An in-memory Graph System for Scalable and Consistent Legacy System Integration

Bilal Arshad (PhD Researcher) – **UK Systems Research Workshop 30th Nov -2nd Dec 2021**

UNIVERSITY of DERBY

Agenda

- Introduction – Automotive Data Integration
- Dealer Management Systems – DMS
- Graph-based data integration
- Entity resolution in data integration
- Evaluation
- Conclusion

Automotive Dealerships - UK

- There are over 4K automotive dealerships within UK
- Each dealership could be part of a Franchise or be independent (multiple dealerships within a franchise)
- Data could be coming from various data sources with varying data types and formats (financial data-sets, telephony, sales, services etc)
- Each dealership could have its own format for data storage
- Unknown datasets coming from multiple data sources that require data transformation – Black Box
- Initially manual mapping is required to extract and transform data to store in a data warehouse

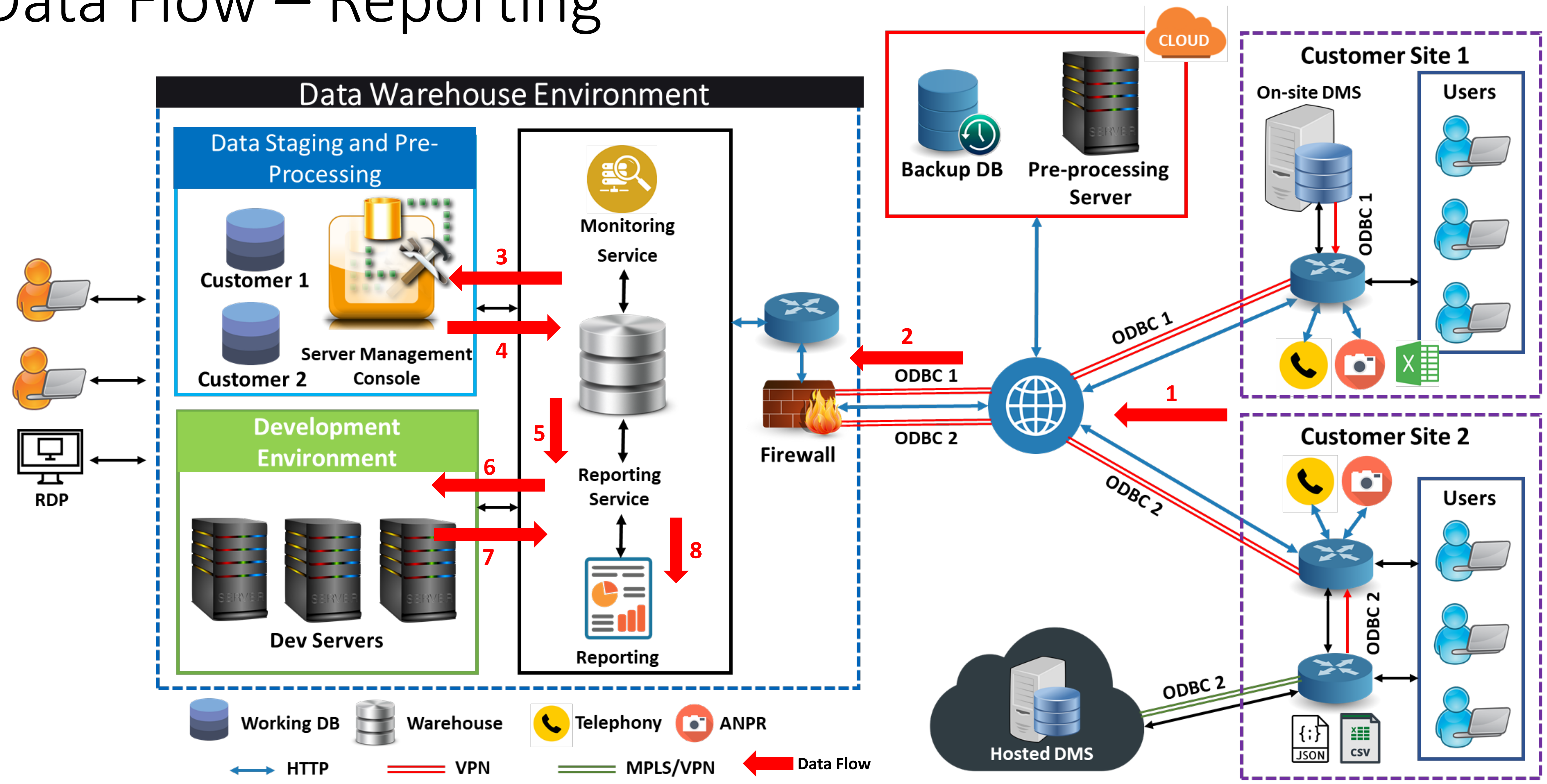
Automotive Data Integration

- In order to provide a 360-view of a dealership’s performance, data from multiple sources is integrated to provide a complete picture
- Management Information System (MIS) allows data from multiple sources to be brought together to provide a comprehensive real-time reporting dashboard with advanced analytics capabilities.
- Advantages of Integration
  - Feeds from management systems, telephony, account packages, sales tracking systems etc
  - Real-time data delivered as usable information
  - Acquire data from legacy systems – Dealer Management Systems (DMS)

Dealer Management Systems - DMS

- These dealerships have deployed Dealer Management Systems (DMS) to manage:
  - Vehicle sales stock
  - Customer leads
  - Service appointments
  - Online advertising appointment
- DMS are Proprietary Software provided by limited market leaders for automotive dealerships – closed source software
- Legacy Systems ~ approx. four decades old
  - Some of these DMS are quite old but limited choice forces dealerships to continue usage
- In order to extract data Windows based license is required – ODBC
  - Currently no support for Linux based licenses thus limitation to design systems around this requirement

Data Flow – Reporting

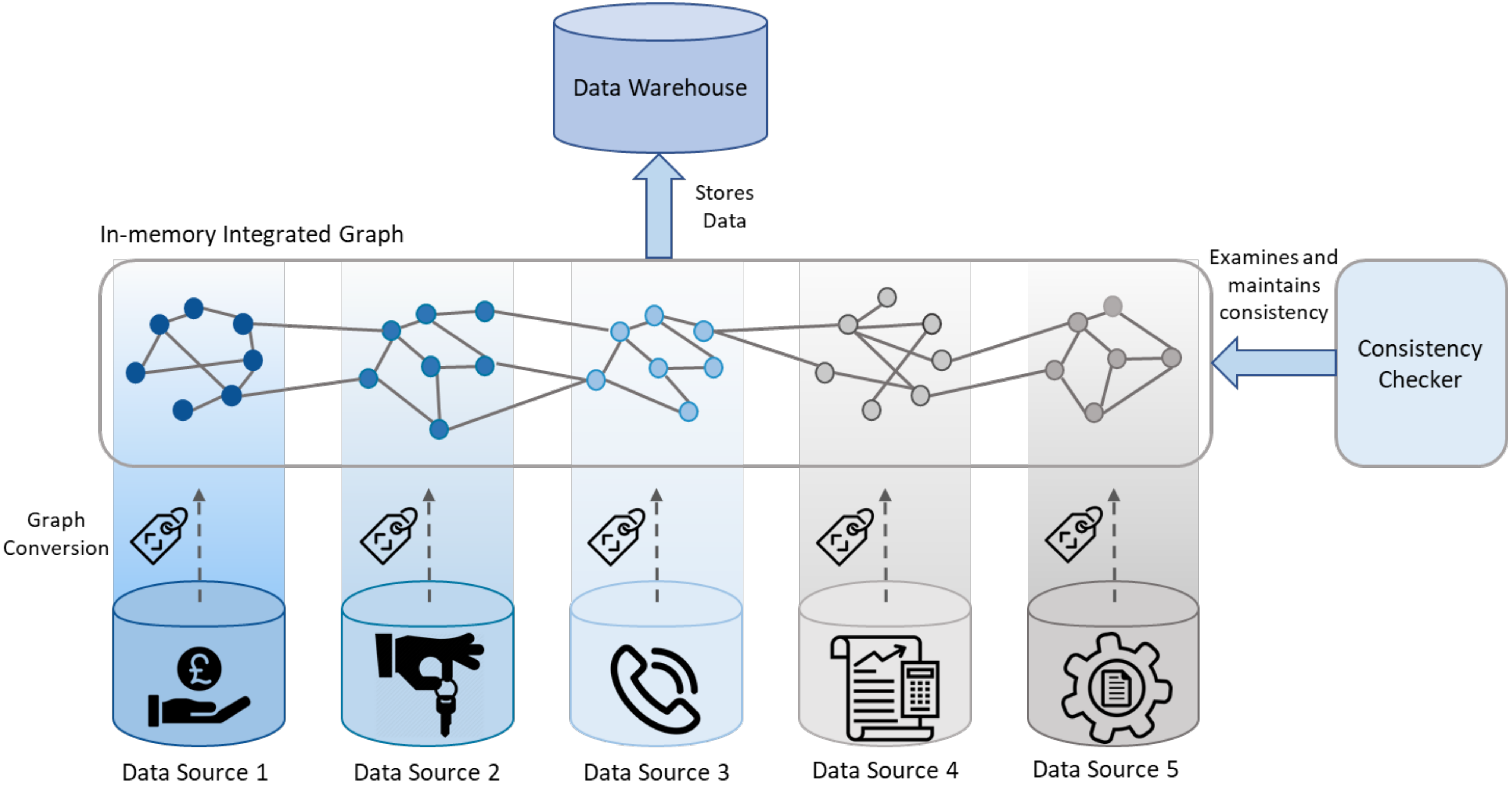


UNIVERSITY of DERBY

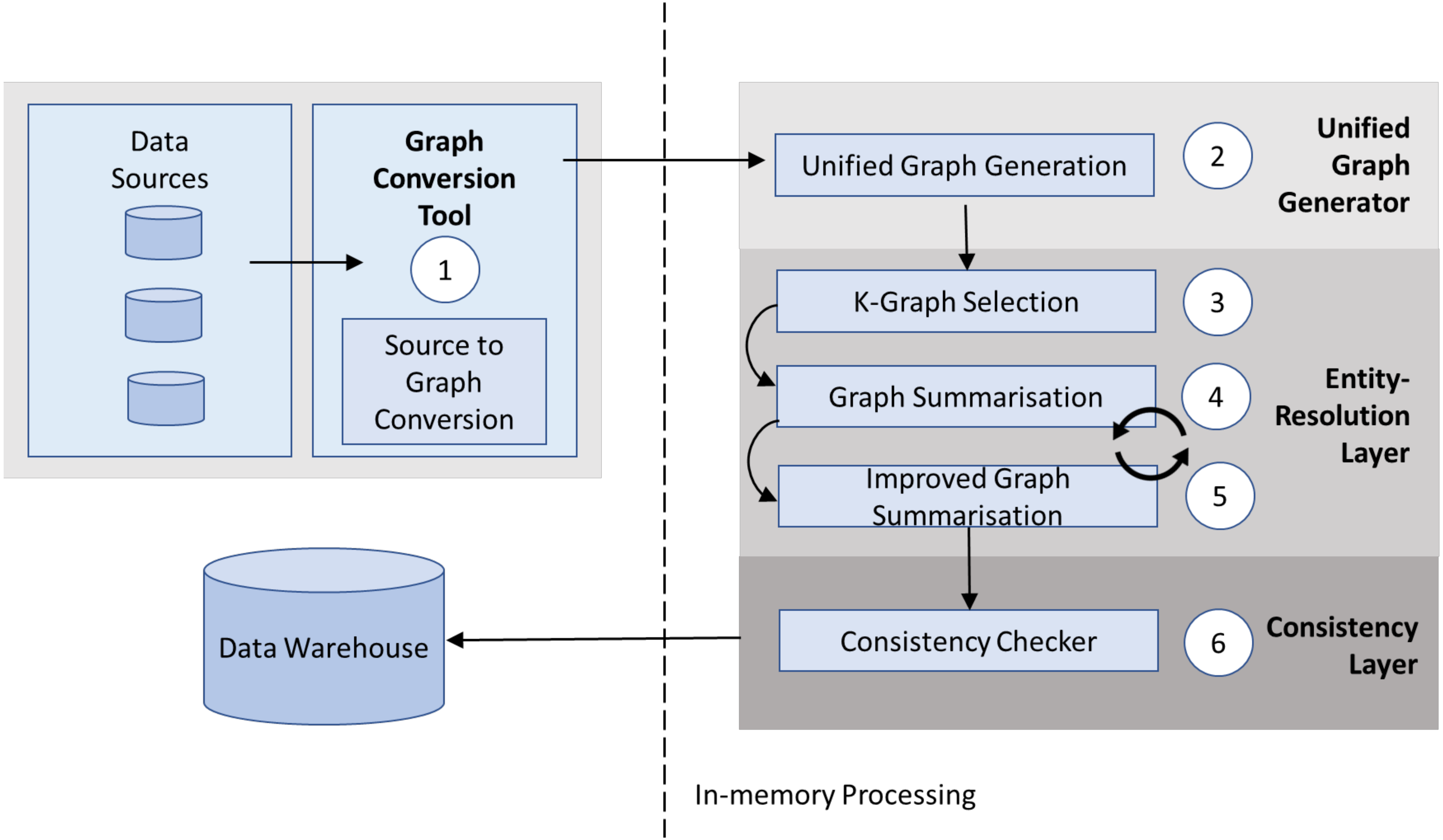
Issues in data integration

- Dealerships require integration of data from DMS and other data sources (telephony, ANPR, edge devices etc.)
- VPN links – cost
  - Solution is to set up an edge device that cuts down the VPN link's cost and send updated data only (lower bandwidth and processing cost)
- Issues:
  - Volume and velocity of change
  - Data consistency as these sources evolve

Graph-based Data Integration



Summarised Architecture



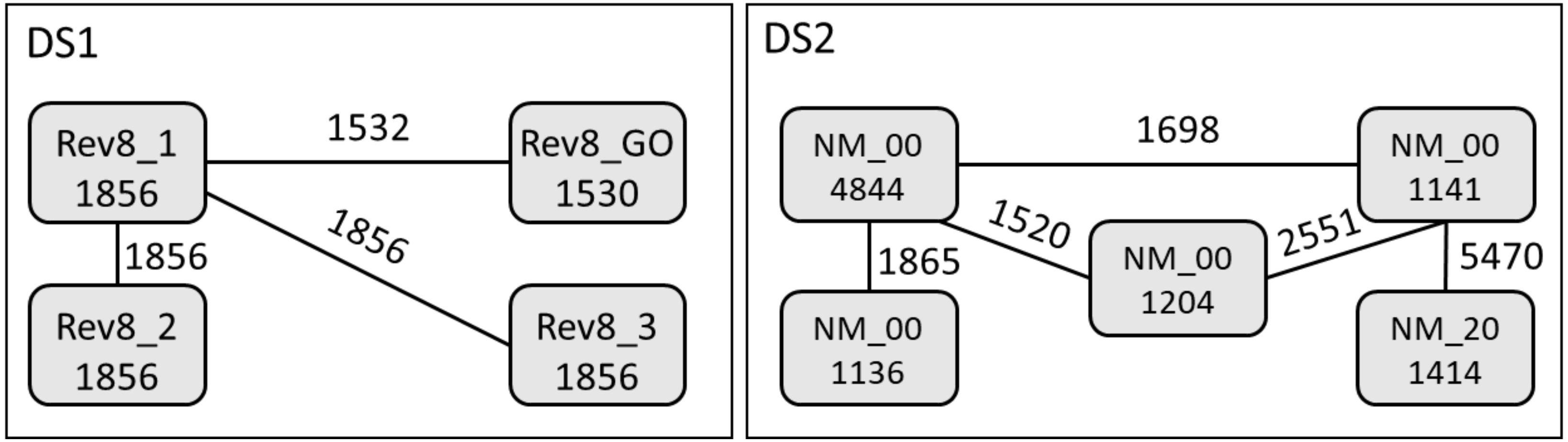
Data-sets for Evaluation

Datasets	Type	No. of Vertices	No. of Edges
Data-set 1	Real World	350	2875
Data-set 2	Real World	11600	65425
Data-set 3	Real World	25767	98598
Data-set 4	Real World	42494	109271
Data-set 5	Synthetic	65536	1048576
Data-set 6	Synthetic	131072	2097152

- We have collated data from three major DMS systems, Drive, Rev8 and Pinnacle
- The synthetic datasets are generated for scales 16 and 17 with average degree of 14 per vertex.
- The structures within these synthetic graphs are similar to the ones present in automotive data sets to ensure uniformity across the testbeds and results.

*DataSynth - Arasu, A., Kaushik, R. and Li, J., 2011. DataSynth: Generating synthetic data using declarative constraints. Proceedings of the VLDB Endowment, 4(12), pp.1418-1421.
*Graph500 RMAT - Murphy, R.C., Wheeler, K.B., Barrett, B.W. and Ang, J.A., 2010. Introducing the graph 500. Cray Users Group (CUG), 19, pp.45-74.

Entity Resolution Evaluation



Structures within DS1 and DS2 along with the number of links and entities

- A subset of the previously mentioned data-sets is presented for evaluation
- We begin by splitting these datasets into two named as DS1 gathered from Rev8 DMS and DS2 gathered from the Drive DMS.

Cluster Sizes in DI Phases

Cluster Size	Initial Clustering	Decomposition		Cluster Merge
		Type-based	Sim-based merge	
1	-	55	186	169
2	-	120	167	164
3	130	185	232	233
4	1690	1720	1604	1608

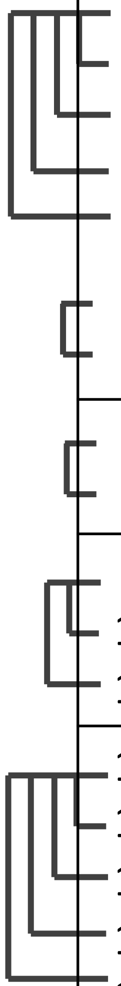
Cluster sizes in integration phases for evaluation dataset - DS1

Cluster Size	Initial Clustering	Decomposition		Cluster Merge
		Type-based	Sim-based merge	
1	-	44	353	327
2	720	796	841	833
3	756	771	818	805
4	1071	1013	439	1085
5	574	570	423	436

Cluster sizes in integration phases for evaluation dataset – DS2

Sample entities from datasets to express Entity Clustering

id	label	source	type
0	KerRidge	Drive	DMS
1	Kerridge	Rev8	DMS
2	Kerridge	Pinnacle	DMS
3	Ker ridge	Drive	DMS
4	ker Ridge	Pinnacle	-
5	kerRidge	Rev8	DMS
6	Kerridge	Rev8	DMS
7	Dlrcode	Drive	DLR
8	dlrcode	Rev8	
9	areasub	Pinnacle	Sub-category
10	areasub	Rev8	Sub-category
11	Areasub	Rev8	Sub-category
12	POS	-	TabCode
13	POS	Drive	TabCode
14	POS	Rev8	TabCode
15	Pos	Pinnacle	TabCode
16	POS	Drive	TabCode



0	KerRidge
1	Kerridge
2	Kerridge
3	Ker ridge
4	ker Ridge
5	kerRidge
6	Kerridge
7	Dlrcode
8	dlrcode
9	areasub
10	areasub
11	Areasub
12	POS
13	POS
14	POS
15	Pos
16	POS

SplitMerge for Entity Resolution in Graphs

Algorithm 5: SplitMerge Clustering Algorithm

Input: Set of entities e from n sources, edge set d, simFun f_{sim}, thresholds t_s,t_m
Output: Set of clusters C

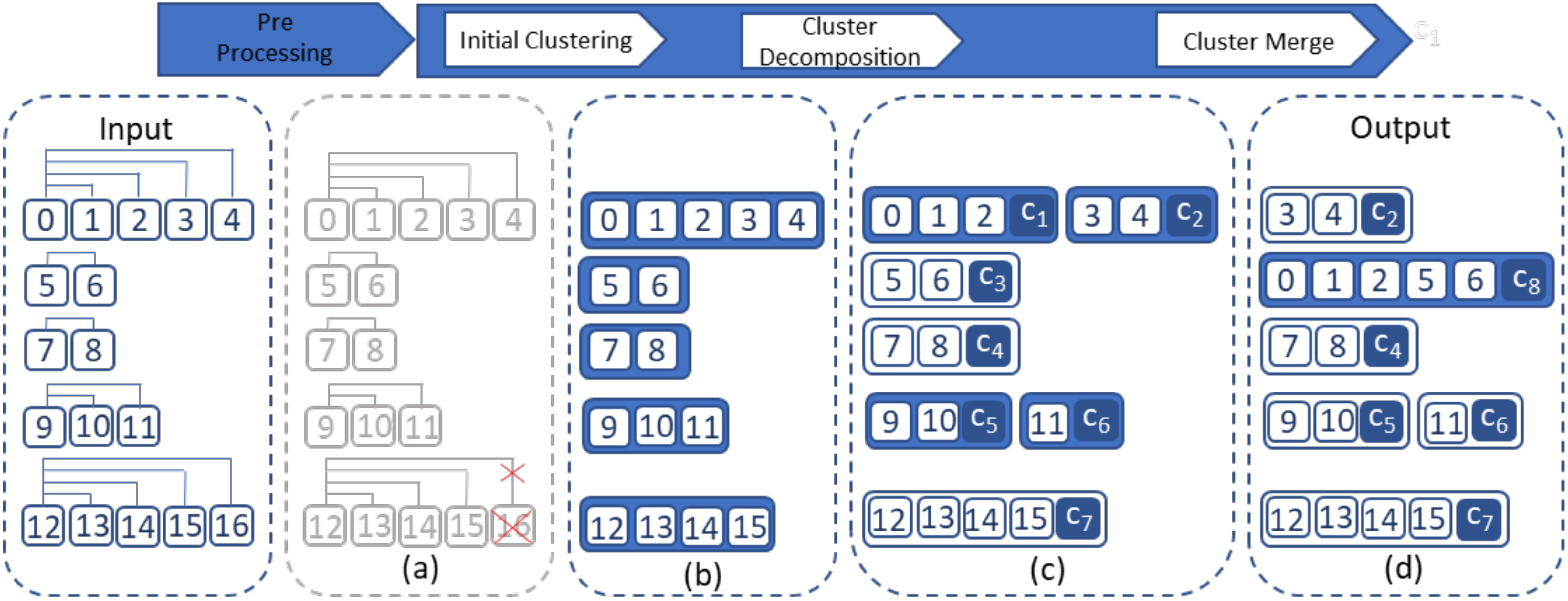
```
C ← ∅
e, l ←
preprocessing(e,l,f_sim) /*preprocessing*/
C_init ← computeConnectedComponents(e,l)
l_c ← computeLinksSim(C_int,f_sim)
C − int ← refineConnectedComponents(C_int,L_c) /*initial clustering*/
foreach c ∈ C_int do /*cluster decomposition*/
C_split ← groupByType(c,L_c)
C_split ← simBasedRefinement(C_split,L_c,t_s)
C_split ← createRepresentatives(C_split)
C ← C ∪C_split
CM ← computeClusterSim (C,f_sim,t_m) /*create cluster mapping CM */
while CM ≠ ∅ do
(c_1,c_2) ← CM.getBestMatch()
c_m ← mergeClusters(c_1,c_2) C ← C  c_1,c_2 ∪ c_m /*cluster merge*/
CM ← adaptMapping(CM,C,c_m,c_1,c_2,f_sim,t_m)
return C
```

SplitMerge Phases:

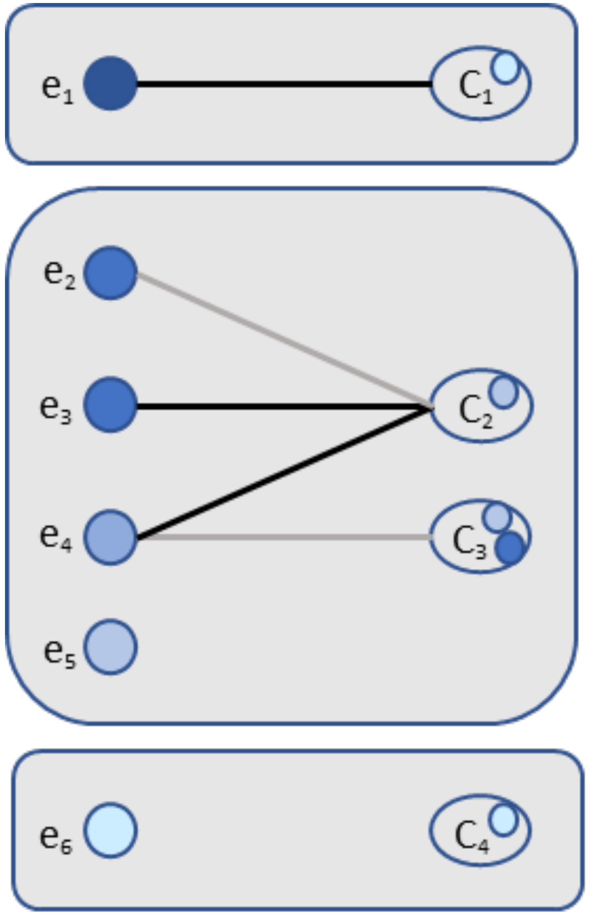
Note: *f_{sim} = Similarity function; t_s, t_m = similarity threshold*

1. Preprocessing
   - property values required for similarity computation are normalized
2. Initial Clustering
   - Connected components
   - In order to phase out deduplicated entities and the refineConnectedComponents (Line 5) connected components is used on one entity per source.
3. Cluster Decomposition – two main approaches
   - Split clusters based on inconsistent semantic types
   - clusters containing inadequate similarity to other cluster members are split up
4. Cluster Merge
   - merge clusters that range below the maximally possible cluster size k

SplitMerge Example continued…



Incremental Clustering Approach



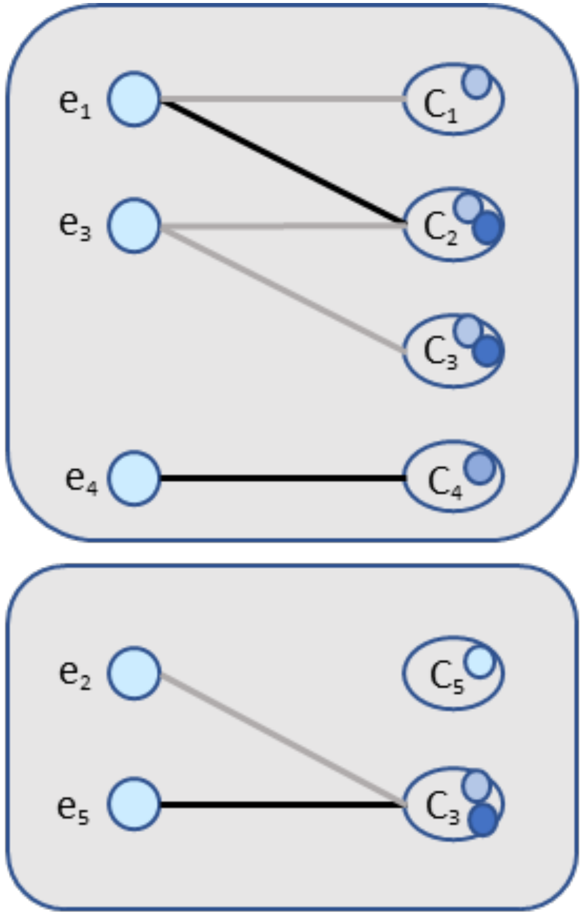
(a) Root Approach

Algorithm 8: Set-based incremental entity clustering - root approach

Input: Existing clusters C_{exist}, edge set l_c, split threshold t_s
Output: Set of clusters C_{result}
$C_{new} \leftarrow$ createInitialClusters(E_{new},$f_{blocking}$)
$C_{exist} \leftarrow$ addBlockingInfo (C_{exist},$f_{blocking}$)
for *block i* ***in Parallel*** **do**
$L_i \leftarrow$ getClusterCandidates (C_{exist},C_{new},f_{sim},t_{min})
$L_{sorted} \leftarrow$ sortLinkSim(L_i)
foreach $(c_{new}, c_{exist}, sim) \in L_{sorted}$ **do**
if $c_{exist} \notin c_{new}$ **then**
continue()
if isSrcConsistent(c_{new},c_{exist}) **then**
c_{exist}.add(c_{new})
C_{new}.remove(c_{new})
return $C_{exist} \cup C_{new}$

- Two different scenarios for cluster generation:
  - root approach and
  - source-specific approach
- (colours signify various data sources, it is assumed that all the links exceed the minimal similarity threshold)

Incremental Clustering Approach



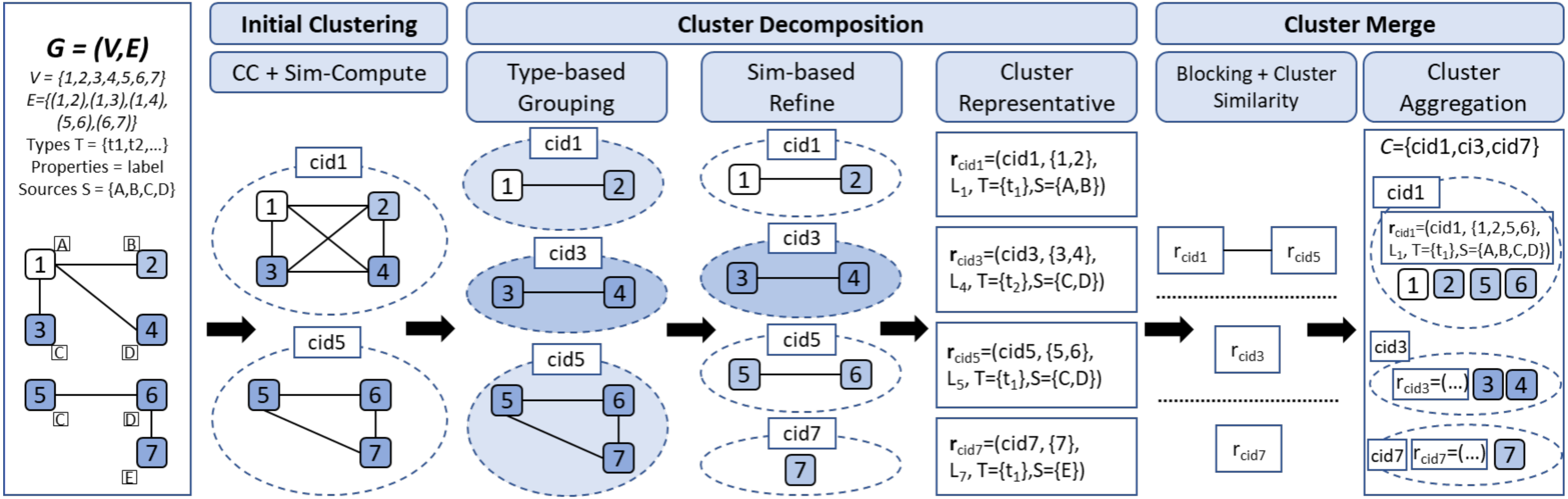
(b) Source-specific Approach

Algorithm 5: SplitMerge Clustering Algorithm

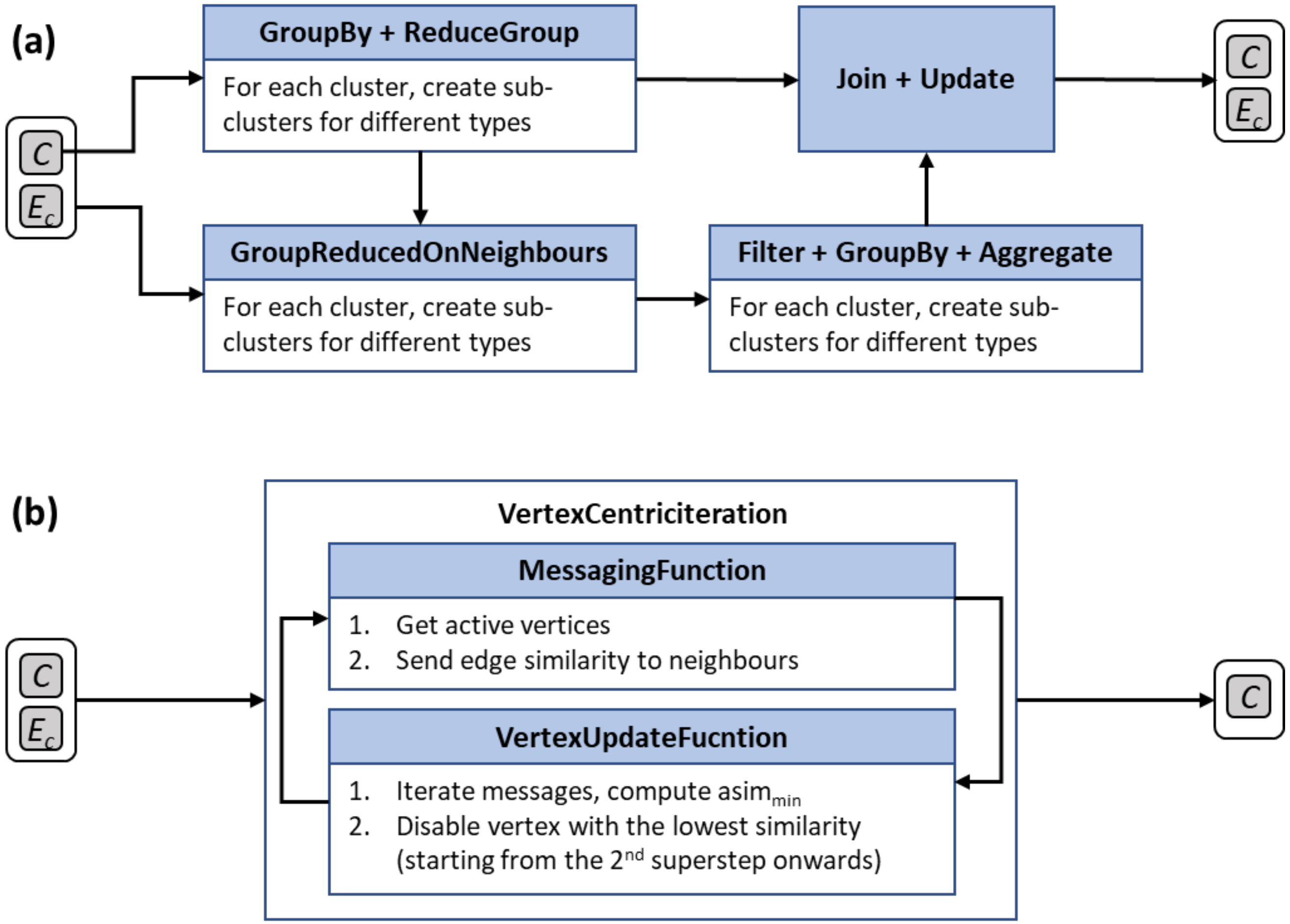
Input: Set of entities e from n sources, edge set d, simFun f_{sim}, thresholds t_s,t_m
Output: Set of clusters C
C $\leftarrow \emptyset$
$e, l \leftarrow$
preprocessing(e,l,f_{sim}) /*initial clustering*/
$C_{init} \leftarrow$ computeConnectedComponents(e,l)
$l_c \leftarrow$ computeLinksSim(C_{int},f_{sim})
$C - int \leftarrow$ refineConnectedComponents(C_{int},L_c) /*initial clustering*/
foreach $c \in C_{int}$ **do** /*cluster decomposition*/
$C_{split} \leftarrow$ groupByType(c,L_c)
$C_{split} \leftarrow$ simBasedRefinement(C_{split},L_c,t_s)
$C_{split} \leftarrow$ createRepresentatives(C_{split})
C $\leftarrow$ C $\cup C_{split}$
CM $\leftarrow$ computeClusterSim (C,f_{sim},t_m) /*create cluster mapping CM */
while CM $\neq \emptyset$ **do**
(c_1,c_2) $\leftarrow$ CM.getBestMatch()
$c_m \leftarrow$ mergeClusters(c_1,c_2) C $\leftarrow$ C c_1,$c_2 \cup c_m$ /*cluster merge*/
CM $\leftarrow$ adaptMapping(CM,C,c_m,c_1,c_2,f_{sim},t_m)
return C

The algorithm resolves source-consistent candidate links between the newer entities and existing set of clusters in parallel with partitioned blocks.

Distributed Clustering

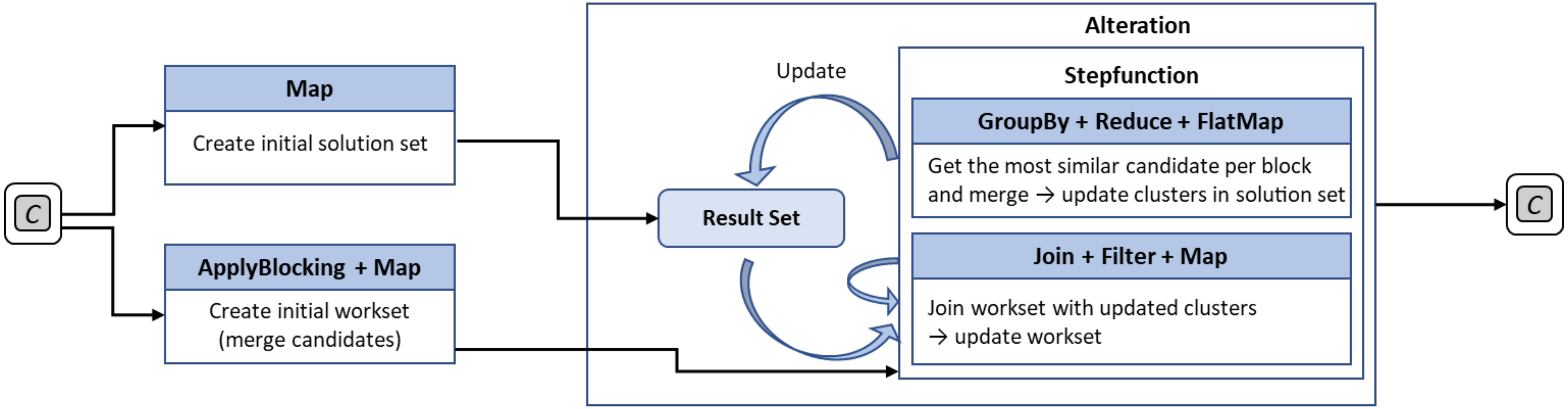


Cluster Decomposition - Distributed ER



- Sub-workflows for type-based grouping shown in (a) and similarity-based refinement in (b)
- The first step in decomposition of the cluster is to break the clusters into sub-clusters based on the compatibility of property types (Type-based grouping).
- The second step is to decompose these clusters using non-similar entities from clusters based on the step Similarity-based refinement.

Cluster Merge – Distributed ER



- Spark’s iterative operative in addition to user-defined functions are used to address the final merge stage.
- Clusters with high similarity which as usually small are aggregated iteratively into large clusters.
- The creation of representatives for each of the cluster enables to reduce the number of potential entities for the merge step.

Evaluation

- Evaluation of these data-sets are based on dynamic graph queries. For each data-set we grouped the queries in five sets (i.e. ten queries per set): each set is homogeneous with respect to its complexity of the queries (e.g. number of connectedcomponents, number of results and so on.).
- For instance, referring to integrated Rev8 data-sets, the first set of queries searches information about services while the second set of queries seeks information about sales.
- For each set, we ran the queries ten times and measured the average response time.

Cluster Sizes and Configuration Results

	Data-set	Node Properties	No. of Nodes	No. of Sources
DS1-A1	Drive	DOC	5,079	4
DS2-C1 DS2-C2	Rev8	POS DlrCode	11,600 42,949	5 5
DS3-N1 DS3-N2	Pinnacle	PinCode CustRef	131,072 500,000	5 10

- Evaluation data-set details

	Perfect Result		Best configuration -results		
	# of clusters	# of edges	conf(t_min,bk)	# of correct edges	F-measure
DS1-A1	790	6497	conf(0.4,1)	6,207	0.981
DS2-C1 DS2-C2	5000 20000	10,340 39,321	conf(0.5,1) conf(0.7,1)	9,589 36,956	0.953 0.846
DS3-N1 DS3-N2	110,440 350,960	101,843 619,528	conf(0.7,6) conf(0.7,6)	100,057 513,975	0.804 0.795

Evaluation of Static vs Dynamic Clustering

DS2 - C2	Incremental		Static	
	Root	Source	CLIP	SplitMerge
run time (sec)	4210	1052	1859 + 72	1859 + 732
Precision recall	0.765 0.865	0.897 0.839	0.868 0.819	0.848 0.833
F-measure	0.812	0.879	0.855	0.845

DS3 – N1	Incremental		Static	
	Root	Source	CLIP	SplitMerge
run time (sec)	642	221	110+105	110+763
Precision recall	0.565 0.844	0.817 0.821	0.860 0.819	0.789 0.862
F-measure	0.676	0.819	0.846	0.832

- Data Quality and Run time for:
  - DS2-C2
  - DS3-N1

Consistency Phases

Phase 1

- Static Graph Computation (Compute graph properties e.g. graph properties such as vertex parents, vertex degree)

Phase 2

- inconsistency_list (parameters – update batch and user defined priority) is responsible for keeping track of changes (similar to log-replication in Raft). The update batch consists of edge or vertex insertions and/or deletions from which a list of inconsistent vertices is built after applying updates.

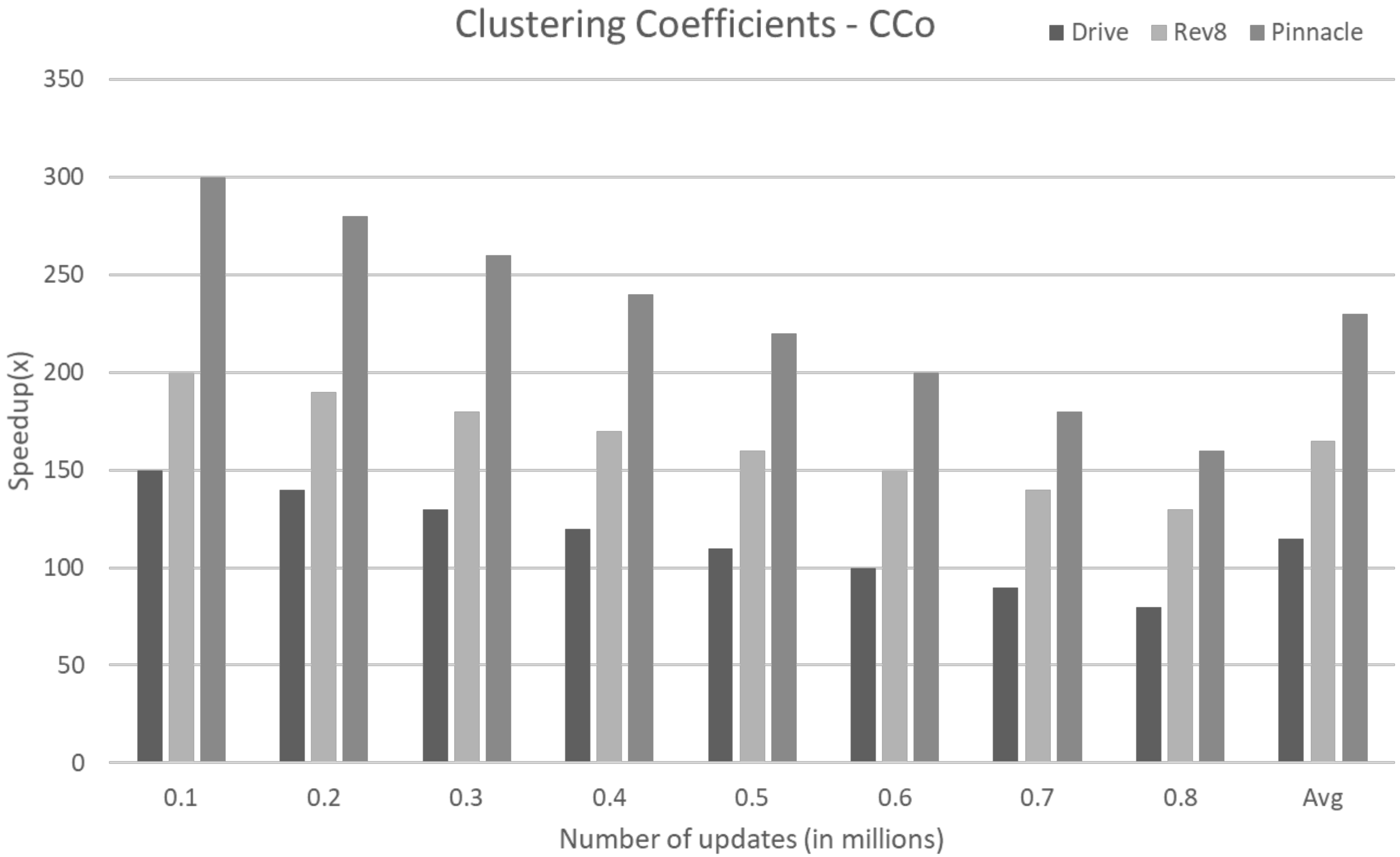
Phase 3

- consistency checker element ensures that only the inconsistent part of the graph is recomputed incrementally as opposed to the entire graph
- consistency algorithm loops iteratively over the inconsistency_list until the consistency list becomes empty.

Phase 4

- merges the graph states to generate the final version of the graph. This includes updating vertex property information and insertion/deletion of edges into the most recent version of the graph.

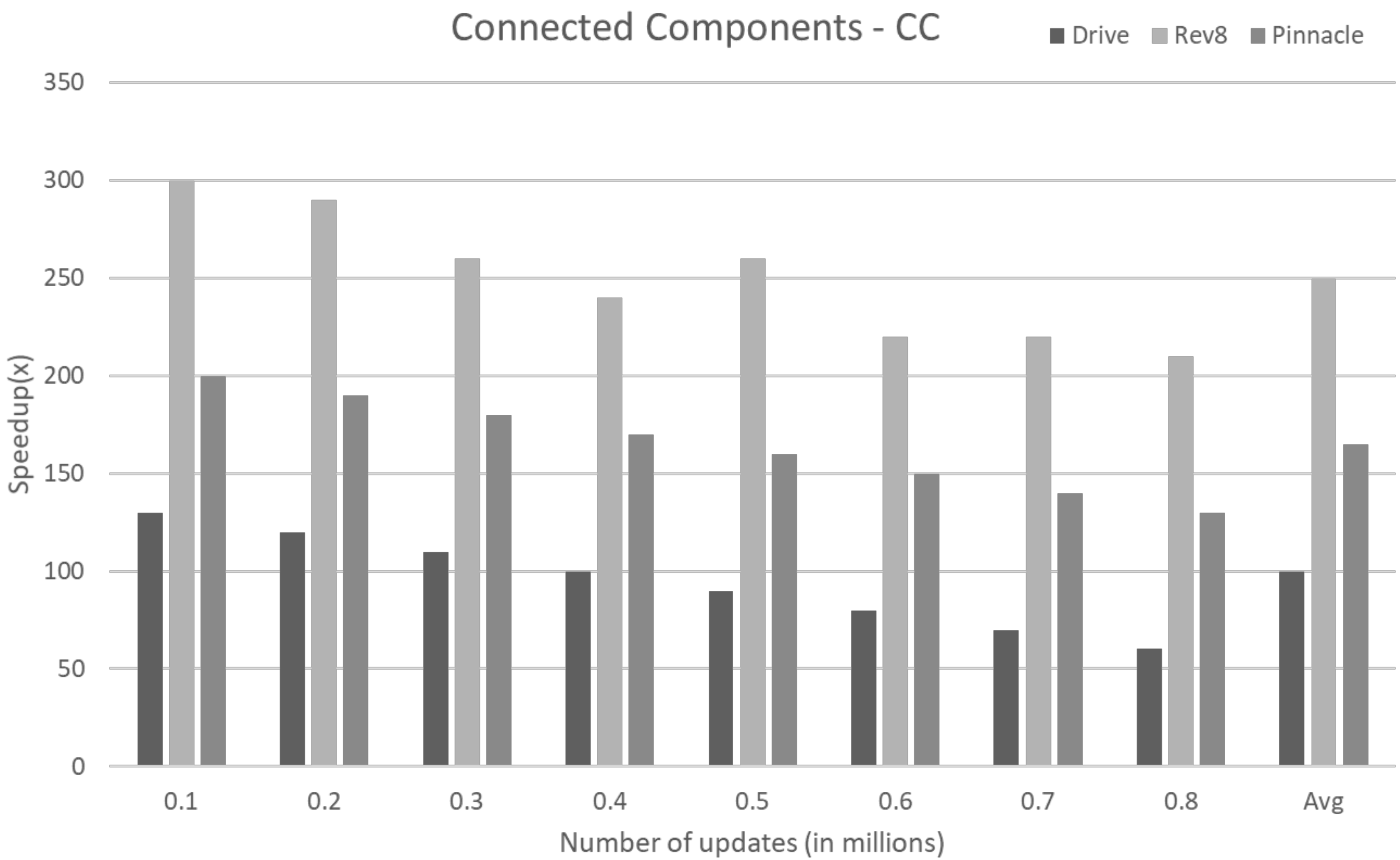
Evaluation – static vs incremental computation (CCo)



Incremental Speedup over static execution verses the update batch size for Clustering Coefficient

UNIVERSITY *of* DERBY

Evaluation – static vs incremental computation (CC)



- Incremental Speedup over static execution verses the update batch size for Clustering Coefficient

Conclusion

- Entity Resolution techniques combined with graphs result in quicker and scalable data integration
- Test the efficacy of the approach on other domains – currently tested approach on clinical datasets (limitations due to limited public data availability)
- Further enhance the solution to provide performance and scalability guarantees
- Employ ML and AI techniques to automate the report generation process within a cloud-based environment

Thank you! Questions?

b.arshad@derby.ac.uk
@bilalarshadali