Additional Files

Appendix A. Stability condition for Lotka-Volterra Model:

$$\frac{dP}{dt} = \alpha P - \beta PQ \tag{A.1}$$

$$\frac{dQ}{dt} = \delta PQ - \gamma Q \tag{A.2}$$

The stability of this model is attained, when there is no growth rate for both P, Q.

$$\begin{aligned} &\alpha P - \beta PQ = 0 \\ \Rightarrow\ &P(\alpha - \beta Q) = 0 \end{aligned}$$

Either P=0 or $\alpha - \beta Q$=0. P=0 is trivial. Hence

$$\begin{aligned} &\alpha - \beta Q = 0 \\ \Rightarrow\ &\alpha = \beta Q; \quad \frac{\alpha}{\beta} = Q \end{aligned}$$

if $\beta = 1$, than $\alpha = Q$
Similarly,

$$\begin{aligned} &\delta PQ - \gamma Q = 0 \\ \Rightarrow\ &Q(\delta P - \gamma) = 0 \\ \Rightarrow\ &\gamma = \delta P; \quad \frac{\gamma}{\delta} = P \end{aligned}$$

if $\delta = 1$, $\gamma = P$. Therefore, stability of the proposed model occurs when $\alpha = Q$ and $\gamma = P$. The stationery point is evaluated as $(\frac{\alpha}{\beta}, \frac{\gamma}{\delta})$, mentioned in Fig.2 (Main File).

Appendix B. Matlab Code for Fig. 4, 5 and 6 (Main File)

$span = 0 : 0.1 : 100;$
$[t, y] = ode45(@Lotka, span, [60; 80]);$
$plot(t, y(:, 1), '-o', t, y(:, 2), '-o')$
$function \quad dydt = Lotka(t, y)$
$dydt = [20 * y(1) - y(1) * y(2); -20 * y(2) + y(1) * y(2)];$
end

Appendix C. R Code for data fitting: Fig.29 (Main File)

The following R code is used to generate the plot:

```
library(plotrix)
library(ellipse)
data = read.csv("Merged.csv")
data1 = read.csv("sheet2.csv"); \\ data set from cloudsim results of LV runs;
predator increasing-prey decreasing \\
data2 = read.csv("sheet1.csv"); \\ data set from cloudsim results of LV runs;
predator decreasing-prey increasing \\
x=data$Cloudlet.Number
y=data$VM.Number
x1= data1$Cloudlet
y1=data1$VM
x2=data2$Cloudlet
y2=data2$VM
#ellipse fitting with 95\% confidence level
eli1 = ellipse(cor(x1,y1),scale=c(sd(x1),sd(y1)),
centre=c(mean(x1), mean(y1)), level = 0.95, npoints = 250)
eli2 = ellipse(cor(x2,y2),scale=c(sd(x2),sd(y2)),
centre=c(mean(x2), mean(y2)), level = 0.95, npoints = 250)
#Calculate the center of ellipse
        [c1x,c2y] = c(mean(eli1[,1]), mean(eli1[,2]))
        [c2x,c2y] = c(mean(eli2[,1]), mean(eli2[,2]))
    sink()
    jpeg("plotfit.jpeg")
```

```
plot(x,y)
draw.ellipse(118.21488,31.59861 , a = 90, b = 25)
draw.ellipse(29.98445,55.116337, a = 20, b = 35)
dev.off()
```

.

Appendix D. Numerical Solution of Lotka-Volterra

To retrieve result more accurately from LV model, we have employed the use of Runge kutta methods of fourth and fifth order, implemented by Fehlberge and denoted as RKF45. Numerical analysis to solve LV model is appreciated due to the inherent difficulty of solving the LV model analytically. We proceed in the following manner:
RKF45 produces an approximate solution in vector form y_n by dividing the solution domain (Euclidean or Hilbert space, typically) into a set of discrete points. We begin with the initial data at time $t_0 = 0$ and estimate the approximation solution at time $t_i = i * h$, i = 1, 2, ...n. The step size h is chosen suitably such that it is not too big or too small. We use RK4 and RK5 (Runge Kutta 4th order and 5th order respectively) at each step i to generate two different solutions and compare the proximity of the solutions thus generated. The approximation is acceptable within a certain tolerance as long as the difference between the two approximations doesn't exceed the predefined tolerance. The step size may be modified to accommodate the tolerance criterion. However, we need to increase the step size if the two approximate solutions agree to more significant digits than required. Numerical methods are sensitive to approximation and thus the following points must be stressed:

1. We use Taylor series expansion of the function around the iteration point at each step to approximate a function. This produces truncation error, large or small depending on the number of terms used in the expansion. If h_n denotes the difference between $n+1^{th}$ and n^{th} iteration, then a fourth order method produces an error of the form Ch^5 for some constant C. This means that a step size of magnitude $\frac{h_n}{2}$ shall reduce the error by a factor of $2^5 = 32$.
2. A 5th order Runge-Kutta method requires executing four function evaluations to obtain local truncation error of order 5. We observe, the numerical solution to the ordinary differential equation can be 5^{th} order

accurate locally but may still not address the issue of global convergence adequately.

3. Roundoff error is inevitable. The estimate of VM's turns out to be a ballpark figure, precisely for this reason.
4. The population dynamics may deviate slightly from the standard assumption about the model for the above reasons.

We adopted Runge-Kutta-Fehlberg 45 (RKF45) method, (ode 45 in Matlab) symbolized by function evaluations with an additional evaluation to accomplish 5th order accuracy. This generates a local error of the order h^6, significantly small if h is chosen to be small enough. Please note, h is chosen to be between 0 and 1.

The parameters for simulation are computed using this method. ode45 of matlab (Appendix B), which employs Runge-kutta method is used rigorously to derive the datasets table D.1, D.2 and D.3 corresponding to the three cases: Prey Increasing-Predator Decreasing, Prey-Predator stability and Prey Decreasing-Predator Increasing.

Time	VM	cloudlets
0	30	50
0.1	73.817541	14.87
0.2	25.19	23.96
0.3	84	42.70
0.4	36.78	12.97
0.5	37.24	58.28
0.6	63.62	12.56
0.7	24.35	30.050
0.8	89.46	30.17
0.9	31.52	14.92
1.0	49.80	62.49
1.1	53.56	11.46
1.2	25.13	38.34
1.4	85.68	20.72
1.5	27.62	18.14
1.6	67.18	57.75
1.7	44.61	11.51
1.8	28.59	48.25
1.9	76.06	15.35
2.0	25.04	22.87

Table D.1: This table demonstrates a scenario (Case 1), where it is required to increase the Prey(VM) number. In the table, time-span from 0 to 2.0 has been taken for better understanding of how the model works. Initially, the predator and the prey numbers are taken as 30 and 50 respectively. Time-span is displayed in the table for every 0.1 interval. As the intention was to increase the number of prey from 30, in the next immediate time-span it can be noticed that the prey number surges to 73, a two fold jump from the initial value. Apart from a few occurrences, through out the period till 2.0, the number of prey is higher than initial value. In the case of predators, the number of predators are less than the initial value 50. Only one occurrence at time-span 1.0, the predator population is more than the initial value. The prey-predator numbers in the table and the figure will be different, if any other initial values are considered. As the proposed model is not a linear function, there is no pattern visible in the prey-predator numbers.

Time	VM	cloudlets
0	80	150
0.1	80	150
0.2	80	150
0.3	80	150
0.4	80	150
0.5	80	150
0.6	80	150
0.7	80	150
0.8	80	150
0.9	80	150
1.0	80	150
1.1	80	150
1.2	80	150
1.4	80	150
1.5	80	150
1.6	80	150
1.7	80	150
1.8	80	150
1.9	80	150
2.0	80	150

Table D.2: Predator Prey stability is the scenario (Case 2) where the same VM(prey) and cloudlets(Predator) numbers need to be maintained. The table D.2 displays the predator, prey numbers at each time point, which are collected after 0.1 time interval. The table also supports the conclusion drawn from the figure that there is no change in VM, cloudlets number as time passes from 0 to 100.

Time	VM	cloudlets
0	60	80
0.1	34.73	66.19
0.2	18.97	69.16
0.3	11.40	82.47
0.4	8.33	103.14
0.5	7.99	132.47
0.6	11.21	168.11
0.7	23.44	197.33
0.8	55.89	180.68
0.9	76.83	112.69
1.0	53.72	72.76
1.1	28.69	64.30
1.2	15.14	71.04
1.4	9.29	87.74
1.5	7.23	114.07
1.6	8.34	150.29
1.7	15.28	189.09
1.8	40.13	200.55
1.9	77.98	137.38
2.0	64.06	78.36

Table D.3: This table captures the situation where the VM(Prey) needs to reduce but cloudlets(Predator) number is required to increase (Case 3). The table displays a few data points used to plot the figure. The initial VM, cloudlets values are 60,80. Except a few, all the VM values are less than initial VM value. In the case of cloudlets, there are a few occurrences, where cloulets values are less than initial value but maximum cloudlets values are higher than initial cloudlet value.

Appendix E. Task Scheduling Algorithms

We compared our model performance with Standard task scheduling algorithms in CloudSim. We list those frequently used algorithms below:

First Come First Serve :

This is one of the most simple algorithms and very easy to implement. The job/ task arriving first in the queue is assigned accordingly to a VM for execution. As it doesn't consider the execution time of the arrived task before allocation, sometimes it doesn't result in an efficient load balancing. A short job has to wait for longer time until a longer job finishes its execution. Therefore, it does not guarantee good response time.

Round-Robin Algorithm:

It is a well known algorithm widely used in scheduling and load balancing. It selects the first VM randomly, assigns the tasks and selects the next VM/node in a circular manner. The advantage of the Round-Robin (RR) algorithm is it's simplicity. Sometimes, RR algorithm doesn't allocate the tasks to VM efficiently because it doesn't consider load, space, response time or any other parameter while allocating. Another variant of RR algorithm is weighted RR algorithm where each VM/node is assigned with a weight. The VM with more weight receives more tasks. If two VMs have equal weight, they both will be allocated with equal number of tasks.

Shortest Job First:

This scheduling algorithm selects the task having lowest execution time and assigns to a VM first. The job which has the highest execution time will be given the lowest priority. If two jobs demand equal execution time, it follows FCFS scheduling.

Longest Job First:

The job with the longest execution time is assigned to a VM. This is in stark contrast to SJF algorithm. SJF has disadvantages such as starvation, where a job with longest execution time waits for long time. If there is a flow of jobs which are shorter in execution time, then the longest job will not be assigned to any VM. To overcome this, LJF can be used in Cloud environment.

Opportunistic Load Balancing Algorithm:

Opportunistic Load Balancing (OLB) algorithm tries to keep the nodes busy irrespective of their current workloads. It assigns the task to a node in a random fashion. As it doesn't consider the current workload before assigning the task, sometimes it doesn't produce desired performance.

Min-min Load Balancing Algorithm:

This is a static load balancing algorithm as it needs to know all relevant parameters before assigning the task to a node. It calculates the probable execution time and the completion time of all the tasks waiting in the queue. Then the task with minimum execution time is allocated to a node/VM, which requires minimum completion time. Therefore, tasks with maximum execution time has to wait until other tasks/jobs are assigned to the VMs. The completion time has to be updated when a task is assigned to a VM so that the task is removed from the meta task list. The entire process continues till the meta task list becomes empty. The Min-min algorithm performs better than many other load balancing algorithms. But the algorithm needs to have knowledge of the execution time, completion time in advance before taking decision regarding allocation of the tasks.

Appendix F. Resource Allocation, Scaling by Parameter Tuning and Timesharing: Algorithms

Algorithm 1 Lotka-Volterra algorithm in Cloud Dynamics

1: **procedure** LOTKA-VOLTERRA(p,q) ▷ p is prey(VM), q is predator(Cloudlet)
2: $p \leftarrow VMs$ ▷ Initialize VM
3: $q \leftarrow cloudlets$ ▷ Initialize cloudlets
4: **while** $VM = 0$ **do**
5: **while** $(\gamma \geq P) and (Q \geq \alpha)$ **do**
6: $\gamma \leftarrow \gamma + \epsilon$ ▷ ϵ is infinite small number
7: $\gamma Q \geq \alpha P$
8: **end while**
9: **end while**
10: **while** $VM \leftarrow maxVMandcloudlets \neq 0$ **do**
11: **while** $(\alpha > Q) and (\gamma < P)$ **do**
12: $\alpha \leftarrow \alpha + \epsilon$ ▷ ϵ is infinite small number
13: $\alpha P \geq \gamma Q$
14: **end while**
15: **end while**
16: **return**
17: **end procedure**

Algorithm 2 Scaling by Parameter Tuning

1: **procedure** LV-PARAMETER-TUNING ▷
2: $maxT \leftarrow MaximumThreshold$
3: $minT \leftarrow MinimumThreshold$
4: $VM \leftarrow Number - of - VMs - in - present - VM - pool$ ▷ Initialize VM
5: $T \leftarrow Time$
6: **while** $T \neq 0$ **do**
7: **while** $CPU - Utilization > maxT$ **do** ▷ Trigger LV, VM increasing cloudlets decreasing
8: $(new - VM > allocated - VM) and (new - VM < allocated - VM + VM$ ▷
9: $VM \leftarrow VM + additional - VM - in - pool$
10: **end while**
11: **while** $CPU - Utilization < minT$ **do** ▷ Trigger LV, VM decreasing cloudlets increasing
12: $(new - VM < allocated - VM) and (new - VM > 0)$ ▷
13: $VM \leftarrow VM - LV - generated - number$
14: **end while**
15: $T \leftarrow T - 1$
16: **end while**
17: **return**
18: **end procedure**

Algorithm 3 LV-Timeshared Algorithm

1: **procedure** LV-TIMESHARED ▷
2: $scloudlets \leftarrow cloudlets - submitted - for - execution$
3: $cloudlets_Q \leftarrow existing - cloudlets - in - queuee$
4: $T \leftarrow Time - Allocated$
5: $oVM \leftarrow occupancy - of - VMs$
6: **while** $T \neq 0$ **do**
7: call procedure $LV - PARAMETER - TUNING$
8: **if** $oVMs > MaxThreshold$ **then**
9: ▷ VM increasing, cloudlets decreasing
10: **while** $oVM > available - VM$ **do**
11: add $cloudlets_Q \leftarrow scloudlets + cloudlets_Q$
12: **end while**
13: **end if**
14: **if** $oVMs < MinThreshold$ **then**
15: ▷ VM decreasing, cloudlets increasing
16: add $cloudlets_Q \leftarrow scloudlets + cloudlets_Q$
17: **end if**
18: $T \leftarrow T - 1$
19: **end while**
20: **return**
21: **end procedure**

Appendix G. Implementation in R: Ref. to 6.4, main file

We have used the ellipse package in R to generate the ellipse given the parameters h, k, a, b. The plotrix package is used to plot the ellipse and estimate the parameters with 95% confidence. The function 'ellipse' (defines in the ellipse package) allows us to draw a two-dimensional ellipse that traces a bivariate normal density contour for a given mean vector, covariance matrix, and probabilistic proximity of the points lying on the ellipse trace. Using this function, we have computed the center of the ellipse with a reasonable degree of precision. We then computed the distance of the closest point and the furthest point from the dataset and set these values as b and a (major/minor axes of the ellipse, tweaking these values would generate a family of concentric ellipses which would help explain the elastic behavior of the model).
where, a = Length of longitudinal axis and b = Length of transverse axis.
Finally the ellipse that fits the curve is drawn using the 'plotrix' library. Let us consider the following example.
On giving an input x (Cloudlet.Number)=38.34 to the first ellipse equation, we obtain $y = 23.317108996715966$ where expected output is $y = 25.13$. This gives us an accuracy of about 92.98 %.
Similarly on giving an input x (Cloudlet.Number)=150.29 to the second ellipse equation, we get $y = 8.240179917046074$ where expected output is $y = 8.34$. This gives us an accuracy of about 98.8 %. These two samples, randomly selected from the pool of simulated data, are sufficient to testify the goodness of fit of the predictive model. We may forecast the required VM population using the fit when the cloudlet population is known.

Appendix H. Performance Benchmarks: Relevant Definitions

- Make span : Make-span is an another metric, used in this paper to measure the performance improvement of various algorithms such as reactive scaling, proactive scaling, Cloudlet time shared algorithm, etc, after the introduction of the Lotka-Volterra model into the afore mentioned algorithms. Reference of make-span can be observed across the paper such as subsection VM Elasticity, Proactive Scaling under "Model Implementation and Outcome: Technical Discussion" section.

- Response time : It is a widely used QoS parameter in Cloud computing. We have shown that the implementation of Lotka-Volterra model has

improved the response time performance metric significantly. Multiple occurrences of this particular QoS can be observed in various sections of this paper namely "Model Implementation and Outcome: Analysis of results".

- Utilization : Utilization has been used heavily in this paper. In the algorithms such as LV-Timeshared algorithm, the invocation of Lotka-Volterra has occurred whenever the utilization is touching the maximum/ minimum predefined threshold.

- Reduction in SLA violation: SLA violation is the most important performance metric of a Cloud data center. Revenue of a Cloud service provider is tightly coupled with SLA violation rate. After the introduction of LV model into various algorithms, SLA violation rate has been reduced, which has improved the quality of service and job satisfaction. More details can be found in "Model Implementation and Outcome: Analysis of results" section.