Computer Engineering and Intelligent Systems
ISSN 2222-1719 (Paper) ISSN 2222-2863 (Online)
Vol 3, No.12, 2012
www.iiste.org

Conceptual Design and Implementation of a Cloud Computing Platform Paradigm

Kola Ayanlowo[1], O. Shoewu[2], Segun O. Olatinwo[3], Tobi Samuel Fadiji[4], Segun Adeyanju[3]

1. Department of Computer Science, Moshood Abiola Polytechnic, Abeokuta, Nigeria.

2. Department of Electronic and Computer Engineering, Lagos State University, Epe Campus, Nigeria.

3. Department of Computer Science and Engineering, Ladoke Akintola University of Technology, Ogbomoso, Nigeria.

4. Department of Mechanical Engineering, Stellenbosch University, Stellenbosch, South Africa.

*E-mail: blessedkay@yahoo.com, segunlatinwo@gmail.com, engrshoewu@yahoo.com

Abstract
In recent times, organizations all over the world have stopped expanding infrastructures and building competencies in IT for enhanced efficiencies. Rather, they focus on their primary lines of businesses and "simply" connect to an existing IT cloud in the neighborhood or on the internet for their IT demands. Cloud computing is a new paradigm of large-scale distributed computing that centralizes the data and computation on the virtual "super computer" with unprecedented storage and computing capabilities. This paper focuses on the design of a conceptual framework and implementation of a cloud computing platform. This study attempts to design a platform on which users can plug-in anytime from anywhere and utilize enormous computing resources at a relatively low cost. Alongside the design, the mathematical model structures that support the design of the framework are explicitly described. The study is of paramount importance because the new framework provides opportunity to avoid network congestions that degrade performance among other shortcomings being experienced in some implementation cases.
Keywords: Cloud Computing, Framework, Platform, Paradigm

1. Introduction

As more facets of work and personal life move online and the Internet becomes a platform for virtual human society, a new paradigm of large-scale distributed computing has emerged. Web-based companies, such as Google and Amazon, have built web infrastructure to deal with the internet-scale data storage and computation. If we consider such infrastructure as a "virtual computer", it demonstrates a possibility of new computing model, i.e., centralize the data and computation on the "super computer" with unprecedented storage and computing capability, which can be viewed as a simplest form of cloud computing. Cloud computing is Internet-based system development in which large scalable computing resources are provided "as a service" over the Internet to users.

The concept of cloud computing incorporates web infrastructure, software as a service (SaaS), Web 2.0 and other emerging technologies, and has attracted more and more attention from industry and research community. For example, from the view of end-user, the cloud computing service moves the application software and operation system from desktops to the cloud side, which makes users be able to plug-in anytime from anywhere and utilize large scale storage and computing resources. On the other hand, the cloud computing service provider may focus on how to distribute and schedule the computer resources. Nevertheless, the storage and computing on massive data are the key technologies for a cloud computing infrastructure.

Cloud computing is defined to describe both a platform and a type of application. A cloud computing platform dynamically provides, configures, reconfigures and deprovides servers as needed. Servers in the cloud can be physical machines or virtual machines. Advanced clouds typically include other computing resources such as storage area networks (SANs), network equipment, firewall and other security devices. In this paper the design and the implementation of a conceptual framework for a cloud computing platform shall be described alongside with the mathematical model structure that supports the design, while taking into consideration at the design stage, 'storage and computing of massive data' ability by the cloud computing infrastructure. A cloud is a pool of virtualized computer resources that can:

- Host a variety of different workloads, including batch-style back-end jobs and interactive, user-interfacing applications.
- Allow workload to be deployed and scale-out quickly through the rapid provisioning of physical or virtual machines.
- Monitor resource use in real time to enable rebalancing of allocations when needed.

Support redundant, self-recovering, high-scalable programming models that allow workloads to recover from many unavoidable hardware/software failures. This study is of paramount significance because the new mechanism provides opportunity to avoiding network congestions that degrade performance. Cloud computing represents an exciting opportunity to bring on-demand applications to customers in an environment of reduced risk and enhanced reliability.

In this study, the emerging trend, construction and implementation of cloud computing platform is described. This study presents the attempt to implement a platform for a domain specific cloud computing service, with large scale web text mining as targeted application.

The rest of the paper is organized as follows. Section two of the paper consists of the literature review about cloud computing, security issues, development and design of a cloud computing platform, maintenance practice among others. Section three presents the mathematical model designs of a cloud computing platform including the analysis of mathematical algorithms that support the design and implementation of a cloud computing platform. Section four consists of the implementation testing and results of the cloud computing platform using matlab application based on the result of the simulation. Section five and six of the paper consists of the conclusion and recommendation based on the result of the design.

2. Cloud Computing Paradigm

2.1 History of Cloud Computing

The Cloud is a term with a long history in telephony, which has in the past decade, been adopted as a metaphor for internet based services, with a common depiction in network diagrams as a cloud outline. The underlying concept dates back to 1960 when John McCarthy opined that "computation may someday be organized as a public utility"; indeed it shares characteristics with service bureaus which date back to the 1960s. The term *cloud* had already come into commercial use in the early 1990s to refer to large ATM networks. By the turn of the 21st century, the term "cloud computing" had started to appear, although most of the focus at this time was on Software as a service.

Cloud Computing Paradigm: Cloud computing is a relatively new way of referring to the use of shared computing resources, and it is an alternative to having local servers handle applications. Cloud computing groups together large numbers of compute servers and other resources and typically offers their combined capacity on an on-demand, pay-per-cycle basis. The end users of a cloud computing network usually have no idea where the servers are physically located—they just spin up their application and start working. Cloud computing is fully enabled by virtualization technology (hypervisors) and virtual appliances.

A virtual appliance is an application that is bundled with all the components that it needs to run, along with a streamlined operating system. In a cloud computing environment, a virtual appliance can be instantly provisioned and decommissioned as needed, without complex configuration of the operating environment.

This flexibility is the key advantage to cloud computing, and what distinguishes it from other forms of grid or utility computing and software as a service (SaaS). The ability to launch new instances of an application with minimal labor and expense allows application providers to:

• Scale up and down rapidly, recover from a failure, bring up development or test instances, roll out new versions to the customer base, and efficiently load test an application.

Cloud computing allows us to locate computing resources anywhere in the world. No longer does the computer (whether it is a PC or supercomputer) have to be collocated with a user or institution. With high bandwidth optical networks it is now possible to collocate cloud computing resources with renewable energy sites in remote locations.

'Cloud computing' has been taken up to describe a computing paradigm in which software and services are accessed over a network. In contrast to traditional software distribution models, where applications are licensed per user and installed on end-user devices, users access applications on the internet when needed.

Cloud computing builds on earlier types of grid or on-demand computing, where applications and services are provided as a utility model, such as that used for electrical power. In such a model, users are charged for the actual usage of services. This model can offer advantages when compared to traditional licensing mechanisms and allows usage to be matched to actual demand.

One of the prime advantages in today's economy of using cloud computing services is that they allow organizations to turn capital expenses into operating expenses because they subscribe to a service, rather than having to make upfront investments in the technology used. Cloud computing has been made possible through advances in processors, virtualization technology, storage technology, fast, inexpensive servers and widespread, reliable broadband connectivity.

Technology vendors offering cloud computing services are building massive datacenters, comprising in some cases hundreds of thousands of servers and processors to create immensely powerful, scalable systems.

2.2 Layers in Cloud Computing

There are several recognized layers in cloud computing. The vendors in these layers have very different service offerings and operating models. Some vendors concentrate on building and maintaining a huge data center, while others concentrate on building a user friendly and feature-rich application. The layers, from bottom to top, are: infrastructure, storage, platform, application, services, and client.

2.2.1 Infrastructure

At the bottom is the infrastructure of the service, or the platform virtualization. You get the kind of server environment you want. This is the basic offering; customers still need to handle the server, all software installation, and maintenance by themselves. The cloud computing infrastructure does differ from traditional hosting services because of scalability, and pay-as-you-go pricing. A start-up company might be very interested in getting the scalability, and in not paying for the time they're not using the service. It is convenient, especially if you are trying to grow the traffic on your Web application but don't know how soon, or how well, you will succeed.

2.2.2 Storage

With the storage layer, you get a database or something similar, and pay per gigabyte per month. A storage layer is nothing new or special, except for the full stack of services. It is, of course, vital. There are several possibilities for storage. Some are traditional relational databases, and some are proprietary solutions such as Google's Bigtable or Amazon's SimpleDB.

2.2.3 Platform

The platform layer has solution stacks such as Ruby on Rails, LAMP, or Python Django. Now things start to get interesting. That fictitious start-up company doesn't have to deal with the installation of server software, or keep their versions updated, because that comes with the service. They can focus on developing and marketing their application.

2.2.4 Application

The application layer contains applications that are offered as services. The most famous examples are probably Salesforce.com and Google Docs, but there are hundreds if not thousands of (real) applications that can be purchased as services. Popular Web applications such as Facebook, and LinkedIn are cloud services. In these cases, the customer probably does not know if the application is run in a scalable data center, in an ordinary hosting service, or in the service providers basement.

2.2.5 Services

The services layer contains interoperable machine-to-machine operations over the network. The most prevalent example of this layer is Web services. Other examples include payments systems, such as PayPal, and mapping services, such as Google Maps and Yahoo Maps.

2.2.6 Clients

At the top of the stack is the client layer, which contains the users of the cloud systems. Clients are, for example, desktop users (thin client or thick client), and mobile users (Symbian, Android, iPhone).

2.4 Architectural Considerations

Designing an application to run as a virtual appliance in a cloud computing environment is very different than designing it for an on-premise or SaaS deployment. We discuss the following considerations. To be successful in the cloud, my application must be designed to scale easily, tolerate failures and include management tools. Organizations and individuals can benefit from mass computing and storage, provided by large companies with stable and strong cloud architectures. On the other hand, companies that desire to build massive, scalable environments, utilizing virtualization and cloud computing will increase their future margin of success greatly.

3. Mathematical Model and Design Methodology of a Conceptual Framework for a Cloud Computing Platform

3.1 Conceptual Model Design Stages/Guidelines

The best way to do this is to follow some basic application design guidelines:

- Start simple: Avoid complex design and performance enhancements or optimizations in favor of simplicity. It is a good idea to start with the simplest application and rely on the scalability of the cloud to provide enough servers to ensure good application performance. Some common design techniques to improve performance include caching, server affinity, multi-threading and tight sharing of data, but they all make it more difficult to distribute your application across many servers.
- Split application functions and couple loosely: Use separate systems for different pieces of application functionality and avoid synchronous connections between them. Again, as demand grows, you can scale each one independently instead of having to scale the entire application when you hit a bottleneck. The separation and reusability of functions inherent in SOA make it an ideal architecture for the cloud.
- Network communication: Design the application to use network-based interfaces and not inter-process communication or file-based communication paradigms. This allows you to effectively scale in the cloud because each piece of the application can be separated into distinct systems.
- Consider the cluster: Rather than scale a single system up to serve all users, consider splitting your system into multiple smaller clusters, each serving a fraction of the application load. This is often called "sharding" and many web services can be split up along one dimension, often users or account. Requests can then be directed to the appropriate cluster based on some request attribute or users can be redirected to a specific cluster at login.

To deploy a clustered system, determine the right collection of servers that yield efficient application performance, taking any needed functional redundancy into account; for example, 2 web, 4 application and 2 database servers. You can then scale the application by replicating the ideal cluster size and splitting the system load across the servers in the clusters.
Advantages of cloud computing when it comes to scalability are inexpensive testing, reduced risk, ability to segment the customer base, auto-scaling based on application load

3.2 Fail

Inevitably, an application will fail, no matter what its environment. When you design an on premise or SaaS application, you typically consider several "doomsday" scenarios. The same must be true for designing an application that runs in the cloud.

3.2.1 Build-in resiliency and fault tolerance: To tolerate failure, applications must operate as a part of a group, while not being too tightly coupled to their peers. Each piece of the application should be able to continue to execute despite the loss of other functions. Asynchronous interfaces are an ideal mechanism to help application components tolerate failures or momentary unavailability of other components.

3.2.2 Distribute the impact of failure: With a distributed cloud application, a failure in any one application cluster affects only a portion of the application and not the whole application. By spreading the

load across multiple clusters in the cloud, you can isolate the individual clusters against failure in another cluster.

3.2.3 Get back up quickly: Automate the launching of new application clusters in order to recover quickly. Application components must be able to come up in an automated fashion, configure themselves and join the application cluster. Cloud computing provides the ideal environment for this fast startup and recovery process.

3.2.4 Data considerations: When an application fails, data persistence and system state cannot be taken for granted. To ensure data preservation, put all data on persistent storage and make sure it is replicated and distributed. If system state is stored and then used in the recovery process, treat it like data so the system can be restarted from the point of failure.

3.2.5 Test your "doomsday" scenario: Cloud computing makes it easy to bring up an instance of your application to test various failure scenarios. Because of the flexible nature of cloud computing, it is possible to simulate many different failure scenarios at a very reasonable cost. Single instances of a system can be taken off-line to see how the rest of the application will respond. Likewise, multiple recovery scenarios can be planned and executed ahead of any real production failure.

3.2.6 Be aware of the real cost of failure: Of course the ideal situation is avoiding any application failure, but what is the cost to provide that assurance? A large internet company once said that they could tolerate failure as long as the impact was small enough as to not be noticeable to the overall customer base. This assertion came from an analysis of what it would cost to ensure seven nines of application uptime versus the impact of a failure on a portion of the customer base.

3.3 Manage

Deploying cloud applications as virtual appliances makes management significantly easier. The appliances should bring with them all of the software they need for their entire lifecycle in the cloud. More important, they should be built in a systematic way, akin to an assembly line production effort as opposed to a hand crafted approach. The reason for this systematic approach is the consistency of creating and re-creating images. We have shown how effectively scaling and failure recovery can be handled by rapid provisioning of new systems, but these benefits cannot be achieved if the images to be provisioned are not consistent and repeatable.

When building appliances, it is obvious that they should contain the operating system and any middleware components they need. Less obvious are the software packages that allow them to automatically configure themselves, monitor and report their state back to a management system, and update themselves in an automated fashion. Automating the appliance configuration and updates means that as the application grows in the cloud, the management overhead does not grow in proportion. In this way appliances can live inside the cloud for any length of time with minimal management overhead. When appliances are instantiated in the cloud, they should also plug into a monitoring and management system. This system will allow you to track application instances running in the cloud, migrate or shutdown instances as needed, and gather logs and other system information necessary for troubleshooting or auditing.

Without a management system to handle the virtual appliances, it is likely that the application will slowly sprawl across the cloud, wasting resources and money. Your management system also plays an important role in the testing and deployment process. We have already highlighted how the cloud can be used for everything from general testing to load testing to testing for specific failure scenarios. Including your testing in your management system allows you to bring up a test cluster, conduct any testing that is required and then migrate the tested application into production. The uniform resources that underlie the cloud mean that you can achieve a rapid release to production process, allowing you to deliver updated features and functions to your customers faster. Finally, by automating the creation and management of these appliances, you are tackling one of the most difficult and expensive problems in software today: variability. By producing a consistent appliance image and managing it effectively, you are removing variability from the release management and deployment process. Reducing the variability reduces the chances of mistakes – mistakes that can cost you money.

The advantages of designing your application for management in the cloud include:

- Reducing the cost and overhead of preparing the application for the cloud.
- Reducing the overhead of bringing up new instances of the application.
- Eliminating application sprawl
- Reducing the chance for mistakes as the application is scaled out, failed over, upgraded, etc.

3.4 Analytical Network Cloud Models

The idea of an ad-hoc cloud is to deploy cloud services over an organization's existing infrastructure, rather than using dedicated machines within data centres. Key to this approach is the ability to manage the use of computational and storage resources on individual machines, by the cloud infrastructure, to the extent that the cloud is sufficiently non-intrusive that individuals will permit its operation on their machines. A cloud may be viewed as comprising the union of a dynamically changing set of cloudlets, each of which provides some particular functionality. Each cloudlet runs on a potentially dynamically changing set of physical machines. A given machine may host parts of multiple cloudlets. The mappings between cloudlets and physical resources must be carefully managed in order to yield desirable high-level properties such as performance, dependability and efficient resource usage.

Cloud computing requires the management of distributed resources across a heterogeneous computing environment. These resources typically are, from the user viewpoint, "always on". While techniques exist for distributing the compute resources and giving a viewpoint of the user of "always on", this has the potential to be highly inefficient in terms of energy usage. Over the past few years there has been much activity in building "green" (energy efficient) equipment (computers, switches, storage), and energy efficient data centres.

3.5 Domain Model

There are two main aspects to the proposed domain model. One is the traffic matrix describing the dependencies of the load parameters of *inbound* and *outbound* links, which is described in the following section. The other aspect is the continuous adaption of this matrix based on measurements done on the live network. As mentioned above the purpose of the domain model is to describe the distribution of traffic flowing into a domain to other domains. We will investigate the scenario where we know two things: the loads on the outbound links at a given time and the origin of these loads by share of the inbound links.

3.5.1 Transit Matrix

As mentioned above, we need information about “traffic forking” — the distribution of traffic from an inbound link to the outbound links. Gathering this information from the ingress nodes is not easy in all cases because this might require knowledge of intra-domain topology and routing.

It is often easier to determine how much of the load on an outbound link comes from a specific inbound link. As a result we get the outbound loads Ot = (O1, t, . . . ,Om,t) at time t and the relative contributions ji of inbound link j to the load on outbound link i, for every pair (i, j). Thus, the load going from inbound link j to the outbound link is given by jiOi,t. Earlier, we stated the assumption that there is only negligible congestion, and therefore packet loss, in a single network domain. We can therefore state that “inbound load = outbound load”, or more formally.

3.5.2 Queuing Theory Approach

Traditionally, analytical network models have been based on the queuing theory originating from operations research and the like. Although creating such a model for a given system is often non-trivial, the results are both accurate and efficient. On the other hand, larger systems become very complicated to model.

In the simple case of one queue per inter-domain link we can use a classic M/M/1/K queue, that is, a queue with exponentially distributed inter-arrival time _ and service time s, a single “processing station” (the physical link) and a system capacity K. The arrival and service rates are given by:

$$\lambda = 1/E(\tau) \text{ and } \mu = 1/E(s) \quad (1)$$

This system is a birth and death process as shown in Figure 5. For a birth and death process of this kind the probability pi of the system to be in state i is given by:

$$p_i = \begin{cases} \frac{1-\lambda/\mu}{1-(\lambda/\mu)^{K+1}}, & i = 0 \\ (\lambda/\mu)^i p_0, & i > 0 \end{cases} \quad (2)$$

if $\lambda \neq \mu$, and

$$p_0 = p_1 = \ldots = p_K = \frac{1}{K+1}$$

if $\lambda \neq \mu$, . Because here we are only concerned with the changes to the traffic load caused by the queuing system, there is now a very simple way to simulate the dropping behaviour. Arriving packets will only be dropped if the queue is full, which is the case with probability pK. It is therefore sufficient to randomly drop an adequate fraction of the arriving packets, or in the case of a input load parameter It to write:

$$O_t = (1 - p_K) I_t \tag{3}$$

3.5.3 Assured Forwarding: As mentioned above Assured Forwarding defines three dropping precedence. The differences in behaviour towards this precedence are usually implemented by beginning to drop packets at different fill levels of the queue. This can again be modelled by a birth and death process, although a more complicated one. The arrival rate consist of three rates λ_l, λ_m and λ_h, for low, medium and high dropping precedence, respectively, with $\lambda = \lambda_l + \lambda_m + \lambda_h$. The system capacity is again K. Medium packets can only be queued if the system contains less than m packets, and high packets only if it contains less than h. The system state probabilities for i > 0 are

$$p_i = \begin{cases} \left(\frac{\lambda}{\mu}\right)^i p_0, & 0 < i \leq h \\ \left(\frac{\lambda - \lambda_h}{\mu}\right)^{i-h} p_h, & h < i \leq m \\ \left(\frac{\lambda_l}{\mu}\right)^{i-m} p_m, & m < i \leq K \end{cases} \tag{4}$$

State 0 consequently occurs with probability

$$\begin{aligned} p_0 = 1 - \Bigg[& \sum_{i=1}^{h} \left(\frac{\lambda}{\mu}\right)^i p_0 + \\ & \left(\frac{\lambda}{\mu}\right)^h \sum_{i=h+1}^{m} \left(\frac{\lambda - \lambda_h}{\mu}\right)^{i-h} p_0 + \\ & \left(\frac{\lambda}{\mu}\right)^h \left(\frac{\lambda - \lambda_h}{\mu}\right)^{m-h} \sum_{i=m+1}^{K} \left(\frac{\lambda_l}{\mu}\right)^{i-m} p_0 \Bigg] \end{aligned} \tag{5}$$

After some transformations and using the terms

$$A = \left(1 - \frac{\lambda}{\mu}\right), \; B = \left(1 - \frac{\lambda - \lambda_h}{\mu}\right), \; C = \left(1 - \frac{\lambda_l}{\mu}\right) \tag{6}$$

we can write

$$\begin{aligned} p_0 = ABC / \Bigg[& 1 - \left(\frac{\lambda}{\mu}\right)^{h+1} BC - \\ & \left(\frac{\lambda}{\mu}\right)^h \left(\frac{\lambda - \lambda_h}{\mu}\right)^{m-h+1} AC - \\ & \left(\frac{\lambda}{\mu}\right)^h \left(\frac{\lambda - \lambda_h}{\mu}\right)^{m-h} \left(\frac{\lambda_l}{\mu}\right)^{K-m+1} AB \Bigg] \end{aligned} \tag{7}$$

Analogous to the simple queue above the output loads are then calculated using

$$\begin{aligned} O_{h,t} &= I_{h,t} \cdot \sum_{i=0}^{h-1} p_i \\ O_{m,t} &= I_{m,t} \cdot \sum_{i=0}^{m-1} p_i \\ O_{l,t} &= I_{l,t} \cdot (1 - p_K) \end{aligned} \tag{8}$$

Note that the probabilities used also change for every t. Due to the rather heavy calculations involved the above model is not suited to very small sampling intervals.

$$\begin{pmatrix} p_0 & p_1 & \cdots & p_K \\ L & 1/\mu + L & \cdots & K/\mu + L \end{pmatrix} \tag{9}$$

3.5.4 Schedulers: Queuing systems with multiple queues and a single outgoing interface need one or more schedulers to decide which queue is allowed to send when the interface is done sending a packet. Some of the most frequently used schedulers are the Weighted Fair Queuing and Priority schedulers.

To model WFQ we can almost immediately use the Fair Queuing approach. Instead of nesting functions to determine the output loads of sub-models we can directly use the service rates $s_i\mu \ (i = 1, \ldots, n)$, and instead of the output bandwidth B we use a known service rate μ. Going through the algorithm yields the adjusted service rates for the queues. By recalculating the models with these rates we get the final outbound loads for every queue.

Priority schedulers can be modelled with a slightly modified version of the approach in Section 3.1.1. The system consists of n queues with arrival rates $\lambda_1, \ldots, \lambda_n$ and service rates $\nu_1, \ldots, \nu_n$. The service rates $\nu_1, \ldots, \nu_{n-1}$ are fixed and have the property

$$\sum_{i=1}^{n} \nu_i \leq \mu \tag{10}$$

where μ is the service rate of the priority scheduler itself. ν_n is given by

$$\nu_n = 1 - \sum_{i=1}^{n} \nu_i \tag{11}$$

The output loads of the queues $1, \ldots, n-1$ does not change. That of queue n is obtained by evaluation it with service rate ν_n.

4. Result and Discussions

The cloud systems processing resource and scheduling modeling result is presented in figure 8. The complete model shows the whole scheduling process ranging from data generation, showing newly generated data on queue to the processed data in cloud. The n partitioned tasks generated were as well shown. All stages of data processing 'Is Data Processed' Partitioned and Aggregation Processing of servers.

Figure 9 shows the N-Partitioned tasks generation. The jobs combine regenerated entity, replicated entity for partitioned task regeneration, time required to generate next partition, synchronization delay and next partition.

The prepared partitioned tasks for aggregation was modeled with the following variables: Number of Partitioned Tasks waiting for aggregation, the aggregate as OUT point for the aggregation queue, aggregation processing time per partitioned task, synchronization delay, reset the processing time to the aggregation time following the next partition. Results obtained were described by figures 10, 11, and 12.

5. Conclusion

From rapid application testing and development environment, cloud computing represents an exciting opportunity to bring on-demand applications to customers in an environment of reduced risk and enhanced reliability.

6. Recommendations

Researchers and information technology stakeholders monitoring a large number of clients should see the need to embrace the new development in data warehouse storage enhancement and increased performance to help improve the total throughput of their applications.

References

[1] China Web InfoMall, http://www.infomall.cn, 2008.
[2] The Hadoop Project, http://hadoop.apache.org/, 2008.
[3] The Kosmos F.S. Project, http://kosmosfs.sourceforge.net/, 2008.
[4] Tianwang Search, http://e.pku.edu.cn, 2008.
[5] Source Code of Tplatform Implementation, http://net.pku.edu.cn/˜ webg/tplatform, 2009.
[6] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A.Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: a distributed storage system for structured data. In OSDI ’06: Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation, pages 15–15, 2006.
[7] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In OSDI ’04: Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation, pages 137–150, 2004.
[8] G. Sanjay, G. Howard, and L. Shun-Tak. The google file system. In Proceedings of the 17th ACM Symposium on Operating Systems Principles, pages 29–43, 2003.
[9] H. Yang, A. Dasdan, R. Hsiao, and D. S. Parker. Map-reduce-merge: simplified relational data processing on large clusters. In SIGMOD ’07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data, pages 1029–1040, 2007.
[10] Z. Yang, Q. Tu, K. Fan, L. Zhu, R. Chen, and B. Peng. Performance gain with variable chunk size in gfs-like file systems. In Journal of Computational Information Systems, pages 1077–1084, 2008.
[11] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica. Improving mapreduce performance in heterogeneous environments. In OSDI ’07: Proceedings of the 8th USENIX.

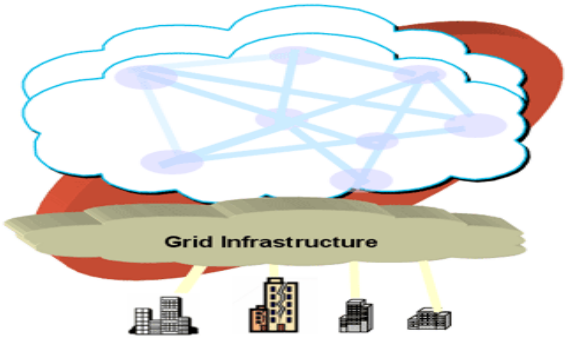

Figure 1: Grid Infrastructure

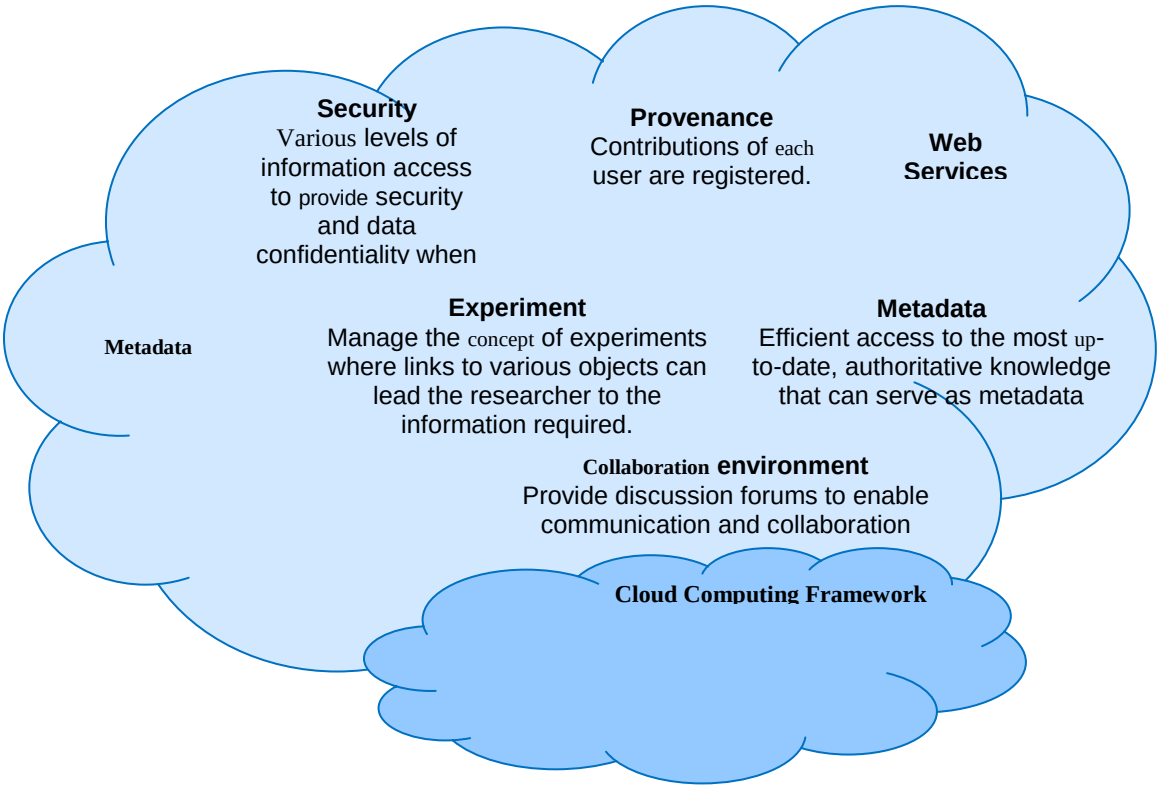

Figure 2: Cloud Computing Framework -1

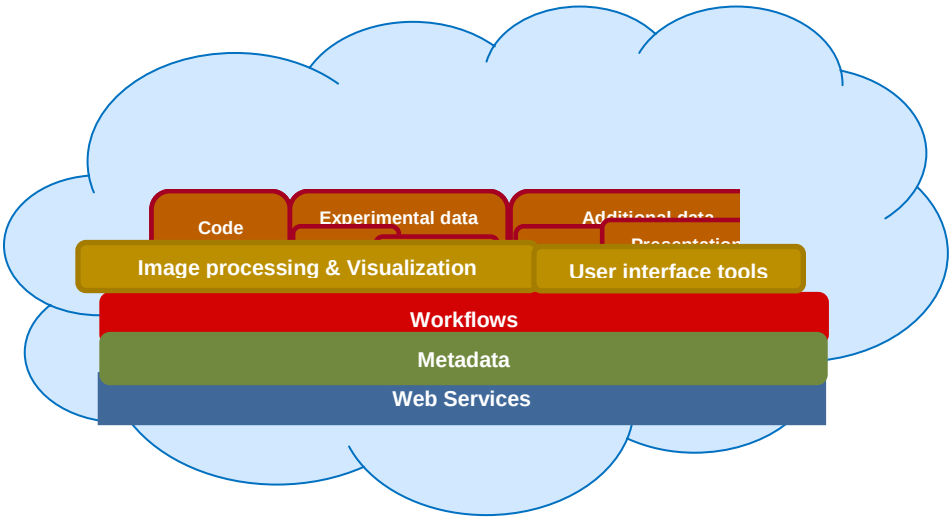

Figure 3: Cloud Computing Framework -2

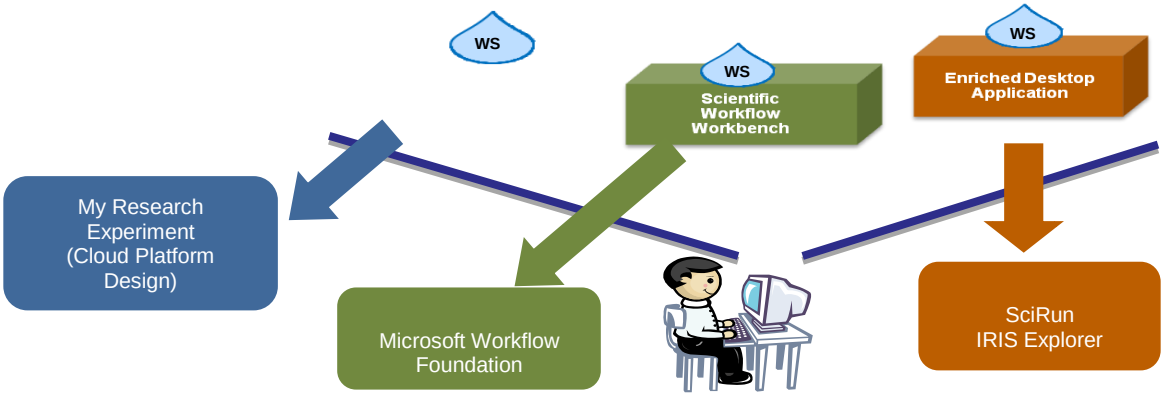

Figure 4: Cloud Computing Framework -3

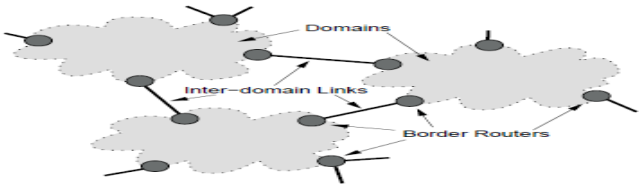

Figure 5: The modeling view for an Analytical Network Cloud

In traditional packet-based simulators are used, the “world” is modeled in terms of nodes and links with individual capacities and delay characteristics.

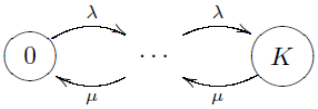

Figure 6: Birth and Death Process to describe Queuing Theory Approach Forwarding

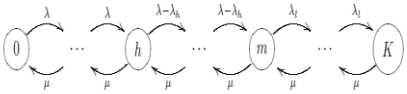

Figure 7: AF Birth and Death Process to describe Assured

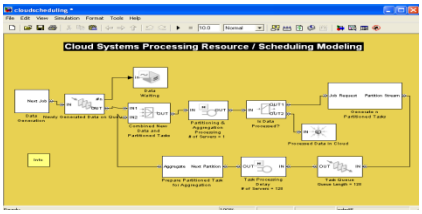

Figure 8: Cloud Systems Processing Resource and Scheduling Modeling Generation

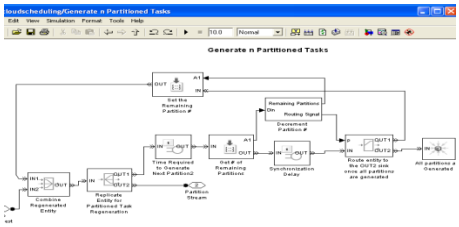

Figure 9: N-Partitioned Tasks

Figure 10: Prepared Partitioned Tasks for Aggregation

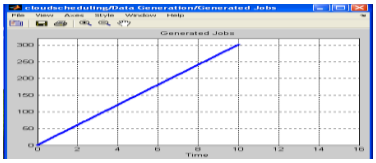

Figure 11: Plot of Generated Jobs against Time queue

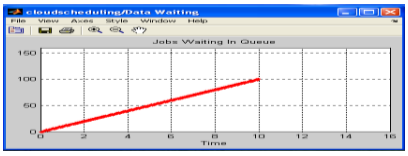

Figure 12: Plot of Jobs waiting in