Reducing Accumulative Relative Tardiness (ART) for soft realtime load balancing

Abstract

Many systems today have incorporated in them the idea of task priorities and have a growing requirement of soft real-time guarantees while making scheduling (load balancing) decisions. In this project, I am looking at the idea of using reinforcement learning to tune weights of the weighted round robin (WRR) load balancer to distribute resources between tasks to minimize the overall tardiness in a system.

Introduction

I am looking at load balancing, possibly in edge clusters, as a soft real time system. This system might have different deadlines for different processes or for processes generated by certain sources. But may not have any deadline requirements for many others.

There are two distinct parts of the system:

- a ***load balancer***
  - This is responsible for actually balancing load for the different upstream servers. Servers that can satisfy a particular request type will be grouped together. We will call these ***service groups***. Each server in a group will be given a weight.
- a ***scheduler***
  - The scheduler looks at the aggregate state of the load balancer. There will be an initial observation period during which will end when the scheduler has enough data points. This will be manually decided. Once the scheduler has enough data points, the scheduler will club together similar data points into clusters. Each cluster will now behave as a state for the decision making stage of the scheduler. The overall idea is to get the scheduler to tweak the weights in different service groups to move the load balancer from one state to another more desirable state.
- Observe and collect statistics from the load balancer to create different clusters to broadly define different "states" of the system.
  - This allows us to define a finite number of states.
  - The number of clusters have to be pre-defined. The dimensionality of the data used is a good guide in that direction.

- Once states are determined, use reinforcement learning to find an optimal scheduling solution that looks to reduce tardiness given optimization goals.
  - Optimization goals will be provided in terms of service times.
  - ***tardiness*** $:= max[0, (t - d)]$
    1. t => total time spent in the system by a job
    2. d => deadline for the process.

Project Goal

$$Goal := min \sum_{i=1}^{N(T)} max[0, \frac{(t_{ij} - d_j)}{d_j}]$$

(1a)

So,

$$G_t = max \sum_{i=1}^{N(T)} min[0, \frac{(d_j - t_{ij})}{d_j}]$$

(1b)

where,

$$d_j = \mu_{t_j} + c_j \sigma_{t_j}$$

(2a)

$$\mu_{t_j} = \frac{1}{n_j} \sum_i t_{ij} 1(j)$$

(2b)

$$\sigma_{t_j} = [\frac{1}{n_j} \sum_i t_{ij}^2 1(j)] - \mu_{t_j}$$

(2c)

$$t_j = \sum_i t_j 1(j)$$

(2d)

$$1(j) = \begin{cases} 1, & \text{if task is of type j} \\ 0, & \text{otherwise} \end{cases}$$

$$c_j = 1 + \frac{1}{p_j}$$

(2f)

T is the ***time interval at which the reward is measured*** and $N(T)$ is the ***number of jobs that go through that system in that time***.

where:

p_j := priority of task type j. For the moment the priorities are static and manually assigned.

$$c_j \in (1, 2)$$

$\mu_{t_j}$is the mean of all tasks of type j completed when the scheduler was observing the system

$\sigma_{t_j}$is the standard deviation of all tasks of type j completed when the scheduler was observing the system

$\frac{(t_{ij}-d_{ij})}{d_{ij}}$ is the ***Relative Tardiness*** of the job.

We are interested in minimizing the ***Accumulative Relative Tardiness (ART)*** - $\sum_{i=1}^{N(T)} max[0, \frac{(t_{ij}-d_{ij})}{d_{ij}}]$ of the overall system.

States and storing them

The system will be under an initial period of observation while it's under scripted load. These observations are to be used to create a broad definition of "states" which will consist of a bunch of similar actual states of the system. We will add the actual parameters later.

There can be a predefined number of distinct states K, possibly depending on the dimensionality of the data being considered. Each state can be stored as a mean vector $\boldsymbol{\mu}$ and a covariance matrix Σ. The states can be recalibrated if the number of observed data points that lie outside all states; $\boldsymbol{s} \notin \{\boldsymbol{\mu}_k \pm 3\Sigma_k\} \forall k \in [1, K]$; cross a predefined threshold, $N_{threshold}$. Recalibration will include another observation stage.

State parameters

For each request type and for each server. These counters are specific to NGINX load balancers.

- requestMsec
  - The average of request processing times in milliseconds. requestMsecs
  - times
    - The times in milliseconds at request processing times.
  - msecs
    - The request processing times in milliseconds.
- requestBuckets
  - msecs

    - The bucket values of histogram set by vhost_traffic_status_histogram_buckets directive.
    - counters
        * The cumulative values for the reason that each bucket value is greater than or equal to the request processing time including upstream.
- connections
    - active
        * The current number of active client connections.
    - reading
        * The total number of reading client connections.
    - writing
        * The total number of writing client connections.
    - waiting
        * The total number of waiting client connections.
    - accepted
        * The total number of accepted client connections.
    - handled
        * The total number of handled client connections.
    - requests
        * The total number of requested client connections.
- requestCounter
    - The total number of client connections forwarded to this server.
- responses
    - 1xx, 2xx, 3xx, 4xx, 5xx
        * The number of responses with status codes 1xx, 2xx, 3xx, 4xx, and 5xx.
- weight
    - Current weight setting of the server.
- maxFails
    - Current max_fails setting of the server.

Load Balancing

The Weighted Round Robin (WRR) forms the basis of the load balancing algorithm. Different types of tasks have different priorities. For each task type, the backend servers can have different weights. The idea is to readjust the weights of the servers, for each task type, so that the overall resources are distributed between the jobs in the system so as to maximize G_t.

Role of the WRR weights as our control

Each backend server is allocated a weight for each request type. For example, server i will have the weight w_{ij} for task type j.

Total share of a particular server for a particular task type is $\frac{w_{ij}}{\sum_j w_{ij}}$.

Total share of resources allocated to any task j is:

$$R_j = \sum_i (\frac{w_{ij}}{\sum_j w_{ij}})$$

(3) Since, $\sum_j R_j = S$, where S is the total number of servers, adjusting the weights in a WRR provides a way to control the amount of total resources provided to a task type.

(The assumption here is that the servers are homogeneous.)

Role of the Reinforcement Learning Model

Now that we have our states, a goal and our control, the role of the reinforcement learning model would be to learn how to tweak the controls in order to move towards the goal along the best possible path.

Training the RL model

Major issue

The desirable states may be unobservable unless states are observed and classified during training.

TODO

How to train the model? Especially with simulation.