Homework assignment (DM), Évaluation de Performances, INFO4 Polytech Grenoble

Jonatha Anselmi

2023-02-10

Votre devoir sera rédigé en français ou en anglais sous forme d'un document HTML généré à l'aide de R/Markdown et publié sur rpubs en prenant soin de bien laisser le code apparent et de fixer la graine de votre générateur à l'aide de la fonction set.seed au tout début du document afin qu'il soit possible de reproduire vos données avec exactitude.

Vous enverrez l'url rpubs de votre devoir par mail à jonatha.anselmi AT inria.fr en indiquant dans le sujet [INFO4-EP] DM avant le 2023-03-31 à 12h00.

Scientific context

In the context of decision making under explorable uncertainty, scheduling with testing is a powerful technique used in the management of computer systems to improve performance via better job-dispatching decisions. Before dispatching an incoming job to the right computational resources, a scheduler may run some testing algorithm against the job itself to extract some information about its structure, e.g., its size, and properly classify it. This algorithm takes as input the description of a job and retrieves information by performing operations at various levels of complexity. With an increasing level of complexity, a testing algorithm may consist in guessing a job size by i) analyzing the job input parameters, ii) analyzing the job source code (e.g., looking at the Kolmogorov complexity), iii) running a code optimizer, or iv) running a prediction toolbox constructed from previous data via machine learning mechanisms. The acquisition of such knowledge comes with a cost because the testing algorithm delays the dispatching decisions, though this is under control. The objective here is to analyze the impact of such extra cost in a load balancing setting by investigating the following questions: does it really pay off to test jobs? If so, under which conditions?

Approach

We consider a service system composed of N servers, a sequence of jobs and a scheduler (or dispatcher); see the figure.

Servers

Servers have unlimited buffer, operate with unit speed and adopt the First-in First-out scheduling discipline.

Jobs

Jobs join the system over time following a Poisson process with rate $\Lambda := \lambda N$, with $\lambda < 1$ and each of them leaves the system after service completion at its designated server. Thus, inter-arrival times follow an exponential distribution with rate Λ.

The n-th arriving job has a size given by random variable $X(n)$ and this is the amount of work required by the n-th job on one server, or equivalently its service time at any server since servers operate at unit rate. We assume that the sequence $(X(n))_n$ is independent, identically distributed and also independent of the arrival process. For stability, we assume that $\rho := \lambda\mathbb{E}[X(n)] < 1$.

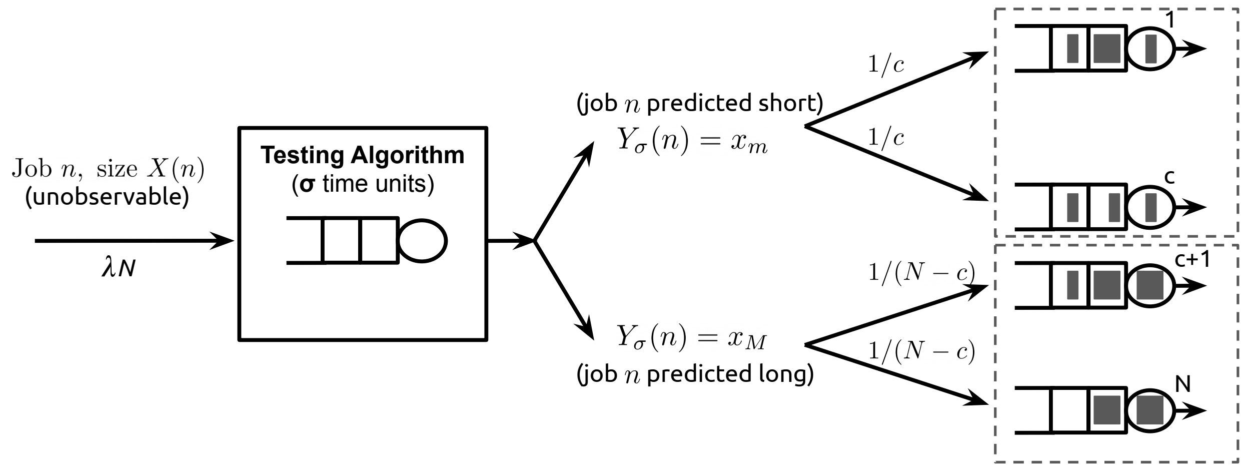

Figure 1: Notation and architecture for load balancing with job-size testing. It is assumed that $c \in \mathbb{N}$.

We also assume that jobs are classified as either *long*, i.e., of size x_M, or *short*, i.e., of size x_m, with $0 < x_m < x_M < \infty$. Thus, $\mathbb{P}(X(n) = x_m) + \mathbb{P}(X(n) = x_M) = 1$ for all n. We will be interested in applying our results to the case where job sizes have a "heavy-tailed" distribution in the sense that

$$\mathbb{P}(X(n) = x_M) = \alpha\, x_M^{-\beta}, \quad \alpha > 0,\, \beta > 0,$$

with x_M "large".

Scheduler

When jobs arrive, they immediately join a scheduler, which is in charge of dispatching each job to exactly one queue. In order to make such decisions, it is allowed to execute a *testing algorithm* against each job. This algorithm extracts some information that is exploited to make a prediction about its actual size. The *testing time*, i.e. the amount of time dedicated to the execution of the testing algorithm, is deterministic and under the control of the system manager. We assume that the same testing time is applied to all jobs, say $\sigma \geq 0$.

We model the scheduler as an $M/M/1$ queue with arrival rate Λ and mean service time σ. This is meant to model congestion and to capture the impact of the runtime phenomena happening during the execution of the testing algorithm, which stochastically perturb the service times,

We let $Y_\sigma(n)$ denote the $\{x_m, x_M\}$-valued random variable associated to the prediction of the exact (unknown) size, $X(n)$, of the n-th arriving job when the testing algorithm is executed for σ time units. We assume that the prediction of the n-th job $Y_\sigma(n)$ can only depend on $X(n)$ and σ. With a slight abuse of notation, (X, Y_σ) denotes an auxiliary random vector having the same distribution of the job size and prediction pair $(X(n), Y_\sigma(n))$.

To summarize the information available at the scheduler:

- The scheduler knows λ, N and the distribution of the job size X.
- For all job n, it does not know the exact job size $X(n)$ but can execute a testing algorithm for σ time units to obtain a prediction, $Y_\sigma(n)$, of the size of job n.
- Finally, the scheduler knows the joint probability mass function of (X, Y_σ) for all $\sigma \geq 0$, which in practice can be learned from the data.

Dispatching Policy

To reduce the interference between short and long jobs, for job dispatching we assume that there exists a real number $c \in [0, N]$ such that all jobs that are *predicted* as short ($Y_\sigma(n) = x_m$) resp. long ($Y_\sigma(n) = x_M$) are sent uniformly at random to servers $1, \ldots, \lfloor c \rfloor + 1$ resp. $\lfloor c \rfloor + 1, \ldots, N$. We will refer to c as {cutoff server}.

Thus, conditioned on $Y_\sigma = x_m$ resp. $Y_\sigma = x_M$, a job is sent to server $i = 1, \dots, \lfloor c \rfloor$ resp. $\lfloor c \rfloor + 2, \dots, N$ with probability $1/c$ resp. $1/(N-c)$ and to server $\lfloor c \rfloor + 1$ with probability $1 - \lfloor c \rfloor / c$ resp. $1 - (N - 1 - \lfloor c \rfloor)/(N-c)$. In particular, note that i) server $\lfloor c \rfloor + 1$ is the only server that can receive jobs that are predicted as short or long and that this happens only if c is not an integer, ii) if $c < 1$ resp. $c > N-1$, then no server that only processes jobs predicted as short resp. long exists.

Testing Algorithm: Quality of Prediction vs Running Time

The quality of job-size predictions depends on the amount of processing time dedicated to the execution of the testing algorithm σ. After testing, we assume that the scheduler disposes of a probability distribution about the exact size that depends on σ. We define the connection between the quality of predictions and running time through the *profile matrix*

$$P_{x,y}(\sigma) := \mathbb{P}(X = x \cap Y_\sigma = y), \quad x, y \in \{x_m, x_M\},\, \sigma \geq 0$$

and $P(\sigma) := [P_{x,y}(\sigma) : x, y \in \{x_m, x_M\}]$, i.e., the joint probability mass function of (X, Y_σ). In practice, this matrix can be learned from the data by user profiling methods. For now, we do not impose a specific structure on $P(\sigma)$.

Performance Measure

The performance measure of interest is the mean waiting time of jobs, say $D_c^{(N)}(\sigma)$. This is the sum of the mean delay (waiting time + service time) at the scheduler and the mean waiting time at the servers. Let us refer to the mean waiting time at the servers by $R_c^{(N)}(\sigma)$.

Within the given set of dispatching policies, $R_c^{(N)}(\sigma)$ corresponds to the mean waiting time of three parallel M/G/1 queues because i) the output process of an M/M/1 queue, i.e., of the scheduler, is Poisson (by Burke's theorem) and ii) the thinning of a Poisson process produces again a Poisson process,

$$\begin{aligned} R_c^{(N)}(\sigma) = \frac{\Lambda}{2} \frac{\mathbb{P}(Y_\sigma = x_m)^2\, p_m^2 \mathbb{E}[X^2 \mid Y_\sigma = x_m]}{\lfloor c \rfloor - \Lambda \mathbb{P}(Y_\sigma = x_m) p_m\, \mathbb{E}[X \mid Y_\sigma = x_m]} \mathbb{I}_{c \geq 1} + \frac{\Lambda}{2} \frac{p_z^2 \mathbb{E}[Z^2]}{1 - \Lambda p_z\, \mathbb{E}[Z]} \\ + \frac{\Lambda}{2} \frac{\mathbb{P}(Y_\sigma = x_M)^2 p_M^2 \mathbb{E}[X^2 \mid Y_\sigma = x_M]}{N - 1 - \lfloor c \rfloor - \Lambda \mathbb{P}(Y_\sigma = x_M) p_M\, \mathbb{E}[X \mid Y_\sigma = x_M]} \mathbb{I}_{c \leq N-1} \end{aligned}$$

provided that $\Lambda \mathbb{P}(Y_\sigma = x_m) p_m\, \mathbb{E}[X \mid Y_\sigma = x_m] < \lfloor c \rfloor$, $\Lambda p_z\, \mathbb{E}[Z] < 1$ and $\Lambda \mathbb{P}(Y_\sigma = x_M) p_M\, \mathbb{E}[X \mid Y_\sigma = x_M] < N - 1 - \lfloor c \rfloor$ and infinity otherwise. In the previous expression,

$$p_m = \frac{\lfloor c \rfloor}{c} \mathbb{I}_{c \geq 1},\ p_M = \frac{N - 1 - \lfloor c \rfloor}{N - c} \mathbb{I}_{c \leq N-1},\ p_z = \mathbb{P}(Y_\sigma = x_m)\,(1 - p_m) + \mathbb{P}(Y_\sigma = x_M)\,(1 - p_M)$$

and Z is an auxiliary random variable equal to $X \mid Y_\sigma = x_m$ with probability $\mathbb{P}(Y_\sigma = x_m)(1 - p_m)/p_z$ and to $X \mid Y_\sigma = x_M$ otherwise.

Note that, if $p_z > 0$, the three summation terms in $R_c^{(N)}(\sigma)$ refer to the mean waiting time in the groups of servers $\{1, \dots, \lfloor c \rfloor\}$, $\{\lfloor c \rfloor + 1\}$ and $\{\lfloor c \rfloor + 2, \dots, N\}$, respectively.

Homework

Step 0: Preliminaries

1. Identify the set of testing times that make the scheduler stable (finite mean response time).
2. Provide a mathematical formula for the mean delay (waiting + service time) of jobs at the scheduler.
3. For $p = 1, 2$, provide a mathematical formula for $\mathbb{E}[X^p \mid Y_\sigma = x_m]$ as a function of the profile matrix $P_{x,y}(\sigma)$, $x, y \in \{x_m, x_M\}$ and $\sigma \geq 0$.

Step 1: Code writing

Write a code in R to simulate the job dynamics of the system described above. Towards this purpose, you can use/modify the code that we have written during the "Job dispatching among parallel servers" TD session, reported below for your convenience.

Step 2: Validation

Assume that $N = 10$, $\rho = 0.5$, $x_m = 1$, $x_M = 10^6$, $\alpha = \beta = 1$ and

$$\begin{aligned}
P_{x_m,x_m}(\sigma) &= (1 - e^{-10\sigma})(\mathbb{P}(X = x_m) - P_{x_m,x_m}(0)) + P_{x_m,x_m}(0)\\
P_{x_M,x_M}(\sigma) &= (1 - e^{-\sigma})(\mathbb{P}(X = x_M) - P_{x_M,x_M}(0)) + P_{x_M,x_M}(0)\\
P_{x_m,x_M}(\sigma) &= \mathbb{P}(X = x_m) - P_{x_m,x_m}(\sigma)\\
P_{x_M,x_m}(\sigma) &= \mathbb{P}(X = x_M) - P_{x_M,x_M}(\sigma).
\end{aligned}$$

Show that the mean waiting time at the servers obtained by simulation corresponds to the mathematical formula above ($R_c^{(N)}(\sigma)$); use that $\mathbb{P}(Y_\sigma = x_m) = \mathbb{P}(Y_\sigma = x_m \cap X = x_m) + \mathbb{P}(Y_\sigma = x_m \cap X = x_M) = P_{x_M,x_m}(\sigma) + P_{x_m,x_m}(\sigma)$.

Step 3: Performance evaluation by simulation

Within the same parameter setting used for the previous step:

1. Make a plot of $\min_{c\in[0,N]} D_c^{(N)}(\sigma)$ as a function of σ.
2. Is it possible to find $\sigma > 0$ such that it is convenient to test job sizes (with respect to the situation where jobs run untested)?

Appendix (Job dispatching among parallel servers)

We consider a system composed of K servers working in parallel, each with its own queue. In the following, "queues" and "servers" are synonyms. Queues have an infinite buffer and operate under the First-In First-Out (FIFO) scheduling discipline. Server k operates with speed μ_k. Jobs join the network via an exogenous process with rate λ and are dispatched to queue k according to some dispatching algorithm; see below. After processing, jobs leave the system and never return.

Job sizes are independent and classified as either long, i.e., of size x_M, or short, i.e., of size x_m, with $0 < x_m < x_M < \infty$. Let X denote the random variable of the job sizes. We also assume that job sizes have a "heavy-tailed" distribution in the sense that

$$\mathbb{P}(\text{Long job}) = \mathbb{P}(X = x_M) = \alpha\, x_M^{-\beta}, \quad \alpha > 0,\ 0 \le \beta \le 1$$

with x_M "large". Here, the tail of X decays polynomially and we will be particularly interested in the case where $\beta \le 1$ as this implies high variance when x_M grows; say $x_m = 1$ and $x_M = 10^3$.

We are interested in evaluating the performance induced by the following dispatching algorithms:

1. **Random (RND)**: Upon arrival, each job is dispatched to queue i with probability $1/K$ independently of all else.
2. **Round-Robin (RR)**: Upon arrival, the n-th job is sent to queue $1 + (n \bmod K)$.
3. **Least Loaded (LL-d)**: Upon arrival, each job is dispatched to the queue with the shortest workload among d selected uniformly at random.
4. **Size Interval Task Allocation (SITA-c)**: Upon arrival, a short resp. job is sent uniformly at random among servers $\{1, \dots, c\}$ resp. $\{c + 1, \dots, K\}$, for some c.

The performance metric of interest is the waiting time, i.e., the average time that jobs spend waiting before receiving service.

The following code simulates the dynamics induced by the dispatching algorithms mentioned above.

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

LB_random <- function(N,K,mu,InterArrival,JobSize) {

  WaitingTime=rep(0,K);
  WaitingTimeCum=rep(0,K);

  for (n in 1:N) #for each job
  {
    # Dispatching decision of RANDOM
    U = sample(x=1:K,1);
    for (k in 1:K) #for each server
    {
      # Update the waiting time of server k (Lindley's recursion)
      WaitingTime[k] = max(WaitingTime[k] + ifelse(U==k,1,0)*JobSize[n]/mu[k] - InterArrival[n], 0);
    }
    WaitingTimeCum[U] = WaitingTimeCum[U]+WaitingTime[U];
  }

 return(sum(WaitingTimeCum)/N);
}

LB_RR <- function(N,K,mu,InterArrival,JobSize) {

  WaitingTime=rep(0,K);
  WaitingTimeCum=rep(0,K);

  for (n in 1:N) #for each job
  {
    # Dispatching decision of Round Robin
    U = n%%K;
    U = U+1;
    for (k in 1:K) #for each server
    {
      # Update the waiting time of server k (Lindley's recursion)
      WaitingTime[k] = max(WaitingTime[k] + ifelse(U==k,1,0)*JobSize[n]/mu[k] - InterArrival[n], 0);
    }
    WaitingTimeCum[U] = WaitingTimeCum[U]+WaitingTime[U];
  }

 return(sum(WaitingTimeCum)/N);
}

LB_LLd <- function(N,K,mu,InterArrival,JobSize,d) {

  WaitingTime=rep(0,K);
  WaitingTimeCum=rep(0,K);

  for (n in 1:N) #for each job
  {
    # Dispatching decision of Least Loaded - d
    s=sample(x=1:K, d, replace = F);
    k_win=s[which.min(WaitingTime[s])];
    for (k in 1:K) #for each server
    {
      # Update the waiting time of server k (Lindley's recursion)
      WaitingTime[k]=max(WaitingTime[k]+ifelse(k_win==k,1,0)*JobSize[n]/mu[k_win]-InterArrival[n],0);
    }
    WaitingTimeCum[k_win] = WaitingTimeCum[k_win]+WaitingTime[k_win];
  }

 return(sum(WaitingTimeCum)/N);
}

LB_SITAc <- function(N,K,mu,InterArrival,JobSize,c,xm) {

  WaitingTime=rep(0,K);
  WaitingTimeCum=rep(0,K);

  for (n in 1:N) #for each job
  {

    # Dispatching decision of SITA-c
    k_win=sample(x=(c+1):K);
    if(JobSize[n]==xm)
    {
      # job n is short
      k_win=sample(x=1:c);
    }

    for (k in 1:K) #for each server
    {
      # Update the waiting time of server k (Lindley's recursion)
      WaitingTime[k]=max(WaitingTime[k]+ifelse(k_win==k,1,0)*JobSize[n]/mu[k_win]-InterArrival[n],0);
    }
    WaitingTimeCum[k_win] = WaitingTimeCum[k_win]+WaitingTime[k_win];
  }

 return(sum(WaitingTimeCum)/N);
}

################################################################################
N=2e6; # number of jobs to simulate
K=50;  # number of servers
mu = sample(x=1:1, K, replace = TRUE); # service rates

set.seed(17); # Set seed for reproducibility

beta=0.8;
xM=1e3;
xm=1;
JobSize=sample(c(xm,xM),N,replace=T,prob=c(1-1/xM^beta,1/xM^beta)); # Job sizes

EX=xm*(1-1/xM^beta) + xM^(1-beta); # Expected value of job sizes

B=10;
```

```r
xdata=(0.5:(B-0.5))/B;
W_RANDOM=rep(0,length(xdata));
W_RR=rep(0,length(xdata));
W_LLd=rep(0,length(xdata));

cnt=1;
for (rho in xdata)
{
  # For stability
  lambda=K * rho/EX; #"overall arrival rate"<"overall service rate"=(mu[1]+...+ mu[K])/EX
  InterArrival = rexp(n=N, rate = lambda);    # Inter-arrival times

  W_RANDOM[cnt]=LB_random(N,K,mu,InterArrival,JobSize);
  W_RR[cnt]=LB_RR(N,K,mu,InterArrival,JobSize);
  W_LLd[cnt]=LB_LLd(N,K,mu,InterArrival,JobSize,2);

  print(paste("Waiting Times with rho =",format(round(rho, 2), nsmall = 2),"and K =",K))

  print(paste("Random: ", W_RANDOM[cnt]))
  print(paste("RR: ", W_RR[cnt]))
  print(paste("LL-d: ", W_LLd[cnt]))

  EX2=xm*xm*(1-1/xM^beta) + xM^(2-beta);
  W=(0.5*lambda/K)*EX2/(1-rho);
  print(paste("Random (Theory): ", W))


  #c=floor(K/2); # making sure that c is an integer
  #print(paste("SITA-c: ", LB_SITAc(N,K,mu,InterArrival,JobSize,c,xm)))

  cnt=cnt+1;
  cat("\n")
}

length(xdata)
xdata
length(W_RANDOM)
plot(xdata, W_RANDOM, xlim=c(0,1), ylim = c(0, max(W_RANDOM)*1.1), type="o",
     col="blue", pch="o", lty=1, ylab="",xlab="rho",
     main=paste("Average Waiting Times - K=",K))
points(xdata, W_RR, col="red", pch="*")
lines(xdata, W_RR, col="red",lty=2)
points(xdata, W_LLd, col="dark red", pch="*")
lines(xdata, W_LLd, col="dark red",lty=2)
legend("topleft",legend=c("RANDOM","RR","LL2"), col=c("blue","red","black"),
                                   pch=c("o","*","+"),lty=c(1,2,3), ncol=1)
```