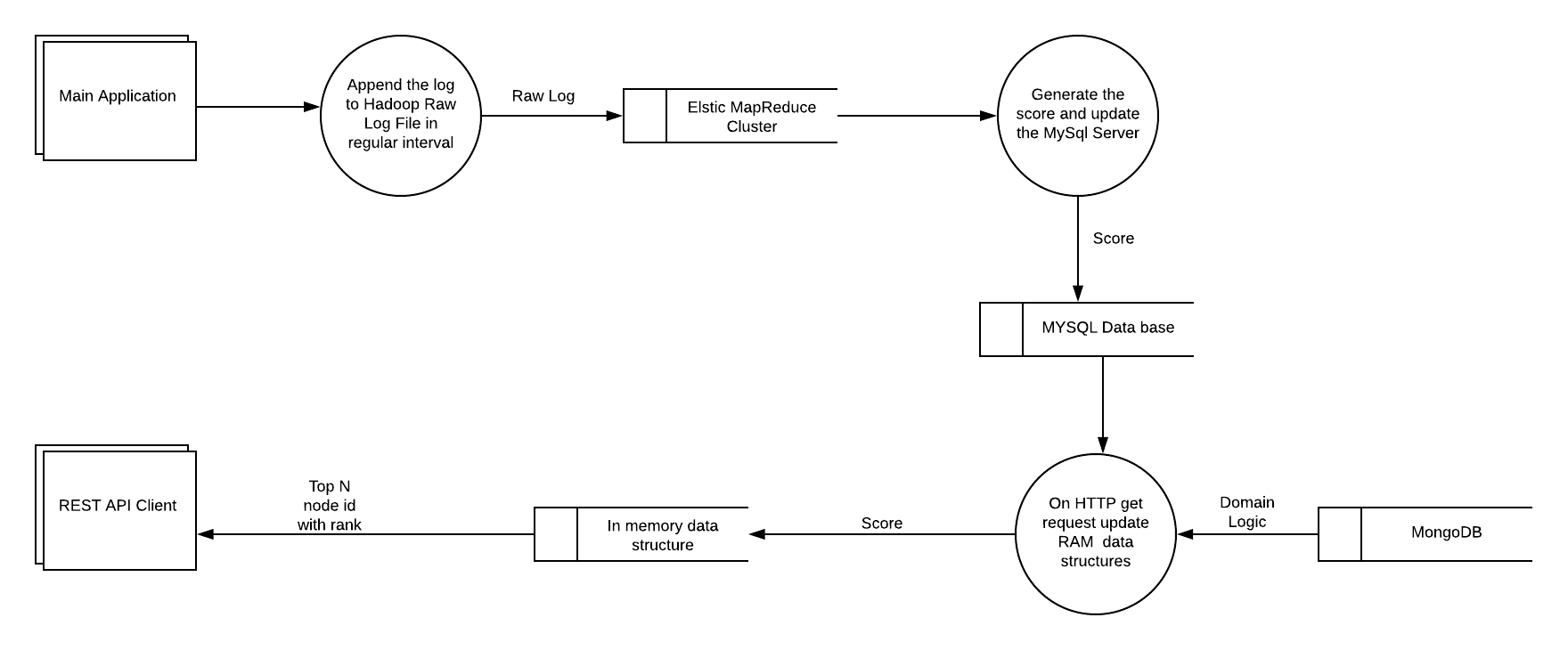
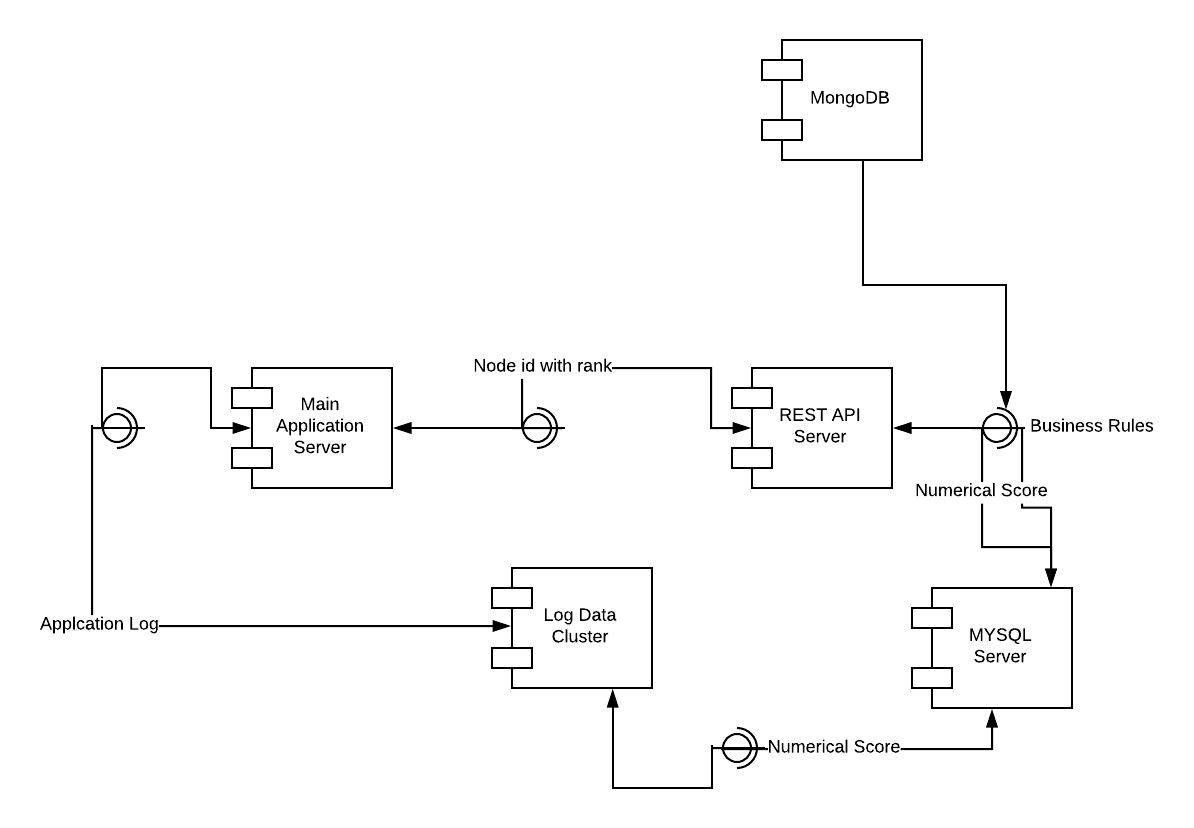
**Recommendation Engine First Row (Documentation after Implementation)**

Good News!! Your recommendation engine is live now. Model is updatable, i.e., in each training iteration, it will update the store model not replace it. It is possible only when your model is a function of f and f(x+y) = f(x) + f(y). Count and sum are the examples of this kind of function. Our model is a function of the count. Model is exposed as a Rest API to integrate with other systems. To make the REST API response fast, we stored all information required for the recommendation in shared data structured in RAM. To respond the request API does not need to fire any query in other than RAM data structure. We also implemented parallel execution of each query in one response. The training process is scalable with the size of input raw log data as it is implemented in Spark. You can add a slave node anytime to Spark cluster. So there is no upper limit in input data. Training system can handle input data size more than it’s RAM size.

Below UML diagram will describe the data flow



Below UML diagram will describe the infrastructure components of system

`

**Recommended Hardware**

REST API Server: RAM size should be highest available ones. Assuming application server will send maximum 30 concurrent requests to REST API, the processor should have six cores CPU.

Log data cluster and MYSQL server performance are not crucial in the system. We can take midsize instances from AWS.

**Algorithmic Context – Logic behind the magic**

We did our proprietary implementation of Naïve Bayes classifier. Input for our model is a table like below.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Node** | **Identifier** | **ContentType** | **Author** | **Platform-Channelname** | **prevNode** | **is\_watched** | **city** | **region** | **country** |
| 0 | 10168290 | 44 | post | 1272558 | roku-moviesbyfawesome |  | 1 | Bogota | Distrito Especial | Colombia |
| 1 | 10185615 | 29 | post | 1459180 | roku-moviesbyfawesome |  | 1 | Prosper | Texas | United States |
| 2 | 10185615 | 29 | post | 1459180 | roku-moviesbyfawesome |  | 1 | Highland Lakes | New Jersey | United States |
| 3 | 10185615 | 29 | post | 1459180 | roku-moviesbyfawesome | 10234264 | 1 | Kannapolis | North Carolina | United States |
| 4 | 10185615 | 29 | post | 1459180 | roku-moviesbyfawesome | 10257819 | 1 | Kannapolis | North Carolina | United States |

The objective of the algorithm is to calculate the conditional probability of a Node will be watched for a particular feature value.

Let assume we are calculating the score of node= 10033207 for feature = identifier (device id) and feature value=49 (choose any device id according to your choice)

Feature count = Number of records where that device id (49) is present

Conditional count = Number of records with is\_completed=1, identifier = 49, node= 10033207

Actual probability will be the ratio of the conditional count and feature count, but we are not dividing this stage because the model will not be updatable.

We further assumed people like the node mean they like the subcategories of the video. So we are mapping the count of nodes to count of subcategories.

Let say you have five nodes N1, N2 … N5 and three categories S1, S2 and S3. Category S1 belongs to N1, N3, and N4. Then

Count of S1 for a particular feature value (let say identifier = 49)

= Sum of count for N1, N3 and N4 for that feature value(identifier= 49)

This has two advantages.

1. The number of categories is much less than the number of nodes. So REST API search space is reduced, so response time improves.

2. Subcategories are assigned to the nodes manually. So we include that human intelligence to our model.

The final score:

1. Score = Conditional Count/ Feature count

2. For given set of feature value score for each category

= sum of the score for that category for each feature value in input JSON of curl command. If feature value is not present then score is 1.

3. Let say your top categories are C1, C2 … CN. And their score is S1, S2 … SN.

4. Now for Node N1 if C1, C3 and C5 present in the node, the score will be =( S1+S3 + S5) / N

5. Now for Node N2 if C2, and C5 present in the node, the score will be = (S2 + S5) / N

**Rules from Business:**

1. Give some extra weight to previous watched node categories because someone just watched a tv serial episodes so in his recommendation that tv following episodes will get extra weight. Amount of extra weight is 0.5
2. Give some extra weight to all previous watched nodes by that device categories because someone just watched mostly tv serial episodes so in his recommendation that tv following episodes will get extra weight. Amount of extra weight is 0.25
3. Choose top 40 nodes according to score and then return 30 nodes randomly. Audience will get more new feel in each view.
4. Block adult contents.
5. If previous node is movie recommend only movie
6. If previous node is kid content recommend only kid and entertainment content.
7. If previously observed node comes in recommendation output, remove it from the list.

We have developed a descriptive language like Hibernate to define the rule by business owner.

For detail logic of Business rules described in below diagram (I will provide tomorrow)

**How to deploy:**

1. Download the code from <http://code.ifood.tv/project/video_recommendation/> in spark name node.
2. Install all dependency written in spark\_requirement.txt in all nodes of spark cluster and copy GeoIp database(GeoLiteCity.dat). It is also checked in.

sudo yum groupinstall "Development Tools"

sudo yum -y --enablerepo=epel install geoip

sudo yum -y --enablerepo=epel install GeoIP-devel

sudo pip install GeoIP

1. Download the code from <http://code.ifood.tv/project/video_recommendation/> in REST API Server.

Install all dependency written in requirement.txt

virtualenv venv

source venv/bin/activate

pip install -r requirement.txt

if virtualenv is not installed in machine install it using yum.

1. Update the MySQL, Mongo info in config files (both spark and rest api server)
2. Copy the dev.pem equivalent in production in /home/hadoop/ path and update the file name in wrapper.sh and wrapper\_regular.sh
3. For first time
   1. copy the initial data to hadoop cluster.
   2. hadoop dfs –put log2.txt /tmp/

Make sure you give your data file name log2.txt and copy under /tmp/ directory in hadoop this step is not required.

* 1. copy the threshold.txt to under /tmp/ directory in hadoop
  2. run the wrapper\_wrapper.sh and wait for it to finish

1. Add wrapper\_wrapper\_regular.sh in cron job with 5 min interval. Make sure you are copying the new log in /home/hadoop/data/ location in hadoop name node.
2. Add reboot.sh in cron with 30 min interval.