Advanced Topics on Cloud Computing **Spring 2021**

Lecture 3: Load Balancing

Lecturer: Dr. Georgios Paschos *Amazon*

This lecture focuses on *Load Balancing*: jobs arrive at a cloud computing system, and we must select for each one an available server. What are the best practices for routing these jobs?

3.1 Offline Load Balancing: Makespan Minimization

We begin by modeling the scheduling of a fixed set of jobs in a multiprocessor environment. The problem statement is: "Given J jobs, where job j has processing length l_j, and S identical servers, what is the minimum possible time required to finish execution of all jobs?" There are several constraints:

- A server can only work on one job at a time.
- A job, once started, must run to completion.

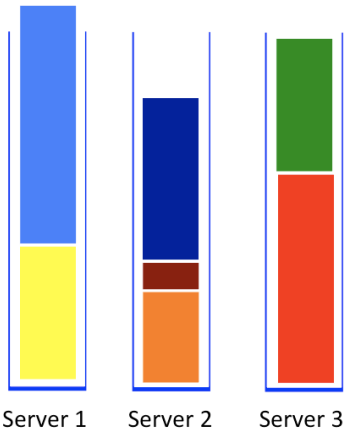

Figure 3.1: Assignment of jobs to servers.

3.1.1 Problem formulation and complexity

Let x_{js} be 1 if job j is assigned to server s and 0 otherwise. The total length of jobs at server s is called *makespan* of server s, and is equal to:

$$T_s = \sum_{j=1}^{J} x_{js} l_j.$$

Define also an auxiliary variable t which will play the role of the maximum total length. The multiprocessor scheduling problem can be defined as:

Minimum makespan problem:

$$\min_{x,t} t \tag{3.1}$$
$$\text{s.t. } T_s = \sum_{j=1}^{J} x_{js} l_j, \quad \forall s$$
$$t \geq T_s, \quad \forall s.$$

3.1.2 Greedy approximation

The Greedy algorithm inspects the jobs one by one, and assigns it to the server with the least length so far.

Theorem 3.1 *Suppose t^* is the minimum value of (3.1), and t^G the makespan of Greedy. We have:*

$$t^G \leq 2t^*.$$

Proof: We first observe that the minimum makespan is larger than maximum job length $t^* \geq \max_j l_j$. Also, the LP relaxation is a lower bound, hence, $t^* \geq \frac{1}{S}\sum_{j=1}^{J} l_j$.

W.l.o.g. assume server s has maximum load in Greedy, and j was the last assigned job in that server. Every other server must have at least as much total length as $T_s - l_j$, hence $T_s - l_j \leq \frac{1}{S}\sum_{j=1}^{J} l_j$. It then follows:

$$\begin{aligned} t^G = T_s &= (T_s - l_j) + l_j \\ &\leq \frac{1}{S}\sum_{j=1}^{J} l_j + l_j \\ &\leq 2t^* \end{aligned}$$

■

The above algorithm can improve to a 4/3–approximation by ordering first the jobs in a decreasing length. Finally, although the problem is NP-hard, there exist Fully Polynomial Time Approximation Schemes (FPTAS) for this problem [6].

3.2 Online Load Balancing

We consider a system with S servers and one load balancer, as in Fig. 3.2. Time is slotted $t = 1, 2, \ldots$. Server s can process μ_s number of jobs per slot. At slot t the load balancer receives $A(t)$ new jobs; $A(t) \in \mathbb{N}$ is i.i.d., with $\mathbb{E}[A(t)] = a$ and $\mathbb{E}[A^2(t)] < \infty$. The ***load balancing policy*** selects any one server from $\{1, \ldots, S\}$ to *route* the arriving jobs. Let $d_s^\pi(t) = 1$ if $A(t)$ jobs are added to server s.

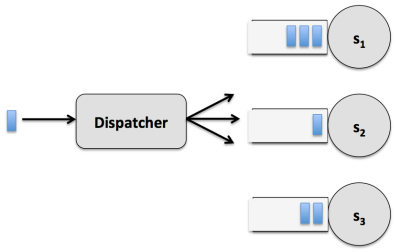

Figure 3.2: Load Balancing system for $S = 3$ servers.

To track the remaining number of jobs at server s, we can use the queue-update equation:

$$Q_s(t+1) \leq [Q_s(t) - \mu_s]^+ + A(t)d_s^\pi(t). \tag{3.2}$$

where *queue length* $Q_s(t)$ expresses the remaining work of server i and constitutes a Markov process. We use the notation $[.]^+ = \max(0, .)$. We say that server s is *strongly stable* if:[1]

$$\lim_{T\to\infty} \frac{\sum_{t=1}^{T} Q_s(t)}{T} = 0.$$

The queue length of a server $Q_s(t)$ is indicative of the delay that jobs experience when routed to that server. If the server is strongly stable, its queue length may grow arbitrarily large, but it also becomes empty frequently and the job delay is finite. Hence, a first-order performance indicator of a load balancing policy is to keep all servers strongly stable.

3.2.1 Load Balancing Capacity

The capacity of a load balancing system is a value $a_{\max}$ such that for any $a < a_{\max}$ we may guarantee the existence of a load balancing policy that can keep all servers stable.

Theorem 3.2 (Capacity Upper Bound) *Suppose $a > \sum_{s=1}^{S} \mu_s$, and consider any load balancing policy, the system is unstable. Therefore, $\sum_{s=1}^{S} \mu_s$ is an upper bound on capacity.*

Proof: Fix any $\epsilon > 0$ such that $a = \sum_{i=1}^{S} \mu_i + \epsilon$. First, sum up telescopically (3.2) from $t = 0$ up to T:

$$Q_s(T) \geq Q_s(0) + \sum_{t=1}^{T} A(t)\mathbb{1}_{\{s=d(t)\}} - \sum_{t=1}^{T} \mu_s,$$

where inequality follows from not applying the $[.]^+$ operator. Then assuming the system starts empty ($Q_s(0) = 0$), summing up for all servers, we have for all T:

$$\sum_{s=1}^{S} Q_s(T) \geq \sum_{t=1}^{T} \left(A(t) - \sum_{s=1}^{S} \mu_s \right).$$

Summing up to T, dividing by T and taking limit, we obtain:

$$\lim_{T\to\infty} \frac{\sum_{s=1}^{S} Q_s(T)}{T} \geq \lim_{T\to\infty} \frac{\sum_{t=1}^{T} \left(A(t) - \sum_{s=1}^{S} \mu_s \right)}{T} = a - \sum_{s=1}^{S} \mu_s = \epsilon > 0.$$

Since no queue can be made negative, there must be at least one s for which $\lim_{T\to\infty} \frac{\sum_{t=1}^{T} Q_s(t)}{T} > 0$. ■

3.2.2 Join the Shortest Queue

The algorithm *Join the Shortest Queue (JSQ)* routes the newly arrived jobs to the server with the smallest queue:

$$d_s^{JSQ}(t) \in \arg\min_{s\in\{1,\ldots,S\}} Q_s(t).$$

Theorem 3.3 (JSQ is maximally stable.) *If the mean traffic satisfies $a < \sum_{i=1}^{S} \mu_s$, and the jobs are dispatched according to JSQ, then the system is strongly stable.*

Proof: First, there exist a stationary probabilistic allocation d_s^* and $\epsilon > 0$ such that $ad_s^* = \mu_s - \epsilon$. Define the Lyapunov function to be the squared norm of the queue length vector $L(x) = \frac{1}{2}\sum_s x_s^2$. The Lyapunov drift is the conditional evolution of the Lyapunov function:

$$\Delta(t) = \mathbb{E}[L(Q(t+1)) - L(Q(t))|Q(t)] = \frac{1}{2}\sum_s \mathbb{E}[Q_s^2(t+1) - Q_s^2(t)|Q(t)]$$

[1] A technical note: load balancing policies may not be stationary, hence the induced queue lengths may not have a limit. In such cases, we must replace the limit in the definition of stability with the limit of supremum, see [3] for a detailed treatment. In this lecture, we restrict admissible policies to be stationary; it has been shown that in most situations, restricting to stationary policies does not compromise optimality.

Using (3.2) we may infer that

$$Q_s^2(t+1) - Q_s^2(t) \leq \mu_s^2 + A^2(t)d_s^{JSQ}(t) - 2Q_s(t)(\mu_s - A(t)d_s^{JSQ}(t))$$

Taking conditional expectation, we obtain:

$$\begin{aligned}\Delta(t) &= \frac{1}{2}\sum_s \mathbb{E}[Q_s^2(t+1) - Q_s^2(t)|Q(t)] \\ &= \frac{\sum_s \mu_s^2 + a^2}{2} - \sum_s Q_s(t)(\mu_s - ad_s^{JSQ}(t)).\end{aligned}$$

Finally, observe that JSQ is minimizing the term $\sum_s Q_s(t)d_s^{JSQ}(t)$. Hence, we can write:

$$\sum_s Q_s(t)ad_s^{JSQ}(t) \leq \sum_s Q_s(t)ad_s^* = \sum_s Q_s(t)(\mu_s - \epsilon)$$

where $d_s^*(t)$ is the decision of the stationary randomized policy that splits flow a to servers such that $ad_s^* = \mu_s - \epsilon$. Combining we obtain:

$$\begin{aligned}\Delta(t) &= \frac{\sum_s \mu_s^2 + a^2}{2} - \sum_s Q_s(t)(\mu_s - ad_s^{JSQ}(t)) \\ &\leq B - \epsilon\sum_s Q_s(t).\end{aligned}$$

where B is constant larger than $(\sum_s \mu_i^2 + a^2)/2$. Hence, whenever $\sum_i Q_i(t) > B/\epsilon$, the Lyapunov drift is negative, which by Lyapunov theorem implies that the Markov chain is stable. ■

What is remarkable about JSQ is that this simple Greedy rule can stabilize the server system whenever the arrivals are less than the capacity, without knowing the value of the randomness of the arrivals.

The above framework generalizes to (1) multiple load balancers interconnnected to servers via a bipartite graph, or (2) a network of queues, see the backpressure algorithm [1]. The work in [2] extends JSQ to respond to overload. See [3] for a framework that extends the maximal stability to optimization, and [4] for analysing the asymptotic behaviour of JSQ in fluid and diffusion limits.

3.3 The power of 2 choices

Despite the simplicity of JSQ, many server systems are so large and so fast, that in fact checking the queue state of each server whenever a new job arrives may harm their performance. In this section, we discuss an unexpected result due to [5], saying that we can limit our checks to only two server queues.

First, consider an experiment where we have n balls and each one is thrown randomly and independently at one of the available m bins.

Theorem 3.4 (Balls into bins) *With high probability (w.h.p.) the maximum number of balls in a bin is* $\mathcal{O}(\frac{\log n}{\log\log n})$.

Proof: Applying union bound, we can characterize the probability of the overload event $B \geq k$:"there is a server with load more than k":

$$\mathbb{P}(B \geq k) \leq \sum_{i=1}^{m} \mathbb{P}(B_i \geq k) = m\mathbb{P}(B_i \geq k),$$

where B_i is the number of balls in bin i, which is binomially distributed, $\mathbb{P}(B_i = j) = \binom{m}{j}\left(\frac{1}{m}\right)^j\left(1 - \frac{1}{m}\right)^{n-j}$. Using

Sterling's approximation (a):

$$\begin{aligned}
\mathbb{P}(B_i \geq k) &= \sum_{j=k}^{m} \binom{m}{j} \left(\frac{1}{m}\right)^j \left(1-\frac{1}{m}\right)^{n-j} \\
&\leq \sum_{j=k}^{m} \binom{m}{j} \left(\frac{1}{m}\right)^j \\
&\overset{(a)}{\leq} \sum_{j=k}^{m} \left(\frac{me}{j}\right)^j \left(\frac{1}{m}\right)^j = \sum_{j=k}^{m} \left(\frac{e}{j}\right)^j \leq \sum_{j=k}^{m} \left(\frac{e}{k}\right)^j = \sum_{j=0}^{m-k} \left(\frac{e}{k}\right)^{j+k} \\
&\leq \left(\frac{e}{k}\right)^k \sum_{j=0}^{\infty} \left(\frac{e}{k}\right)^j = \left(\frac{e}{k}\right)^k \frac{1}{1-e/k} = \Theta\left(\frac{1}{k^k}\right)
\end{aligned}$$

In order to obtain our scaling law, we would like to choose some k for which $\mathbb{P}(B \geq k) \leq \frac{m}{n^t}$, which we could achieve using $1/k^k \leq 1/n^t$, or $k \log k \geq t \log n$. Pick $t > 1$ and $k = \frac{t \log n}{\log \log n}$, we then have that:

$$k \log k = \frac{t \log n}{\log \log n}(\log t \log n + t \log \log n - \log \log \log n) \geq t \log n(t + \log t \log n) \geq t \log n$$

This proves that:

$$\mathbb{P}\left(B \geq \frac{t \log n}{\log \log n}\right) = \mathcal{O}\left(\frac{m}{n^t}\right), \text{ for any constant } t > 1,$$

and hence the probability becomes arbitrarily small for large n proving the result. ■

Then, repeat the same experiment, but this time, place each ball by picking two random bins, and assigning the ball to the one with the least number of balls.

Theorem 3.5 (Power of 2 choices) *With high probability (w.h.p.) the maximum number of balls in a bin is* $\mathcal{O}(\log \log n)$.

For proof see [5]. The remarkable observation is that the two random choices provide an exponentially better load balancing result. This observation inspired many load balancing algorithms of the type: "*Upon the arrival of a job, sample at random d servers, and route the job to the server with the smaller queue length*".

References

[1] L. Tassiulas and A. Ephremides, *Stability Properties of Constrained Queueing Systems and Scheduling Policies for Maximum Throughput in Multihop Radio Networks*, IEEE Trans. On Automatic Control, 1992.

[2] C.-P. Li, G. Paschos, L. Tassiulas, and E. Modiano, *Dynamic Overload Balancing in Server Farms*, IFIP Networking, 2014.

[3] M. Neely, *Stochastic network optimization with application to communication and queueing systems*, Synthesis Lectures on Communication Networks, 2010.

[4] M. van der Boor and S. Borst, *Scalable load balancing in networked systems: A survey of recent advances*, arXiv1806.05444, 2018.

[5] M. Mitzenmacher, *The Power of Two Choices in Randomized Load Balancing*, P.h.D. thesis, Harvard, 1991.

[6] D. Hochbaum and D. Shmoys, *A Polynomial Approximation Scheme for Scheduling on Uniform Processors: Using the Dual Approximation Approach*, SIAM Journal on Computing, 539–551, 1988.