

Using Binary Attribute Tree to Improve Multi-Attribute and Range Queries' Response Time

Sunway University College
School of Computer Technology

Goh Chiao Wei

Dr Lim Tong Ming

Introduction

- Demand for data processing grows extensively.
- Traditional client/server architecture faces bottleneck by such huge data processing demand.
- Attempt to use our home grown P2P OODB to provide:
 - More productive database platform using OO model through class reuse
 - Remove impedance mismatch issue as software is developed using object oriented programming language
- This proposed OODB is built on top of OpenChord, an implementation of Chord, a Distributed Hash Table (DHT) system.

Problem Statements

- Traditional client/server architecture is not scalable as network size scales
- Moving (wifi connected Netbooks in a large warehouse for example doing goods picking and packing activities) nodes have frequent join/leave characteristics
- Traditional client/server architecture becomes bottleneck when incoming queries exceed the processing ability of the server
- Unstructured P2P does not guarantee the availability of the desired objects/data due to flooding search approach
- Unstructured P2P might take longer routing time to the destination node – not suitable for a database overlay network

Objectives

- Search objects in OODB over structured P2P overlay network.
- Support multi attribute search in DHT.
- Support range search in DHT.
- Improve queries response time (some reviewed works [2], [3]).

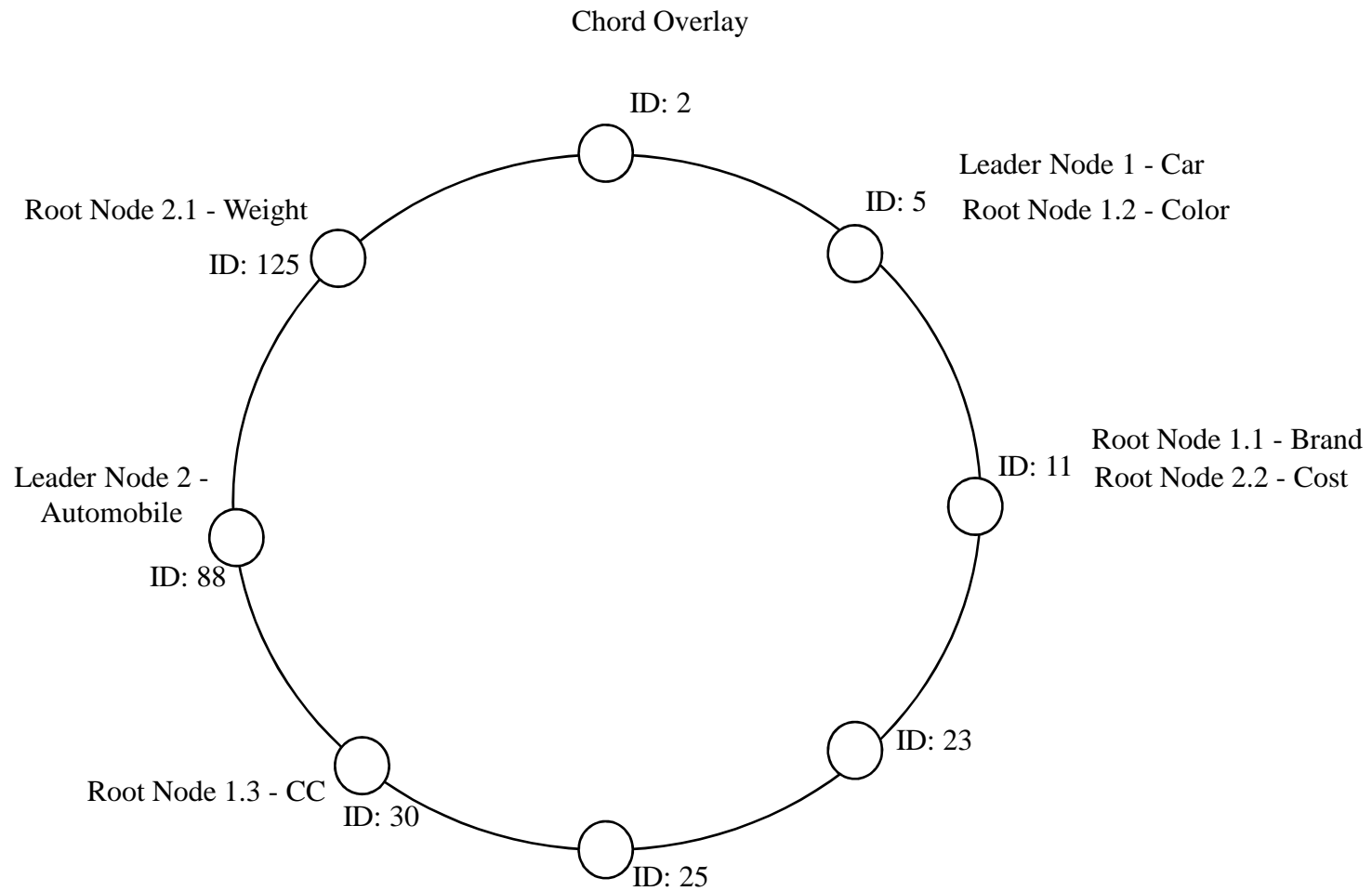
Literature Reviews

- Prefix Hash Tree[1]
 - Search data items stored at the leaf nodes which has the longest prefix match between the node label and the item.
 - **Cons:** This technique is intended for searching n-bit keys with only binary value.
- Mercury[3]
 - Attribute hub forms a ring topology and each node in the ring topology handles a certain range of numeric values.
 - **Cons:** It searches the data in ring topology that might takes longer response time.
- Range Query Model based on DHT in P2P System[2]
 - Several B+ trees are form to responsible for different attributes.
 - Nodes in B+ trees are mapped to the nodes in Chord using the hash function provided by Chord.
 - **Cons:** The queries must always start with the root nodes of the trees.

Why structured DHT?

- Promise to deliver the desired object set as long as the objects exist in (any) peers on the overlay network.
- Guarantees search request reaches the destination nodes in logarithm loops, which greatly reduces query response time.
- Provides index facility to map data to the nodes, in which each data item is hashed to a key value and store the key value into the successor node for the key value.

Chord Overlay Depiction



Overall Proposed Solution: Structure of attribute trees for a class (Example: Car Class)

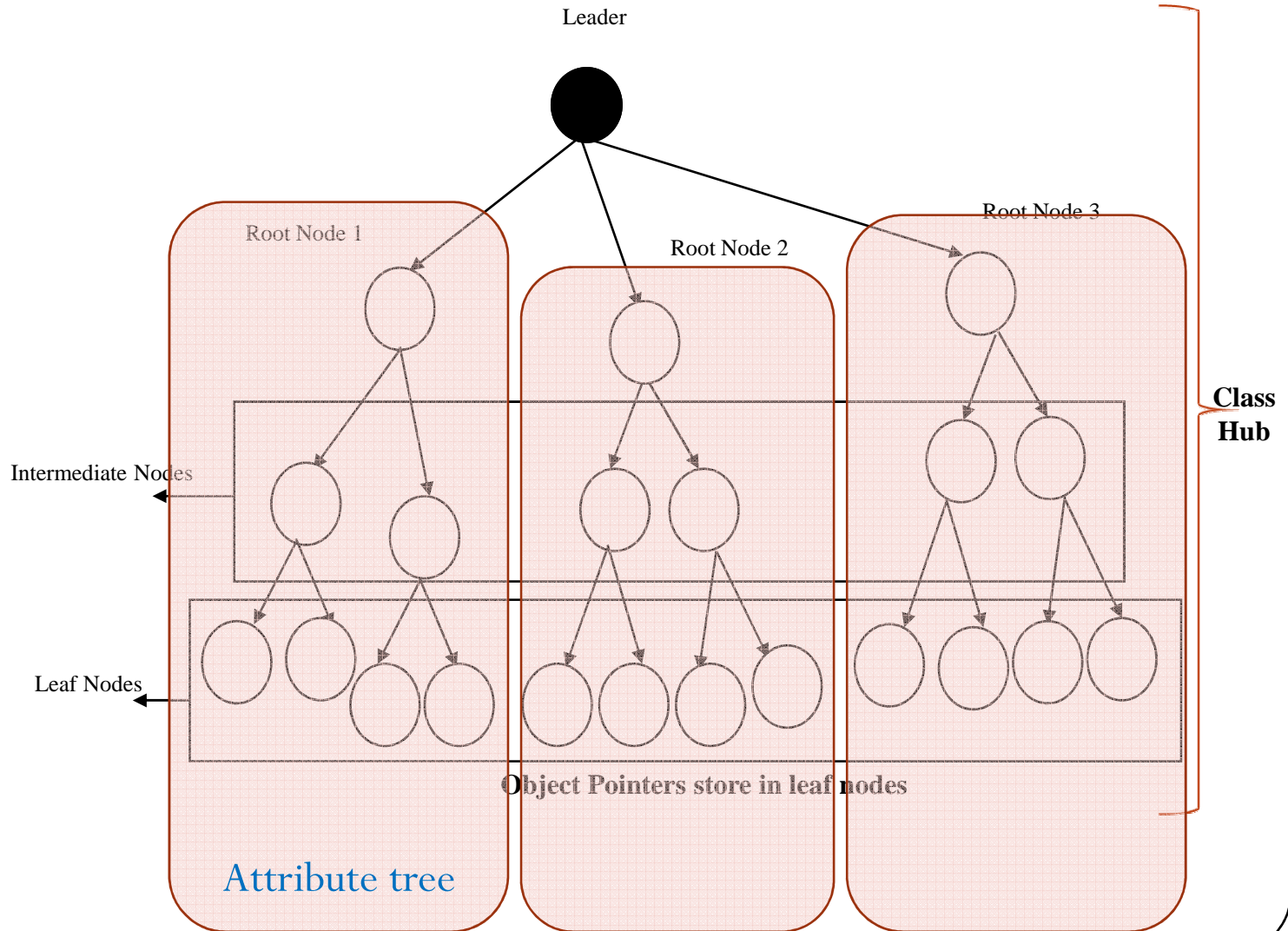
•**Leader Node** for **Car** class is the node with node id 5 in the Chord overlay.

•**Root Node 1** is the root node for attribute **Brand** for the car class, map to the node with node id 11.

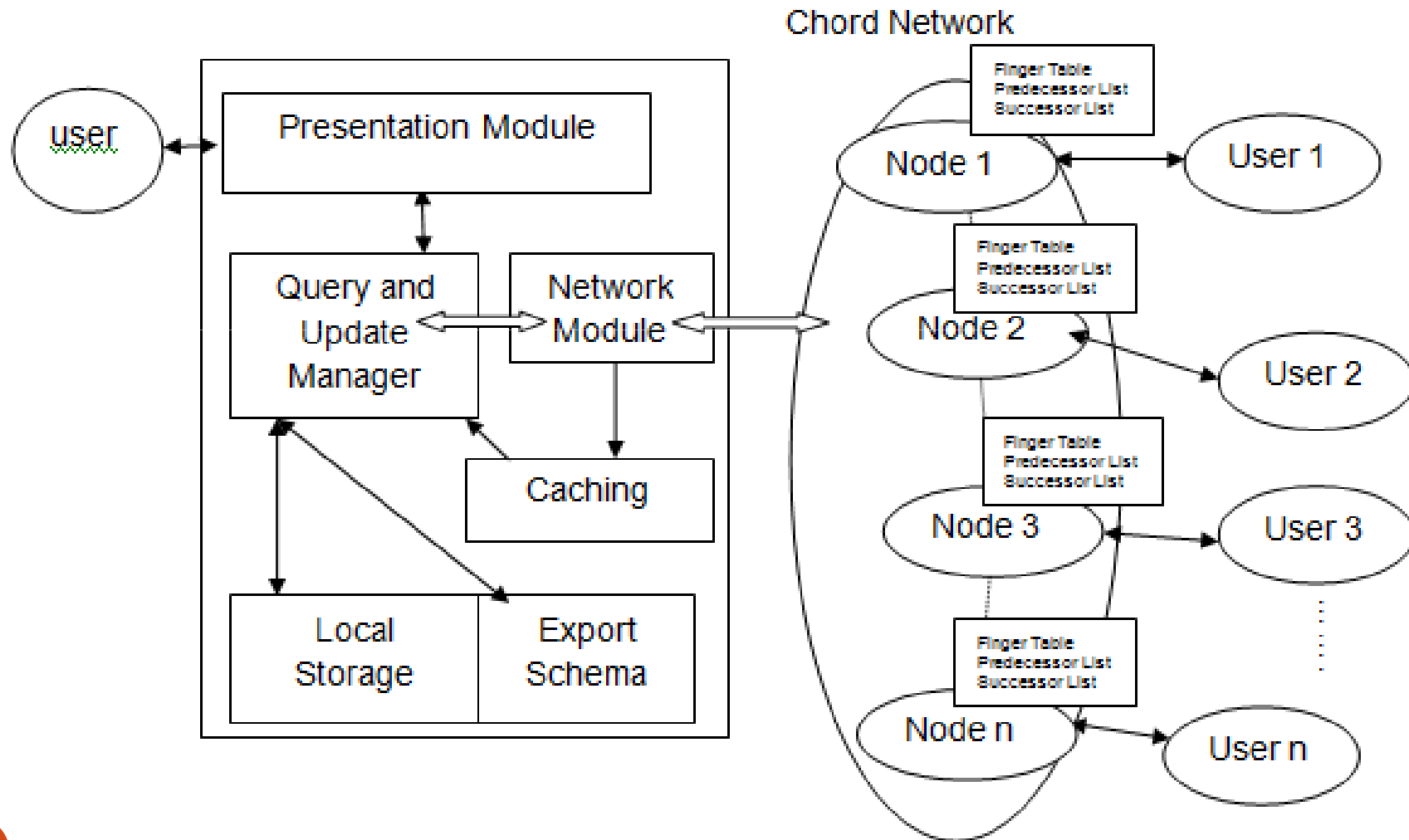
•**Root Node 2** is the root node for attribute **Color**, map to the node with id 5.

•**Root Node 3** is the root node for attribute **CC**, map to the node with id 30.

•Detail of **Intermediate** and **Leaf Nodes** are in subsequent pages.



Our Prototype: POMS Node Structure



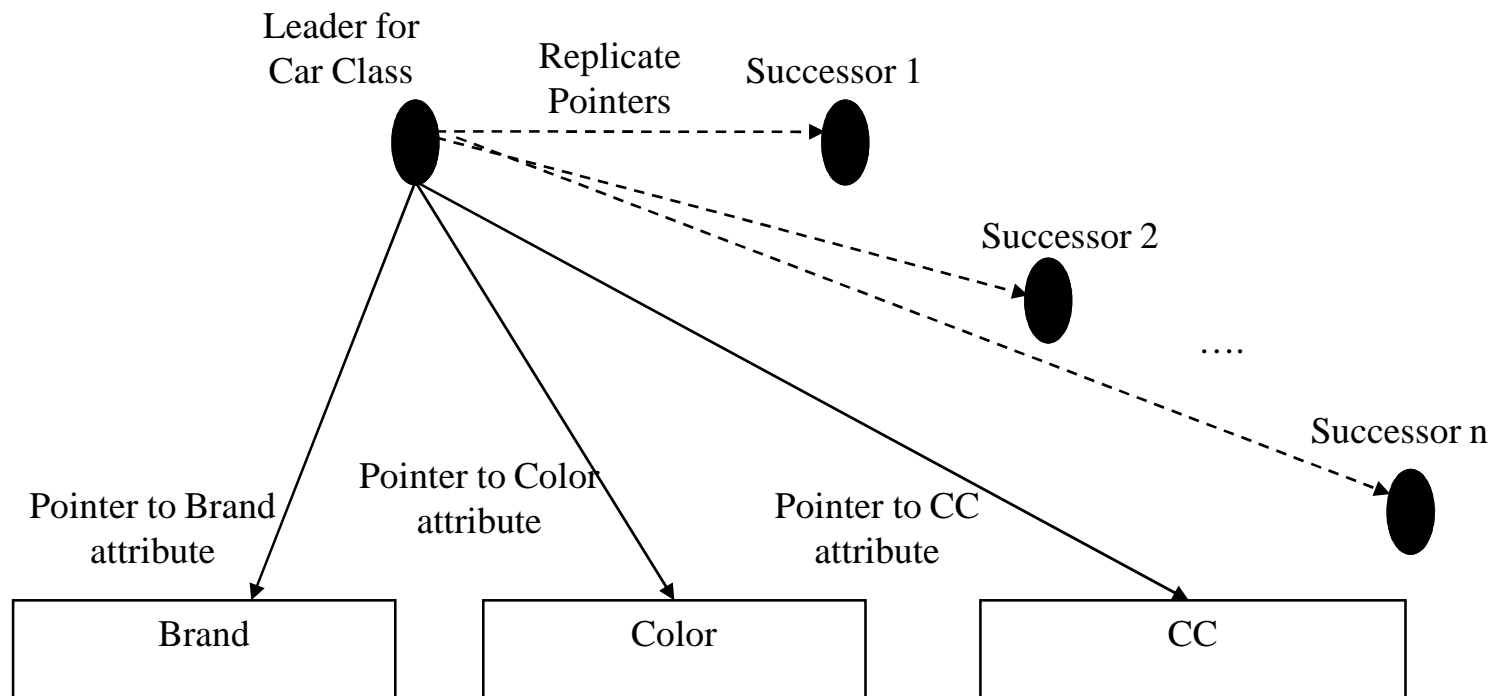
- Nodes organized in ring topology.
- Few core modules in POMS are:
 - Presentation Module – the bridge between user and the system, providing GUI.
 - Query and Update Manager - for the data query, update data and concurrency control.
 - Caching – caches previous object sets and node information of the *Attribute Tree*.
 - Local storage - local database in which it stores all the objects in the local machine.
 - Export Schema - allows the objects in the schema to be shared and modified by other nodes.
 - Network Module – connect the local node to other nodes in the network.

Object Pointers

- Object pointers are the pointers leading the search request route to the location of the actual objects of database.
- Store in the leaf nodes of the attribute trees.
- Store in the form of $\langle \textit{Node Identifier}; \textit{OID}; \textit{Value} \rangle$
 - *E.g.* $\langle 25; \textit{car@nmj222}; 850 \rangle$

Class Hubs

- Class hub is a structure that provide
 - Pointers that point to groups of all the related nodes involved for a particular class.
- Leader is the gateway that passes the query searching of the class into the correct attribute trees.
- Leader is selected based on the hashing of class name.
- Leader holds the pointers to the root nodes of the attribute trees.



- Refer to page 7 and 8 for the mapping of leader and root nodes into the Chord overlay network.
- Leader node replicates all the pointers of the root nodes to n successor nodes, where n is determined by user.
- This is to avoid single point of failure in leader node and share the workload s of leader node to avoid bottleneck occur in leader node when too much queries arriving leader node.

Attribute Trees

- An *Attribute Tree* corresponds to an attribute in a class.
- Logical tree that accommodates pointers of the actual object location (object pointer) for the values of the corresponding attribute.
- Each node has at most two children node.
- Three types of nodes:
 - Root node – The first node of the tree. Aware the total number of object pointers in the tree.
 - Intermediate nodes – Nodes in between root node and leaf nodes. Store information of sub-tree below them and pointer of their parent node.
 - Leaf nodes – Nodes at the bottom level of the tree. Store the object pointers.

- Mapping of nodes are not one-to-one relation.
- Node in Chord maps to the node in tree by hashing the range of values it responsible.
- Two operations in the attribute tree:
 - Split – Density of the leaf node exceed the threshold.
 - Merge – Density of the leaf node below the ratio.
- Example of **Car** class is depicted in [page 12](#) and [15](#).

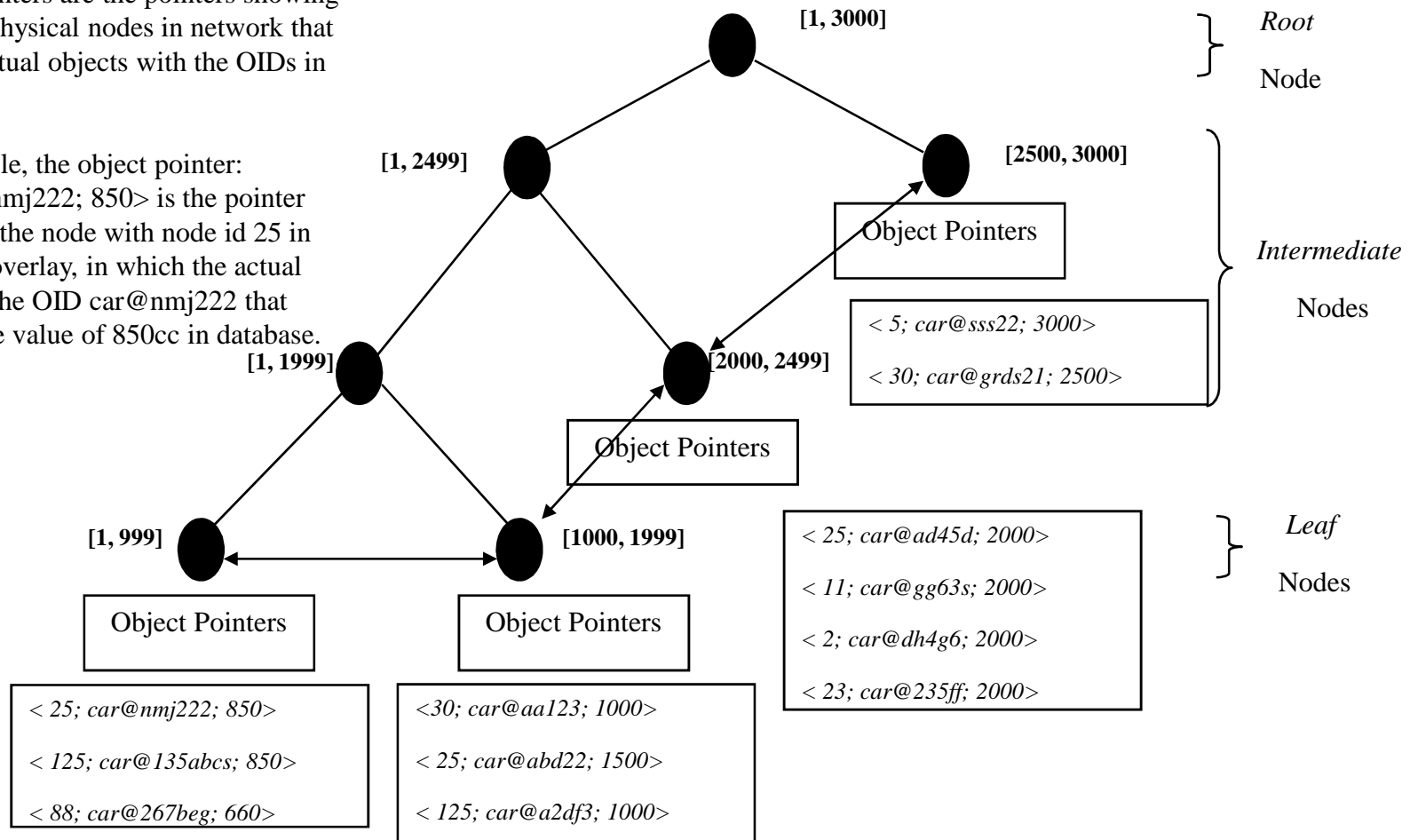
Sample Attribute Tree

•Object pointers are store in leaf nodes.

•Object pointers are the pointers showing the actual physical nodes in network that store the actual objects with the OIDs in database.

•For example, the object pointer: $\langle 25; \text{car@nmj222}; 850 \rangle$ is the pointer pointing to the node with node id 25 in the Chord overlay, in which the actual object has the OID car@nmj222 that contains the value of 850cc in database.

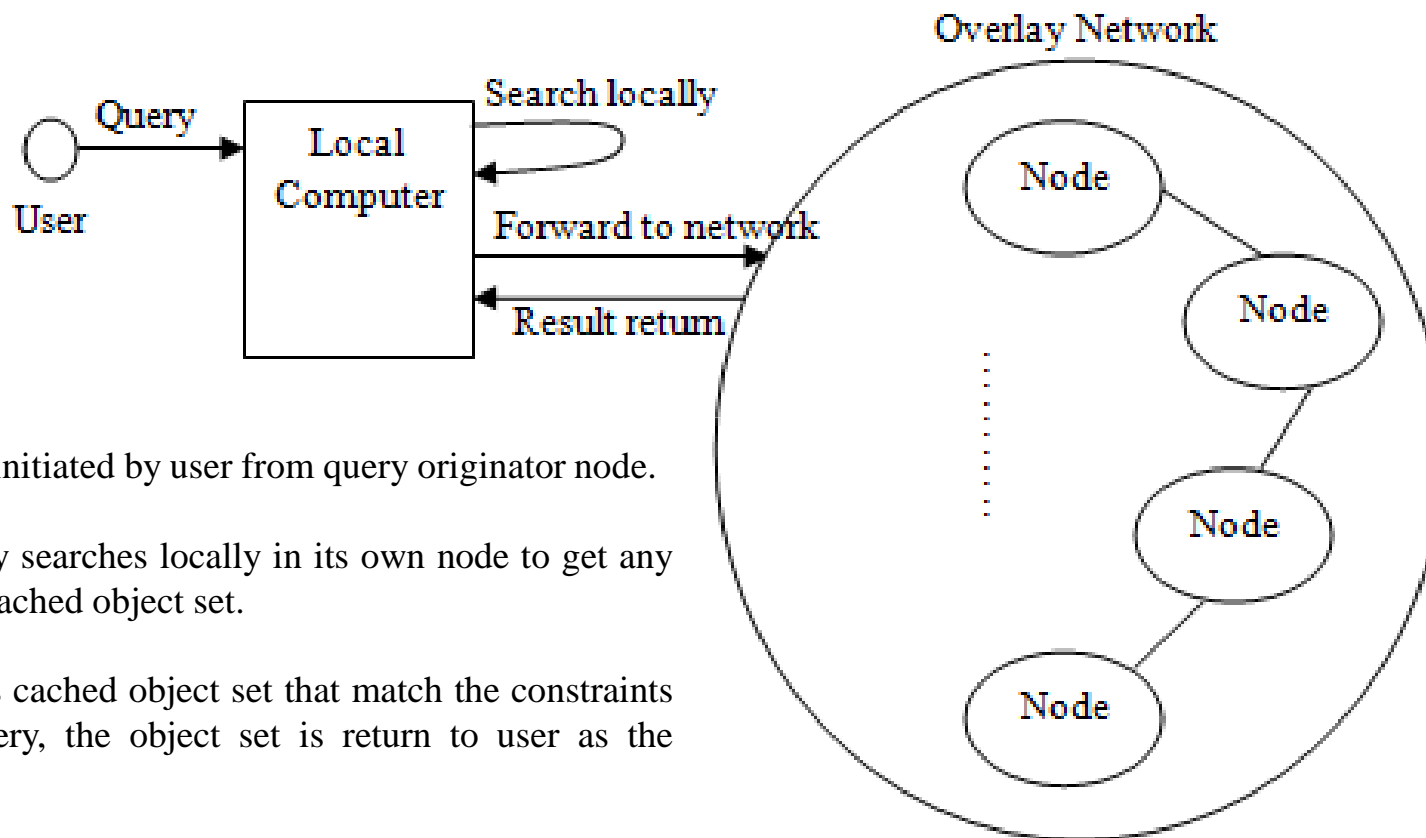
CC Attribute Tree for Car class hub



Query

- Check object set cached in caching manager first.
- Forwards query to the leader of the class hub.
- Leader forwards query to root nodes of correspond attribute trees.
- Query is then forwarded to the corresponding node in the tree.
- Result is send back to the query originator upon successful operation.
- The result is compiled into a list and sends request to those node that hold the actual objects.
- Object set is cached into the query originator and the object set is cached into the nodes along the path of how the query is traversed too.

High level view of query process



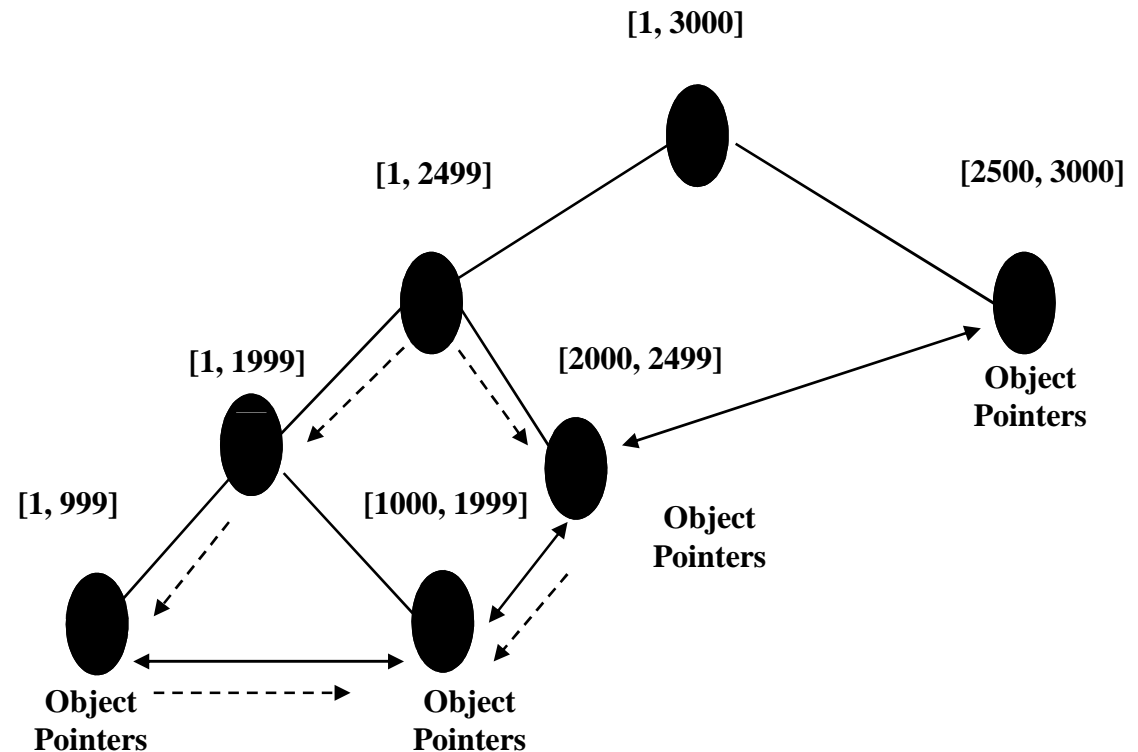
- Query is initiated by user from query originator node.
- The query searches locally in its own node to get any previous cached object set.
- If there is cached object set that match the constraints of the query, the object set is return to user as the result.
- If there is no cached object set, the query is sends to the Chord overlay to get the desired object set.

Range Query

- Adopt lower-upper bound approach.
- Query is splits into two sub-queries, which corresponding to the lower bound of the query and upper bound of the query.
- The lower bound query forwards the query to its left child node.
- The upper bound query forwards the query to its right child node.
- The process continues until both the lower and upper bound queries reach *leaf* node.
- From *leaf* nodes, they send the query towards each other through the sibling nodes and finally meet each other

Range Query Depiction

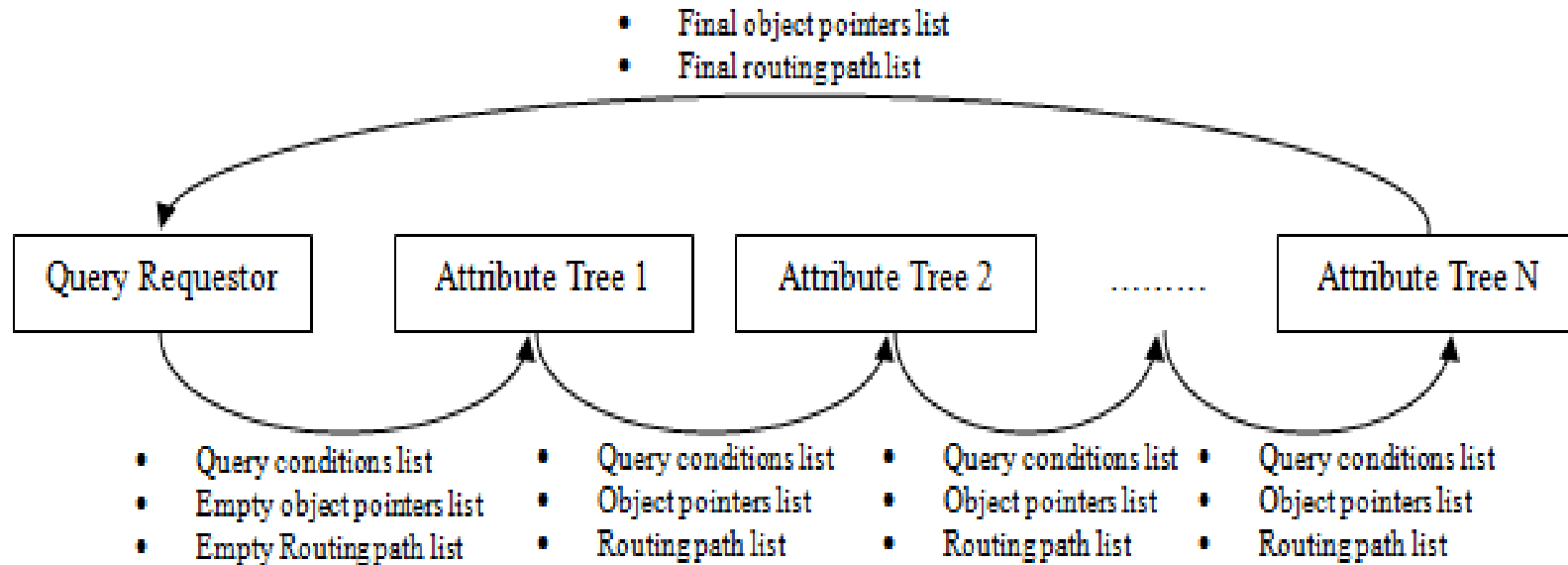
- For example, user wants to search all the car objects ranging from 660 – 2000 CC.
- Query is sent to the node responsible for the range [1,2499] because it is the node that covers the searching range.
- From the node [1,2499], the query is decomposed into lower bound and upper bound queries.
- The lower bound query forwards to the left child node while upper bound query forwards to the right child node.
- Both end queries further forwards down until reaching leaf nodes at both ends.
- Object pointers that fulfill the constraints in query are selected out. Both queries send towards each other until they meet.
- The result from both ends are compiled and get the final desired result.



Multi-Attribute Query

- Decompose query into n sub-query.
- Each sub-query corresponds to one attribute, in which correspond to one attribute tree.
- Proposed a store and forward method.
- Query starts with the first sub-query in first attribute tree.
- Once the query has finish in the first attribute tree, the query is forward to the second attribute tree together with the result from previous attribute tree.
- The process continues until the last attribute tree.

Multi-Attribute Query Depiction



- The query are decomposed into n sub-queries where each sub-query corresponds to a constraint for an attribute.
- The query is forwards to the first attribute tree where it corresponds to the first sub-query.
- After the searching in the first attribute tree has been accomplished, the list of object pointers that fulfill the first constraint of first sub-query and the routing path list (the list of nodes that have been traversed during the query searching) are bring together to the next attribute tree.
- The process continues until the query reaching the last attribute tree.
- The result is compiled as final object pointer list and routing path list and return to the user in query originator.

What has your design achieved?

- The attribute trees are adaptive to the bounds of attribute values.
- Able to cache previous object sets and node information for faster query response time and avoid single point of failure.
- Supporting range and multi-attribute queries.
- Implement the design in a warehouse environment of 30 peers.
- Peers connected in wireless local environment within a domain.
- Also test in simulation, using PeerSim simulation tool.

Conclusion

- Utilize class hub and attribute tree to achieve the multi-attribute and range query search.
- Split and merge operation to make the attribute tree more adaptive to the changing of the attribute value.
- A store and forward method to reduce the workload of the query originator as well as reducing the network traffic.
- Previous object sets are cached in the nodes along the routing path for faster query response time.

References

1. S. Ramabhadran, S. Ratnasamy, J. M. Hellerstein and S. Shenker. *Prefix Hash Tree An Indexing Data Structure over Distributed Hash Tables*.
2. D. Wang and M. Li. *A Range Query Model based on DHT in P2P System*. IEEE 2009.
3. A. R. Bharambe, M. Agrawal and S. Seshan. *Mercury: Supporting Scalable Multi-Attribute Range Queries*. ACM August 2004.
4. C. Sartiani, P. Manghi, G. Ghelli and G. Conforti. *XPeer: A Self-organizing XML P2P Database System*.
5. I. Clarke, O. Sandberg, B. Wiley and T. W. Hong. *Freenet: A Distributed Anonymous Information Storage and Retrieval System*.