International Journal of Engineering & Technology, 7 (2.4) (2018) 123-130

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research Paper

Resource provisioning methodology for cloud environment with producer and consumer favorable: an approach of virtual cloud compiler

V. Vivek [1] *, R. Srinivasan [2], R. Elijah Blessing [3]

[1] *Research Scholar School of Computer Science and Engineering, Karunya Institute of Technology & Science, Coimbatore, India*
[2] *Research Supervisor School of Computer Science and Engineering, Karunya Institute of Technology & Science, Coimbatore, India*
[3] *Professor School of Computer Science and Engineering, Karunya Institute of Technology & Science, Coimbatore, India*
**Corresponding author E-mail:*

Abstract

Cloud computing is a model where traditional resources such as CPU cycles, storage, security etc. are delivered through web based. It is a technology which has ability to change large part of software development cycle, 3D rendering or any other computationally expensive tasks execution. Much amount of time is wasted on compiling and rendering such computationally expensive tasks due to low power machines, which directly proportional to efficiency of user who is working on that project. Extreme computational tasks such as weather forecast, DNA analyses, encryption breaking takes so much time in consumer grade computing devices that they are realistically not possible to perform. We have proposed a novel approach to perform payload distribution, for the users who wanted to run their computationally expensive tasks efficiently. We have used virtualization technique on data center resources to perform scheduling. Up to 32% cost has been reduced in an environment of 30 users when our technology used instead of traditional standalone desktop environment. This is achieved by replacing 30 standalone computers with a powerful server and thin clients like Raspberry pi as clients. Time wasted in computational task such as rendering and compiling is greatly reduced. We have not only improved the efficiency, but also make sure both cloud producer and consumer are favorable. With simulations and outcomes, we validate that our methodology for payload distribution performs well.

Keywords: *Cloud Computing; Heavy Computational Task. Payload Distribution; Virtual Compiler*

1. Introduction

In summary cloud computing is a model where, unutilized resources are efficiently utilized to perform high level computing with a lower cost. In the current era technology is the top most key factor. Any digital computer including the home based personal computers to office desktop computers are configured with high-end resources. Most of the users are not completely utilizing these available resources. To address this issue, grid, utility and cloud computing has evolved the other. Cloud basic principle is to lease computing power and store capacity to your desktop through web based access. The National Institute of Standards and Technology (NIST) defines cloud computing as a "pay-per-use model for enabling available, convenient and on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [1] [2]. All computing services are broadly categorized in to three types, they are software as service (Saas), platform as a service (Pass) and infrastructure as service (Iaas) [3] [4]. Users can lease these computing services / resources in four methods such as public, private, community and hybrid. In cloud, there are mainly two actors one is cloud provider and the other is user. The general communication between these two are show in fig 1. User send his request to the cloud provider for a resource, up-on receiving the request from the user the provider performs matchmaking [5] operation to search for the resource which can satisfy the user request and allocates it. There-after the user runs his application on the assigned resources. Various cloud deployment models and their characteristics are given in fig 2.

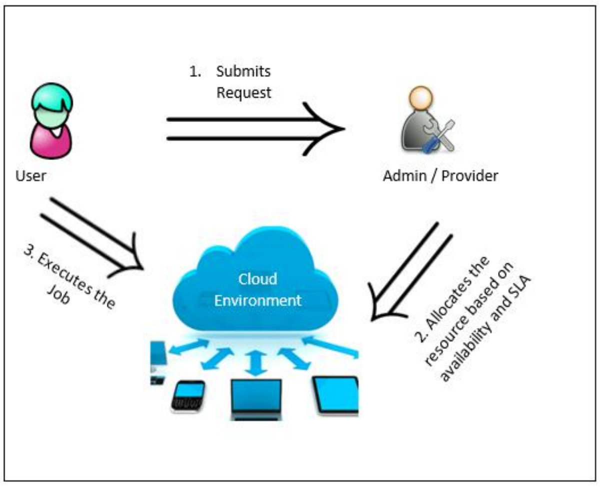


Fig 1: Cloud Environment and Usage Scenario

Rapid fast growing technology drives many organizations to adopt cloud environment for compiling their projects. Currently most of the compilers are standalone which are manually installed in the user devices. These compilers take text as an input and convert it to be executable and object code. For every different scripting language and database, user have to install it separately which may require more storage and configuration process. Installation and the performance of the compilers depends on the user device configuration. Scripting languages like C++, Java, .Net, C#, Visual Basic, Perl, Python, Ruby, CSS, Java script require very high-end devices to run individually and also consumes lot of power. When there is a need to run these scripts along with the databases like SQL or mongoDB it requires even more high-end configured devices and more power [6]. It is expensive to purchase such high-end devices for an individual or small scale organization. As a result, in most scenarios user will end up crashing the compliers when they run heavy scripts or database quires, which requires reinstallation or admin control to restore back [7]. As some scripting languages like Java and .Net need database connectivity, these settings have to be configured manually by the user for every project which is an additional task. There are applications which are platform dependent, for example few compliers like Ns3 will run on Linux environment.

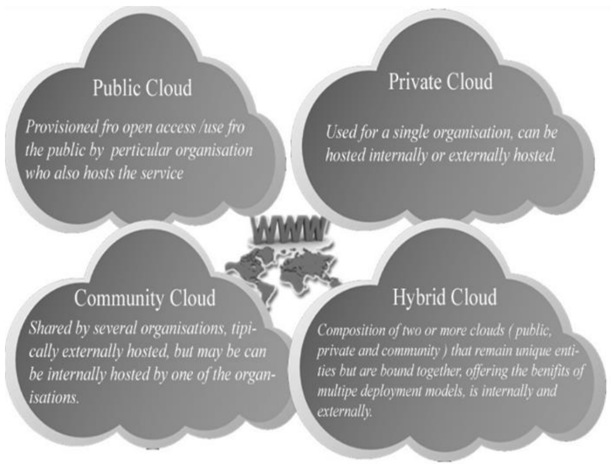

Public Cloud
Provisioned fro open access /use fro the public by perticular organisation who also hosts the service
Private Cloud
Used for a single organisation, can be hosted internally or externally hosted.
WWW
Community Cloud
Shared by several organisations, tipically externally hosted, but may be can be internally hosted by one of the organisations.
Hybrid Cloud
Composition of two or more clouds (public, private and community) that remain unique entities but are bound together, offering the benifits of multipe deployment models, is internally and externally.

Fig .2: Cloud Deployment Models & Characteristics.

In our paper, we propose a novel approach where we design a cloud environment for users to run their scripts and database queries through web browsers. We have used thin client devices for efficient, fair and starvation free distribution of payloads. The remaining sections of this paper are, categorized as following, section 2 related work, section 3 proposed system architecture, section 4 implementation of proposed architecture, section 5 comparisons of existing and proposed model, and final section is conclusion.

2. Related Work

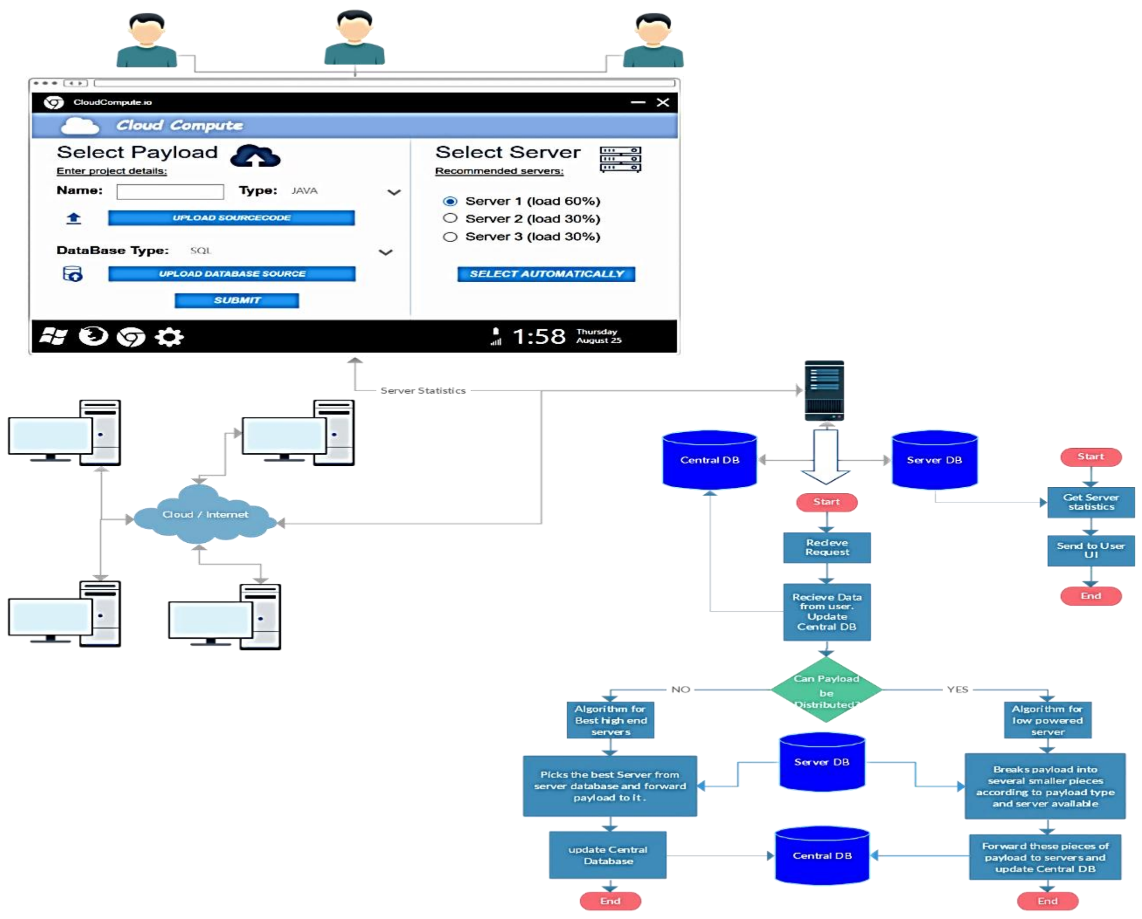
There are various approaches existing where payload distribution is done and executed in different machines. One such approach is Single System Image (SSI) [8] [9] [10], in this methodology resources from various low powered machines are pulled up for making single large virtual machine. The environment in which operating system and other applications run in such a setup is logically and technically same as running inside a single powerful server. This allows to break payloads at OS level, which may result in greater execution efficiency in some scenarios. OpenMosix [11] [12] and OSCAR [13] [14] are projects which achieves SSI. There is various downside of this approach, some of them are,

a) Although process migration is supported which allows one process to shift to another physical CPU or other CPU – core inside a CPU, in this case other CPU is resides in a different computer. This increases execution time drastically, as the execution data has to be copied from memory of one computer to another.
b) There is no reliable way to know exact CPU load in any computer which is contributing to SSI.
c) Only Linux systems are supported by SSI, hence computers running another OS cannot contribute to SSI [14].
d) To properly scale this type of architecture, a lot of low level (kernel level) changes have to be made which can be very time consuming and only highly skilled IT professionals can perform it.
e) Many new hardware components like Nvidia, tesla graphic accelerator and Intel Xeon phi do not support SSI at the time of writing due to driver related issues [23].

Web based solution was chosen to achieve efficient payload distribution as this provides complete compatibility between different OS. The “servers” (contributors) can install compute packages which are written for various OS like windows and Linux, this allows accurate CPU load to be reported back to broker. Further “client” (consumer) can communicate to broker just like visiting any other website. This solution is highly scalable, reliable, easy to maintain as different modules i.e. broker, compute-package, web services (front end to consumer) can run independently and can be updated without affecting other modules. Through this approach new “compute-packages” can be made to support new type of payload execution. By choosing high level languages like php to make framework and compute packages, reliability and security is assured as higher level languages gets frequent updates and is supported by large number of developers. Choosing such framework further helps in installation of this architecture as Product based Service (PBS) by organization who want to install this architecture in their own hardware. Most of the system architecture including broker, front end and most part of ‘compute-packages’ is written in php for easy maintenance and scalability.

3. System architecture

The system architecture is explained by taking two different types of requests (request is raised when consumer submits data to broker for processing) where one can be broken down into various payloads efficiently like breaking a hash and the one where it cannot be broken down at all for



example compiling a java source code. All the other types of requests sit in between these two types of requests in terms of how efficiently they can be broken into smaller payloads and executed independently [15]. All the servers have different types of compute packages installed in them which facilitates payload execution and communication to broker. There are different compute packages available which work independently. Hence owner of server can choose which type of payloads are executed in their hardware which, reduces security vulnerability as only specific type of payload gets executed in servers. Fig 3 represents the overall system architecture.

1) Case 1 – (Requests which cannot be broken into smaller pieces) this may be considered as a worst-case scenario for this architecture, as request cannot be broken into smaller payload. Hence multiple resources cannot be used to execute task (payload) quickly. In this case, broker will automatically select single most powerful server and execute payload on that. Flow of such type of requests has been show in Fig 4. This will greatly improve efficiency of the worst-case scenarios where request cannot be broken into smaller payloads.

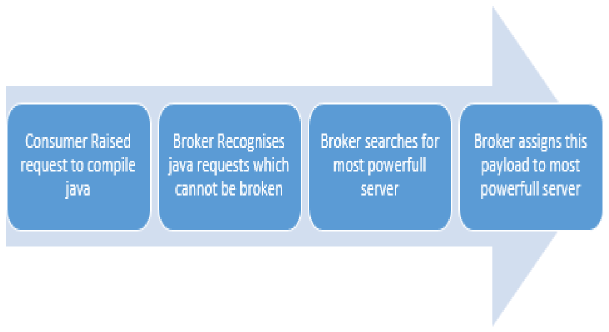

Consumer Raised request to compile java
Broker Recognises java requests which cannot be broken
Broker searches for most powerful server
Broker assigns this payload to most powerfull server

Fig. 4: Requests, Which Cannot Be Broken Into Smaller Pieces.

As soon as consumer uploads data, java source code in this example, the source code along with all the details will get stored in central ddatabase along with unique job ID. Broker will analyze request and once concludes that it cannot be broken down into smaller pieces / payloads, it will search for most powerful server with low CPU load and pass the unique job ID to this server. Once server receives the job ID, it will fetch necessary files like source code and execute payload, which will compile that source code and upload the executable file back to broker.

2) Case 2 – (Requests can be efficiently broken into smaller payloads) This may be considered as a best case scenario , as the request raised by user can be broken into various small payloads effectively. In this case, broker will break the requests into various small parts and assign them to comparatively lower powered server or the servers which have high CPU load. This utilizes low powered machine efficiently, without increasing execution time. Flow of such type of requests has been show in Fig 5.

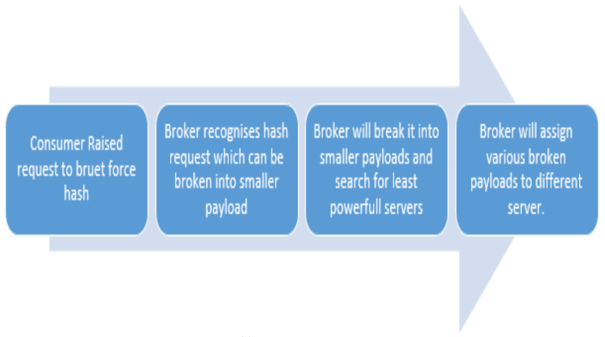

Consumer Raised request to bruet force hash
Broker recognises hash request which can be broken into smaller payload
Broker will break it into smaller payloads and search for least powerfull servers
Broker will assign various broken payloads to different server.

Fig 5: Requests Can Be Efficiently Broken Into Smaller Payloads.

As soon as user enters hash and other-necessary details, the broker will raise a request and enter all details into the central database along with a unique job ID. Broker will then analyze request and conclude that it can be broken into various payloads. After this broker, will divide the single payload into multiple payloads and store them with multiple unique job IDs, each payload will contain only partial number of total words / patterns which has to be brute forced. Broker will then assign these job IDs to various severs which will get hash and wordlist according to the job ID given to them. Once they finish execution of their payload, they will report result back to the broker.

4. Implementation

For such architecture implementation, three thin clients (Raspberry pi) , three standard desktop computers as servers and one standard desktop computer as broker were used. All the machines are running on open source Linux operating system. It should be noted that severs can also run on different operating systems, as compute packages can be ported to windows operating system. Compute packages are dependent on other software's to execute their payloads efficiently. These packages are built to complete separate execution environment of payloads and user activities in servers. All the other software's used in compute package are free and open-source in most cases, the complete scenario is shown in Fig 6 below.

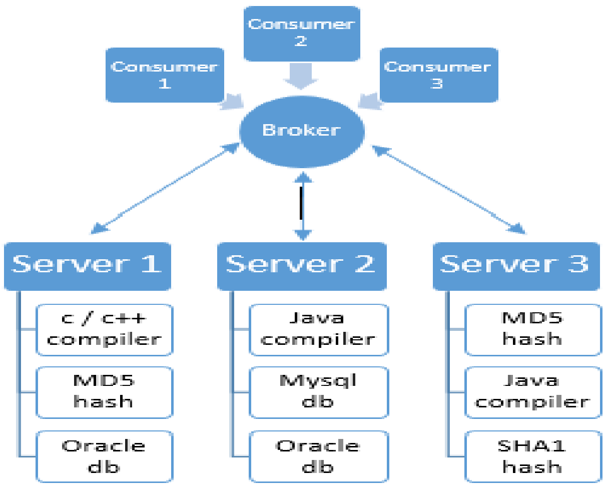
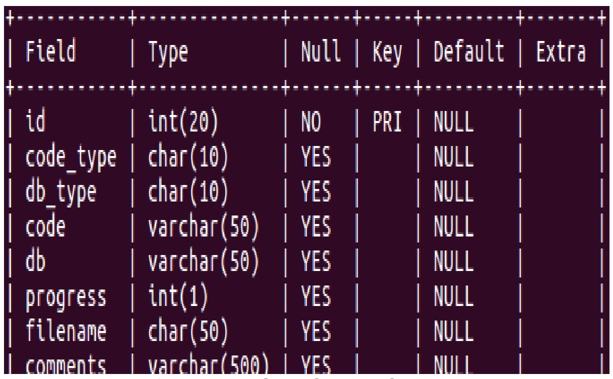


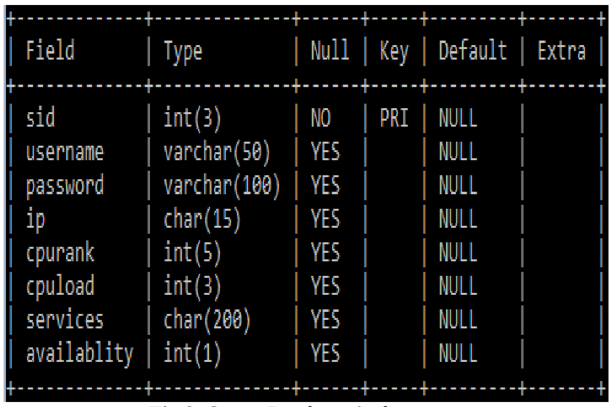
Fig 6: Implementation Architecture.

Entire frame work of this architecture is coded in php, with minimum plugins to keep it secure and efficient. Broker provides a light-weight form through which consumers can upload their data and raise request. Broker also maintains a Central Database (given in Fig 7) which is coded in MySQL to store different requests as Job IDs, and a server database (given in Fig 8) which stores current server status, sessions, etc.



```
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| id        | int(20)      | NO   | PRI | NULL    |       |
| code_type | char(10)     | YES  |     | NULL    |       |
| db_type   | char(10)     | YES  |     | NULL    |       |
| code      | varchar(50)  | YES  |     | NULL    |       |
| db        | varchar(50)  | YES  |     | NULL    |       |
| progress  | int(1)       | YES  |     | NULL    |       |
| filename  | char(50)     | YES  |     | NULL    |       |
| comments  | varchar(500) | YES  |     | NULL    |       |
```

Fig. 7: Central Database Architecture.



```
+------------+--------------+------+-----+---------+-------+
| Field      | Type         | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
| sid        | int(3)       | NO   | PRI | NULL    |       |
| username   | varchar(50)  | YES  |     | NULL    |       |
| password   | varchar(100) | YES  |     | NULL    |       |
| ip         | char(15)     | YES  |     | NULL    |       |
| cpurank    | int(5)       | YES  |     | NULL    |       |
| cpuload    | int(3)       | YES  |     | NULL    |       |
| services   | char(200)    | YES  |     | NULL    |       |
| availablity| int(1)       | YES  |     | NULL    |       |
+------------+--------------+------+-----+---------+-------+
```

Fig 8: Server Database Architecture.

CPU rank is given based on which CPU is used in server, when server is registered in broker, it sends its CPU name and ram to the broker then broker will calculate its rank by fetching various benchmark values available online and update server table. Higher CPU rank means more powerful CPU. Another column 'availability' is used to maintain state of various servers. Here, list of these codes is given below

1) future use
2) not available
3) Job already running
4) CPU load is 86% to 100%
5) CPU load is 76% to 85%
6) CPU load is 61% to 75%
7) CPU load is 46% to 60%
8) CPU load is 31% to 45%
9) CPU load is 16% to 30%
10) CPU load is 0% to 15%

Server owners can select the maximum level they want to share their resources. For example, if server owner wants to share his resources till CPU usage reaches 75%, the compute package will keep updating broker with various values from 9 to 5 depending on current usage, but once CPU load goes beyond 75% compute package will update sever availability as 1 and no more payloads will be given to this server. There are various algorithms used while selecting servers for different types of payloads. For the worst-case scenarios where request cannot be broken into different payload loads, most powerful server is used to complete the task. It should be noted that algorithm is preconfigured to select only those servers which full fill requirements of request raised by user. This can be done by reading data in 'services' column of server database.

Algorithm 1– This algorithm is optimized specially for worst-case scenarios i.e. for payloads like java, c/c++ , pearl , R , etc.

```
Server = fetch_servers(50000 , 9);
// fetch servers with rank 50,000 and above AND with status code 9
Rank = 50000, avail = 9 , flag = 1; lable:
For (;(server.isEmpty());rank = rank – 1000) {
If (10000 <rank < = 30000) {
ranktemp = 50000 , availtemp = 3 ;
For (;(server.isEmpty()) && ranktemp > = 30000;ranktemp = ranktemp – 1000){
Server = fetch_server(ranktemp, availtemp);
}
}
If (5000 < rank < = 10000) {
ranktemp = 30000 , availtemp = 3 ;
For (;(server.isEmpty()) && ranktemp > = 10000;ranktemp = ranktemp – 1000){
Server = fetch_server(ranktemp, availtemp);
}
}

If (rank < = 10000 && flag == 1) {
Break ();
askuser();
}
Avail = 9;
For (;(server.isEmpty()) && avail > = 4; avail--){
Server = fetch_server(rank , avail)
}
}
if ( server.isEmpty() ) {
print "All server are busy , please try again later"
}
Else {
assign_job(server);
}
askuser() {
Print "Only low end servers are available , this might take more time than expected to execute payload"
Print "Enter yes to continue , no to wait for 10 minutes and retry , quit to remove request"
Choice = input ();
Switch (choice) {
Case 'yes': flag = 0;
goto lable;
Case 'no': sleep (10);
goto lable;
Case 'quit': delete request ();
}
```

In this algorithm, the broker will first search high end servers with 0 to 15% CPU load, if it doesn't find any such servers then it will search for high end server with 16% to 30% load, and so on till it gets to 85% load, as executing such payloads on high end servers with some load is more efficient than executing it in low end servers with very low CPU loads. Once the algorithm reaches 30,000 rank and still didn't found any suitable server, it will go back to the high-end servers with more than 85% load, which are ready to share their resources. If no server is found, it will continue the same process with server with ranking 30,000 to 10,000. When no sever if found even at 10,000 rank with less than 85% load, the algorithm will again search servers from rank 30,000 to 10,000 with more than 85% load. After multiple attempts, user will be prompted if he wants to continue searching as, executing payloads on very low servers can be very time consuming and in-efficient. If user agrees to execute it will search for even lower ranked servers. Finally, if there is no server found than it will pass the message that no suitable sever is found.

Another type of algorithm is used when request can easily be divided into multiple payloads, this algorithm is optimized for cracking / brute forcing Hashes. Here, algorithm will divide large wordlist submitted by user to smaller wordlists according to server's type and number of servers available.

Algorithm 2-

```
totalfilesize = getFileSize();
baseval = 20
For (rank = 1; rank < 90000; rank = rank + 5000) {
Server [] = search_servers(rank); // returns server array based on availability.
For (availability = 9; availability < = 3; availability ++) {
Server list [] = server [availability]; // get linear list of all severs.
For-each serverNumber in serverlist[] {
Server = serverlist[serverNumber] ;
currentVal = baseval * availability ;
createjob (Server , currentVal);
totalfilesize = totalfilesize = currentVal;
}
}
baseval = baseval + 50;
}
```

This algorithm will divide large wordlist into smaller wordlists and make multiple jobs based on that. Algorithm starts with lower powered devices, as this type of payload can be divided into very small payloads. The original request it divided into smaller payload according to availability and rank of server, smaller rank (about 5000) server with 50% load will receive payload of 120 units, while higher end devices with rank (30,000) with 50% system load will receive 1620 units of payload. There were various compute packages installed on server. Some of the compute packages are given below -

Java Compiler - java compute package enables processing of java payloads, .i.e compilation of java source code to java executable file. This compute package is dependent on following software's – Unzip, javac , jar. Here is the portion of code used in making java compute package –

1) $sql = "select code,filename from cloud where id ='".$ID."';";
2) $result = $conn->query($sql);
3) $row = $result->fetch_array(MYSQLI_ASSOC); // fetch the row
4) result->free(); //free mem
5) $name = $row["filename"];
6) $path = $row["code"];
7) $conn->close();
8) Echo exec ("unzip ". $path." -d upload/". $ID);
9) Echo exec ("cd upload/". $ID."/ && javac ".$name.". Java");
10) Echo exec ("cd upload/". $ID."/ && jar cf ".$name.". Jar ". $name." *.class");

The line 1 to 3 are used to connect to central database and fetch files & necessary details based on job ID provided by broker. line 4 and 7 are used to free memory and connections so that less memory is used by server and central database. line 5 and 6 retrieves details of java source code like filename, name of main class , etc. in line 8,9,10, - unzip will decompress the data received from central database then javac is used to compile this code and finally jar is used to form a executable from this compiled code and sent back to central DB and broker. Any errors occurred during these process is logged into Central database according to job ID. This helps in debugging payloads and compute packages.

MySQL DB - This compute package is an auxiliary package , i.e. It is generally installed with other compute packages such as java compute package or c compute package to provide database connectivity to payloads which need database connectivity in order to run. Here is the portion of code used in making of MySQL package

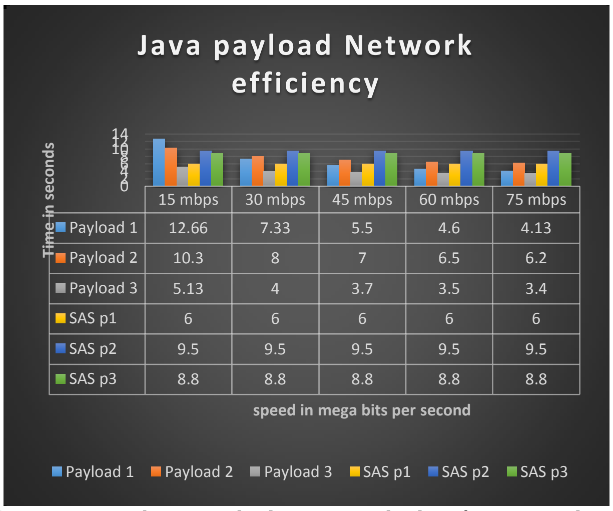
1) $conn = new mysqli($servername, $username, $password, $dbname);
2) $sql = "select db from cloud where id ='".$ID."';";
3) $result = $conn->query($sql);
4) $row = $result->fetch_array(MYSQLI_ASSOC); // fetch the row
5) $Result->free();//free mem
6) $db = $row["db"];
7) $Command = "mysql -u root -pyash -D cloud < ".$db;
8) Echo shell_exec($command);

Line 1 to 4 is used to connect to central Database and fetch source file, on the basis of job ID given by broker. line 7 and 8 are used to login to separate account in mysql instance and import database by importing source file fetched from central DB. If Mysql is already installed in server then separate user is created in Mysql DB during installation of this package with limited permissions to secure payload execution and separate user and payload Databases.

MD5 hash – This type of packages is very efficient as the requests of such type of compute resource can be broken down into many smaller payloads without affecting integrity of the problem. Due to such efficient payload distribution, low powered devices like Raspberry pi, mobile phone and old laptops can also act as a server and receive small payloads. This compute package is depended on the open source software called 'hash-cat'

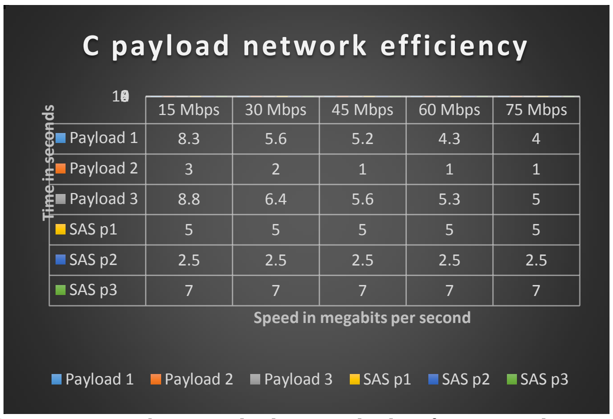
1) $Conn = new mysqli($servername, $username, $password, $dbname);

2) $sql = $sql = "select code, db from cloud where id ='". $ID."';";
3) $Result = $conn->query($sql);
4) $row = $result->fetch_array(MYSQLI_ASSOC); // fetch the row
5) $result->free(); //free mem
6) $wordlist = $row["db"];
7) hash = $row["code"];
8) $echo shell_exec("hashcat -m 0 -a 0 ". $hash." ".$wordlist);



	15 mbps	30 mbps	45 mbps	60 mbps	75 mbps
Payload 1	12.66	7.33	5.5	4.6	4.13
Payload 2	10.3	8	7	6.5	6.2
Payload 3	5.13	4	3.7	3.5	3.4
SAS p1	6	6	6	6	6
SAS p2	9.5	9.5	9.5	9.5	9.5
SAS p3	8.8	8.8	8.8	8.8	8.8

Fig 9: Non-Distributive Payloads - Java Payload Performance and Network Efficiency.



	15 Mbps	30 Mbps	45 Mbps	60 Mbps	75 Mbps
Payload 1	8.3	5.6	5.2	4.3	4
Payload 2	3	2	1	1	1
Payload 3	8.8	6.4	5.6	5.3	5
SAS p1	5	5	5	5	5
SAS p2	2.5	2.5	2.5	2.5	2.5
SAS p3	7	7	7	7	7

Fig 10: Non-Distributive Payloads - C Payload Performance and Network Efficiency.

Line 1 to 7 are used to connect to central database and retrieve hash and wordlist based on job ID. Then in line number 8 an open source program called Hashcat is used to brute force hash, although this software supports many hashes like SHA1, md4 , etc , in compute package only MD5 is supported. Hashcat is designed to efficiently work on OpenCL supported devices, which means graphic accelerators embedded in lower power devices like ARM based CPUs can also be used to brute force. Performance metrics different types of payloads are given below. Java and C payloads were considered for non-distributed payload and MySQL is considered for distributed payload.
Java Payload Metrics with reference to Fig 9,
Handshake time (when a server makes a connection to broker and both get in sync) is about 1 second and Latency of connection of all test scenarios were from 1 ms to 70 ms, In these metric evaluation, there types of payloads were used –

a) "Payload 1 "- Execution time 2 seconds, source code size 10 MB.
b) "Payload 2 "- Execution time - 5 seconds, source code size - 5 MB.
c) "Payload 3" – Execution time – 3 seconds, source code size – 2 MB.
d) "SAS p1" – Stand Alone system executing payload 1 – 6 seconds.
e) "SAS p2" – Stand Alone system executing payload 2 – 9.5 seconds.
f) "SAS p3" - Stand Alone system executing payload 3 – 8.8 seconds.

C Payload Metrics with reference to Fig 10.

a) "payload 1" - execution time – 3.5 sec, code size 5MB
b) "payload 2" – execution time – 1 sec. code size 3MB
c) "payload 3" – execution time – 4.4 sec, code size 4.2MB
d) SAS – payload 1, execution time – 5 sec
e) SAS - payload 2, execution time – 2.5 sec
f) SAS – payload 3, execution time – 7sec

MySQL and PostGre Payloads Metrics regarding Fig 11.
We saw a slight increase in performance with increase in number of total records (queries) in database.

a) Test case 1 (4k records): 100 seconds for cloud compute vs. 120 for local.
b) Test case 2 (10k records): 250 seconds for cloud compute vs. 300 for local.
c) Test case 3 (20k records): 500 seconds for cloud compute vs. 600 for local.

We can use multiple machines to provide additional redundancy and backup for the data.
Hashing Payload Metrics regarding Fig 12.

a) Test case 1 = Stand Alone system: 8 Million Calculation per sec; Cloud compute = 5 x 5 M c/s (5 low powered machines)
b) Test case 2 = Stand Alone system: 8 Million Calculation per sec; Cloud compute = 10 x 5 M c/s (10 low powered machines)
c) Test case 3 = Stand Alone system: 8 Million Calculation per sec; Cloud compute = 5 x 8 M c/s (5 medium powered machines)
d) Test case 4 = Stand Alone system: 8 Million Calculation per sec ; Cloud compute = 5 x 20 M c/s (5 high end machines)

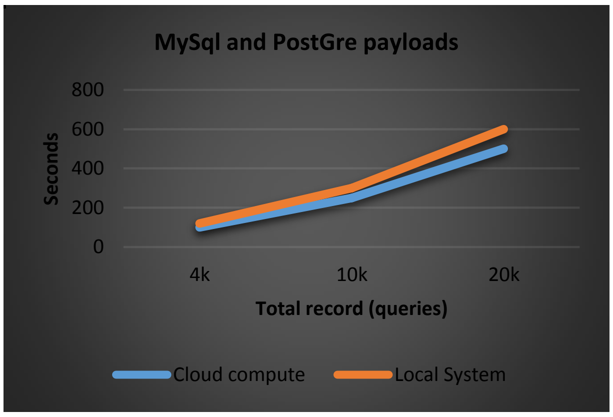
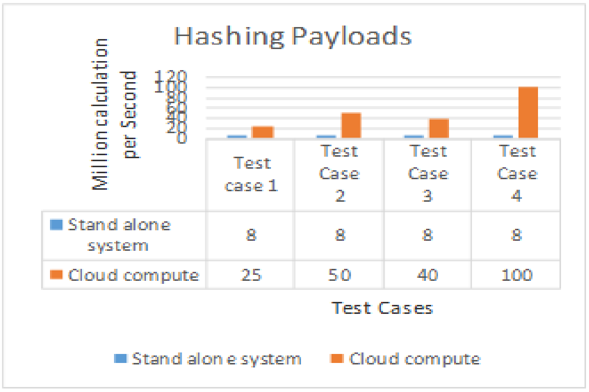


Fig 11: Distributive Payloads – Mysql and Postgre Payload Performance and Network Efficiency.



	Test case 1	Test Case 2	Test Case 3	Test Case 4
Stand alone system	8	8	8	8
Cloud compute	25	50	40	100

Fig 12: Test Cases –Hashing Payload.

5. Conclusion

The methodology proposed in this paper for using resources offered by servers who have excess compute power and used by consumers who needs extra compute power, not only distributes compute power in efficient way, but also reduces overall carbon footprint of doing any type of expensive computational tasks, while overcoming various short-comings of other distributed computing systems such as SSI (Single system Image). Such as managing payloads at higher level of OS architecture to monitor CPU usage and distributing payloads across multiple systems without adverse effects on throughput of the system. We were able to efficiently distribute both “Distributable payloads” such as MySQL, Postgre sql DBs and hashes and “Non-Distributable” payloads, such as compilation programming languages like java , c++ , etc. We made algorithms, which efficiently ranks each server based on their CPU power and current CPU usage and picks high end machines for non-distributable payloads and low-end machines for distributable payloads. This allows vast number of devices to participate as servers (From high-end workstations to low powered IoT devices such as Raspberry pi). Algorithm also ensures that server doesn’t get overwhelmed by executing payloads of consumers by constantly checking current CPU usage and diverting payloads if CPU usage goes above desired percentage. In other environments, such as a traditional lab environment, this approach has proven to reduce cost up to 35% and power usage was cut down up to 1/10th of implementing same amount of nodes , while maintaining compute power of individual nodes in a traditional lab environment! In future, 92% percentage of workloads will be handled by cloud data centers; versus only eight percent being processed by traditional data centers. [18] [19] With the rise in dependency on cloud computing and rise of IoT devices it is necessary to use proper tools and algorithms as specified in this paper to handle data at such a volume, much more efficiently. Modular approach was kept in mind while designing these methodologies which allows addition of various modular component like simulation, support for machine learning, inclusion of other compiled languages, etc. IoT is one of the most rapidly growing field, which uses various types of SoC (System on Chip) devices to perform its core operation, many of these devices are way more powerful than what is required to perform these core operations like maintaining databases, syncing various nodes, etc. This extra power can be easily used for other tasks such as an independent node for AI or Machine Learning, or simply contributing to other modules such as java compiler, c++ compiler, etc.

References

[1] Mell, P., & Grance, T. “The NIST Definition of Cloud Computing” (Draft) Recommendations of the National Institute of Standards and Technology. Nist Special, 145(6), [7]. National Institute of Standards and Technology, Information Technology Laboratory. Retrieved from http://csrc.nist.gov/publications/drafts/800145/Draft-SP 800145_ cloud definition. Pdf (2011).

[2] Saurabh Kumar Garg, Christian Vecchiola, Rajkumar Buyya “Mandi: a market exchange for trading utility and cloud computing services” The Journal of Supercomputing June 2013, Volume 64, Issue 3, pp 1153-1174.

[3] M. N. O. Sadiku, S. M. Musa and O. D. Momoh, "Cloud Computing: Opportunities and Challenges," in IEEE Potentials, vol. 33, no. 1, pp. 34-36, Jan.-Feb. 2014. doi: 10.1109/MPOT.2013.2279684

[4] Sunilkumar S. Manvi,, Gopal Krishna Shyam, "Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey Review Article "Journal of Network and Computer Applications, Volume 41, May 2014, Pages 424-440.

[5] Giuseppe Di Modica, Orazio Tomarchio, "Matchmaking semantic security policies in heterogeneous clouds", Future Generation Computer Systems, Volume 55, February 2016, Pages 176-185.

[6] Adam Chlipala. 2015. An optimizing compiler for a purely functional web-application language. In Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming (ICFP 2015). ACM, New York, NY, USA, 10-21.

[7] N. A. B. S. Chebolu and R. Wankar, "A novel scheme for Compiler Optimization Framework," 2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Kochi, 2015, pp. 2374-2380.

[8] E. Hendriks. “BProc: The Beowulf distributed process space.” ACM Proceedings of ICS, 2002.

[9] Ke Wang, Xiaobing Zhou, Kan Qiao, Michael Lang, Benjamin McClelland, and Ioan Raicu. 2015. towards Scalable Distributed Workload Manager with Monitoring-Based Weakly Consistent Resource Stealing. In Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '15). ACM, New York, NY, USA, 219-222

[10] http://www.cloudbus.org/papers/SSI-CCWhitePaper.pdf

[11] http://www.openmosix.org/

[12] de Robles, Marie Yvette B.; Arnejo, Zenith O.; Pabico, Jaderick P, "On Web-grid Implementation Using Single System Image",Computer Science - Distributed, Parallel, and Cluster Computing, arXiv:1507.01067

[13] G. Vallee, S. L. Scott, C. Morin, J. Y. Berthou and H. Prisker, "SSI-OSCAR: a cluster distribution for high performance computing using a single system image," 19th International Symposium on High Performance Computing Systems and Applications (HPCS'05), 2005, pp. 319-325.

[14] Philip Healy, Theo Lynn, Enda Barrett, and John P. Morrison. 2016. Single system image. J. Parallel Distrib. Comput. 90, C (April 2016), 35-51. DOI=http://dx.doi.org/10.1016/j.jpdc.2016.01.004

[15] Frederic Magoules, Jie Pan, -Fei Teng, "Cloud Computing: Data Intensive Computing and Scheduling", Chapman & Hall/CRC ٢٠١٣Numerical Analysis and Scientific Computing,

[16] Chaisiri, S.; Bu-Sung Lee; Niyato, D., "Optimization of Resource Provisioning Cost in Cloud Computing," Services Computing, IEEE Transactions on, vol.5, no.2, pp.164,177, April-June 2012doi: 10.1109/TSC.2011.7

[17] Amazon EC2 Reserved Instances, [ONLINE]

[18] http://www.forbes.com/sites/joemckendrick/2016/11/13/with-internet-of-things-and-big-data-92-of-everything-we-do-will-be-in-the-cloud/#639f6196593f

[19] http://www.cisco.com/c/en/us/solutions/service-provider/visual-networking-index-vni/index.html#cloud-forecast http://aws.amazon.com/ec2/reserved-instances, 2013.

[20] http://www.cloudbus.org/

[21] Transitioning to the Private Cloud with Confidence Cisco Web Learning White papers. http://www.cisco.com/web/learning/le21/le34/downloads/689/rsa/Cisco_transitioning_to_the_private_cloud_with_confidence.pdf

[22] V. Vivek; Srinivasan R; Elijah Blessing Rajsingh Resource Provisioning Methodologies: An Approach of Producer and Consumer Favorable in Cloud Environment’, International Journal of Emerging Technology and Advanced Engineering, Volume 3, pp.8-13, Special Issue 4, October 2013 (ISSN 2250 – 2459)

[23] Philip Healy, Theo Lynn, Enda Barrett, and John P. Morrison. 2016. Single system image. J. Parallel Distrib. Comput. 90, C (April 2016), 35-51. DOI=http://dx.doi.org/10.1016/j.jpdc.2016.01.004

[24] Jiayin Li; Meikang Qiu; Jian-Wei Niu; Yu Chen; Zhong Ming, "Adaptive resource allocation for preemptable jobs in cloud systems," Intelligent Systems Design and Applications (ISDA), IEEE 2010 10th International Conference on, vol., no., pp.31,36, Nov. 29 2010-Dec. 1 2010.

[25] Kyong Hoon Kim; Buyya, R., "Policy-based Resource Allocation in Hierarchical Virtual Organizations for Global Grids," Computer Architecture and High Performance Computing, 2006. SBAC-PAD '06. IEEE 18TH International Symposium on, vol., no., pp.36, 46, Oct. 2006.

[26] Amit Nathani, Sanjay Chaudhary, Gaurav Somani, Policy based resource allocation in IaaS cloud, Future Generation Computer Systems, Volume 28, Issue 1, January 2012, Pages 94-103, ISSN 0167-739X.

[27] Baker, T.P., "A stack-based resource allocation policy for realtime processes," Real-Time Systems Symposium, 1990. IEEE Proceedings., 11th, vol., no., pp.191,200, 5-7 Dec 1990

[28] Apostol, E.; Leordeanu, C.; Cristea, V., "Policy Based Resource Allocation in Cloud Systems," P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2011 IEEE International Conference on, vol., no., pp.285,288, 26-28 Oct. 2011.

[29] Mochizuki, K.; Kuribayashi, S.-i., "Evaluation of Optimal Resource Allocation Method for Cloud Computing Environments with Limited Electric Power Capacity," Network-Based Information Systems

(NBiS), 2011 IEEE 14th International Conference on, vol., no., pp.1,5, 7-9 Sept. 2011.
[30] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. 2012. Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing. Future Gener. Comput. Syst. 28, 5 (May 2012), 755-768. DOI=10.1016/j.future.2011.04.017.
[31] Warneke, D.; Odej Kao, "Exploiting Dynamic Resource Allocation for Efficient Parallel Data Processing in the Cloud," Parallel and Distributed Systems, IEEE Transactions on, vol.22, no.6, pp.985, 997, June 2011. doi: 10.1109/TPDS.2011.65
[32] Kim, Tai-hoon and Adeli, Hojjat and Cho, Hyun-seob and Gervasi, Osvaldo and Yau, StephenS. In addition, Kang, Byeong-Ho and Villalba, JavierGarcía, a Dynamic Resource Allocation Model for Virtual Machine Management on Cloud, Springer Berlin Heidelberg.
[33] K. Ye, X. Jiang, D. Huang, J. Chen, and B. Wang, "Live Migration of Multiple Virtual Machines with Resource Reservation in Cloud Computing Environments", in Proc. IEEE CLOUD, 2011, pp.267-274.
[34] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. 2005. Live migration of virtual machines. In Proceedings of the second conference on Symposium on Networked Systems Design & Implementation - Volume 2 (NSDI'05), Vol. 2. USENIX Association, Berkeley, CA, USA, 273-286.
[35] William Voorsluys, James Broberg, Srikumar Venugopal, and Rajkumar Buyya. 2009. Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation. In Proceedings of the 1st International Conference on Cloud Computing (CloudCom '09), Martin Gilje Jaatun, Gansen Zhao, and Chunming Rong (Eds.). Springer-Verlag, Berlin, Heidelberg, 254-265.
[36] Xiaoqiao Meng, Canturk Isci, Jeffrey Kephart, Li Zhang, Eric Bouillet, and Dimitrios Pendarakis. 2010. Efficient resource provisioning in compute clouds via VM multiplexing. In Proceedings of the seventh international conference on Autonomic computing (ICAC '10). ACM, New York, NY, USA, 11-20. DOI=10.1145/1809049.1809052.
[37] Zhen Xiao; Weijia Song; Qi Chen, "Dynamic Resource Allocation Using Virtual Machines for Cloud Computing Environment," Parallel and Distributed Systems, IEEE Transactions on , vol.24, no.6, pp.1107,1117, June 2013 doi: 10.1109/TPDS.2012.283
[38] Xingwei Wang; Jiajia Sun; Min Huang; Chuan Wu; Xueyi Wang, "A Resource Auction Based Allocation Mechanism in the Cloud Computing Environment," Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International, vol., no., pp.2111,2115, 21-25 May 2012 doi: 10.1109/IPDPSW.2012.260
[39] Advances in Computing, Communication, and Control Communications in Computer and Information Science Unnikrishnan, Srija Surve, Sunil Bhoir, Deepak R 10.1007/978-3-642-36321-4_2 T Market-Driven Continuous Double Auction Method for Service Allocation in Cloud Computing Springer Berlin Heidelberg 2013-01-01 cloud computing continuous double auction intelligent agent market-driven agents resource allocation Farajian, Nima Zamanifar, Kamran 14-24.
[40] Kuo-Chan Huang; Bo-Jyun Shen; Tsung-Ju Lee; Hsi-Ya Chang; Yuan-Hsin Tung; Pin-Zei Shih, "Resource allocation and dynamic provisioning for Service-Oriented applications in cloud environment," Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on, vol., no., pp.839,844, 3-6 Dec. 2012. doi: 10.1109/CloudCom.2012.6427592
[41] Kui Ren; Cong Wang; Qian Wang, "Security Challenges for the Public Cloud," Internet Computing, IEEE, vol.16, no.1, pp.69, 73, Jan.-Feb. 2012. doi: 10.1109/MIC.2012.14
[42] Dimitrios Zissis, Dimitrios Lekkas, Addressing cloud computing security issues, Future Generation Computer Systems, Volume 28, Issue 3, March 2012, Pages 583-592, ISSN 0167-739X, http://dx.doi.org/10.1016/j.future .2010.12.006.