

2. Background and Related Work

The problem that many researchers, organizations face today lies in the growing amount of data. Most of emerging technologies utilize the available Cloud platforms to cope with analysis of the growing data. These systems utilize parallel and/or distributed processing by dividing large task into smaller independent sub-tasks and executing them in multiple processors or nodes to achieve higher computational speedup.

Microsoft excel provides beautiful interface to store data, ability to create graphs and perform 'what-if?' analysis of data and they are considered best medium to collect and analyse data. However, the main drawback with excel is it doesn't provide support for large scale analytic function or invoking external computational model to perform such tasks. As such, Excel becomes just a data store without the ability to perform the necessary intensive computation. Our aim here is to bridge the gap between the interactive, user friendly, graphical Excel spreadsheet to the cloud based computation systems.

2.1. Related Work

Excel Spreadsheets have been around for quite some time and the idea of outsourcing compute intensive analysis tasks to external resources isn't new. While some work has been done on cloud application like Excel Datascope [5], it mainly focuses on sharing Excel resources in Cloud and performing analysis tasks on the spreadsheets available on the Cloud storage. This application is primarily useful in academic research domain where resource sharing is main purpose and performing analysis are limited to certain available tasks. Our application contrasts to this by mainly focusing on the offloading the tasks from spreadsheet and allowing user to define the job removing the limitation in performing fixed set of tasks only.

ExcelGrid [4] is related work to our application, which outsources tasks from excel to grid infrastructures. It utilises similar approach to send jobs to grid architecture where each job composed of smaller independent tasks are distributed among tightly coupled grid system. This system achieves good speedup on increasing number of nodes in execution. While this system is very good on doing the job, it requires expertise and is rather more expensive to provision resources on demand. Our application aims to address this issue by utilising Aneka Clouds which can be configured easily without any domain expertise and also cost can be greatly reduced by using the pay-as-you-go model as such applications do not need to run all the time.

3. Architecture

ExcelCloud application is designed to enable current excel software to export their tasks and to utilize power of cloud computing to execute these tasks parallel on multiple nodes to achieve higher performance. As such, ExcelCloud acts as a middleware between Excel and Cloud platform.

Architecture as shown in Figure 1. consists of two major components; Excel add-in and an ExcelCloud Server. This architecture is proposed keeping in mind that as add-in development at client end is separated from server, it gives greater flexibility to develop/upgrade client and server component separately.

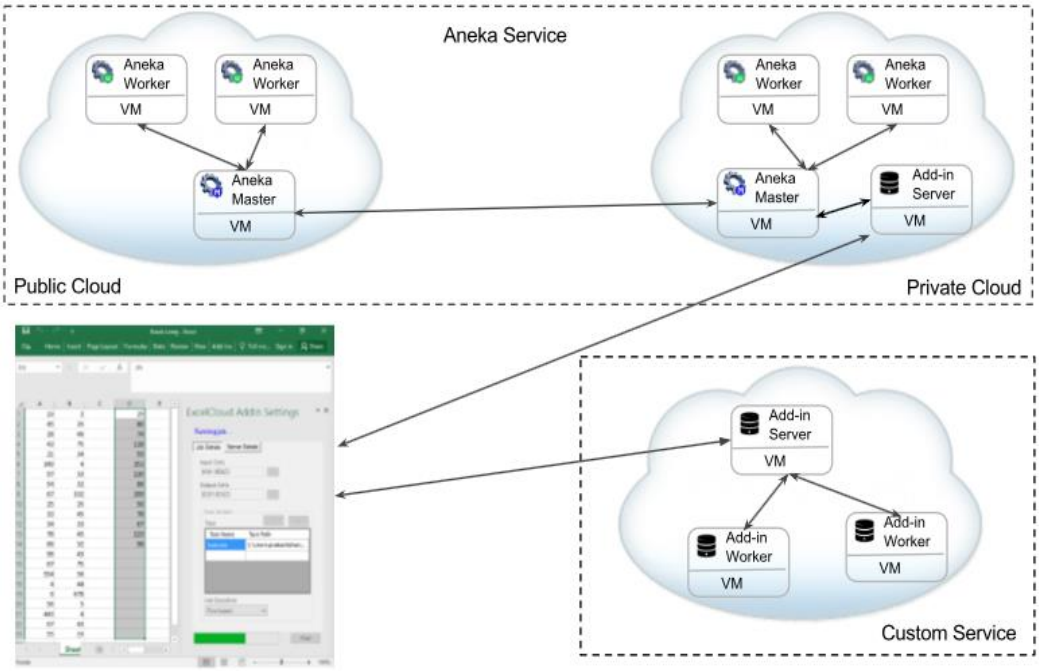

Figure 1: ExcelCloud Architecture

3.1. ExcelCloud add-in

Excel add-in consists of the user interface where user can configure jobs. As shown in Figure 2, the add-in interface is a windows form user control that allows user to define job by selecting input cells, output cells, executable task. It also provides options to enter ExcelCloud server and Aneka server details. The add-in also provides notification on job status as well as a progress on the job execution. Besides the interface excel add-in also provides socket communication to the ExcelCloud server.

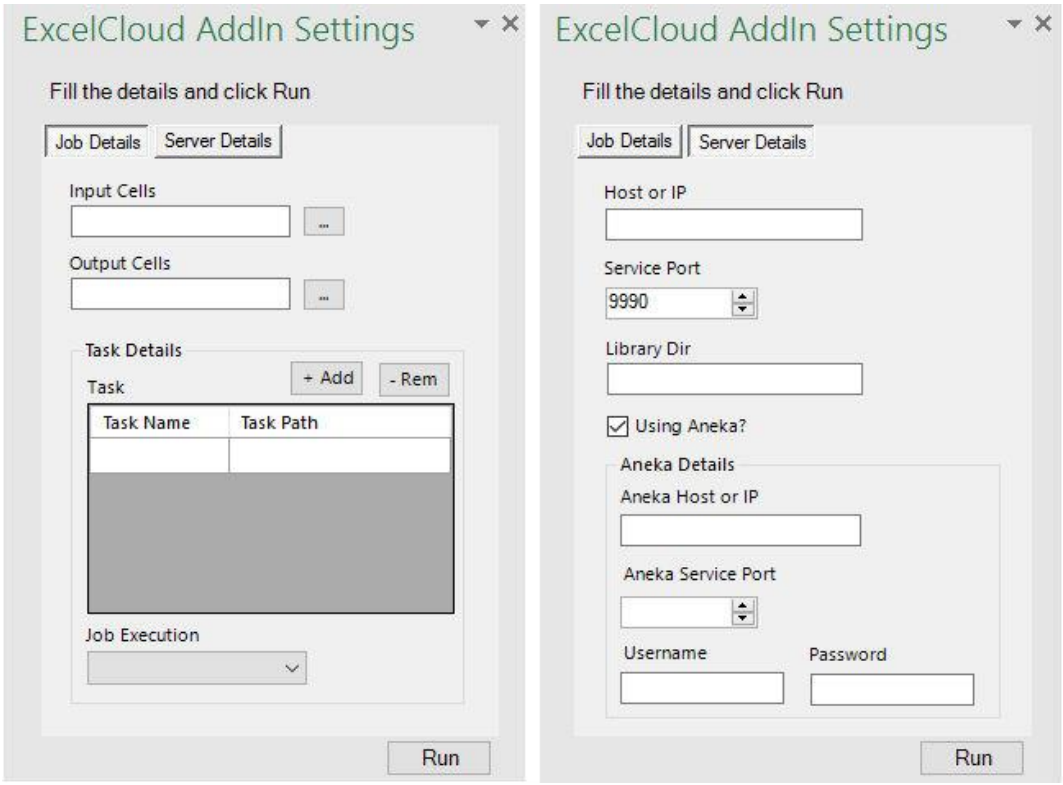

Figure 2: ExcelCloudAddin GUI

3.2. ExcelCloud Server

ExcelCloud server is the server deployment of ExcelCloud which uses either Aneka Service or custom service where the execution takes place. ExcelCloud server provides the gateway to the Cloud platform. ExcelCloud server consists of a windows form where you can set the host and port number that the server will listen to. This makes it easier to use ports that are enabled in firewall for transmission of jobs from Excel add-in. The form will also display which port the server is listening to with the status of server.

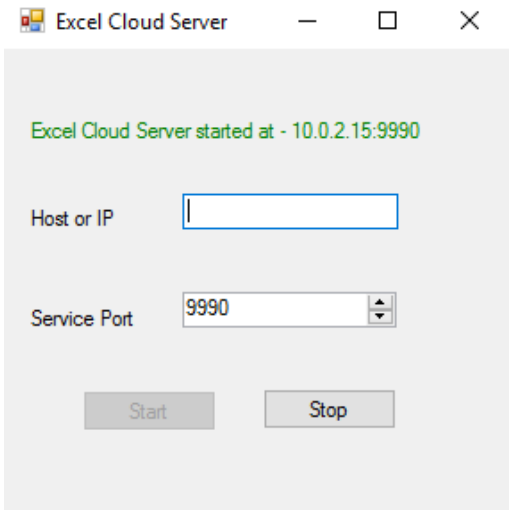

Figure 3: Addin Server GUI

Running the application from GUI will invoke the add-in to configure job request which is then transmitted to server using TCP connection. Job request is merely a json string containing all the data to execute job in Cloud. The inputs parameters are the values in Spreadsheet cells. User can choose any executable file that is available in server's library directory to execute. Once the execution type is chosen and Job is run, add-in will configure the job request and try to connect to the ExcelCloud server. Once connected, request is sent with all the information required to run job in the server.

Job in our application refers to a set of tasks each of which performs some computation over some inputs and produces output. At ExcelCloud server the information received is used to configure job that includes defining the task being executed, assigning input parameters for task and running each of the tasks as an individual work unit. The tasks are then either submitted to Aneka Service using Aneka's Task Model or executed using custom service. Once the execution of each independent task is completed, the output is processed and sent back to ExcelCloudAddin. This allows each work unit to work on independent task asynchronously and send output once the task is completed without having to wait for all the tasks to be completed. Once the output is received by add-in the progress is updated and notification is provided.

4. Design and Implementation

Based on the proposed architecture, the system is developed using object oriented C# programming language to run on Microsoft .NET framework. Add-in is developed using VSTO which is a set of development tools available to provide various templates to develop add-ins for office applications. The add-in utilises ribbon available in Excel to show or hide the Excel Cloud GUI. The ExcelCloud server is also developed in same programming language however, it runs in an earlier version of .NET which is a requirement for working of Aneka.

4.1. System Overview

Figure 4. shows the user interaction diagram between the add-in and the server component of the software package. The system starts by starting the ExcelCloud server instance at the Cloud platform. The server package provides with a GUI to fill the host and port details where the server will be listening for job requests. The server application displays a notification message suggesting the status of the of server i.e. started, stopped, could not start. After the server has been started, server asynchronously keeps listening for job requests. Figure 2. doesn't incorporate the steps mentioned so far and shows the start from the server already listening for job request and then user initiating the process.

At ExcelCloud Addin once the add-in has been installed it is automatically loaded to excel workbook. When all the job and server details are filled in and Run is clicked the form is first validated. After the form passes the validation, a job request is configured and appropriate notification is displayed in add-in GUI. After configuring job request a json string that contains all the job details is submitted to server using a Submit Job Request method. The add-in then displays a progress bar, disables the GUI form and shows Running Job notification and then starts listening for any output send by server. When a response is received, it is processed and updated in the spreadsheet by the add-in. Throughout this process the add-in is notified with a progress status in a progress bar. After the job is completed a notification stating job completion is displayed, progress bar is removed and the form is re-enabled.

When a job request is received at server, job is configured from the request, after which smaller independent tasks are prepared from it. Each task description will have and executable file and inputs passed in as arguments. Once these tasks are created they are now submitted to either the Aneka service or the custom service which will send the output to add-in asynchronously.

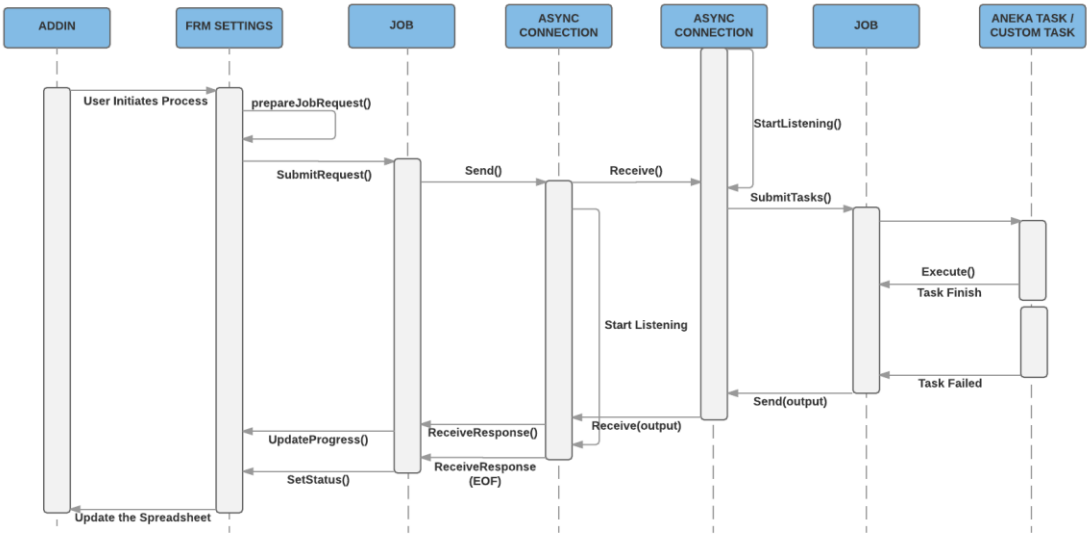

Figure 4: ExcelCloud Interaction Diagram

4.1.1. Software Requirements

For both server and add-in to work, it is necessary to have some required packages installed. Table 1. and 2. lists the requirements for running the ExcelCloud Server and AddIn application. In Cloud platform, it is also important to open required ports in firewall (both in cloud console and Windows machine) for communication between server and add-in. The software pair communicate using TCP packets so it is important to allow TCP packets through the port which the server will be listening to. Beside this if using Aneka, it is important to open ports for Aneka daemon and Aneka TCP port.

Requirement	Detail
64 bit Microsoft OS	The server works on 64 bit Microsoft OS only.
.NET v3.5	Server component is developed in .NET v3.5 as it works with Aneka v3.1.
Aneka v3.1	Aneka must be installed if you are using Aneka service

Table 1: ExcelCloudServer Requirements

Requirement	Detail
32 or 64 bit Microsoft OS	The add-in works with both 64 bit Microsoft OS only.
.NET v4.5.2	Add-in is developed in latest version of .NET (during the time development).

Excel 2010 or newer	Minimum requirement for addin is Excel 2010

Table 2: ExcelCloudAddin Requirements

4.2. Code Implementation

As mentioned earlier the software is developed in C# programming language using OOP paradigm. .NET framework is used as it provides various functionality that can be easily implemented in the application. Both add-in and server utilize the .NET framework to create GUI forms and for socket communication. In add-in .NET framework allows .NET Framework CLR to expose add-in functionality to programming language like C# which can then further be used to read job description from spreadsheet and communicate to server. This section is further divided into two parts to view both the add-in and server implementation at class level.

4.2.1. ExcelCloudAddin Implementation

ExcelCloudAddin consists of five classes that perform the work of displaying a GUI form, defining job, exposing spreadsheet values to the form, configuring job request and sending and receiving job request and response. The Addin initialises by loading the ThisAddin class which is a default class file for excel add-in template. This class allows defining the user interface and displaying the GUI in spreadsheet. Simultaneously, a Ribbon is also loaded through the ManageTaskPaneRibbon class file. This will add a toggle button to Show or Hide the ExcelCloud GUI form in the Add-ins menu of excel. Below is detailed description on three other classes:

4.2.1.1. *FrmSettings*

FrmSettings is inherited from Windows Form User Control, and it contains all the form elements required to setup the job. This class allows interaction with the spreadsheet to choose the input cells and output cells. It also contains OpenFileDialog control to choose the executable file to run on the Cloud. It also contains element to define server details. Once Run is clicked FrmSettings validates the form and invokes Job class and assigns specific attributes based on the form inputs. FrmSettings also manages all the Status messages and exposes the progressbar that can be updated based on job status. Table 3. shows the list of status that the job execution could be in. Based on the status of the Job, FrmSettings also toggles the progress bar and displays progress as it receives output.

Status code	**Status Detail**	**Notification Message**

0	Validation Failed	Please complete all fields before submitting job.
1	Preparing Job	Preparing job…
2	Job Sent to run	Running job…
3	All Job Completed	Job completed successfully.
4	Job Failed	Error encountered - Check server log for more information.
5	Couldn't Connect	Error encountered - Could not connect to server.

Table 3: FrmSettings status list

4.2.1.2. *Job*

This class contains the attributes to describe a job. Job class also stores the number of tasks that can be created out of the Job and number of arguments that each task accepts. This helps in preparing and submitting tasks at server. SubmitRequest() method at Job creates a connection to server and sends all the job details as a Json string and an EOF message to server. It then asynchronously waits for response from server and updates the progress bar at FrmSettings. Once all the jobs are completed EOF message is received from the server which lets Job class to trigger job complete status (status code 3).

4.2.1.3. *AsyncConnection*

AsyncConnection provides an asynchronous TCP socket connection to the server. AsyncConnection is invoked by first calling StartClient() which tries to connect to server at certain host IP and port number. If couldn't connect, it will trigger statusUpdate (code 5) at FrmSettings. This class allows sending TCP packets as well as receiving. Finally, this class exposes a function to close the current connection once the Job is completed. AsyncConnection utilizes another class StateObject to store the state of the connection and to store the message received the server.

4.2.2. ExcelCloudServer Implementation

At Cloud platform, usually where the Aneka master node is running or Custom service is running ExcelCloud Server is started. ExcelCloud Server contains Windows form User Control GUI to manage the starting and stopping the server. This package includes six class files that are used for displaying the GUI form, managing connection

with ExcelCloud Add-in, preparing and submitting job and finally executing job either at Aneka or Custom Service. The package initiates with default Program class that loads a Server configuration GUI form in a thread. FrmServerConfig form checks if the supplied port can be opened at the host. Once the port is opened the Program call will the call AsyncConnection to start listening to the port. Any message received from the socket are then configured as a job similarly as in Addin. After this Job class will prepare the tasks based on Execution type selected by user and submit the task to Aneka Task or Custom Task. Below is the detailed description on Job, Aneka Task and Custom Task Class.

4.2.2.1. *Job*

Similar to Addin Job class, allows contains all the job description attributes, once a SubmitTasks() method is invoked, Job class prepares the tasks and the input for each task. Figure 5. shows the code for generating task and submitting it to Aneka Service. Each task will be iterated over all the task executable file selected by user. The tasks will also depend on Job Execution type i.e. if job execution type is 'Row based' the number of tasks will the number of rows that the user has selected in spreadsheet as input cells and number of arguments or input for task, will be the number of columns selected in the spreadsheet. In this way number of tasks will be number of executables multiplied by number of rows for 'Row based' Job Execution type and the number of arguments will be the number of columns. A task Identifier (taskId) is given to each task and submitted to either Aneka or Custom Service.

```
int taskId = 1;
foreach (KeyValuePair<string, string> taskFile in taskFiles)
{
    for (int i = 0; i < numTasks; i++)
    {
        args = String.Empty;
        for (int j = 0; j < numParams; j++)
        {
            args += (jobExecution.Equals("Row based")) ? inputDatas[(i * numParams) +
          j] + " " : inputDatas[(j * numTasks) + i] + " ";
        }

        Debug.WriteLine("Running task:" + taskFile.Key);
        CustomTask taskExecutor = new CustomTask(serverDetails["libraryDir"] + "/" +
          taskFile.Key, args);
        taskExecutor.taskID = taskId;
        taskExecutor.Execute();
        taskId++;
    }
}
```

Figure 5: Task Creation and Submission

4.2.2.2. *AnekaT*

We use task programming model of Aneka for this project as it allows configuring task and submitting task to work unit easily. Aneka provides with two classes to work in task programming model. First is AnekaTask, which represents the work unit in the Task Model. Second is the ITask interface, which is executed and needs to passed to the AnekaTask work unit. ITask class only exposes Execute method, which is the method executed when the task is run in any node in cloud. AnekaT implements ITask class and the Execute method in AnekaT runs by firstly searching the library directory for it and then executing it in command line with the provided arguments. The result obtained is then written to console output which is read by Job class and sent to the add in.

4.2.2.3. *CustomTask*

CustomTask class file, similar to AnekaT searches for executable in library directory and runs it in command line and write the output to console. Unlike AnekaT, each task instance will need to invoke the Execute method on CustomTask to execute the task. This output is read by Job class and sent to the add in. CustomTask currently runs in only one machine where the ExcelCloudServer is running however, in future it can be extended by using MPI (like MS-MPI) to run on multiple nodes.

5. Performance Evaluation

To evaluate the performance of the system on Cloud infrastructures, we developed an application that computes mathematical functions on an input parameter and waits for some time ranging from one to sixty seconds before signalling task completion. This simulates a real time task for our application to analyse it's performance. The test is conducted in Microsoft Azure platform and job is sent from a laptop at Melbourne Australia.

Aneka during performance test has shown great results by using multiple nodes in parallel to achieve good speedup [7]. In our test using Aneka server, we conducted tests several times with a job that contains 80 tasks in total. Varying number of nodes were used, with different computing power. Figure 6. shows the performance of Aneka service of varying number of worker nodes. The performance improvement in increasing the number of nodes can be seen from the graph. Table 7. shows the resources used in the test.

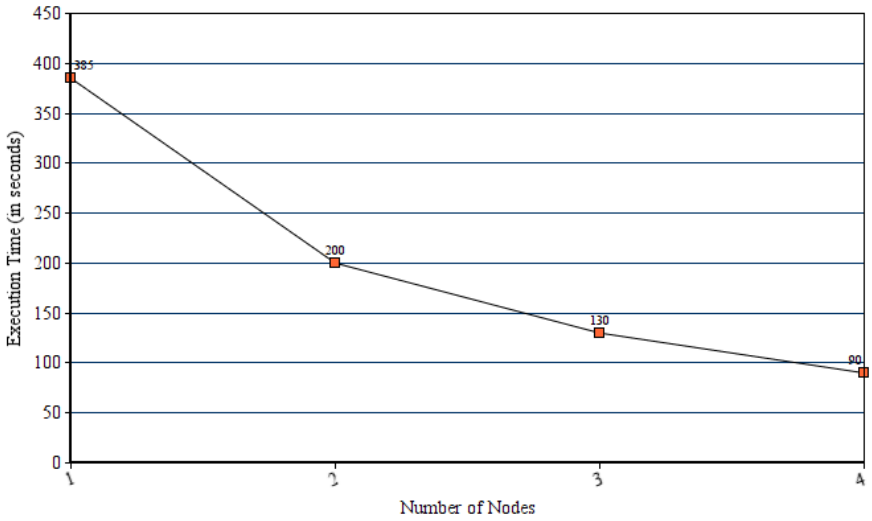

Figure 6. Performance of ExcelCloud in varying Aneka nodes

Name	OS	Aneka	CPU	RAM
azure-server1	Windows Server 2012 R2	Master	1.80 GHz	1 GB
azure-server2	Windows Server 2012 R2	Worker	2.40 GHz	3.50 GB
azure-server3	Windows Server 2012 R2	Worker	2.40 GHz	3.50 GB
azure-server4	Windows Server 2012 R2	Worker	2.40 GHz	1.50 GB
azure-server5	Windows Server 2012 R2	Worker	2.40 GHz	1.50 GB

Table 4: Azure Cloud resources

6. Conclusion and Future Work

Excel Spreadsheets have been vital in business, academics, research for a long time. While the increasing number of data is one of the challenge on performance of Excel Spreadsheet, the compute intensive analytic tasks are another. In this report we presented an application to execute such intensive tasks in clouds. We presented how the system can communicate the job description to the clouds where after execution the task can be successfully returned back to excel. From the performance evaluation of the system we can say that successful deployment of ExcelCloud can achieve good execution speedup by utilizing multiple nodes at cloud platform. Also, this proves to be a cheaper solution as the resource can be run whenever necessary without having to buy the whole infrastructure in the first place.

We also presented Aneka and its task programming model. Aneka provides .NET API for developing distributed application, monitoring, accounting, scheduling and provisioning Cloud. Using Aneka our application was able to achieve good performance without having to spend more time on load balancing and other factors.

As a note for future work, we need to add security features that includes encrypting data being communicated between the application pair. As the application currently doesn't support uploading client's executable file to server, and all the files must be uploaded to server's library directory before running task, this can be incorporated in future work which would make the application friendlier. Another point to be noted is the custom service currently runs the program in command line and returns the result, however, this can be improved by extending the custom service to implement MS-MPI and/or other service.

7. References

[1] Buyya, Rajkumar, Christian Vecchiola, and S Thamarai Selvi. *Mastering cloud computing: foundations and applications programming*. Newnes, 2013.

[2] "A Brief History of Spreadsheets - UMD Department of Computer Science." 2015. 2 Jun. 2016 https://www.cs.umd.edu/class/spring2002/cmsc434-0101/MUIseum/applications/spreadsheethistory1.html

[3] "Excel 2010 Performance: Improving Calculation Performance." 2015. 2 Jun. 2016 <https://msdn.microsoft.com/en-us/library/office/ff700515(v=office.14).aspx>

[4] Nadiminti, Krishna et al. "ExcelGrid: A .NET plug-in for outsourcing Excel spreadsheet workload to enterprise and global grids." *Proceedings of the 12th International Conference on Advanced Computing and Communication (ADCOM 2004* 15 Dec. 2004.

[5] Barga, Roger et al. "Excel DataScope for Data Scientists." *e-Science All Hands Meeting* Sep. 2010.

[6] Vecchiola, Christian, Xingchen Chu, and Rajkumar Buyya. "Aneka: a software platform for .NET-based cloud computing." *High Speed and Large Scale Scientific Computing* 18 (2009): 267-295.

[7] Alrokayan, Mohammed, and Rajkumar Buyya. "A Web portal for management of Aneka-based MultiCloud environments." Proceedings of the Eleventh Australasian Symposium on Parallel and Distributed Computing-Volume 140 29 Jan. 2013: 49-56.

8. Appendices

The plugin code is released under Apache license to increase its reuse and new innovations freely. The link for downloading the code is as listed below:

https://github.com/prakashbhandari/ExcelCloudAddin
https://github.com/prakashbhandari/ExcelCloudServer

README:
The application comprises of ExcelCloudAddin and ExcelCloudServer. The source code can be downloaded from the github URL listed above or from the usb stick provided.

INSTALL:
Install files are located in the publish directory of each packages in usb stick. Setup files are generated using ClickOnce deployment, so installation can be done simply by double clicking the setup file.

ExcelCloudServer is to be installed in the cloud. For testing purposes it can be installed on local machine as well.

ExcleCloudAddin is to be installed in the client computer from which job is supposed to be offloaded to the cloud. Once installed Addin Settings form can be displayed or hidden by clicking the Show/Hide on the Addins menu of Excel.

Note: Executable task file chosen in the Job details, must be uploaded to the server's library directory before running the job.

Video on Addin Usage
https://www.youtube.com/watch?v=J0pute_jqCw