Distributed and cloud computing environments are presently slanting and more prevalent for computation by various organizations like Google, Amazon, Microsoft etc. as the cloud size increase there is huge expansion in power consumption over the server farms. Also with increase in request load over a datacenter increases the request failure probability. To overcome these issues request should be scheduled in more efficient manner, improving resource utilization, request failure count and reliability of system. Recent studies show that the failure probability of a server increases linearly with the increase of independent resources (processors), and result into request failure at datacenters. So to resolve this issue various approaches have been proposed to improve the performance of cloud environment.

In this section, we have proposed a set of learning based algorithms for task allocation to minimize the request failure and to improve QoS (Quality of Service) over a data center. Proposed approaches aim to provide a global best schedule with least scheduling time complexity. Proposed algorithms has proven to have better performance in term of load and request failure rate as compared to previously proposed task allocation algorithm for cloud IaaS.
4.1. Approach 1: Fault and QoS based Genetic Algorithm for Task Allocation in Cloud Infrastructure

To overcome these issues a fault aware learning based resource allocation algorithm is been proposed using Genetic algorithm. Genetic algorithm helps us to find a solution, which cannot be, achieved my any static or dynamic algorithm. More over fault tolerant genetic algorithm help to find a fittest solution in term of least makespan (Time taken to complete a request) and least request failure probability. Proposed algorithm uses Poisson probability distribution for random request failure at virtual machine i.e. at host and datacenter level. On the other hand, request failure over a datacenter may occur randomly due to storage, network failure or VM crashes. Based on fault over a datacenter and computing capability of a system, we have proposed a task allocation policy to minimize the total makespan over the system and reduce request failure probability. According to algorithm collect the information of data center resources and capability, and the count of failure occurred over a period of time on a datacenter.

4.1.1. Proposed algorithm
Proposed FGA (Fault aware Genetic Algorithm) is divided into four phases which are as follows:

a) Initialization
b) Evaluation and selection
c) Crossover
d) Mutation

a) Initialization
In this phase we have a set of tasks \( T_1, T_2,T_3,T_4,T_5,T_6,... T_n \) and a set of resources in term of virtual machine \( \text{VM}_1,\text{VM}_2,\text{VM}_3,\text{VM}_4,\text{VM}_5,... \text{VM}_m \) are pre allocated on hosts in distributed datacenters. Here we initialize asset of sequences or schedules allocated randomly, each sequences act a chromosomes for genetic algorithm. The complete set of chromosomes is said to be a population, acting as an input for algorithm. Next population is initialized which is a
set of schedules generated randomly, by allocating tasks randomly to virtual machines available.

\textbf{b) Evaluation and selection}

In this phase we evaluate the fitness value for each schedule in a population or chromosome, which depends up on the computing capability, total time taken to complete the schedule, average utilization and the failure probability of complete schedule. Fitness value is evaluated using a fitness function defined below.

Where

\( F_i \) : Faults occurred on a system over the time \( T \)

\( FR_i \) : Fault rate that is the number of request failed due to system failure over time \( t \).

\( FP_i \) : Failure probability over a Host \( i \).

\( RE_i \) : Reliability of a Host \( i \).

\( \lambda \) : Fault rate over a time \( T \)

Since faults over a datacenter are random in nature and follows Poisson distribution, which over a period of time \( t \) and \( t + \Delta T \) can be defined as:

\[
F Pi(t \leq T \leq t + \Delta T | T > t) = \frac{\exp(-\lambda t) - \exp(\lambda(t+\Delta T))}{\exp(-\lambda t)} \quad (4.1)
\]

\[
F Pi(t) = 1 - \exp(-\lambda \Delta t) \quad (4.2)
\]

\[
RP_i = e^{-\lambda t} = e^{t/m} \quad (4.3)
\]

If

\( VM_{MIPS_i} \) : MIPS of \( i \)th virtual machine

\( T_{Leng_i} \) : Length of \( i \)th Task

Then the predicted time to complete a task \( T_i \) is defined:

\[
T_{-Exei} = \left( \frac{T_{Lenght_i}}{VM_{-MIPS_i}} \right) \quad (4.4)
\]

\[
Total \_time = \sum_{i=1}^{n} \frac{T_{-Length_i}}{VM_{-MIPS_i}} \quad (4.5)
\]
The fitness value for a chromosome is defined by the fitness function gives as:

\[ \text{Fitness}_{\text{chromosome}_i} = \alpha(Total\_\text{time}_{i}) + \beta(FP_{i}) \]  \hspace{1cm} (4.6)

Where

\[ \alpha + \beta = 1 \]  \hspace{1cm} (4.7)

Based on the fitness value of chromosome the fittest one is selected having least fitness value. The population is sorted based on the fitness value and best two are selected for next phase.

c) **Crossover**

In this step two fittest solutions based on least make span and failure probability is selected. We have used multi point crossover to generate new fittest schedule/ chromosome. This module is responsible for generation of new schedule/ chromosome by combining two selected having least fitness value and interchange two or more scheduled tasks between selected fittest schedules. The new generated schedule is added to the existing population.

Steps to generate crossover are as follows.
1. The two fittest chromosomes are selected
2. A new fittest chromosome is generated using multi point cross over by interchanging the set of schedules between two chromosomes.
3. The new chromosome replaces the chromosome with highest fitness value.

d) **Mutation**

In this phase new merging the new offspring, and modifying the existing chromosomes with new solution. This forms a new set of schedules and population which form a better solution after each iteration. After specific count of iteration predefined as an input to genetic algorithm, best chromosome is selected i.e. the chromosome with least fitness value is selected for schedule.
**Proposed algorithm**

**Fault Based Genetic Algorithm Task Allocation**

Algorithm: FGATA(VM List VM, Task list T, population size Po, Iteration Itr)

//Input : Po, VM, Itr and T

1. VM <- VM_List() ;
2. Fl <- getFault();
3. i <- No. of VM
4. Ti <- Task_List();
5. C <- Genetic_algo (Vmi, Ti, Po, Itr);
6. Allocate_Resource( C); // processing the client request.

Figure 4.1 Proposed FGS algorithm Initialization

**Genetic Algorithm**

Genetic _algo (VMi, Ti, Po, Itr)

//Input : Po, VM, Itr and Ti

1. Po <- Initiate_Population( Ti, );
2. Evaluation();
3. C1 <- getFittest1();
4. C2 <- getFittest2();
5. Crossover(C1,C2)
6. Mutation(Po , C1, C2);
7. Return( getFittest());

6. End

Figure 4.2 Proposed fault aware genetic algorithm.

1. Evaluation(){
2. For each Ci, i=0 - po
3. For each Ti
4. temp= α (T/Vmi) + β (FP)
5. Fitnessi = Fitnessi + temp
6. End
7. END
8. }

Figure 4.3 Proposed FGA evaluation phase
Proposed algorithm provides a benefit over existing static scheduling algorithm, that it can search for best global solution rather than assuming the local best solution as the best solution. Moreover, the proposed algorithm takes into consideration the faulty behavior of cloud, which helps in find a solution with similar high utilization and least failure probability.

![Flow diagram for Proposed FGA allocation phase](image)

**Fitness chromosome**

\[ Fitness_{chromosome} = \alpha(Total\_time) + \beta(FPI) \]

![Flow diagram for Proposed FGA flow diagram](image)
4.1.2. Experiment and Results

For simulation CloudSim 3.0 API is used. CloudSim 3.0 [72] provides linear power model simulation to find the power consumption in cloud. Proposed fault aware genetic task allocation algorithm is implemented in CloudSim replacing existing task scheduling to find the global best schedule. Proposed algorithm is being tested over various test cases with 10 servers D0-D9 and Poisson distribution model for random request and fault model in distributed environment.

Testing of proposed algorithm is done with basic Genetic algorithm proposed by Suraj, S. Rin [62]. Testing is done for 1000, 1500, 2000, 2500, 3000, 3500 requests with population size been 100, 200, 300, 400. Iteration for simulation of each simulation is 100. Results are shown in figures below. Table 4.1 shows the environment specification and parameters used for simulation.

<table>
<thead>
<tr>
<th>Server</th>
<th>RAM (Mb)</th>
<th>MIPS</th>
<th>Storage (Gb)</th>
<th>Core</th>
<th>PE</th>
<th>HOST</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0</td>
<td>2000</td>
<td>10000</td>
<td>100000</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>D1</td>
<td>2000</td>
<td>10000</td>
<td>100000</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>D2</td>
<td>2000</td>
<td>10000</td>
<td>100000</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>D3</td>
<td>2000</td>
<td>10000</td>
<td>100000</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>D4</td>
<td>2000</td>
<td>10000</td>
<td>100000</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>D5</td>
<td>2000</td>
<td>10000</td>
<td>100000</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>D6</td>
<td>2000</td>
<td>10000</td>
<td>100000</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>D7</td>
<td>2000</td>
<td>10000</td>
<td>100000</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>D8</td>
<td>2000</td>
<td>10000</td>
<td>100000</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>D9</td>
<td>2000</td>
<td>10000</td>
<td>100000</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>
Figure 4.6 Comparison of improvement in request completed

Figure 4.7 Comparison of improvement in request failed

Figure 4.6 & 4.7 compares the improvement in number of request failed and request competed with increase in number of requests over the system. The failure count reduces over the proposed system in increase in completed requests over the system.
Figure 4.8 Comparison of failure probability with variable resources

Figure 4.9 Comparison of failure probability with variable request
Figure 4.8 discourses the improvement in failure probability with increase in number of resources since with increase in number of VM’s the probability of failure increases over the system. Figure 4.9 shows the improvement in failure probability with increase number of request count. Figure 4.10 & 4.11 shown the increase in reliability with increase number of VM’s and request counts. Figure 4.11 shows improvement in reliability as the number of requests increases. Figure 4.12 shows the drawback of proposed algorithm with small increase in complete execution time as the number of request completed increases.

From experimental result section, it is clear that proposed fault aware GA (Genetic Algorithm) provides better QoS (Quality of service) as compared to previous proposed GA algorithm. The main idea of this algorithm in cloud computing is to complete maximum number of requests with least failure probability, proposed algorithm shown that it can maximize reliability and minimize the number of request failed. This strategy has proven that it provides better QoS in term of high reliability with increase in number of requests and resources with failure probability.
Figure 4.11 Comparison of reliability with variable requests

Figure 4.12 Comparison of execution time with variable resources
4.2. Approach 2: Task Allocation Using Big Bang-Big Crunch in Cloud Infrastructure

In this approach, we have proposed a task allocation algorithm based on Big Bang-Big Crunch (BBC) algorithm. This algorithm is motivated from the physics behind creation of universe theory in astrology. BBC algorithms refers to the evolution of universe and end of universe which says universe is a finite space which once expanded with force binding it and will end into a single point referred as a black hole. Algorithm also suggests that any element of universe cannot be suggested as center of universe. Similar to this we have proposed a task allocation algorithm to find a single best solution from a large set of solutions. Where generation of universe is referred as Big Bang phase and dissipation of universe in black hole near the center is said to big crunch phase.

Proposed algorithm uses Poisson probability distribution for random request at virtual machine i.e. at host and datacenter level. Based on computing capability of a system, we have proposed a task allocation policy to minimize the total makespan over the system and reduce time complexity of solution. According to algorithm collect the information of data center resources and its capability. Proposed algorithm is similar to Genetic algorithm (GA) but the problem size reduces after each phase and will give you a single point solution i.e. the global solution. But in existing GA the population size remain same and there is no guarantee that the global best is achieved.

4.2.1. Proposed Algorithm

Existing proposed algorithms are either discus about task scheduling or resource utilization and some of them talk about task or VM migrating to fulfill requests but the existing algorithms are static or dynamic in nature and they may suffer from local minima solution considering that as the best solution. But a better solution for task allocation may be possible. So to overcome these learning based algorithms were proposed like Genetic algorithm and PSO (particle swarm optimization). The issue with these algorithms is that they have very high time complexity more over they depend upon the iteration and the initial; population size, which affects their solution. If the population size of the iterations/generation are less then there less probability to get best solution.
Proposed algorithm is divided into four phases which are as follows:

a) Big Bang / Initialization phase

b) Evaluation phase

c) Crossover / Center of mass

d) Big Crunch phase

a) **Initialization**

In this phase we have a set of tasks \(T_1, T_2, T_3, T_4, T_5, T_6, \ldots, T_n\) and a set of resources in terms of virtual machine \(VM_1, VM_2, VM_3, VM_4, VM_5, \ldots, VM_m\) are pre-allocated on hosts in distributed datacenters. Here we initialize a set of sequences or schedules allocated randomly, each sequence acts as a chromosome for the genetic algorithm. The complete set of chromosomes is said to be a population, acting as an input for the algorithm. Next population is initialized which is a set of schedules generated randomly, by allocating tasks randomly to virtual machines available.

b) **Evaluation and selection**

In this phase we evaluate the fitness value for each set of sequence or chromosome, which depends upon the computing capability, total time taken to complete the schedule. Fitness value is evaluated using a fitness function defined below.

Where

- \(VM_{MIPS}_i\): MIPS of \(i^{th}\) virtual machine
- \(T_{Leng}_i\): Length of \(i^{th}\) Task
- \(Fitness_{chomosome}_i\): Fitness value of chromosome/sequence \(i\)

Then the predicted time to complete a task \(T_i\) is defined:

\[
T_{Exei} = \frac{T_{Leng}_i}{VM_{MIPS}_i}
\]  

(4.8)

\[
Total\_time = \sum_{i=1}^{n} \frac{T_{Leng}_i}{VM_{MIPS}_i}
\]  

(4.9)
The fitness value for a chromosome is defined by the fitness function given as:

\[
Fitness\_\text{chromosome}_i = \alpha (Total\_time_i)
\]  
(4.10)

Based on the fitness value of chromosome the fittest one is selected having least fitness value. The population is sorted based on the fitness value and best two are selected from next phase.

c) Crossover

In this step two fittest solutions based on least make span are selected based on the center of mass and the population sequence near to center of mass are selected for cross over.

The steps for selection are as follows:
1. Find Center of mass of fitness values of the sequences in population using mean.
2. Find the sequence having fitness value with least difference from the center of mass.

We have used multi point crossover to generate new fittest sequences/chromosome. Steps to generate crossover are as follows.
4. The two fittest chromosomes are selected with least difference from center of mass and one having least fitness value.
5. A new fittest chromosome is generated using multi point cross over by interchanging the set of schedules between two chromosomes.
6. The new chromosome replaces the chromosome with highest fitness value.

\[
C\_\text{Mass} = \frac{\sum_{i=0}^{n} Fitness\_\text{chromosome}_i}{Population\_Size()}
\]  
(4.11)

d) Big Crunch phase

In this phase new merging the new offspring, which can be a better solution from all other chromosomes/sequences. A new population is generated with new offspring generated and removing two chromosomes with least fitness value i.e. the worst solution from the population, decreasing the population size by one. These steps are repeated for number of iterations. After specific count of iteration of proposed algorithm stop the iterations, when the
population size is one. This is said to be the stopping condition of BBC and the last solution is the best solution for a definite time interval and iteration. Each iteration can also be referred as “generation” to create new fittest solution.

Proposed algorithm

**Big Bang-Big Crunch Algorithm Task Allocation**

Algorithm: BBC (VM List VM; Task list T, population size Po, Iteration Itr)

//Input: Po, VM; Itr and T
1. VMᵢ ← VM_List();
2. i ← No. of VM
3. Ti ← Task_List();
4. C ← BBC_algo(VMi,Ti, Po, Itr);
5. Allocate_Resource(C); // processing the client request.
6. End

Figure 4.13 Proposed BBC algorithm initialization

**BBC Algorithm**

BBC_algo(VMi,Ti, Po, Itr)

//Input: Po, VMᵢ, Itr and Ti
1. Po ← Initiate_Population(Ti);
2. Evaluation();
3. CenterMass(); // find mean of all fitness values
4. C₁ ← getFittest1();
5. C₂ ← getFittest2();
6. Po ← Crossover(C₁, C₂)
7. Big_Crunch(Po, C₁, C₂);
8. Return(getFittest());
9. End

Figure 4.14 Proposed BBC Algorithm
Evaluation
1. Evaluation()
2. For each $C_i$, $i=0 - po$
3. For each $T_i$
4. $\text{temp} = (T_i/Vm_i)$
5. $\text{Fitness}_i = \text{Fitness}_i + \text{temp}$
6. End
7. END
8. }

Figure 4.15 Proposed Evaluation Phase

Big_Crunch
1. Big_Crunch(Po, C1, C2)
2. {
3. Delete_worst(); // delete element with least fitness value.
4. Delete_worst();
5. }

Figure 4.16 Big Crunch Phase

Allocate_Resource(C)
1. $C_i \leftarrow \text{Getchromosomes}()$
2. For each $C_i$
3. $\text{Allocate}(C_i)$
4. END
5. }

Figure 4.17 Allocation Phase

Proposed algorithm provides a benefit over existing static scheduling algorithm, that it can search for best global solution rather than assuming the local best solution as the best solution. Moreover, the proposed algorithm takes into consideration the faulty behavior of cloud, which helps in find a solution with similar high utilization and less time complexity as compared to Genetic algorithm. Figure 4.18 shows the flow diagram for proposed algorithm and interaction between each module.
4.2.2. Experiment and Results

Simulation has been performed on a simulation test bead using CloudSim 3.0 [72] tool kit for cloud simulation. CloudSim provides a cloud infrastructure environment with all environmental parameter to study the performance of cloud. Proposed Big Bang Big Crunch algorithm for task allocation is implemented in CloudSim replacing existing Round Robin algorithm. The algorithm aims to reduce the scheduling time and find an global best schedule with least make span. Proposed algorithm is being tested over various test cases with 10 servers D0-D9 and Poisson distribution model for random request in distributed environment, with each server having two hosts each.

Testing of proposed algorithm is done with basic Genetic algorithm proposed by Suraj, S. Rin [62]. Testing is done for 1000, 1500, 2000, 2500, 3000, 3500 requests with population size been 100, 200, 300, 400. Iteration for simulation of each simulation is 100. Results are shown
in figures below. Table 4.2 shows the environment specification and parameters used for simulation.

Table 4.2: Experimental parameters used for simulation environment

<table>
<thead>
<tr>
<th>Server</th>
<th>RAM (Mb)</th>
<th>MIPS</th>
<th>Storage (Gb)</th>
<th>Core</th>
<th>PE</th>
<th>HOST</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0</td>
<td>2000</td>
<td>10000</td>
<td>100000</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>D1</td>
<td>2000</td>
<td>10000</td>
<td>100000</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>D2</td>
<td>2000</td>
<td>10000</td>
<td>100000</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>D3</td>
<td>2000</td>
<td>10000</td>
<td>100000</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>D4</td>
<td>2000</td>
<td>10000</td>
<td>100000</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>D5</td>
<td>2000</td>
<td>10000</td>
<td>100000</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>D6</td>
<td>2000</td>
<td>10000</td>
<td>100000</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>D7</td>
<td>2000</td>
<td>10000</td>
<td>100000</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>D8</td>
<td>2000</td>
<td>10000</td>
<td>100000</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>D9</td>
<td>2000</td>
<td>10000</td>
<td>100000</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure 4.19 Comparison of improvement in execution time
Figure 4.20 Comparison of improvement in execution time with changes in population size.

Figure 4.19 compares the improvement in execution time with increase in number of requests over the system. Figure 4.20 shows the improvement in execution time with increasing population size. Execution time has reduced over the proposed system with increase in completed request count over the system.

Figure 4.21 Comparison of execution time of individual requests.

For 1000 request count
Figure 4.22 Comparison of execution time of individual requests.  
For 1500 request count

Figure 4.23 Comparison of execution time of individual requests.  
For 2000 request count
Figure 4.24 Comparison of execution time of individual requests.
For 2500 request count

Figure 4.25 Comparison of execution time of individual requests.
For 3000 request count
Figure 4.26 Comparison of execution time of individual requests.

For 3500 request count

Figure 4.21, 4.22, 4.23, 4.24, 4.25 & 4.26 shows the improvement in distribution of execution time for requests using proposed BBC algorithm for task allocation over cloud. The execution time of the requests has improved and majority of requests are completed in small execution time as compared to genetic algorithm.

Figure 4.27 Comparison of Average Start time of system with increase in request count
Figure 4.27 discourses the improvement in average start time with increase in number of request over the system, which shows the proposed algorithm proves to provide better start time than the conventional genetic algorithm. Figure 4.28 discourses the improvement in average finish time, which reduces with increasing requests over the system. The experiment has been performed over 1000, 1500, 2000, 2500, 3000, and 3500 request count. Proposed algorithm proves to provide reduced finish time as compared to existing genetic algorithm.

From experimental result section, it is clear that proposed BB-BC provides better QoS (Quality of service) as compared to previous proposed GA algorithm. The main idea of this algorithm in cloud computing is to complete maximum number of requests with least execution time, proposed algorithm shown that it can provide better execution time over large requests with reduces average start time and average finish time over the system. Proposed algorithm reduces the number of iteration required to achieve a global best solution with least scheduling time. This strategy has proven that it provides better QoS in term of high reliability with increase in number of requests and resources with least scheduling time with decrease in execution time with increase in population size and number of requests. Proposed algorithm insures the schedule achieve is global best solution.
4.3. Approach 3: Fault Tolerant Big Bang-Big Crunch for Task Allocation in Cloud Infrastructure

In this approach, we have proposed a fault aware Big Bang-Big Crunch (BBC) algorithm for task allocation in cloud infrastructure. The algorithm is motivated from Big Bang-Big Crunch (BBC) theory of creation of universe in astrology. BBC algorithm is similar to the algorithm as proposed in previous approach. Similar to this we have proposed a task allocation algorithm to find a single best solution from a large set of solutions but in a faulty cloud environment. Where generation of universe is referred as Big Bang phase and dissipation of universe in black hole near the center is said to big crunch phase. Proposed algorithm aim to improve the performance of task allocation algorithm and reduce the request failure count. Proposed algorithm improves the reliability of system and finds the global schedule for tasks.

4.3.1. Proposed Algorithm

Existing algorithm and above proposed approached either take into consideration improvement in scheduling delay or fault tolerant behavior of cloud. Some other existing approaches take into consideration VM migration of better task management, cost improvement and power efficiency, which are static or dynamic in nature. These algorithm suffers from either cannot find global best solution or request failure in cloud. Proposed algorithm tries to improve both the parameters together. Algorithm is been tested with variable iterations and population sizes with different cloud infrastructures.

Proposed algorithm is divided into four phases which are as follows:

a) Big Bang / Initialization phase
b) Evaluation phase
c) Crossover / Center of mass
d) Big Crunch phase
a) **Initialization**

In this phase we have a set of tasks (T1, T2, T3, T4, T5, T6…, Tn) and a set of resources in term of virtual machine (VM1, VM2, VM3, VM4, VM5…, VMm) are pre-allocated on hosts in distributed datacenters. Here we initialize asset of sequences or schedules allocated randomly, each sequence act a chromosome for genetic algorithm. The complete set of chromosomes is said to be a population, acting as an input for the algorithm. Next population is initialized which is a set of schedules generated randomly, by allocating tasks randomly to virtual machines available.

b) **Evaluation and selection**

In this phase we evaluate the fitness value for each set of schedule or chromosome, which depends on the computing capability, total time taken to complete the schedule and the failure probability of the schedule.

Where

- Fi : Faults occurred on a system over the time T
- FRi : Fault rate that is the number of request failed due to system failure over time t.
- FPi : Failure probability over a Host i.
- REi : Reliability of a Host i.
- \( \lambda \) : Fault rate over a time T

Since faults over a datacenter are random in nature and follows Poisson distribution, which over a period of time \( t \) and \( t + \Delta T \) can be defined as:

\[
F Pi(t \leq T \leq t + \Delta T | T > t) = \frac{\exp(-\lambda t) - \exp(\lambda(t+\Delta T))}{\exp(-\lambda t)}
\]  
(4.12)

\[
F Pi(t) = 1 - \exp(-\lambda \Delta t)
\]  
(4.13)

\[
RPi = e^{-\lambda t} = e^{t/m}
\]  
(4.14)

Equation 4.12 shows the fault over a time T and \( \Delta T \) using Poisson probability distribution. Equation 4.14 represents the evaluation of reliability for a system.
If
VM_MIPS i: MIPS of ith virtual machine
T_Lengi: Length of ith Task
Fitness_chromosome i : Fitness value of chromosome/sequence i
Then the predicted time to complete a task Ti is defined:
\[ T_{\text{Exe}} = \frac{T_{\text{Length}} i}{VM_{\text{MIPS}} i} \] (4.15)

\[ \text{Total time} = \sum_{i=1}^{n} \frac{T_{\text{Length}} i}{VM_{\text{MIPS}} i} \] (4.16)

The fitness value for a chromosome is defined by the fitness function gives as:
\[ \text{Fitness}_{\text{chromosome}} i = \alpha(\text{Total time}) + \beta(FPi) \] (4.17)
\[ \alpha + \beta = 1 \] (4.18)

Based on the fitness value of chromosome the fittest one is selected having least fitness value. The population is sorted based on the fitness value and best two are selected from next phase.

c) Crossover
In this step two fittest solutions based on least fitness value are selected based on the center of mass and the population sequence near to center of mass are selected for cross over. The steps for selection are as follows:
1. Find Center of mass from the sequences in population using mean.
2. Find the sequence having fitness value with least difference from the center of mass.
3. The selected sequence is used for generation of next fit element. The selected sequence be S1.
4. Select a second best sequence having least fitness value. The selected sequence be S2.

We have used multi point crossover to generate new fittest sequences/ chromosome. Steps to generate new fittest sequence using crossover are as follows.
1. A new fittest chromosome is generated using multi point cross over by interchanging the set of schedules between two chromosomes.

2. The new chromosome replaces the chromosome with highest fitness value chromosome.

\[
C_{\text{Mass}} = \frac{\sum_{i=1}^{n} \text{Fitness}_i \; \text{chromosome}}{\text{PopulationSize}()}
\]

(4.19)

These steps help to find the global best solution as in each iteration the solution moves the mean toward the best solution by using crossover and generation new best solution.

d) Big Crunch phase

In this phase the new offspring generated by merging the two best solutions, can be better solution than all existing chromosomes/sequences. A new population is generated with new offspring generated in previous step and removing the chromosomes with highest fitness value i.e. the worst solution from the population, decreasing the population size by one. Repeat steps b, c & d and stop the iterations, when the population size is one or the integration count is zero. This is said to be the stopping condition of BBC and the last solution is the best solution for a definite time interval and iteration. Each iteration can also be referred as “generation” to create new fittest solution.

Proposed algorithm

Fault Big Bang-Big Crunch Algorithm Task Allocation

Algorithm:-BBC (VM List VM, Task list T, population size Po , Iteration ltr)
//Input : Po, VM, ltr and T
1. VM = VM_List();
2. i = No. of VM
3. T = Task_List();
4. C = BBC_algo(Vmi, Ti, Po, ltr);
5. Allocate_Resource(C); // processing the client request.
6. End

Figure 4.29 Proposed FBBC algorithm initialization
**BBC Algorithm**

```c
BBC_algo(VMi, Ti, Po, Iteration)
//input: Po, VMi, itr and Ti
1. Po ← Initiate_Population(Ti);
2. While (Iteration > 0)
3. { Evaluation_fitness();
4.   CenterMass(); // find mean of all fitness values
5.   C1 ← getFittest1();
6.   C2 ← getFittest2();
7.   C3 ← Crossover(C1, C2)
8.   Big_Crunch(C3);
9.   iteration ← ;
10. }
11. Return(getFittest());
12. End
```

Figure 4.30 Proposed FBBC algorithm

**Evaluation**

1. Evaluation_fitness(){
2.   For each Ci i=0 to po // For each population
3.     Fitness = Make_span() + TotalFault();
4.   End
5. END
6. }

Figure 4.31 Proposed FBBC evaluation phase

**Fitness**

1. getFittest1()
2. {
3.   fitness_mean=0;
4.   mass_diff=0;
5.   mass_diff_t=0;
6.   // Loop through individuals to find fittest
7.   for (int i = 0; i < populationSize(); i++)
8.   {
9.     fitness_mean= fitness_mean+ tours[i].getFitness();
10. }
11. fitness_mean=fitness_mean/populationSize();
12. mass_diff=tours[0].getFitness()-fitness_mean;
13. for (int i = 0; i < populationSize(); i++)
14. {
15.   mass_diff_t=fitness_mean-tours[i].getFitness();
16.   if (mass_diff >= mass_diff_t)
17.     {
18.       fittest = getTour(i);
19.       mass_diff=mass_diff_t;
20.     }
21. }
22. return fittest;
23. }

Figure 4.32 Get Fittest with least difference from Center of mass
Proposed algorithm provides a benefit over existing static scheduling algorithm, that it can search for best global solution rather than assuming the local best solution as the best solution. Moreover, the proposed algorithm takes into consideration the faulty behavior of cloud, which helps in find a solution with similar high utilization, least failure probability, high reliability and less time complexity as compared to Genetic algorithm. Figure 4.36 shows the flow of algorithm and interaction among various phases of task allocation.
4.3.2. Experiment and Results

Simulation has been performed on a simulation test bead using CloudSim 3.0 [72] tool kit for cloud simulation. CloudSim provides a cloud infrastructure environment with all environmental parameter to study the performance of cloud. Proposed fault aware Big Bang Big Crunch algorithm for task allocation is implemented in CloudSim replacing existing Round Robin algorithm. The algorithm aims to reduce the scheduling time and find an global best schedule with least make span. Proposed algorithm is being tested over various test cases with 10 servers D0-D9 and Poisson distribution model for random request in distributed environment, with each server having two hosts each.

Testing of proposed algorithm is done with basic Genetic algorithm proposed by Suraj, S. Rin [62]. Testing is done for 1000, 1500, 2000, 2500, 3000, 3500 requests with population size been 100, 200, 300, 400. Iteration for simulation of each simulation is 100. Results are shown.
in figures below. Table 4.3 shows the environment specification and parameters used for simulation.

### Table 4.3
Experimental parameters used for simulation environment

<table>
<thead>
<tr>
<th>Server</th>
<th>RAM (Mb)</th>
<th>MIPS</th>
<th>Storage (Gb)</th>
<th>Core</th>
<th>PE</th>
<th>HOST</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0</td>
<td>2000</td>
<td>10000</td>
<td>100000</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>D1</td>
<td>2000</td>
<td>10000</td>
<td>100000</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>D2</td>
<td>2000</td>
<td>10000</td>
<td>100000</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>D3</td>
<td>2000</td>
<td>10000</td>
<td>100000</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>D4</td>
<td>2000</td>
<td>10000</td>
<td>100000</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>D5</td>
<td>2000</td>
<td>10000</td>
<td>100000</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>D6</td>
<td>2000</td>
<td>10000</td>
<td>100000</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>D7</td>
<td>2000</td>
<td>10000</td>
<td>100000</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>D8</td>
<td>2000</td>
<td>10000</td>
<td>100000</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>D9</td>
<td>2000</td>
<td>10000</td>
<td>100000</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure 4.37 Comparison of improvement in scheduling time

Figure 4.37 shows the improvement in time takes to find a best schedule to allocate resources. In this figure both algorithms are learning based algorithm but proposed BBC algorithm proves to have less scheduling time. Figure 4.38 & 4.39 compares the improvement in
number of request failed and request competed with increase in number of requests over the system. The failure count reduces over the proposed system with increasing request count and proposed algorithm also shows improvement completed request count over the system.

![Comparison of improvement in request failed](image1.png)

Figure 4.38 Comparison of improvement in request failed

![Comparison of improvement in request completed](image2.png)

Figure 4.39 Comparison of improvement in request completed

Figure 4.40 discourses the improvement in failure probability with increase in number of resources, since with increase in number of request the probability of failure increases over
the system. Figure 4.41 shows the improvement in reliability with increase number of request count over a system.

Figure 4.40 Comparison of failure probability with variable request count

Figure 4.41 Comparison of reliability with variable request count

Figure 4.42 shows the drawback of proposed algorithm with small increase in total execution time. Overall result shows that the proposed algorithm improves the fault tolerant behavior of
system by reducing the request failure count of the system and improving the reliability of the system.

Figure 4.42 Comparison of execution time with variable request count.

From experimental result section, it is clear that proposed fault aware BBC provides better QoS (Quality of service) as compared to previous proposed GA algorithm. The main idea of this algorithm in cloud computing is to complete maximum number of requests with least failure probability, proposed algorithm shown that it can maximize reliability and minimize the number of request failed. This strategy has proven that it provides better QoS in term of high reliability with increase in number of requests and resources with failure probability.
4.4. Approach 4: Load and Fault Aware Honey Bee Scheduling Algorithm for Cloud Infrastructure

There exist many load balancing algorithms proposed for grid and distributed computing environments [30]. But they do not take into consideration cloud as non-faulty and QoS of datacenter. There are many cloud IaaS frameworks that provide cloud computing services and virtualization services to the user like OpenNode [73], CloudStack [74], Eucalyptus [75], CloudSigma [76], EMOTIVE (Elastic Management of Tasks in Virtualized Environments) [77] and Archipel.[78]

There are many solutions been proposed over the time based on priority, cost, rank based which is used in OpenNebula [79] and round robin and power aware scheduling algorithm used in Eucalyptus and many more. But they do not take in to consideration the QoS parameters of the datacenters like fault rate, initialization time, MIPS and many more. So to overcome this issue and make system more reliable, fault and load aware honey bee scheduling algorithm is proposed.

4.4.1 Proposed Algorithm

Proposed algorithm is inspired for natural behavior of honey bee to find the best solution for designing optimal scheduling algorithm. The algorithm requires a number of parameters to be set, specifically: quantity of scout bees (n), quantity of nice websites out of m selected sites (e), number of websites selected out of n visited sites (m), number of bees recruited for high-quality e web sites, number of bees recruited for the opposite (m-e) selected sites, initial size of patches which includes site and its neighborhood and stopping criterion.

Steps of proposed algorithm are as follows:
Step1. Initialize scout bees equal to number of datacenters.
Step2. Recruit scout bees for selected sites (more bees for best e sites) and evaluate fitness value for datacenter.
Step4. Assign bees to search randomly and evaluate their fitnesses for a request.
Step4. Stop when all bees have arrived, else wait.
Step5. Select the fittest bee from each datacenter.
Step 6. Assign remaining bees to search randomly and evaluate their fitnesses for each request.

Step 7. End While no request in queue.

In first step, the bee algorithm starts with the scout bees (n) being placed randomly in the search space. In step 2, the algorithm conducts searches in the neighborhood of the selected sites, assigning more bees to search near to the best ‘e’ sites i.e. search for new datacenters. In step 3 the fitnesses of the datacenters visited by the scout bees are evaluated. In step 4 waiting until all bees are arrived. In step 5, 6 bees that have the highest fitnesses are chosen as “selected bees” and sites visited by them are chosen for allocation of resources. In step 7 repeat all above steps until there is request in queue.

Most complicated part of this algorithm is fitness value calculation. Proposed algorithm take into consideration parameters of datacenter which are used for calculating fitness value for a datacenter are as follows.

a) Initiation Time: How long it takes to deploy a VM.
 b) System load: Number of busy or allocated Machine Instruction per Second (MIPS) of a datacenter.
 c) Network load: Allocated network bandwidth out of total available bandwidth provided.
 d) Fault Rate: It is defined as the number of faults over a period of time.

In above mentioned parameters allocated MIPS (MP) and Bandwidth of a datacenter changes as the number of virtual machine allocated on a datacenter changes, but fault rate, initialization time that is the time taken to allocate resource at datacenter also increases as the load increases. Fitness (FT), allocated MIPS (MP), Fault rate (FR), Initialization time (IT), Network load (N_L).

\[
FT = \alpha_1 \frac{1}{N_L} + \alpha_2 \frac{1}{FR} + \alpha_3 \frac{1}{MP} \tag{4.20}
\]

\[
\alpha_1 < 1, \; \alpha_2 < 1 \; \& \; \alpha_3 < 1 \tag{4.21}
\]

\[
\alpha_1 + \alpha_2 + \alpha_3 = 1 \tag{4.22}
\]
Fault rate (FR) is:

\[ FR(t) = f(\text{MP}, \text{N}_L) \]  \hspace{1cm} (4.23)

Where FR(t) is number of faults over the time t, which is function of system and network load over the time t. \(\alpha_1\), \(\alpha_2\) and \(\alpha_3\) are constant which represents the ratio of parameters contribution to fitness value. Figure 4.43 is the pseudo code for proposed algorithm with all its steps.

```
Algorithm: Honey bee allocation
1. Honey_bee_allocation(Req_list r)
   Input: Requests list r
2. Initialize scout bees equal to number of datacenters.
3. Recruit scout bees for selected sites (more bees for better sites)
4. Evaluate fitness value for datacenter.
5. Assign scout bees to search randomly and evaluate their fitness for a request.
6. Stop when all bees have arrived, else wait.
7. Select the fittest bee
8. Assign remaining bees to search randomly and evaluate their fitness for each request.
9. End While no request in queue.
Output: All request been scheduled.
```

Figure 4.43 Proposed fault aware honey bee algorithm

4.4.2. Experiment and Result

Proposed fault aware honey bee algorithm is simulated using CloudSim 2.0 simulator [24]. CloudSim originally support Round robin, cost based algorithm, and FIFO algorithm for scheduling the resource sequentially. Originally CloudSim 2.0 API does not support faults in cloud environment. So firstly occurrence of fault is added as a parameter for datacenter which responds to failure probability of the datacenter.

This CloudSim API is used to set up a cloud infrastructure environment for simulation. So that environment includes all cloud IaaS request functions and environmental parameters, host and datacenter parameters. Proposed algorithm is implemented in cloudsim changing existing algorithm to study and improve the performance. Comparative study is made between basic load aware honey bee (BLHB) and proposed fault based load aware honey bee algorithm (FLBH). Figure 4.44 shows the improvement in number of requests failed over a
system with increasing request counts. Where we have considered 3 servers with 2 host each. Table 4.4 shows the failure rate of respective server.

<table>
<thead>
<tr>
<th>Server Name</th>
<th>Fault rate</th>
<th>FR(t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server1</td>
<td>0.143</td>
<td></td>
</tr>
<tr>
<td>Server2</td>
<td>0.125</td>
<td></td>
</tr>
<tr>
<td>Server3</td>
<td>0.5</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.44 Comparison of request failure count.

Figure 4.45 Comparison of request completed count.
Fig 4.44 shows the number of request failed using proposed and basic honey bee algorithm. Experiment shows that proposed algorithm have less number of request failures as compared to basic honey bee algorithm.

Fig 4.45 shows the number of request completed using proposed and basic honey bee algorithm, proposed algorithm proves to improve the request completion count as compared to existing algorithm. This graph shows the algorithm when tested with 60, 100, 200, 300, 400 requests. So the result shows the improvement of proposed algorithm over BLHB in fault aware environment.

<table>
<thead>
<tr>
<th>Request count</th>
<th>60</th>
<th>100</th>
<th>200</th>
<th>300</th>
<th>400</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLBH</td>
<td>8</td>
<td>15</td>
<td>28</td>
<td>48</td>
<td>68</td>
</tr>
<tr>
<td>BLHB</td>
<td>13</td>
<td>23</td>
<td>43</td>
<td>71</td>
<td>89</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Request count</th>
<th>60</th>
<th>100</th>
<th>200</th>
<th>300</th>
<th>400</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLBH</td>
<td>52</td>
<td>85</td>
<td>172</td>
<td>252</td>
<td>332</td>
</tr>
<tr>
<td>BLHB</td>
<td>47</td>
<td>77</td>
<td>157</td>
<td>229</td>
<td>311</td>
</tr>
</tbody>
</table>

### 4.5. Comparative Analysis of Learning Based Algorithms

In this section we have performed a comparative study over all proposed learning based algorithm. The study is performed over various parameters like scheduling time, failed request count and request completed count using exiting genetic algorithm (GA) and proposed fault aware genetic algorithm, Big bang big crunch algorithm and fault aware big bang big crunch algorithm.
Figure 4.46 Comparison of scheduling delay

Figure 4.47 Comparison of failed request count

Figure 4.46 shows the comparison of scheduling delay by all four stated algorithms. Figure shows that Genetic algorithm, Fault aware GA takes same scheduling delay where as BBC and fault aware BBC takes almost same delay. BBC proves to be better in term of having least scheduling delay. Figure 4.47 compares the request failure count using all four algorithms with varying request count. Comparison shows the order of improvement in proposed algorithms which shows Fault aware BBC(FBBC) as best having least request
failures and the in the list is BBC, the is Fault aware GA(FGA). Existing GA proves to perform worst with highest request failure count.

Figure 4.48 compares the performance in term of request completion count over all stated algorithms. FBBC proves to perform best with highest request completion count and at second number BBC proves to have better performance then FGA algorithm. GA algorithm proves to perform worst with least number of requests completed.

4.6. Conclusion

The main achievement of this work is to find the rich literature and solve the issue of resource allocation in fault aware cloud environment. The results obtained with our approach were very competitive with most of the well known algorithms in the literature and justified over the large collection of requests. Proposed resource allocation algorithm proves to provide better fault tolerance as compared to existing algorithm with least request failure, reduced average utilization, Scheduling delay, high request completion count, Failure probability and improved reliability of system.