

# Basic Auto-scaling Évaluation de Performances 2024, TD #2, INFO4, Polytech Grenoble

Jonatha Anselmi

2024-02-22

## Auto-scaling

Auto-scaling is a powerful mechanism for the efficient management of serverless, edge and cloud computing systems, both from the perspective of user-perceived performance and overall power consumption. It allows the *net* service capacity to scale up or down in response to the current load and the latter distributes incoming jobs across the set of available servers [1,2].

In auto-scaling systems, servers can be in three states: warm, cold and initializing. A server is said *warm* if turned on, *cold* if turned off and *initializing* if making the transition from cold to warm [1]. An initializing server performs basic startup operations such as connecting to database, loading libraries, etc. This is the time to provision a new function instance. Only warm servers are allowed to receive jobs. A server is also said *idle-on* if warm but not processing any job, and *busy* if warm and processing some job. Typically, billing policies charge per number of warm and initializing servers used per time unit.

Our aim is to evaluate the performance of some basic auto-scaling models via simulation in R. More specifically, we will be interested in performance measures related to idle-on servers, which correspond to energy waste.

## M/M/K/K queueing model

A first basic model for autoscaling is given by a  $M/M/K/K$  queue. Here, the interpretation is that  $K$  represents the nominal capacity, i.e., the maximum number of servers that can be up and running. Then, if  $n_t$  is the number of jobs in the system at time  $t$ , then  $n_t$  also represents the number of busy servers. Therefore,  $K - n_t$  may model the number of cold servers, provided that

1. a server becomes cold immediately after processing a job (or equivalently, the expiration rate is very high so that it can be ignored).
2. servers cannot enter the initialization phase (or equivalently, the initialization rate is very high so that it can be ignored).

Therefore, no idle-on server will actually exist within this model.

The  $M/M/K/K$  queue can be described by a continuous-time Markov chain with state space  $S = \{0, 1, \dots, K\}$  and transition matrix  $Q = (q_{i,j})$  where all the non-zero elements are  $q_{i,i+1} = \lambda$  for all  $i = 0, \dots, K-1$  and  $q_{i,i-1} = i\mu$  for all  $i = 1, \dots, K$ . Here,  $\lambda$  is the arrival rate of jobs and  $\mu$  is the service rate of each server.

Let us write a code to simulate the M/M/K/K queue.

```
knitr::opts_chunk$set(echo = TRUE)

simulate_MMKK <- function(lambda, mu, K){

  N=2e5; # number of events to simulate;

  p_reject=0; # blocking probability
```

```

p_reject_transient=rep(0,N); # transient blocking probability
T=0; # simulation time

# a state is the number of jobs (or warm servers) currently in the system
state=0; #initial state

for (i in 1:N) {

  U=lambda+state*mu;
  time_in_state=rexp(1,U)
  T=T+time_in_state;

  if (state==K) {
    # For the moment, let us find the overall time spent in state K
    p_reject=p_reject+time_in_state;
  }

  p_reject_transient[i]=p_reject/T;

  if (runif(1)<lambda/U){
    # An arrival occurred
    if (state<K){
      # The incoming job has room so it is accepted
      state=state+1;
    }
  } else {
    # A departure occurred
    state=state-1;
  }
}

# The blocking probability by simulation
p_reject=p_reject/T;

# The blocking probability by theory (The Erlang B formula)
p_reject_theory=0;
normalizing_constant=0;
for (i in 0:K){
  normalizing_constant = normalizing_constant + (lambda/mu)**i / factorial(i);
}

p_reject_theory = (lambda/mu)**K / factorial(K);
p_reject_theory = p_reject_theory / normalizing_constant;

# Print the reject probabilities (simulation and theory)
events=1:N;
plot(events, p_reject_transient, type = "l", lty = 1)
abline(h = p_reject_theory, col="red")
legend("bottomright", legend = c("Reject prob. by simulation",
  "Reject prob. by theory - Erlang B formula"), col=1:2, pch=1);

print(paste("Reject probability: ",p_reject," (by simulation)"));

```

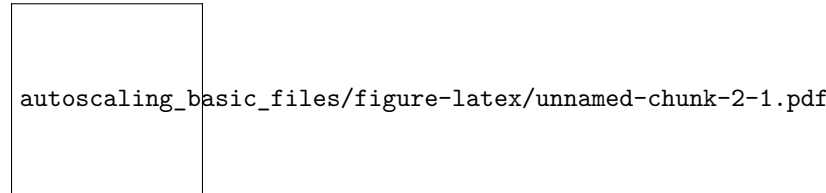
```
print(paste("Reject probability: ",p_reject_theory," (by theory - Erlang B formula)"));
}
```

Let's simulate.

```
knitr::opts_chunk$set(echo = TRUE)

set.seed(13); # Set seed for reproducibility
K=50;         # number of servers
lambda=0.6*K; # arrival rate
mu=1;         # service rate

simulate_MMKK(lambda,mu,K);
```



```
## [1] "Reject probability: 0.00023858477525209 (by simulation)"
## [1] "Reject probability: 0.000220944324998379 (by theory - Erlang B formula)"
```

### M/M/K/K queueing model + expiration rate

Let us now go further. Building on the M/M/K/K queue, we want to add the feature that when a server becomes idle-on, it goes cold by itself after a (non-negligible) scale down delay, or *expiration time*, provided that during such time the server received no job. This scale-down rule is common in serverless computing platforms.

By simulation, our goal here is to

1. calculate the fraction of jobs that find no idle-on server when they arrive, which we will refer to as  $p_{\text{wait}}$ .
2. evaluate the stationary distribution of the number of idle-on servers.

Let us first model this setting. We define a continuous-time Markov chain with state space  $S = \{(i, j) \in \mathbb{N}^2 : i + j \leq N\}$  where  $i$  resp.  $j$  represents the number of busy resp. idle-on servers. The transition rate matrix  $Q = (q_{i,j})$  where, from a generic state  $(i, j)$ , the non-zero rates are

$$\begin{cases} q_{(i,j),(i+1,j-1)} = \lambda & \text{if } i < K \text{ and } j > 0 \\ q_{(i,j),(i,j)} = \lambda & \text{if } i < K \text{ and } j = 0 \\ q_{(i,j),(i-1,j+1)} = i\mu \\ q_{(i,j),(i,j-1)} = j\gamma \end{cases}$$

Here,  $\lambda$  is the arrival rate of jobs,  $\mu$  is the service rate of each server and  $\gamma$  is the expiration rate of each server. We are particularly interested in the how often the transition  $(i, 0) \rightarrow (i + 1, 0)$  occurs. In fact, this transition models a job that finds no idle-on server upon arrival, implying that a cold server immediately becomes warm (and busy).

Now, let us write a code that simulates this Markov chain.

```
knitr::opts_chunk$set(echo = TRUE)

simulate_MMKK_plus_expiration <- function(lambda, mu, gamma, K){
```

```

N=2e5; # number of events to simulate;

num_jobs=0;      # overall number of jobs that arrived
num_jobs_wait=0; # overall number of jobs that found no idle-on server upon arrival

p_wait_transient=rep(0,N);
p_dist_idleon=rep(0,K+1); # probability distribution of idle-on servers

T=0; # simulation time

# a state is the pair (busy, idleon).
busy=0; idleon=floor(K/2); #initial state

for (i in 1:N) {

  U=lambda+busy*mu+idleon*gamma;
  time_in_state=rexp(1,U);
  T=T+time_in_state;

  p_dist_idleon[idleon+1]=p_dist_idleon[idleon+1]+time_in_state;

  rnd=runif(1)
  if (rnd<lambda/U){
    # An arrival occurred

    # update performance measures
    num_jobs=num_jobs+1;
    if (idleon==0){
      num_jobs_wait=num_jobs_wait+1;
    }

    if (busy<K) {
      busy=busy+1;
      if (idleon>0) {
        idleon=idleon-1;
      }
    }

  } else if (rnd<(lambda+busy*mu)/U) {
    # A departure occurred
    busy=busy-1;
    idleon=idleon+1;
  } else {
    # A server expiration occurred
    idleon=idleon-1;
  }

  p_wait_transient[i]=num_jobs_wait/num_jobs;
}

p_wait=num_jobs_wait/num_jobs;
p_dist_idleon=p_dist_idleon/T;

```

```

# Print p_wait
print(paste("p_wait = ",p_wait));
events=1:N;
plot(events, p_wait_transient, type = "l", lty = 1);
legend("topright", legend = c("Transient behavior of p_wait"), col=1:1, pch=1);

return(p_dist_idleon);
}

```

Now, let's simulate. If a time unit is 10 milliseconds, which corresponds to the mean service time of jobs in several serverless computing applications, a realistic choice for the expiration rate is  $\gamma = 0.01$  or lower [1,2,3].

```

knitr::opts_chunk$set(echo = TRUE)

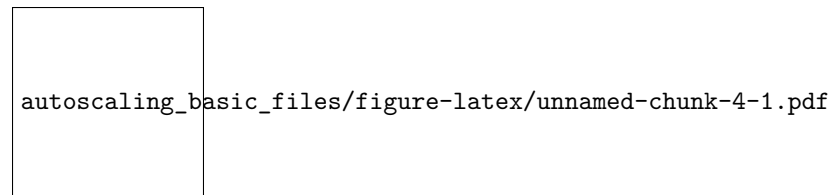
set.seed(10); # Set seed for reproducibility

K=50;          # number of servers
lambda=0.3*K; # arrival rate
mu=1;          # service rate
gamma=0.01;    # expiration rate

p_io<-simulate_MMKK_plus_expiration(lambda,mu,gamma,K);

```

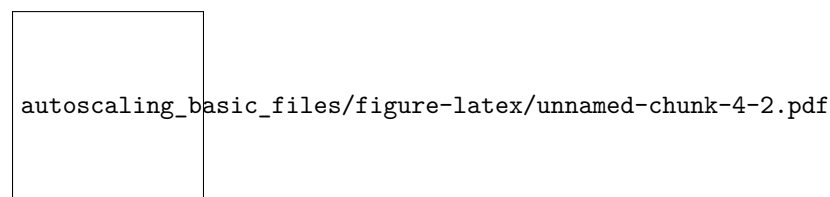
```
## [1] "p_wait = 0.00711483075935013"
```



```

states=0:K;
plot(states, p_io, type = "l", lty = 1, col=1, lwd=2, xlab="Number of idleon servers");
grid(nx = NULL, ny = NULL, lty = 2, col = "lightgray", lwd = 1)

```



```

mean_idleon = sum((0:(length(p_io)-1))*p_io);
var_idleon  = sum( ((0:(length(p_io)-1))*2) * p_io) - mean_idleon**2;
std_idleon  = sqrt(var_idleon);

```

```
cat(paste("mean number of idleon servers = ", mean_idleon, sep=""), '\n');
```

```
## mean number of idleon servers = 9.90629801019302
```

```
cat(paste("variance of idleon servers = ", var_idleon, sep=""), '\n');
```

```
## variance of idleon servers = 17.4789808393413
```

Note the probability mass function of the idle-on servers: does it look familiar? To answer this question, we look for some candidate probability mass functions that may approximate the shape obtained in the

above plot - of course, we can always compute the distribution by solving the global balance equations of the underlying Markov chain, but this is not the goal now. Possible candidates are the binomial and the Poisson distribution. After some numerical attempts, one notice that these models do not really justify the simulation data, so we rule them out. Another alternative is to see whether the empirical distribution fits a Gaussian model. Since the Gaussian model takes negative values, which in principle we do not want to consider, a patch would consist in conditioning over the positive values of the given Gaussian. We now test both of these models.

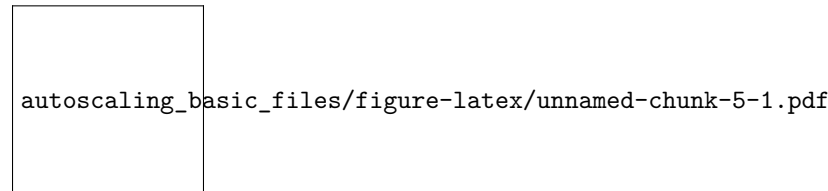
```
knitr::opts_chunk$set(echo = TRUE)
```

```
set.seed(10); # Set seed for reproducibility
```

```
K=50;           # number of servers
lambda=0.3*K;  # arrival rate
mu=1;          # service rate
gamma=0.01;    # expiration rate
```

```
p_io<-simulate_MMKK_plus_expiration(lambda,mu,gamma,K);
```

```
## [1] "p_wait = 0.00711483075935013"
```



```
states=0:K;
plot(states, p_io, type = "l", lty = 1, col=1, lwd=2, xlab="Number of idleon servers");
```

```
mean_idleon = sum((0:(length(p_io)-1))*p_io);
var_idleon  = sum( ((0:(length(p_io)-1))**2) * p_io) - mean_idleon**2;
std_idleon  = sqrt(var_idleon);
```

```
cat(paste("mean number of idleon servers = ", mean_idleon, sep=""), '\n');
```

```
## mean number of idleon servers = 9.90629801019302
```

```
cat(paste("variance of idleon servers = ", var_idleon, sep=""), '\n');
```

```
## variance of idleon servers = 17.4789808393413
```

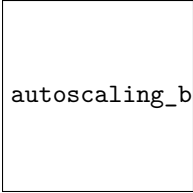
```
# Gaussian model
```

```
s_ <- 0:K; lines(s_, dnorm(s_, mean_idleon, std_idleon),type='l', lty = 2,col=2, lwd=2);
```

```
# Conditioned Gaussian model
```

```
s_ <- 0:K; lines(s_, dnorm(s_, mean_idleon, std_idleon)/(1-pnorm(0, mean_idleon, std_idleon)),type='l',
grid(nx = NULL, ny = NULL, lty = 2, col = "lightgray", lwd = 1)
```

```
legend("topright", legend = c("PMF of idle-on servers", "Gaussian", "Gaussian | Gaussian>0"), col=1:3, p
```



autoscaling\_basic\_files/figure-latex/unnamed-chunk-5-2.pdf

Both of these models seem to well justify the data! Some questions for further investigation:

1. Are these models robust to realistic changes in the model data?
2. Even though both models justify the data, they have a drawback: the parameters of the Gaussian used above depend on the distribution of exactly what we want to model! Can we approximate them simply in terms of the model parameters?

**[HOMEWORK] M/M/K/K queueing model + expiration rate + initialization rate**

The natural step forward is to develop an extended model (with an associated code) that also includes non-negligible initialization rates; say that  $\alpha$  is the rate at which a server goes from cold to warm. How does the state space and transition rate matrix change?

**References**

- [1] N. Mahmoudi and H. Khazaei. “Performance modeling of serverless computing platforms”. IEEE Transactions on Cloud Computing, 2020.
- [2] J. Anselmi “Asynchronous Load Balancing and Auto-scaling: Mean-field Limit and Optimal Design”, IEEE/ACM Transactions on Networking, 2024
- [3] L. Wang, M. Li, Y. Zhang, T. Ristenpart, and M. Swift. “Peeking behind the curtains of serverless platforms”. In Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference, 2018