Optimizing Router Engine

Jayanth Kumar

Approach

SImulation

```
self.run():
        self.set_vendor_availability():
                for vendor in vendor_list:
                        for each row in opened vendor csv:
                                vendor.is_available = row[‘API Available’]

        self.cater_request_output():
                for row in opened request time csv:
                        ans = self.route(row)
                        write ans['Request Index', 'Vendors tried']  in output csv
```

Strategies

- **Dummy Route** : Route all request to vendor1
  Some requests are never fulfilled, when vendor1 is down
- **Simple Route**: Route request to vendor1, if it is up, else to vendor2 and then, to vendor3 in order
  All requests are fulfilled but the request tries the order
- **Steady State Traffic Route** : Route request based on steady state traffic quota only
- **Dynamic Traffic Route**: Routing request based on dynamic traffic quota based on failure & comeback

Dynamic Traffic Route Algorithm

```
def route_dynamic_traffic:
        a = get_index_multinomial_single_roll(traffic_prob_list)
        for vendor in self.vendors_list[a:]:
                vendor_tried_list.append(vendor.label)
                self.checkpoint_till_time_sec(vendor, time_sec)
                if vendor.is_available[time_min - 1]:
                        heappush(vendor.stats['success'], time_sec)
                        write_row['Vendors tried'] = '|'.join(vendor_tried_list)
                        break
                else:
                        heappush(vendor.stats['failure'], time_sec)
        traffic_prob_list = self.recalculate_traffic_prob(time_sec)
        return write_row
```

Strategy Pattern & Dependency Injection

- Vendor and RouterEngine Class
- Vendor has state for traffic quota, availability and stats
- Router engine has state of vendor list and route strategy
- Different strategies for routing injected during runtime as method
- Same set of checkpoint and recalculation functions works around those strategies, keeping same behaviour intact but with different intent
- Decorators can be added for checkpointing and book-keeping

Checkpointing & traffic quota calculation

```
while time_sec - vendor.stats['success'][0] >=
vendor.comeback_threshold_sec:
        heappop(vendor.stats['success'])

while time_sec - vendor.stats['failure'][0] >=
vendor.comeback_threshold_sec:
        heappop(vendor.stats['failure'])

for vendor in self.vendor_list:
        success_count = len(vendor.stats['success'])
        failure_count = len(vendor.stats['failure'])
        total = success_count + failure_count
```

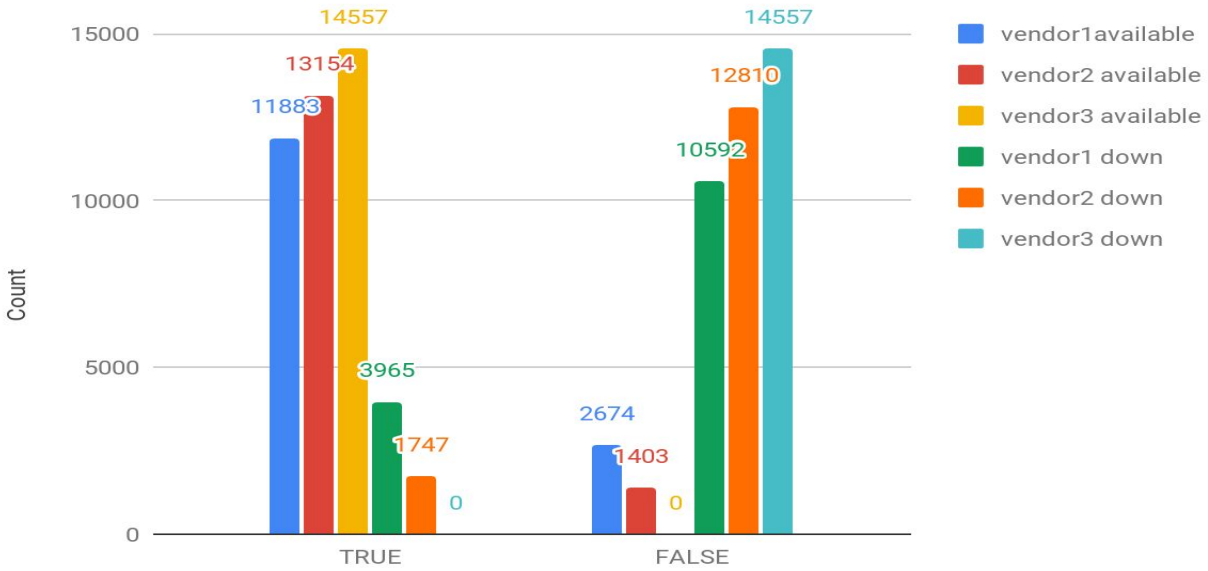
traffic quota calculation

```
if ftm_failure_percent > vendor.failure_threshold:
        vendor.is_down[time_sec] = True
        vendor.traffic_p = (vendor.steady_state_traffic * .001)
        carry_over += (vendor.steady_state_traffic * .009)
else:
        vendor.traffic_p = vendor.steady_state_traffic * .01
         each_vendor.traffic_p += carry_over
         carry_over = 0
elif vendor.is_down[time_sec] and ctm_success_percent >
vendor.comeback_threshold:
        vendor.is_down[time_sec] = False
```

Analysis

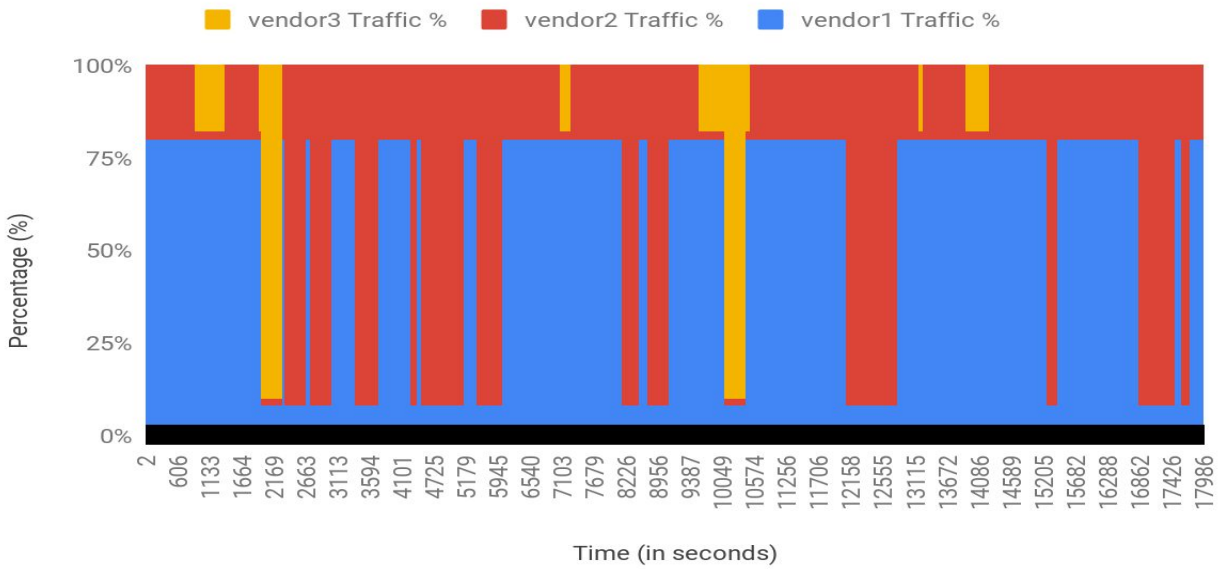
Histogram of Availability & Down

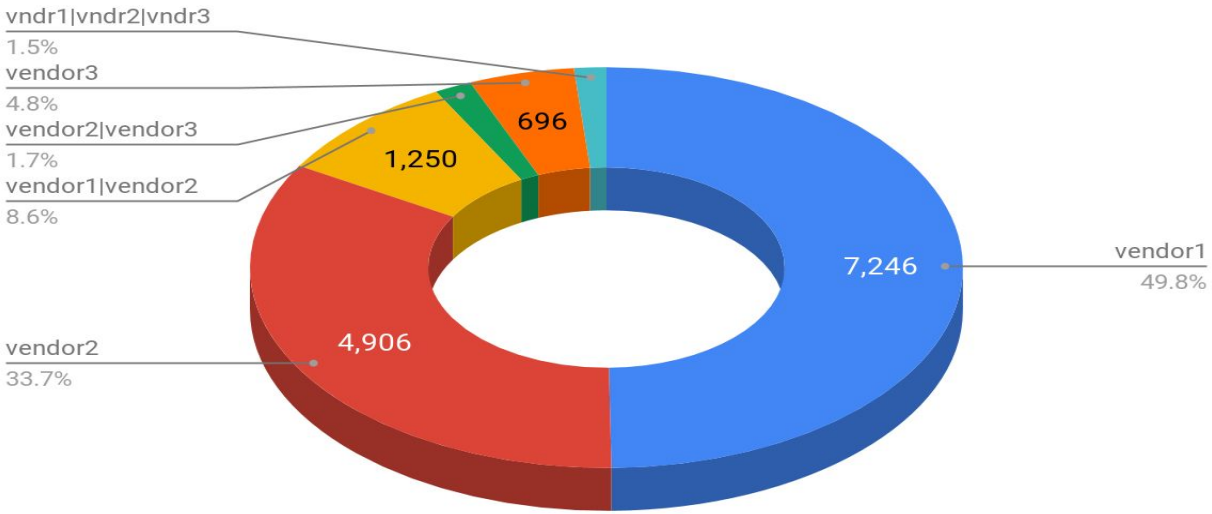
Vendor 3 always available(True) and never down(False)



Traffic percentage per vendor

0.8 0.2 0
0.8 0.02 0.18
0.08 0.92 0
0.08 0.02 0.90





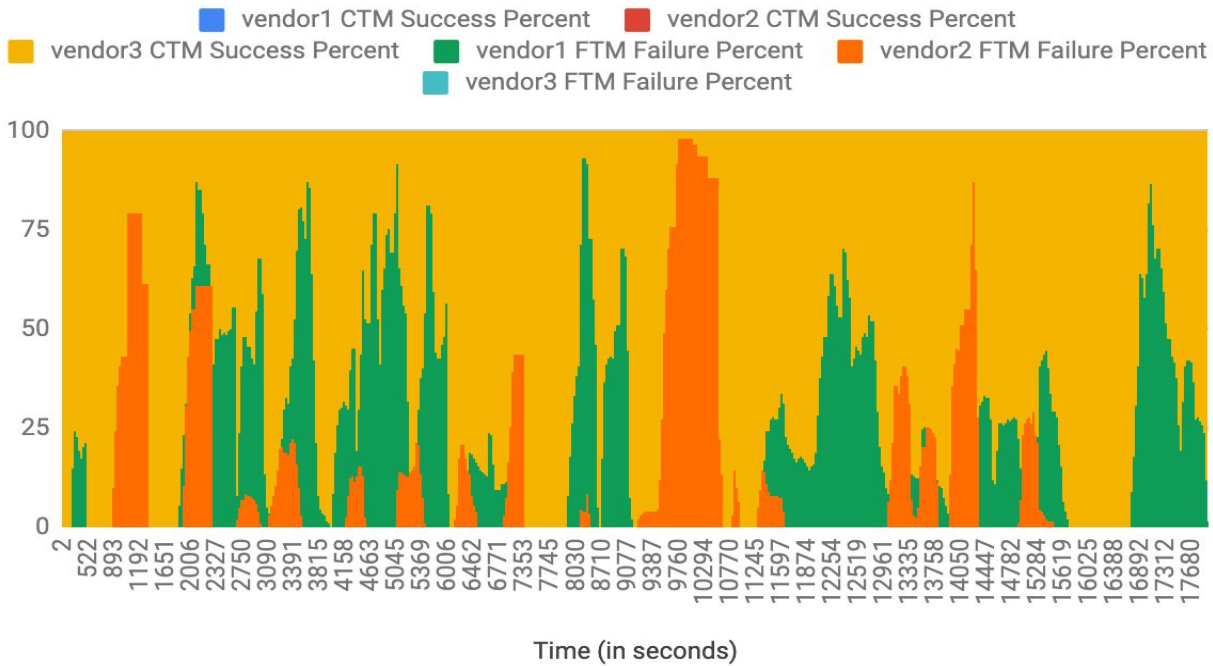
Proportion of Venders Tried

Vendors tried	COUNT
vendor1	7246
Vendor1|vendor2	1250
vendor1|vendor2|vendor3	213
vendor2	4906
vendor2|vendor3	246
vendor3	696
Grand Total	14557

Vender API Success & Failure Percent

Vendor 3 most success

Vendor 1 and Vendor 2 lot of failure



Percent vs Vendor Tried

For each vender in tried, vendor API comeback success is high and failure rate is low

Similarly, the percent order also justifies the tried order