Aalto University
School of Electrical Engineering

Parallel queues and dispatching

Pasi Lassila
Department of Communications and Networking

Aim of the lecture

- Introduce dispatching problem
- Disciplines for minimizing the mean delay
- Assignment 3

Contents

- Introduction to dispatching problem
- Dispatching policies for classical dispatching in parallel queues
- Dispatching with redundant requests
- Assignment 3

Modeling job processing in data centers

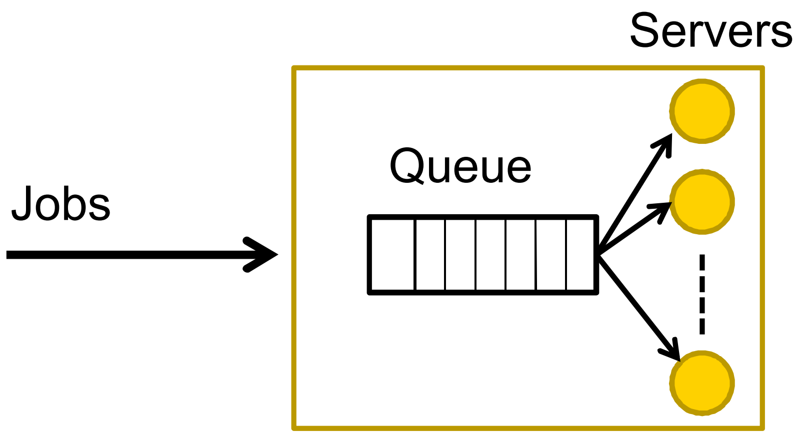
- Data centers consist of large numbers of servers

- Job/task scheduling
  - Computational jobs/tasks arrive in a centralized job scheduler
  - Based on some knowledge about the state of the system, scheduler decides which server will process the task

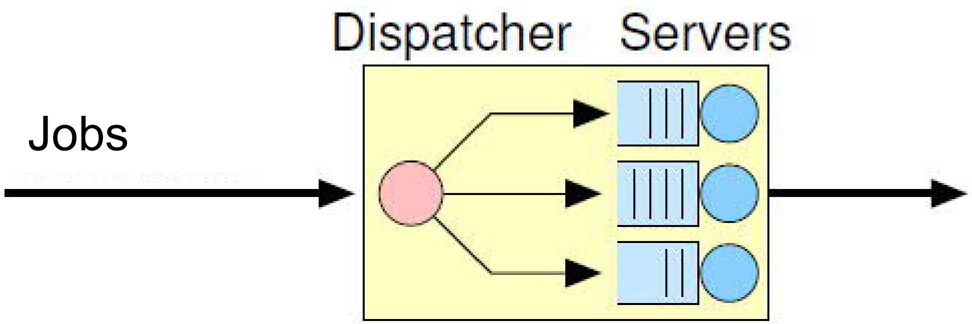
- In the literature, there are two approaches to model data centers
  - Centralized queue with multiple servers
  - Multiple parallel queues

Central queue and parallel queues

- Central queue
  - Jobs arrive to a centralized queue
  - Servers allocated dynamically



- Parallel queues
  - Jobs arrive at the dispatcher / scheduler
  - Dispatcher selects the queue where to route new jobs



- We focus on the parallel queue setting

Performance objectives

- Objective:
  - Minimize delay (performance)
  - Or in more recent literature the energy-performance tradeoff
    - To save energy some servers may be switched off
    - But turning them back on introduces a delay penalty, the set up delay
    - Performance can be a weighted sum of energy and delay

- Here we focus on just minimizing the delays (performance)

- Issues in optimizing the performance
  - Queueing discipline used for serving jobs: FIFO, PS or even SRPT?
  - Information available for selecting the server
    - Heterogeneous/homogeneous servers
    - Number of jobs at servers or even size-information
  - Something completely different? – Rethink how jobs are served!

Modeling the service times

- Service times often highly variable
  - Much more variable than exponential
  - Both in scientific computing as well as in web server workloads
  - Typically bounded Pareto-like behavior
  - Exponential still used for analysis!

- Service time distribution
  - In classical models, service times between servers are independent
  - This means that the service time on any server is sampled every time independently from its respective distribution in a given queue
  - Thus, the service time models also the random variations in the execution environment of the job
  - An alternative would be to define that each job has an intrinsic size (in flops) and the queues (servers) serve jobs at a rate r flops/s

Contents

- Introduction to dispatching problem
- Dispatching policies for classical dispatching in parallel queues
- Dispatching with redundant requests
- Assignment 3

The dispatching problem

- Jobs arrive at the dispatcher according to a Poisson process with rate λ
- There are N servers that process/serve jobs
- Each server is modelled as a single-server FIFO queue (with infinite number of waiting places)
  - Scheduling discipline is assumed to be FIFO
  - Reasonable for modeling scientific computing clusters, where jobs can not be pre-empted and are served until they finish
  - For web server clusters PS discipline may be more accurate, since web requests are served in parallel
- Thus, the model consists of N parallel queues

Heterogeneous servers

- Servers are heterogeneous
  - We assume independent service times across servers
  - Then the service time distribution depends on the queue n
  - Let S_n denote the service time of jobs in the n :th queue obeying the distibution $F_n(t) = P\{S_n \leq t\}$
  - Also, the service rate of queue n is $\mu_n = \frac{1}{E[S_n]}$

- Now the maximal stability condition becomes

$$\boxed{\lambda < \sum_{n=1}^{N} \mu_n}$$

Static load balancing policy

- Consider a simple probabilistic policy $\{p_1, \ldots, p_N\}$ with p_n denoting the probability of dispatching the arrival to server $n = 1, \ldots, N$

- Reasonable objective: balance loads at each queue
  - Require load at each queue is some constant k

$$\boxed{\frac{p_n \lambda}{\mu_n} = k \quad \Rightarrow \quad p_n = \frac{k \mu_n}{\lambda}}$$

  - On the other hand, $\sum p_n = 1$, and we obtain $k = \frac{\lambda}{\mu_1 + \cdots + \mu_N}$ and

$$\boxed{p_n = \frac{\mu_n}{\mu_1 + \cdots + \mu_N}}$$

  - Static policy that does not use any state information, also does not depend on λ
  - Still maximally stable!

Performance under static load balancing

- Under probabilistic policy, each queues is stochastically independent
  – Recall we have Poisson arrivals!
- Each queue is an independent FIFO queue with arrival rate $p_n\lambda$ and assuming exponential service times delay at queue n is

$$E[T_n] = \frac{1}{\mu_n - p_n\lambda}$$

- Overall mean delay is thus

$$E[T] = \sum_n \frac{p_n}{\mu_n - p_n\lambda} = \frac{N}{(\mu_1 + \cdots + \mu_N) - \lambda}$$

  – As if *N* servers in series each with total service rate $(\mu_1 + \cdots + \mu_N)$

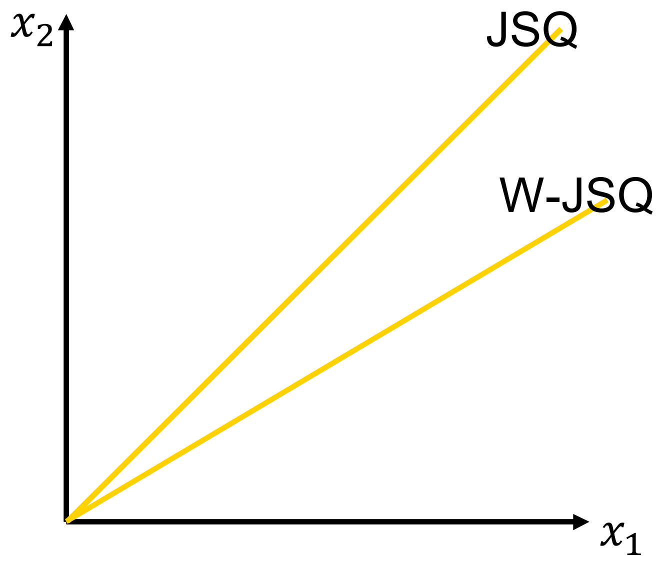
Dynamic JSQ dispatching policy

- JSQ (Join-the-Shortest-Queue)
  - The dispatcher knows the number of jobs in each queue, i.e., the state vector $(x_1, \ldots x_N)$ and the job is placed in the queue n with smallest x_n
  - In case of ties, they are broken randomly

- Exact optimality results are scarce for the dispatching problem!
  - JSQ is optimal for minimizing the delays if all servers have homogeneous rates, $\mu_n = \mu, \forall\, n,$ and service times are exponential (Winston, 1977)
  - For heterogeneous servers, JSQ is still maximally stable – tries to always balance the queue lengths
  - In case of 2 heterogeneous servers, it is known that a switching curve characterizes the optimal policy but form is not known (Hajek, 1984)
    - Switching curve: curve $x_2 = f(x_1)$ in state space (x_1, x_2) where above curve arrival dispatched to, say, queue 1 and below it to queue 2

JSQ for heterogeneous servers

- Weighted JSQ (W-JSQ)
  - For queue i, the mean waiting time is x_i/μ_i , $i = 1, \ldots, N$
  - In the W-JSQ policy, the arriving job is routed to the queue i with smallest mean waiting time , i.e., i such that $\frac{x_i}{\mu_i} = \min\{\frac{x_1}{\mu_1}, \ldots, \frac{x_N}{\mu_N}\}$
  - Will favor queues with higher service rate μ
  - Ties broken randomly



- Example with 2 servers with $\mu_1 > \mu_2$
  - JSQ corresponds to a switching curve along the diagonal
  - W-JSQ corresponds to a switching curve $x_2 = \frac{\mu_2}{\mu_1} x_1$
  - Above the curve decision is to route job to queue 1

Aalto University
School of Electrical Engineering

Performance of JSQ-like policies

- No closed form performance results available for dynamic JSQ-like policies
- Assuming exponential service times, for a given policy the system corresponds to an N-dimensional Markov process
  - In principle, steady state distribution can be solved for moderate loads and small N
  - So numerical analysis is possible
- But the policies can be also simulated easily!

Contents

- Introduction to dispatching problem
- Dispatching policies for classical dispatching in parallel queues
- Dispatching with redundant requests
- Assignment 3

Redundant requests technique

- Dynamic policies require state information
  - Not always easy to manage collection of this in large systems
  - Is it possible to avoid use of any state information but still perform (almost) as good as JSQ-like policies?

- Redundant requests idea
  - Load levels of servers vary randomly
  - Even if idle, service time on two servers can be very different, e.g., due to seek times or, generally, availability of the data required by the job
    - Also background load of servers causes random delays for process scheduling
  - To minimize the impact of service time variability, send replicas of the job to several servers and wait until fastest one finishes, i.e., delay is the minimum of these random variables

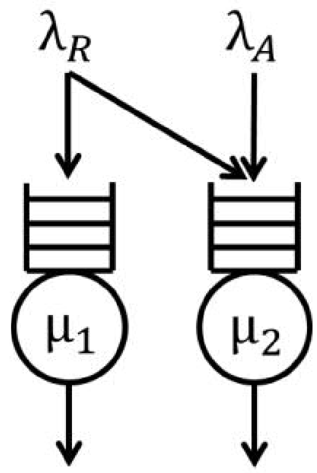
Redundant requests technique

- Properties
  - Simplicity: does not require any state information!
  - Can achieve smaller mean delays, especially useful for controlling the tail of the delay distribution (Ananthanarayanan, 2013)
  - But the system also wastes effort on processing multiple replicas
  - Also, extra overhead from need to replicate and to remove other jobs after fastest one finishes

- Even Google applies this idea! (Dean and Barroso, 2013)

- Question: does it always work?
  - Depends, for example on assumptions of the service times

Markov analysis of redundant requests (Gardner et al., 2015)



(a) ℕ model

- Consider simple 2 server model ("N-model")
  - Service times exponential and independent with rates μ_1 and μ_2
  - Class R jobs are replicated to servers 1 and 2 and arrive according to Poisson process with rate λ_R
  - Class A jobs are only sent to server 2 and arrive according process with rate λ_A
- For stability we must have

$$\lambda_R + \lambda_A < \mu_1 + \mu_2 \ , \quad \lambda_A < \mu_2$$

- Mean delay of class R and class A

$$E[T_R] = \frac{1}{\mu_1 + \mu_2 - (\lambda_R + \lambda_A)}$$

$$E[T_A] = E[T_R] - \frac{1}{\mu_1 + \mu_2 - \lambda_A} + \frac{1}{\mu_2 - \lambda_A}$$

When to replicate?

- In our original model we have jobs arriving at total arrival rate λ
- Assume that jobs are replicated with probability p_R
  - Since $E[T_A] = E[T_R] + a$, where $a > 0$, non-redundant class A always suffers from replication for any value of p_R
  - We can minimize this effect by always replicating, i.e., optimal $p_R = 1$
  - Thus, for exponential (and independent) service times it is always optimal to replicate
- Is it always optimal to replicate for any service time distribution?
  - Assume that job sizes are deterministic, then there is no benefit from replicating since the slower server is always just doing useless extra work, i.e., it is the worst solution
  - Thus, optimal replication probability clearly depends on service time variability
  - Also, our assumption of independence of the service times affects!

References

[1] G. Ananthanarayanan, A. Ghodsi, S. Shenker and I. Stoica, ”Effective straggler mitigation: Attack of the clones”, In Proc. of NSDI, 2013, pp. 185-198.

[2] J. Dean and L.A. Barroso, “The tail at scale”, Commun. ACM 56, no. 2, 2013, pp. 74-80.

[3] K. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter and E. Hyytiä, “Reducing Latency via Redundant Requests: Exact Analysis”, SIGMETRICS Perform. Eval. Rev. 43, 1 (June 2015), pp. 347-360.

[4] B. Hajek, "Optimal control of two interacting service stations," IEEE Trans. Automatic Control, Vol. 29, June 1984.

[5] W. Winston, “Optimality of the shortest-processing-time discipline”, J. Appl. Prob. 14, 1977, pp. 181-189.

Contents