DML (Decision Making Layer) HLD

Team

Name	Role
Nishchal	Dev
Aakarsh	Dev
Rohan M	Dev
Akshay	Functional lead
Abhinav	Architect
Navneet	Mentor / advisor

Terminologies

- **Hub** - Any first-mile hub, sort center, or last-mile distribution center can be termed a hub.
- **Node -** A hub can be referred to as a node in a graph.
- **FM** - First-mile hub. Directly mapped with the supplier.
- **LM** - Last mile hub. Directly mapped with the destination Pincode.
- **MM** - Mid-mile hub.
- **Link** - Lane between two hubs.
- **DML** - Decision-making layer
- **SOR** - System of records

PRD

https://coda.io/d/Auto-DML-v1-PRD_dGlUuiJxD-P/Laap-Auto-DML-v1-PRD_suszd#_luvlx

https://coda.io/d/Auto-DML-v1_d1HVSeMqwL1/Auto-DML-v1_suGUR?utm_source=slack#_luzEj

Objective

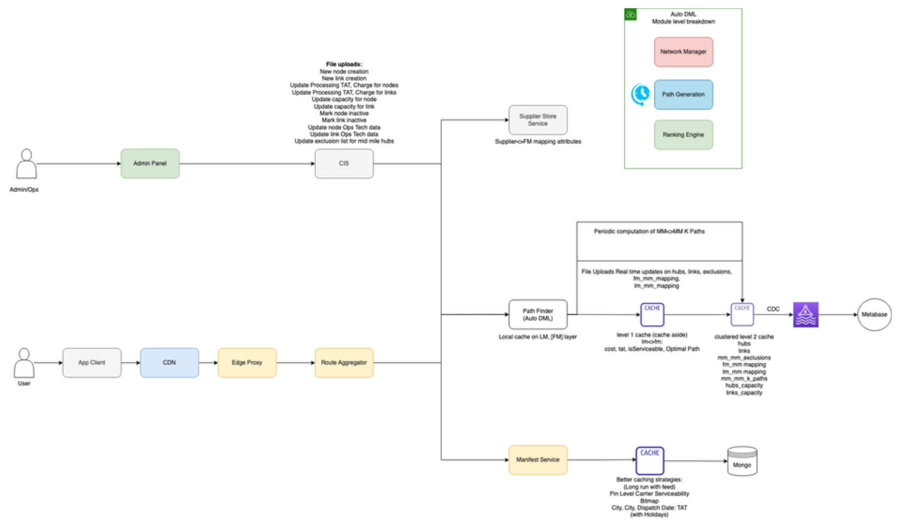
The end-state vision of the auto DML is to build an intelligent layer that will encompass not just LaaP orders but also any supply chain orders including e2e. This layer should be able to understand key parameters about the supplier, and customer and also have an intelligent network mapping built basis which the most efficient supply chain can be chosen. In the end state, the auto DML layer will include features like weight, category, pricing, TAT, customer & seller characteristics, 3PL capacity, node level TAT & pricing, etc. which will enable the layer to formulate the decision of choosing the appropriate supply chain.

The auto DML layer is responsible for managing the hubs and the connections between them.

Major modules

1. Network management layer
2. Path generator for MM layer
3. Ranking engine
4. Route aggregator

HLD



Link: https://app.diagrams.net/#G1A8tS0OCb1uoUnEWrEU6yrl8vT0sgnmsQ

Network management layer

Functional requirements

1. Admin/Ops team should be able to upload the network configuration for DML.
   a. Admin/ops would be able to add/delete new nodes and links, adjust their processing capacity and turnaround time, and update the names of ops/tech partners responsible for handling the shipment.
   b. Additionally, they would be able to update the exclusion list for the mid-mile layer, which would provide us with functionality to avoid any specific routes.
2. Publish the data to Presto for data analytics.

Responsibilities and operations

Important APIs	RPS
Creation of a new node	One off cases
Creation of a new link	One off cases
Update Processing TAT, Charge for nodes	10 rps
Update Processing TAT, Charge for links	60 rps

Update capacity for node	10 rps
Update capacity for link	60 rps
Mark node inactive	One off cases
Mark link inactive	One off cases
Update node Ops Tech data	One off cases
Update link Ops Tech data	One off cases
Add/Update seller Id to FM hubs mapping	One off cases
Add/Update PIn code and lm hubs mapping	One off cases
Force reload level 2 cache for few entries	One off cases
Force reload level 1 cache for few entries	One off cases

Module level load estimations:

Total expected write RPS at Peak = ~150 RPS (File uploads)

Data Store/ System of Records

Database: Redis (Used as persistent cache)

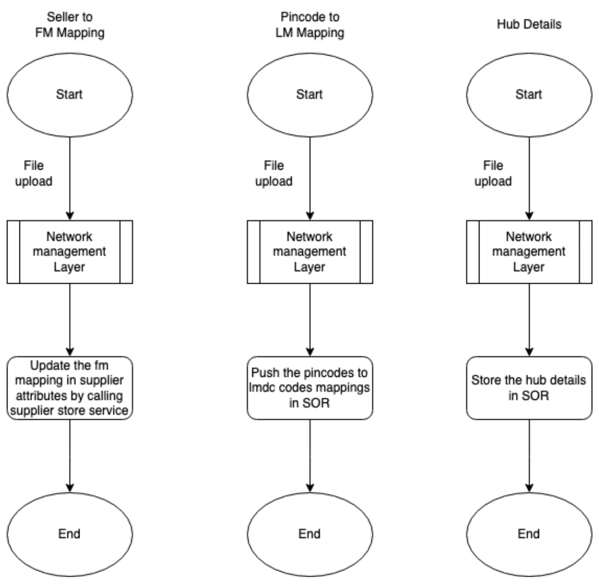
Data structure - **Hash set**

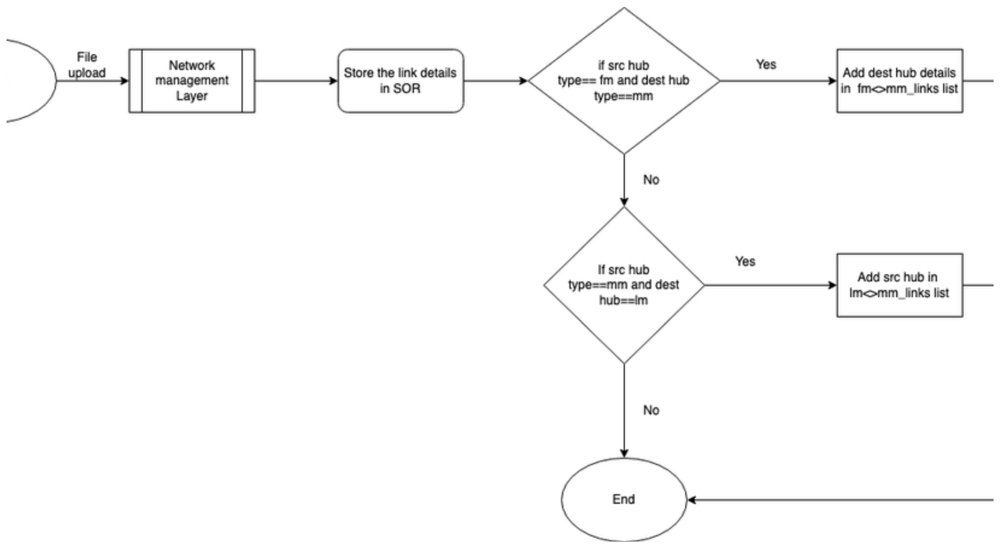
Key Name	Sample data	Description	Appx Documents Count	Est document size	Total Size
hubs	`{hubs_<hubId> : active, capacity etc}`	Hub level metadata	12200	2 KB	25 MB
links	`{links_<hubId1>_<hubId2> : active, tat, cost etc}`	Links between any 2 hubs	100000	500 B	50 MB
fm_mm_mapping	`{fm_mm_mapping_<hubId> : [mmHubIds..]}`	Path between every FM to MM. Assuming FM connected upto 2 MMS	2000	100 B	0.2 MB
lm_mm_mappings	`{lm_mm_mapping_<hubId> :` `[mmHubIds..]}`	Path between every MM to LM mapped in reverse way. Assuming LM connected upto 2 MMs.	10000	100 B	1 MB
mm_layer_paths	`{mm_mapping_<mm1>_<mm2> : [{path1}, {path2}]}`	K Shortest Paths between any mid mile layer nodes	40000	1 KB	40 MB
pincode_lm_mapping	`{pincode :[lmIds..]}`	Pin code to LM Mapping	20000	200 B	4 MB
exclusion_list	`{exlusion_list_<mm1>_<mm2> : [mm3, mm4]}`	List of all MM Hubs to be excluded for a given MM<>MM combination	40000	100 B	4 MB

Total estimated size: ~ 200 MB

Flows

1. Seller to FM mapping upload
2. Pincode to LM mapping upload
3. Hub details
4. Network mapping(Lanes)





File upload
Network management Layer
Store the link details in SOR
if src hub type== fm and dest hub type==mm
Yes
Add dest hub details in fm<>mm_links list
No
If src hub type==mm and dest hub==lm
Yes
Add src hub in lm<>mm_links list
No
End

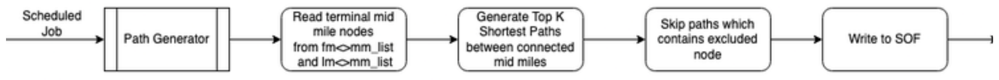
Instrumentation and alerting

1. Alerts will be configured when there are any API failures for file uploads. Thresholds will be decided later.
2. All the writes will be synced to metabase for audit and analytical purposes.
3. CIS will be storing the history of all the files being uploaded.
4. Alert on the Kafka connect produce rate.
5. Alert on any Kafka connect failures.

Path generator for MM layer

Functional requirements

1. Periodically run a process to generate the top K paths between any 2 mid-mile nodes.
2. Iterate over the fm<>mm_list and lm<>mm_list to find the mid miles between which we need to find the top K paths.
3. Ability to filter out nodes that are inactive while generating the top K paths.
4. Ability to filter out paths if it includes the excluded nodes.
5. Store the generated paths in SOR.



Scheduled Job
Path Generator
Read terminal mid mile nodes from fm<>mm_list and lm<>mm_list
Generate Top K Shortest Paths between connected mid miles
Skip paths which contains excluded node
Write to SOF

Module level load estimations:

Total expected write RPS at Peak = 70 RPS (Assuming 40k writes in 10 mins)

Instrumentation and alerting

1. Alerts will be configured in case the cron didn't run for >1 period of time.
2. Collect the metrics like the time taken for each path generation process run.

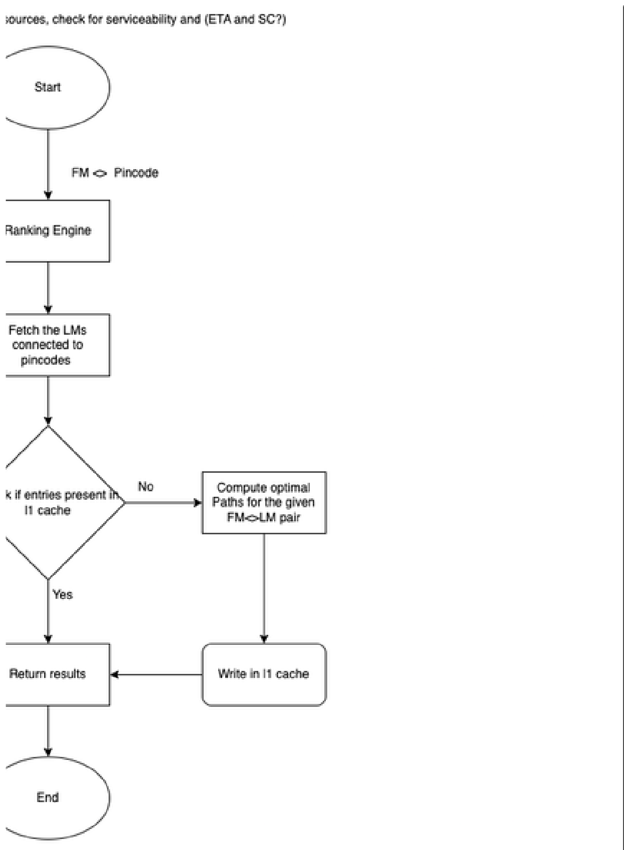
Ranking engine

Functional requirements

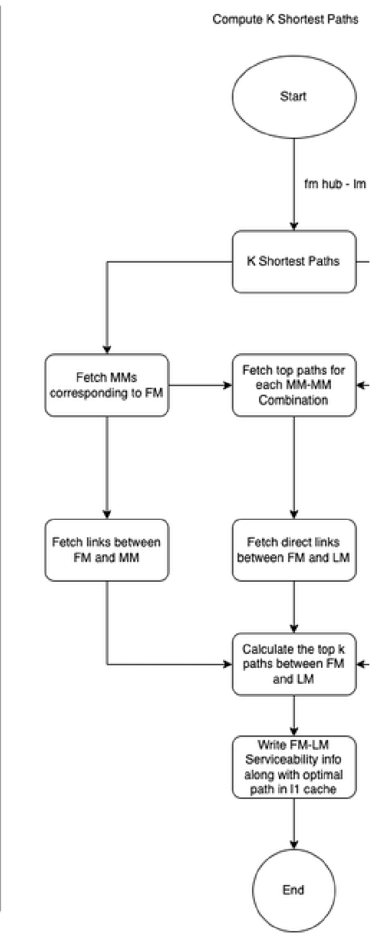
1. Should be able to resolve the LMs from the destination details in the request.
2. Provide an API to fetch the serviceability information i.e. isServiceable, cost, tat, optimal path for a list of FM and LM pairs (Order placement)
3. Provide an API to fetch the serviceability information and path(s) for a list of FM and LM pairs for shipment manifestation.
4. Provide an API to fetch the basic serviceability information i.e. isServiceable, cost, tat for PDP Check (PDP Serving and Feed might come later) for a list of FM and LM pairs.
5. Provide an API to check Destination Pin/Location Serviceability
6. Provide an API to update the capacity for a list of hubs and links in a particular path

Flows

Serviceability flow

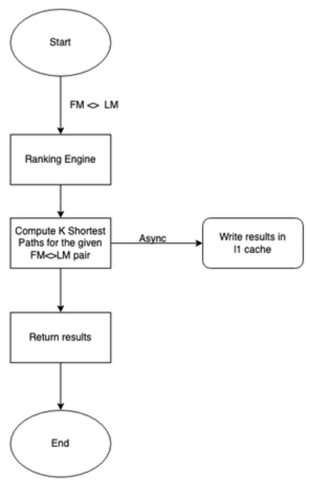


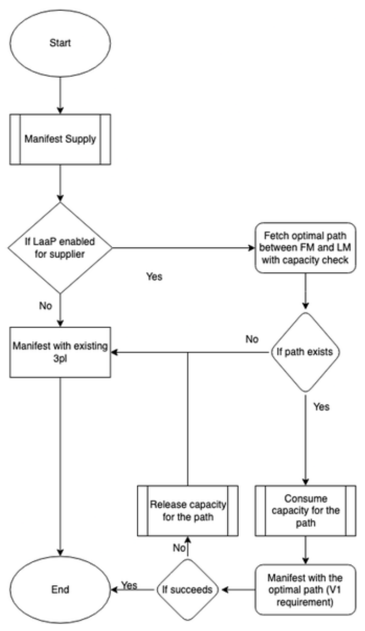
sources, check for serviceability and (ETA and SC?)
Start
FM <> Pincode
Ranking Engine
Fetch the LMs connected to pincodes
k if entries present in l1 cache
No
Compute optimal Paths for the given FM<>LM pair
Yes
Return results
Write in l1 cache
End



Compute K Shortest Paths
Start
fm hub - lm
K Shortest Paths
Fetch MMs corresponding to FM
Fetch top paths for each MM-MM Combination
Fetch links between FM and MM
Fetch direct links between FM and LM
Calculate the top k paths between FM and LM
Write FM-LM Serviceability info along with optimal path in l1 cache
End

Manifestation flow


Start
FM <> LM
Ranking Engine
Compute K Shortest Paths for the given FM<>LM pair
Async
Write results in l1 cache
Return results
End


Start
Manifest Supply
If LaaP enabled for supplier
Yes
Fetch optimal path between FM and LM with capacity check
No
If path exists
No
Manifest with existing 3pl
Yes
Release capacity for the path
Consume capacity for the path
No
End
Yes
If succeeds
Manifest with the optimal path (V1 requirement)

Module level load estimations:

V1 Scope

Total expected read RPS at Peak = **5K RPS**

Future Scope

If we enable serviceability on PDP and Product Feed

Total expected read RPS at Peak = **5K + 50K + 50 K = 105K RPS**

Data Store: L1 cache structure

Database: **Redis/Aerospike**

TTL: TBD

Data structure - **Key-Value pairs**

Key Name	Sample data	Description	Appx Documents Count	Est document size	Total Size
fm_lm_layer_paths	`{fmLmPaths_fm1_lm1 : path, serviceable, tat, cost}`	The optimal path with TAT and Cost between any FM and LM hub	Possible keys - 20 M	200 B	4 GB

Instrumentation and alerting

1. Alerts will be configured in case there are any API failures like 4xx and 5xx.

Route Aggregator

Functional requirements

1. Ability to call Auto DML API and Manifest Serving API and return the relevant serviceability response back to the caller.
   a. If LaaP is serviceable, it takes higher priority over other 3pls.
   b. Else, Serviceability info for 3pls will be considered.
   c. In the future, the best of LaaP and 3PL will be returned

Non Functional requirements

1. Consistency
   a. All the writes to the service have to be consistent in SOR.
2. High availability
3. Fault tolerance
   a. The system should be resilient to faults at the application/caching/persistence layer.
4. High throughput
   a. To achieve high throughput, we are utilizing multiple layers of caching.
5. Low latencies
   a. Local caching combined with far cache helps us achieve low latencies. As we are expecting a high hit rate of local caching which will reduce the overall latency of the system.

In-house graph solution vs SaaS Graph offerings

We decided to go ahead with the in-house graph solution instead of SaaS offerings in the market due to the following reasons.

We did some level of benchmarking on various graph databases in the market and there were some limitations based on our requirements. Here is the sheet which contains the Graph DB Test Results for reference.

GRAPH DB	PROS	CONS
ArangoDB	High throughput on non weighted queries and node and link lookups. Inbuilt library support for K Shortest paths and All Shortest paths between any 2 nodes. Multi model datastore.	Cyclic dependency not handled while quering the K Paths. Low throughput with weighted queries. Enterprise offering comes with min 32 GB RAM per node in the cluster where as our data size is < 1 GB
Neo4j	Better performance on weighted queries	Required inbuilt graph traversal present only on Analytical server (AuraDS) AuraDS is not clustered and is designed for low throughput High Latency compared to Arango
RedisGraph	In memory DB (Befits our data size)	Being deprecated in near future Poor response times (initial impressions during POC)
Memgraph	In memory DB (Befits our data size)	Inbuilt query not present for k shortest paths Loading custom procedures for calculating k shortest paths was taking way longer Low throughput Cloud offering (all managed) is not mature for production level workloads as highlighted by their Sales personnel. License Cost for EE is on the higher side

Redis persistence

1. By using AOF on every write. AOF persistence logs every write operation received by the server.
   a. Redis persistence
2. The eviction policy will be set to no eviction.
3. The cluster config will be one master and 2 slaves. We will be not using clustered server here since the writes can happen on any nodes using that config on the basis of shard key slots.
4. Alerts will be set on memory usage.

Holiday infra

O2S Delay

Key attributes:

1. Dispatch commitments
2. Supplier Sunday working flag
3. State holiday

Approach:

1. The dispatch date is calculated in the route aggregator.
2. Core algorithm for o2s
   a. Calculate the max dispatch date based on commitments for a supplier based on dispatch commitments with state holidays considered.
   b. If the Sunday working flag is disabled for a supplier and Sunday lies in between queryDate and max dispatch date based on commitments then add another working day by considering holidays to the max dispatch date.
   c. The final dispatch date will be sent to the downstream auto DML system.

S2D Delay

Key attributes:

1. TAT at hubs
2. TAT at links
3. State holidays
4. Sunday delivery promise at LM hub level

Approach:

1. Get the EDD by the summation of TAT for hubs and links in the path.
2. Add a number of days based on if any state holidays lie within the EDD range.
3. If EDD lies on Sunday and if the Sunday delivery promise is false then add another buffer working day to the EDD.
4. Consider SMT promise relaxation for the final EDD calculation.

Future scope

1. Admin Panel UI to manage the entire network.
2. Auto DML should be capable of listening to certain events from the Meesho eco-system and take certain decisions based on the events like Disabling a hub when the shipment is within a hub for a long time etc
3. Graph implementation offering as a library.
4. Mature Holiday infra for Auto DML (Individual entity level tracking)
5. We should consider the cost of the overall path for prioritization.

Important metrics: Business

1. Count the number of shipments falling back to the 3pls from LAAP
   a. An issue with the downstream orch layer
   b. Capacity exhausted
   c. Paths exist with inactive nodes
2. Capacity exhaustion within x hours in a day
3. Shipment count via Auto DML and manual DML. The ultimate goal is to make manual DML=0.

FAQs

1. **Why are we keeping the entire hub and link data in Redis?**
   a. For now, the data size (inclusive of hub/link static details like Ops Tech Partner, and Hub Code) is pretty small. If the metadata grows in the future, we will decouple the primary deciding factors from the rest of the data (which might be stored in a disk-based persistence store).
2. **How many MMs can be connected with an FM or an LM?**
   a. An FM or LM can be connected to at max 2 MM hubs.
3. **Why are we precomputing MM-MM paths and not finding the path at runtime itself?**
   a. MM-MM has a many-to-many mapping with no limited connection limit. Hence to compute the MM-MM path at the runtime would add a significant amount of cases, and our algorithm computes all the paths between MM-MM in a significantly less amount of time.
   b. By computing the TAT and Cost in real-time, we are ensuring we will be committing the most optimal path (staleness can creep in for records updated within the K Path Generation Interval) with the correct TAT
4. **Are we ensuring the file uploads reflect in real time?**
   a. For order placement and manifestation, the real-time TAT, Cost, and Capacity (only for manifestation for now) are returned
   b. For the rest of the cases, we can have a max delay equal to the sum of the TTLs on all the caching layers.
5. **On what basis are we going to select the top k paths?**
   a. For V1, We will be selecting the path with the minimum number of connections/hops. If we find any paths with similar hops then we will sort it based on the increasing order of TAT.
   b. In future, we can have weights attached to every path which can be treated as the deciding factor as per the use case
6. **What happens if manifestation via Laap with optimal path fails?**

   a. Orders will be manifested via existing 3pls.
      Example- If there are two potential lane options for an FM Hub 1 to LM Hub 2 - one option with a total of 3 nodes and another option with a total of 4 nodes - the system will always choose the 3-node option, which represents the optimal path.
      If the capacity of the 3-node option becomes exhausted, the fallback should be to the end-to-end (E2E) 3pls rather than the 4-node option.

7. **Why did we choose hashset over key-value pairs in SOR?**
   a. We have a heterogeneous set of data, and if we use simple key-value pairs, then we have to add a prefix or suffix for each key which will consume additional storage.
8. **How can hubs be identified as “fm” and “lm” in the network generation process?**
   a. Every Hub will have a specific responsibility in the network. Hub cannot act as both MM and FMDC.