International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

# COPY RIGHT

ELSEVIER
SSRN



IJIEMR Transactions, online available on 6th Feb 2020. Link
:http://www.ijiemr.org/downloads.php?vol=Volume-09&issue=ISSUE-02

Title: INTIGRATION OF ANEKA FOR APPLICATION OF MULTITHREADING0

Volume 09, Issue 01, Pages: 6-15.

Paper Authors

**V.SHOBHA RANI1, DEEPTHI KOTHAPETA**

Chaitanya Deemed to be University


USE THIS BARCODE TO ACCESS YOUR ONLINE PAPER

To Secure Your Paper As Per UGC Guidelines We Are Providing A Electronic Bar Code

International Journal for Innovative Engineering and Management Research
PEER REVIEWED OPEN ACCESS INTERNATIONAL JOURNAL
www.ijiemr.org

# INTIGRATION OF ANEKA FOR APPLICATION OF MULTITHREADING

***V.SHOBHA RANI[1], **DEEPTHI KOTHAPETA[2]**
* Assistant Professor, Dept. Of Computer Science, Chaitanya Deemed to be University
** Assistant Professor, Dept. Of Computer Science, Chaitanya Deemed to be University,
shobhareddy19@gmail.com, deepthivaishu18@gmail.com

**ABSTRACT:**
For higher processing and computing power, chip multiprocessors (CMPs) have become the new mainstream architecture. This shift to CMPs has created many challenges for fully utilizing the power of multiple execution cores. One of these challenges is managing contention for shared resources. Most of the recent research address contention for shared resources by single-threaded applications. However, as CMPs scale up to many cores, the trend of application design has shifted towards multi-threaded programming and new parallel models to fully utilize the underlying hardware. There are differences between how single- and multi-threaded applications contend for shared resources. Therefore, to develop approaches to reduce shared resource contention for emerging multi-threaded applications, it is crucial to understand how their performances are affected by contention for a particular shared resource. In this research, we propose and evaluate a general methodology for characterizing multi-threaded applications by determining the effect of shared-resource contention on performance. *Multithreading is becoming increasingly important, both as a program structuring mechanism and to support efficient parallel computations. This paper surveys research in analysis for multithreaded programs, focusing on ways to improve the efficiency of analyzing interactions between threads and to detect data races.*

## 1. INTRODUCTION

Aneka allows different kind of applications to be executed on the same Grid/Cloud infrastructure. In order to support such flexibility it provides different abstractions through which it is possible to implement distributed applications. These abstractions map to different execution models. Currently Aneka supports three different execution models:

Task Execution Model
Thread Execution Model
MapReduce Execution Model
Parameter Sweep Model

Each execution model is composed by three different elements: the WorkUnit, the Scheduler, the Executor, and the Manager. The WorkUnit defines the granularity of the model; in other words, it defines the smallest computational unit that is directly handled by the Aneka infrastructure. Within Aneka,

International Journal for Innovative Engineering and Management Research
PEER REVIEWED OPEN ACCESS INTERNATIONAL JOURNAL
www.ijiemr.org

a collection of related work units define an application. The Scheduler is responsible for organizing the execution of work units composing the applications, dispatching them to different nodes, getting back the results, and providing them to the end user. The Executor is responsible for actually executing one or more work units, while the Manager is the client component which interacts with the Aneka system to start an application and collect the results. A view of the system is given in the figure below.

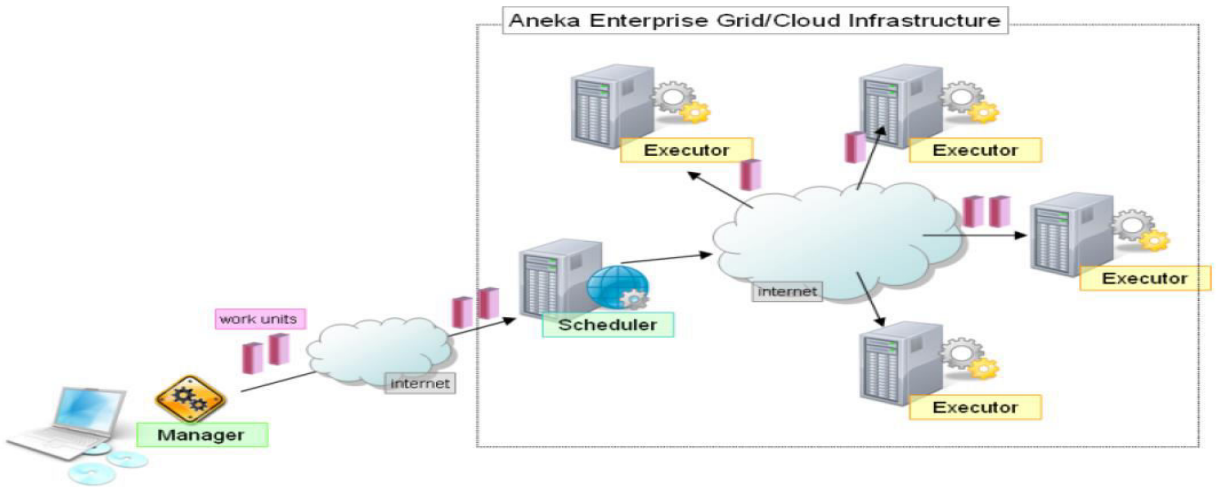


*Figure 1. System Components View.*

Hence, for the Thread Model there will be a specific WorkUnit called AnekaThread, a Thread Scheduler, a Thread Executor, and a Thread Manager. In order to develop an application for Aneka the user does not have to know all these components; Aneka handles a lot of the work by itself without the user's contribution. Only few things the users are required to know:

how to define AnekaThread instances specific to the application that is being defined;

how to create a AnekaApplication and starts the execution of threads;

how to control the AnekaApplication and collect the results.

Aneka is a .NET-based application development Platform-as–a-Service (PaaS), which offers a runtime environment and a set of APIs that enable developers to build customized applications by using multiple programming models such as *Task Programming*, *Thread Programming* and *MapReduce Programming*, which can leverage the compute resources on either public or private Clouds [1]. Moreover, Aneka provides a number of services that allow users to control, auto-scale, reserve, monitor and bill users for the resources used by their applications. One of key

International Journal for Innovative Engineering and Management Research
PEER REVIEWED OPEN ACCESS INTERNATIONAL JOURNAL
www.ijiemr.org

characteristics of Aneka PaaS is to support provisioning of resources on public Clouds such as Windows Azure, Amazon EC2, and GoGrid, while also harnessing private Cloud resources ranging from desktops and clusters, to virtual datacentres when needed to boost the performance of applications, as shown in **Figure 2**. Aneka has successfully been used in several industry segments and application scenarios to meet their rapidly growing computing demands.

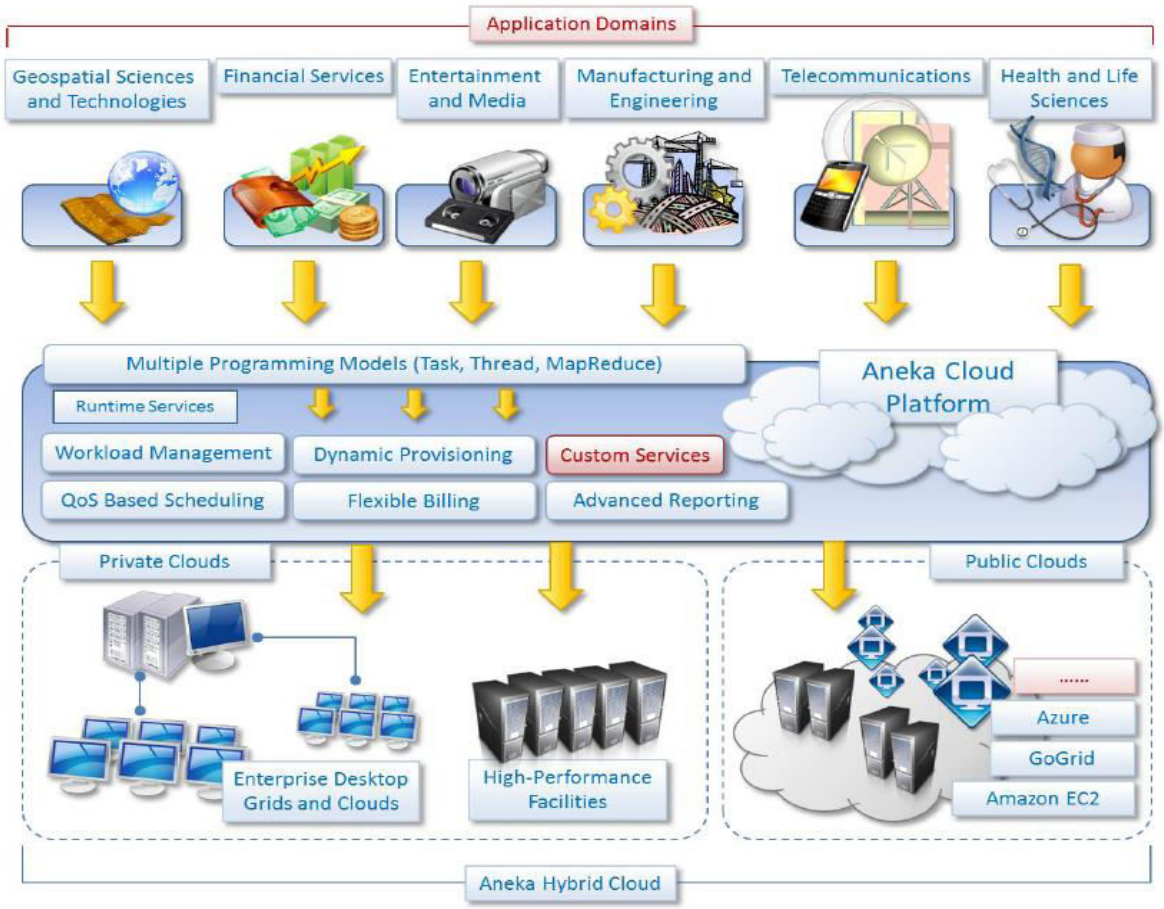


***Figure 2: Aneka Cloud Application Development Platform.***

Multithreading is a widely used structuring technique for modern software. Programmers use multiple threads of control for a variety of reasons: to build responsive servers that interact with multiple clients, to run computations in parallel on a multiprocessor for performance, and as a structuring mechanism for implementing rich user interfaces. In general, threads are useful whenever the software needs to manage a set of tasks with varying interaction latencies, exploit multiple physical resources, or execute largely independent tasks in response to multiple external events. Developing parallel applications requires an understanding of the problem and its logical structure. Understanding the dependencies and the

International Journal for Innovative Engineering and Management Research
PEER REVIEWED OPEN ACCESS INTERNATIONAL JOURNAL
www.ijiemr.org

correlation of tasks within an application is fundamental for designing the right program structure and to introduce parallelism where appropriate. Decomposition is a useful technique that helps to understand whether a problem is divided into components (or tasks) that can be executed concurrently.

The two main decomposition/partitioning techniques used area: domain and functional decompositions.

## 2. BACKGROUND

In this section, we present the architecture of Aneka PaaS, and then depict the overall view on Windows Azure Platform and Windows Azure Service Architecture. We also discuss the advantages brought by the integration, along with the limitations and challenges faced.

### 2.1 Overview of Aneka Cloud Application Development Platform

Figure 2 shows the basic architecture of Aneka. The system includes four key components, including Aneka Master, Aneka Worker, Aneka Management Console, and Aneka Client Libraries [1].

The Aneka Master and Aneka Worker are both Aneka Containers which represents the basic deployment unit of Aneka based Clouds. Aneka Containers host different kinds of services depending on their role. For instance, in addition to mandatory services, the Master runs the Scheduling, Accounting, Reporting, Reservation, Provisioning, and Storage services, while the Workers run execution services. For scalability reasons, some of these services can be hosted on separate Containers with different roles. For example, it is ideal to deploy a Storage Container for hosting the Storage service, which is responsible for managing the storage and transfer of files within the Aneka Cloud. The Master Container is responsible for managing the entire Aneka Cloud, coordinating the execution of applications by dispatching the collection of work units to the compute nodes, whilst the Worker Container is in charge of executing the work units, monitoring the execution, and collecting and forwarding the results.

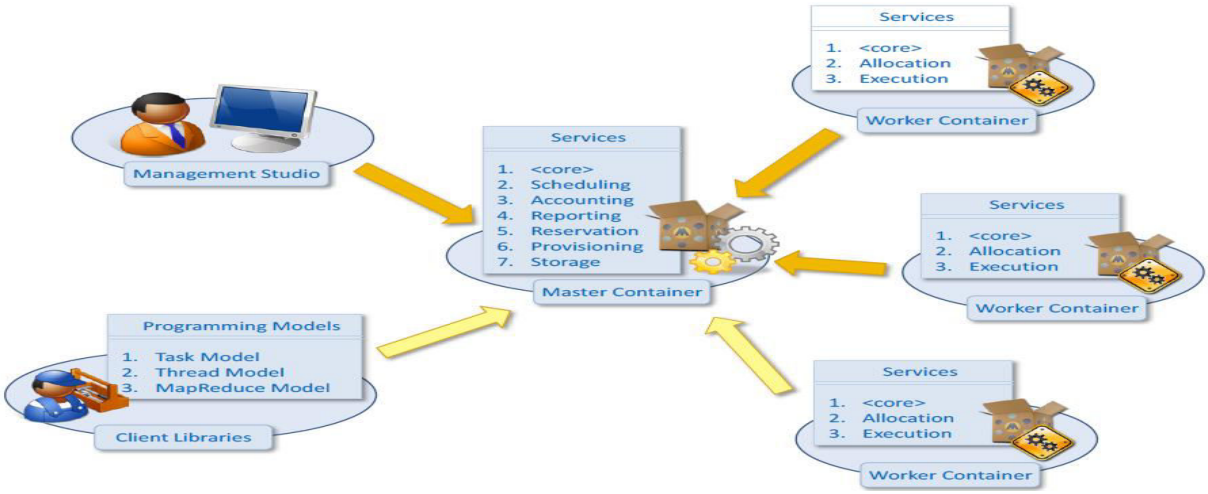


*Figure 3: Basic Architecture of Aneka.*

The Management Studio and client libraries help in managing the Aneka Cloud and developing applications that utilize resources on Aneka Cloud. The Management Studio is an administrative console that is used to configure Aneka Clouds; install, start or stop Containers; setup user accounts and permissions for accessing Cloud resources; and access monitoring and billing information. The Aneka client libraries, are Application Programming Interfaces (APIs) used to develop applications which can be executed on the Aneka Cloud. Three different kinds of Cloud programming models are available for the Aneka PaaS to cover different application scenarios:: *Task Programming*, *Thread Programming* and *MapReduce Programming* These models represent common abstractions in distributed and parallel computing and provide developers with familiar abstractions to design and implement applications.

**Map-Reduce on the virtualized environment:**

As a leading framework for big data analytics that is pioneered by Google and popularized by the open-source Hadoop, MapReduce is leveraged by a large number of enterprises to parallelize their data processing on distributed computing systems. It decomposes a job into a number of parallel map tasks, followed by reduce tasks that merge all intermediate results generated by map tasks to produce the final results. The MapReduce test app was written in Python in this study. MapReduce jobs are usually executed on clusters of commodity PCs, which require a large investment in hardware and management. GIL in Python: Parallel execution is forbidden in python because there is Global Interpreter Lock. The GIL ensures that only one thread runs in the interpreter at once. Simplifies many low-level details (memory management, callouts to C extensions, etc.). It's not a simple mutex lock. It's a binary semaphore constructed from a pthreads mutex and a condition variable. The GIL is an instance of this lock.

**Multithreading/Multiprocessing in Java:**

Concurrency refers to things happening in some unspecified order. Multitasking-executing multiple programs by interleaving instructions via time slicing - is good way to think about this sense of concurrency. Parallelism (or "true" parallelism) refers to things happening at literally the same time. This requires hardware support (coprocessors, multi-core processors, networked machines, etc.). All parallelism is concurrent, but not all concurrency is parallel. We ran multiple threads in Java but couldn't do so in python due to GIL lock.

**Map-Reduce on the virtualized environment:**

As a leading framework for big data analytics that is pioneered by Google and popularized by the open-source Hadoop, MapReduce is leveraged by a large number of enterprises to parallelize their data processing on distributed computing systems. It decomposes a job into a number of parallel map tasks, followed by reduce tasks that merge all intermediate results generated by map tasks to produce the final

results. The MapReduce test app was written in Python in this study. MapReduce jobs are usually executed on clusters of commodity PCs, which require a large investment in hardware and management.

**GIL in Python:**

Parallel execution is forbidden in python because there is Global Interpreter Lock. The GIL ensures that only one thread runs in the interpreter at once. Simplifies many low-level details (memory management, callouts to C extensions, etc.). It's not a simple mutex lock. It's a binary semaphore constructed from a pthreads mutex and a condition variable. The GIL is an instance of this lock.

## 3. ANEKA THREAD MODEL SAMPLES

The *examples* directory in the Aneka distribution contains some ready to run applications that show how is it possible to use the services provided by Aneka to build non-trivial applications. The examples concerning the Thread Execution Model are the following:

*Mandelbrot*

*ThreadDemo* (within the Tutorials folder)

The *ThreadDemo* has been fully explored in this tutorial. For what concerns the Mandelbrot example we will simply give some hints on how to explore the Visual Studio Projects related to the sample and see how the Thread Model has been used to distributed the application.

### 3.1 *Mandelbrot*

#### 3.1.1 Mandelbrot Set

The *Mandelbrot set* is a set of complex numbers for which the following iteration:

$$z = z_0$$

$$z_{n+1} = z_n^2 + z$$

does not diverge to infinity. This means that there exist a number N that can be considered the upper bound of the previous iteration. What makes interesting the Mandelbrot set is the fact that when applied to complex numbers it generates a bi-dimensional figure whose border does not simplify if magnified. In other word Mandelbrot set creates very interesting and fascinating fractals. These fractals can be easily generated by a computer program by using the following algorithm:

International Journal for Innovative Engineering and Management Research
PEER REVIEWED OPEN ACCESS INTERNATIONAL JOURNAL
www.ijiemr.org

```
1. let be W and H the size of the image we want to compute
2. let be z = 0+i0
3. then the range of complex values is [0+i0, W+iH]
4. for each complex number c = x+iy in [0+i0, W+iH]
    4.1 set s = true
    4.2 for i:0 to max_iterations do:
    4.2.1 compute z = z*z + c
        4.2.2 if |z| > 2 then set s = false and break
    4.3 if s is true c is in the Mandelbrot set (color: black)
    4.4 if s is false c is not in the Mandelbrot set (color: gradient)
5. end
```

The Escape Time algorithm is based on the assumption that no complex number with a modulus bigger than 2 can be part of the Mandelbrot set. This can be used as a quick condition to check whether the sequence of numbers generated for each c diverges or not. For those complex numbers that belong to the Mandelbrot set this condition will always hold and the iterations will continue indefinitely. The algorithm then imposes a maximum number of iterations after which the given number can be reasonably considered part of the Mandelbrot set. The algorithm presented does not guarantee a perfect drawing of the Mandelbrot set but the bigger it is the number of iterations the more precise is the resulting Mandelbrot set.

## 4. APPLICATION THREADING

This chapter covers general topics in application threading, particularly with respect to parallel performance. The topics occasionally refer to API-specific issues but much of the advice applies to any parallel programming method. The chapter begins with a discussion of data vs. functional decomposition. The opening topic gives advice on choosing the most appropriate threading method for either parallel model. This is followed by topics on granularity and load balance. These are critical issues in parallel programming because they directly affect the efficiency and scalability of a multithreaded application.

Tailoring thread behavior to a particular runtime environment is often overlooked in multithreaded programs. On a single-user system, for example, allowing idle threads to spin may be more efficient than putting them to sleep. On shared systems, however, forcing idle threads to yield the CPU may be more efficient. The issues involved in threading for high turnaround vs. high throughput are discussed. Many algorithms contain optimizations that benefit serial performance but inadvertently introduce dependencies that inhibit parallelism. It is often possible to remove such dependencies through simple transformations. Techniques for exposing parallelism by avoiding or removing artificial dependencies are discussed.

The next two topics describe how to choose an appropriate number of threads and how to

International Journal for Innovative Engineering and Management Research
PEER REVIEWED OPEN ACCESS INTERNATIONAL JOURNAL
www.ijiemr.org

minimize overhead due to thread creation. Creating too many threads hurt performance for many reasons, including increased system overhead, decreased granularity, increased lock contention, etc. Therefore, it is a good idea to control the number of threads through runtime heuristics and thread pools. Heuristics allow the programmer to create threads based on workload requirements that may not be known until runtime. Thread pools to limit the overhead of thread creation is described. The advice in this topic is primarily for applications threaded with Pthreads or the Win32 thread API. Thread pools are already used in the Intel OpenMP implementation.

The chapter closes with techniques for handling order-dependent output and loop optimizations designed to boost OpenMP performance.

## 5. METHODOLOGY

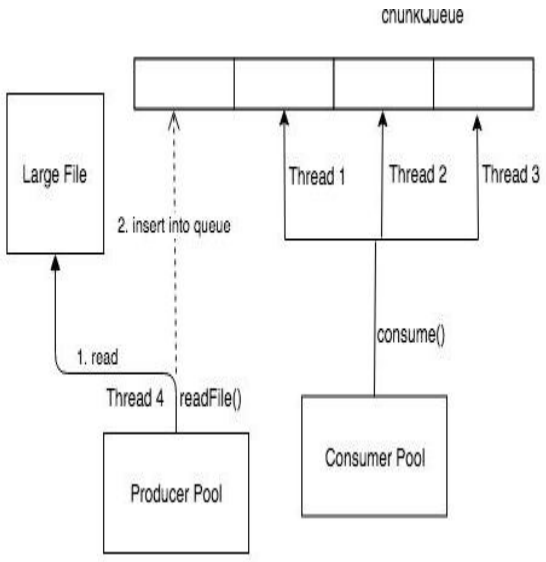


Figure 4. Producer and consumer with a blocking queue.

The model shown in the figure above uses a blocking queue. The producer reads a large file which is the dataset and divides the file into chunks. Each chunk are inserted to chunk queue. The consumer pool consisting of a number of worker threads consumes the chunk of data and process them. When the worker threads in the consumer pool are done with processing the main thread aggregates the result and writes to an output file.

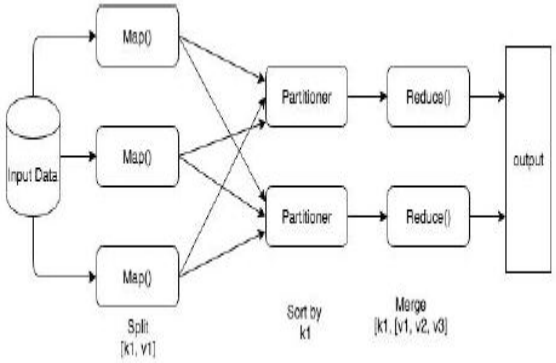


Figure 5. Mapreduce programming model.

In MapReduce Programming Model implementation, we read the data file and split the data in each line using mapper function. The output of mapper is passed to partitioner which aggregates the values associated with the key. The Reducer function then uses that output from the partitioner to merge and aggregate the overall result.

## 6. CONCLUSION

In this tutorial we have introduced the Thread Model for developing distributed applications based on remotely executable threads with Aneka. The Thread Model allows developers to quickly virtualize multi-threaded applications with Aneka. It

introduces the concept of *AnekaThread* that represents a thread that is executed on a remote computing node in the Aneka network. The *AnekaThread* class exposes a subset of the operations offered by the *System.Threading.Thread* class, this makes the transition from a local multi-threaded application to a distributed multi-threaded application straightforward. We have implemented two different programming models, multithreading and map reduce. Multithreading outperforms the MapReduce model in our case. MapReduce is basically a programming model with an associated implementation for processing and generating big data sets with a parallel , distributed algorithm on a cluster .

## REFERENCES

[1] Christian Vecchiola, Xingchen Chu, and Rajkumar Buyya, *Aneka: A Software Platform for .NET-based Cloud Computing*, High Speed and Large Scale Scientific Computing, 267-295pp, W. Gentzsch, L. Grandinetti, G. Joubert (Eds.), ISBN: 978 - 1-60750-073-5, IOS Press, Amsterdam, Netherlands, 2009.

[2] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic, *Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility*, Future Generation Computer Systems, 25(6):599-616, Elsevier Science, Amsterdam, The Netherlands, June 2009.

[3] David Chappell, *Introducing the Windows Azure Platform*. David Chappell & Associates, October 2010.

[4] David Chappell, *Introducing Windows Azure*. David Chappell & Associates, October 2010.

[5] Henry Li, *Introducing Windows Azure*, ISBN: 978-1-4302-2469-3, Apress, 2009.

[6] Sriram Krishnan, *Programming Windows Azure: Programming the Microsoft Cloud*, ISBN: 978-0-596-80197-7, O'REILLY, Sebastopol, CA, USA, May 2010.

[7] Dariusz Rafał Augustyn and Łukasz Warchał, *Cloud Service Solving N-Body Problem Based on Windows Azure Platform*, Proceedings of the 17th Conference on Computer Networks, Communications in Computer and Information ScienceUstron, Poland, June 15-19, 2010.

[8] Wei Lu, Jared Jackson, and Roger Barga, *AzureBlast: A Case Study of Developing Science Applications on the Cloud*, Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, Chicago, Illinois, USA, June, 2010.

AUTHOR 1:

NAME: V.SHOBHA RANI

QUALIFICATIONS: M.Sc(CS).,M.Tech.

DESIGNATION: Assistant Professor

DEPARTMENT: Computer Science

Chaitanya Deemed to be University

AUTHOR 2:

NAME: DEEPTHI KOTHAPETA

QUALIFICATIONS: M.Sc(CS).,M.Tech

DESIGNATION: Assistant Professor

DEPARTMENT: Computer Science

Chaitanya Deemed to be University