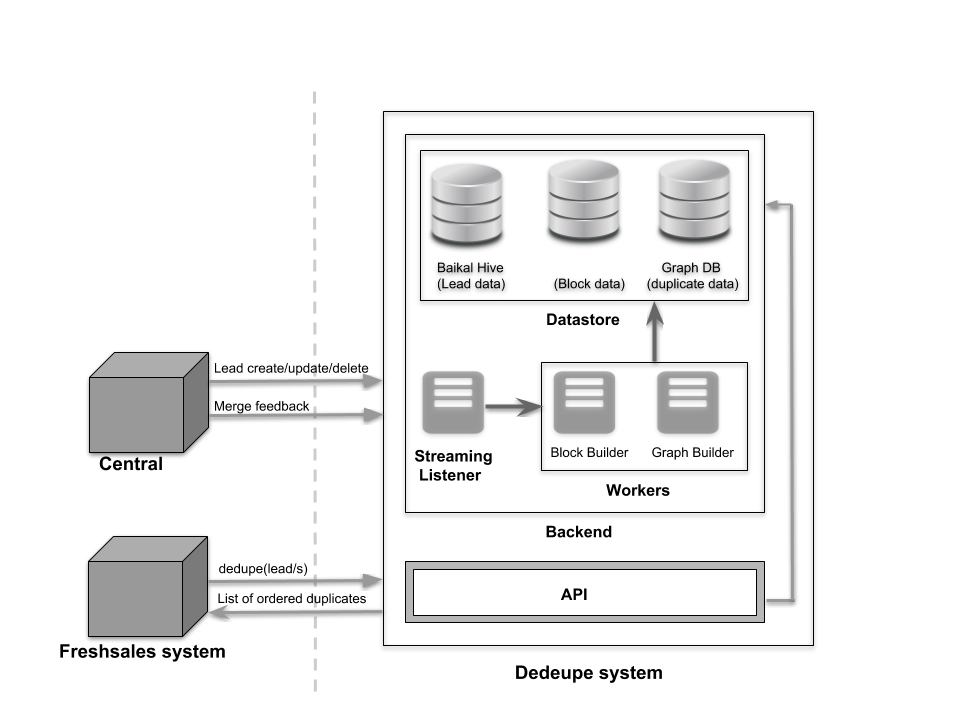
# Title: Engineering Architecture

|  |  |
| --- | --- |
| **Engineering Challenges** | **Design Considerations to solve the prob** |
| Calculating similarity score between all pairs of leads within an account is exponential in terms of complexity. Eg: FD account has 3M leads which would result in ~(3M)2 pairs. | Blocking using Record Linkage library to reduce the number of pairs. |
| Efficient way of pairwise scoring | Distributed account-wise blocking. Score only for a subset of pairs |
| Name fields are all proper nouns, so ML model will not learn similarities between names. | We have to do cleaning of such fields to standardise each name field. |
| Realtime querying ability for duplicate leads | Graph db(nodes are leads and edge weights are similarity scores). Graph traversal to find the n strongly connected nodes. |
| Record linkage doesn’t have distributed processing support. | Either find a work around to use record linkage in distributed way or don’t build graph in realtime |
| Order the duplicates that are suggested based on an efficient ranking algorithm. | Use the edge weight and the no. of hops between nodes to calculate the rank. |
| Handle unique leads (floating nodes in graph) which will be a huge % | After blocking, do not compute score for a block with just 1 lead. So, don’t insert these nodes in the graph. |
| Efficient pairwise scoring during add/update a node in the graph | Block information is maintained separately along with the field(rule) value used for blocking. Block the new lead and score the new/updated lead only against that block |
| Efficient pairwise scoring across blocks —> yet to be decided whether it is a requirement |  |
| Dedupe of bulk lead upload |  |
| Merging two graphs —> yet to be decided whether it is a requirement |  |

System design:



Interactions with data outside dedupe system:

Kafka (streaming jobs):

1. Lead\_create  → add node to graph
2. Lead\_update  → update the graph (create edges or update calculated scores)
3. Lead\_delete → delete node from graph
4. Merge data (the new payload we have requested) → Update the graph(create edges or update actual scores)

dedupe API endpoint:

1. For realtime querying for a single lead

Todo: Finalize the contracts

Graph db:

1. All leads are represented in a weighted undirected graph.
2. Each node represents a lead.
3. There is an edge between 2 nodes when the ml model has predicted a similarity score between them.
4. There are two edge weights for each edge. The calculated similarity scores between the connected nodes and the actual score(0 or 1) based on whether the user merged or not.  
   For leads that are not actually suggested to the user, the actual score call be NULL

Offline processes:

Preprocess data

1. All name fields(First name, last name, company name,etc.) have to be cleaned before using in the ML model.
2. Lib used: Modified name\_clever

Training the model:

1. Train the supervised model on a golden dataset.
2. Feature creation is based on Record Linkage library which forms pairs of leads and computes features like distance between each pairs.

Todo:

1. Finalize the golden dataset
2. Explore if parallel processing is possible in RecordLinkage

Building the graph:

1. Feature creation for leads which are not a part of golden dataset using RecordLinkage
2. Pick few pairs for which similarity score has to be computed(a few pairs in a block or even pairs across blocks maybe).
3. The selection criteria should make sure no lead is completely missed. All leads should be part of the graph.
4. For each selected pair, predict the similarity score using supervised ml model.
5. Build a graph with all the lead pairs in the above dataset + training dataset

Todo:

1. Figure out which graph which db to be used
2. Benchmark on the time taken to form graph on avg size account
3. Explore nepture and aws lambda

Update graph based on feedback:

1. Every time a merge is done by the user, merge data is captured from Kafka central using a streaming job.
2. For every pair of leads(primary, secondary records) in the suggestions, update the actual score to 0 or 1 in the graph based on user's choice.

Retraining the model(low priority):

1. Every time a merge is done by the user, merge data is captured from Kafka central using a streaming job.
2. Re-label/add all lead pairs from the suggestions in golden dataset based on the user’s choice.
3. Retrain the model once every x days.
4. Every time the model is retrained, the graph is built again(as mentioned above). Once the new graph is created, the old graph is dropped.

Todo: To decide how to store the golden dataset

Adding a node to the graph:

1. Every time a new lead is created, read the new lead from Kafka using a streaming job.

Todo: Figure out the most optimal algorithm

Deleting a node from the graph:

1. Every time a lead is deleted, read the deleted lead from Kafka using a streaming job.

Todo: Figure out the most optimal algorithm

Merging two graphs:

1. Every time a new set of leads are imported, the new graph that is created is merged to existing graph either online or offline.

Todo:

1. Figure out the most optimal algorithm

2. Decide whether is better Graph merge vs loop of node insert. Decision based on most optimal solutions and the number of contracts we need to maintain.

Realtime processes:

Single lead dedupe:

1. Whenever a user lands on a lead(query node) page, traverse the existing graph starting at the query node and print all the traversed nodes.
2. We can restrict the number of similar leads shown based on a configurable count parameter or number of levels we need to traverse.
3. Rank the suggestions

Todo:

1. Most optimal ranking algorithm has to be decided
2. Figure out the way to choose number of leads

Appendix:

Multiple leads dedupe(import from csv):

Method 1:

1. Do exact match based dedupe in realtime and create and merge the new graph offline.

Features: Low accuracy, low latency

Method 2 :

1. Each island in the graph represents a group of very similar records.
2. Within each island, the node with the maximum number of edges is the head node which can be considered as a representative of the other nodes within the island.
3. Create a graph based on new leads alone and find list of all head nodes found in the new graph.
4. We can find a dotted mapping between the two graphs by calculating similarity score between all pairs of head nodes between the two graphs.
5. If the similarity score is above a particular threshold, consider them as dedupe.
6. For all nodes in a particular island in new graph, the dedupes are all nodes in the island of the similar head node in the existing graph.

Features: Medium accuracy, medium latency

Method 3:

1. Create and merge the new graph with the existing graph and run single lead dedupe of each new lead.

Features: High accuracy, high latency

Todo:

1. Figure out if support is available in existing graph dbs for these functionalities.
2. Benchmark the time taken for the each methodology to see which method gives real time results.
3. Is there a better method to dedupe after graph merge in method 3, Instead of single lead dedupe?
4. Is there a better method to dedupe in method 2, instead of showing all nodes in both the islands(in old and new graph)?