Controlling the number of active instances in a cloud environment

Diego Goldsztajn
Universidad ORT Uruguay
goldsztajn@ort.edu.uy

Andres Ferragut
Universidad ORT Uruguay
ferragut@ort.edu.uy

Fernando Paganini
Universidad ORT Uruguay
paganini@ort.edu.uy

Matthieu Jonckheere
Universidad de Buenos Aires
mjonckhe@dm.uba.ar

ABSTRACT

We study a cloud environment in which computing instances may either be reserved in advance, or dynamically spawned to serve a fluctuating or unknown load. We first consider a centralized scheme where a system operator maintains the job queue and controls the spawning of additional capacity; through queueing models and their fluid and diffusion counterparts we explore the tradeoff between queueing delay and the service capacity variability. Secondly, we consider the setting of a dispatcher who must immediately send jobs, with no delay, to decentralized instances, and in addition may summon extra capacity. Here the capacity scaling problem couples with one of load balancing. We show how the popular join-the-idle-queue policy can be combined with an adequate rule for spawning instances, yielding an equilibrium with no queuing delay and controlling service capacity variability; we accommodate as well the case where spawned instances incur startup delay. Finally, we analyze the question of deciding, for a given pricing structure for the cloud service, how many fixed instances should be reserved in advance. The behavior of these policies is illustrated by simulations.

1. INTRODUCTION

In modern computer networks, services are usually provided over the "cloud", meaning that the resources required to serve a given computing task (processing time, memory, etc.) are requested dynamically to a mutualized computing infrastructure residing in one or multiple data centers. Companies like Amazon or Google provide the infrastructure as a service, and application providers such as Netflix or Dropbox rent the capacity to match supply with demand.

In this context, the question arises on *how* to scale the number of allocated resources to match the load. From a customer perspective, resources are available on-demand, and thus the system can adapt quickly to changes in the load profile. However, two main issues have to be addressed. The first issue is that, typically, the infrastructure providers offer a discount on *reserved* instances, meaning that purchasing capacity on a long term contract is preferable over on-demand requests. Ensuring a predictable number of computing instances is crucial for calculating the necessary long-term reservations, and thus minimize the costs incurred by on-demand instances [3, 8]. A second issue is that instance creation may not be immediately achieved, due to the setup time of the requested virtualized infrastructure over the real hardware. For a recent discussion on this issue see [6].

In this work, we explore some of the tradeoffs present in this setting: in Section 2, we model a cloud computing system as a queue with a fixed number of servers coming from long term reservations, and a variable number of on-demand instances. We analyze an algorithm for spawning new instances on demand *in feedback* by borrowing ideas from control theory, extending our previous work [2]. We provide a fluid limit analysis of the system, identify the key parameters and show how the system can tradeoff queueing delay with variability in capacity usage for a given demand.

Our first model deals with a central queue that dispatches jobs to servers. However, typical load balancers in cloud systems try to make immediate decisions, routing jobs as they arrive. Our second model, presented in Section 3 incorporates the spawning of instances in feedback to typical load balancers such as random routing, power-of-d choices [5,7] or Join the Idle Queue [4]. In the load balancing literature, the number of active instances is typically kept fixed. We show that adding a simple rule for controlling the number of active instances yields excellent performance, by enabling the system to track automatically the load profile while keeping utilization high. We also analyze the effect of adding delay in instance creation, extending some of the results of [6], and giving simple design rules to achieve near-optimal performance.

Finally, in Section 4 we analyze the problem of purchasing reserved capacity from a customer perspective, and relate it to the well known newsvendor problem. We show that having a proper approximation of the (random) steady state number of required instances to serve the load allows the customer to optimally choose the number of long term reservations, and we apply this to the models derived. Conclusions are given in Section 5

2. DYNAMIC SCALING WITH CENTRAL QUEUEING

Consider a system where computing jobs arrive as a Poisson process of rate λ. Each job may be allocated a comput-

IFIP WG 7.3 Performance 2017. Nov. 14-16, 2017, New York, NY USA

ing instance (assumed homogeneous), and its service duration is exponentially distributed with parameter μ. In what follows, we will assume that $\mu = 1$, which amounts to a choice of units. In this case, λ also represents the system load. Upon arrival, the request may be allocated an instance immediately or may be queued if all instances are busy.

We consider that the system has two types of instances: a fixed number N of *reserved* instances, purchased in advance through a long-term contract, and a dynamically adjusted number $m(t) \geqslant 0$ of *on-demand* instances, called *helpers* in the following for brevity. Let $n(t)$ denote the number of jobs present in the system at time t. If the allocation policy is work conserving, the service rate of the system is then $\min\{n, N+m\}$.

As an example, if we choose $m \equiv 0$, i.e. no extra capacity is ever purchased, then the system becomes a finite server queue with N servers. Of course, in that case, if the number of reserved instances is below the load ($\lambda > N$) the system will be unstable. On the other extreme, if we can spawn a new instance immediately upon arrival when the reserved ones are full, we have $m(t) = (n(t) - N)^+$, and thus the service rate is $n(t)$ at all times. In this case, the system scales automatically to meet demand, behaving like an infinite server queue.

Consider now that the customer has acquired the reserved instances N, which may be less than λ. To meet demand in real time, the system provides an auto-scaling feature that dynamically spawns new instances. A simple procedure is the one discussed previously: spawning a new instance for each new job. In this case, the queueing delay for new jobs is 0, but the system provider loses control on the number of active helpers in the system, which can turn out to be very expensive in the long run.

To improve this situation, we propose the following decentralized alternative to control the number of on-demand instances: each job currently in the queue requests a new helper at a *spawning rate* β (i.e. a new instance is created after a random $\exp(\beta)$ time for each job in the queue). Moreover, each helper instance only lasts in the system a time $\exp(\gamma)$, i.e. γ is the *recalling rate*. In this setting, the state of the system $(n(t), m(t))$ behaves as a continuous time Markov chain with transition rates depicted in Fig. 1.

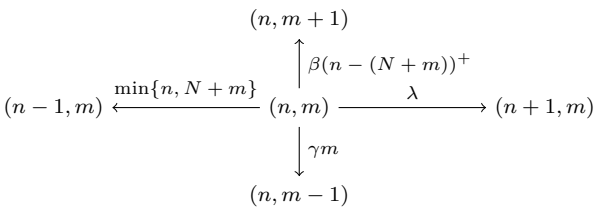


Figure 1: Transition rates for the system.

By performing a large scale limit of the system ($\lambda \to \infty$) and scaling the number of reserved instances accordingly, we can derive the fluid limit for the Markov chain of Figure 1, which is given by the second order dynamics:

$$\dot{n} = \lambda - \min\{n, N+m\}, \tag{1a}$$

$$\dot{m} = \beta(n - m - N)^+ - \gamma m. \tag{1b}$$

We now analyze the system in the fluid scale. The equilibrium of the above dynamics depends on the load: if $\lambda < N$, the equilibrium is simply $n^* = \lambda$, $m^* = 0$ and no extra instances are needed. The interesting case is when $\lambda > N$ and thus we need a positive number of extra instances to cope with the load. In that case, imposing equilibrium in (1) we get:

$$n^* = \lambda + \frac{\gamma}{\beta}(\lambda - N), \quad m^* = \lambda - N.$$

Note that in equilibrium, the average number of servers (reserved or not) should match the load, i.e. $N + m^* = \lambda$. Moreover, the number of jobs in the system is larger than λ. Therefore, the queue size in equilibrium is given by:

$$q^* = n^* - (N + m^*) = \frac{\gamma}{\beta}(\lambda - N) > 0.$$

Since we are interested in keeping a predictable profile in the number of helpers, we would like to model the behavior of the system around the equilibrium to estimate the steady state variances. To do so, we perform a second order analysis of the system by linearizing the dynamics and writing the following stochastic differential equation.

Denoting by δn, δm the incremental quantities, the resulting dynamics are:

$$d(\delta n) = [-\delta m]dt + \sqrt{\lambda}[dW_1 - dW_2], \tag{2a}$$

$$d(\delta m) = [\beta\delta n - (\beta+\gamma)\delta m]dt + \sqrt{\gamma(\lambda - N)}[dW_3 - dW_4]. \tag{2b}$$

Here, W_i are standard Brownian motions, representing arrivals, departures, helper creation and destruction, i.e. the randomness in the system around equilibrium. The coefficients of the noise inputs come from the local representation of the Markov chain. Taking $x = [\delta n,\ \delta m]^T$, we can rewrite the above in state-space as $dx = Axdt + BdW$ with:

$$A = \begin{pmatrix} 0 & -1 \\ \beta & -(\beta+\gamma) \end{pmatrix},$$

$$B = \begin{pmatrix} \sqrt{\lambda} & -\sqrt{\lambda} & 0 & 0 \\ 0 & 0 & \sqrt{\gamma(\lambda-N)} & -\sqrt{\gamma(\lambda-N)} \end{pmatrix}$$

The steady-state covariance matrix of the above system is given as the solution of the Lyapunov equation: $AQ + QA^T + BB^T = 0$, yielding:

$$Q = \begin{pmatrix} \frac{\gamma+1-\beta}{\beta}\lambda - \frac{\gamma}{\beta(\beta+\gamma)}N & \lambda \\ \lambda & \lambda - \frac{\gamma}{\beta+\gamma}N \end{pmatrix},$$

Since we are interested only in the number of helpers, from the above analysis we can recover $E[(\delta m)^2]$ as:

$$E[(\delta m)^2] = \lambda - \frac{\gamma}{\beta+\gamma}N.$$

As a comparison point, let us go back again to the infinite server policy with $\lambda > N$: in that case, the total number of instances is $n = N + m$. By a similar approach, one can estimate $E[(\delta m)^2]$ in this case, yielding $E[(\delta m)^2] = \lambda$. Therefore, our control policy enables the system operator to reduce variability. at the expense of queueing delay. Define now $\alpha := \gamma/\beta$, i.e. the recalling rate to spawning rate ratio. For a given load λ, the system operator has a tradeoff between the following two quantities:

$$q^* = \alpha(\lambda - N),$$

$$E[(\delta m)^2] = \lambda - \frac{\alpha}{1+\alpha}N.$$

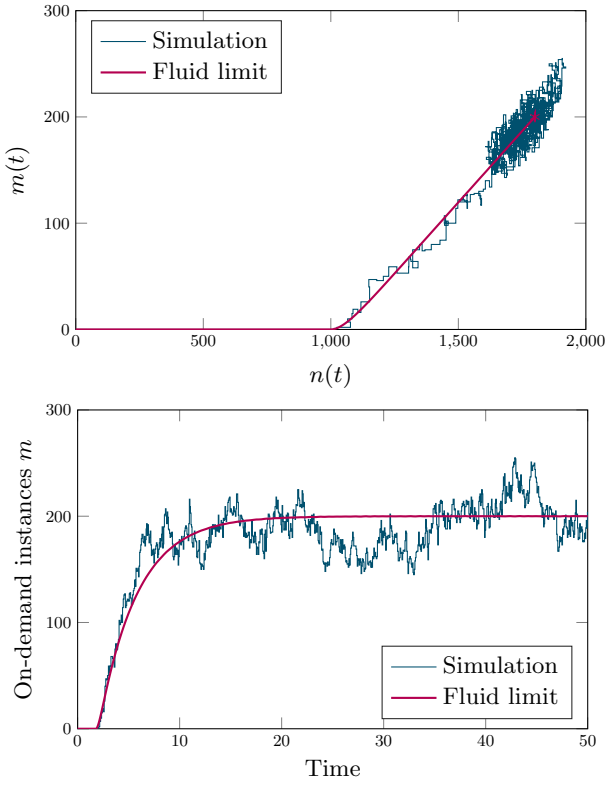


Figure 2: Simulation experiment of the spawning dynamics with $\lambda = 1200$, $N = 1000$ and $\alpha = 3$. Above: state-space evolution and fluid limit. Below: number of on-demand servers.

Increasing α means that the recalling rate is high, which reduces variability but also increases queueing delay. From a provider perspective, one can use the preceding expressions to tune the ratio α such as queueing delay is controlled, and the variability is reduced to acceptable levels. We are currently exploring the design space of policies to get even better performance.

As an example, in Figure 2, we plot system evolution for $\lambda = 1200$, $\mu = 1$, $N = 1000$ and $\alpha = 3$. The first graph shows the state-space evolution of the system and its fluid limit approximation. We also plot the number of on-demand instances in the system as a function of time. In steady state, the latter achieves a measured variance of $\sigma^2(m) = 440$ (standard deviation ≈ 20.98), whereas the predicted value from the model parameters is 450. As a comparison point, the variability of the infinite server case would be $\lambda = 1200$, i.e. a standard deviation of ≈ 34 instances.

3. DYNAMIC SCALING AND LOAD BALANCING

In the previous section we analyzed a system where jobs are queued in the dispatcher until a server becomes available. However, in cloud systems this type of central queueing is undesirable, and we now turn our attention to systems that dispatch their tasks immediately to running instances, and therefore the problem must incorporate online load balancing.

In the decentralized load balancing literature [1], systems are modeled in the "mean field" regime: a cluster of N equally behaving instances, with service rate $\mu = 1$, is assigned jobs through a dispatcher, which receives jobs at rate $N\lambda$, with $\lambda < 1$. Thus λ again represents the "load" of the system in the sense of an equivalent $M/M/1$ queue.

The state of the system, for fixed N, would be the vector $Z \in \mathbb{N}^N$ storing the number of jobs queued at each instance. However, a simpler state is available through aggregation of identical systems: if $X_i(t)$ denotes the number of queues with exactly i jobs, for $i = 0, 1, \ldots$, then $X(t) = (X_0(t), X_1(t), \ldots)$ is in general a Markov process in the subset of $\mathbb{N}^{\mathbb{N}}$ that satisfies $\sum_{i=0}^{\infty} x_i = N$.

An equivalent formulation is to choose as state $S_i(t)$, the number of queues with at least i jobs. Then $S(t)$ lives in the set of positive nonincreasing sequences with $S_0 = N$. Note that $X_i(t) = S_i(t) - S_{i+1}(t)$, $i = 0, 1, \ldots$. Also, for any policy where routing may only depend on the number of jobs in the queue (and not on individual identification of the queue), $S(t)$ (or equivalently $X(t)$) is a detailed enough description of the state.

Typically the fluid (mean field) limit is taken as the number of instances $N \to \infty$, and $S_i(t)/N \Rightarrow s(t)$, where $s(t)$ satisfies some kind of infinite dimensional differential equation. Note that with this scaling $s_0(t) \equiv 1$.

As an example, the random load balancer satisfies the following fluid limit:

$$\begin{aligned} &s_0 \equiv 1, \\ &\dot{s}_i = \lambda(s_{i-1} - s_i) + (s_{i+1} - s_i) \quad i \geqslant 1. \end{aligned}$$

And its equilibrium is:

$$s_i^* = \lambda^i,$$

provided the stability condition $\lambda < 1$ holds.

More interesting choices for the load balancing are power-of-d choices [5, 7] and Join the Idle Queue (JIQ). For instance, JIQ proposes to keep a list of the current empty queues and only route to them, provided there are any available. Otherwise it defaults to random routing to keep an empty dispatcher queue. The mean field dynamics for fixed servers is as follows:

$$\begin{aligned} &s_0 \equiv 1, \\ &\dot{s}_1 = \lambda \mathbf{1}_{\{s_0 > s_1\}} + (s_2 - s_1), \\ &\dot{s}_i = \lambda(s_{i-1} - s_i)\mathbf{1}_{\{s_1 = s_0\}} + (s_{i+1} - s_i) \quad i \geqslant 2. \end{aligned}$$

Its equilibrium can be readily computed under the stability condition $\lambda < 1$ to yield:

$$s_0^* = 1, \quad s_1^* = \lambda, \quad s_i^* = 0 \text{ for } i > 1.$$

Thus in equilibrium there is a positive number of free instances to use, provided the system is dimensioned correctly.

3.1 Dynamic scaling for JIQ

Our main concern is that the above analysis assumes that one has a large but *fixed* number of instances, and thus if the system is not dimensioned correctly ($\lambda < 1$) then stability is lost. We would like to add some kind of "auto-scaling" feature, to the above, harnessing the power of cloud systems to scale. The auto-scaling policy should work in feedback with the number of instances in use and stabilize the system for any λ. Let us explore how to incorporate this into the load-balancing models.

The simplest way to stabilize the system is to create a new instance for each arriving job, corresponding to an $M/M/\infty$ system. In this case there is no queueing delay, and the number of *active* instances will scale automatically to λ. However, one loses control of the number of instances invoked, and moreover it may be infeasible to immediately create an instance for an arriving job due to setup delays.

The main idea here is that the number of servers is not fixed *a priori*, so S_0 is adjusted in feedback. This can be formalized in the following way: let λ scale as $L\lambda$ with $L \to \infty$ and take $\bar{S}(t) = S(t)/L$. Then $\bar{S}(t)$ converges to $s(t)$, an infinite dimensional dynamics in the cone of decreasing sequences, but s_0 may not be 1. In fact, s_0 is the (fluid) amount of instances invoked to cope with the demand.

As an example, take the following feedback rule: every arriving job is immediately assigned to an idle server, as in JIQ, provided there is any. Also, upon arrival, the system calls for a server creation to replace the one used. To adjust to demand, idle queues are killed at rate γ (i.e. if not used for an $\exp(\gamma)$ time they switch off). This feedback rule yields the following mean field dynamics:

$$\dot{s}_0 = \lambda - \gamma(s_0 - s_1), \tag{3a}$$

$$\dot{s}_1 = \lambda \mathbf{1}_{\{s_0 > s_1\}} + (s_2 - s_1), \tag{3b}$$

$$\dot{s}_i = \lambda \frac{1}{s_0}(s_{i-1} - s_i)\mathbf{1}_{\{s_1 = s_0\}} + (s_{i+1} - s_i) \quad i \geqslant 2. \tag{3c}$$

Note that in eq. (3c), the extra term $\frac{1}{s_0}$ in the random routing part is due to the fact that s_0 may not be 1.

Imposing equilibrium conditions in (3) we arrive at:

$$s_0^* = \left(1 + \frac{1}{\gamma}\right)\lambda, \quad s_1^* = \lambda, \quad s_i^* = 0 \text{ for } i > 1;$$

valid for any positive λ. It is worth noting that the system auto-scales: the number of working queues in equilibrium matches the system load λ, and the system automatically provisions an extra proportion of idle queues equal to λ/γ. Therefore, controlling the "turning-off" rate γ one can drive the system to automatically work with a small amount of idle instances.

A second order analysis of the system can be performed, along the same lines of the previous section. As the scale of the system gets large, the state space collapses to $s_i = 0$ for all $i > 1$. Considering the arrival, departure and instance recalling perturbations, we get the following stochastic dynamics around equilibrium:

$$d[\delta s_0] = -\gamma \delta s_0 dt + \gamma \delta s_1 dt + \sqrt{\lambda} dW_1 - \sqrt{\lambda} dW_2;$$
$$d[\delta s_1] = -\delta s_1 dt + \sqrt{\lambda} dW_1 - \sqrt{\lambda} dW_3.$$

where W_i are independent standard Brownian motions accounting for job arrivals, instance recalling and departing jobs. The linearized system can be written in state space form taking $X = [\delta s_0 \delta s_1]^T$ as $dX = AX dt + B dW$ where:

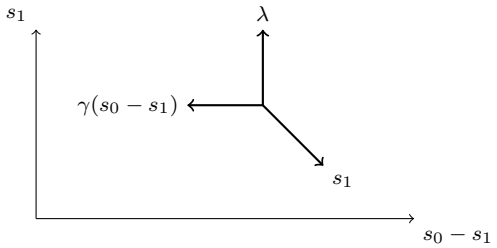
$$A = \begin{bmatrix} -\gamma & \gamma \\ 0 & -1 \end{bmatrix} \quad B = \begin{bmatrix} \sqrt{\lambda} & -\sqrt{\lambda} & 0 \\ \sqrt{\lambda} & 0 & -\sqrt{\lambda} \end{bmatrix};$$

Solving the Lyapunov equation we can compute the covariance matrix in steady state:

$$Q = \begin{bmatrix} \left(1 + \frac{1}{\gamma}\right)\lambda & \lambda \\ \lambda & \lambda \end{bmatrix}.$$

From the above analysis, we can conclude that the total number of active instances in the system will be approximately Gaussian with mean and variance $\lambda(1 + 1/\gamma)$. Moreover, computing the variance of $s_0 - s_1$ from Q, we can deduce that the total number of idle instances will be approximately Gaussian with mean and variance λ/γ, and $s_0 - s_1$ is not correlated to s_1.

This fact can be better understood using the state space collapse: when the scale is large, the dynamics reduces to the following Markov chain in the number of working and idle servers:



The invariant distribution of the above Markov dynamics is product form, with a Poisson distribution for $s_0 - s_1$ of mean λ/γ and a Poisson distribution for s_1 with mean λ. [1]

This analysis also suggests a simple rule for choosing the recalling rate γ to avoid operating the system with no idle queues (and thus routing at random increasing queueing delay). Since $s_0 - s_1 \approx N(\lambda/\gamma, \lambda/\gamma)$, a rule of thumb for choosing γ is to let λ/γ be two standard deviations above 0, which leads to:

$$\frac{\lambda}{\gamma} - 2\sqrt{\frac{\lambda}{\gamma}} > 0 \Rightarrow \gamma = \frac{\lambda}{4}.$$

This rule can be easily implemented in practice in an approximate fashion: simply switch off a server that is still idle after 4 arrivals.

In Figure 3 we plot the time evolution of such a system, for $\lambda = 100$ and γ chosen as before, and we compare it with the fluid limit solution. We also plot the number of idle instances available, showing that the system automatically keeps most of the time a low but nevertheless positive number of idle instances to cope with new arrivals. To illustrate the state space collapse , in Figure 4, we plot the average number of instances with more than i jobs, i.e. the empirical estimation of S_i, in steady state. Note that $S_2 \approx 0$ and $S_3 = 0$, meaning that the system seldom uses queues with more than 1 job, and in this particular run instances with 2 jobs were never allocated more jobs.

3.2 Considering instance creation delay

The preceding analysis assumes that computing instances can be spawned with no delay: in that case, JIQ is almost emulating the infinite server queue. However, we can incorporate instance creation delay in the model, by following the same ideas as in [6].

Consider now that each instance invoked requires a random setup time with exponential distribution of parameter ν (i.e. $1/\nu$ is the average setup time). Let $u(t)$ be the number of instances in the process of setting up, and assume that each new arrival spawns a new instance creation. As

[1] The above analysis also hints that the results may be insensitive to the job size as well as the recalling rate distributions

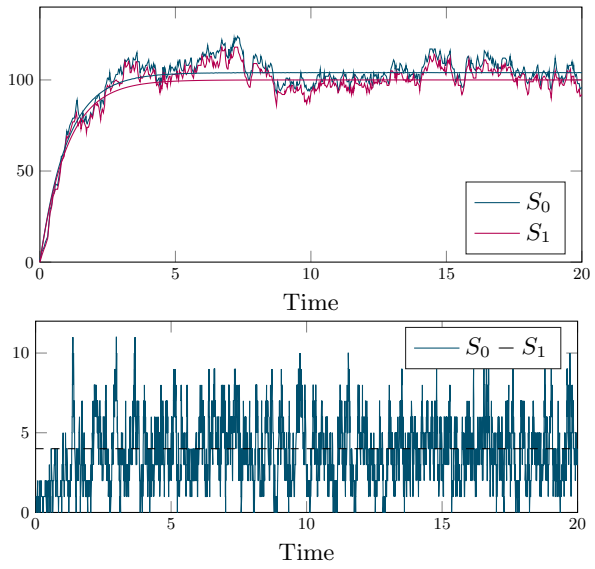


Figure 3: Time evolution of the JIQ system with instance scaling feedback for $\lambda = 100$, $\mu = 1$ and $\gamma = \lambda/4$. Above: total number S_0 and currently working S_1 instances and their fluid limit. Below, number of idle instances in the system.

before, idle instances are recalled at rate γ. Load balancing is performed by following the JIQ policy. The dynamics of the system in the fluid scale are now:

$$\dot{u} = \lambda - \nu u, \tag{4a}$$
$$\dot{s}_0 = \nu u - \gamma(s_0 - s_1), \tag{4b}$$
$$\dot{s}_1 = \lambda \mathbf{1}_{\{s_0 > s_1\}} - (s_1 - s_2), \tag{4c}$$
$$\dot{s}_i = \lambda \left(\frac{s_{i-1} - s_i}{s_0} \right) \mathbf{1}_{\{s_0 = s_1\}} - (s_i - s_{i+1}). \tag{4d}$$

The equilibrium can be readily computed to yield:

$$s_0^* = \left(1 + \frac{1}{\gamma}\right) \lambda, \quad s_1^* = \lambda, \quad s_i^* = 0;$$

and for the setting up instances we get:

$$u^* = \frac{\lambda}{\nu}.$$

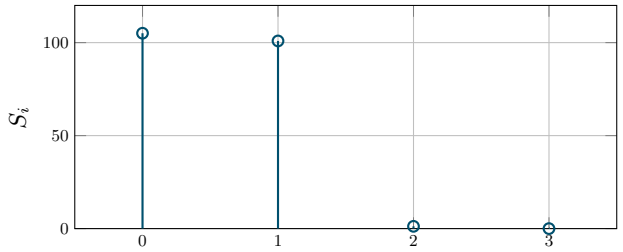


Figure 4: Empirical estimation of S_i in steady state for the JIQ system with instance scaling feedback. $\lambda = 100$, $\mu = 1$ and $\gamma = \lambda/4$.

Therefore, the system auto-scales in a similar fashion, over-provisioning idle instances to route arriving jobs and and "warming up" a suitable number of instances to replace them.

With the same techniques, we can write a linear approximation of the system's behavior around the equilibrium as:

$$d[\delta s_0] = -\gamma \delta s_0 dt + \gamma \delta s_1 dt + \nu \delta u dt + \sqrt{\lambda} dW_1 - \sqrt{\lambda} dW_2;$$
$$d[\delta s_1] = -\delta s_1 dt + \sqrt{\lambda} dW_3 - \sqrt{\lambda} W_4;$$
$$d[\delta u] = -\nu \delta u dt - \sqrt{\lambda} dW_1 + \sqrt{\lambda} dW_3.$$

As before, W_i are standard Brownian motions accounting for the server spawning and recalling, as well as arrival and departures, respectively. Going into the state space formulation and computing the covariance matrix, we get the following estimate for the variance of the number of idle instances:

$$E[\delta(s_0 - s_1)^2] = \frac{\lambda}{\gamma} + \frac{\lambda}{\nu + 1} \left(\frac{1}{\gamma + 1} + \frac{\nu}{\gamma + \nu} \right).$$

Recall that, for the case of no delay $E[\delta(s_0 - s_1)^2] = \lambda/\gamma$, so the delay increases the variability of the number of idle instances, while not changing its mean value λ/γ. Therefore, one should design the system with a lower value of the recall rate γ to avoid working with zero idle queues and routing randomly. Unfortunately, a simple design strategy such as the one presented for JIQ is not available in this case.

In Figure 5 we plot the evolution of the system, assuming the setting up delay is 10% of the processing time ($\nu = 10$), for the same recalling rate $\lambda/4$ as before. Note that the system again auto scales, but operates with a higher probability of running out of instances. However, the impact on performance is minimal, at the expense of keeping a suitable amount of instances in the process of setup.

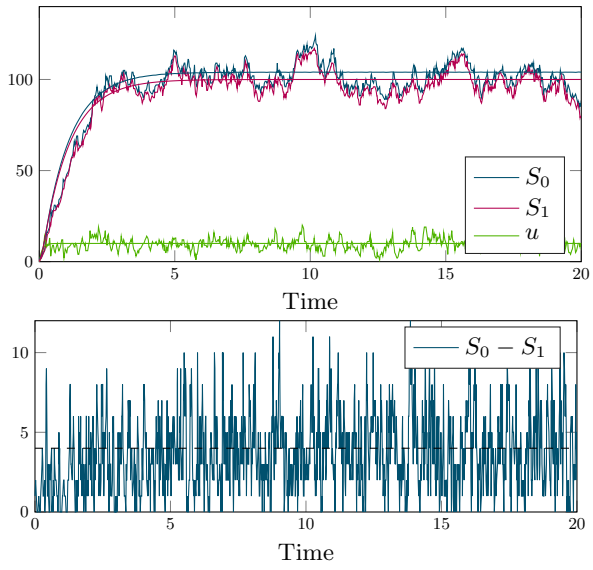


Figure 5: Time evolution of the JIQ system with scaling feedback and creation delay, for $\lambda = 100$, $\mu = 1$, $\nu = 10$ and $\gamma = \lambda/4$. Above: total number of instances S_0, currently working S_1 and setting up U and their fluid limit. Below, number of idle instances in the system.

4. SIZING LONG TERM RESERVATIONS

Finally, let us analyze the problem of long term reservations: typically cloud instances can be purchased at a *reserved* price with a cost p_r (per unit of time). These reserved instances are available at all times. If extra capacity is needed, instances can be spawned on-demand at a cost p_d, where typically $p_d > p_r$, otherwise there would be no point in making reservations. The problem from a customer perspective is to determine how many reserved instances should be purchased. In steady state, the cost incurred per unit of time is given by:

$$C(N; p_r, p_d) = p_r N + p_d E\left[(S(t) - N)^+\right].$$

where $S(t)$ is the total number of active instances.

The general long term reservation problem is therefore:

$$\min_N p_r N + p_d E\left[(S - N)^+\right] \tag{5}$$

where S is the number of active instances in steady state, and is policy dependent.

As a first example, consider the idealized case where instances are created immediately for each arriving job, and thus the system behaves as an infinite server queue. In that case, $S = (n - N)^+$ an the above problem becomes.

$$\min_N p_r N + p_d E\left[(n - N)^+\right] \tag{6}$$

Note that $E[(n - N)^+] = \int_N^\infty (n - N) dF_n(n)$, where $F_n(n)$ is the steady-state distribution of the number of jobs in the system. Furthermore:

$$\frac{\partial}{\partial N} E[(n - N)^+] = -\int_N^\infty dF_n(n) = -(1 - F_n(N)).$$

Differentiating the cost function in (6) and applying the above result we get the optimality condition:

$$\frac{p_r}{p_d} = 1 - F_n(N) \tag{7}$$

This optimality condition is well known in economics, in the context of the Newsvendor Problem. In our case, reserving extra capacity incurs a cost whenever this capacity is not fully utilized, and (7) yields the exact tradeoff.

We now specialize this result to the particular policy at hand: since $n(t)$ follows an $M/M/\infty$ queue, in steady state $n(t) \sim \text{Poisson}(\lambda)$ (recall $\mu = 1$). For a large scale system ($\lambda \gg 1$), this can be properly approximated by a Gaussian distribution with mean λ and variance λ. By using this Gaussian approximation we get the optimal number of reserved instances:

$$N^* \approx \lambda + z_{p_r/p_d}\sqrt{\lambda}$$

where z_α is such that $P(Z > z_\alpha) = \alpha$, i.e. the Gaussian right quantile.

For the above long term reservation, the optimal cost is:

$$C^*(\lambda) = p_r\lambda + p_d\sqrt{\lambda}\phi(z_{p_r/p_d})$$

where $\phi(z)$ is the standard Gaussian pdf.

Consider now the JIQ policy with feedback. In that case, the total number of active instances is $S(t) = S_0(t)$ which in steady state is approximately Gaussian with mean and variance $\lambda(1 + 1/\gamma)$. Therefore, proceeding in the same fashion as before, the optimal reservation is:

$$N^* \approx \lambda\left(1 + \frac{1}{\gamma}\right) + z_{p_r/p_d}\sqrt{\lambda\left(1 + \frac{1}{\gamma}\right)}$$

And the optimal cost can be computed as:

$$C^*_{JIQ}(\lambda) = p_r\lambda\left(1 + \frac{1}{\gamma}\right) + p_d\sqrt{\lambda\left(1 + \frac{1}{\gamma}\right)}\phi(z_{p_r/p_d})$$

Note that, while there is an increase in cost due to the extra instances that the system keeps active, choosing γ as in Section 3 this increase in cost can be made small. With a similar procedure, we can also analize the case of instance creation delay.

5. CONCLUSIONS

In this paper, we analyzed the problem of spawning on-demand instances in a cloud computing environment, to serve a given stream of job requests. We studied two settings to scale the number of instances in feedback: a centralized one with queue at the dispatcher, spawning extra instances with the build-up of the queue, and a second one where the dispatcher immediately needs to route the arriving jobs, and thus couples the analysis with traditional decentralized load balancing schemes. In both scenarios we performed fluid an diffusion approximations to evaluate the performance and tradeoffs involved. We also analyzed the case of long term reservations for the policies involved. We are currently exploring the appropriate scaling of the spawning rates to minimize queueing delay while keeping control of the on-demand servers, as well as integrating both the provider and client perspectives in a common optimization framework, as well as considering heterogeneous instances and pricing models.

6. REFERENCES

[1] D. Gamarnik, J. N. Tsitsiklis, and M. Zubeldia. Delay, memory, and messaging tradeoffs in distributed service systems. In *ACM SIGMETRICS 2016, Juan-les-Pins, France*, pages 1–12, 2016.

[2] D. Goldsztajn, A. Ferragut, and F. Paganini. A feedback control approach to dynamic speed scaling in computing systems. In *CISS'17, Baltimore, MD*, 2017.

[3] M. Lin, A. Wierman, L. L. Andrew, and E. Thereska. Dynamic right-sizing for power-proportional data centers. *IEEE/ACM Trans. Netw.*, 21(5):1378–1391, 2013.

[4] Y. Lu, Q. Xie, G. Kliot, A. Geller, J. R. Larus, and A. Greenberg. Join-idle-queue: A novel load balancing algorithm for dynamically scalable web services. *Performance Evaluation*, 68(11):1056–1071, 2011.

[5] M. Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Trans. on Parallel Distrib. Syst*, 12(10):1094–1104, 2001.

[6] D. Mukherjee, S. Dhara, S. Borst, and J. van Leeuwaarden. Optimal service elasticity in large-scale distributed systems. In *ACM SIGMETRICS 2017, Urbana-Champaign, IL*, 2017.

[7] N. D. Vvedenskaya, R. L. Dobrushin, and F. I. Karpelevich. Queueing system with selection of the shortest of two queues: An asymptotic approach. *Problemy Peredachi Informatsii*, 32(1):20–34, 1996.

[8] L. Zheng, C. Joe-Wong, C. G. Brinton, C. W. Tan, S. Ha, and M. Chiang. On the Viability of a Cloud Virtual Service Provider. In *ACM SIGMETRICS 2015, Portland, OR*, pages 235–248, 2016.