**ASSIGNMENT : 3**

**1 . Feasibility Analysis :**

Emerging Cloud Computing technologies allow for an inexpensive use of mass quantities of storage, bandwidth and computing resources using the pay-per-use model on which it thrives. The adoption of broadband infrastructure in various forms (such as ADSL, 3G and Fibre) across cities allow for high bandwidth, low latency, high-reliability connections to the internet. Combining these ideas and focussing on the more fine-grained Infrastructure as a Service (IaaS) through the use of the Amazon Web Services (AWS) platform, this dissertation intends to make the case that the implementation of DevOps Systems ‘on the cloud’ is a feasible venture.

Private cloud is an alternative distribution paradigm for cloud services. Assets in this paradigm are fully dedicated for organization; thus, no anonymous organization can participate in those assets which can be hosted in the enterprise headquarter or outside. It is top for companies with active random computing requirements which need continuous management over their compute resources.

Containers make more efficient use of system resources than virtual machines. The latter generally require memory and storage space to be assigned to them before they start. Even if the apps running inside a virtual machine are not actually using all of the resources assigned to it, the virtual machine still monopolizes those resources. That’s not efficient.

Containers also offer the advantage of not having to duplicate the processes already running on the host system. With a container, you can run only the processes you need for whichever application you want to host inside the container. In contrast, virtual machines have to run a complete guest operating system, including many of the same processes that are already running on the server host.

**2 . Mathematical Model :**

Let S = {U, N, T, S, status, result, F} Where,

* U = { u1, u2, u3, .., ui.} Finite set of users (Candidates).
* N = { Master, Slave }

Where,

Master = Kubernetes master node. Slave = Kubernetes worker node.

* Slave = { slave1, slave2, ., slavei }
* T = { t1, t2, t3, ., ti } Finite set of test Scenarios.
* S = { s1, s2, s3, ., si } Finite set of scores of a user.
* status = { status1, status2, .., satatusi } Test status.
* Functionalities:

Yes/No = authenticate (uname, passwd)

Interface Candidate Web Brower (Wi)

Si = get test Scenario Score(ti)

Rank = apply ranking algorithm (ti, si) Result = generate test result.

**Cloud Computing is NP-Complete**

Due to formulation of resource assignment in a distributed cloud computing environment, which we term the CLOUD COMPUTING demand satisfiability problem, is NP -complete, using transformations from the PARTITION problem and 3-SATISFIABILITY, two of the “core” NP-complete problems.

For Ex :

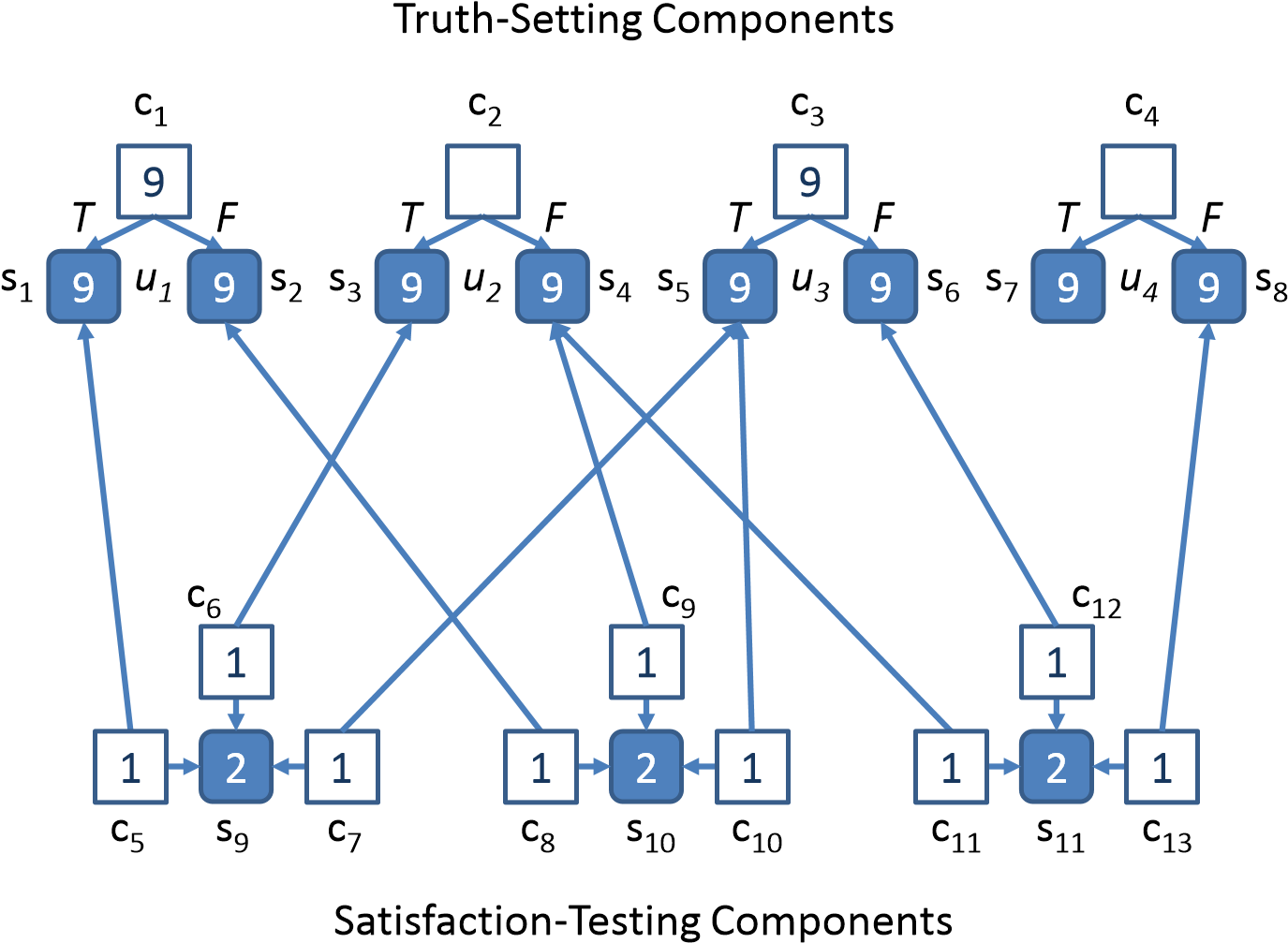
Let there be a set of users, each with a given level of demand for resources, and a set of servers, each with a given level of capacity, where each customer may be served by two or more of the servers. The general problem of determining whether there is an assignment of users to servers such that each customer’s demand may be satisfied by available resources is NP-complete.

To prove CLOUD COMPUTING is NP-complete, we need to show that

1. Since, 3-SAT is a NP-COMPLETE problem and deducing any problem to it makes the problem NP-COMPLETE.There is a transformation of any instance of a known NP-complete problem, in this case, 3-SAT, into an instance of CLOUD COMPUTING that is satisfiable if and only if there is a solution to the original instance of 3-SAT.

1. This transformation from 3-SAT to CLOUD COMPUTING can be done in polynomial time

* **Transformation of 3-SAT Problem (Known NP-Complete) to instance of Cloud Computing in polynomial time**



This example CLOUD COMPUTING instance is a transformation from an instance of 3SATISFIABILITY corresponding to four Boolean variables  and

 or equivalently,

.

Given a truth assignment of  or  to variable , we set each truth-setting component accordingly so that the “switch” customer gets its resources from the selection customer corresponding to . This leaves sufficient resources in the selection labeled with the same truth value as . Since there is a satisfying assignment to the instance of 3-SATISFIABILITY, this means that each satisfaction-testing component has at least one term that is satisfied, thus can find at least  resource from the truth-setting layer. Therefore, there are at most  resources needed by the satisfaction-testing component, which can be satisfied by the “gap-filling” server.