Thesis

Multi-Attribute Query Resolution in Structured Peer-to-Peer Networks

Submitted by

Shibayan Chatterjee

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Fall 2016

Abstract

Multi-Attribute Query Resolution in Structured Peer-to-Peer Networks

Collaborative grid and cloud computing applications may be implemented by forming virtual clusters on demand. Formation of such clusters will require diverse resources with specific attributes to execute and support the specific application and its performance requirements. This thesis focuses on formation of such systems by looking up resources over a distributed P2P system. Discovery of appropriate resources from an enormous distributed pool of resources becomes tedious, but it should be resolved with minimum cost and latency. There can be a number of attributes characterizing each resource, and these attributes may have ranges of values depending upon the characteristics of the resources. Thus, when performing a look-up for appropriate resources with a P2P approach, the objective moves from a single-dimensional look-up problem to a multi-dimensional look-up problem. The look-up operation has to be optimized in terms of delay in resolution, hop-count, communication cost, and overhead cost. This thesis develops and evaluates an architecture aimed at resolving this n-dimensional P2P look-up problem. The optimality of the proposed solution is evaluated in terms of communication overhead, delay, and hop count for look up. The complexity of n-dimensional look-up problem is further reduced in terms of dimensionality by utilizing the correlation between different attribute characteristics.

The proposed architecture uses a structured peer-to-peer network in the form of multiple rings, grouped together on the basis of individual attributes thus forming a Ring of Rings (ROR). Chord protocol is used to maintain the scalability and for having communication cost for lookup within logarithmic time complexity. Communication within the network is done using Bloom filters as a data structure, which represents the resources satisfying different attribute values. The novelty of the architecture and the communication methodology lies in the fact that architecture facilitates any number of attributes having any range of values. Furthermore, use of Bloom filters for communication reduces the overhead normally required to carry around long lists of resources to perform the final intersection that resolves the query. Using Bloom filters greatly reduces the communication overhead and the cost of communication. The ROR architecture coupled with Bloom-filter messages reduces the message sizes considerably, but it introduces a certain amount of false positives. The findings indicate that with the optimum number of hash-functions and the optimum sized Bloom-filter, a ROR peer-to-peer architecture to search for multi-attribute queries can be much more efficient than the conventional systems used for the resolution of same type of queries.

Queries associated with many systems request attributes which are correlated to each other. The research also addresses the identification of such resources more efficiently based on correlation of the attributes. The ROR architecture can be tuned for resolving these kinds of queries. The new architectures proposed are localized caching and overlapped ring architecture. The network configuration responsible for resolving the query is the same structured P2P network, where caching and overlapped ring strategies fit in to resolve multiple correlated attributes. These architectures also use Bloom-filter as a data structure to resolve the queries with minimum communication cost and overhead.

Performance of proposed architectures is evaluated using simulations, with the resource traces for simulation generated using the ResQue resource generation tool. The results obtained for a simulation environment consisting of 700 resources, each with 12 different types of attributes, and the number of nodes from 24 to 84 indicates a 30% reduction in communication overhead, 7% reduction in delay in response, and 10% reduction in average load per node. The optimum number of hash functions for the Bloom-filter is 6, and the m/n ratio is kept 10 where m is the size of Bloom-filter and n is the number of resource count. The optimum number of nodes responsible for each attribute is found to be 5. In case of correlated attributes, for the same set of resource count and number of nodes the caching and overlapped ring architecture (using Bloom-filters), provides 58.5% and 57.3% reductions in hop-count, 56.63% and 52.06% reductions in delay in response, message sizes gets reduced by 10% and the average load per node gets reduced by 29% and 34% respectively. The comparison is done over the ROR architecture using Bloom-filters. The number of hash functions for Bloom-filters is considered 4, m/n ratio to be 10 and with 4 nodes per attribute in case of co-related attribute resolution.

Acknowledgements

I would like to thank my advisor, Dr. Anura P. Jayasumana for his valuable guidance and support throughout the course of my thesis. He guided me to understand both the bigger picture as well as the bits and bytes of the problem. I would also like to thank Dr. Chuck Anderson for his valuable remarks, and Dr. Ali Pezeshki and Dr. Sangmi Lee Pallickara for being part of my advisory committee.

I would also like to thank my sister and my mom, and also to that guy who remarked that my work can be used for performing large scale DDoS attack.

Table of Contents

List of Tables

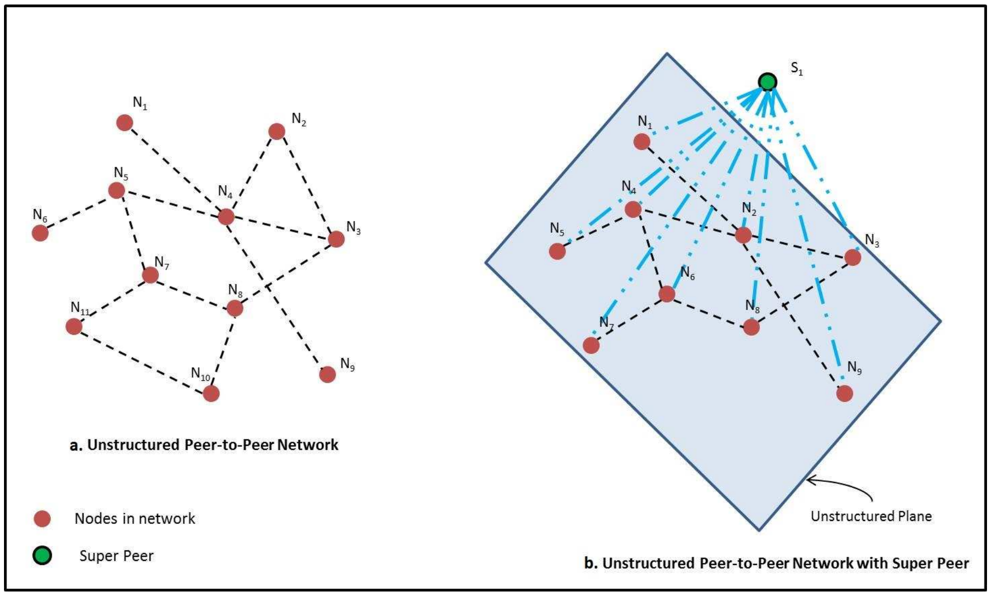
List of Figures

CHAPTER 1

Introduction

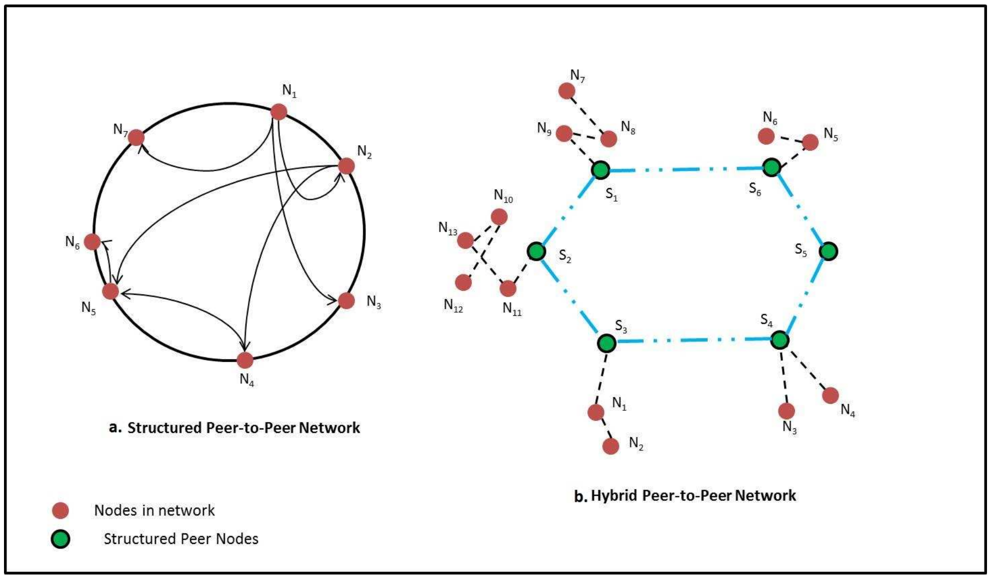
There has been a large number of Peer-to-Peer(P2P) applications over last few years, ranging from video streaming, on-line chatting, video on demand, to file sharing, and data analysis. The P2P traffic has increased considerably as a result of these large scale usage of these applications[1]. A P2P network is a logical overlay network, on top of an already existing physical network. These P2P networks have also given rise to Collaborative P2P Systems, Grid Networks and Cloud Environments, which are meant to provide facility to run numerous platforms, for various applications to provide services to the end user. Each peer in a P2P network corresponds to a node in a P2P network and each of these peers has a similar role and a contribution to the entire P2P network. Each node in a P2P system resides in a host present in the actual physical network, connected through logical links corresponding to a physical path in the network. The physical path is determined by several routing algorithms, and depending upon the optimized path there might be one or more links in a P2P system, only criteria being the physical link exists. The flexibility of having an overlay topology and decentralized control gives the P2P system its distributed characteristics. As a result it can be used in several applications, also since it is distributed it has the capacity to scale up to form a fault tolerant, distributed system. The hosts running these systems can be idle resources for a huge computing task, so load distribution and optimum utilization of resources are achieved.

P2P systems can be majorly classified into two types: *structured P2P* and *unstructured P2P* [2]. In an unstructured P2P the network topology is completely random and arbitrary, no specific pattern is maintained within the system as shown in Figure 1.1.a.



a. Unstructured Peer-to-Peer Network
Nodes in network
Super Peer
Unstructured Plane
b. Unstructured Peer-to-Peer Network with Super Peer

FIGURE 1.1. Unstructured and Super Peer Unstructured P2P Architecture



a. Structured Peer-to-Peer Network
b. Hybrid Peer-to-Peer Network
Nodes in network
Structured Peer Nodes

FIGURE 1.2. Structured and Hybrid P2P Architecture

On the other hand, in structured P2P each of the peers are connected through a regular, conventional topology where the hosts are present in a specific pattern and sequence, as shown in Figure 1.2.a.

To achieve enhanced performance and optimum results within an unstructured network, sometimes there comes the need of a centralized single host within the distributed system leading rise to *super peer unstructured P2P*, shown in Figure 1.1.b. In this type of architecture, the super peer plays only certain roles and has specific controls within the distributed architecture. Multiple structured P2P can also be modified leading to a cluster of structured P2P networks named as *Hybrid Peer to Peer Architecture* where the data distribution and responsibility of each peer gets enhanced to support multiple features at the same time, as shown in Figure 1.2.b.

The applications a which P2P system runs are meant to have zero downtime due to significant redundancy and distributed nature. Also the platforms which support them should also have the capability to be fault tolerant, provide QoS and enhanced performance to the end user. It is for this reason a P2P architecture, best fit in. Aggregation of these platforms and application hosting servers has become necessary in conventional grids, desktop grids, cloud systems, search engines and grid networks to cater to the user demands from time to time. A structured P2P uses Distributed Hash Table (DHT) within its own topology and provides efficient data modification (insertion and deletion) and lookup. These data in the hash table may correspond to items such as books, music files, videos or other resources and each of these resource item in a structured P2P has a unique key which provides access to the resource. A resource can be looked up or be retrieved by looking up the corresponding hash key, hence providing accuracy in resource modification and lookup. The organization of the peers in a structured P2P during topology modification requires lot of efforts to maintain the DHT making it vulnerable showcasing a property called *churn*. In other words *churn* can be defined as the maintenance of property and purpose of a system as a whole irrespective of configuration changes within it. On an unstructured P2P having random topology, resource lookup is possible only through random walks or flooding. In case of robust and scalable P2P systems, it therefore becomes necessary to have resources with better configurations in terms of CPU, Disk Space, and Memory, high transmission and receiving rate, high load sustenance capacity, and low failure rate. While many traditional applications such as music sharing used resources which has necessary configuration for only a single specific type of attribute, many emerging applications require dealing with resources having configurations for multiple attributes. DHT for a particular resource no longer suffices to provide efficient lookup for these resources since it becomes necessary to look up the resource based on one or more of its attributes for different characteristics of the resource. For example to acquire data for atmospheric studies and weather data [3], it is necessary to have computational systems. Weather data has a large size, containing lot of information thus it is necessary to have resource for the former with large memory ($MSize$) and disk space ($DSize$). Also since the information is dynamic the resources should have high load balancing capacity and the information acquired must be processed rapidly. It therefore needs computational systems having large number of cores ($NCore$) and high CPU speed (CPU_{speed}) and it is necessary to characterize these resources on one minute ($1mLd$) loading, five minutes ($5mLd$) loading, and fifteen minutes ($15mLd$) loading, etc. The information acquired about these systems needs to be broadcasted quickly to have current and consistent resource information and to avoid calamities. For example, resource look-up query in the P2P system may ask for addresses of the systems with certain minimum transmission (T_x) and receiving rate (R_x). In totality, it can be said that resources specification and look-up are needed which go beyond a single attribute to multiple attributes.

1.1. Problem Statement

In order to cater to queries related to multi-attribute, it therefore becomes necessary to move from a single dimensional P2P attributes to a n-dimensional or multi-dimensional analysis of these attributes. It becomes essential to have lookup for multi-attributes which needs to be implemented with the same efficiency as that of a single attribute. It is necessary to develop and characterize efficient look up systems which can resolve queries for these multiple attribute resources with minimum latency, low overhead, low communication cost and bare minimum errors in the resolution. The prior solutions address questions such as how to resolve muti-attribute queries, see MAAN [5], SkipMard [6], LORM [16], SCRAP, MURK [17] and $d - Torus$ [18]. There are database methodologies e.g. MongoDB [8], Cassandra [9], HBase [10] which can both store and retrieve data with multi-attributes. MongoDB [8] is an open-source document database that provides high performance, high availability and automatic scaling. MongoDB stores every record as a document which has data structure comprising of field and value pairs. The documents are stored in JSON format. The values for each field may include other documents, arrays and array of documents. Cassandra [9] provides a highly scalable and highly available decentralized database system which supports multiple attributes for data. This database model is fault tolerant (has high replication parameter), and highly elastic (read and write throughput increases linearly as new machines are added) with no interruptions or downtime to the application. Apache HBase [10] is based upon Google's Big Table [11]. It provides linear and modular scalability with random, real time consistent read and write access to Big Data. It also provides automatic and configurable sharding of tables with automatic failover support between region servers.
The methodology used in this thesis targets providing a solution with a system which provides efficient look up within logarithmic time complexity, i.e. the communication cost within the

network is logarithmic in time complexity, for any number of attributes for each resource and for any range of each of the attributes. The architecture comprises of several structured peer-to-peer ($P2P$) architectures in the form of rings which are brought together, forming ring of rings (ROR). The architecture uses Chord [4] to maintain the scalability of the system, for catering to multiple types of attributes, and range of values for each attribute. The proposed approach reduces the communication cost and message overhead by using Bloom filters as data structures but at the same time is able to achieve errors in resolution of queries nearly equal to zero. Also since this system resolves queries for each attribute in parallel and are not interdependent on each other, it provides the facility to list down all the resources that the system is aware of by not restricting it only to a limited number of resources.

Next in analyzing the attributes it has been noted that some of the attributes maintains a particular correlation amongst each other. These types of attributes are considered as correlated attributes since the values of one of the attributes is correlated to one or more other attributes. The queries generated for each attributes are also found to be correlated and hence it becomes necessary to analyze these attributes together. In this way there will be considerable decrease in the hop count, and the latency of response is also reduced. Two methodologies have been discussed in here which looks into the analysis of each of these type of attributes, the first is the caching architecture and the second is the overlapped ring architecture. Each of these architectures are then put in place with the existing ROR architecture and analyzed. The data-structure used in this case also uses Bloom-filter, to act as a data structure for each of the resources and this reduces communication cost and overhead cost for each of the queries.

1.2. Contribution

The methodologies proposed here contributes to providing a multi-attribute P2P system which is scalable, both in terms of the number of attributes, which it is capable of resolving, and the range values of each attribute. The architecture, since it uses $P2P$, is decentralized, so there is no bottle neck in terms of resolving each individual query. Also it uses DHT with *Chord* protocol, providing load balancing, scalability, availability, and flexible naming as its inbuilt features. The lookup, for each of the queries are resolved in logarithmic time complexity in terms of hop count. Large systems resolving these multi-attribute queries will incur high communication cost and high memory overhead. To address this problem Bloom-filters are being used which provides constant message sizes to be carried around, reducing the communication cost and memory overhead considerably. The messages carried around are also provided with a checksum, so the data corruption is also avoided. In order to reduce the latency of response and hop count, the **Caching** and **Overlapped** ring architecture provide an ideal methodology to address the same. Both these later architectures also contribute the same features as those of the main methodology, unlike the arrangements of the rings.

1.3. Outline

This dissertation is arranged as follows. Chapter 2 explains multi-attribute features of resources, the need for multi-attributes. The existing architectures for multi-attribute query are discussed. The proposed multi-attribute P2P methodology along with its architecture and the resolution schemes are discussed in Chapter 3. Next the correlated attributes are introduced along with their characteristics. Two resolution methodologies **Caching** and **Overlapped** are presented in Chapter 4 and Chapter 5 respectively. The query generation and its resolution are discussed in Chapter 6. The novelty of the proposed architectures are also discussed in the same. The conclusion in Chapter 7 provides insight to the necessary improvements and the future work.

CHAPTER 2

Literature Review

This chapter reviews prior research related to our work on Multi-Attribute P2P systems. It starts with a well known P2P lookup architecture, "Chord" and then moves to the other multi-attribute architectures developed and researched over time. Next we review "ResQue", a method to generate multi-attribute resource characteristics for large-scale simulations and finally we review the results that provide insight to the network operation of the Bloom-filters.

2.1. MAAN: A Multi-Attribute Addressable Network for Grid Information Services

Chord[4], a distributed look-up protocol provides efficient solution to locate node that contains a particular key corresponding to a data item. Given a key, it efficiently traces the node responsible for that key. As a result data location can be easily implemented using Chord, by associating each data to a key value, and then storing the data at the node, where the key maps. Also the solution which Chord provides, is robust in terms of changing network configurations i.e. it works efficiently even if nodes join and leave the network. Chord is scalable, with minimum communication cost and the state maintained by each node scales logarithmically with change in the number of nodes present in Chord network.
Although, Chord offers efficient and scalable single-key based registration and lookup service for decentralized resources, it can not support range queries and multi-attribute based lookup. The MAAN[5] approach addresses this problem by extending Chord with locality preserving hashing and a recursive multi-dimensional query resolution mechanism. MAAN uses SHA1 hashing to assign a m bits identifier to each node and the attribute value with

string type. However, for attributes with numerical values MAAN uses locality preserving hashing functions to assign each attribute value an identifier in the m-bit space. Instead of only supporting one attribute based lookup, the MAAN scheme also extends the above routing algorithm for range queries to support multi-attribute lookup. In this multi-attribute setting, we assume each resource has M attributes $(a_1, a_2, ..., a_M)$ and corresponding attribute value pairs $< a_i, v_i >$, where $1 \leq i \leq M$. For each attribute a_i, its attribute value v_i is in the range of $[v_{i,min}, v_{i,max}]$ and conforms to a certain distribution with distribution function $D_i(v)$. Thus, a uniform locality preserving hashing function $H_i(v) = D_i(v) * (2m - 1)$ for each attribute a_i is generated. With these hashing functions all attribute values can be mapped to the same m-bit space in Chord. Chord assumes the distribution function $D_i(v)$ to be uniform, so that it can be mapped to the Chord ring. Correspondingly the $H_i(v)$ function generated as a result also gets uniformly distributed in the ring. Each resource will register its information (attribute value pairs) at node $n_i = successor(H(v_i))$ for each attribute value v_i, where $1 \leq i \leq M$. Resource registration request for attribute value v_i is routed to its successor node using Chord routing algorithm with key identifier $H(v_i)$. Each node categorizes the indices of $< attributevalue, resource - info >$ pairs by different attributes. When a node receives a resource registration request from resource x with attribute value $a_i = v_{ix}$ and resource information r_x, it adds the $< v_{ix}, r_x >$ pair to corresponding list for attribute a_i. When a node searches for interested resources, it composes a multi-attribute range query which is the combination of sub-queries on each attribute dimension, i.e. $v_{il} \leq a_i \leq v_{iu}$ where $1 \leq i \leq M$, v_{il} and v_{iu} are the lower bound and upper bound of the query range respectively. There are two approaches to search candidate resources for multi-attribute range queries: *iterative* and *single attribute* dominated query resolution.

Iterative Query Resolution:

The iterative query resolution scheme is very straightforward. If node n wants to search resources by a query of M sub-queries on different attributes, it iteratively searches all candidate resources for each sub-query on one attribute dimension, and intersects these search results at query originator. The search algorithm proposed is reused for single attribute based lookup in Section 3.1. The only modification is to carry a $< attribute >$ field in each search request to indicate which attribute we are interested in. The search request is as follows: $SEARCHREQUEST(k, a, R, X)$, where a is the name of the attribute we are interested in, and k, R and X are the same as in a single attribute based query. When a node receives a query request and it intersects with the query range, it only searches the index which matches the attribute name in the search request. Though this approach is simple and easy to implement, it is not very efficient. For $M - attribute$ queries, it takes $\mathcal{O}(\sum_{i=1}^{M}((\log N) + K_i))$ routing hops to resolve the queries, where K_i is the number of nodes intersects the query range on attribute a_i. We define selectivity s_i as the ratio of query range width in identifier space to the size of the whole identifier space, i.e. $s_i = \frac{H(v_{iu})H(v_{il})}{2m}$. Suppose attribute values are uniformly distributed on all N nodes, then we have $K_i = s_i * N$ and routing hops would be $\mathcal{O}(\sum_{i=1}^{M}((\log N) + K_i))$. Thus, the routing hops for searching increase linearly with the number of attributes in the query.

Single Attribute Query Resolution:

The Iterative Query Resolution search result of a multi-attribute query must satisfy all the sub-queries on each attribute dimension and it is the intersection set of all resources which satisfies each individual sub-query. Suppose X is the set of resources satisfying all sub-queries, and X_i is the set of resources satisfying the sub-query on attribute a_i, where $1 \leq i \leq M$. So we have $X = \cap X_i$ and each X_i is a superset of X. The iterative query resolution approach computes all X_i using M iterations and calculates their intersection set.

However, since we register the resource information for each attribute dimension, resources in the set of X_i also contain the information of other attribute value pairs. The single attribute dominated query resolution approach can utilize this extra information and only need to compute a set of candidate resources X_k which satisfies the subquery on the attribute a_k. Then it apply the sub-queries for other attributes on these candidate resources and computes the set X which satisfies all sub-queries. Here, we call attribute a_k dominated attribute. There are two possible approaches to apply these sub-queries. One approach is to apply them at the query originator after it receives all candidate resources in X_k. Since the set X_k is typically much larger than X, search requests and responses might contain many candidate resources which do not satisfy other sub-queries. Thus this approach will introduce unnecessarily large search messages and increase communication overhead. Another approach is to carry these sub-queries in the search request, and apply them locally at the nodes which contains candidate resources in X_k. This approach is more efficient because search requests and responses only carry the resources satisfying all sub-queries. The search request in single attribute dominated approach is as following: $SEARCHREQUEST(k, a, R, O, X)$. k, a, R are the same as those in iterative query resolve approach. O is a list of sub-queries for all other attributes except a, and X is a list of discovered resources satisfying all sub-queries. When node n wants to issue a search request with $R = [l, u]$, it first routes the request to node $n_l = successor(H(l))$. The node n_l, searches its local index corresponding to attribute a for the resources with attribute value in the range of $[l, u]$ and with all other attributes satisfying sub-queries in O, and appends them to X. Then it checks whether it is also the successor of $H(u)$. If true, it sends back a search response to node n with the resources in X. Otherwise, it forwards the search request to its immediate successor n_s. n_s repeats this process until the search request reaches node $n_u = successor(H(u))$.Since this approach only need to do one iteration for the dominated attribute a_k, it takes $\mathcal{O}((\log N) + N * S_k)$ routing hops to resolve the query. We can further minimize the routing hops by choosing the attribute with minimum selectivity as the dominated attribute. Thus, the routing hops will be $\mathcal{O}((\log N) + N * S_{min})$, where S_{min} is the minimum selectivity for all attributes. In the single attribute dominated approach, the number of routing hops is independent of the number of attributes, and thus scales perfectly in the number of attributes of a query. On the other hand, it incurs the memory cost of registering all attributes for a resource if any of its attributes is registered; and it incurs more updating overhead of attribute values change. However, the good query performance of the single attribute dominated approach will typically outweigh the greater updating cost in the Grid environment since node registration operations (of $OS - Type, CPU - Speed, Memory - Size, CPU - Count$, etc.) are typically far less frequent than query operations (to find suitable machines).

2.2. SkipMard

SkipMard [6] is based on Skip List-based tools to design a structured P2P network without DHT. For this reason, our primary consideration is locality property. The advantage of DHTs approaches is that it can easily guarantee the load balance, efficiency and scalability on P2P systems. But one of the primary drawbacks is that the use of random-looking hash values of the key generated by hashing functions destroys the P2P natural locality property. The locality property ensures that the query messages are only routed within two location-near nodes. Another important drawback is that regular DHT technique only supports the exact matching. Therefore, multiple exact matching operations have to be repeatedly executed to satisfy a complex range query, and a number of nodes are visited repeatedly in the regular DHT-based P2P overlay network. SkipMard (Multi-attribute resource discovery using Skip List-based tools) is our design of a preliminary P2P data structure, which naturally supports the multi-attribute resource discovery. In this part, we will introduce the data structure of SkipMard, the routing algorithms and the join/leave algorithms. And then we will analyze the runtime of SkipMard.

Architecture:

The design of SkipMard is primarily based on the existing Skip List-based algorithm – Skip Graph. It has been further developed to extend Skip Graph to naturally support multi-attribute queries from one-attribute queries. In SkipMard, we employ a location address space and multiple pre-defined resource-attribute tables. A node in this proposal usually refers to a computer that holds an independent IP address or URL, and hosts multiple resources and attributes. Each node corresponds to an ID in the location address space by the key number. Each attribute belongs to one pre-defined resource-attribute table. All attributes in this node can be classified into different resource categories according to these pre-defined tables, and values of attributes are enumerated to represent a group. For example, a node has 3 resources: CPU, memory and disk. Each resource defines one attribute, speed, size and capacity respectively. The values of each attribute can be defined in Table 1. SkipMard is a Skip Graph-like multi-level data structure defined as a generalization of a Skip List. As in Skip Graph, each node is a member of multiple doubly linked lists. The level 0 contains all nodes in order. However, compared with the data structure of Skip Graph, SkipMard has 3 obvious different properties.

- Instead of the membership vector $m(x)$ in Skip Graph, SkipMard uses a resource vector $rv(x)$. An element in the membership vector is generated a value 0 or 1 by a binary random function and this value itself is meaningless, while each element in a resource vector concretely stands for one resource and the value of an element represents an attribute value. The length of a resource vector is equal to the number of maxlevel, or the high of a SkipMard.

- Skip Graph uses a binary element in its membership vector, this approach has only 2^i linked lists at i level, where i is $[0, \ldots, maxlevel]$. But in SkipMard, we have p^i linked lists at one level, where $p \geq 2$ (such as $p = 3, 4, \ldots$), one of which is called *layer*. If p is equal to 2, then this SkipMard is an instance of Skip Graph.
- Each node in Skip Graph only has left and right neighbor on one level. But in SkipMard, each node on one level has not only the left and right neighbor in the same layer, but also contains the pointers to store the closest neighbor with different layers. We call those neighbors Crossing Nearest Layer Neighbor. Because each node in SkipMard contains more neighbor information than in Skip Graph, the expected space of routing table of SkipMard is $\mathcal{O}(m * \log n)$, where n is total size of nodes and m is total number of layers.

A resource vector of each node can contain multiple attributes of computational resources that we are interested in. Given an application of SkipMard on the internet, we can assume the deployment of computing resources is in a randomized distribution. Thus SkipMard has $\mathcal{O}(\log n)$ expected routing time.

SkipMard provides two routing algorithms and both have $\mathcal{O}(\log n)$ expected time. One is to implement one-attribute query by routing a key. The other is for multi-attribute queries by routing a vector. A prefix matching technique and an approximate closest-point method are used in the routing algorithms. In the algorithm of routing by a resource vector, we briefly introduce a structure parallel flooding strategy for dynamic multi-attribute query.The routing algorithms used in Skip Mard are:

- **Approximate Closest-Point Method:** In routing algorithms for one-attribute, SkipMard employs an approximate closest-point method for a specified searching key that enables each node to always try to route this key to the node with the closest key value. Unlike Skip Graph, which just uses the comparison operations to determine the next node where the requested message is sent, SkipMard uses a minimal distance function to determine the next node which is closer to requested node than the previous one. Skip Graph adopts one side forwarding approach, while SkipMard always goes around the requested node and tries to converge.
- **Routing by Key:** Routing algorithm by a key in SkipMard belongs to one-attribute query. It applies an approximate closest-point method to calculate a minimal distance from all neighbor of the current node to this requested node. This minimal distance determines the next accessing node where a searching message is sent. Like Skip List and Skip Graph, the search starts at the top-most level of any node and then goes down from top-level to the bottom level. The expected time of routing by a key is $\mathcal{O}(\log n)$.
- **Routing by resource Vector:** Multi-attribute resources usually include static attributes and dynamic attributes. To discover dynamic attributes is more difficult than static attributes because of their intermittence and randomization. Most current resource discovery projects use the flooding technique to solve dynamic resource discovery. In SkipMard, we also propose a parallel structured flooding technique in routing algorithm of searching for a vector to discover dynamic attributes. A Time-To-Live (TTL) mechanism is used in flooding to guarantee the efficiency and termination of queries. In this routing algorithm, a user-defined dynamic resource requirement is saved to a data pointer, and a TTL_{num} is specified as a maximum

number of resources that satisfy the requirements or the step length. An internal function Dynamic provides the comparison between user-defined dynamic requirements and a local dynamic table, which is maintained by a node itself. If a node can satisfy the dynamic requirement, then TTL_{num} will decrease by one, which represents a successful finding. If TTL_{num} is still greater than zero, then a parallel structured flooding approach will be applied. The query message will be sent to the neighbors of the two side of the current node in parallel until TTL_{num} decreases to 0. A dynamic multi-attributes query will return multiple satisfied results and the maximum number will be less than the $2 * TTL_{num}$. The expected time of searching for a vector is $\mathcal{O}(\log n)$ and the message is $\mathcal{O}(\log n) + \mathcal{O}(k)$, where k is a step length of TTL_{num} in the parallel structured flooding.

2.3. ResQue

In *ResQue* [7], a novel mechanism is being adopted to generate random nodes having both static and dynamic attributes that are useful in evaluating the performance of large-scale P2P resource discovery schemes and job schedules. The presented methodology is applicable to any multivariate resource dataset, and PlanetLab node traces are utilized as an example.

- A multi-attribute resource model is defined using a selected set of static and dynamic attributes that are essential to characterize a node.
- The characteristics of nodes are presented. The findings show that attribute values are skewed, follow a mixture of probability distributions, complex correlation patterns among attributes, and time series of dynamic attributes are non-stationary. These characteristics make it nontrivial to generate random nodes with multiple attributes.

- The vectors of static attributes are generated using empirical copulas that capture the entire dependence structure of multivariate distribution of attributes.
- The time series of dynamic attributes are randomly drawn from a library of multivariate-time-series segments extracted from PlanetLab traces. These segments are determined by identifying the changes in the regression coefficients of time series corresponding to a selected attribute. Time series corresponding to rest of the attributes are split at the same breakpoints and randomly drawn together to preserve their contemporaneous correlation.
- A tool is developed to automate the synthetic data generation process and its output is validated using statistical tests. The tool generates n random nodes with as static and ad dynamic attributes. Dynamic attribute values can be generated up to a given time t (ranging from several hours to weeks) with sampling interval s.

Node-Model and Characteristics

Following nine attributes are selected to describe a node:

(1) CPUSpeed (CSp): Processor clock speed in GHz. Provides insight on relative computing power of a node.

(2) NumCores ($NCore$): Number of processor cores. Indicates how much parallelism in processing is possible.

(3) CPUFree ($CFree$): (100 CPU utilization) Indicates to what extent the CPU(s) is available for processing. If multiple cores are available, the average value is given.

(4) 1MinLoad ($1mLd$): One minute exponentially weighted moving average of number of active processes competing or waiting for CPU. Indicates how long a user process has to wait. Both CPUFree and 1MinLoad are complementary to each other as

a large CPU load does not necessarily mean high CPU utilization (e.g., processes could be blocked for I/O). Similarly, 5MinLoad ($5mLd$), and 15MinLoad ($15mLd$)

(5) MemSize ($MSize$): Size of volatile memory in GB.

(6) MemFree ($MFree$): Free user-level memory as a percentage. Indicates how much memory is available for user processes.

(7) DiskFree ($DFree$): Free disk space in GB.

(8) TxRate (T_X): Average transmission rate in bps. In conjunction with bandwidth limit specified by most nodes, it provides insight on amount of available bandwidth.

(9) RxRate (R_X): Average receive rate in bps.

CPUSpeed, NumCores, and MemSize are static attributes (number of static attributes $a_s = 3$) while the rest are dynamic (number of dynamic attributes $a_d = 6$). The analysis of these attributes are based on the methodology which is applicable to other systems, e.g., SETI@home [24] as well. Resource discovery solutions and scheduling algorithms for latency sensitive applications are typically interested only in short-term trends. Therefore, we capture statistical characteristics that are valid for several minutes to few weeks.

Random Vector of Static Attributes:

Some of the attributes hold strong correlation among some of the other attributes as well as with their specific structures in time series, attribute values of random nodes cannot be drawn from independent distributions. Therefore, we have to rely on the joint distribution of attributes. Static and dynamic attributes are handled separately as the time series of dynamic attributes are nonstationary and also have specific temporal structures. As the correlation among attributes is nonlinear and complex, it is insufficient to use the matrix of Pearsons correlation coefficients to establish the dependence among random variables. Alternatively, copulas [26] can be used to capture the entire dependence structure of multivariate distributions. Copulas are functions that couple multivariate distribution functions to their marginal distributions. A copula $C(u)$ is a multivariate joint distribution defined on the $d-$dimensional unit cube $[0,1]d$, $(u_1,\ldots,u_d) \in [0,1]d$, such that every marginal distribution u_i is uniform on the interval $[0,1]$. Let F denote the $d-$dimensional distribution function (CDF) with marginals $F_1,\ldots,F_d$ then a copula C exists such that for all real $u = (u_1,\ldots,u_d)$:

$$F(u) = C(F_1(u_1),\ldots,F_d(u_d)) \tag{2.1}$$

Several well-known copula families are available, e.g., Gaussian and Archimedean copulas. However, these copulas tend to be symmetric along the axis of correlation. Alternatively, empirical copulas are useful while analyzing data with complex and/or unknown underlying distributions. Empirical copula also supports any number of dimensions, and its bivariate function is given by[7]:

$$C_n(\frac{i}{n},\frac{j}{n}) = \frac{No.of pairs(x,y)s.t.x \leq x_{(i)} and y \leq y_{(i)}}{n} \tag{2.2}$$

where $1 \leq i, j \leq n$, $x_{(i)}$ is the ordered statistics of x, and n is the number of data points. It is proven that empirical copula converges uniformly to the underlying copula. After deriving the copula, dependent random numbers can be generated. The density distribution of the attributes CSp, $MFree$ and T_X is as shown in Figure 2.1 [7]. Those numbers can then be transformed into original marginal distributions using inverse transforms. The distribution of some of the attributes CSp, $MFree$ and T_X is as shown in Figure 2.2 [7].

Tool for Random Node Generation:

A tool has been built to automate the synthetic data generation process, by combining

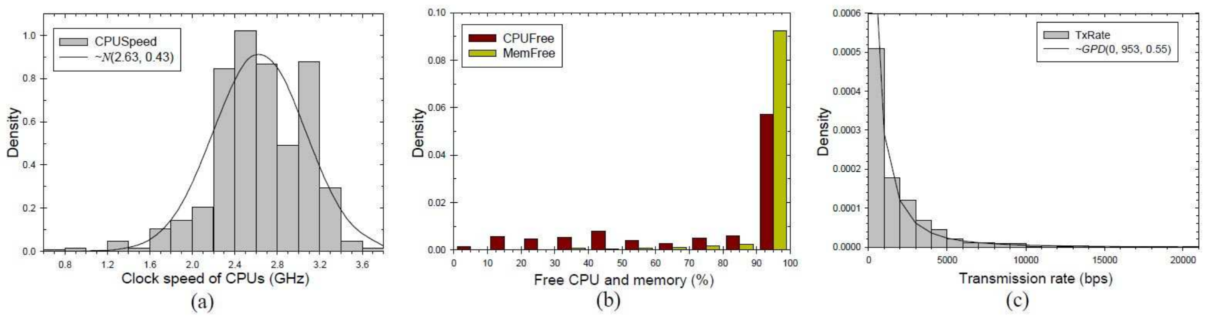


FIGURE 2.1. Density distribution for CSp, CPUFree, MFree, and T_X

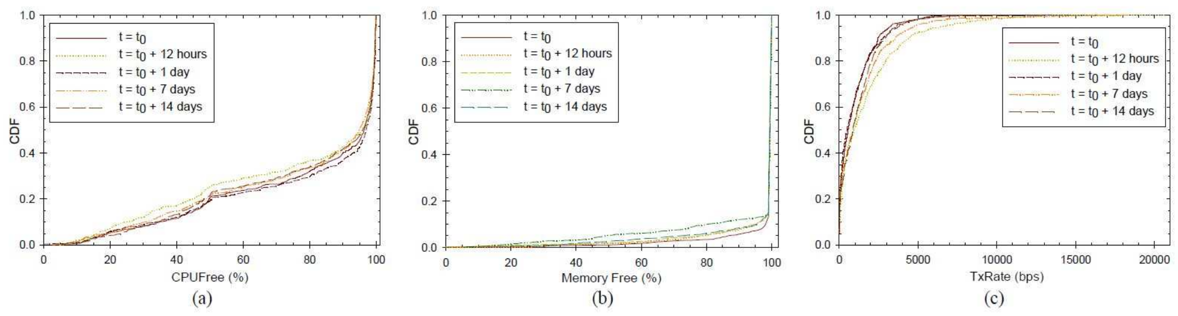


FIGURE 2.2. Marginal distribution values with time for CSp, $MFree$ and T_X

the empirical-copula-based static attribute generation and time-series-library-based dynamic attribute generation. Figure 2.3 [7] is a flowchart illustrating the technique.

It can generate synthetic traces corresponding to a set of nodes, e.g., n random nodes with as static and dynamic attributes. As the distribution of dynamic attributes is stable over few weeks, the technique can be used to generate data from several minutes to few weeks. The change of values of attributes within a span of day is as shown in Figure 2.4 [7].

Instantaneous values of dynamic attributes are also fed to the copula generator [26] to generate random vectors with instantaneous dynamic attributes that may be useful in evaluating scheduling algorithms. $NCores$ from copula is fed to draw random samples module to establish the dependence between static and dynamic attributes. If desired, a user may use only a subset of the attributes supported by the tool. Several additional attributes (e.g., $5mLd$, $DSize$, and Location) are included in the MATLAB-based tool that

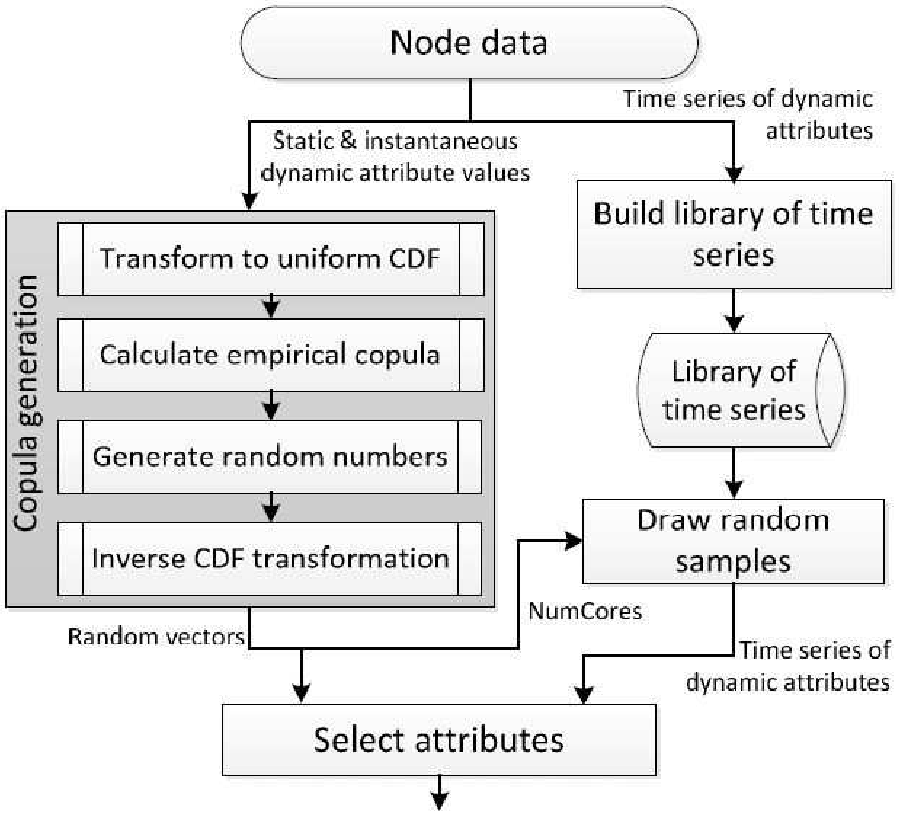

Node data
Time series of dynamic attributes
Static & instantaneous dynamic attribute values
Copula generation
Transform to uniform CDF
Calculate empirical copula
Generate random numbers
Inverse CDF transformation
Build library of time series
Library of time series
Draw random samples
NumCores
Random vectors
Time series of dynamic attributes
Select attributes

FIGURE 2.3. Flow-chart of random node generation tool

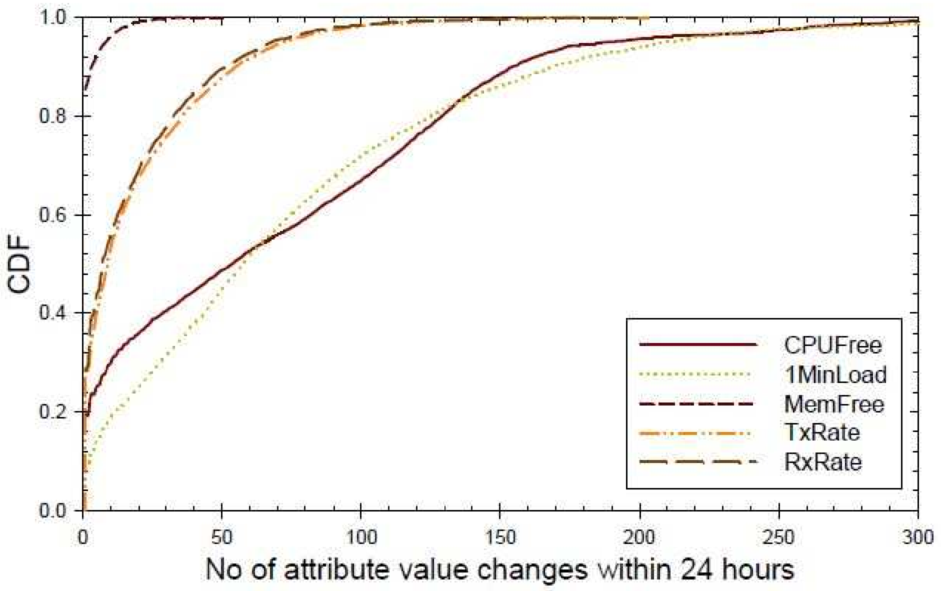


FIGURE 2.4. Change of attribute values with time

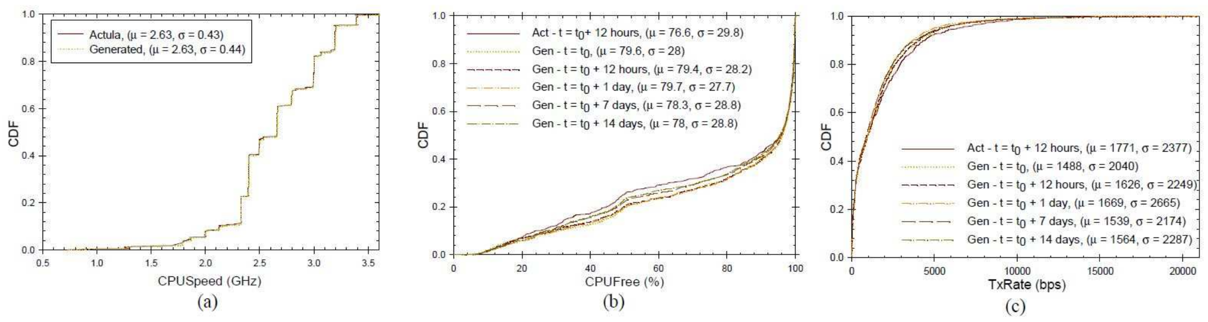


FIGURE 2.5. Comparison of actual nodes parameters with nodes parameters generated by tool

is downloadable from [25]. A comparison is made with synthetic data with actual data as there are no other comparable models that capture the correlation among dynamic attributes.

Figure 2.5 plots the distribution of both the actual and generated attributes for CSp, $CFree$ and T_X. It can be seen that the generated attributes closely match the distributions observed in systems operating under real time environments.

2.4. Network Application of Bloom Filters

A Bloom filter is a simple space-efficient randomized data structure for representing a set in order to support membership queries. Bloom filters allow false positives but the space savings often outweigh this drawback when the probability of an error is made sufficiently low. Burton Bloom introduced Bloom filters in the 1970s [13], and ever since they have been very popular in database applications. Recently they started receiving more widespread attention in the networking literature.

In this paper [14], a survey was made on the several ways in which Bloom filters have been used and modified for a variety of network problems, with the aim of providing a unified mathematical and practical framework for them and stimulating their use in future applications. First the mathematics behind Bloom filters is described, their history, and some important variations. Then the consideration of four types of network-related applications of Bloom filters are discussed:

- Collaborating in overlay and peer-to-peer networks: Bloom filters can be used for summarizing content to aid collaborations in overlay and peer-to-peer networks.
- Resource routing: Bloom filters allow probabilistic algorithms for locating resources.
- Packet routing: Bloom filters provide a means to speed up or simplify packet routing protocols.
- Measurement: Bloom filters provide a useful tool for measurement infrastructures used to create data summaries in routers or other network devices.

Emphasize on this simple categorization is very loose; some applications fit into more than one of these categories, and these categories are not meant to be exhaustive. Indeed, suspection was done that new applications of Bloom filters and their variants will continue to bloom in the network literature. Also, emphasize are being provided that only brief summaries of the work of many others. The theme unifying these diverse applications is that a Bloom filter offers a succinct way to represent a set or a list of items. There are many places in a network where one might like to keep or send a list, but a complete list requires too much space. A Bloom filter offers a representation that can dramatically reduce space, at the cost of introducing false positives. If false positives do not cause significant problems, the Bloom filter may provide improved performance. This property of Bloom filter principle, is given emphasis

The Bloom filter principle: Wherever a list or set is used, and space is at a premium, consider using a Bloom filter if the effect of false positives can be mitigated.

Mathematical Principles: We begin by presenting the mathematics behind Bloom filters. A Bloom filter for representing a set $S = x_1, x_2, ..., x_n$ of n elements is described by an array of m bits, initially all set to 0. A Bloom filter uses k independent hash functions $h_1, \ldots, h_k$ with range $1, \ldots, m$. For mathematical convenience, we make the natural assumption that these hash functions map each item in the universe to a random number uniform over the range $1, \ldots, m$. For each element $x \in S$, the bits $h_i(x)$ are set to 1 for $1 \leq i \leq k$. A location can be set to 1 multiple times, but only the first change has an effect. To check if an item y is in S, we check whether all $h_i(y)$ are set to 1. If not, then clearly y is not a member of S. If all $h_i(y)$ are set to 1, we assume that y is in S, although we are wrong with some probability. Hence, a Bloom filter may yield a false positive, where it suggests that an element x is in S even though it is not. Figure 1 provides an example. For many applications, false positives may be acceptable as long as their probability is sufficiently small. To avoid trivialities we will silently assume from now on that $kn < m$.

The probability of a false positive for an element not in the set, or the false positive rate, can be estimated in a straightforward fashion, given our assumption that hash functions are perfectly random. After all the elements of S are hashed into the Bloom filter, the probability that a specific bit is still 0 is[13]

$$p = (1 - \frac{1}{m})^{kn} = \mathrm{e}^{(-\frac{kn}{m})} \tag{2.3}$$

We let $p = \mathrm{e}^{(-\frac{kn}{m})}$, and note that p is a convenient and very close (within $\mathcal{O}(1/m)$) approximation for p. Now, let ρ be the proportion of 0 bits after all the n elements are inserted in the table. The expected value for ρ is of course $E(\rho) = p$. Conditioned on ρ, the probability of a false positive is[13]

$$(1 - \rho) * k = (1 - p) * k \tag{2.4}$$

and this is the second approximation. The first one is justified by the fact that with high probability ρ is very close to its mean. In general, it is often easier to use the asymptotic approximations p and f in analysis, rather than p and f. It is worth noting that sometimes Bloom filters are described slightly differently as, instead of having one array of size m shared among all the hash functions, each hash function has a range of $\frac{m}{k}$ consecutive bit locations disjoint from all the others. The total number of bits is still m, but the bits are divided equally among the k hash functions. Repeating the analysis above, it is found that in this case the probability that a specific bit is 0 is[13]

$$(1-\frac{k}{m})^n = \mathrm{e}^{-\frac{kn}{m}} \tag{2.5}$$

Asymptotically, then, the performance is the same as the original scheme. However, since for $k \geq 1$[13]

$$(1-\frac{k}{m})^n \leq (1-\frac{1}{m})^{kn} \tag{2.6}$$

(with equality only when $k = 1$), the probability of a false positive is actually always at least as large with this division. Since the difference is small, this approach may still be useful for implementation reasons; for example, dividing the bits among the hash functions may make parallelization of array accesses easier.

Applications: P2P/Overlay Networks

An early peer-to-peer application of Bloom filters is due to Marais and Bharat [19] in the context of a desktop web browsing assistant called Vistabar. Cooperative users of Vistabar store annotations and comments about the web pages that they visited in a central repository. Conversely, they see these comments whenever they load an annotated page. Rather than make a request for each URL encountered, Vistabar downloads periodically a Bloom filter corresponding to all annotated URLs.

- **Approximate Set Reconciliation for Content Delivery**

  Byers, Considine, Mitzenmacher, and Rost [20] demonstrate another area where Bloom filters can be useful in peer-to-peer applications. They suggest that peers may want to solve the following type of approximate set reconciliation problems. Suppose peer A has a set of items S_A, and peer B has a set of items S_B. Peer B would like to send peer A a succinct data structure so that A can start sending B items that B does not have, that is, items in $S_A - S_B$. One approach is to have B send A a Bloom filter; A then runs through its elements, checking each one against the Bloom filter, and sending any element that does not lie in S_B according to the filter. Because of false positives, not all elements in $S_A - S_B$ will be sent, but most will. The authors also consider an alternative data structure that uses Bloom filters, but allows for faster determination of elements in $S_A - S_B$ when the difference is small [21]. This work demonstrates that Bloom filters can also be useful as subroutines inside of more complex data structures. The application [20] addresses the distribution of large files to many peers in overlay networks. The authors argue for encoded content. In this setting, peers may wish to collaborate during downloads, by receiving encoded packets from other peers as well as from the source, thus effectively increasing the download rate. If the encoded content is over a large alphabet, the problem of determining which encoded packets peer B needs that peer A has is simply the problem of determining $S_A - S_B$. Since the content is redundantly encoded, obtaining a large fraction of $S_A - S_B$ rather than the entire set is sufficient in this situation.

- **Set Intersection for Keyword Searches**

  Reynolds and Vahdat use Bloom filters in a similar fashion as [20], except that their goal is to find the set intersection instead of the set difference [22]. Their approach is essentially the same as for database joins. Peer B can send a Bloom filter representing S_B to A; peer A then sends the elements of S_A that appear to be in S_B according to the filter. False positives yield elements of S_A that are in fact not in S_B, but, if desired, B can then determine these elements to find $S_A \cap S_B$ exactly. The Bloom filter approach allows $S_A \cap S_B$ to be determined with fewer bits transmitted than A sending the entire set S_A. Reynolds and Vahdat describe how using this approach for set intersection allows for efficient distributed inverted keyword indices for keyword search in an overlay network over a peer-to-peer architecture. When a document is published, the author also selects a set of keywords for the document. Each node in the network is responsible for a set of keywords in the inverted index; hashes of the keyword determine the responsible nodes. To handle conjunctive queries involving multiple nodes, the set intersection methods above are used to reduce the amount of information that needs to be sent to determine the appropriate documents.

CHAPTER 3

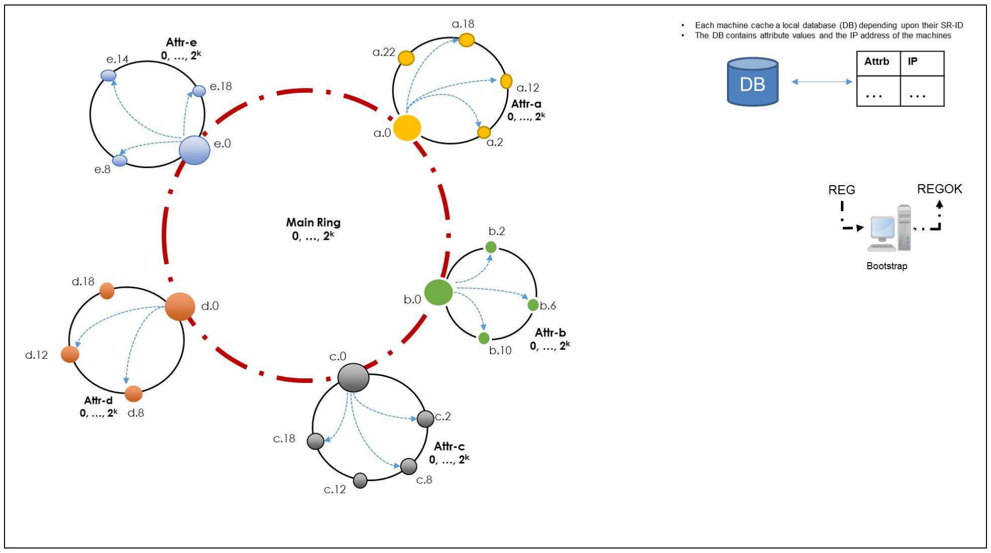
Multi-Attribute query resolution

To look for resources having multiple attributes, with each attribute having a range of values, it is necessary to have a system which can fit in multiple types of attributes and also fit in a range of values corresponding to each attribute. Also the system needs to scale up both in terms of the number of attributes and in range of values for each attribute. The architecture proposed here is composed of a structured P2P architecture arranged in the form of rings corresponding to different attributes. As a whole, it comprises of multiple rings grouped together to form a ring of rings (ROR). This ROR architecture addresses both the requirement of any number of attributes as well as any range values for each individual attribute.

3.1. System Architecture

An architecture is proposed here comprises of a set of nodes present in a Main Ring (MR), and sets of nodes present in several Sub-Rings (SR) as shown in Figure 3.1. Each node present in the MR is responsible for a different type of attribute from those covered by the other nodes in MR and a node in SR is responsible for a range value for the attribute that the SR is responsible for. The MR nodes are aware of all the other nodes present in MR, and the attributes they are responsible for. Each MR node also contribute in the range value of attributes by being a part of one SR as well. The messages exchanged, to and fro within the network are Bloom-filter messages. Here we discuss how the ROR architecture comprising of MR and SR, together with Bloom-filter based methodology resolve multi-attribute queries.

The architecture uses “Chord” protocol [4] for look up and routing within each SR. Chord, provides efficient lookup and routing, since it uses $\mathcal{O}(n \log n)$ number of hops for


- Each machine cache a local database (DB) depending upon their SR-ID
- The DB contains attribute values and the IP address of the machines

Attrb	IP
…	…

FIGURE 3.1. ROR Architecture

efficient routing. Also it acts as a distributed hash table spreading keys evenly over the nodes present in each SR, providing in-built features of a load balancer. Also since Chord is fully distributed, every nodes provides equal contribution towards query resolution and hence provides robustness in the architecture. Also the cost of scaling up is logarithmic in terms of the number of nodes so even large systems, necessary for large number of attributes and large range of values for lookup are feasible without any parameter tuning or constraints in lookup for the keys and hence providing flexible naming. Chord assigns keys by consistent hashing and the routing cost is very less, since every node maintains information only about $\mathcal{O}(n \log n)$ of other nodes and requires $\mathcal{O}(n \log n)$ messages for look up [4]. Also for the SRs for the range of attributes, the amount of ring each node "owns" is determined by the distance to its immediate predecessor. Since the range value of each attribute, is uniformly distributed over the ring, the distribution is tightly approximated by an exponential distribution with mean $2m/N$, where m is the range and N is the number of nodes in the SR [4].

Each node in the MR maintains a hash table (HT_{MR}) having a ¡key, value¿ pair. The "key" in HT_{MR} represents each attribute and the corresponding "value" represents the node

MR id. This kind of hash table has direct addressing feature. The nodes in MR comes up and the notifies the preexisting nodes by exchanging their id and the attribute they are responsible for. Each node getting such a notification updates its HT_{MR} as well. The nodes coming up first become a part of the MR until all the different attributes gets one node to be responsible for it in the MR, and the later nodes coming up become part of SR for each of the different attributes. The nodes in SR while joining a specific SR, knows the MR node for that attribute, exchange keys and arrange themselves in a structured manner forming a distributed architecture as in Chord network. The nodes in the MR maintain both HT_{MR} as well as Finger Table (FT) based on their SR ids, whereas the SR nodes maintain only a FT. The number of the nodes in the MR depends upon the maximum number of attributes possible, which the system will be responsible for, resolving queries related to any of those attributes. The SRs should have atleast one node (apart from the node contributing in MR) to divide the key space of the entire attribute range. Each of the SRs has a bitspace allocated $[0; ...; (2^k - 1)]$ where $k \epsilon \mathbb{Z}^+$. A node in SR calculates a multiplying factor (MF) as mentioned in Equation 3.1 to map out the range of the attribute it will be responsible for in the entire bitspace. This hash table, HT_{MR} is used by the MR nodes to route queries from one node to another. HT_{MR} is mentioned as routing table (RT) for the MR nodes.

$$MF = \frac{r_n}{2^k - 1}, where \quad r_n = [u_n - l_n] \tag{3.1}$$

The nodes in SR share the $< key, value >< k, v >$, where k is the IP address of the resource and v is the value of the attribute for that resource, from their respective MR nodes. So for any node having id k_2, responsible for range $[k_1; k_2]$ will be responsible for attributes having range of values $[k_1 * MF; k_2 * MF]$. Here k_1 and k_2 represents the lower limit and upper limit respectively of the bitspace that the node with id k_2 is responsible for. The nodes in MR and SR caches these contents locally w.r.t their SR id.

3.2. Query Propagation

A multi-attribute query which undergoes resolution is represented as follows:

$Q \equiv (a_{1q} = [l_{1q}, u_{1q}]; a_{2q} = [l_{2q}, u_{2q}]; ...; a_{rq} = [l_{rq}, u_{rq}])$

where Q is the query; $(a_{1q}, a_{2q}, ..., a_{rq}) \epsilon A$, and $[l_{1q}, l_{2q}, ..., l_{rq}]$ are the lower limits and $[u_{1q}, u_{2q}, ..., u_{rq}]$ are the upper limits for different attributes. The user generated query Q arrives at a random node present in the network. Let's consider a query arriving a node Q_{node}. The Q_{node} looks into the first attribute in Q, and if it finds that Q has an attribute which it is responsible for, it parses out Q, retrieves the upper and lower limits of its attribute and send out the rest of the query to the node responsible for the next immediate attribute in Q. If it does not find its attribute in the query, it looks into the first attribute present in Q, and looks for the node responsible for it from its RT. It then forwards Q, to the node responsible for it in MR. The node which gets the Q checks whether it has got query for the attribute it is responsible for, if it is so it parses out Q, retrieves the upper limit and lower limit of its attribute and send out the rest of the query to the node responsible for the next immediate attribute in Q, which it again comes to know from RT. The next node then parses out its own attribute with the upper limit and the lower limit and sends out the remaining query. This process continues until the query reaches the last node with its own attribute only along with the range and has nothing to forward. The query propagation is shown in Figure 3.2.

The node in MR checks the range of the query attribute $[l_q, u_q]$, and converts it to a range $[\frac{l_q}{MF}, \frac{u_q}{MF}]$ to find out the node id in the SR which is responsible for this range. It

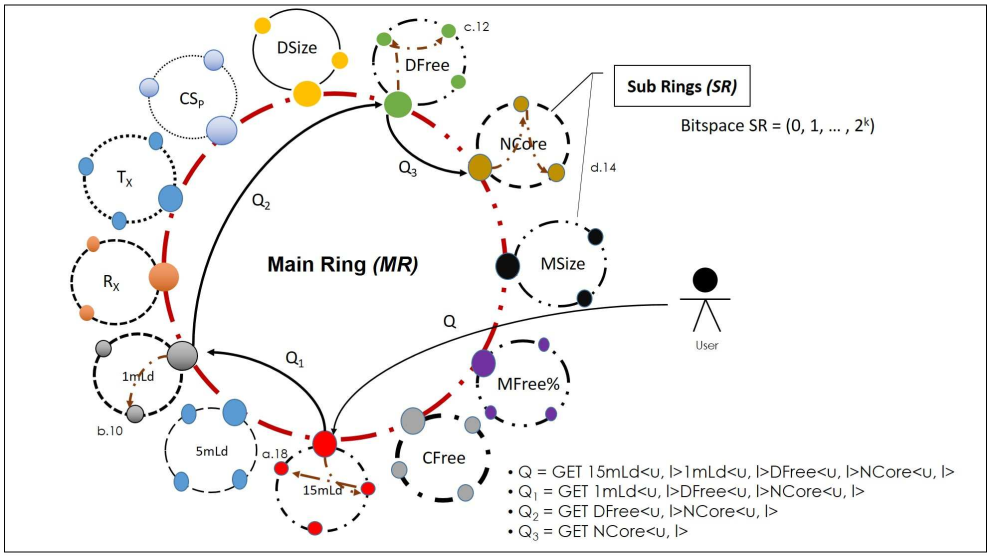

Main Ring (MR)
Sub Rings (SR)
Bitspace SR = (0, 1, … , 2^k)
DSize
DFree
c.12
CS_P
NCore
d.14
T_X
Q_2
Q_3
MSize
R_X
Q
User
Q_1
1mLd
b.10
5mLd
a.18
15mLd
MFree%
CFree
• Q = GET 15mLd<u, l>1mLd<u, l>DFree<u, l>NCore<u, l>
• Q_1 = GET 1mLd<u, l>DFree<u, l>NCore<u, l>
• Q_2 = GET DFree<u, l>NCore<u, l>
• Q_3 = GET NCore<u, l>

FIGURE 3.2. Query Propagation in ROR Architecture

refers its FT and finds out which is the node in the SR that is responsible for the key-value pair. In the process if it finds that the node responsible for the range of the attribute is itself, it resolves it locally.

3.3. Bloom-Filter Creation

The node after receiving the query finds out the list of nodes which falls within the range of $[l_{rq}, u_{rq}]$ and makes a normal Bloom-filter (nBF) as shown in Figure 3.3.

The parameters necessary for the creation of nBF are size of nBF (m), number of Hash-Functions (k), and the total number of resources (N) that needs to be represented by the Bloom-filter. Since for different ranges of attributes there will be different sizes of N, m is kept constant and the optimal number of k, is calculated from Equation 3.2.

$$k = \frac{m}{N} * \ln 2 \tag{3.2}$$

The hash functions required to be chosen for hashing up the IP address must be unique enough, so that the hash values generated must have minimum overlap [12]. These hash

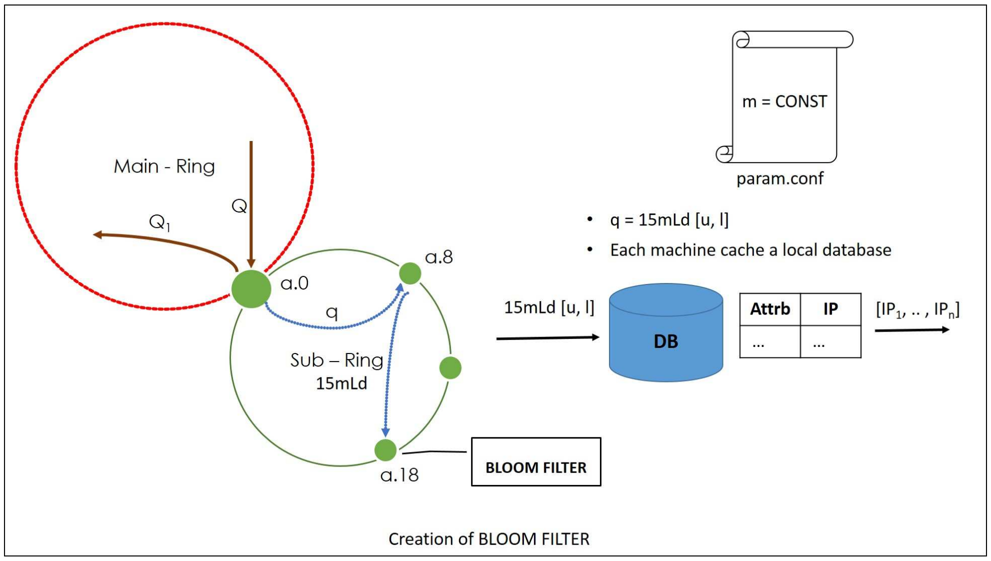


Figure 3.3. Bloom-Filter Creation at each node

values represent the indices of the Bloom-filter which are set to 1 if it is 0 initially or is left unchanged. A nBF data structure doesn't have any false-negative associated with it, but it a has certain amount of false-positive probability (f_p) associated with it [13]. For optimal conditions, and accurate results the value of f_p should be as much minimum as possible. For minimum f_p, the value of k and false positive probability p is as mentioned below in Equation 3.3:

$$\ln p = -\frac{m}{N} * (\ln 2)^2 \tag{3.3}$$

The insert operation in nBF is as described in Algorithm 1 which is shown in Figure 3.4. The notations used for defining the algorithms follow in Table 3.1.

Each of the IP addresses is hashed k number of times, every time with a unique hash function from the set $H()$. Then its modulus w.r.t m is calculated. The modulus represent indices of Bloom-filter, which are then used to populate the nBF by setting the respective indices to 1 if it is 0 initially, or else it is left unchanged.

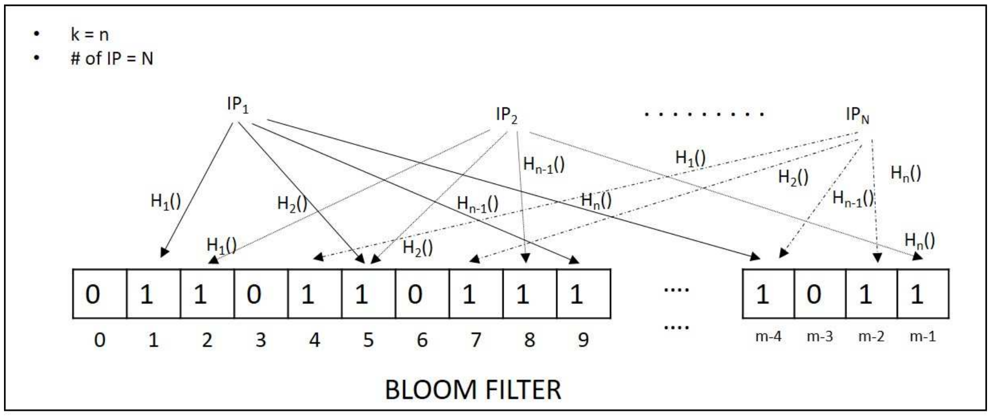


FIGURE 3.4. Insert operation in Bloom Filter

TABLE 3.1. Notations

Notation	Meaning
k	Number of hash functions
m	Size of Bloom-filter
N	Num. of elements in the set
nBF	Normal Bloom-Filter
rBF	Resultant Bloom-Filter
$H()$	Hash functions $[H_1(), H_2(), ..., H_n()]$
$hvals$	Hash values obtained
$vAttrb$	Attribute value $\epsilon\mathbb{Z}^+$
l_{rq}	Lower limit of resources
u_{rq}	Upper limit of resources
ip	IP address
Q_{attr}	Query attributes
Q_{ip}	Query IPs
IP_{final}	Final list of IPs to be sent to user

3.4. Bitwise-AND and Resultant Bloom-Filter creation

The nodes in SR send the nBFs to the Q_{node}, with a check-sum to prevent data corruption. In most network applications, the Bloom-filters are carried around and since the size of a Bloom-filter is always kept constant, it can be used to represent very large datasets into a fixed sized data structure [14]. Using a Bloom-filter can therefore prevent carrying around the long list of IP addresses. This methodology is extremely beneficial in case of system which are highly scalable as it will incur less communication cost and memory overhead.

Algorithm 1 Insert in Bloom-Filter

```
 1: $m \leftarrow CONST$
 2: $IP = []$ initialized as a blank list
 3: for Each $vAttrb$ in $Attr_r$ do
 4:     if $(vAttrb > l_{rq})$ then
 5:         if $(vAttrb < u_{rq})$ then
 6:             $IP.append$(the selected $ip$)
 7:         end if
 8:     end if
 9: end for
10: $N \leftarrow len(IP)$
11: $k \leftarrow$ from equation equ(a)
12: for Each $ip$ in $IP$ do
13:     $hvals \leftarrow k$ hash values for $ip$ obtained from $H()$
14:     for Each hash value $(h)$ in $hvals$ do
15:         $i \leftarrow h \mod m$
16:         $nBF[i] \leftarrow 1$
17:     end for
18: end for
19: Send $nBF$ to $Q_{node}$
```

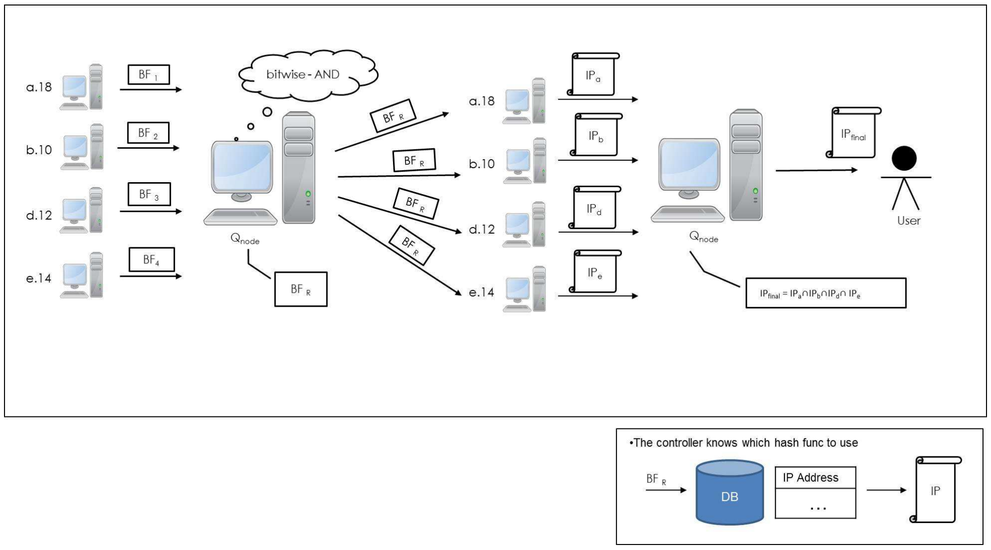


FIGURE 3.5. Bit-wise AND operation and sending back resultant Bloom Filter to nodes

The Q_{node} does a bit-wise AND operation on all the nBFs received (one for each individual attribute) and makes the resultant Bloom-filter rBF. It then sends back the rBF to the respective nodes (in SR), as shown in Figure 3.5. The algorithm for calculating the rBF is as described in Algorithm 2.

Algorithm 2 Calculate resultant Bloom-Filter

```
1: for Each attribute in $Q_{attr}$ do
2:     if $nBF$ received for the first time then
3:         $rBF \leftarrow nBF$
4:     else
5:         $rBF \leftarrow$ bitwise-AND of $(rBF, nBF)$
6:     end if
7: end for
8: Send $rBF$ back to the nodes for individual attribute
```

The bit-wise AND operation of the Bloom-filters creates the intersection operation of the list of IP addresses and the resultant Bloom-filter represents the final list of IP addresses which stands as query resolution. This operation prevents the checking of each individual IP address from the long list of IP address for each individual attribute [15]. The algorithmic time complexity for this methodology is $\mathcal{O}(M_{ip} * N_{attr})$, where M_{ip} is the size of each IP list and N_{attr} is the number of attributes queried for. In case of large scale systems, magnitudes of both M_{ip} and N_{attr} will be very high. Using the conventional system to resolve queries for these sorts of systems will therefore incur much time delay in response. On the other hand, in case of ROR using Bloom-filter messages, M_{ip} will be constant (only corresponding to the size of Bloom-filter). Also the magnitude of Bloom-filter size is much less than M_{ip}, in case of former. Since, out of the two parameters M_{ip} and N_{attr}, M_{ip} is the dominant parameter, reducing its magnitude greatly reduces the time complexity and hence the time of operation for multi-attribute query resolution. Similarly in terms of space complexity, the traditional approach will require $(M_{ip} * N_{attr})$ space whereas by using Bloom-filters the space required will be only $const * N_{attr}$ ($const$ is the size of Bloom-filter). The later approach hence reduces the space complexity in turn reducing the communication cost and message overhead.

An individual SR nodes after receiving the rBF checks for the list of IP addresses which pass through the rBF, using the same hash functions from $H()$ it utilized to create the nBF and retrieves the final list of IP addresses for each separate attribute. After generating the list of IP addresses these nodes send it back to the Q_{node}, as described in Algorithm 3.

Algorithm 3 Extracting IP Address

```
1: $IP_{final} = []$ initialized as a blank list
2: for Each ip list in $IP$ do
3:     $hvals \leftarrow k$ hash values for $ip$ obtained from $H()$
4:     for Each hash value ($h$) in $hvals$ do
5:         $i \leftarrow h \mod m$
6:         if All $i$ satisfies $rBF$ then
7:             $Q_{ip}.append$(the selected $ip$)
8:         end if
9:     end for
10: end for
11: Send $Q_{ip}$ back to the $Q_{node}$
```

The Q_{node} creates a final list obtained through the intersection of the list of IP addresses received and cleans up the residue IPs, if any. This final list of IP address is then sent to the user, refer Figure 3.3 with an amount of f_p supposed to be generated during the process. The algorithm for the final list of IP is as described in Algorithm 4

Algorithm 4 Generate final list of IP Address

```
1: $IP_{final} = []$ initialized as a blank list
2: for Each $Q_{ip}$ obtained for each query in $Q_{attr}$ do
3:     if $Q_{ip}$ obtained for the first time then
4:         $IP_{final} \leftarrow Q_{ip}$
5:     else
6:         $IP_{final} \leftarrow IP_{final} \wedge Q_{ip}$
7:     end if
8: end for
9: Send $IP_{final}$ to the user
```

The source code for the implementation of ROR architecture is presented in Appendix-A.

CHAPTER 4

Caching Methodology

In certain situations, it is found that one set of attribute values, are highly correlated to a different set of attribute values. Each of these attributes has range of values associated with them. In the description in previous chapters, Queries related to different attributes are resolved by resolving them separately. Here we present a new methodology to more efficiently resolve the attribute values. One of these methodology discussed here is the **Caching** Mechanism. The main objective behind the methodology is the reduction in hop count, processing time, and latency of response. Also as the queries generated for these kind of attributes are correlated, e.g. if a query is looking for an attribute with some specific value there might be range of values for the other set of attributes. In terms of statistical measure, it is noticed that the query related to range values for each individual attribute fluctuates together, it means the values for attribute variables increase or decrease in parallel. This kind of correlation is also referred as positive correlation. This analysis of correlation of attributes can also be understood from the probability distribution (p.d.f) for each pair of correlated attribute. For example the p.d.f for DSize and DFree is as shown in Figure 4.1 and that for T_X and R_X is shown in Figure 4.2.

It can be noted from the distribution of these attributes that they have mean (μ) and std-deviation (σ) close to each other and also the distribution for each of these attributes also lies in the same range. As a result fluctuation for values of both these attributes lies more or less in the same range.

Since in scalable systems like conventional grids, desktop grids, cloud systems, etc.. resolution of multi-attribute queries needs to be done at a higher rate with minimum latency in

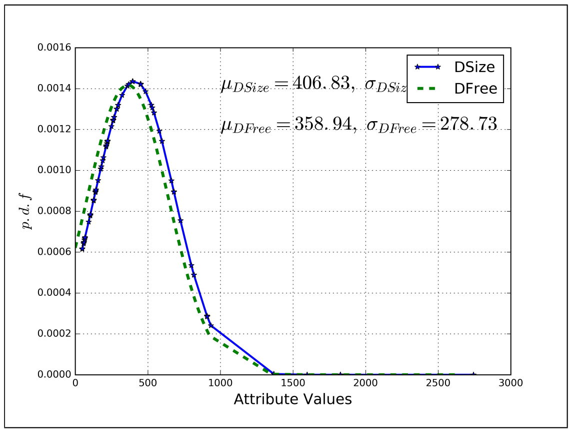


FIGURE 4.1. Probability Distribution of DSize and DFree

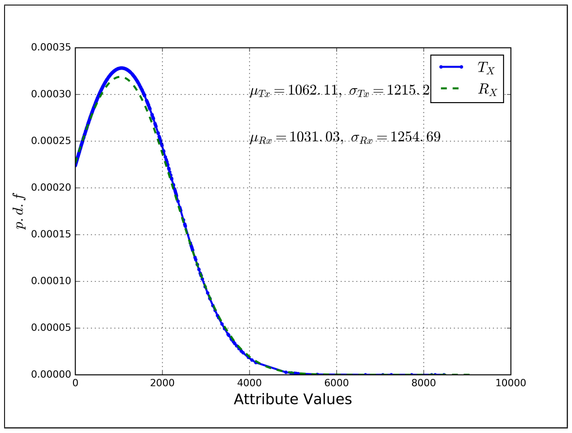


FIGURE 4.2. Probability Distribution of T_X and R_X

response, it therefore becomes necessary to have an architecture which resolves these queries based on the correlation feature of the attribute values. This architecture targets and remains successful in resolving the look-up in minimum time, reducing the latency in response. The novelty of the proposed architecture discussed here lies in the arrangement of values for each correlated architecture pair within the same sub-ring in such a way that the query for both the attributes are resolved while traversing along the same sub-ring. This is done in a way in which a value or range of values of one correlated attribute should always point to the other attributes having a set of values in a range, so that the lookup process does not traverse the entire range of the other attributes to resolve the query.

To design the architecture for the resolution of correlated attributes in a manner discussed above, we started looking into the correlated attribute magnitudes for each and every resource of PlanetLab nodes [23]. These correlated attribute resource values used for simulation are generated using the simulation tool *ResQue* [7]. The attributes considered for the simulation environment are attributes of PlanetLab Nodes [23]. A typical set (S) of resources of PlanetLab Nodes with different correlated attributes namely, one minute loading (1mLd), five Minutes loading (5mLd), fifteen minutes loading (15mLd), disc size (DSize), disc free

Table 4.1. PlanetLab Co-related Attribute Details

Attrb	*max*	*min*	*range*	Co-related Attrb	*p.c.f*
1mLd	100.6	7.3	93.3	[5mLd, 15mLd]	[0.95, 0.91]
5mLd	39.74	0.03	39.71	[1mLd, 15mLd]	[0.95, 0.99]
15mLd	37.43	0.0	37.43	[1mLd, 5mLd]	[0.91, 0.99]
DSize	2742.78	47.55	2695.22	[DFree]	[0.99]
DFree	2640.43	0.00	2640.43	[DSize]	[0.99]
T_x	8465.0	0.0	8465	[R_x]	[0.89]
R_x	9078.0	0.0	9078.0	[T_x]	[0.89]

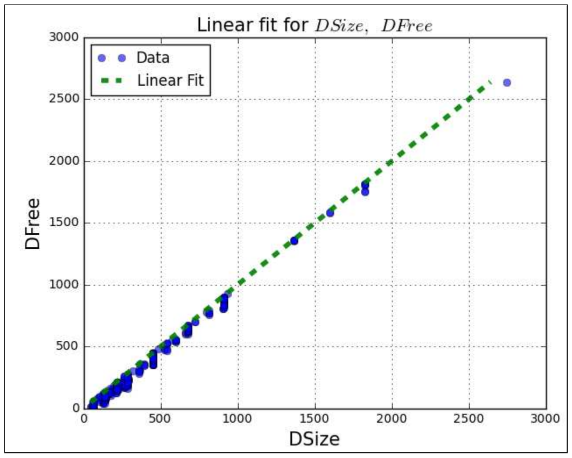


Figure 4.3. Variation of magnitude of DFree w.r.t DSize

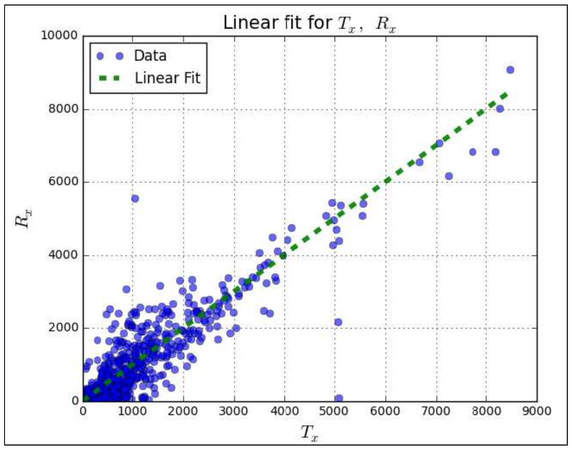


Figure 4.4. Variation of magnitude of R_X w.r.t T_X

(DFree), transmission rate (T_x), receiving rate (R_x) alongwith their *min*, *max*, *range*, and Pearsons correlation coefficient (*p.c.f*) for each of the correlated attributes is provided in Table 4.1.

The variation of magnitude for the correlated attributes DFree w.r.t DSize is shown in Figure 4.3, R_X w.r.t T_X is shown in Figure 4.4, 5mLd w.r.t 1mLd is shown in Figure 4.5 and 15mLd w.r.t 1mLd is shown in Figure 4.6. The Figures 4.3-6 also shows the linear curve fit for each of the correlated attributes, when they are plotted w.r.t each other.

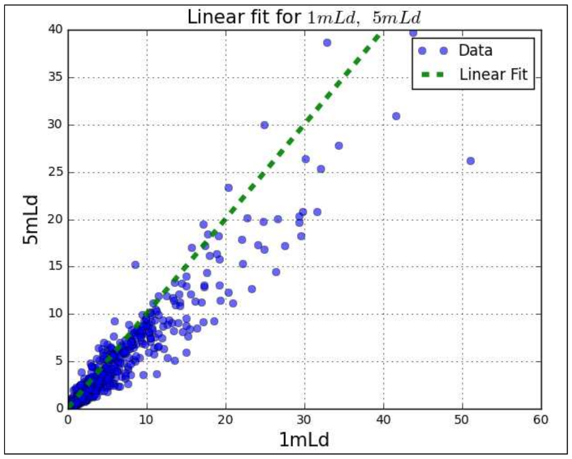


FIGURE 4.5. Variation of magnitude of 5mLd w.r.t 1mLd

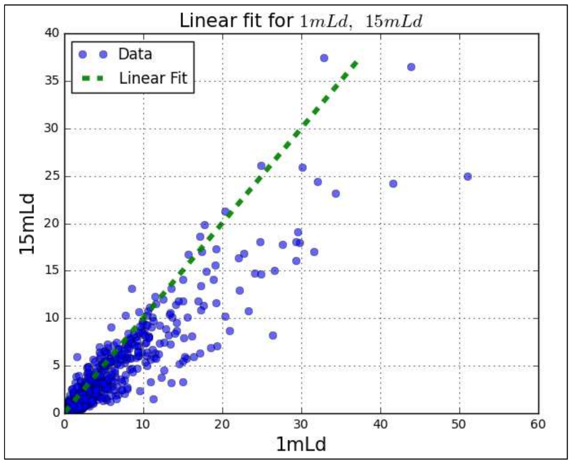


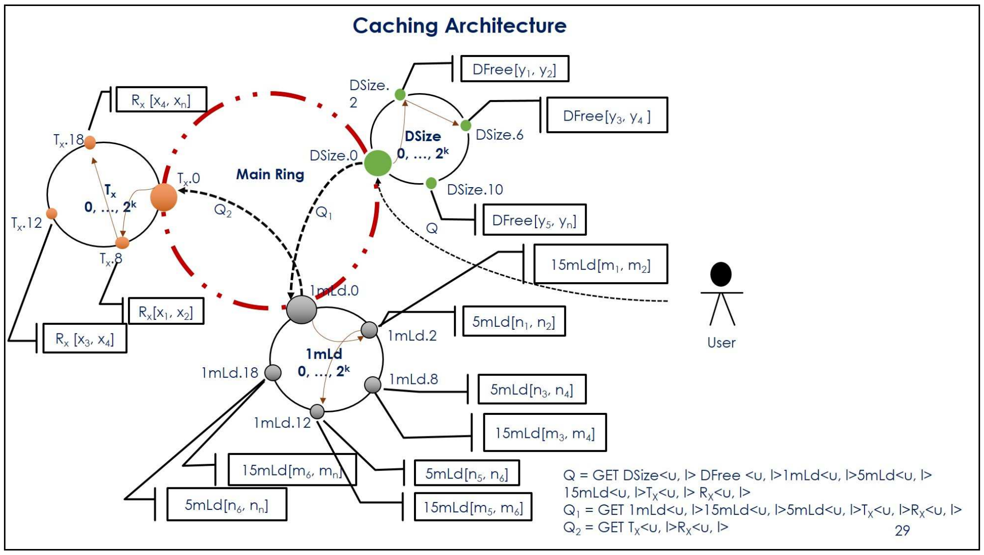
FIGURE 4.6. Variation of magnitude of 15mLd w.r.t 1mLd

4.1. System Architecture

From the Figures 4.1-4, it can be noticed that for a single value of one attribute say e.g. attribute DSize there are range of values of the other attribute DFree to which the former is co-related. Similarly for a value of attribute T_X, there are range of values for R_X, for 1mLd there are range of values for 5mLd, and 15mLd. These range of values for each pair of correlated attributes are as described in Table 4.2, representing the minimum, maximum and number of resources for each pair of correlated attributes $[DSize, DFree]$, $[T_x, R_x]$ and $[1mLd, 5mLd, 15mLd]$ which has a range of values on the basis of other attributes. Also it is noticed that there are considerable number resources which falls in this range of values. So instead of making separate sub rings (SR) in the ROR P2P architecture for each of the different attributes, it is therefore possible to cache the data of other attributes in the same SR with increasing value of one of the attribute. The network architecture for this kind of arrangement is as shown in Figure 4.7. Each of the correlated attributes are placed in the same SR, e.g. DSize and DFree are correlated and they both are placed in the same SR, ref Figure 4.7. Each node in SR, responsible for values of DSize caches the corresponding range

TABLE 4.2. Resource Details

	$DSize$	$DFree$	Num. of Resources
$DSize_{min}$	47	$[0 - 4]$	1
$DSize_{max}$	2742	$[2640 - 2640]$	270
	T_x	R_x	Num. of Resources
Tx_{min}	0	$[0 - 0]$	1
Tx_{max}	8465	$[9078 - 9078]$	20
	$1mLd$	$5mLd$	Num.. of Resources
$1mLd_{min}$	7.3	$[0 - 2]$	40
$1mLd_{max}$	51	$[39 - 39]$	1
	$1mLd$	$15mLd$	Num. of Resources
$1mLd_{min}$	7.3	$[0 - 2]$	1
$1mLd_{max}$	51	$[36 - 37]$	129



Q = GET DSize<u, l> DFree <u, l>1mLd<u, l>5mLd<u, l>
15mLd<u, l>T_X<u, l> R_X<u, l>
Q_1 = GET 1mLd<u, l>15mLd<u, l>5mLd<u, l>T_X<u, l>R_X<u, l>
Q_2 = GET T_X<u, l>R_X<u, l>

FIGURE 4.7. Caching Architecture for Co-related attributes

of values for DFree at the same time. In case there are three correlated attribute, the nodes in SR corresponding to values to one attribute caches the values for the other two attributes at the same time. For example in Figure 4.7, the nodes in SR responsible for $1mLd$, caches the values for $5mLd$ and $15mLd$ as well.

Table 4.3. Caching Architecture Hash-Table formation for Resources

$< key >$	$< value >$	$< IPAddress >$
k_1	$[v_1 1, ..., v_{1m}]$	$[ip_{11}, ..., ip_{1m}]$
...	[...]	[...]
k_n	$[v_{n1}, ..., v_{nm}]$	$[ip_{n1}, ..., ip_{nm}]$

4.2. Caching Architecture Formation

The nodes in SR as well as the main ring (MR), while coming up calculates the range values for each of the attribute it will be responsible for on the basis of its SR id, $([\frac{l_q}{MF}, \frac{u_q}{MF}])$ where MF is obtained from Equation 4.1.

$$MF = \frac{r_n}{2^k - 1}, where\, r_n = [u_n - l_n] \tag{4.1}$$

The nodes in SR responsible for the co-related attribute pair, e.g $[DSize, DFree]$ will prepare a Hash Table, (format $key \leftrightarrow < values, ipaddresses >$) where key represents each individual value of one attribute (here $DSize$), and $values$ in $< values, ipaddresses >$ will be range of values of the other attribute or attributes (here $DFree$), for that particular key value. If there are more than two attributes co-related to each other then multiple Hash Tables needs to be prepared for each tuple. The $ipaddress$ are IP addresses of the machines for each $[DSize, DFree]$ tuples. Like wise the nodes in SR responsible for $[T_x, R_x]$ and $[1mLd, 5mLd, 15mLd]$ will also maintain a hash table respectively. The structure of the hash table for the attributes are as shown in Table 4.3

The algorithm for selecting the list of IP addresses for the range of the correlated attributes is as mentioned in Algorithm 5.

Algorithm 5 Selection of IP Addresses in Caching

```
1: $IP = [ip_1, ip_2, .., ip_n]$ list of IP Addresses
2: $Attrb = [attr_1, attr_2, attr_3]$ list of correlated attributes
3: $ind_{attr_2} = []$ initialized a blank list
4: $ind_{attr_3} = []$ initialized a blank list
5: **for** Each $vAttrb$ in $attr_1$ **do**
6:     **for** Each $vAttrb$ in $attr_2$ **do**
7:         **if** $(vAttrb >= l_{rq})$ and $(vAttrb <= u_{rq})$ **then**
8:             $ind_{attr_2}.append$(indice of vAttrb)
9:         **end if**
10:        **for** Each indice in $ind_{attr_2}$ **do**
11:            $vAttrb \leftarrow attr_3$[indice]
12:            **if** $(vAttrb >= l_{rq})$ and $(vAttrb <= u_{rq})$ **then**
13:                $ind_{attr_3}.append$(indice of vAttrb)
14:            **end if**
15:        **end for**
16:    **end for**
17: **end for**
18: $resultant_{IP} = []$ initialized a blank list
19: **for** Each indice in $ind_{attr_3}$ **do**
20:     $resultant_{IP}.append(IP$[indice]$)$
21: **end for**
```

4.3. Query Resolution

A sample query (Q) for multi-attributes looks like

$$Q \equiv (a_{1q} = [l_{1q}, u_{1q}]; a_{2q} = [l_{2q}, u_{2q}]; ...; a_{rq} = [l_{rq}, u_{rq}])$$

where $(a_{1q}, a_{2q}, ..., a_{rq}) \epsilon A$ and $[l_{1q}, l_{2q}, ..., l_{rq}]$ are the lower limits and $[u_{1q}, u_{2q}, ..., u_{rq}]$ are the upper limits for each of the attributes. The user generates query Q to the Q_{node}. The selection of Q_{node} is from the MR node and is entirely random. The Q_{node} looks into the first attribute in Q, and forwards it to the node responsible for it in MR. The node which gets Q checks whether it has got query for the attribute it is responsible for, if so, it parses out Q, retrieves the upper limit and lower limit of the attribute and send out the rest of the query to the node responsible for the immediate attribute, which it comes to know from the Routing Table (RT). The preparation of RT is exactly in the same way as it is for the ROR

architecture. The next node on receiving the query, parses out its own attribute, with the upper limit and the lower limit and sends out the remaining query. This process continues until the query reaches the last node with its own attribute only and has nothing to forward. The node after receiving its own query retrieves the list of IP addresses using Algorithm 5. After retrieving the IP address of the resources makes a normal Bloom-filter (nBF). The parameters necessary for the creation of nBF are k representing number of Hash-Functions (hf), N representing the number of resources (here count of IP addresses) and m representing the size of nBF. The parameter k derived from m and N is as mentioned in Equation 4.2

$$k = \frac{m}{N} * \ln 2 \tag{4.2}$$

The BF data structure doesn't have any false-negative associated with it, but it has certain amount of false-positive (f_p) associated with it[14]. By convention and improved results the value of f_p should be as much minimum as possible. For minimum value of f_p, the false positive probability p per fraction of bit is considered to be 1, as mentioned below in Equation 4.3

$$\ln p = -\frac{m}{N} * (\ln 2)^2 \tag{4.3}$$

Each of the IP address is hashed k number of times with the hash functions in $H()$ and the *mod* is calculated using m. The *mod* value represents the indices of the Bloom-filter, and is used to populate the nBF by setting the represented indices to 1 if it is 0 initially, or else it is left unchanged.

These nBFs are then sent to the Q_{node} with a check-sum, to prevent data loss. The Q_{node} does a bit-wise AND operation with all the nBFs received and makes the resultant Bloom-filter (rBF) and sends it back to the respective clients with a check-sum, as shown in Figure 3.3. The algorithm for calculating the rBF is as mentioned in Algorithm 6.

Algorithm 6 Calculate resultant Bloom-filter

```
1: for Each attribute in $Q_{attr}$ do
2:     if $nBF$ received for the first time then
3:         $rBF \leftarrow nBF$
4:     else
5:         $rBF \leftarrow$ bitwise-AND of $(rBF, nBF)$
6:     end if
7: end for
8: Send $rBF$ back to the nodes
```

Now, the clients after receiving the rBF checks the IP addresses which pass through the rBF. For this it uses the same set of hash functions mentioned in $H()$, it utilized to create the nBF. After generating the list of IPs these clients send it to the Q_{node}, as mentioned in Algorithm 7.

Algorithm 7 Extracting IP Address

```
1: $IP_{final} = []$ initialized as a blank list
2: for Each ip list in $IP$ do
3:     $hvals \leftarrow k$ hash values for $ip$ obtained from $H()$
4:     for Each hash value ($h$) in $hvals$ do
5:         $i \leftarrow h \mod m$
6:         if All $i$ satisfies $rBF$ then
7:             $Q_{ip}.append$(the selected $ip$)
8:         end if
9:     end for
10: end for
11: Send $Q_{ip}$ back to the $Q_{node}$
```

The Q_{node} creates a final list of IP addresses obtained through intersection of the list of IP addresses received and the residue IPs are removed. This final list of IP address is then sent to the user, refer Figure 3.3. In the process of creating the final list of IP address, an amount of f_p is generated during the process. The algorithm for the final list of IP is as described in Algorithm 8

Algorithm 8 Generation of final list of IP Address

```
1: $IP_{final} = []$ initialized as a blank list
2: **for** Each $Q_{ip}$ obtained for each query in $Q_{attr}$ **do**
3:     **if** $Q_{ip}$ obtained for the first time **then**
4:         $IP_{final} \leftarrow Q_{ip}$
5:     **else**
6:         $IP_{final} \leftarrow IP_{final} \wedge Q_{ip}$
7:     **end if**
8: **end for**
9: Send $IP_{final}$ to the user
```

The source code for the implementation of Caching methodology is presented in Appendix-B.

CHAPTER 5

Overlapped Architecture

Overlapped ring architecture is another methodology, proposed here to resolve multi-attribute queries where the attributes are co-related to each other. In overlapped ring architecture the SRs of individual attributes overlap on one another. The idea behind overlapped architecture is since the attributes are correlated, it is possible to find out the range of values of one attribute from the values of other attribute with which the former is correlated. To make this possible, a characteristics equation needs to found out which satisfies the trend of values for each pair of correlated attributes. The characteristic equation is obtained by having a curve fitting on each pair of values of the correlated attributes.

The curve fitting is done by plotting the correlated attributes w.r.t each other, having any one of the attribute on a log scale, and then figuring the best characteristic equation that fits well on the values of the attributes. The plots for each of the correlated attributes, having the values of one attribute in logarithmic scale with their corresponding characteristic equation is as shown in Figure 5.1-4. The log scale plot for the correlated attributes $[DSize, DFree]$ is as shown in Figure 5.1 and their characteristic equation is mentioned in Equation 5.1, for $[T_X, R_X]$ the log scale plot is shown in Figure 5.2 and their characteristic equation is mentioned in Equation 5.2, for $[1mLd, 5mLd]$ the log scale plot is shown in Figure 5.3 and their characteristic equation is mentioned in Equation 5.3 and for $[1mLd, 15mLd]$ the log scale plot is shown in Figure 5.4 and their characteristic equation is mentioned in Equation in 5.4.

$$f(x) = \ln x^{-10.206} + 1.0284 * x \tag{5.1}$$

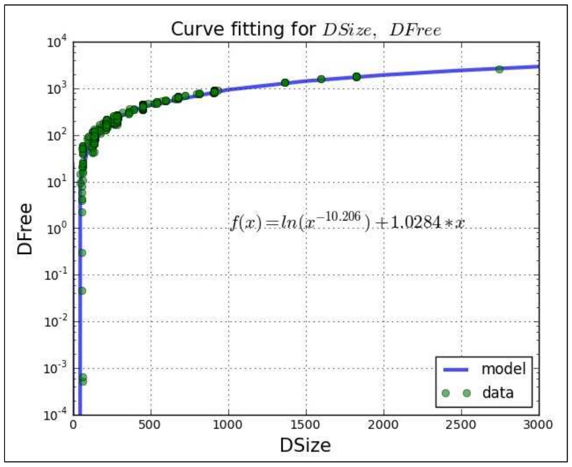


FIGURE 5.1. Variation of DFree (logarithmic) w.r.t DSize with their Characteristic Equation

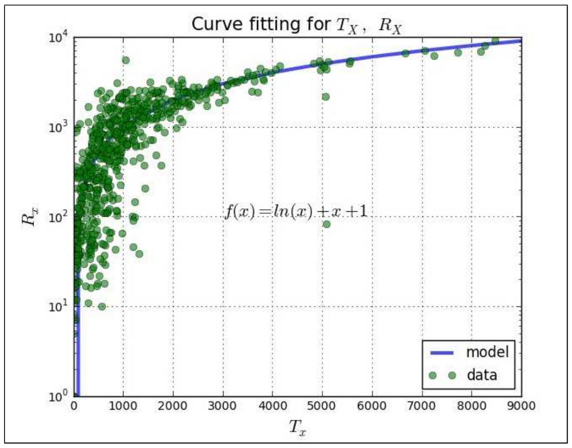


FIGURE 5.2. Variation of R_X (logarithmic) w.r.t T_X with their Characteristic Equation of R_X w.r.t T_X

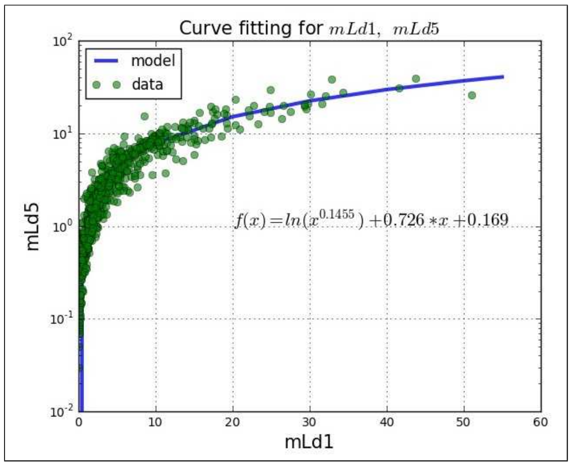


FIGURE 5.3. Variation of 5mLd (logarithmic) w.r.t 1mLd with their Characteristic Equation of 5mLd w.r.t 1mLd

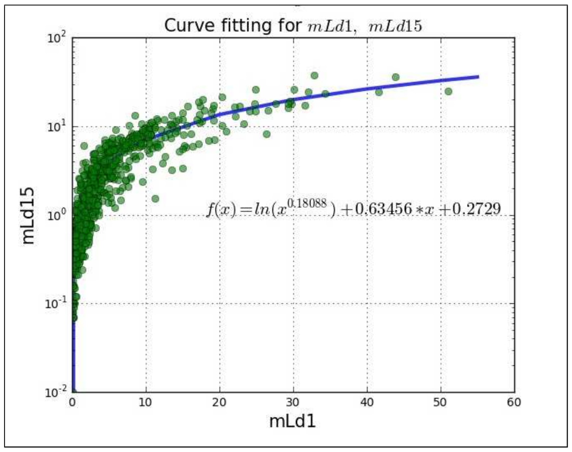


FIGURE 5.4. Variation of 15mLd (logarithmic) w.r.t 1mLd with their Characteristic Equation of 15mLd w.r.t 1mLd

$$f(x) = \ln x + x + 1 \tag{5.2}$$

$$f(x) = \ln x^{0.1455} + 0.726 * x + 0.169 \tag{5.3}$$

$$f(x) = \ln x^{0.18088} + 0.63456 * x + 0.2729 \tag{5.4}$$

5.1. System Architecture

The nodes in SR and also in the MR, while coming up gets their SR id, and hence calculate the range values for the attributes (in the overlapped architecture) it will be responsible for on the basis of its SR id. The ranges are mentioned as $[\frac{l_q}{MF}, \frac{u_q}{MF}]$ where MF is obtained from Equation 5.5. The individual rings which are responsible for each different types of attribute and are correlated to each other are placed overlapped with each other. For example, since $DSize$ and $DFree$ are correlated, both these attributes are placed overlapped on top of one another as shown in Figure 5.5

$$MF = \frac{r_n}{2^k - 1}, where\, r_n = [u_n - l_n] \tag{5.5}$$

After getting the range of one attribute, which each of these nodes are responsible for, it maps the values of corresponding range of other correlated attribute from the characteristics equations mentioned in Equation 5.1-4. In other words, a mapping operation is done to calculate the values of one attribute from the known values of the other attribute. For example, a mapping function f, for the attributes $[DSize, DFree]$ will look like $f : (DSize_1, ..., DSize_n) \Leftrightarrow (DFree_1, ..., DFree_n)$. This mapping function is the characteristic equation obtained from curve fitting, discussed before. For example the mapping function for the pair $[DSize, DFree]$ will be Equation 5.1. These mapping function causes the values of DFree to be retrieved from DSize and vice-versa for the entire range. As a result the SRs for each of these attributes which corresponds to the range values for these attributes can be made to overlap on top of each other. Similarly for $[T_X, R_X]$, $[1mLd, 5mLd]$, and

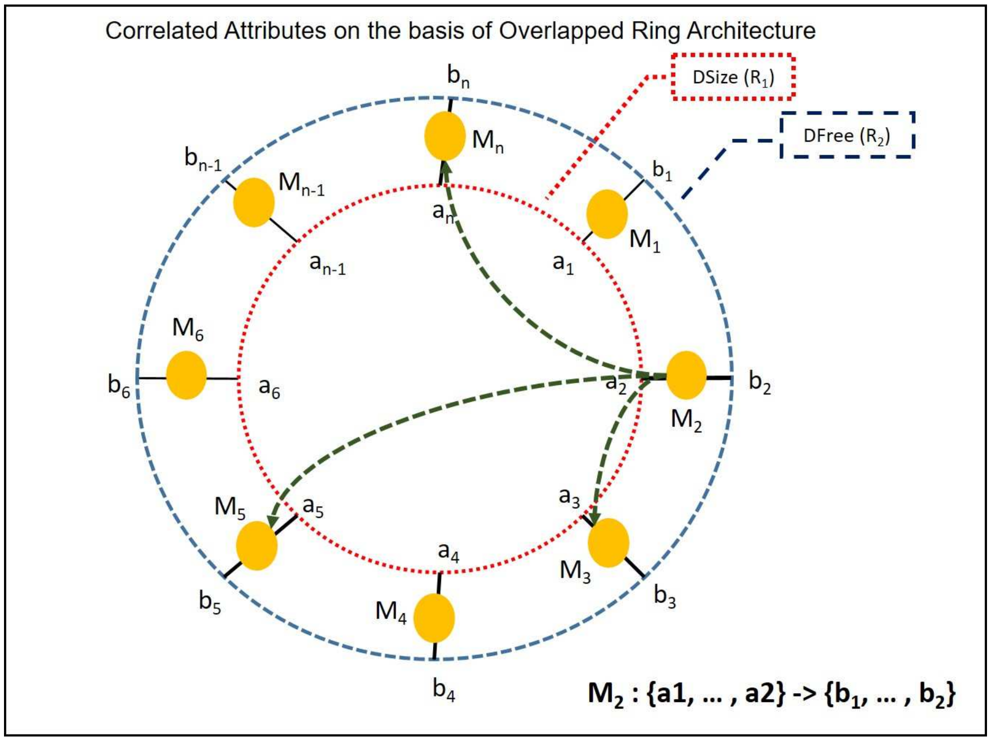
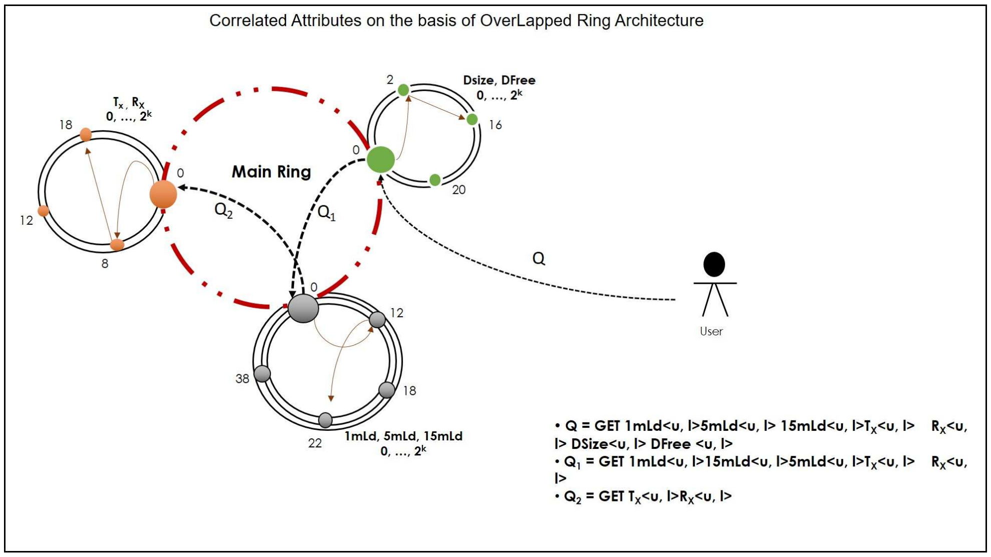

Correlated Attributes on the basis of Overlapped Ring Architecture
DSize (R1)
DFree (R2)
b_n M_n a_n
b_1 M_1 a_1
b_2 M_2 a_2
b_3 M_3 a_3
b_4 M_4 a_4
b_5 M_5 a_5
b_6 M_6 a_6
b_{n-1} M_{n-1} a_{n-1}
$M_2 : \{a1, ... , a2\} \to \{b_1, ... , b_2\}$

Figure 5.5. Overlapped Ring formation for DSize and DFree

Table 5.1. Overlapped Ring Architecture Hash-Table formation for Resources

$< key_1, key_2, ..., key_n >$	$< val_1, val_2, ..., val_n >$	$< IPAddresses >$
$< k_{11}, ..., k_{1n} >$	$< v_{11}, ..., v_{1n} >$	$< ip_{11}, ..., ip_{1n} >$
$< ... >$	$< ... >$	$< ... >$
$< k_{m1}, ..., k_{mn} >$	$< v_{m1}, ..., v_{mn} >$	$< ip_{m1}, ..., ip_{mn}]$

$[1mLd, 15mLd]$, the mapping function for each pair is represented by the characteristic equations mentioned in Equation 5.1-4. Since these equations map the entire range of values for each attribute pair, the SRs responsible for the range value of each attribute are made to be overlapped on each other. Based on the range of values of the mapping function the nodes in the overlapped ring will retrieve the IP address of the resources which falls in that range hence resolving the query. Here also the nodes maintain a Hash Table as mentioned in Table 5.1 (format $key \leftrightarrow < values, ipaddresses >$), where the $< key >$ represent each individual value one attribute and $< values >$ are the respective values of the other attribute with $ipaddress$ mentioning the IP address of the resource.



• Q = GET 1mLd<u, l>5mLd<u, l> 15mLd<u, l>T_X<u, l> R_X<u, l> DSize<u, l> DFree <u, l>
• Q_1 = GET 1mLd<u, l>15mLd<u, l>5mLd<u, l>T_X<u, l> R_X<u, l>
• Q_2 = GET T_X<u, l>R_X<u, l>

FIGURE 5.6. Overlapped Ring Architecture for Co-related attributes

5.2. Overlapped Architecture Formation

The initial part of the ring formations follows the same pattern as the caching architecture formation (ref. Chapter 4). The nodes while coming up knows the set of correlated attributes it will be responsible for and calculates the range on the basis of the characteristic equation. At the end each of these overlapped rings are placed on the ROR P2P architecture to resolve queries for multi-attributes as shown in Figure 5.6. The algorithm for selecting the IP address in an overlapped ring is as described in Algorithm 9

5.3. Query Resolution

A query (Q) for multi-attributes looks like

$Q \equiv (a_{1q} = [l_{1q}, u_{1q}]; a_{2q} = [l_{2q}, u_{2q}]; ...; a_{rq} = [l_{rq}, u_{rq}])$

where, $(a_{1q}, a_{2q}, ..., a_{rq}) \epsilon A$ and $[l_{1q}, l_{2q}, ..., l_{rq}]$ are the lower limits and $[u_{1q}, u_{2q}, ..., u_{rq}]$ are the upper limits for each of the attributes. User generates query Q to the Q_{node}. This query node can be selected in random from the list of nodes present in MR. The Q_{node} looks into

Algorithm 9 Selection of IP Addresses in Overlapped Ring

```
1: $IP = [ip_1, ip_2, .., ip_n]$ list of IP Addresses
2: $Attrb = [attr_1, attr_2]$ list of correlated attributes
3: $ind_{attr_2} = []$ initialized a blank list
4: $f \leftarrow$ Characteristic function for $attr_1$
5: for Each $vAttrb$ in $attr_1$ do
6:     Calculate $vAttrb$ for $attr_2$ using $f$
7:     $vAttrb_{attr_2} \leftarrow$ from $f$
8:     if $vAttrb_{attr_2}$ in $range(l_{rq}), u_{rq})$ then
9:         $ind_{attr_2}.append($indice of $vAttrb_{attr_2})$
10:    end if
11: end for
12: $resultant_{IP} = []$ initialized a blank list
13: for Each indice in $ind_{attr_2}$ do
14:     $resultant_{IP}.append(IP[$indice$])$
15: end for
```

the first attribute in Q, and forwards it to the node responsible for it in MR. The node which gets Q checks whether it has got query for the attribute it is responsible for, if it is so it parses out Q, retrieves the upper limit and lower limit of the attribute and send out the rest of the query to the node responsible for the next immediate attribute, which it comes to know from the RT (formed using the usual procedure mentioned in Chapter 3). The next node then parses out its own attribute with the upper limit and the lower limit and sends out the remaining query. This process continues until the query reaches the last node with its own attribute only and has nothing to forward.

The node after retrieving the IP address for the resources makes a normal Bloom-filter (nBF). The parameters necessary for the creation of nBF are k representing number of Hash-Functions (hf), N representing the number of resources (here count of IP addresses) and m representing the size of nBF. The parameter k derived from m and N is as mentioned in Equation 5.6

$$k = \frac{m}{N} * \ln 2 \tag{5.6}$$

The BF data structure doesn't have any false-negative associated with it, but it has certain amount of false-positive (f_p) associated with it[14]. By convention and improved results the value of f_p should be as much minimum as possible. For minimum value of f_p, the false positive probability p per fraction of bit in 1, is as mentioned below in Equation 5.7

$$\ln p = -\frac{m}{N} * (\ln 2)^2 \tag{5.7}$$

Each of the IP address is hashed k number of times with the set of hash functions from $H()$ and the modulus is calculated w.r.t m. The modulus represents indices of Bloom-filter, and are then used to populate the nBF by setting its respective indices to 1 if it is 0 initially, or else it is left unchanged.

These nBFs are then sent to the Q_{node} with a check-sum, to prevent data loss. The Q_{node} does a bit-wise AND operation with all the nBFs received and makes the resultant Bloom-filter (rBF) and sends it back to the respective clients with a check-sum, as shown in Figure 3.3. The algorithm for calculating the rBF is as described in Algorithm 10.

Algorithm 10 Calculate resultant Bloom-filter

```
1: for Each attribute in $Q_{attr}$ do
2:     if $nBF$ received for the first time then
3:         $rBF \leftarrow nBF$
4:     else
5:         $rBF \leftarrow$ bitwise-AND of $(rBF, nBF)$
6:     end if
7: end for
8: Send $rBF$ back to the nodes
```

Now, the clients after receiving the rBF checks for IP addresses which pass through the rBF, using the same set of hash function from $H()$, it utilized to create the nBF. After generating the final list of IP address these clients send it to the Q_{node}, as described in Algorithm 11.

Algorithm 11 Extracting IP Address

```
1: $IP_{final} = []$ initialized as a blank list
2: for Each ip list in $IP$ do
3:     $hvals \leftarrow k$ hash values for $ip$ obtained from $H()$
4:     for Each hash value $(h)$ in $hvals$ do
5:         $i \leftarrow h \mod m$
6:         if All $i$ satisfies $rBF$ then
7:             $Q_{ip}.append($the selected $ip)$
8:         end if
9:     end for
10: end for
11: Send $Q_{ip}$ back to the $Q_{node}$
```

The Q_{node} creates a final list obtained through intersection of the list of IPs received each individual client. In the process, the residue IP addresses are removed. This final list of IP address is then sent to the user, refer Figure 3.3 with the amount of f_p supposed to be generated during the process. The algorithm for the final list of IP is as described in Algorithm 12.

Algorithm 12 Generation of final list of IP Address

```
1: $IP_{final} = []$ initialized as a blank list
2: for Each $Q_{ip}$ obtained for each query in $Q_{attr}$ do
3:     if $Q_{ip}$ obtained for the first time then
4:         $IP_{final} \leftarrow Q_{ip}$
5:     else
6:         $IP_{final} \leftarrow IP_{final} \wedge Q_{ip}$
7:     end if
8: end for
9: Send $IP_{final}$ to the user
```

The source code for the implementation of Overlapped Ring architecture is presented in Appendix-C.

CHAPTER 6

Simulation and Results

This chapter presents the simulation and results of all the architectures described till now (Multi-Ring: Chapter 3, Caching: Chapter 4, Overlapped Ring: Chapter 5) is discussed. To start with, a set (S) of resources having resource count (R) is considered, where $R \epsilon \mathbb{Z}^+$ can be represented as:

$R \equiv (a_1 = [l_1, u_1]; ...; a_n = [l_n, u_n]; ip = [ip_1, .., ip_n])$

Here $[a_1, a_2, ..., a_n]\ \epsilon A$ where $A \epsilon \mathbb{Z}^+$ represents the set of attributes, with lower limits as $[l_1, l_2, ..., l_n]\ \epsilon \mathbb{Z}^+$ upper limits as $[u_1, u_2, ...u_n]\ \epsilon \mathbb{Z}^+$ respectively and $[ip_1, ip_2, .., ip_n]\ \epsilon IP$ where IP is a set of unique ip addresses for each specific resource.

The multi-attribute resources used for simulation are generated from the resource generation simulation tool *ResQue* [7]. The attribute values are considered to be static, where the values of the attributes remain constant over time. These static attributes are of PlanetLab Nodes [23]. A typical S of resources of PlanetLab Nodes is as mentioned in Table 6.1 with min, max, $range$ and mean (μ), std devn (σ) for each resource.

The CDF plots for each different attribute is shown in the Figure 6.1-4. CDF plot for MSize, MFree%, CSp, NCore, and CFree for each resource is shown in Figure 6.1. CDF plot for DSize and DFree for each resource is shown in Figure 6.2.

CDF plot for T_X, and R_X for each resource is shown in Figure 6.3. CDF plot for 1mLd, 5mLd, and 15mLd for each resource is shown in Figure 6.4.

The probability distribution function (p.d.f) for each attribute is as shown below. The p.d.f for $CFree$ is as shown in Figure 6.5 and CSp is as shown Figure 6.6.

Table 6.1. PlanetLab Attribute Details

Attrb	*max*	*min*	*range*	μ	σ
CSp	3.6	0.731	2.869	2.885	0.301
NCore	8.0	1.0	7.0	6.041	0.301
CFree	100.6	7.3	93.3	96.637	7.927
1mLd	100.6	7.3	93.3	5.342	6.50
5mLd	39.74	0.03	39.71	4.197	5.134
15mLd	37.43	0.0	37.43	3.85	4.71
MSize	31.36	0.21	31.156	2.945	2.221
$MSize_p$	100.0	1.0	99.0	97.95	9.12
DSize	2742.78	47.55	2695.22	406.83	275.40
DFree	2640.43	0.00	2640.43	358.94	278.73
T_x	8465.0	0.0	8465	1062.11	1215.202
R_x	9078.0	0.0	9078.0	1031.029	1254.69

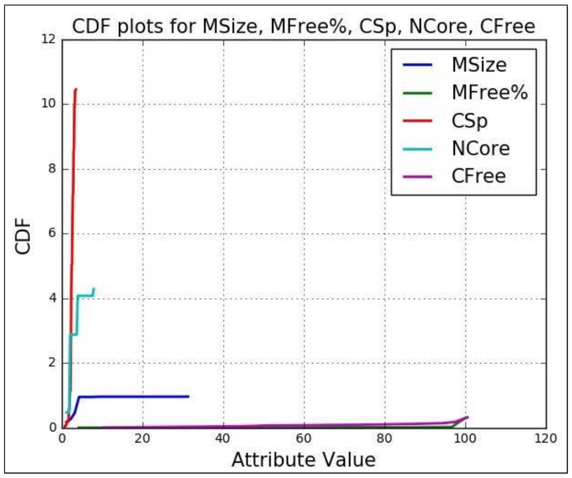


Figure 6.1. CDF plot for MSize, MFree%, CSp, NCore, and CFree for each resource

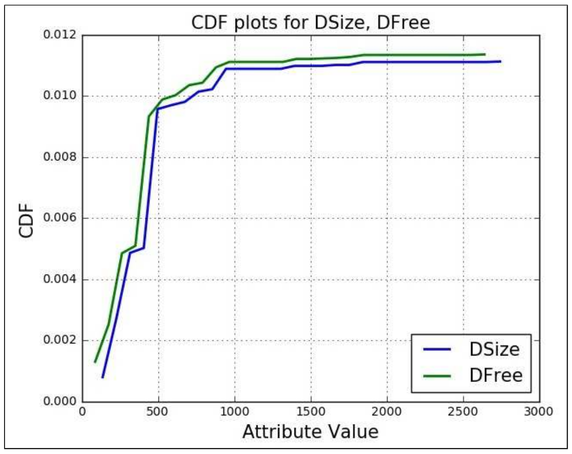


Figure 6.2. CDF plot for DSize and DFree for each resource

The p.d.f for memory size (MSize) and memory free MFree, are shown in Figure 6.7 and Figure 6.8.

The p.d.f for number of cores (NCore) is as shown in Figure 6.9 and the p.d.f for disk size DSize, and disk free DFree is as shown in Figure 6.10. The p.d.f for DSize and DFree as shows has a μ and σ for both the resources are correlated.

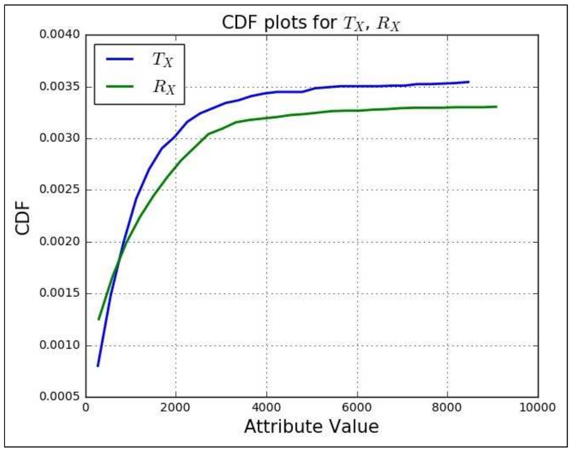


FIGURE 6.3. CDF plot for T_X, and R_X for each resource

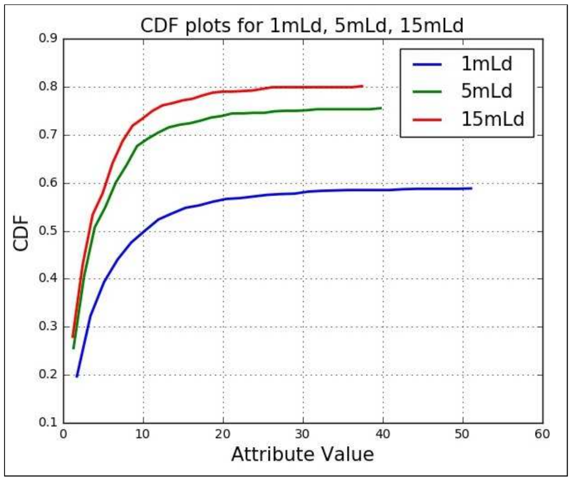


FIGURE 6.4. CDF plot for 1mLd, 5mLd, and 15mLd for each resource

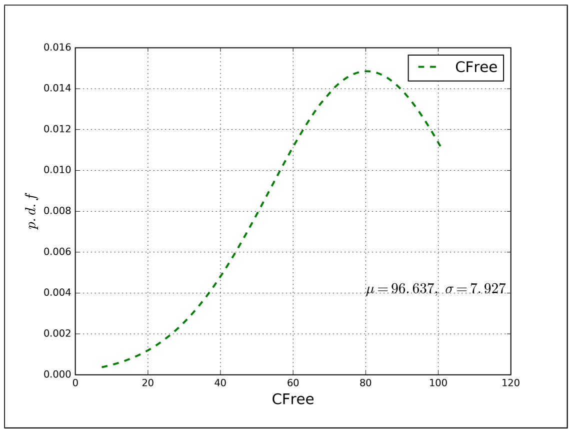


FIGURE 6.5. p.d.f for CFree

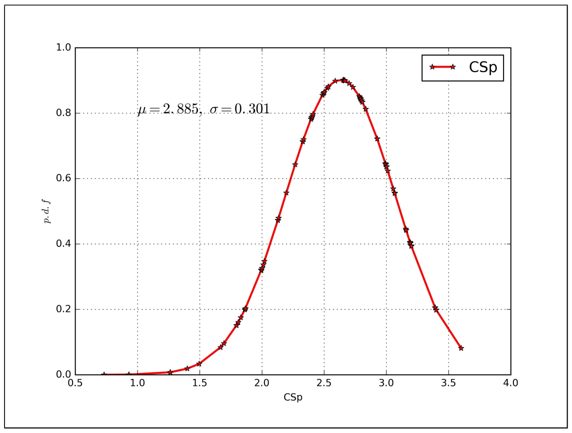


FIGURE 6.6. p.d.f for CSp

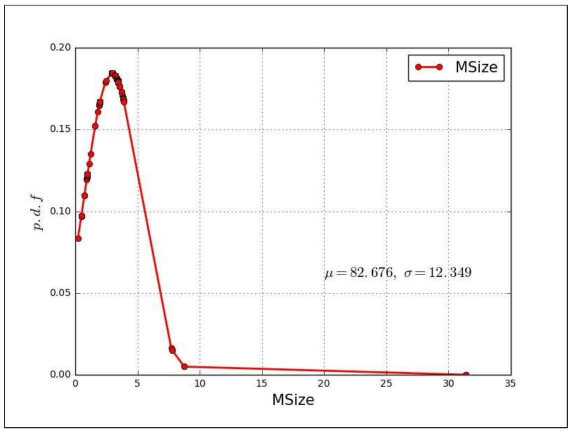


FIGURE 6.7. p.d.f for MSize

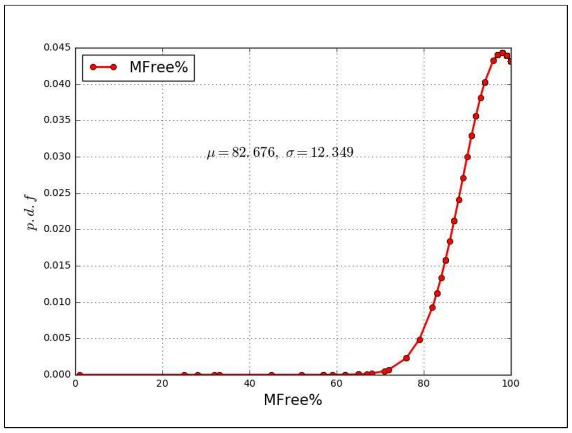


FIGURE 6.8. p.d.f for MFree

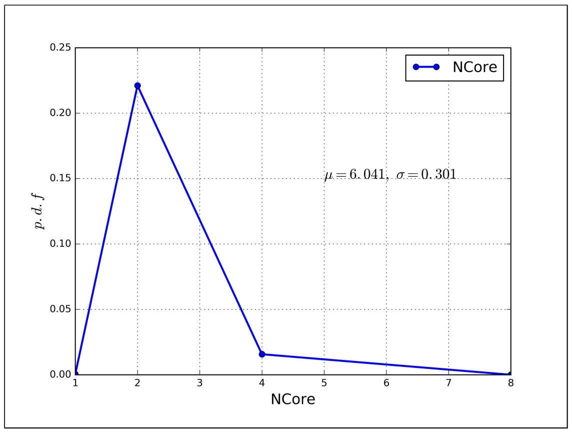


Figure 6.9. p.d.f for NCore

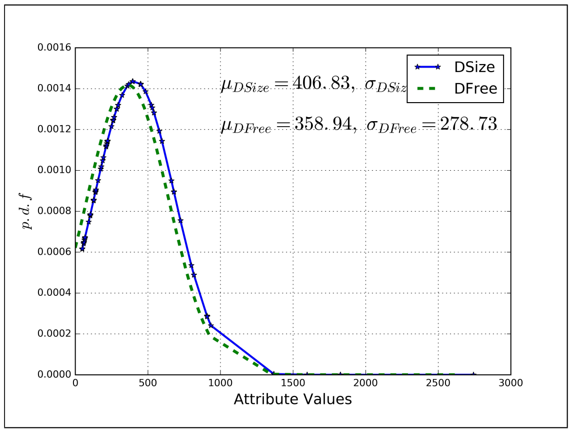


Figure 6.10. p.d.f for DSize and DFree

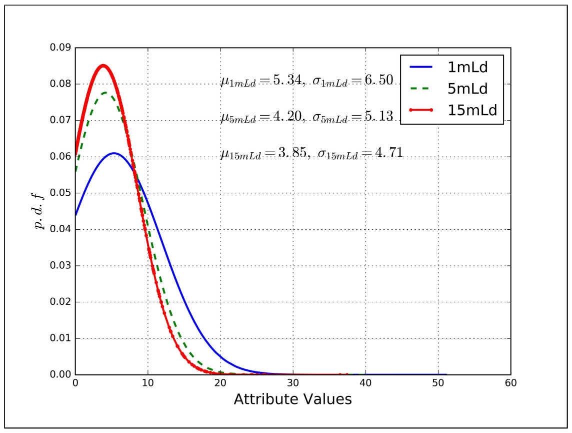


Figure 6.11. p.d.f for 1mLd, 5mLd, and 15mLd

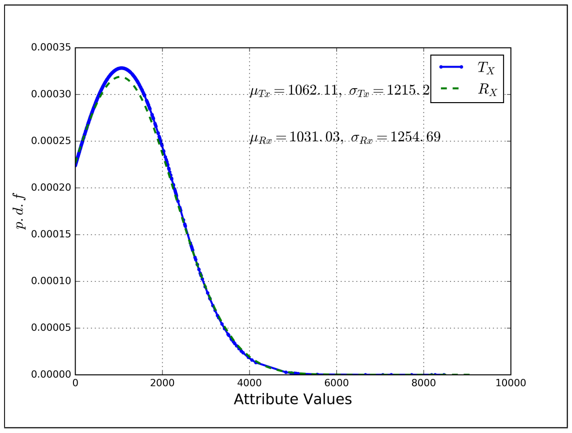


Figure 6.12. p.d.f for T_X and R_X

The p.d.f for 1mLd, 5mLd and 15mLd is as shown in Figure 6.11 and the p.d.f for T_X and R_X is as shown in Figure 6.12. The p.d.f for 1mLd, 5mLd and 15mLd and that of T_X, R_X also shows that they are correlated with μ and σ nearly close to each other.

6.1. Multi-Ring Architecture for Query resolution

In this section multi attribute query resolution with Multi-Ring architecture is discussed. This section also presents the simulation results and analysis for the same. For comparison

Table 6.2. Multi-Attribute Queries for Testing ROR architecture

	Q_1	Q_2
CSp	$[3.6 \sim 0.93]$	$[3.39 \sim 1.26]$
$NCore$	$[8 \sim 1]$	$[8 \sim 1]$
$CFree$	$[100.2 \sim 23.6]$	$[100.6 \sim 7.8]$
$1mLd$	$[18.56 \sim 0.08]$	$[24.85 \sim 0.07]$
$5mLd$	$[12.94 \sim 0.08]$	$[19.78 \sim 0.03]$
$15mLd$	$[11.82 \sim 0.1]$	$[18.05 \sim 0]$
$MSize_p$	$[3.9 \sim 0.73]$	$[3.84 \sim 0.92]$
$MFree$	$[100 \sim 82]$	$[100 \sim 88]$
$DSize$	$[2742.78 \sim 58.87]$	$[909.01 \sim 57.06]$
$DFree$	$[2640.43 \sim 0.30]$	$[896.86 \sim 0]$
R_X	$[8465 \sim 0]$	$[5400 \sim 0]$
T_X	$[9078 \sim 0]$	$[5561 \sim 0]$

and indicating the novelty and efficiency of our methodology, a conventional ROR architecture is used. This conventional ROR architecture doesn't use Bloom-filter messages for its query resolution.

6.1.1. Query Formation:

Queries for our simulation are generated by considering set of values for each attribute. The number of values to be considered in the set is decided by the user. Then lower limit and upper limit values are calculated for each of these set of values corresponding to each attribute. All these attributes along with their lower limit and upper limit, combined together forms a query. Example queries (Q) Q_1, Q_2 with ranges are as mentioned in the Table 6.2.

The source code for generating queries for ROR architecture is presented in Appendix-D. The total number of queries generated to test ROR architecture is 100, and the total run time is approximately 150 minutes.

6.1.2. Simulation Environment:

The number of resources, number of nodes in the test-bed, and the number of attributes for each resource for our simulation is as mentioned

TABLE 6.3. Test Environment Specifications for ROR architecture

Num. of Attributes	12
Num. of Resources	700
Num. of Nodes in test bed	$[24, 36, ..., 84]$

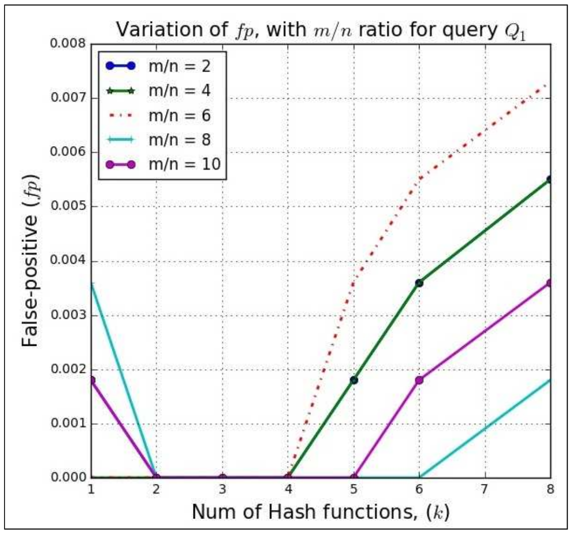


FIGURE 6.13. Variation of false positive for query Q_1 in ROR architecture

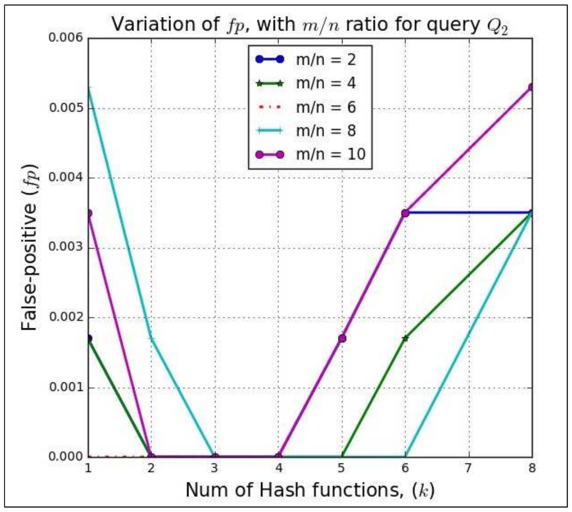


FIGURE 6.14. Variation of false positive for query Q_2 in ROR architecture

in the Table 6.3. Each of the queries Q are generated at random, fed to the Q_{node} which resolves it, and sends the reply back to the user.

The f_p values are calculated for each of the queries for different values of k and m/n ratio. The value of n is count of ip addresses (IP_{list}) for each attributes. The value of n used, is calculated and obtained at each node receiving the query for each and every single attribute. The value of m is kept constant (for performing bit-wise AND). It is noticed from the analysis that the optimum value for k is 6 and m/n ratio is 10, for all the ranges of query Q_1 as shown in Figure 6.13 and that for query Q_2 is shown in Figure 6.14

6.1.3. RESULTS AND ANALYSIS: The number of nodes in the simulation are then increased from 24 to 84 and average hop-counts, average delay in response, average size of messages exchanged and average load per node are measured. Here hop-counts refer to the total number of hops it took to resolve a multi-attribute query. This count is the cumulative count starting from query reaching the Q_{node}, till the generation of final list of resources corresponding to the query. Each query undergoes the following steps and the hops for each step are noted and added up.

- Query traverse along MR starting from Q_{node}, where each MR node parses out attributes they are responsible of along with range
- Each queried attribute along with their range traverses along SR, to create Bloom-filter
- Each of the Bloom-filters from SRs are sent back to Q_{node}
- Q_{node} performs bitwise AND, and creates the resultant Bloom-filter
- Q_{node} sends the Bloom-filter back to the SR nodes
- SR nodes after getting the resultant Bloom-filter creates list of resources
- These list of resources are sent back to the Q_{node}, to perform the final intersection
- After the final intersection, the list of resources generated are sent to user, as a response to the query

Since in our simulation the number of queries generated are 100 for each network configuration (24, 36, 48, .., 84) the average number of hop-count taken to resolve a query with that configuration is noted. The variation in average hop-count for each configuration is as shown in Table 6.4. The variation of hop-count with number of nodes in the test-bed is as shown in Figure 6.15. It is noted that with the increase in the number of nodes the hop-count increases at a logarithmic rate. As shown in Figure 6.15, the rate of increase in hop-count reduces after the number of nodes in the test-bed increases after 60.

The variation of hop-count follows a logarithmic trend as the network scales up. With the increase in number of nodes in SR, the bitspace within that SR is shared by more number

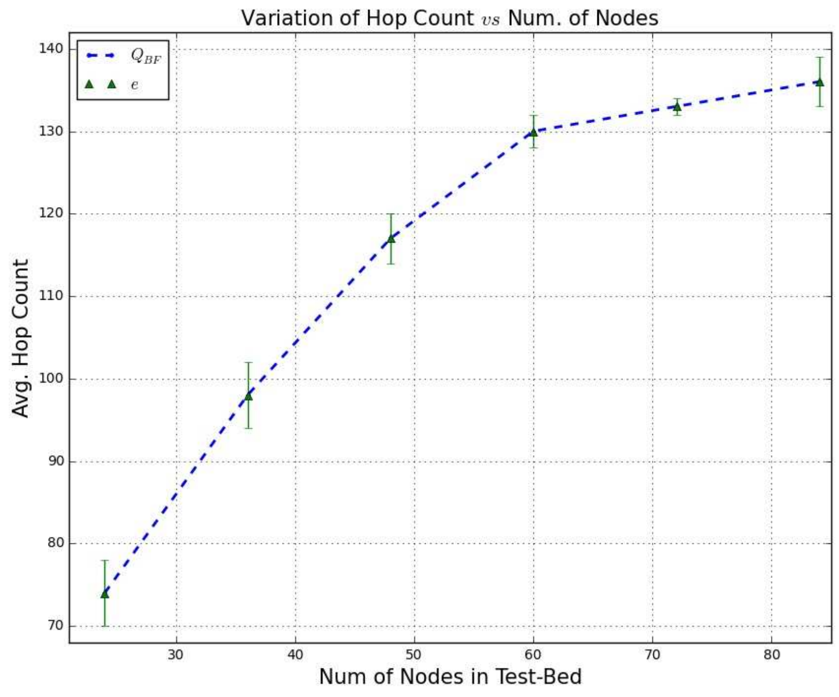


FIGURE 6.15. Variation of avg. Hop-count with Num. of Nodes in ROR architecture

TABLE 6.4. Variation of avg. Hop-count for ROR architecture

Num. of Nodes	Hop-counts in ROR using BF
24	74
36	98
48	117
60	130
72	133
84	136

of nodes, hence more number of ranges for that attribute. Now with the arrival of a query within a SR, this query needs to traverse through more number of nodes to get the queried range satisfied. This causes the hop-count to increase with the increase in the number of nodes. The upper and lower bounds for the hops is as mentioned in Table 6.5

But since Chord protocol is followed both in MR as well as SR the overall complexity for increase in hops is logarithmically bounded. For example, the query to traverse along the MR it takes $\mathcal{O}(M \log N_{MR})$, where M is the number of attributes, and N_{MR} the number of nodes

TABLE 6.5. Ranges of avg. Hop-count for ROR architecture

Num. of Nodes	Range of Hop-counts in ROR using BF
24	[78, 70]
36	[102, 94]
48	[120, 114]
60	[132, 128]
72	[134, 131]
84	[139, 133]

TABLE 6.6. Variation of avg. Delay (msec) in response for ROR architectures

Num. of Nodes	Conv. Sytem (*msec*)	Using BF (*msec*)	Diff (%*age*)
24	738.015	675.119	8.5
36	854.25	790.056	7.5
48	989.18	918.298	7.15
60	1068.796	988.899	7.47
72	1203.431	1139.005	5.35
84	1269.066	1189.416	6.27

in MR. Each queried attribute then takes $\mathcal{O}(\log N_{SR})$ hops within a SR to get it resolved. So the overall complexity for any query to get resolved will take $\mathcal{O}(M * \log N_{MR} * \log N_{SR})$ which is $\mathcal{O}(M \log N)$ where M and N are positive integers.

The average delay in response is as shown in Figure 6.16, is also found to increase with the increase in number of nodes in the network, but as compared to the conventional ROR system the delay is reduced. As mentioned in Table 6.6 the average percent reduction in delay is nearly 7%.

The load per node for each query greatly reduces for each query, with the increase in number of nodes present in the network, indicating the distributed feature of Chord provides load balancing and robustness. Using Bloom-filters the load at each node gets reduced, as compared to the conventional ROR architecture. The variation of average load with number of nodes is as shown in Figure 6.17. The load reduction is nearly 10%, using Bloom-filters as mentioned in Table 6.7.

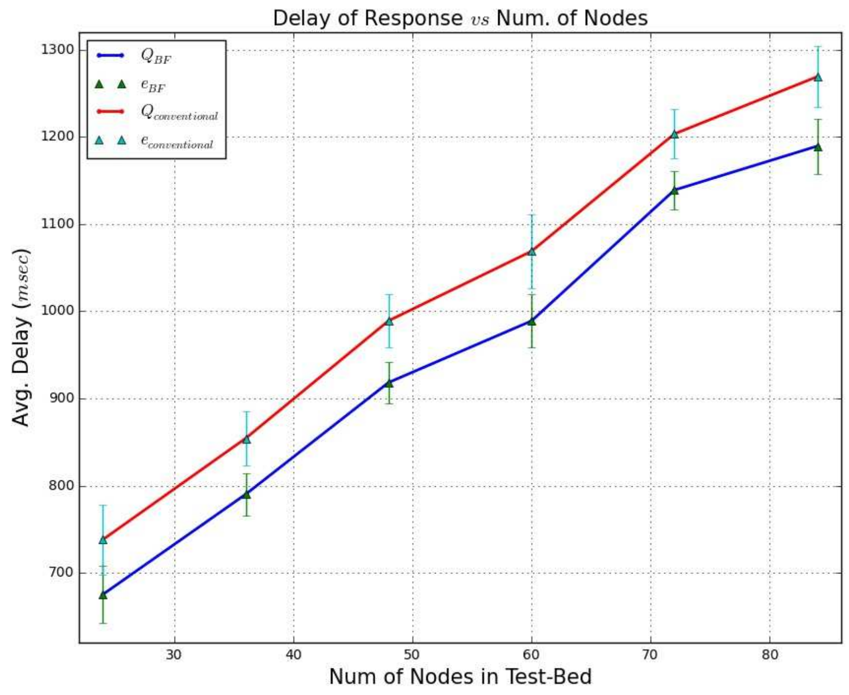


FIGURE 6.16. Variation of avg. Delay in response with Num. of Nodes in ROR architecture

TABLE 6.7. Variation of avg. Load per node for ROR architectures

Num. of Nodes	Conv. Sytem	Using BF	Diff (%age)
24	1297.901	1192.065	8.15
36	896.980	801.494	10.64
48	652.574	601.208	7.87
60	522.474	476.329	8.83
72	435.395	391.108	10.17
84	373.195	323.806	13.23

The size of messages exchanged using Bloom-filter is get reduced by 30%, as compared to the conventional ROR attribute resolution methodology. The variation of message sizes exchanged with the variation in the number of nodes in the test bed is as shown in Figure 6.18. The difference in sizes exchanged is listed in Table 6.8. It is noted that the reduction in message sizes exchanged averages out to nearly 30%, and this indicates that cost of communication and the memory overhead gets reduced using Bloom-filters, than it would

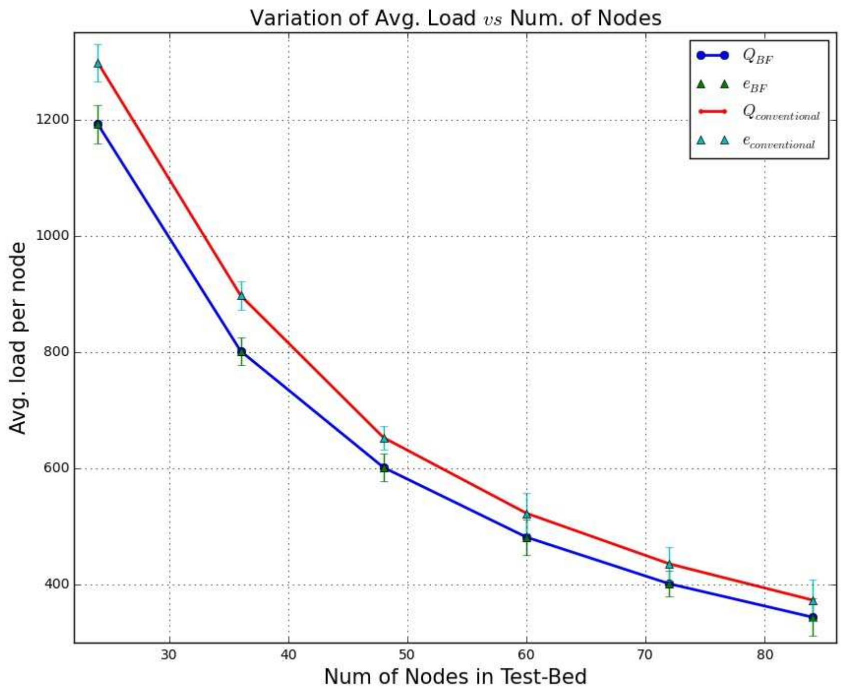


FIGURE 6.17. Variation of avg. Load per Node with Num. of Nodes in ROR architecture

TABLE 6.8. Variation of avg. Msg Size exchanged (bytes) for ROR architectures

Num. of Nodes	Conventional Sytem	Using BF	Diff ($\%age$)
24	220544	143353	35
36	215398	142162	34
48	190955	130301	31.76
60	213476	143028	33
72	220598	156806	28.9
84	211466	151108	28.5

had been if the long list of IP addresses been exchanged between nodes, as in the conventional system.

So it can be inferred that by using the ROR architecture with Bloom-filter messages metrics like hop count, delay in response, load per node and communication cost are significantly reduced for query resolution. It provides great improvement than the conventional ROR system for all the above metrics. Here in this simulation the optimal parameters are

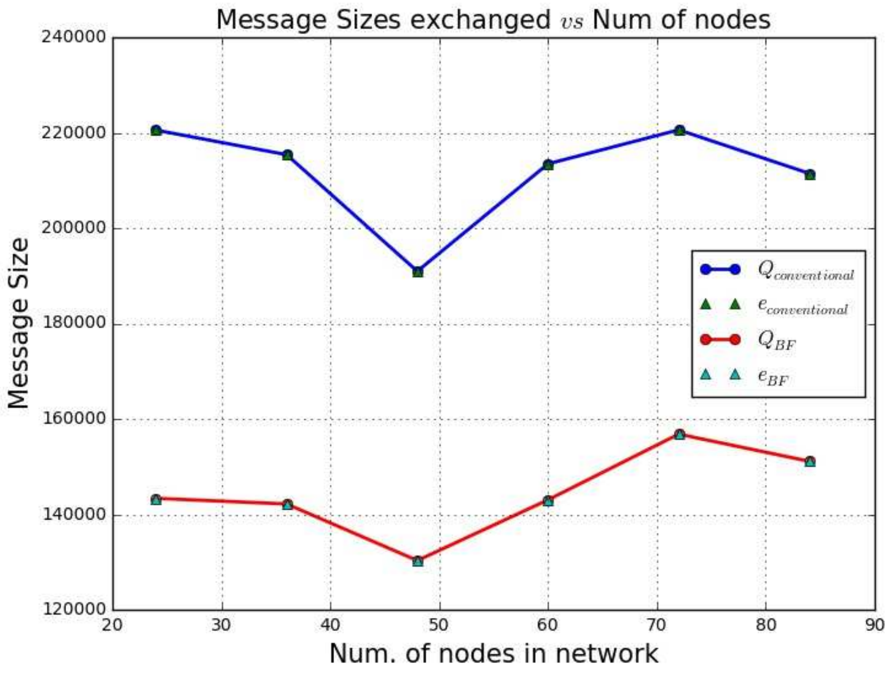


FIGURE 6.18. Variation of avg. Msg Size exchanged with Num. of Nodes in ROR architecture

$k = 6$, $m/n = 10$, and number of nodes for simulation as 60, i.e 5 nodes per attribute. The variation in f_p for the mentioned values of k and m/n ratio w.r.t resource count obtained from simulation is as shown in Figure 6.19.

The difference in resource count obtained from simulation and that of actual count for ROR architecture is as shown in Table 6.9. The difference is less when the number of resources corresponding to a query is less. For example, when the resource count is in the range of $[0, 399]$, ΔN is 0, (refer Table 6.9) whereas when the resource count is in the range $[600, 698]$, ΔN is 3. Here ΔN is the difference in resource count obtained from simulation to that of the actual count. This difference increases with the increase in resource count corresponding to each individual query. The reason being with more number of resources the Bloom-filter gets more and more populated i.e more indices of Bloom-filter are set to 1 from 0. This phenomenon gets reflected in the Resultant Bloom-filter as well. As a result more number

Table 6.9. Variation of difference in resource count with Num. of Resources for ROR architectures

Range of Resource Count (N)	ΔN
$[0, 399]$	0
$[400, 499]$	1
$[500, 599]$	2
$[600, 698]$	3
700	0

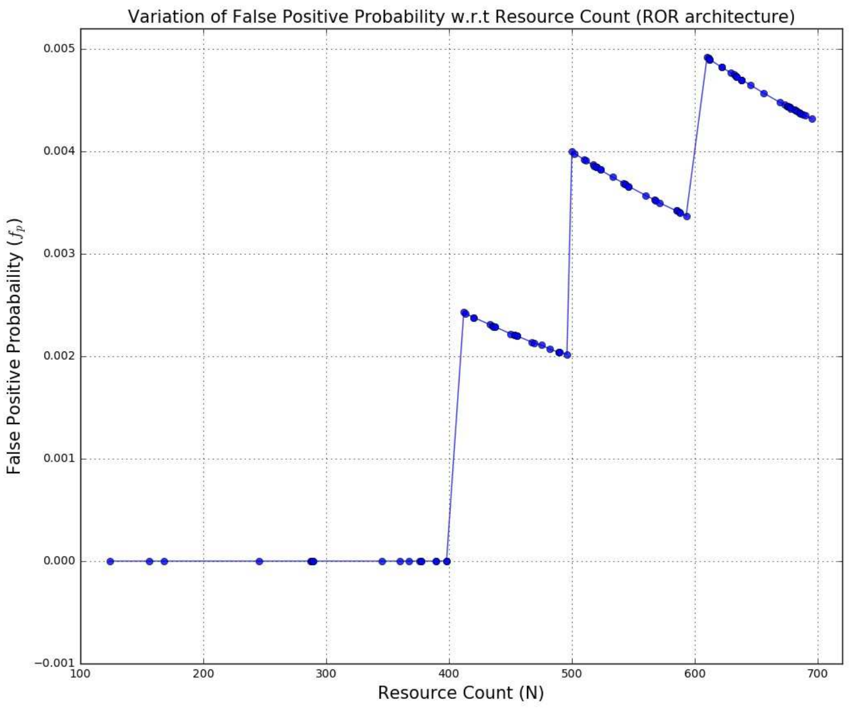


Figure 6.19. Variation of False Positive Probability (f_p) with Resource Count

of resources are able to pass the resultant Bloom-filter hence causing the final set obtained from simulation to grow.

If the Table 6.9, is looked closely it can be observed that ΔN remains constant for a particular range of resource count, both for high value as well as low value within that range. This causes the false positive probability (f_p) to go down within a range, as shown in Figure 6.19. Since ΔN remains constant, f_p reduces within a particular range.

TABLE 6.10. Multi-Attribute Queries for Testing Caching and Overlapped Ring architectures

	Q_1	Q_2
$1mLd$	$[253.55 \sim 0.25]$	$[91.32 \sim 0.09]$
$5mLd$	$[226.18 \sim 0.17]$	$[78.09 \sim 0.04]$
$15mLd$	$[157.63 \sim 0.2]$	$[75.36 \sim 0.07]$
$DSize$	$[1825.89 \sim 63.1188]$	$[909.01 \sim 65.4566]$
$DFree$	$[1790.6 \sim 10.9158]$	$[906.543 \sim 15.5237]$
T_X	$[4915.0 \sim 0.0]$	$[24369.0 \sim 0.0]$
R_X	$[5671.0 \sim 0.0]$	$[16126.0 \sim 0.0]$

So, it can be inferred from Figure 6.19 that the f_p decreases with the increase in resource count with a range.

6.2. Caching and Overlapped Architecture Simulation

This section presents the results and simulation of Caching Methodology and Overlapped Ring architectures.

6.2.1. Query Formation: Example Queries (Q), Q_1, Q_2 are generated with ranges as mentioned in the Table 6.10 using the same methodology as in Multi-Ring Architecture. These queries Q are generated at random, fed to the Q_{node} which propagates it to the test bed resolves the query along with the Q_{node} and send the reply back to the user. The f_p values are calculated for each of the queries for different values of k and m/n ratio. The value of n is the count of IP addresses for each and every attribute. The value of m is kept constant (for performing bit-wise AND).

6.2.2. Simulation Environment: The number of resources, node count in the network, and the number of correlated attributes are as mentioned in the Table 6.11

The variation of f_p for different values of m, for query Q_1 is as shown in Figure 6.20 and for Q_2 is as shown in Figure 6.21. From each of the f_p values, with respective values of k and m/n, the optimum value chosen for k is 4 and the m/n ratio is chosen to be 10. Each

Table 6.11. Test Environment Specifications for Caching and Overlapped Ring architecture

Num. of correlated attributes	7
Num. of Resources	700
Num. of Machines in test bed	[6, 9, 12, 15, 18, 21]

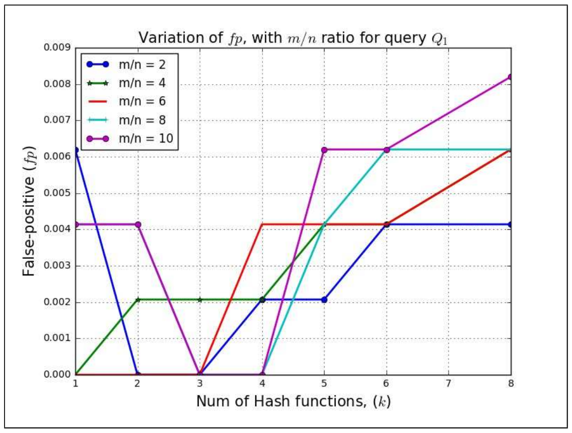


Figure 6.20. Variation of false positive for query Q_1 for Caching and Overlapped Architectures

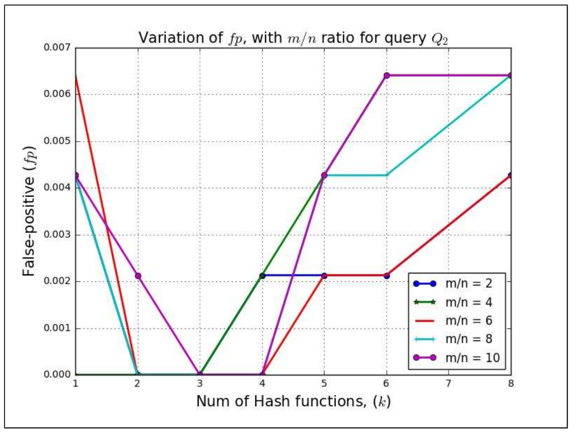


Figure 6.21. Variation of false positive for query Q_2 for Caching and Overlapped Architectures

of the queries are then tested varying scales of network and the performance is evaluated for this particular value of k and m/n.

The source code for generating queries for Caching and Overlapped Ring architecture is presented in Appendix-D. The total number of queries generated to test the architecture is 100, and the total run time is approximately 125 minutes each.

6.2.3. Results and Analysis: The number of nodes in the test-bed are then increased from 6 to 21. For each configuration, average hop count and delay are measured. The average hop count for each configuration is mentioned in Table 6.12. Here also the hop-count refers to the cumulative number of hops it took to get a query resolved, similar to what is described for the ROR architecture. The plot for variation in hop count is as shown in Figure 6.22.

Table 6.12. Variation of avg. Hop Count for Caching and Overlapped Ring Architectures

Num. of Nodes	$Hop - Count_{Caching}$	$Hop - Count_{OverlappedRing}$
6	30	31
9	31	34
12	35	36
15	40	39
18	41	42
21	42	43

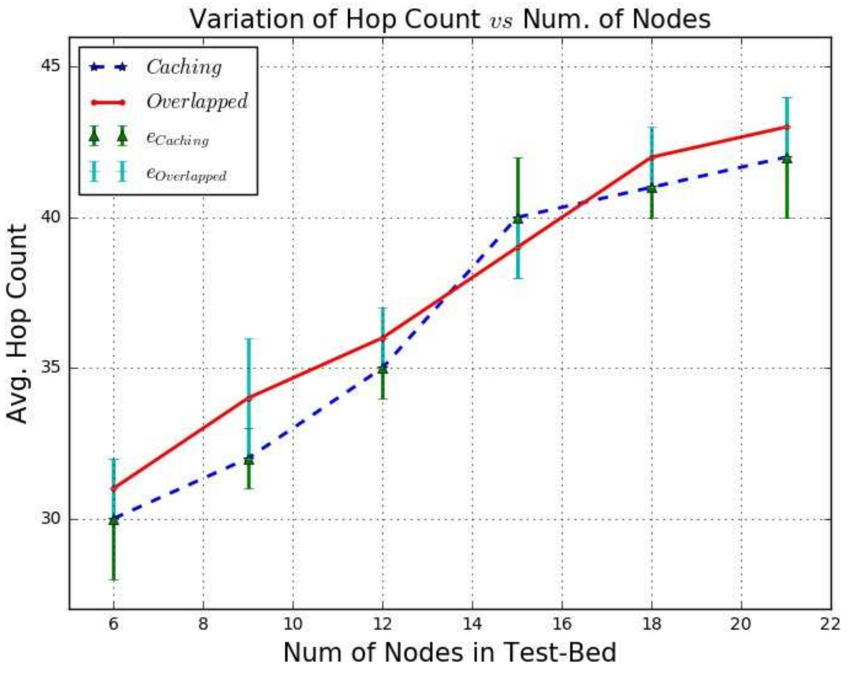


Figure 6.22. Variation of avg. Hop Count with Num. of Nodes for Caching and Overlapped Architecture

The hop-count increases with the increase in the number of nodes present in the network, but due to the use of Chord protocol in MR and SR the hop-counts are logarithmically bounded. The range of hop-counts for Caching and Overlapped ring architecture for each configuration is as mentioned in Table 6.13.

The variation of average delay (in msec) is as mentioned in Table 6.14. The plot for average delay for both the architectures is as shown in Figure 6.23.

TABLE 6.13. Variation of range in avg. Hop Count for Caching and Overlapped Ring Architectures

Num. of Nodes	$Hop - Count_{Caching}$	$Hop - Count_{OverlappedRing}$
6	[32, 28]	[32, 30]
9	[33, 31]	[36, 32]
12	[34, 36]	[37, 35]
15	[42, 38]	[40, 38]
18	[42, 40]	[43, 41]
21	[44, 40]	[44, 42]

TABLE 6.14. Variation of avg. Delay (msec) for Caching and Overlapped Ring Architectures

Num. of Nodes	$Delay_{Caching}(msec)$	$Delay_{OverlappedRing}(msec)$
6	197.223	218.512
9	217.336	233.125
12	221.005	240.0
15	240.128	255.36
18	258.748	263.158
21	278.158	280.956

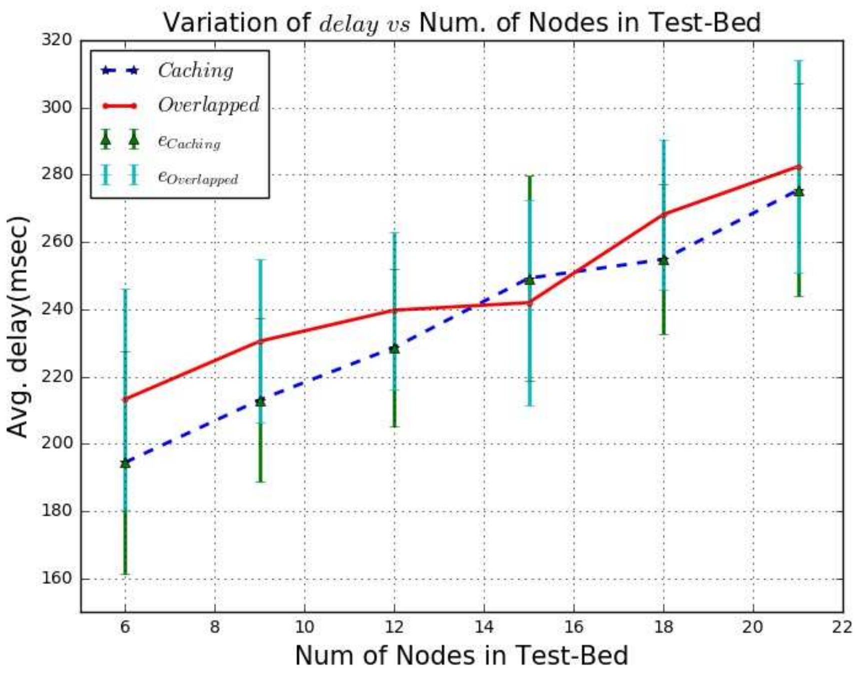


FIGURE 6.23. Variation of avg. Delay in response with Num. of Nodes for Caching and Overlapped Ring Architectures

Table 6.15. Variation of avg. Message Size (bytes) for Caching and Overlapped Ring Architectures

Num. of Nodes	$Msg.Size_{Caching}(bytes)$	$Msg.Size_{OverlappedRing}(bytes)$
6	100675	100680
9	98786	98868
12	100983	100965
15	101348	101485
18	100786	101158
21	103345	103348

It can be noticed that there is a linear increase in the average hop count and an exponential increase in the average delay of response for each of the architectures with increase in the number of nodes in the test bed. Since the bitspace in each SR is divided proportionately for the entire range of each individual attribute so the hop count increases linearly. Also the hop count for the *overlapped* ring is slightly more, than that of the *caching* technique, because the logarithmic mapping from one attribute to the others do not satisfy the entire range exactly in one hop, so it became necessary to have one more hop. It is noted that the *delay* in response for *overlapped* is less than that of *caching*, since the query for each individual value of the attributes are resolved at the same time.
The average message sizes (in bytes) exchanged to and fro from nodes within the network are as mentioned in Table 6.15. The variation is as shown in Figure 6.24.

The average load per node (*lpn*) for both architectures are as mentioned in Table 6.16 and Figure 6.25 shows the load distribution for both architectures. It can be noted that with the addition of every single node in both architectures, the load per node reduces. This indicates the distributed feature of the architectures, the load for query resolution gets distributed more when the number of nodes increases.

As the network scales up the loading per node decreases, with loading much less in *caching* architecture as compared to *overlapped* rings architecture. This is because in the overlapped

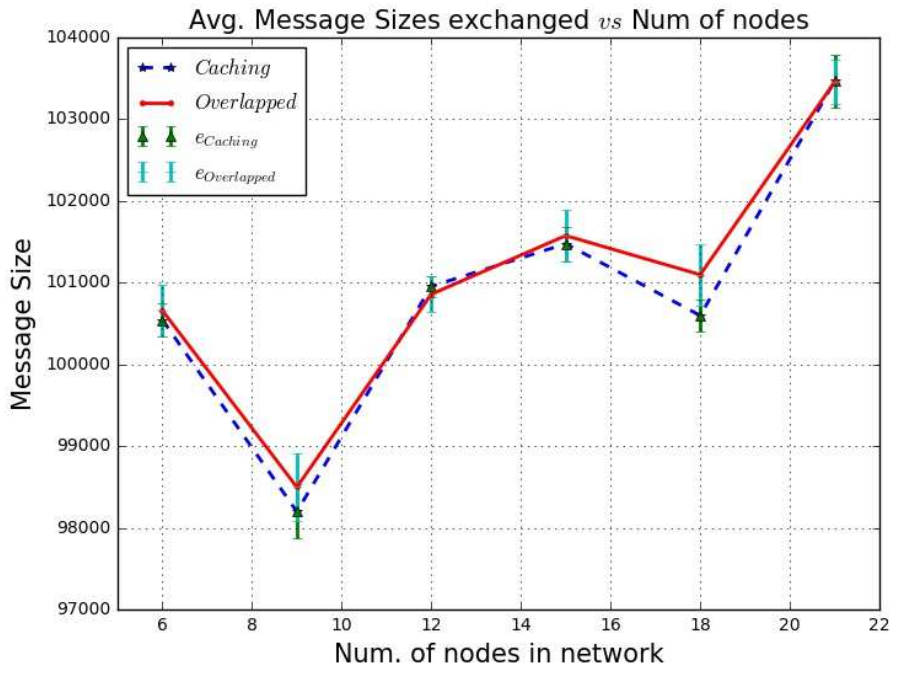


FIGURE 6.24. Variation of avg. Message Size exchanged (bytes) with Num. of Nodes for Caching and Overlapped Ring Architectures

TABLE 6.16. Variation of avg. Load per Node with Num. of Nodes for Caching and Overlapped Ring Architectures

Num. of Nodes	$lpn_{Caching}$	$lpn_{OverlappedRing}$
6	23.56	26.85
9	18.92	22.69
12	15.98	17.01
15	13.25	14.83
18	11.93	13.01
21	9.83	10.00

rings architecture, the values for each attribute are mapped from values of another attribute, which takes comparably more computational time in processing the query.

The difference in resource count obtained from simulation and that of actual count for Caching architecture is as shown in Table 6.17. The difference is less when the number of resources corresponding to a query is less. For example, when the resource count is in the range of $[0, 299]$, $\Delta N_{Caching}$ is 0, (refer Table 6.17) whereas when the resource count is in

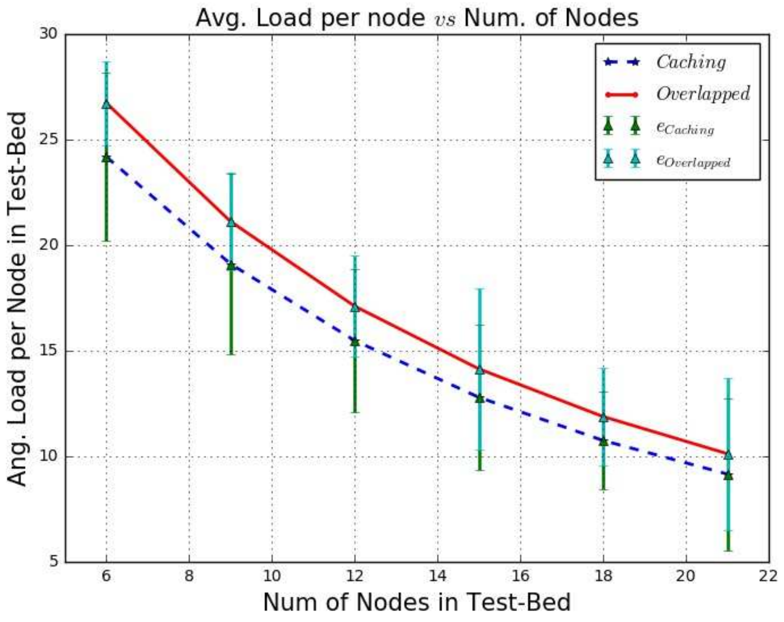


FIGURE 6.25. Variation of avg. Load per Node with Num. of Nodes for Caching and Overlapped Ring Architectures

TABLE 6.17. Variation of difference in resource count with Num. of Resources for Caching Architectures

Range of Resource Count (N)	$\Delta N_{Caching}$
$[0, 299]$	0
$[300, 399]$	1
$[400, 499]$	2
$[500, 599]$	3
$[600, 678]$	3
700	0

the range $[600, 698]$, $\Delta N_{Caching}$ is 3. Here ΔN is the difference in resource count obtained from simulation to that of the actual count. This difference increases with the increase in resource count corresponding to each individual query. The reason being with more number of resources the Bloom-filter gets more and more populated i.e more indices of Bloom-filter are set to 1 from 0. This phenomenon gets reflected in the Resultant Bloom-filter as well. As a result more number of resources are able to pass the resultant Bloom-filter hence causing the final set obtained from simulation to grow.

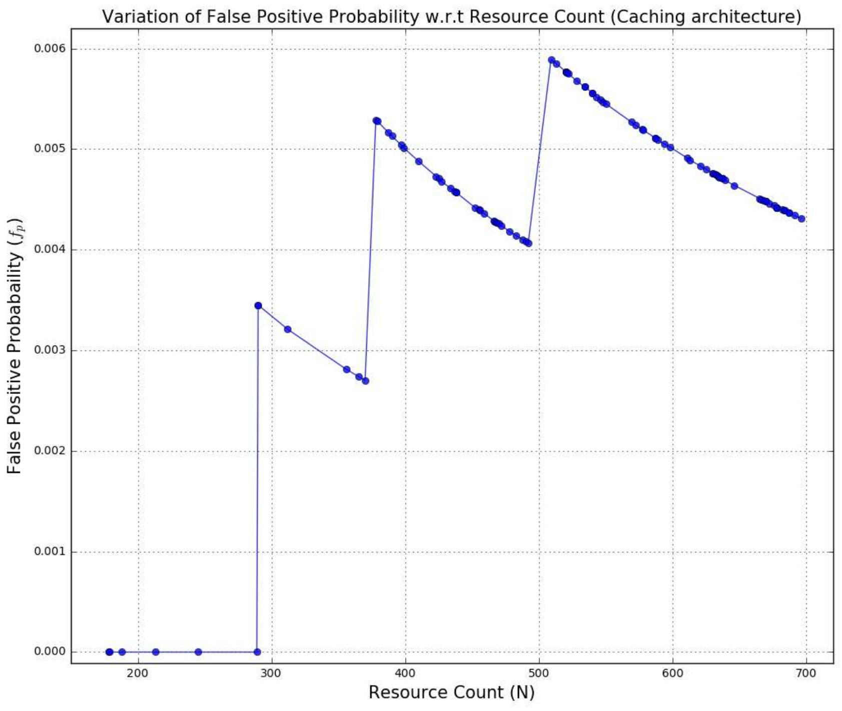


FIGURE 6.26. Variation of False Positive Probability (f_p) with Resource Count (Caching Architecture)

If the Table 6.17, is looked closely it can be observed that $\Delta N_{Caching}$ remains constant for a particular range of resource count, both for high value as well as low value within that range. This causes the false positive probability (f_p) to go down within a range, as shown in Figure 6.26 for the Caching Architecture. Since ΔN remains constant, f_p reduces within a particular range.

So, it can be inferred from Figure 6.26 that the f_p decreases with the increase in resource count with a range. Similarly the $\Delta N_{OverlappedRing}$ values for Overlapped Ring architecture is as described in Table 6.18. Here also the difference increases with the increase in number of resources.

TABLE 6.18. Variation of difference in resource count with Num. of Resources for Overlapped Ring Architectures

Range of Resource Count (N)	$\Delta N_{OverlappedRing}$
$[0, 349]$	0
$[350, 449]$	1
$[450, 549]$	2
$[550, 649]$	3
$[650, 679]$	4
700	0

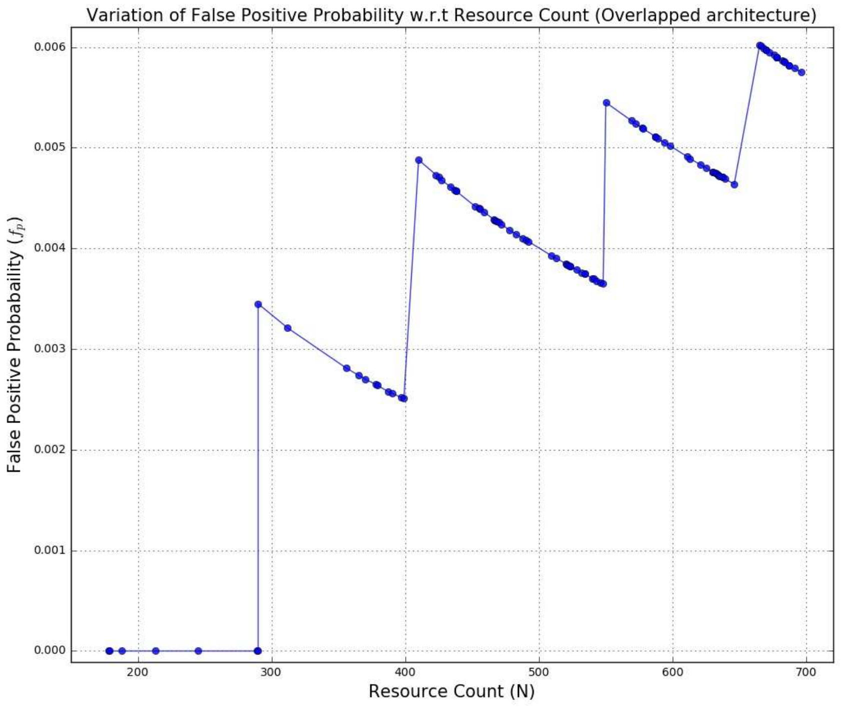


FIGURE 6.27. Variation of False Positive Probability (f_p) with Resource Count (Overlapped Ring Architecture)

The phenomenon of having the difference constant within a range of resource count is also reflected here, as shown in Figure 6.27. It causes the false positive probability f_p to decrease within a range as well.

Table 6.19. Variation of avg. Hop Count for all architectures for correlated attributes

Nodes per SR	$Hops_{Caching}$	$Hops_{OverlappedRing}$	$Hops_{BF}$	$Hops_{Conventional}$
2	30	31	73	74
3	32	34	98	98
4	35	36	119	119
5	40	41	129	128
6	41	42	131	130
7	42	43	135	135

6.3. Comparison of Caching and Overlapped Architectures with Multi-Ring Architecture

This subsection presents a comparative analysis of avg. hop-count, avg. delay, and avg. load per node for query resolution of correlated attributes with all the architectures discussed namely, conventional ROR, RORs using bloom filters, Caching Methodology and Overlapped Ring architecture. The variation of these results for each individual architecture are obtained by keeping the number of nodes same in each of the sub-rings for the architectures and then varying the number of nodes per rings for each respectively.

The variation in hop count is as mentioned in Table 6.19. It is noted that the hop count reduces considerably for each configuration. The average reduction in hop count is 58.5% for the Caching architecture and 57.3% for the Overlapped Ring architecture as compared to the ROR architecture using Bloom-filter messages. The reduction takes places since the queries for each individual attribute are solved on the basis of correlation. The queries don't have to travel a separate SR for its resolution. The plot for variation in average hop count is as shown in Figure 6.28

The delay in response for query resolution also get reduced. The variation in delay (in msec) is as mentioned in Table 6.20. The average reduction in delay is 56.63% for Caching Architecture and 52.06% for the Overlapped Ring architecture. The reason behind it remains the same since, the query doesn't travel the other SRs for its resolution.

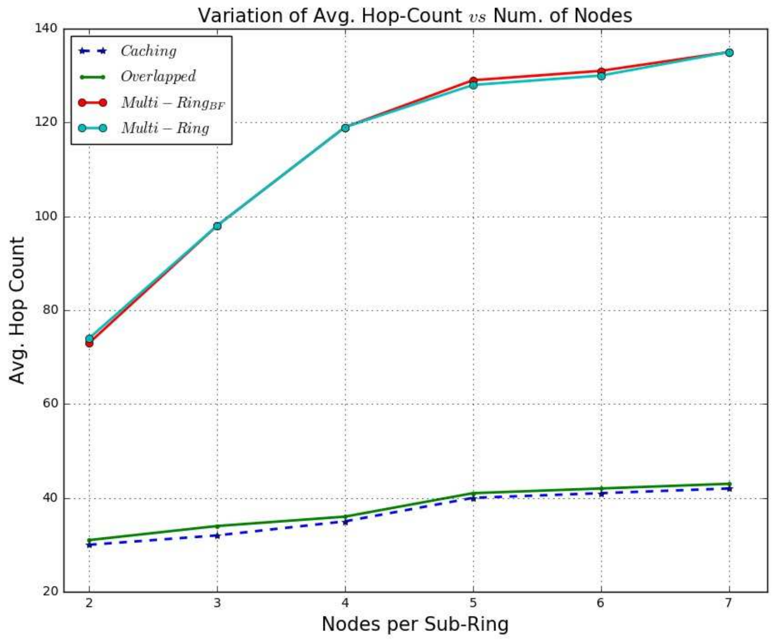


FIGURE 6.28. Variation in avg. Hop Count with Num. of Nodes for all architectures for correlated attributes

TABLE 6.20. Variation of avg. Delay (msec) for all architectures for correlated attributes

Nodes per SR	$Delay_{Caching}$	$Delay_{OverlappedRing}$	$Delay_{BF}$	$Delay_{Conventional}$
2	198.225	219.124	457.118	458.015
3	217.232	233.128	590.225	536.158
4	222.156	240.02	753.080	729.1
5	240.158	255.39	779.936	780.796
6	278.259	273.185	898.936	886.431
7	318.741	302.985	927.363	930.066

The plot for variation in avg. Delay (in msec) is as shown in Figure 6.29

The variation in average message sizes exchanged (in bytes) is as mentioned in Table 6.21. The average message sizes exchanged gets reduced by 10.4% for both Caching architecture and Overlapped Ring architectures as compared to the multi-ring architecture using Bloom-filter. The reduction is caused since the messages from extra SRs are not taken into consideration.

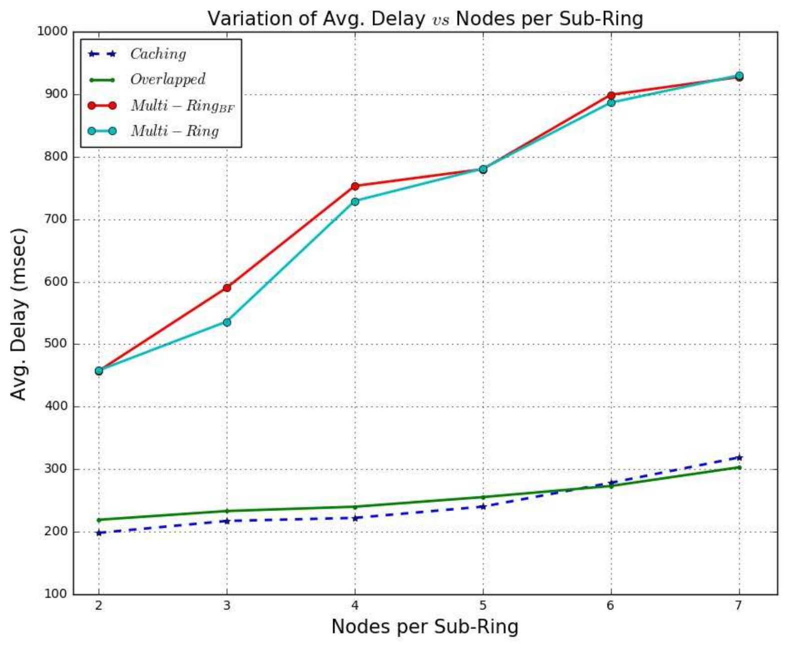


FIGURE 6.29. Variation in avg. Delay with Num. of Nodes for all architectures for correlated attributes

TABLE 6.21. Variation of avg. Message size (bytes) for all architectures for correlated attributes

Nodes per SR	$Msg_{Caching}$	$Msg_{OverlappedRing}$	Msg_{BF}	$Msg_{Conventional}$
2	100675	100681	112356	146063
3	98786	98856	121162	158723
4	100984	100958	123222	163886
5	101346	101483	121112	157446
6	100780	101156	123859	161017
7	103344	103347	123085	161242

The plot for variation in average message size (bytes) exchanged is as shown in Figure 6.30

The variation in load per node (lpn) is as mentioned in Table 6.22. The lpn for the Caching and Overlapped Ring Architectures are higher than both the ROR architectures, with and without Bloom-filters. The reason being, the nodes for resolution of queries in the overlapped and caching architectures have to resolve multiple attributes at the same time. Also with same number of nodes in the SRs, the number responsible for the resolution of

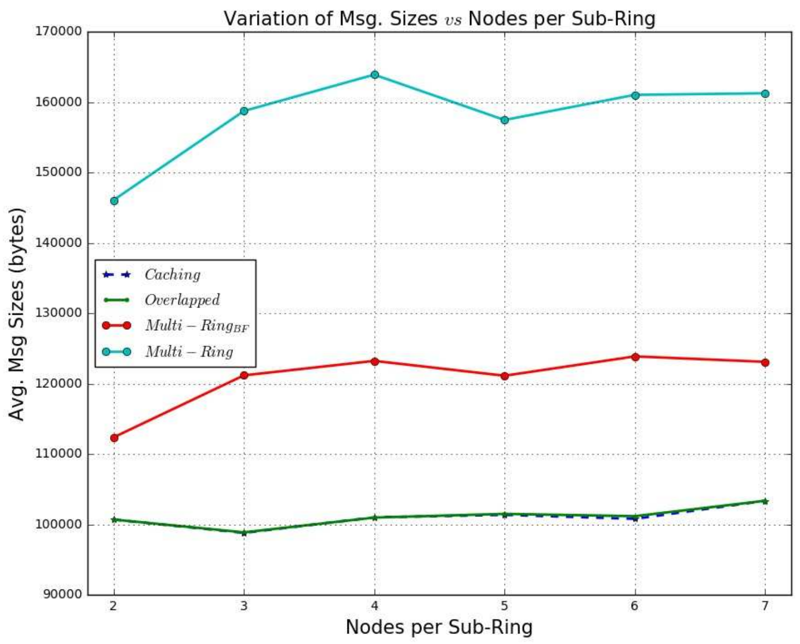


FIGURE 6.30. Variation in avg. Msg. Size exchanged with Num. of Nodes for all architectures for correlated attributes

TABLE 6.22. Variation of avg. Load per node for all architectures for correlated attributes

Nodes per SR	$lpn_{Caching}$	$lpn_{OverlappedRing}$	lpn_{BF}	$lpn_{Conventional}$
2	124.73	135.451	88.451	88.98
3	119.79	125.314	85.314	86.14
4	114.98	116.746	81.746	82.20
5	103.59	105.605	78.605	79.09
6	101.68	103.046	73.046	74.108
7	98.99	99.896	72.896	73.806

queries for each attribute is high in multi-ring architectures than the overlapped and caching architectures. Having more number of nodes greatly distributes the load amongst themselves while query resolution than resolving the same query with less number of nodes.

The average lpn for Caching architecture is 29% higher than the multi-ring architecture (using Bloom-filter), and that of Overlapped Ring is 34% higher than the multi-ring architecture for the same configuration. The plot for variation in load per node is as shown in Figure 6.31.

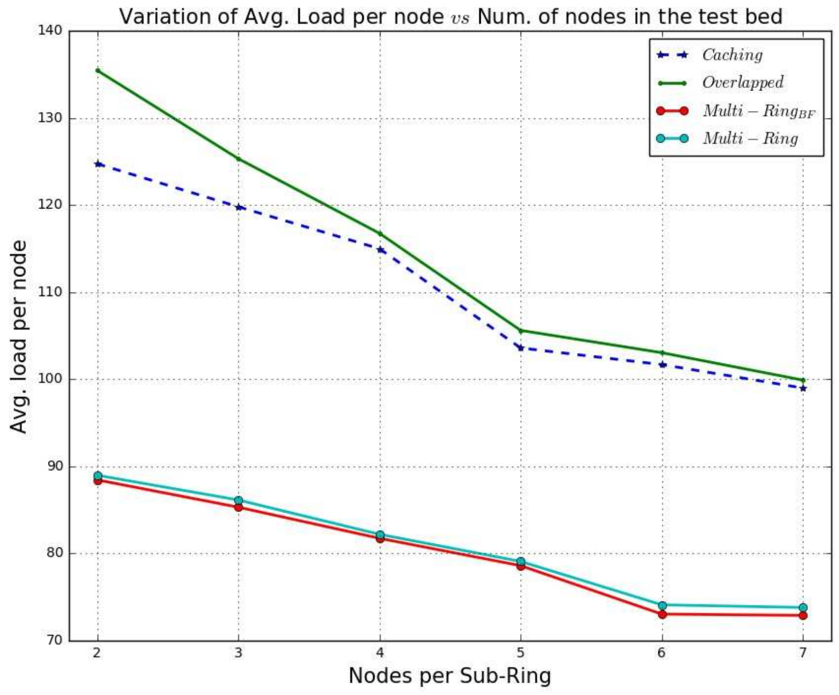


FIGURE 6.31. Variation in avg. Load per node with Num. of Nodes for all architectures for correlated attributes

TABLE 6.23. Variation of difference in resource count with Num. of Resources for ROR Architecture (Correlated Attributes)

Range of Resource Count (N)	ΔN_{ROR}
$[0, 399]$	0
$[400, 499]$	1
$[500, 599]$	2
$[600, 698]$	3
700	0

The difference in resource count for ROR architecture (the one obtained from simulation and the actual count) for correlated attributes is mentioned in Table 6.23. The difference (ΔN_{ROR}), increases with the increase in number of resources.

The false positive probability (f_p), increases with the increase in count of resources, but within a range f_p decreases, since the number resources corresponding to that query

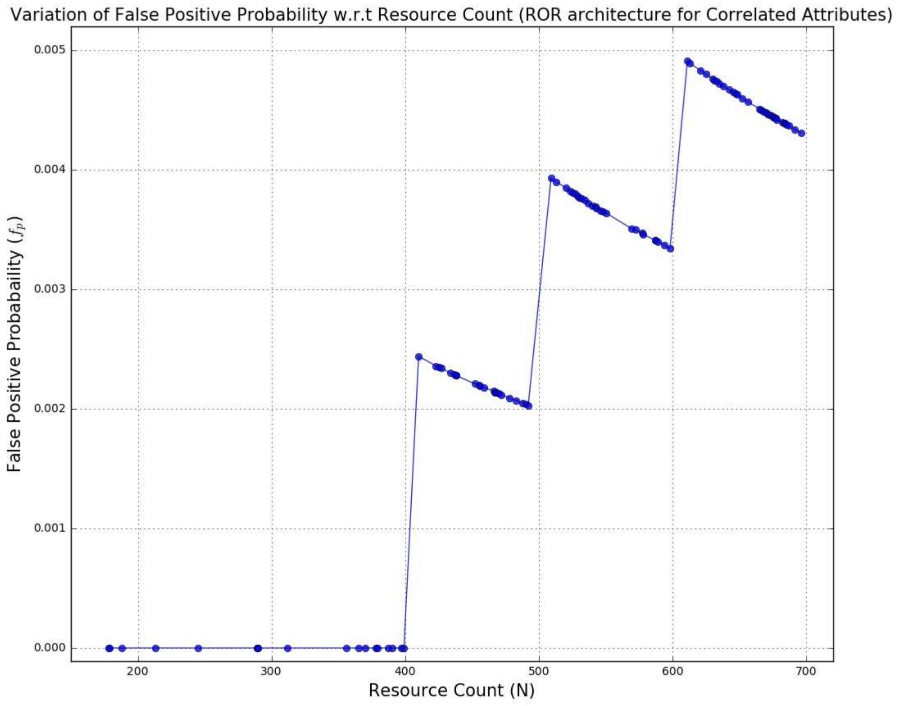


FIGURE 6.32. Variation of False Positive Probability (f_p) with Resource Count (ROR Architecture for Correlated attributes)

increases as shown in Figure 6.32. So, with the increase in resources, f_p value increases but stays within a limit for a specific range.

So, Caching and Overlapped ring architectures provides better optimized solution for multi-attribute query resolution, in terms latency of response, and communication cost. Only drawback being the *lpn* to be high for these architectures. It is therefore recommended to use high performance and high configuration machines when using these kind of architectures for the resolution of same set of queries.

CHAPTER 7

Conclusion and Future Work

This chapter deals with the summary and conclusion of the work that has been done related to Multi-Attribute Query resolution using Structured Peer-to-Peer Networks, using Bloom-filter messages and having *ROR* architecture. The attributes chosen here are those of *PlanetLab* Nodes. The resources are generated using a simulation tool "ResQue". Each of the architectures namely ROR, Caching and Overlapped Rings are discussed in detail. The query generated for each of the architectures are also discussed, and how they are resolved. The results and their analysis follows next related to resolution of multi-attributes as well as correlated attributes for each architecture. The later part of the chapter deals with the future work which can be built on top of this methodology so as to achieve more deterministic set of approaches for resolving multi-attribute queries. The trade-offs related to the use of the Bloom-filter can also be minimized by having more accurate values of the size of Bloom-filters and the number of hash function for a countable set of elements.

7.1. Summary and Conclusion

A multi-attribute query resolution methodology is proposed, discussed and analyzed here using *ROR* structured P2P architecture and using Bloom-filter messages. The approach targets in reducing message overhead, communication cost, and latency in response. The selection of resources for each attribute falling in a query range has a worst case algorithmic time complexity of $\mathcal{O}(n)$ and its lookup takes $\mathcal{O}(N \log N)$ for each individual attribute (N representing number of nodes). The resources represented using Bloom-filters takes $\mathcal{O}(n)$. The Bloom-filter operation has a worst case time complexity $\mathcal{O}(m * n)$, where n is the number of attributes and m is the size of Bloom-filter. The significant improvement is obtained because the size of Bloom-filter remains constant. Since these Bloom-filters are carried around at the time of query resolution, the long list of resources gets represented within a small data structure thereby reducing communication cost. It is found that this methodology causes significant reduction in message overhead, communication cost and latency in response. The space complexity using this methodology is $\mathcal{O}(m * n)$, where m is constant and is significantly small as compared to query resolution methodology using conventional ROR architecture (not using Bloom-filter messages) for large scale systems. For resolving the multi-attribute query as a whole considering each individual attribute along-with their range, a resultant Bloom-filter needs to be generated using bitwise-AND operation. The time-complexity for bitwise-AND is $\mathcal{O}(m)$, which is constant for a chosen Bloom-filter size for query resolution. The ideal choice for the number of hash functions and size of Bloom-filter has a trade-off with its parameters namely size, number of hash functions and number of resources represented by the Bloom-filter to attain minimum false positive probability. Using this methodology, the delay in response for query resolution, average load per node, communication cost and message sizes exchanged, to resolve a particular query has been significantly reduced than the conventional ROR architecture, without using Bloom-filter messages. The results obtained for a simulation environment consisting of 700 resources, each with 12 different types of attributes, and the number of nodes varying from 24 to 84 indicates a 30% reduction in communication overhead, 7% reduction in delay in response, and 10% reduction in average load per node. The optimum number of hash functions for the Bloom-filter is 6, and the m/n ratio is kept 10 where m is the size of Bloom-filter and n is the number of resource count. The optimum number of nodes responsible for each attribute is found to be 5. This research shows that a structured ROR P2P architecture with Bloom-filter messages as a novel feature is useful to resolve multi-attribute queries.

For attributes which are correlated, two different architectures for query resolution of correlated multi-attribute are also discussed here, namely *Caching* and *OverlappedRing* architecture. For selecting resources falling in query range *Caching* architecture has presents algorithmic time complexity of $\mathcal{O}(m * n^2)$ for three correlated attributes and $\mathcal{O}(m * n)$ for two correlated attributes. Overall the worst-case time complexity is $\mathcal{O}(m * n^2)$. For resource selection in *OverlappedRing* architecture, the algorithmic time complexity is $\mathcal{O}(n)$ irrespective of the number of correlated attributes queried for to get resolved. In terms of space complexity for each methodology, both caching and overlapped ring has space complexity of $\mathcal{O}(m * n)$. The ideal choice for the number of hash functions and size of Bloom-filter has a trade-off if it needs to be generalized for any type of query resolution to attain minimum value of false positive probability. Using these methodologies the hop count, delay in response for query resolution, communication cost and message message overhead has been significantly reduced than the conventional *ROR* architecture. However, the load per node for each of these kind of architecture being high, the nodes participating in these architectures need machines with high resources values, to resolve queries. The nodes participating in ROR architecture for resolving same set of queries does not require machines with that much high resource values, as compared to former. Significant research efforts are still needed to reduce the time complexity to gain deterministic performance and to enhance key phases of resource aggregation. In case of correlated attributes, for the same set of resource count and number of nodes (as in ROR architecture) the caching and overlapped ring architecture (using Bloom-filters), provides 58.5% and 57.3% reductions in hop-count, 56.63% and 52.06% reductions in delay in response, message sizes gets reduced by 10% and the average load per node gets reduced by 29% and 34% respectively. The comparison is done over the ROR architecture using Bloom-filters. The number of hash functions for Bloom-filters is considered 4, m/n ratio to be 10 and with 4 nodes per attribute in case of co-related attribute resolution. This research shows that *Overlapped* Rings and *Caching* techniques along with Bloom-filter messages in a peer to peer system are useful to resolve correlated multi-attribute queries.

7.2. Future Work

Significant research efforts are still needed to reduce the time complexity, hop count and delay to gain deterministic performance and to enhance key phases of resource aggregation. Also the metrics for Bloom-filter needs to be generalized for any type of query resolution, by looking into the parameters for resources. To get query resolved it is necessary to execute the simulation several times by iterating for multiple values of m and k for various values of n to have the f_p considerably less. This procedure for selection of metrics for Bloom-filter before starting query resolution needs to be implemented with minimum time span. This research shows that a structured *ROR* P2P architecture with Bloom-filters is useful to resolve multi-attribute queries.

Bibliography

[1] Thomas Karagiannis, Andre Broido, Michalis Faloutsos, Kc claffy *Transport Layer Identification of P2P Traffic* Oct' 2004, IMC04, Taormina, Sicily, Italy.

[2] Vivek Vishnumurthy, Paul Francis *A Comparison of Structured and Unstructured P2P Approaches to Heterogeneous Random Peer Selection* 2007 USENIX Annual Technical Conference.

[3] H. M. N. Dilum Bandara, Anura P. Jayasumana, Michael Zink *Radar Networking in Collaborative Adaptive Sensing of Atmosphere: State of the Art and Research Challenges* Dec. 2012 Proc. IEEE Globecom Workshop on Radar and Sonar Networks.

[4] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications* August 27-31, 2001 SIGCOMM01.

[5] Min Cai, Martin Frank, Jinbo Chen, Pedro Szekely *MAAN: A Multi-Attribute Addressable Network for Grid Information Services* 2003 Proceedings of the Fourth International Workshop on Grid Computing.

[6] Tao He, Jun Ni, Alberto M Segre1, Shaowen Wang, Boyd M Knosp *SkipMard: A Multi-attribute Peer-to-Peer Resource Discovery Approach* 2007 Second International Multi-symposium on Computer and Computational Sciences.

[7] H. M. N. Dilum Bandara, Anura P. Jayasumana *On Characteristics and Modeling of P2P Resources with Correlated Static and Dynamic Attributes* Dec. 2011, In Proc. IEEE GLOBECOM '11.

[8] *https://docs.mongodb.com/manual/introduction/*

[9] *http://cassandra.apache.org/*

[10] *https://hbase.apache.org/*

[11] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber *Bigtable: A Distributed Storage System for Structured Data* OSDI'06: Seventh Symposium on Operating System Design and Implementation, Seattle, WA, November, 2006.

[12] Ilya Mironov *Hash functions: Theory, attacks, and applications* Nov. 2005, Microsoft Research, Silicon Valley Campus.

[13] B. H. Bloom *Space/time trade-offs in hash coding with allowable errors* Commun. ACM, 13(7):422426, 1970.

[14] A. Z. Broder, M. Mitzenmacher *Network applications of bloom filters: A survey.* Internet Mathematics, 1:485509, January 2004.

[15] Mark C. Jeffrey, J. Gregory Steffan *Understanding Bloom Filter Intersection for Lazy Address-Set Disambiguation* SPAA11, June 46, 2011, San Jose, California, USA.

[16] Haiying Shen, Amy Apon, Cheng-Zhong Xu *LORM: Supporting Low-Overhead P2P-based Range-Query and Multi-Attribute Resource Management in Grids* 2007 IEEE.

[17] Prasanna Ganesan, Beverly Yang, Hector Garcia-Molina *One Torus to Rule them All: Multi-dimensional Queries in P2P Systems* June 17-18, 2004 Seventh International Workshop on the Web and Databases (WebDB 2004).

[18] Paolo Costa, Jeff Napper, Guillaume Pierre, Maarten van Steen *Autonomous Resource Selection for Decentralized Utility Computing.*

[19] H. Marais and K. Bharat. *Supporting Cooperative and Personal Surfing with a Desktop Assistant* In Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology, pp. 129138. New York:ACM Press, 1997.

[20] J. Byers, J. Considine, M. Mitzenmacher, and S. Rost. *Informed Content Delivery over Adaptive Overlay Networks* ACM SIGCOMM Computer Communication Review (Proceedings of the 2002 SIGCOMM Conference) 32:4 (2002), 4760.

[21] J. Byers, J. Considine, and M. Mitzenmacher. *Fast Approximate Reconciliation of Set Differences* Boston University Technical Report 2002-019, July 2002.

[22] P. Reynolds and A. Vahdat. *Efficient Peer-to-Peer Keyword Searching* In Middleware 2003: ACM/IFIP/USENIX International Middleware Conference, Rio de Janeiro, Brazil, June 1620, 2003, Proceedings, Lecture Notes in Computer Science 2672, pp. 2140. New York: Springer, 2003.

[23] https://www.planet-lab.org/

[24] E. Heien, D. Kondo, and D. P. Anderson *Correlated resource models of Internet end hosts* In Proc. 31st Int. Conf. on Distributed Computing Systems (ICDCS 11), June 2011.

[25] CMPRG Correlated Multi-attribute P2P Resource Generator, available: *http://www.cnrl.colostate.edu/Projects/CP2P/*.

[26] J. C. Strelen *Tools for dependent simulation input with copulas* In Proc. 2nd Int. Conf. on Simulation Tools and Techniques, Mar. 2009.

APPENDIX A

Multi-Attribute Formation Source Code

This chapter deals presents the source code for the simulations of the Multi-Ring Architecture for Multi-Attribute Query Resolution. The source code presented is written in Python 2.7. Package used are *numpy*, *pandas*.

User Instructions:

Copy the src code in a file named "controller.py", the parts mentioned later respectively as "work.py", "workMR.py", "workSR.py", "bloomfilter.py", and "bitwiseand.py". The "parameterCal.py", "locationAssgn.py" is mentioned in Appendix-E. The contents of each of the files mentioned above are presented in each respective sections. All the src code files are to be put in the same working directory along with a resource list generated from "ResQue". The methodology of generating resource list is mentioned in [25].

Executions:

To execute the src code, only the controller src code needs to be executed, every thing else comes to play on its own, it's been automated. To execute:

```
sudo chmod +x controller.py
sudo ./controller.py -f <Multiattribute - filename> -n <num of Nodes in network>
   -m <Num of bits Mainring> -s <Num of bits Sub ring> -o <hostname>
```

Before the start of simulation, the Bootstrap server should be running, see the instructions for running Bootstrap server in Appendix - E. The number of nodes in the network should be greater than or equal to the number of attributes mentioned of the resources. "Multiattribute - filename" is the file containing list of resources generated from ResQue. "hostname" refers to the host name of the machine where the simulation is executed. Number of bits for the main ring and number of bits for the sub ring depends upon the user. The simulation makes IPC calls using TCP connections (TCP sockets). There will be log files created for each individual node present in the node, corresponding to each of its PID. These log files are useful for error checking and result analysis. At the end of successful execution the ROR architecture will be formed ready to accept test cases from the user as mentioned in the next appendices.

- **Multi-Attribute Architecture Formation, contents for "controller.py"**

```
# import files
import numpy as np
import pandas as pd
import random
import math
import os
import time
import getopt
import sys
import socket
from parameterCal import calParam
from locationAssgn import locAssgn, workAssgnMainID, workAssgnSubID
from multiprocessing import Pool, Process, Manager, Lock
from work import workAssgn
from workMR import workAssgnMR
from workSR import workAssgnSR
from bloomfilter import query_resolution
from bitwiseand import bitand


buff = 51200          # Recv. Buffer size

processCount = 0

userPort = 10500


os.system("clear")

f = open('controller.log', 'w') # Controller log file

f.close()

f = open('controller.log', 'a')


nAttrb = 0


# Data Structures

IPAddress = []

Attrb = {}

idMainRing = []

idSubRing = []

idMainSubRing = []

numNodes = 0

normalBF = []

countingBF = []

processes = []

ports = []

dataBase = []
```

```
reply_ports = []
bloomFilters = {}


# Client module with send only
def client_without_recv(Msg, port, hostname, f):
    port = int(port)
    ip = str(socket.gethostbyname(hostname))
    time.sleep(5)
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((ip, port))
    sock.send(Msg)
    sock.close()
    return


# Client with both send & recv
def client(msg, port, hostname):
    port = int(port)
    ip  = str(socket.gethostbyname(hostname))
    time.sleep(5)
    global buff
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((ip, port))
    sock.send(msg)
    data = sock.recv(buff)
    sock.close()
    return data


# Spawing of processes for Main Ring and Sub Ring Nodes
def startProcess(bitsMR, bitsSR, bitSpaceMR, bitSpaceSR, l, Attrb, n,
    workAssgnMainID, workAssgnSubID, hostname):
    var = "Process-"+str(n+1)
    print "Starting: " + var
    print "Main PID: " + str(os.getppid())
    print "PID: " + str(os.getpid())
    PID = os.getpid()
    MPID = os.getppid()
    print "-------------------------------------------------"
    var1 = "Process-"+str(n+1)+".log"
    f1 = open(var1, 'w')
    f1.close()
    f1 = open(var1, 'a')
    f1.write("----- Starting Process-"+str(n+1)+" "+
        time.strftime("%c")+'-----\n')
    f1.write(' Parent ID: '+str(MPID)+ '\n')
    f1.write(' PID: '+str(PID)+' \n')
    port = 10000

    f1.write("----- Message Propagation -----\n")
    line = "REG"+" "+str(var)+" "+str(PID)
    length = len(line)
```

```
msg = "00"+str(length)+" "+line

f1.write("----- "+str(var)+" --> "+" Controller ----- \n")

f1.write("Message: " +str(msg) +" \n")


reply = client(msg, port, hostname)

f1.write("----- Controller --> "+str(var)+" ----- \n")

f1.write("Message: " +str(reply) +" \n")

print "----- Controller --> "+str(var)+" -----"

print reply


words = reply.split(" ")

word = words[1]

if (word == "REGOK"):

    w = str(words[2])

    if (w == "MR"):

        attrName = words[5]

        portNum = words[6]

        mainRingID = words[7]

        subRingID = words[8]

        print "I am "+str(var)+" responsible for "+str(attrName)+"

            listening at "+str(portNum)+" MR-ID: "+str(mainRingID)+"

            SR-ID: "+str(subRingID)

        f1.write("-- Work Allocation -- \n")

        f1.write("Process Name: "+str(var)+" Ring Type: "+str(w)+" \n")

        f1.write("Attribute: "+str(attrName)+" \n")
        f1.write("Port Num: "+str(portNum)+" \n")
        f1.write("Main Ring ID: "+str(mainRingID)+" \n")
        f1.write("Sub Ring ID: "+str(subRingID)+" \n")
        wMR = workAssgnMR()
        bsReply = wMR.contactBS(hostname, w, portNum, attrName,
            mainRingID, subRingID, f1, var)
        time.sleep(2)
        wMR.analyzeData(hostname, bsReply, portNum, f1, attrName,
            mainRingID, subRingID, bitSpaceMR, bitsMR, bitSpaceSR,
            bitsSR, Attrb)

    if (w == "SR"):
        attrName = words[5]
        portNum = words[6]
        subRingID = words[7]
        print "I am "+str(var)+" responsible for "+str(attrName)+"
            listening at "+str(portNum)+" SR-ID: "+str(subRingID)
        f1.write("-- Work Allocation -- \n")
        f1.write("Process Name: "+str(var)+" Ring Type: "+str(w)+" \n")
        f1.write("Attribute: "+str(attrName)+" \n")
        f1.write("Port Num: "+str(portNum)+" \n")
        f1.write("Sub Ring ID: "+str(subRingID)+" \n")
        wSR = workAssgnSR()
        bsReply = wSR.contactBS(hostname, w, portNum, attrName,
            subRingID, f1, var)
            time.sleep(2)

            wSR.analyzeData(bsReply, hostname, f1)

            time.sleep(10)

            wSR.contact_MR_host(subRingID, portNum, hostname, attrName,

                bitSpaceSR, bitsSR, f1)

            wSR.contact_SR_host(hostname, subRingID, portNum, attrName,

                bitSpaceSR, bitsSR, Attrb, f1)


    else:

        print "Unknown Reply from Controller..!!"

        f1.write("Unknown Reply from Controller..!!" +" \n")

    return


def update_message(data):

    words = data.split(" ")

    num_of_words = len(words)

    hop_count = int(words[3])

    hop_count = hop_count + 1

    words[3] = str(hop_count)

    k = num_of_words

    line = " "

    while (k > 0):

        if int(k) != num_of_words:

            if str(words[k-1]) != " ":

                line = words[k - 1]+" "+line
        k -= 1
    mod_data = line
    mod_data = mod_data.strip()
    return mod_data

# Main method for the start of simulation
def main(argv):
    num_of_bloomfilters = 0
    num_of_bloomfilters_recv = 0
    iplist_count = 0

    f.write('--- Shibayan: Thesis Multi-Attribute Query Resolution ---\n')
    f.write('--- Controller Log --- '+time.strftime("%c")+' ---\n')
    filename = ''

    try:
        opts, args = getopt.getopt(argv,"hf:n:m:s:o:",["ifile=",
        "nodes=","bitsMain=","bitsSubring=","hostname="])
    except getopt.GetoptError:
        print 'python controller.py -f <Multiattribute - filename> -n
            <num of nodes in network>
         -m <Num of bits Mainring> -s <Num of bits Sub ring> -o
             <hostname>'
        sys.exit(2)
    for opt, arg in opts:
```

```
    if opt == '-h':
        print 'python controller.py -f <Multiattribute - filename> -n
            <num of nodes in network>
         -m <Num of bits Mainring> -s <Num of bits Sub ring> -o
             <hostname>'
        sys.exit()
    elif opt in ("-f", "--ifile"):
        filename = arg
    elif opt in ("-n", "--nodes"):
        nodes = arg
    elif opt in ("-m", "--bitsMain"):
        bM = arg
    elif opt in ("-s", "--bitsSubring"):
        bS = arg
    elif opt in ("-o", "--hostname"):
        hostname = arg

numNodes = int(nodes)
bitsMain = int(bM)
bitsSubring = int(bS)

print "------- Attributes --------"
print "File name: ", filename
print "Main Ring Size: ", (2**(bitsMain))
print "Sub Ring Size: ", (2**(bitsSubring))
print "Number of Nodes: ", numNodes
print "---------------------------"

f.write('-------- Attributes ---------\n')
f.write('Reading from file: '+str(filename)+'\n')
f.write('Main Ring Capacity: '+str(2**(bitsMain))+'\n')
f.write('Sub Ring Capacity: '+str(2**(bitsSubring))+'\n')
f.write('Num of machines in n/w: '+str(numNodes)+'\n')
f.write('-----------------------------\n')

rf = calParam()
nAttrb, ipCount, line_params,_ = rf.readFile(filename, numNodes,
    (2**bitsMain), Attrb, f)
f.write("params: "+str(line_params)+" \n")
if nAttrb == -1:
    exit(0)

bitspaceMain = 2**(bitsMain)
bitspaceSubRing = 2**(bitsSubring)

loc = locAssgn()
idMainRing, idMainSubRing, idSubRing =
    loc.bitspaceAlloc(bitspaceMain, bitspaceSubRing, numNodes, nAttrb,
    f)
```

```
workAssgnMainID, workAssgnSubID = loc.workAllocation(idMainRing,
    idMainSubRing, idSubRing, Attrb, f)
loc.generateIP(ipCount, IPAddress, f)
loc.finalDataset(Attrb, IPAddress, f)

ports = random.sample(range(13000, 30000), numNodes)          # Random
    port selection for each individual node
mrPorts = []
k = 0
for k in range(nAttrb):
    mrPorts.append(ports[k])
print "Main PID: " + str(os.getpid())
f.write('\n-------- Spawing Multiple Processes ----------\n')
f.write("Main PID: " + str(os.getpid())+ ' \n')
lock = Lock()
n = 0

for n in range(numNodes):
    p = Process(target = startProcess, args = (bitsMain, bitsSubring,
        bitspaceMain, bitspaceSubRing, lock, Attrb, n,
        workAssgnMainID, workAssgnSubID, hostname))
    p.start()
    processes.append(p)

w = workAssgn()
```

```
print "Waiting for Machines to initialize...."

f.write("Waiting for network machines to initialize \n")

print "Entering Server Mode..!!, Port: 10000"

print "Waiting for Connections..!!"

f.write("Entering Server Mode; listening at PORT: 10000 \n")


btnd = bitand() # Making object for bitwise and operation


global processCount

send_time = 0

hop_count = 0


sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

ip = str(socket.gethostbyname(hostname))

server_address = (ip, 10000)

sock.bind(server_address)

print "Server Address: "+str(server_address)

sock.listen(numNodes)

while True:

    conn, addr = sock.accept()

    data = conn.recv(buff)

    print "----- "+str(addr)+" --> Controller -----"

    print " Message: "+str(data)

    f.write("----- "+str(addr)+" --> Controller ----- \n")

    f.write("Message: "+str(data)+" \n")
```

```
words = data.split(" ")

word = words[1]

if (word == "REG"):

    portNum = ports[processCount]

    pName = words[2]

    pNum = words[3]

    msg = w.regWorkers(nAttrb, data, portNum, pName, pNum,

        processCount, f, workAssgnMainID, workAssgnSubID, dataBase)

    processCount = processCount + 1

    conn.send(msg)


elif (word == "GET"):

    if str(words[2]) == "LIST":

        msg = str(nAttrb)+" "+str(ipCount)+str(line_params)

        msgL = len(msg)

        Msg = "00"+str(msgL)+" GETOK LIST "+msg

        print Msg

        f.write("----- Controller --> "+str(addr)+" -----\n")

        f.write("Message: "+str(Msg)+" \n")

        conn.send(Msg)


    if str(words[2]) == "IPLIST":

        ip = str(socket.gethostbyname(hostname))

        time.sleep(5)

        msg_to_be_checked = data
```

```
            num_of_words = len(words)
            send_time = words[num_of_words - 1]
            result_num = num_of_words - 5
            num_of_bloomfilters = int(result_num / 3)
            print "I shall be getting "+str(num_of_bloomfilters)+"
                bloomfilters from the clients"
            f.write("I shall be getting "+str(num_of_bloomfilters)+"
                bloomfilters from the clients \n")
            Data = update_message(data)
            print "Forwarding Query to Peers"
            f.write("Forwarding Query to Peers \n")
            l = (len(mrPorts) - 1)
            slInd = random.randint(0, l)
            selectPort = ports[slInd]
            client_without_recv(Data, selectPort, hostname, f)


    elif (word == "NBF"):
        print "Got normal bloom filter from address: "+str(addr)
        num_of_bloomfilters_recv = num_of_bloomfilters_recv + 1
        print "Bloom filters received till now:
            "+str(num_of_bloomfilters_recv)
        f.write("Bloom filters received till now:
            "+str(num_of_bloomfilters_recv)+" \n")
        len_of_words = len(words)
        bloom_client_port = str(words[len_of_words - 2])
        reply_ports.append(bloom_client_port)

        hops = int(words[len_of_words - 1])

        hop_count = hop_count + hops

        btnd.anding(data, reply_ports, hostname, hop_count, Attrb,

            num_of_bloomfilters, num_of_bloomfilters_recv,

            bloomFilters, f)


    elif (word == "EXIT"):

        k = 0

        print "Requesting all sub-processes to exit !"

        f.write("Requesting all sub-processes to exit ! \n")

        for k in range(len(ports)):

            target_port = int(ports[k])

            client_without_recv(data, target_port, hostname, f)

        print "Stopped all processes"

        print "Exiting simulation environment"

        f.write("Exiting simulation environment \n")

        f.write("Stopping all processes \n")

        time.sleep(2)

        exit(0)


    elif (word == "IPLIST"):

        iplist_count = iplist_count + 1

        print "IPList received till now "+str(iplist_count)

        hops = int(words[2])
```

```
                hop_count = hop_count + hops

                btnd.analyse_ip(msg_to_be_checked, iplist_count,

                    num_of_bloomfilters, send_time, hop_count, Attrb, data,

                    hostname, f)


            else:

                Msg = "UNKNOWN REQUEST..!!"

                MsgL = len(Msg)

                msg = "0"+str(MsgL)+" "+Msg

                f.write("----- Controller --> "+str(addr)+" -----\n")

                f.write("Message: "+str(msg)+" \n")

                conn.send(msg)


        return


if __name__ == "__main__":

    main(sys.argv[1:])
```

- **Work assignment for each individual nodes, contents for “work.py”**

```
import time

#from controller import submemberCount


submemberCount = 0

class workAssgn:
```

```
def regWorkers(self, nAttrb, data, portNum, pName, pNum, processCount, f,
    workAssgnMainID, workAssgnSubID, dataBase):
global submemberCount
f.write("----- "+str(pName)+" --> Controller ----- \n")
f.write("Message: "+str(data)+" \n")
#portMain = '0'
#l = len(workAssgnSubID)
#print "Process Count: " +str(processCount)+ " nAttrb: " +str(nAttrb)
time.sleep(2)
#print "Sub Member Count1: ", submemberCount
if processCount < nAttrb:
attr = "Attr"+str(processCount)
Msg = "REGOK MR"+" "+pName+" "+str(pNum)+"
    "+str(workAssgnMainID[processCount][0])+" "+str(portNum)+"
    "+str(workAssgnMainID[processCount][1])+"
    "+str(workAssgnMainID[processCount][2])
MsgL = len(Msg)
msg = "00"+str(MsgL)+" "+Msg
f.write("----- Controller --> "+str(pName)+" -----\n")
f.write("Message: "+str(msg)+" \n")
dataBase.append([str(workAssgnMainID[processCount][0]), str(portNum)])
if processCount >= nAttrb:
#print "Sub Member Count2: ", submemberCount
attr = str(workAssgnSubID[submemberCount][1])
'''
```

```
k = len(dataBase)

i = 0

for i in range(k):

if attr == dataBase[i][0]:

portMain = str(dataBase[i][1])

'''

Msg = "REGOK SR"+" "+pName+" "+str(pNum)+" "+attr+" "+str(portNum)+"
    "+str(workAssgnSubID[submemberCount][0])

MsgL = len(Msg)

msg = "00"+str(MsgL)+" "+Msg

f.write("----- Controller --> "+str(pName)+" -----\n")

f.write("Message: "+str(msg)+" \n")

submemberCount = submemberCount + 1

return msg
```

- **Working of Main-Ring Nodes, contents for "workMR.py"**

```
import socket

import time

import random

from multiprocessing import Process

from bloomfilter import query_resolution


bsPort = 11000

conPort = 10000

buff = 5120000
```

```
MembersMR = []

RoutingTableMR = []

FingerTableSR = []

query = []

processes_px = []

normalBF = []


qr = query_resolution()


def client_without_recv(Msg, port, hostname, f1):

global buff

port = int(port)

ip = str(socket.gethostbyname(hostname))

time.sleep(5)

# Creating TCP Socket

global buff

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

sock.connect((ip, port))

sock.send(Msg)

sock.close()

return


def sub_process(k, mod_msg, forwarding_port, Attrb, hostname, indx,

    FingerTableSR, attrN, portNum, set_val, hop_count, f1):

global selected_IP
```

```
f1.write("Entering subprocess \n")

print "Entering Sub Process"

k = int(k)

if k == 0:

print "Sending parsed out message to other members.."

f1.write("Sending parsed out message to other members.. \n")

f1.write("Modified message: "+str(mod_msg)+" \n")

f1.write("Forwarding Port: "+str(forwarding_port)+" \n")

time.sleep(10)

client_without_recv(mod_msg, forwarding_port, hostname, f1)

return


elif k == 1:

f1.write("Processing data \n")

print "Processing data: "

u_limit = query[indx][1]

l_limit = query[indx][2]

f1.write("u_limit: "+str(u_limit)+" \n")

f1.write("l_limit: "+str(l_limit)+" \n")

print "u_limit: "+str(u_limit)

print "l_limit: "+str(l_limit)

normalBF = qr.make_bloom_filter(Attrb, attrN, u_limit, l_limit, f1)

sending_updated_bloomfilter_to_controller(normalBF, hostname, portNum,
    set_val, hop_count, attrN, f1)

return
```

```
else:

print "Unknown process condition .."

print "Doing nothing .."

return


def update_message(data):

words = data.split(" ")

num_of_words = len(words)

hop_count = int(words[3])

hop_count = hop_count + 1

words[3] = str(hop_count)

k = num_of_words

line = " "

while (k > 0):

if int(k) != int(num_of_words - 1):

if str(words[k-1]) != " ":

line = words[k - 1]+" "+line

k -= 1

mod_data = line

return mod_data


def sending_updated_bloomfilter_to_controller(normalBF, hostname,
    portNum, set_val, hop_count, attrN, f1):
```

```
f1.write("Sending updated normal & counting bloomfilter to controller..
    \n")
print "Sending updated normal & counting bloomfilter to controller.."
k = 0
chkSum_normalBF = 0
for k in range(len(normalBF)):
chkSum_normalBF = chkSum_normalBF + normalBF[k]
if int(set_val) == 0:
print "Size of updated normal bloomfilter: "+str(len(normalBF))+" Check
    Sum: "+str(chkSum_normalBF)
f1.write("Size of updated normal bloomfilter: "+str(len(normalBF))+"
    Check Sum: "+str(chkSum_normalBF)+" \n")
msg = "NBF "+str(attrN)+" "+str(normalBF)+" "+str(chkSum_normalBF)+"
    "+str(portNum)+" 1"
msgL = len(msg)
Msg = str(msgL)+" "+str(msg)
time.sleep(2)
#msg_updated = update_message(Msg)
client_without_recv(Msg, conPort, hostname, f1)


if int(set_val) == 1:
print "Size of updated normal bloomfilter: "+str(len(normalBF))+" Check
    Sum: "+str(chkSum_normalBF)
f1.write("Size of updated normal bloomfilter: "+str(len(normalBF))+"
    Check Sum: "+str(chkSum_normalBF)+" \n")
```

```
msg = "NBF "+str(attrN)+" "+str(normalBF)+" "+str(chkSum_normalBF)+"
    "+str(portNum)+" "+str(hop_count)
msgL = len(msg)
Msg = str(msgL)+" "+str(msg)
time.sleep(2)
#msg_updated = update_message(Msg)
client_without_recv(Msg, conPort, hostname, f1)
return


class members:


def addmembers(self, name, port, i):
MembersMR.append([name, port, i])
return


def updateRoutingTable(self, mainringID, attrN, pNum, mrID, f1):
print "MR-Attr: "+str(attrN)+" MR-ID: "+str(mrID)+" Port: "+str(pNum)+"
    joining the network..!!"
f1.write("Updating Routing-Table for MR-Attr: "+str(attrN)+ " ID:
    "+str(mrID)+ " Port: "+str(pNum)+" \n")
if mainringID == attrN:
f1.write("Updation failed for "+str(attrN)+" ID "+str(mrID)+" Port:
    "+str(pNum)+" \n")
return -1
```

```
else:

RoutingTableMR.append([attrN, pNum, mrID])

k = 0

for k in range(len(RoutingTableMR)):

f1.write(str(RoutingTableMR[k]) + " \n")

return 1


def makeFingerTable(self, bitsSR, subRingID, hostname, portNum, f1):

f1.write("Making Finger Table on self.. \n")

bitsSR = int(bitsSR)

ip = str(socket.gethostbyname(hostname))

subRingID = int(subRingID)


k = 0

for k in range(bitsSR):

func = (pow(2,(k))+subRingID)

succ = (pow(2,(k+1))+subRingID)

interval = [func, succ]

FingerTableSR.append([interval, subRingID, ip, portNum])


k = None

f1.write("--- My Finger Table --- \n")

for k in FingerTableSR:

f1.write(str(k)+" \n")

return
```

```
def updateFingerTable(self, attrN, pNum, srID, subRingID, f1):

print "Updating finger table for id: "+str(srID)+ " port: "+str(pNum)

f1.write("Updating finger table for id: "+str(srID)+" port: "+str(pNum)+"
    \n")

if int(srID) == int(subRingID):

print "Matching key found ! new ID: "+str(srID)+ " and my id:
    "+str(subRingID)

print "Updation of finger table not possible !"

f1.write("Updation not possible, matching keys new id: "+str(srID)+" and
    m id: "+str(subRingID)+" \n")

return -1

else:

srID = int(srID)

subRingID = int(subRingID)

pNum = int(pNum)

if srID > subRingID:

k = 0

for k in range(len(FingerTableSR)):

first = int(FingerTableSR[k][0][0])

second = int(FingerTableSR[k][0][1])

if srID > first and srID <= second:

idPresent = int(FingerTableSR[k][1])

distNew = abs(srID - subRingID)

distOld = abs(idPresent - subRingID)
```

```
if idPresent == subRingID:
FingerTableSR[k][1] = srID
FingerTableSR[k][3] = pNum


else:
if distNew < distOld:
FingerTableSR[k][1] = srID
FingerTableSR[k][3] = pNum
else:
#print "Updation not possible, memberID present is less than the incoming
    memberID"
print "Updation not possible MemberID present: "+str(subRingID)+"
    Incoming memberID: "+str(srID)
#f1.write("Updation not possible, memberID present is less than the
    incoming memberID")
f1.write("Updation not possible MemberID present: "+str(subRingID)+"
    Incoming memberID: "+str(srID)+" \n")
else:
if srID > first and srID > second:
idPresent = int(FingerTableSR[k][1])
distNew = abs(srID - subRingID)
distOld = abs(idPresent - subRingID)
if idPresent == subRingID:
FingerTableSR[k][1] = srID
FingerTableSR[k][3] = pNum
```

```
else:
if distNew < distOld:
FingerTableSR[k][1] = srID
FingerTableSR[k][3] = pNum
else:
#print "Updation not possible, memberID present is less than the incoming
    memberID"
print "MemberID present: "+str(subRingID)+" Incoming memberID: "+str(srID)
f1.write("Updation not possible, memberID present is less than the
    incoming memberID")
f1.write("MemberID present: "+str(subRingID)+" Incoming memberID:
    "+str(srID)+" \n")
if srID < subRingID:
k = 0
for k in range(len(FingerTableSR)):
first = int(FingerTableSR[k][0][0])
second = int(FingerTableSR[k][0][1])
if srID <= first and srID <= second:
idPresent = int(FingerTableSR[k][1])
distNew = abs(srID - subRingID)
distOld = abs(idPresent - subRingID)
if idPresent == subRingID:
FingerTableSR[k][1] = srID
FingerTableSR[k][3] = pNum
```

```
else:
if distNew > distOld:
FingerTableSR[k][1] = srID
FingerTableSR[k][3] = pNum
else:
print "Updation not possible, memberID present is less than the incoming
    memberID"
print "MemberID present: "+str(subRingID)+" Incoming memberID: "+str(srID)
f1.write("Updation not possible, memberID present is less than the
    incoming memberID")
f1.write("MemberID present: "+str(subRingID)+" Incoming memberID:
    "+str(srID)+" \n")
elif srID > first and srID <= second:
idPresent = int(FingerTableSR[k][1])
distNew = abs(srID - subRingID)
distOld = abs(idPresent - subRingID)
if idPresent == subRingID:
FingerTableSR[k][1] = srID
FingerTableSR[k][3] = pNum
else:
if distNew > distOld:
FingerTableSR[k][1] = srID
FingerTableSR[k][3] = pNum
else:
```

```
print "Updation not possible, memberID present is less than the incoming
   memberID"
print "MemberID present: "+str(subRingID)+" Incoming memberID: "+str(srID)
f1.write("Updation not possible, memberID present is less than the
   incoming memberID")
f1.write("MemberID present: "+str(subRingID)+" Incoming memberID:
   "+str(srID)+" \n")
print "My new Finger Table "
k = None
for k in FingerTableSR:
print k
f1.write("--- My new Finger Table ---\n")
k = None
for k in FingerTableSR:
f1.write(str(k)+" \n")
return 1

m = members()

class work_MR():

def parse_data_and_forward(self, msg, attrN, hostname, Attrb, portNum,
   f1):
global selected_IP
flag = 0
```

```
indx = 0

set_val = 0

f1.write("Query message: "+str(msg)+" \n")

print "Query message: "+str(msg)+" at: "+str(attrN)

msg = msg.strip()

words = msg.split(" ")

l = len(words)

#print "length of words at parse_data_and_forward: ", str(l)

k = 4

while (k <= (l - 1)):

#print "k here: "+str(k)+" word: "+str(words[k])

an = str(words[k])

k += 1

r1 = str(words[k])

k += 1

r2 = str(words[k])

k += 1

query.append([an, r1, r2])


#print "Query: ", query

k = 0

for k in range(len(query)):

if str(query[k][0]) == str(attrN):

print "I got query for my attribute "+str(attrN)

f1.write("I got query for my attribute"+str(attrN)+" \n")
```

```
flag = 1

indx = int(k)


if flag == 1:

if len(query) == 1:

set_val = 1

print "Nothing to forward"

print "Calling bloom filter functions !"

u_limit = query[0][1]

l_limit = query[0][2]

f1.write("Processing data \n")

print "Processing data: "

f1.write("u_limit: "+str(u_limit)+" \n")

f1.write("l_limit: "+str(l_limit)+" \n")

print "u_limit: "+str(u_limit)

print "l_limit: "+str(l_limit)

time.sleep(2)

hop_count = int(words[3])

hop_count = hop_count + 1

normalBF = qr.make_bloom_filter(Attrb, attrN, u_limit, l_limit, f1)

sending_updated_bloomfilter_to_controller(normalBF, hostname, portNum,

    set_val, hop_count, attrN, f1)


else:

k = 0
```

```
line = " "

for k in range(len(query)):

if k != indx:

attr_n = str(query[k][0])

u = str(query[k][1])

l = str(query[k][2])

line = line + str(attr_n)+" "+str(u)+" "+str(l)+" "

attr_main = attr_n


k = 0

for k in range(len(RoutingTableMR)):

if str(attr_main) == str(RoutingTableMR[k][0]):

forwarding_port = str(RoutingTableMR[k][1])

print "Attribute: "+str(attr_main)+" Forwarding Port:

    "+str(forwarding_port)

#f1.write("Outgoing Message : \n")

hop_count = int(words[3])

hop_count = hop_count + 1

words[3] = str(hop_count)

new_msg = "GET IPLIST "+str(hop_count)+line

msgLen = len(new_msg)

mod_message = "00"+str(msgLen)+" "+new_msg

print "Analyzing my query & Sending out the rest of the query..!"

print "Query modified: ", str(mod_message)

#mod_message = update_message(modified_message)
```

```
hop_count = 0

num_p = 2

k = 0

for k in range(num_p):

px = Process(target=sub_process, args=(k, mod_message, forwarding_port,

    Attrb, hostname, indx, FingerTableSR, attrN, portNum, set_val,

    hop_count, f1))

time.sleep(2)

px.start()

processes_px.append(px)

return


else:

print "I don't have any query for my attribute"

print "Picking out the first machine to send query"

f1.write("I don't have any query for my attribute \n")

f1.write("Picking out the first machine to send query \n")

attr_name = query[0][0]

#msg_updated = update_message(msg)

for k in range(len(RoutingTableMR)):

if str(attr_name) == RoutingTableMR[k][0]:

forwarding_port = int(RoutingTableMR[k][1])

client_without_recv(msg, forwarding_port, hostname, f1)

return
```

```
wr = work_MR()


def serverMode(bitsMR, attrName, portNum, hostname, subRingID, Attrb, f1):

global buff

global conPort

global selected_IP

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

ip = str(socket.gethostbyname(hostname))

portNum = int(portNum)

server_address = (ip, portNum)

sock.bind(server_address)

print "Server Address: "+str(server_address)

sock.listen(20)

while True:

conn, addr = sock.accept()

data = conn.recv(buff)

words = data.split(" ")

print "Data at sub process: ", str(data)

word = words[1]

if str(word) == "REG":

word1 = words[2]

if str(word1) == "MR":

f1.write(str(addr)+" --> "+str(attrName)+":"+str(portNum)+" \n")

f1.write(str(data)+" \n")

print str(addr)+" --> "+str(attrName)+":"+str(portNum)
print str(data)

attrN = str(words[3])

pNum = int(words[4])

mrID = int(words[5])

ret = m.updateRoutingTable(bitsMR, attrN, pNum, mrID, f1)

if ret == 1:

msg = "REGOK MR "+str(attrN)+" ID: "+str(mrID)

msgL = len(msg)

Msg = "00"+str(msgL)+" "+str(msg)

f1.write(str(attrName)+":"+str(portNum)+" --> "+str(addr)+" \n")

f1.write(str(Msg)+" \n")

conn.send(Msg)

else:

msg = "REGNOTOK MR "+str(attrN)+" ID: "+str(mrID)

msgL = len(msg)

Msg = "00"+str(msgL)+" "+str(msg)

f1.write(str(attrName)+":"+str(portNum)+" --> "+str(addr)+" \n")

f1.write(str(Msg)+" \n")

conn.send(Msg)


if str(word1) == "SR":

f1.write(str(addr)+" --> "+str(attrName)+":"+str(portNum)+" \n")

f1.write(str(data)+" \n")

print str(addr)+" --> "+str(attrName)+":"+str(portNum)

print str(data)
```

```
attrN = str(words[3])

pNum = int(words[4])

srID = int(words[5])

ret = m.updateFingerTable(attrN, pNum, srID, subRingID, f1)

if ret == 1:

msg = "REGOK SR "+str(attrN)+" ID "+str(srID)

msgL = len(msg)

Msg = "00"+str(msgL)+" "+str(msg)

f1.write(str(attrName)+":"+str(portNum)+" --> "+str(addr)+" \n")

f1.write(str(Msg)+" \n")

conn.send(Msg)

else:

msg = "REGNOTOK MR "+str(attrN)+" ID: "+str(mrID)

msgL = len(msg)

Msg = "00"+str(msgL)+" "+str(msg)

f1.write(str(attrName)+":"+str(portNum)+" --> "+str(addr)+" \n")

f1.write(str(Msg)+" \n")

conn.send(Msg)


elif str(word) == "GET":

word1 = words[2]

if str(word1) == "IPLIST":

wr.parse_data_and_forward(data, attrName, hostname, Attrb, portNum, f1)


elif str(word) == "EXIT":
```

```
print "Exiting simulation environment, at "+str(attrName)+"-MR"

f1.write("Exiting simulation environment, at "+str(attrName)+"-MR \n")

exit(0)


elif str(word) == "RBF":

msg, hopCount = qr.get_ip_list(data, Attrb, attrName, f1)

Msg = "IPLIST "+str(hopCount)+" "+str(msg)

msgL = len(Msg)

Msg_sent = str(msgL)+" "+Msg

Msg_sent = Msg_sent.strip()

print "Final Message to be sent to controller: "

print str(Msg_sent)

f1.write("Final Message to be sent to controller: \n")

f1.write(str(Msg_sent)+" \n")

client_without_recv(Msg_sent, conPort, hostname, f1)


else:

f1.write("Unknown request from: "+str(addr)+" -->

    "+str(attrName)+":"+str(portNum)+" \n")

f1.write(str(data)+" \n")

print "Unknown request from: "+str(addr)+" -->

    "+str(attrName)+":"+str(portNum)

print str(data)

return
```

```
def clientmode(Msg, attrName, portNum, a, p, hostname, f1):

p = int(p)

ip = str(socket.gethostbyname(hostname))

time.sleep(5)

global buff

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

sock.connect((ip, p))

f1.write(str(attrName)+":"+str(portNum)+" --> "+str(a)+":"+str(p)+" \n")

f1.write(str(Msg)+" \n")

sock.send(Msg)

data = sock.recv(buff)

f1.write(str(a)+":"+str(p)+" --> "+str(attrName)+":"+str(portNum)+" \n")

f1.write(str(data)+" \n")

sock.close()

return data


class workAssgnMR:


def contactBS(self, hostname, w, portNum, attrName, mainRingID,

    subRingID, f1, var):

global bsPort

global buff

f1.write("-- "+str(var)+" --> Boostrap Server -- \n")

print str(var)+" --> Boostrap Server"
```

```
msg = "REG"+" "+str(var)+" "+str(w)+" "+str(attrName)+" "+str(portNum)+"
    "+str(mainRingID)+" "+str(subRingID)
msgL = len(msg)
Msg = "00"+str(msgL)+" "+str(msg)
print Msg
f1.write(" "+str(Msg)+" \n")
port = int(bsPort)
ip = str(socket.gethostbyname(hostname))
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((ip, port))
sock.send(Msg)
data = sock.recv(buff)
f1.write("-- Boostrap Server --> "+str(var)+" -- \n")
f1.write(" "+str(data)+" \n")
sock.close()
print "Boostrap Server --> "+str(var)
print data
return data


def analyzeData(self, hostname, msg, portNum, f1, attrName, mainRingID,
    subRingID, bitSpaceMR, bitsMR, bitSpaceSR, bitsSR, Attrb):
words = msg.split(" ")
#id = None
m.makeFingerTable(bitsSR, subRingID, hostname, portNum, f1)
word = words[2]
```

```
if str(word) == "FIRST":

print "I am the first one to join the network"

print "Entering server mode !"

f1.write("--- AttrName: "+str(attrName)+" MR-ID: "+str(mainRingID)+"

    SR-ID: "+str(subRingID)+" -- \n")

f1.write("Entering Server Mode... \n")

serverMode(bitsMR, attrName, portNum, hostname, subRingID, Attrb, f1)


else:

time.sleep(2)

l = len(words)

l = l-1

print "Contacting Peers !"

f1.write("--- AttrName: "+str(attrName)+" MR-ID: "+str(mainRingID)+"

    SR-ID: "+str(subRingID)+" -- \n")

k = 2

while k < l:

name = words[k]

k += 1

port = words[k]

k += 1

id = words[k]

m.addmembers(name, port, id)

k += 1
```

```
f1.write("Contacting Peers ! \n ")

k = 0

for k in range(len(MembersMR)):

msg = "REG MR "+str(attrName)+" "+str(portNum)+" "+str(mainRingID)

msgL = len(msg)

Msg = "00"+str(msgL)+" "+str(msg)

a = str(MembersMR[k][0])

p = int(MembersMR[k][1])

i = int(MembersMR[k][2])

reply = clientmode(Msg, attrName, portNum, a, p, hostname, f1)

words = reply.split(" ")

word = str(words[1])

if word == "REGOK":

m.updateRoutingTable(mainRingID, a, p, i, f1)

else:

print "MR-Attr: "+str(a)+" ID: "+str(i)+ " not reponsive !!"

f1.write("MR-Attr: "+str(a)+" ID: "+str(i)+ " not reponsive !! \n")

serverMode(bitsMR, attrName, portNum, hostname, subRingID, Attrb, f1)

return
```

- **Working of Sub-Ring Nodes, contents for “workSR.py”**

```
import socket

import time



bsPort = 11000
```

```
buff = 1024


myRing = []

srhPort = []

srhID = []


mrhostPort = 0

mrhostID = 0


def clientmode(Msg, port, hostname, attrName, portNum, f1):

ip = str(socket.gethostbyname(hostname))

time.sleep(5)

# Creating TCP Socket

global buff

port = int(port)

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

time.sleep(30)

sock.connect((ip, port))

f1.write(str(attrName)+":"+str(portNum)+" --> "+str(ip)+":"+str(port)+"

    \n")

f1.write(str(Msg)+" \n")

sock.send(Msg)

data = sock.recv(buff)

f1.write(str(ip)+":"+str(port)+" --> "+str(attrName)+":"+str(portNum)+"

    \n")
```

```
f1.write(str(data)+" \n")

sock.close()

return data


def server_sub_ring_members(portNum, hostname, subRingID, Attrb,

    attrname, f1):

print "Entering server mode !"

f1.write("Entering server mode \n")

global buff

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

ip = str(socket.gethostbyname(hostname))

# Binding the socket to the port

portNum = int(portNum)

server_address = (ip, portNum)

sock.bind(server_address)

print "Server Address: "+str(server_address)

# Listening for incoming connections

sock.listen(50)

while True:

conn, addr = sock.accept()

data = conn.recv(buff)

words = data.split(" ")

word = words[1]

if str(word) == "REG":

word1 = words[2]
```

```
if str(word1) == "SR":

attrname = words[3]

idSR = words[4]

port = words[5]

f1.write(str(addr)+" --> "+str(attrname)+":"+str(port)+" \n")

f1.write(str(data)+" \n")

print str(addr)+" --> "+str(attrname)+":"+str(port)

print str(data)

f1.write("Updating finger table for "+str(addr)+" \n")

ret = fing.updateFingerTable(port, idSR, subRingID, f1)

if ret == 1:

msg = "REGOK SR "+str(attrname)+" ID "+str(id)

msgL = len(msg)

Msg = "00"+str(msgL)+" "+str(msg)

print str(attrname)+":"+str(portNum)+" --> "+str(addr)

print str(Msg)

f1.write(str(attrname)+":"+str(portNum)+" --> "+str(addr)+" \n")

f1.write(str(Msg)+" \n")

conn.send(Msg)

else:

print "Unknown REG request from "+str(addr)

f1.write("Unknown REG request from "+str(addr)+" \n")


elif str(word) == "IPLIST":

print "Hold on, on its way"
```

```
elif str(word) == "EXIT":

print "Exiting simulation environment, at "+str(attrname)+"-SR"

f1.write("Exiting simulation environment, at "+str(attrname)+"-SR \n")

exit(0)


else:

f1.write("Unknown request from: "+str(addr)+" -->

    "+str(attrname)+":"+str(port)+" \n")

f1.write(str(data)+" \n")

print "Unknown request from: "+str(addr)+" -->

    "+str(attrname)+":"+str(port)

print str(data)

return


class finger_table():


def makeFingerTable(self, bitsSR, subRingID, hostname, portNum, f1):

f1.write("Making Finger Table on self.. \n")

bitsSR = int(bitsSR)

ip = str(socket.gethostbyname(hostname))

subRingID = int(subRingID)


k = 0

for k in range(bitsSR):
```

```
func = (pow(2,(k))+subRingID)

succ = (pow(2,(k+1))+subRingID)

interval = [func, succ]

myRing.append([interval, subRingID, ip, portNum])

k = None

f1.write("--- My Finger Table --- \n")

for k in myRing:

f1.write(str(k)+" \n")

return


def updateFingerTable(self, pNum, srID, subRingID, f1):

# Updating finger table from here

print "Updating finger table for id: "+str(srID)+ " port: "+str(pNum)

f1.write("Updating finger table for id: "+str(srID)+" port: "+str(pNum)+"
    \n")

if int(srID) == int(subRingID):

print "Matching key found ! new ID: "+str(srID)+ " and my id:
    "+str(subRingID)

print "Updation of finger table not possible !"

f1.write("Updation not possible, matching keys new id: "+str(srID)+" and
    m id: "+str(subRingID)+" \n")

return -1

else:

srID = int(srID)

subRingID = int(subRingID)
```

```
pNum = int(pNum)

if srID > subRingID:

k = 0

for k in range(len(myRing)):

first = int(myRing[k][0][0])

second = int(myRing[k][0][1])

if srID > first and srID <= second:

idPresent = int(myRing[k][1])

distNew = abs(srID - subRingID)

distOld = abs(idPresent - subRingID)

if idPresent == subRingID:

myRing[k][1] = srID

myRing[k][3] = pNum


else:

if distNew < distOld:

myRing[k][1] = srID

myRing[k][3] = pNum

else:

print "Updation not possible, memberID present is less than the incoming

    memberID"

print "MemberID present1: "+str(subRingID)+" Incoming memberID:

    "+str(srID)

f1.write("Updation not possible, memberID present is less than the

    incoming memberID")
```

```
f1.write("MemberID present: "+str(subRingID)+" Incoming memberID:
    "+str(srID)+" \n")
else:
if srID > first and srID > second:
idPresent = int(myRing[k][1])
distNew = abs(srID - subRingID)
distOld = abs(idPresent - subRingID)
if idPresent == subRingID:
myRing[k][1] = srID
myRing[k][3] = pNum


else:
if distNew < distOld:
myRing[k][1] = srID
myRing[k][3] = pNum
else:
print "Updation not possible, memberID present is less than the incoming
    memberID"
print "MemberID present2: "+str(subRingID)+" Incoming memberID:
    "+str(srID)
f1.write("Updation not possible, memberID present is less than the
    incoming memberID")
f1.write("MemberID present: "+str(subRingID)+" Incoming memberID:
    "+str(srID)+" \n")
if srID < subRingID:
```

```
k = 0
for k in range(len(myRing)):
first = int(myRing[k][0][0])
second = int(myRing[k][0][1])
if srID <= first and srID <= second:
idPresent = int(myRing[k][1])
distNew = abs(srID - subRingID)
distOld = abs(idPresent - subRingID)
if idPresent == subRingID:
myRing[k][1] = srID
myRing[k][3] = pNum
else:
if distNew > distOld:
myRing[k][1] = srID
myRing[k][3] = pNum
else:
print "Updation not possible, memberID present is less than the incoming
    memberID"
print "MemberID present: "+str(subRingID)+" Incoming memberID: "+str(srID)
f1.write("Updation not possible, memberID present is less than the
    incoming memberID")
f1.write("MemberID present: "+str(subRingID)+" Incoming memberID:
    "+str(srID)+" \n")
elif srID > first and srID <= second:
idPresent = int(myRing[k][1])
```

```
distNew = abs(srID - subRingID)
distOld = abs(idPresent - subRingID)
if idPresent == subRingID:
myRing[k][1] = srID
myRing[k][3] = pNum
else:
if distNew > distOld:
myRing[k][1] = srID
myRing[k][3] = pNum
else:
print "Updation not possible, memberID present is less than the incoming
    memberID"
print "MemberID present: "+str(subRingID)+" Incoming memberID: "+str(srID)
f1.write("Updation not possible, memberID present is less than the
    incoming memberID")
f1.write("MemberID present: "+str(subRingID)+" Incoming memberID:
    "+str(srID)+" \n")
print "My new Finger Table "
k = None
for k in myRing:
print k
f1.write("--- My new Finger Table ---\n")
k = None
for k in myRing:
f1.write(str(k)+" \n")
```

```
return 1

fing = finger_table()

class workAssgnSR:

def contactBS(self, hostname, w, portNum, attrName, subRingID, f1, var):
global bsPort
global buff

f1.write("-- "+str(var)+" --> Boostrap Server -- \n")
print str(var)+" --> Boostrap Server"
msg = "REG"+" "+str(var)+" "+str(w)+" "+str(attrName)+" "+str(portNum)+"
    "+str(subRingID)
msgL = len(msg)
Msg = "00"+str(msgL)+" "+str(msg)
print Msg
f1.write(" "+str(Msg)+" \n")
port = int(bsPort)
ip = str(socket.gethostbyname(hostname))
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((ip, port))
sock.send(Msg)
data = sock.recv(buff)
f1.write("-- Boostrap Server --> "+str(var)+" -- \n")
```

```
f1.write(" "+str(data)+" \n")

sock.close()

print "Boostrap Server --> "+str(var)

print data

return data


def analyzeData(self, msg, hostname, f1):

global mrhostPort

global mrhostID


f1.write("Parsing msg.. \n")

words = msg.split(" ")

#f1.write("Words length: "+str(len(words))+ " \n")

'''

for word in words:

f1.write("Words : "+str(word)+ " \n")

'''

ip = str(socket.gethostbyname(hostname))

mrhostPort = words[3]

mrhostID = words[4]


f1.write("--- Main Ring host details ---\n")

f1.write("IP Address -> " + str(ip)+":" +str(mrhostPort)+" \n")

f1.write("Main Ring host ID -> "+str(mrhostID)+" \n")

f1.write("------------------------------\n")
```

```
if len(words) > 6:

k = 6

f1.write("--- Sub Ring host details ---\n")

while k < (len(words) - 1):

srhostPort = words[k]


srhPort.append(srhostPort)

k += 1

srhostID = words[k]


srhID.append(srhostID)

k += 2

k = 0

for k in range(len(srhPort)):

f1.write("Sub Ring host -"+str(k+1)+" \n")

f1.write("IP Address -> " + str(ip)+":" +str(srhPort[k])+" \n")

f1.write("Sub Ring host ID -> "+str(srhID[k])+" \n")

return


else:

return


def contact_MR_host(self, subRingID, portNum, hostname, attrName,
    bitSpaceSR, bitsSR, f1):

fing.makeFingerTable(bitsSR, subRingID, hostname, portNum, f1)
```

```
f1.write("Contacting Main Ring Host \n")

msg = "REG SR "+str(attrName)+" "+str(portNum)+" "+str(subRingID)

msgL = len(msg)

Msg = "00"+str(msgL)+" "+str(msg)

time.sleep(10)

reply = clientmode(Msg, mrhostPort, hostname, attrName, portNum, f1)

words = reply.split(" ")

if (str(words[1]) == "REGOK"):

print "Updating finger table for main ring host "

f1.write("Updating finger table for main ring host \n")

fing.updateFingerTable(mrhostPort, mrhostID, subRingID, f1)

else:

f1.write("MR Host not responding \n")

print "MR Host not responding"

return


def contact_SR_host(self, hostname, subRingID, portNum, attrName,

    bitSpaceSR, bitsSR, Attrb, f1):

if srhPort:

f1.write("Contacting Sub Ring members \n")

msg = "REG SR "+str(attrName)+" "+str(subRingID)+" "+str(portNum)

msgL = len(msg)

Msg = "00"+str(msgL)+" "+str(msg)

k = 0

for k in range(len(srhPort)):
```

```
port = srhPort[k]
idSR = srhID[k]
time.sleep(10)
reply = clientmode(Msg, port, hostname, attrName, portNum, f1)
words = reply.split(" ")
if (str(words[1]) == "REGOK"):
print "Updating finger table for sub ring host id:"+str(srhID[k])
f1.write("Updating finger table for sub ring host id:"+str(srhID[k])+"
    \n")
fing.updateFingerTable(idSR, port, subRingID, f1)
else:
print "Updation not possible for sub ring host id:"+str(srhID[k])
f1.write("Updation not possible in finger table for sub ring host
    id:"+str(srhID[k])+" \n")
server_sub_ring_members(portNum, hostname, subRingID, Attrb, attrName, f1)
return
```

APPENDIX B

Caching Architecture Formation Source Code

This chapter presents the source code for Caching Mechanism for the resolution of Co-related Attributes. The src code uses "Python2.7", along with "numpy" and "pandas" packages. The simulation to be started with the execution of "caching.py". The other files necessary are "readfile.py", "workCaching.py", "$bloomfilter_{op}.py$", and resource list obtained from "ResQue". The "readfile.py" is mentioned in Appendix-E To execute the src code:

```
sudo chmod +x caching.py
./caching.py -f <Multiattribute - filename> -n <num of nodes in network> -m <Num
   of bits Mainring> -s <Num of bits Sub ring> -o <hostname>
```

The number of nodes present in the network should be greater than or equal to the number of attributes mentioned in resources list obtained from "ResQue". The contents for each file name is mentioned along with each sections. At the end of successful execution of simulation "Caching" architecture will be formed ready to resolve queries related to correlated attributes.

- **Caching Architecture Formation, contents for "caching.py"**

```
import socket
import numpy as np
import pandas as pd
import random
import math
import os
import time

import getopt

import sys

from readfile import calParam

from multiprocessing import Pool, Process, Manager, Lock

from workCaching import startWork

from bloomfilter_op import BloomFilter_Operation


bf = BloomFilter_Operation()

os.system("clear")

f = open('caching.log', 'w')

f.close()

f = open('caching.log', 'a')


Attrb = {}

corrAttrb = []

processes = []

IPAddress = []

RoutingTable = {}

bloomFilters = {}

ip_List = {}


buff = 51200

normalBF = []

counter = 3
```

```python
def client_without_recv(Msg, port, hostname, f):
    global buff
    port = int(port)
    ip = str(socket.gethostbyname(hostname))
    time.sleep(5)
    # Creating TCP Socket
    global buff
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((ip, port))
    sock.send(Msg)
    sock.close()
    return


def client(msg, port, hostname):
    port = int(port)
    ip  = str(socket.gethostbyname(hostname))
    time.sleep(5)
    global buff
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((ip, port))
    sock.send(msg)
    data = sock.recv(buff)
    sock.close()
    return data
```

```
def startProcess(port, corrAttr, lock, Attrb, n, hostname):

    var = "Process-"+str(n+1)

    print "Starting: " + var

    print "Main PID: " + str(os.getppid())

    print "PID: " + str(os.getpid())

    PID = os.getpid()

    MPID = os.getppid()

    print "-------------------------------------------------"

    conPort = 10000

    var1 = "Process-"+str(n+1)+".log"

    f1 = open(var1, 'w')

    f1.close()

    f1 = open(var1, 'a')

    f1.write("----- Starting Process-"+str(n+1)+" "+

       time.strftime("%c")+'-----\n')

    f1.write(' Parent ID: '+str(MPID)+ '\n')

    f1.write(' PID: '+str(PID)+' \n')

    selAttrb = []

    attrKey = []

    valKey = []

    if corrAttr[0] == "mLd1":

        selAttrb = [str(corrAttr[0]), str(corrAttr[1]), str(corrAttr[2])]
```

```
f1.write("I am reposible for correlation of:
    "+str(selAttrb[0])+";"+str(selAttrb[1])+";"+str(selAttrb[2])+"
    \n")
print "I am reposible for correlation of:
    "+str(selAttrb[0])+";"+str(selAttrb[1])+";"+str(selAttrb[2])
ind = []
Key1 = selAttrb[0]
Key2 = selAttrb[1]
Key3 = selAttrb[2]
Val1 = Attrb[Key1].flatten()
Val2 = Attrb[Key2].flatten()
Val3 = Attrb[Key3].flatten()
i = 0
for i in range(len(Val1)):
    key = Val1[i]
    if (key not in attrKey) and (int(key) != 0):
        j = 0
        ind = []
        for j in range(len(Val1)):
            if Val1[j] == key:
                ind.append(j)
        attrKey.append([key])
        # print "Key: "+str(key)+" Ind: "+str(ind)
        k = 0
        ind1 = []
```

```
            ind2 = []
            ip = []
            for k in range(len(ind)):
                ind1.append(Val2[ind[k]])
                ind2.append(Val3[ind[k]])
                ip.append(IPAddress[ind[k]])
            valKey.append([ind1, ind2, ip])
    k = 0
    f1.write("Selected Attribute Keys - Value pair:\n")
    for k in range(len(attrKey)):
        f1.write(str(attrKey[k])+" -> "+str(valKey[k])+"\n")
    print "Length of Keys: ", len(attrKey)
    f1.write("Length of Keys: " +str(len(attrKey))+" \n")
    print "Length of Values: ", len(valKey)
    f1.write("Length of Values: " +str(len(valKey))+" \n")
else:
    selAttrb = [str(corrAttr[0]), str(corrAttr[1])]
    print "I am reposible for correlation of:
        "+str(selAttrb[0])+";"+str(selAttrb[1])
    f1.write("I am reposible for correlation of:
        "+str(selAttrb[0])+";"+str(selAttrb[1])+" \n")
    ind = []
    Key1 = selAttrb[0]
    Key2 = selAttrb[1]
    Val1 = Attrb[Key1].flatten()
```

```
Val2 = Attrb[Key2].flatten()

i = 0

for i in range(len(Val1)):

    key = Val1[i]

    if (key not in attrKey) and (int(key) != 0):

        j = 0

        ind = []

        for j in range(len(Val1)):

            if Val1[j] == key:

                ind.append(j)

        attrKey.append([key])

        # print "Key: "+str(key)+" Ind: "+str(ind)

        k = 0

        ind1 = []

        ip = []

        for k in range(len(ind)):

            ind1.append(Val2[ind[k]])

            ip.append(IPAddress[ind[k]])

        valKey.append([ind1, ip])

k = 0

f1.write("Selected Attribute Keys - Value pair:\n")

for k in range(len(attrKey)):

    f1.write(str(attrKey[k])+" -> "+str(valKey[k]) +"\n")

print "Length of Keys: ", len(attrKey)

f1.write("Length of Keys: " +str(len(attrKey))+" \n")
```

```
    print "Length of Values: ", len(valKey)

    f1.write("Length of Values: " +str(len(valKey))+" \n")

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

ip = str(socket.gethostbyname(hostname))

port = int(port)

ipList = []

server_address = (ip, port)

sock.bind(server_address)

print "Entering server mode at: ", var

f1.write("Entering server mode at: " +str(var)+" \n")

print "Server Address: "+str(server_address)

f1.write("Server Address: "+str(server_address)+" \n")

sock.listen(20)

sw = startWork()

while True:

    conn, addr = sock.accept()

    data = conn.recv(buff)

    words = data.split(" ")

    print "Received conn from: " +str(addr)+"

        "+str(time.strftime("%c"))

    print "Message: ", data

    f1.write("Received conn from: " +str(addr)+"

        "+str(time.strftime("%c"))+" \n")

    f1.write("Message: "+str(data)+" \n")

    word = words[1]
```

```
if word == "GET":
    if words[2] == "LIST":
        if len(selAttrb) == 3:
            line = str(Key1)+" "+str(max(Val1))+"
                "+str(min(Val1))+" "+str(Key2)+" "+str(max(Val2))+"
                "+str(min(Val2))+" "+str(Key3)+" "+str(max(Val3))+"
                "+str(min(Val3))
        if len(selAttrb) == 2:
            line = str(Key1)+" "+str(max(Val1))+"
                "+str(min(Val1))+" "+str(Key2)+" "+str(max(Val2))+"
                "+str(min(Val2))
        conn.send(line)
if word == "IP":
    msg, ipList = sw.getIPList(selAttrb, data, attrKey, valKey, f1)
    msgL = len(msg)
    Msg = "00"+str(msgL)+" BF "+str(msg)
    # print "Generated Bloom filter!"
    client_without_recv(Msg, conPort, hostname, f1)
if word == "RBF":
    msg = sw.select_IP_Address(ipList, data, f1)
    msgL = len(msg)
    Msg = "00"+str(msgL)+" IP "+str(msg)
    client_without_recv(Msg, conPort, hostname, f1)
if word == "EXIT":
    print "Exiting Myself: "+str(var)
```

```
            exit(0)

    return


def generateIPAddress(ipCount):

    k = 0

    for k in range(ipCount):

        ip1 = str(random.randint(100,255))

        ip2 = str(random.randint(100,255))

        ip3 = str(random.randint(100,255))

        ip4 = str(random.randint(100,255))

        ipLine = ip1+"."+ip2+"."+ip3+"."+ip4

        IPAddress.append(ipLine)

    return


def main(argv):

    f.write('--- Shibayan: Thesis Multi-Attribute Query Resolution /

        Caching ---\n')

    f.write('--- Caching Log --- '+time.strftime("%c")+' ---\n')

    filename = ''


    try:

        opts, args = getopt.getopt(argv,"hf:n:m:s:o:",["ifile=","nodes=",

        "bitsMain=","bitsSubring=","hostname="])

    except getopt.GetoptError:
```

```
    print 'python caching.py -f <Multiattribute - filename> -n <num
        of nodes in network> -m <Num of bits Mainring> -s <Num of bits
        Sub ring> -o <hostname>'
    sys.exit(2)
for opt, arg in opts:
    if opt == '-h':
        print 'python caching.py -f <Multiattribute - filename> -n
            <num of nodes in network> -m <Num of bits Mainring> -s
            <Num of bits Sub ring> -o <hostname>'
        sys.exit()
    elif opt in ("-f", "--ifile"):
        filename = arg
    elif opt in ("-n", "--nodes"):
        nodes = arg
    elif opt in ("-m", "--bitsMain"):
        bM = arg
    elif opt in ("-s", "--bitsSubring"):
        bS = arg
    elif opt in ("-o", "--hostname"):
        hostname = arg

numNodes = int(nodes)
bitsMain = int(bM)
bitsSubring = int(bS)
```

```
print "------- Attributes --------"

print "File name: ", filename

print "Main Ring Size: ", (2**(bitsMain))

print "Sub Ring Size: ", (2**(bitsSubring))

print "Number of Nodes: ", numNodes

print "----------------------------"


f.write('---------------- Attributes -----------------\n')

f.write('Reading from file: '+str(filename)+'\n')

f.write('Main Ring Capacity: '+str(2**(bitsMain))+'\n')

f.write('Sub Ring Capacity: '+str(2**(bitsSubring))+'\n')

f.write('Num of machines in n/w: '+str(numNodes)+'\n')

f.write('---------------------------------------------\n')


rf = calParam()

_, nIP = rf.readFile(filename, Attrb, f)

f.write("Co-related params: "+str(Attrb.keys())+" \n")

if len(Attrb.keys()) == -1:

    print "No corelated attributes present"

    exit(0)

corr_attr = rf.calCorrelation(Attrb, f)

generateIPAddress(nIP)

k = None

print "Correlated Attributes: "

for k in corr_attr:
    print k
    f.write(str(k) + "\n")

ports = [19000, 20000, 21000]

RoutingTable['mLd1'] = 19000
RoutingTable['mLd5'] = 19000
RoutingTable['mLd15'] = 19000
RoutingTable['Tx'] = 20000
RoutingTable['Rx'] = 20000
RoutingTable['DSize'] = 21000
RoutingTable['DFree'] = 21000

corrAttr = [['mLd1', 'mLd5', 'mLd15'], ['Tx', 'Rx', '-'], ['DSize',
    'DFree', '-']]
numNodes = len(corrAttr)
lock = Lock()
n = 0
for n in range(numNodes):
    p = Process(target = startProcess, args = (ports[n], corrAttr[n],
        lock, Attrb, n, hostname))
    p.start()
    # p.join()
    processes.append(p)
llist = []
```

```
LongList = []

lenIPs = 0

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

ip = str(socket.gethostbyname(hostname))

server_address = (ip, 10000)

sock.bind(server_address)

count = 0

countIP = 0

print "Entering server mode"

f.write("Entering server mode \n")

print "Server Address: "+str(server_address)

f.write("Server Address: "+str(server_address)+" \n")

sock.listen(numNodes)

while True:

    conn, addr = sock.accept()

    data = conn.recv(buff)

    print "-- "+str(time.strftime("%c"))+" -- "+str(addr)+" -->

        Controller -----"

    print " Message: "+str(data)

    f.write("-- "+str(time.strftime("%c"))+" -- "+str(addr)+" -->

        Controller ----- \n")

    f.write("Message: "+str(data)+" \n")

    words = data.split(" ")

    word = words[1]

    if (word == "GET"):
```

```
if (words[2] == "LIST"):
    k = 0
    line = ''
    for k in range(len(ports)):
        recvMsg = client(data, ports[k], hostname)
        line = line + str(recvMsg) +" "
    print "Msg received: ", line
    line = "GET LIST OK "+str(line)
    msgLen = len(line)
    msg = "0"+str(msgLen)+" "+str(line)
    conn.send(msg)
if (words[2] == "IP"):
    hopCount = int(words[3])
    hopCount += 1
    sendTime = float(words[4])
    i = 0
    for i in range(len(corrAttr)):
        j = 0
        llist = []
        for j in range(len(corrAttr[i])):
            if corrAttr[i][j] is not "-":
                if corrAttr[i][j] in words:
                    k = 5
                    l = len(words)
                    attrb = corrAttr[i][j]
```

```
                        while (k < (l-1)):
                            if words[k] == attrb:
                                k += 1
                                maxV = float(words[k])
                                k += 1
                                minV = float(words[k])
                                llist.append([attrb, maxV, minV])
                                k += 1
                            else:
                                k += 1
            LongList.append([attrb, maxV, minV])
            x = 0
            line = ''
            for x in range(len(llist)):
                y = 0
                for y in range(len(llist[x])):
                    line = line + " " + str(llist[x][y])
            msg = "IP"+str(line)
            msgL = len(msg)
            Msg = "00"+str(msgL)+" "+str(msg)
            print "Sending message to peers: ", Msg
            forwardingPort = RoutingTable[attrb]
            client_without_recv(Msg, forwardingPort, hostname, f)

if word == "BF":
```

```
                count += 1
                bf.analyseBF(count, data, bloomFilters, counter, ports,
                    hostname, f)

            if word == "IP":
                countIP += 1
                bf.find_resultant_IP(hopCount, sendTime, countIP, data,
                    counter, hostname, ip_List, f)

            if word == "EXIT":
                k = 0
                for k in range(len(ports)):
                    client_without_recv(data, ports[k], hostname, f)
                    time.sleep(3)
                print "Exiting Myself !"
                exit(0)
    return

if __name__ == "__main__":
    main(sys.argv[1:])
```

- **Caching Mechn. Working, contents for "workCaching.py"**

```
import socket
import random
import math
import os

import time

import sys

from hashVals import hashfunctions


hf = hashfunctions()


class startWork:

    def getIPList(self, selAttrb, data, attrKey, valKey, f1):

        K_Val = []

        M1_Val = []

        BF = []

        data = data[:-1]

        ll = []

        words = data.split()

        # print "Length of words: ", len(words)

        k = 2

        while k < len(words):

            attrb = words[k]

            k += 1

            maxV = float(words[k])

            k += 1

            minV = float(words[k])

            k += 1

            ll.append([attrb, maxV, minV])
```

```
ipList = self.calParam(l1, selAttrb, attrKey, valKey, f1)
f = open('param.conf', 'r')
line = f.read()
line_words = line.split('\n')
f.close()
print "Reading parameters from conf file "
f1.write("Reading parameters from conf file \n")
i = 0
while i < (len(line_words)):
    if line_words[i] == '# K-Values':
        K_Val = line_words[i+1].split(' ')
    if line_words[i] == '# M-Values':
        M1_Val = line_words[i+1].split(' ')
    i += 1

K_Val = int(K_Val[0]) # Needs to be modified if there are
    multiple values
M = int(M1_Val[0])
N = len(ipList)
if str(K_Val) == "-":
    K = int((M / N) * math.log(2))
    K += 1
else:
    K = K_Val
```

```
print "K-Val: "+str(K)+" M-Val: "+str(M)+" N-Val: "+str(N)

f1.write("K-Val: "+str(K)+" M-Val: "+str(M)+" N-Val: "+str(N)+"

    \n")

k = 0

for k in range(M):

    BF.append("-")

primeNum = hf.prime_num_gen()

f1.write("\n Initial Bloom Filter: \n")

f1.write("\n"+str(BF)+" \n")

i = 0

for i in range(len(ipList)):

    hVal = hf.hash_values(M, K, ipList[i], primeNum, f1)

    # print str(ipList[i])+" -> "+str(hVal)

    j = 0

    for j in range(len(hVal)):

        ind = int(hVal[j])

        if str(BF[ind]) == "-":

            BF[ind] = 1

k = 0

checkSum = 0

for k in range(len(BF)):

    if BF[k] == "-":

        BF[k] = 0

    else:

        checkSum = checkSum + BF[k]
    f1.write("\n")

    f1.write("\n")

    f1.write("Final Bloom Filter: \n")

    f1.write(str(BF)+" \n")

    f1.write(str(checkSum)+" \n")

    Msg = str(BF)+" "+str(checkSum)

    return (Msg, ipList)


def calParam(self, ll, selAttrb, attrKey, valKey, f1):

    ipList = []

    ind = []

    if len(selAttrb) == 2:

        attr1 = selAttrb[0][0]

        maxVal1 = float(ll[0][1])

        minVal1 = float(ll[0][2])

        attr2 = selAttrb[1][0]

        maxVal2 = float(ll[1][1])

        minVal2 = float(ll[1][2])

        f1.write("\n")

        f1.write("------------------------ \n")

        f1.write(str(attr1)+" -> "+

            str(maxVal1)+":"+str(minVal1)+";"+str(attr2)+" ->

            "+str(maxVal2)+":"+str(minVal2)+" \n")

        k = 0

        for k in range(len(attrKey)):
```

```
        if float(attrKey[k][0]) > minVal1 and float(attrKey[k][0])
            < maxVal1:
            ind.append(k)
            f1.write(str(attrKey[k][0])+" -> ")
            # print str(attrKey[k][0])+" -> "
    f1.write("\n")
    # print "Indices: ", ind
    f1.write("Indices: "+str(ind)+" \n")
    k = 0
    for k in range(len(ind)):
        Ind = ind[k]
        i = 0
        for i in range(len(valKey[Ind][0])):
            if float(valKey[Ind][0][i]) > minVal2 and
                float(valKey[Ind][0][i]) < maxVal2:
                ipList.append(valKey[Ind][1][i])
                f1.write(str(valKey[Ind][0][i])+" ->
                    "+str(valKey[Ind][1][i])+" \n")
                # print str(valKey[Ind][0][i])+" ->
                    "+str(valKey[Ind][1][i])
    f1.write("----------------------------------------- \n")
if len(selAttrb) == 3:
    attr1 = selAttrb[0][0]
    maxVal1 = float(l1[0][1])
    minVal1 = float(l1[0][2])
```

```
attr2 = selAttrb[1][0]

maxVal2 = float(l1[1][1])

minVal2 = float(l1[1][2])

attr3 = selAttrb[2][0]

maxVal3 = float(l1[2][1])

minVal3 = float(l1[2][2])

f1.write("\n")

f1.write("--------------------------------------- \n")

f1.write(str(attr1)+" -> "+

    str(maxVal1)+":"+str(minVal1)+";"+str(attr2)+" ->

    "+str(maxVal2)+":"+str(minVal2)+";"+str(attr3)+" ->

    "+str(maxVal3)+":"+str(minVal3)+" \n")

k = 0

for k in range(len(attrKey)):

    if float(attrKey[k][0]) > minVal1 and float(attrKey[k][0])

        < maxVal1:

        ind.append(k)

        f1.write(str(attrKey[k][0])+" -> ")

        # print str(attrKey[k][0])+" -> "

f1.write("\n")

# print "Indices: ", ind

f1.write("Indices: "+str(ind)+" \n")

k = 0

for k in range(len(ind)):

    Ind = ind[k]
```

```
            i = 0
            for i in range(len(valKey[Ind][0])):
                if float(valKey[Ind][0][i]) > minVal2 and
                    float(valKey[Ind][0][i]) < maxVal2:
                    if float(valKey[Ind][1][i]) > minVal3 and
                        float(valKey[Ind][1][i]) < maxVal3:
                        ipList.append(valKey[Ind][2][i])
                        f1.write(str(valKey[Ind][0][i])+" ->
                            "+str(valKey[Ind][1][i])+" ->
                            "+str(valKey[Ind][2][i])+" \n")
                        # print str(valKey[Ind][0][i])+" ->
                            "+str(valKey[Ind][1][i])+" ->
                            "+str(valKey[Ind][2][i])
        f1.write("----------------------------------- \n")
    return ipList


def select_IP_Address(self, ipList, data, f1):
    elemns = data.split(",")
    words = data.split(" ")
    l = len(words)
    chkSum_sent = int(words[l-1])
    ff = open('param.conf', 'r')
    line = ff.read()
    line_words = line.split('\n')
    ff.close()
```

```
i = 0
while i < (len(line_words)):
    if line_words[i] == '# K-Values':
        K_Val = line_words[i+1].split(' ')
    if line_words[i] == '# M-Values':
        M1_Val = line_words[i+1].split(' ')
    i += 1
K_Val = int(K_Val[0]) # Needs to be modified if there are
    multiple values
M = int(M1_Val[0])
N = len(ipList)
if str(K_Val) == "-":
    K = int((M / N) * math.log(2))
    K += 1
else:
    K = K_Val
mParam = int(M1_Val[0])
tempBF = []
ipSelected = []
tempBF.append(int(elemns[0][-1:]))
k = 1
for k in range(1, (len(elemns)-1)):
    tempBF.append(int(elemns[k][1:2]))
tempBF.append(int(elemns[len(elemns) - 1][1:2]))
k = 0
```

```
chkS = 0
for k in range(len(tempBF)):
    chkS = chkS + tempBF[k]
print "----------------------------------"
print "Checksum sent: "+str(chkSum_sent)+"|"+"Checksum recv:
    "+str(chkS)
print "RBF Size sent: "+str(mParam)+"|"+"RBF Size recv:
    "+str(len(tempBF))
print "----------------------------------"
f1.write("-------------------------------- \n")
f1.write("Checksum sent: "+str(chkSum_sent)+"|"+"Checksum recv:
    "+str(chkS)+" \n")
f1.write("RBF Size sent: "+str(mParam)+"|"+"RBF Size recv:
    "+str(len(tempBF))+" \n")
f1.write("-------------------------------- \n")
if int(chkS) == int(chkSum_sent) and int(len(tempBF)) ==
    int(mParam):
    i = 0
    primeNum = hf.prime_num_gen()
    for i in range(len(ipList)):
        hVal = hf.hash_values(M, K, ipList[i], primeNum, f1)
        j = 0
        count = 0
        for j in range(len(hVal)):
            if tempBF[int(hVal[j])] == 1:
            count += 1
        if count == K:
            ipSelected.append(ipList[i])
line = ''
k = 0
for k in range(len(ipSelected)):
    line = line +" "+str(ipSelected[k])
line = line + " " +str(len(ipSelected))
line = line[1:]
# print "IPs: ", str(line)
f1.write("IPs: "+str(line)+" \n")
return line
```

- **Bloom-Filter Oprn for Caching Mechanism, contents for "$bloomfilter_{op}.py$"**

```
import threading
import socket
import time

lock = threading.Lock()

def client_without_recv(Msg, port, hostname, f):
    global buff
    port = int(port)
    ip = str(socket.gethostbyname(hostname))
    time.sleep(5)
    # Creating TCP Socket
    global buff
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((ip, port))
    sock.send(Msg)
    sock.close()
    return

class BloomFilter_Operation:
    def analyseBF(self, count, msg, bloomFilters, counter, ports,
        hostname, f):
        lock.acquire()
        print "Received #-"+str(count)+" "+"BloomFilter"
        f.write("Received #-"+str(count)+" "+"BloomFilter"+" \n")
        words = msg.split(" ")
        l = len(words)
        chkSum_sent = int(words[l-1])
        elemns = msg.split(",")
        # print "Elements: ", elemns[0]
        # print "Elements count: ", len(elemns)

        f1 = open('param.conf', 'r')
        line = f1.read()
        line_words = line.split('\n')
        f1.close()
```

```
print "Reading parameters from conf file "

f.write("Reading parameters from conf file \n")

i = 0

while i < (len(line_words)):

    if line_words[i] == '# M-Values':

        M1_Val = line_words[i+1].split(' ')

    i += 1


mParam = int(M1_Val[0])

tempBF = []

# print "Elements:

    "+str(elemns[0][-1:])+","+str(elemns[len(elemns) - 1][1:2])

# print "Elements: ", elemns[1][1:2]

tempBF.append(int(elemns[0][-1:]))

k = 1

for k in range(1, (len(elemns)-1)):

    tempBF.append(int(elemns[k][1:2]))

tempBF.append(int(elemns[len(elemns) - 1][1:2]))

k = 0

chkS = 0

for k in range(len(tempBF)):

    chkS = chkS + tempBF[k]

# print "Temp BF: ", tempBF

print "---------------------------"
```

```
        print "Checksum sent: "+str(chkSum_sent)+"|"+"Checksum recv:
            "+str(chkS)
        print "BF Size sent: "+str(mParam)+"|"+"BF Size recv:
            "+str(len(tempBF))
        print "----------------------------"
        f.write("------------------------ \n")
        f.write("Checksum sent: "+str(chkSum_sent)+"|"+"Checksum recv:
            "+str(chkS)+" \n")
        f.write("BF Size sent: "+str(mParam)+"|"+"BF Size recv:
            "+str(len(tempBF))+" \n")
        f.write("------------------------ \n")
        if int(chkS) == int(chkSum_sent) and int(len(tempBF)) ==
            int(mParam):
            bloomFilters[count] = tempBF
        lock.release()
        if count == counter:
            self.bitwiseand(bloomFilters, ports, hostname, f)
        return


    def bitwiseand(self, bloomFilters, ports, hostname, f):
        Keys = bloomFilters.keys()
        tempBF = []
        k = 0
        for k in range(len(Keys)):
            if k == 0:
                tempBF = bloomFilters[Keys[k]]
            else:
                i = 0
                for i in range(len(tempBF)):
                    tempBF[i] = bloomFilters[Keys[k]][i] & tempBF[i]
        l = len(tempBF)
        k = 0
        s = 0
        for k in range(len(tempBF)):
            s = s + tempBF[k]
        bloomFilters[len(Keys)+1] = tempBF
        print "------------------------- "
        print "Resultant Bloom Filter: "
        print tempBF
        print "CheckSum: "+str(s)+" Size: "+str(l)
        print "-------------------------- "
        f.write("----------------------- \n")
        f.write("Resultant Bloom Filter: \n")
        f.write(str(tempBF)+" \n")
        f.write("CheckSum: "+str(s)+" Size: "+str(l)+" \n")
        f.write("----------------------- \n")
        self.forward_resultant_BF(tempBF, ports, s, hostname, f)
        return


    def forward_resultant_BF(self, tempBF, ports, s, hostname, f):
```

```
    f.write("Sending resultant bloom filter to the sub rings \n")

    print "Sending resultant bloom filter to the sub rings \n"

    msg = "RBF "+str(tempBF)+" "+str(s)

    lenMsg = len(msg)

    Msg = "00"+str(lenMsg)+" "+str(msg)

    f.write(str(Msg)+" \n")

    k = 0

    for k in range(len(ports)):

        client_without_recv(Msg, ports[k], hostname, f)

    return


def find_resultant_IP(self, hopCount, sendTime, countIP, data,

    counter, hostname, ip_List, f):

    lock.acquire()

    words = data.split(" ")

    c = int(words[len(words) - 1])

    k = 0

    ipTemp = []

    for k in range(2, (len(words) - 1)):

        ipTemp.append(words[k])

    if len(ipTemp) == c:

        ip_List[countIP] = ipTemp

    lock.release()

    if countIP == counter:

        # print "Final data base: ", ip_List
```

```
            f.write("Final data base: " +str(ip_List)+" \n")
            self.IP_list_user(hopCount, sendTime, ip_List, counter,
                hostname, data, f)
        return


    def IP_list_user(self, hopCount, sendTime, ip_List, counter,
        hostname, data, f):
        Keys = ip_List.keys()
        if len(Keys) == counter:
            ipTemp = []
            k = 0
            for k in range(len(Keys)):
                if k == 0:
                    ips = []
                    ips = ip_List[Keys[k]]
                    i = 0
                    for i in range(len(ips)):
                        l = [ips[i], 1]
                        ipTemp.append(l)
                else:
                    ips = []
                    ips = ip_List[Keys[k]]
                    # print "IPS: ", ips
                    i = 0
                    for i in range(len(ipTemp)):
```

```
                j = 0
                for j in range(len(ips)):
                    if ips[j] == ipTemp[i][0]:
                        # print "I am here"
                        c = ipTemp[i][1]
                        c += 1
                        ipTemp[i][1] = c
'''
k = None
for k in ipTemp:
    print k
'''
fList = []
k = 0
for k in range(len(ipTemp)):
    if int(ipTemp[k][1]) == counter:
        fList.append(ipTemp[k][0])
print "Final IP-List: "
f.write("---------------------------------- \n")
f.write("Final IP-List: \n")
k = None
line = ''
for k in fList:
    f.write(str(k)+" \n")
    line = line +" "+str(k)
```

```
f.write("----------------------------------- \n")

# calculate fp

userPort = 10500

ip_List[counter + 1] = fList

msg = "IPLIST-CR "+str("-")+" "+str(sendTime)+"

   "+str(hopCount)+line+" "+str(len(fList))

msgL = len(msg)

print "Sending final message to user: "

f.write("Sending final message to user: \n")

Msg = "00"+str(msgL)+" "+str(msg)

print Msg

f.write(str(Msg)+" \n")

client_without_recv(Msg, userPort, hostname, f)

f.write("End of simulation \n")

print "End of simulation"

return
```

APPENDIX C

Overlapped Ring Architecture Source Code

This chapter presents the source code for Overlapped Architecture for the resolution of Co-related Attributes. The src code uses "Python2.7", along with "numpy" and "pandas" packages. The simulation to be started with the execution of "overlapped.py". The other files necessary are "readfile.py", "workOverlapped.py", "$bloomfilter_{op}.py$", and resource list obtained from "ResQue". The "readfile.py" is mentioned in Appendix-E To execute the src code:

```
sudo chmod +x overlapped.py
./caching.py -f <Multiattribute - filename> -n <num of nodes in network> -m <Num
    of bits Mainring> -s <Num of bits Sub ring> -o <hostname>
```

The number of nodes present in the network should be greater than or equal to the number of attributes mentioned in resources list obtained from "ResQue". The contents for each file name is mentioned along with each sections. At the end of successful execution of simulation, "Overlapped" architecture will be formed ready to resolve queries related to correlated attributes.

- **Over-lapped Ring Architecture Formation, contents for "overlapped.py"**

```
import socket
import os
import sys
import time
import getopt
import random
import numpy as np
from readFile import calParam
from multiprocessing import Pool, Process, Manager, Lock
from workOverlapped import startWork
from Bloomfilter_op import BloomFilter_operation

bf = BloomFilter_operation()

os.system('clear')
f = open('overlapped-ring.log', 'w')
f.close()
f = open('overlapped-ring.log', 'a')

RoutingTable = {}
bloomFilters = {}
ip_List = {}
IPAddress = []
processes = []
Attrb = {}
buff = 51200
counter = 3

def client_without_recv(Msg, port, hostname, f):
    global buff
    port = int(port)
    ip = str(socket.gethostbyname(hostname))
    time.sleep(5)
    # Creating TCP Socket
    global buff
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((ip, port))
    sock.send(Msg)
    sock.close()
    return

def client(msg, port, hostname):
    port = int(port)
    ip  = str(socket.gethostbyname(hostname))
    time.sleep(5)
    global buff
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((ip, port))
    sock.send(msg)
    data = sock.recv(buff)
    sock.close()
    return data

def startProcess(port, corrAttr, lock, Attrb, n, hostname, mccount,
    bitsSubring, mainRingID):
    var = "Process-"+str(n+1)
```

```
print "Starting: " + var

print "Main PID: " + str(os.getppid())

print "PID: " + str(os.getpid())

PID = os.getpid()

MPID = os.getppid()

ipList = []

print "------------------------------------------------"

print "My Main Ring ID: ", mainRingID

print "Num of machines in my subring: ", mccount

subRingIDs = random.sample(xrange(2**bitsSubring), mccount)

print "My Subring mc ID's: ", subRingIDs

conPort = 10000

var1 = "Process-"+str(n+1)+".log"

f1 = open(var1, 'w')

f1.close()

f1 = open(var1, 'a')

f1.write("----- Starting Process-"+str(n+1)+" "+

   time.strftime("%c")+'-----\n')

f1.write(' Parent ID: '+str(MPID)+ '\n')

f1.write(' PID: '+str(PID)+' \n')

f1.write("My Subring mc ID's: "+str(subRingIDs)+"\n")

selAttrb = []

attrKey = []

valKey1 = []

valKey2 = []
```

```
if corrAttr[0] == "mLd1":
    selAttrb = [str(corrAttr[0]), str(corrAttr[1]), str(corrAttr[2])]
    f1.write("I am reposible for correlation of:
        "+str(selAttrb[0])+";"+str(selAttrb[1])+";"+str(selAttrb[2])+"
        \n")
    print "I am reposible for correlation of:
        "+str(selAttrb[0])+";"+str(selAttrb[1])+";"+str(selAttrb[2])
    Key1 = selAttrb[0]
    Key2 = selAttrb[1]
    Key3 = selAttrb[2]
    Val1 = Attrb[Key1].flatten()
    Val2 = Attrb[Key2].flatten()
    Val3 = Attrb[Key3].flatten()
    i = 0
    for i in range(len(Val1)):
        attrKey.append(Val1[i])
    i = 0
    for i in range(len(Val2)):
        valKey1.append(Val2[i])
    i = 0
    for i in range(len(Val3)):
        valKey2.append(Val3[i])
else:
    selAttrb = [str(corrAttr[0]), str(corrAttr[1])]
    print "I am reposible for correlation of:
        "+str(selAttrb[0])+";"+str(selAttrb[1])
    f1.write("I am reposible for correlation of:
        "+str(selAttrb[0])+";"+str(selAttrb[1])+" \n")
    Key1 = selAttrb[0]
    Key2 = selAttrb[1]
    Val1 = Attrb[Key1].flatten()
    Val2 = Attrb[Key2].flatten()
    i = 0
    for i in range(len(Val1)):
        attrKey.append(Val1[i])
    i = 0
    for i in range(len(Val2)):
        valKey1.append(Val2[i])
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
ip = str(socket.gethostbyname(hostname))
port = int(port)
server_address = (ip, port)
sock.bind(server_address)
print "Entering server mode at: ", var
f1.write("Entering server mode at: " +str(var)+" \n")
print "Server Address: "+str(server_address)
f1.write("Server Address: "+str(server_address)+" \n")
sock.listen(20)
sw = startWork()
```

```
while True:

    conn, addr = sock.accept()

    data = conn.recv(buff)

    words = data.split(" ")

    print "Received conn from: " +str(addr)+"

        "+str(time.strftime("%c"))

    print "Message: ", data

    f1.write("Received conn from: " +str(addr)+"

        "+str(time.strftime("%c"))+" \n")

    f1.write("Message: "+str(data)+" \n")

    word = words[1]

    if word == "GET":

        if words[2] == "LIST":

            if len(selAttrb) == 3:

                line = str(Key1)+" "+str(max(Val1))+"

                    "+str(min(Val1))+" "+str(Key2)+" "+str(max(Val2))+"

                    "+str(min(Val2))+" "+str(Key3)+" "+str(max(Val3))+"

                    "+str(min(Val3))

            if len(selAttrb) == 2:

                line = str(Key1)+" "+str(max(Val1))+"

                    "+str(min(Val1))+" "+str(Key2)+" "+str(max(Val2))+"

                    "+str(min(Val2))

            conn.send(line)

    if word == "IP":
            msg, ipList = sw.getIPList(selAttrb, data, attrKey, valKey1,
                valKey2, IPAddress, f1)
            msgL = len(msg)
            Msg = "00"+str(msgL)+" BF "+str(msg)
            client_without_recv(Msg, conPort, hostname, f1)
        if word == "RBF":
            msg = sw.select_IP_Address(ipList, data, f1)
            msgL = len(msg)
            Msg = "00"+str(msgL)+" IP "+str(msg)
            client_without_recv(Msg, conPort, hostname, f1)
        if word == "EXIT":
            print "Exiting Myself: "+str(var)
            exit(0)
    return


def generateIPAddress(ipCount):
    k = 0
    for k in range(ipCount):
        ip1 = str(random.randint(100,255))
        ip2 = str(random.randint(100,255))
        ip3 = str(random.randint(100,255))
        ip4 = str(random.randint(100,255))
        ipLine = ip1+"."+ip2+"."+ip3+"."+ip4
        IPAddress.append(ipLine)
    return
```

```
def main(argv):
    f.write('--- Shibayan: Thesis Multi-Attribute Query Resolution /
        Overlapped Ring ---\n')
    f.write('--- Overlapped-ring Log --- '+time.strftime("%c")+' ---\n')
    filename = ''

    try:
        opts, args = getopt.getopt(argv,"hf:n:m:s:o:",["ifile=","nodes=",
        "bitsMain=","bitsSubring=","hostname="])
    except getopt.GetoptError:
        print 'python overlapped.py -f <Multiattribute - filename> -n
            <num of nodes in network> -m <Num of bits Mainring> -s <Num of
            bits Sub ring> -o <hostname>'
        sys.exit(2)
    for opt, arg in opts:
        if opt == '-h':
            print 'python overlapped.py -f <Multiattribute - filename> -n
                <num of nodes in network> -m <Num of bits Mainring> -s
                <Num of bits Sub ring> -o <hostname>'
            sys.exit()
        elif opt in ("-f", "--ifile"):
            filename = arg
        elif opt in ("-n", "--nodes"):
            nodes = arg
        elif opt in ("-m", "--bitsMain"):
            bM = arg
        elif opt in ("-s", "--bitsSubring"):
            bS = arg
        elif opt in ("-o", "--hostname"):
            hostname = arg

    numNodes = int(nodes)
    bitsMain = int(bM)
    bitsSubring = int(bS)

    print "------- Attributes --------"
    print "File name: ", filename
    print "Main Ring Size: ", (2**(bitsMain))
    print "Sub Ring Size: ", (2**(bitsSubring))
    print "Number of Nodes: ", numNodes
    print "----------------------------"

    f.write('---------------- Attributes -----------------\n')
    f.write('Reading from file: '+str(filename)+'\n')
    f.write('Main Ring Capacity: '+str(2**(bitsMain))+'\n')
    f.write('Sub Ring Capacity: '+str(2**(bitsSubring))+'\n')
    f.write('Num of machines in n/w: '+str(numNodes)+'\n')
    f.write('---------------------------------------------\n')
```

```
mainRingIDs = random.sample(xrange(2**bitsMain), 3)

nummcSR = numNodes - 3

numsrmcCount = random.sample(xrange(nummcSR), 3)

rf = calParam()

_, nIP = rf.readFile(filename, Attrb, f)

f.write("Co-related params: "+str(Attrb.keys())+" \n")

if len(Attrb.keys()) == -1:

    print "No corelated attributes present"

    exit(0)

corr_attr = rf.calCorrelation(Attrb, f)

generateIPAddress(nIP)

k = None

print "Correlated Attributes: "

for k in corr_attr:

    print k

    f.write(str(k) + "\n")


ports = [19000, 20000, 21000]


RoutingTable['mLd1'] = 19000

RoutingTable['mLd5'] = 19000

RoutingTable['mLd15'] = 19000

RoutingTable['Tx'] = 20000

RoutingTable['Rx'] = 20000

RoutingTable['DSize'] = 21000
```

```
RoutingTable['DFree'] = 21000


corrAttr = [['mLd1', 'mLd5', 'mLd15'], ['Tx', 'Rx', '-'], ['DSize',
    'DFree', '-']]
numNodes = len(corrAttr)
lock = Lock()
n = 0
for n in range(numNodes):
    p = Process(target = startProcess, args = (ports[n], corrAttr[n],
        lock, Attrb, n, hostname, numsrmcCount[n],bitsSubring,
        mainRingIDs[n]))
    p.start()
    processes.append(p)
llist = []
LongList = []
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
ip = str(socket.gethostbyname(hostname))
server_address = (ip, 10000)
sock.bind(server_address)
count = 0
countIP = 0
lenIPs = 0
print "Entering server mode"
f.write("Entering server mode \n")
print "Server Address: "+str(server_address)
```

```
f.write("Server Address: "+str(server_address)+" \n")

sock.listen(numNodes)

while True:

    conn, addr = sock.accept()

    data = conn.recv(buff)

    print "-- "+str(time.strftime("%c"))+" -- "+str(addr)+" -->

        Controller -----"

    print " Message: "+str(data)

    f.write("-- "+str(time.strftime("%c"))+" -- "+str(addr)+" -->

        Controller ----- \n")

    f.write("Message: "+str(data)+" \n")

    words = data.split(" ")

    word = words[1]

    if (word == "GET"):

        if (words[2] == "LIST"):

            k = 0

            line = ''

            for k in range(len(ports)):

                recvMsg = client(data, ports[k], hostname)

                line = line + str(recvMsg) +" "

            print "Msg received: ", line

            line = "GET LIST OK "+str(line)

            msgLen = len(line)

            msg = "0"+str(msgLen)+" "+str(line)

            conn.send(msg)
```

```
if (words[2] == "IP"):
    hopCount = int(words[3])
    hopCount += 1
    sendTime = float(words[4])
    i = 0
    for i in range(len(corrAttr)):
        j = 0
        llist = []
        for j in range(len(corrAttr[i])):
            if corrAttr[i][j] is not "-":
                if corrAttr[i][j] in words:
                    k = 3
                    l = len(words)
                    attrb = corrAttr[i][j]
                    while (k < (l-1)):
                        if words[k] == attrb:
                            k += 1
                            maxV = float(words[k])
                            k += 1
                            minV = float(words[k])
                            llist.append([attrb, maxV, minV])
                            k += 1
                        else:
                            k += 1
        LongList.append([attrb, maxV, minV])
```

```
            x = 0
            line = ''
            for x in range(len(llist)):
                y = 0
                for y in range(len(llist[x])):
                    line = line + " " + str(llist[x][y])
            msg = "IP"+str(line)
            msgL = len(msg)
            Msg = "00"+str(msgL)+" "+str(msg)
            print "Sending message to peers: ", Msg
            forwardingPort = RoutingTable[attrb]
            client_without_recv(Msg, forwardingPort, hostname, f)
if word == "BF":
    count += 1
    bf.analyze_bloomfilter(count, data, bloomFilters, counter,
        ports, hostname, f)
if word == "IP":
    countIP += 1
    bf.find_resultant_IP(hopCount, sendTime, countIP, data,
        counter, hostname, ip_List, lenIPs, f)
if word == "EXIT":
    k = 0
    for k in range(len(ports)):
        client_without_recv(data, ports[k], hostname, f)
        time.sleep(3)
```

```
            print "Exiting Myself !"
            exit(0)
    return


if __name__ == "__main__":
    main(sys.argv[1:])
```

- **Overlapped Resource Selection, contents for "workOverlapped.py"**

```
import socket
import random
import math
import os
import time
import sys
import numpy as np
from hashCalculate import hashfunctions


hf = hashfunctions()


class startWork:
    def funcTxRx(self, x):
        return float((np.log(x)) + x + 1)


    def funcDSDF(self, x):
        return float((np.log(x**(-10.20587197)) + (1.02847843 * x)))
```

```
def funcmLd15(self, x):
    return (np.log(x**(0.14554757)) + (0.7263016 * x) + 0.16900252)

def funcmLd115(self, x):
    return (np.log(x**(0.18088713))+ (0.63456338 * x) + 0.27296098)

def getIPList(self, selAttrb, data, attrKey, valKey1, valKey2,
    IPAddress, f1):
    K_Val = []
    M1_Val = []
    BF = []
    data = data[:-1]
    ll = []
    words = data.split()
    k = 2
    while k < len(words):
        attrb = words[k]
        k += 1
        maxV = float(words[k])
        k += 1
        minV = float(words[k])
        k += 1
        ll.append([attrb, maxV, minV])
    f = open('param.conf', 'r')
```

```
line = f.read()
line_words = line.split('\n')
f.close()
print "Reading parameters from conf file "
f1.write("Reading parameters from conf file \n")
i = 0
while i < (len(line_words)):
    if line_words[i] == '# K-Values':
        K_Val = line_words[i+1].split(' ')
    if line_words[i] == '# M-Values':
        M1_Val = line_words[i+1].split(' ')
    i += 1
K_Val = int(K_Val[0]) # Needs to be modified if there are
    multiple values
M = int(M1_Val[0])
ipList = self.calParam(ll, selAttrb, attrKey, valKey1, valKey2,
    IPAddress, f1)
N = len(ipList)
if str(K_Val) == "-":
    K = int((M / N) * math.log(2))
    K += 1
else:
    K = K_Val
print "K-Val: "+str(K)+" M-Val: "+str(M)+" N-Val: "+str(N)
```

```
f1.write("K-Val: "+str(K)+" M-Val: "+str(M)+" N-Val: "+str(N)+"
    \n")
k = 0
for k in range(M):
    BF.append("-")
primeNum = hf.prime_num_gen()
f1.write("\n Initial Bloom Filter: \n")
f1.write("\n"+str(BF)+" \n")
i = 0
for i in range(len(ipList)):
    hVal = hf.hash_values(M, K, ipList[i], primeNum, f1)
    # print str(ipList[i])+" -> "+str(hVal)
    j = 0
    for j in range(len(hVal)):
        ind = int(hVal[j])
        if str(BF[ind]) == "-":
            BF[ind] = 1
k = 0
checkSum = 0
for k in range(len(BF)):
    if BF[k] == "-":
        BF[k] = 0
    else:
        checkSum = checkSum + BF[k]
f1.write("\n")
```

```
    f1.write("\n")
    f1.write("Final Bloom Filter: \n")
    f1.write(str(BF)+" \n")
    f1.write(str(checkSum)+" \n")
    Msg = str(BF)+" "+str(checkSum)
    return (Msg, ipList)

def calParam(self, ll, selAttrb, attrKey, valKey1, valKey2,
    IPAddress, f1):
    ipList = []
    ind = []
    if selAttrb[0] == 'mLd1':
        minVal = float(ll[0][2])
        minpredictedmLd5 = self.funcmLd15(minVal)
        if float(minpredictedmLd5) <= 0.0:
            minpredictedmLd5 = 0.0
        minpredictedmLd15 = self.funcmLd115(minVal)
        if float(minpredictedmLd15) <= 0.0:
            minpredictedmLd15 = 0.0
        maxVal = float(ll[0][1])
        maxpredictedmLd5 = self.funcmLd15(maxVal)
        maxpredictedmLd15 = self.funcmLd115(maxVal)
```

```
print "mLd1: ["+str(maxVal)+","+str(minVal)+"] -> Actual mLd5
    ["+str(float(l1[1][1]))+","+str(float(l1[1][2]))+"] ->
    Actual mLd15
    ["+str(float(l1[2][1]))+","+str(float(l1[2][2]))+"]"
print "mLd1: ["+str(maxVal)+","+str(minVal)+"] -> Overlapped
    mLd5 ["+str(maxpredictedmLd5)+","+str(minpredictedmLd5)+"]
    -> Overlapped mLd15
    ["+str(maxpredictedmLd15)+","+str(minpredictedmLd15)+"]"
print "mLd5 actual range: "+str(float(l1[1][1]) -
    float(l1[1][2]))
print "mLd15 actual range: "+str(float(l1[2][1]) -
    float(l1[2][2]))
f1.write("mLd1: ["+str(maxVal)+","+str(minVal)+"] -> Actual
    mLd5 ["+str(float(l1[1][1]))+","+str(float(l1[1][2]))+"]
    -> Actual mLd15
    ["+str(float(l1[2][1]))+","+str(float(l1[2][2]))+"] \n")
f1.write("mLd1: ["+str(maxVal)+","+str(minVal)+"] ->
    Overlapped mLd5
    ["+str(maxpredictedmLd5)+","+str(minpredictedmLd5)+"] ->
    Overlapped mLd15
    ["+str(maxpredictedmLd15)+","+str(minpredictedmLd15)+"]
    \n")
f1.write("mLd5 actual range: "+str(float(l1[1][1]) -
    float(l1[1][2]))+" \n")
```

```
        f1.write("mLd15 actual range: "+str(float(l1[2][1]) -
            float(l1[2][2]))+" \n")
        i = 0
        for i in range(len(valKey1)):
            if float(valKey1[i]) >= minpredictedmLd5 and
                float(valKey1[i]) <= maxpredictedmLd5:
                if float(valKey2[i]) >= minpredictedmLd15 and
                    float(valKey2[i]) <= maxpredictedmLd15:
                    ind.append(i)
        i = 0
        f1.write("Selected IP Addresses \n")
        f1.write("---------------------------------------- \n")
        for i in range(len(ind)):
            ipList.append(IPAddress[ind[i]])
            f1.write(str(IPAddress[ind[i]])+" \n")
        f1.write("---------------------------------------- \n")
    if selAttrb[0] == 'Tx':
        minVal = float(l1[0][2])
        minpredictedRx = self.funcTxRx(minVal)
        if float(minpredictedRx) <= 0.0:
            minpredictedRx = 0.0
        maxVal = float(l1[0][1])
        maxpredictedRx = self.funcTxRx(maxVal)
        print "Tx: ["+str(maxVal)+","+str(minVal)+"] -> Actual Rx
            ["+str(float(l1[1][1]))+","+str(float(l1[1][2]))+"]"
```

```
print "Tx: ["+str(maxVal)+","+str(minVal)+"] -> Overlapped Rx
    ["+str(maxpredictedRx)+","+str(minpredictedRx)+"]"
print "Rx actual range: "+str(float(l1[1][1]) -
    float(l1[1][2]))
print "Rx overlapped range: "+str(float(maxpredictedRx) -
    float(minpredictedRx))
f1.write("Tx: ["+str(maxVal)+","+str(minVal)+"] -> Actual Rx
    ["+str(float(l1[1][1]))+","+str(float(l1[1][2]))+"] \n")
f1.write("Tx: ["+str(maxVal)+","+str(minVal)+"] -> Overlapped
    Rx ["+str(maxpredictedRx)+","+str(minpredictedRx)+"] \n")
f1.write("Rx actual range: "+str(float(l1[1][1]) -
    float(l1[1][2]))+" \n")
f1.write("Rx overlapped range: "+str(float(maxpredictedRx) -
    float(minpredictedRx))+" \n")
i = 0
for i in range(len(valKey1)):
    if float(valKey1[i]) >= minpredictedRx and
        float(valKey1[i]) <= maxpredictedRx:
        ind.append(i)
i = 0
f1.write("Selected IPAddresses: \n")
f1.write("------------------------- \n")
for i in range(len(ind)):
    ipList.append(IPAddress[ind[i]])
    f1.write(str(IPAddress[ind[i]])+" \n")
```

```
        f1.write("------------------------- \n")
    if selAttrb[0] == 'DSize':
        minVal = float(ll[0][2])
        minpredictedDFree = self.funcDSDF(minVal)
        if float(minpredictedDFree) <= 0.0:
            minpredictedDFree = 0.0
        maxVal = float(ll[0][1])
        maxpredictedDFree = self.funcDSDF(maxVal)
        print "DSize: ["+str(maxVal)+","+str(minVal)+"] -> Actual
            DFree ["+str(float(ll[1][1]))+","+str(float(ll[1][2]))+"]"
        print "DSize: ["+str(maxVal)+","+str(minVal)+"] -> Overlapped
            DFree
            ["+str(maxpredictedDFree)+","+str(minpredictedDFree)+"]"
        print "DFree actual range: "+str(float(ll[1][1]) -
            float(ll[1][2]))
        print "DFree overlapped range: "+str(float(maxpredictedDFree)
            - float(minpredictedDFree))
        f1.write("DSize: ["+str(maxVal)+","+str(minVal)+"] -> Actual
            DFree ["+str(float(ll[1][1]))+","+str(float(ll[1][2]))+"]
            \n")
        f1.write("DSize: ["+str(maxVal)+","+str(minVal)+"] ->
            Overlapped DFree
            ["+str(maxpredictedDFree)+","+str(minpredictedDFree)+"]
            \n")
        f1.write("DFree actual range: "+str(float(ll[1][1]) -
           float(ll[1][2]))+" \n")
        f1.write("DFree overlapped range:
           "+str(float(maxpredictedDFree) -
           float(minpredictedDFree))+" \n")
        i = 0
        for i in range(len(valKey1)):
           if float(valKey1[i]) >= minpredictedDFree and
              float(valKey1[i]) <= maxpredictedDFree:
              ind.append(i)
        i = 0
        f1.write("Selected IPAddresses: \n")
        f1.write("------------------------- \n")
        for i in range(len(ind)):
           ipList.append(IPAddress[ind[i]])
           f1.write(str(IPAddress[ind[i]])+" \n")
        f1.write("------------------------- \n")
    return ipList


def select_IP_Address(self, ipList, data, f1):
    elemns = data.split(",")
    words = data.split(" ")
    l = len(words)
    chkSum_sent = int(words[l-1])
    ff = open('param.conf', 'r')
```

```
line = ff.read()

line_words = line.split('\n')

ff.close()

i = 0

while i < (len(line_words)):

    if line_words[i] == '# K-Values':

        K_Val = line_words[i+1].split(' ')

    if line_words[i] == '# M-Values':

        M1_Val = line_words[i+1].split(' ')

    i += 1

K_Val = int(K_Val[0]) # Needs to be modified if there are

    multiple values

M = int(M1_Val[0])

N = len(ipList)

if str(K_Val) == "-":

    K = int((M / N) * math.log(2))

    K += 1

else:

    K = K_Val

mParam = int(M1_Val[0])

tempBF = []

ipSelected = []

tempBF.append(int(elemns[0][-1:]))

k = 1

for k in range(1, (len(elemns)-1)):
    tempBF.append(int(elemns[k][1:2]))
tempBF.append(int(elemns[len(elemns) - 1][1:2]))
k = 0
chkS = 0
for k in range(len(tempBF)):
    chkS = chkS + tempBF[k]
print "------------------------------"
print "Checksum sent: "+str(chkSum_sent)+"|"+"Checksum recv:
    "+str(chkS)
print "RBF Size sent: "+str(mParam)+"|"+"RBF Size recv:
    "+str(len(tempBF))
print "------------------------------"
f1.write("---------------------------- \n")
f1.write("Checksum sent: "+str(chkSum_sent)+"|"+"Checksum recv:
    "+str(chkS)+" \n")
f1.write("RBF Size sent: "+str(mParam)+"|"+"RBF Size recv:
    "+str(len(tempBF))+" \n")
f1.write("---------------------------- \n")
if int(chkS) == int(chkSum_sent) and int(len(tempBF)) ==
    int(mParam):
    i = 0
    primeNum = hf.prime_num_gen()
    for i in range(len(ipList)):
        hVal = hf.hash_values(M, K, ipList[i], primeNum, f1)
        j = 0
```

```
                count = 0
                for j in range(len(hVal)):
                    if tempBF[int(hVal[j])] == 1:
                        count += 1
                if count == K:
                    ipSelected.append(ipList[i])
        line = ''
        k = 0
        for k in range(len(ipSelected)):
            line = line +" "+str(ipSelected[k])
        line = line + " " +str(len(ipSelected))
        line = line[1:]
        # print "IPs: ", str(line)
        f1.write("IPs: "+str(line)+" \n")
        return line
```

- **Bloom-Filter Operation for Over-lapped Ring Architecture, contents for "$Bloomfilter_{op}.py$"**

```
import threading
import socket
import time


lock = threading.Lock()


def client_without_recv(Msg, port, hostname, f):
    global buff

    port = int(port)

    ip = str(socket.gethostbyname(hostname))

    time.sleep(5)

    # Creating TCP Socket

    global buff

    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    sock.connect((ip, port))

    sock.send(Msg)

    sock.close()

    return


class BloomFilter_operation:

    def analyze_bloomfilter(self, count, msg, bloomFilters, counter,

        ports, hostname, f):

        lock.acquire()

        print "Received #-"+str(count)+" "+"BloomFilter"

        f.write("Received #-"+str(count)+" "+"BloomFilter"+" \n")

        words = msg.split(" ")

        l = len(words)

        chkSum_sent = int(words[l-1])

        elemns = msg.split(",")

        # print "Elements: ", elemns[0]

        # print "Elements count: ", len(elemns)
```

```
f1 = open('param.conf', 'r')

line = f1.read()

line_words = line.split('\n')

f1.close()

print "Reading parameters from conf file "

f.write("Reading parameters from conf file \n")

i = 0

while i < (len(line_words)):

    if line_words[i] == '# M-Values':

        M1_Val = line_words[i+1].split(' ')

    i += 1


mParam = int(M1_Val[0])

tempBF = []

# print "Elements:

    "+str(elemns[0][-1:])+","+str(elemns[len(elemns) - 1][1:2])

# print "Elements: ", elemns[1][1:2]

tempBF.append(int(elemns[0][-1:]))

k = 1

for k in range(1, (len(elemns)-1)):

    tempBF.append(int(elemns[k][1:2]))

tempBF.append(int(elemns[len(elemns) - 1][1:2]))

k = 0

chkS = 0

for k in range(len(tempBF)):
```

```
            chkS = chkS + tempBF[k]
        # print "Temp BF: ", tempBF
        print "---------------------------------"
        print "Checksum sent: "+str(chkSum_sent)+"|"+"Checksum recv:
            "+str(chkS)
        print "BF Size sent: "+str(mParam)+"|"+"BF Size recv:
            "+str(len(tempBF))
        print "---------------------------------"
        f.write("----------------------------- \n")
        f.write("Checksum sent: "+str(chkSum_sent)+"|"+"Checksum recv:
            "+str(chkS)+" \n")
        f.write("BF Size sent: "+str(mParam)+"|"+"BF Size recv:
            "+str(len(tempBF))+" \n")
        f.write("----------------------------- \n")
        if int(chkS) == int(chkSum_sent) and int(len(tempBF)) ==
            int(mParam):
            bloomFilters[count] = tempBF
        lock.release()
        if count == counter:
            self.bitwiseand(bloomFilters, ports, hostname, f)
        return

    def bitwiseand(self, bloomFilters, ports, hostname, f):
        Keys = bloomFilters.keys()
        tempBF = []
```

```
k = 0

for k in range(len(Keys)):

    if k == 0:

        tempBF = bloomFilters[Keys[k]]

    else:

        i = 0

        for i in range(len(tempBF)):

            tempBF[i] = bloomFilters[Keys[k]][i] & tempBF[i]

l = len(tempBF)

k = 0

s = 0

for k in range(len(tempBF)):

    s = s + tempBF[k]

bloomFilters[len(Keys)+1] = tempBF

print "----------------------- "

print "Resultant Bloom Filter: "

print tempBF

print "CheckSum: "+str(s)+" Size: "+str(l)

print "----------------------- "

f.write("---------------------\n")

f.write("Resultant Bloom Filter: \n")

f.write(str(tempBF)+" \n")

f.write("CheckSum: "+str(s)+" Size: "+str(l)+" \n")

f.write("---------------------\n")

self.forward_resultant_BF(tempBF, ports, s, hostname, f)
```

```
    return


def forward_resultant_BF(self, tempBF, ports, s, hostname, f):
    f.write("Sending resultant bloom filter to the sub rings \n")
    print "Sending resultant bloom filter to the sub rings \n"
    msg = "RBF "+str(tempBF)+" "+str(s)
    lenMsg = len(msg)
    Msg = "00"+str(lenMsg)+" "+str(msg)
    f.write(str(Msg)+" \n")
    k = 0
    for k in range(len(ports)):
        client_without_recv(Msg, ports[k], hostname, f)
    return


def find_resultant_IP(self, hopCount, sendTime, countIP, data,
    counter, hostname, ip_List, lenIPs, f):
    lock.acquire()
    words = data.split(" ")
    c = int(words[len(words) - 1])
    k = 0
    ipTemp = []
    for k in range(2, (len(words) - 1)):
        ipTemp.append(words[k])
    if len(ipTemp) == c:
        ip_List[countIP] = ipTemp
    lenIPs = lenIPs + c

    lock.release()

    if countIP == counter:

        # print "Final data base: ", ip_List

        f.write("Final data base: " +str(ip_List)+" \n")

        self.IP_list_user(hopCount, sendTime, ip_List, counter,

            hostname, lenIPs, f)

    return


def IP_list_user(self, hopCount, sendTime, ip_List, counter,

    hostname, lenIPs, f):

    Keys = ip_List.keys()

    if len(Keys) == counter:

        ipTemp = []

        k = 0

        for k in range(len(Keys)):

            if k == 0:

                ips = []

                ips = ip_List[Keys[k]]

                i = 0

                for i in range(len(ips)):

                    l = [ips[i], 1]

                    ipTemp.append(l)

            else:

                ips = []
                ips = ip_List[Keys[k]]
                # print "IPS: ", ips
                i = 0
                for i in range(len(ipTemp)):
                    j = 0
                    for j in range(len(ips)):
                        if ips[j] == ipTemp[i][0]:
                            # print "I am here"
                            c = ipTemp[i][1]
                            c += 1
                            ipTemp[i][1] = c
    '''
    k = None
    for k in ipTemp:
        print k
    '''
    fList = []
    k = 0
    for k in range(len(ipTemp)):
        if int(ipTemp[k][1]) == counter:
            fList.append(ipTemp[k][0])
    print "Final IP-List: "
    f.write("------------------------ \n")
    f.write("Final IP-List: \n")
    k = None
```

```
line = ''
for k in fList:
    f.write(str(k)+" \n")
    line = line +" "+str(k)
f.write("------------------------ \n")
# calculate fp
userPort = 10500
ip_List[counter + 1] = fList
msg = "IPLIST-OR "+str(lenIPs)+" "+str(sendTime)+"
    "+str(hopCount)+line+" "+str(len(fList))
msgL = len(msg)
print "Sending final message to user: "
f.write("Sending final message to user: \n")
Msg = "00"+str(msgL)+" "+str(msg)
print Msg
f.write(str(Msg)+" \n")
client_without_recv(Msg, userPort, hostname, f)
f.write("End of simulation \n")
print "End of simulation"
return
```

APPENDIX D

Test Cases for Multi-Attribute, Caching and Overlapped Ring

This chapter presents the source code for testing Multi-Attribute (ROR) architecture, Caching Mechanism and Overlapped Architecture. The src code mentioned here provides test cases for generating queries to each individual methodologies mentioned above. The src code uses "Python2.7", along with "numpy" and "pandas" packages. The "filename" mentioned as input is the resource list obtained from "ResQue". To execute the src code:

```
sudo chmod +x testmultiring.py
./python testmultiring.py -o hostname -f filename -s segments
```

The hostname is the host name of the machine on which the simulation is to be executed. The entire list of resources are divided into multiple segments (value determined by the user), and the upper limit and lower limit for each individual attribute within a segment is calculated. These attributes along-with their upper limit and lower limit are combined together and passed on as a single query. For example a random query will look like:

```
0032 GET IPLIST NCore 4 2 Tx 1200 800 DSize 200 173 MSize 100 53
```

The query generated will contain all the attributes that is mentioned in the resource list. If the user wants to generate query for a some of the attributes not all that needs to be done manually.

- **Test-Cases for Multi-Attribute query resolution using ROR Architecture, contents for "testmultiring.py"**

```
import sys

import socket

import os

import time

import datetime

import getopt

import random

import pandas as pd

import numpy as np


os.system("clear")


f = open('search.log', 'w')

f.close()

f = open('search.log', 'a')


randserver_port = 10500

buff = 4096

controller_Port = 10000

boostrap_port = 11000

Vals = []

ncore = []

processes_ui = []

Attrb = {}

seg = []
```

```
queryAttrb = {}

querycrAttrb = {}

port_alive = [10000, 10500]

tSent = 0


def cal_time():

    ts = time.time()

    st = datetime.datetime.fromtimestamp(ts).strftime('%H:%M:%S')

    return ts


def client_without_recv(msg, port, hostname, f):

    port = int(port)

    ip = str(socket.gethostbyname(hostname))

    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    sock.connect((ip, port))

    sock.send(msg)

    sock.close()

    return


def client(msg, port, hostname, f):

    port = int(port)

    ip = str(socket.gethostbyname(hostname))

    global buff

    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    sock.connect((ip, port))
    print "Contacting server... "

    sock.send(msg)

    time.sleep(3)

    print "Wait for the reply..."

    msg = sock.recv(buff)

    sock.close()

    return msg


def ip_list(data, f):

    print "IP-List: ", str(data)

    words = data.split(" ")

    print "length: ", len(words)

    return


class search():


    global controller_Port


    def getlist(self, hostname, f):

        msg = "GET LIST"

        msgL = len(msg)

        Msg = "00"+str(msgL)+" "+str(msg)

        f.write("Contacting controller for getting attribute list ... \n")

        f.write(str(Msg) + " \n")

        print "Contacting controller for getting attribute list ..."
        print "Message: "+str(Msg)

        rep = client(Msg, controller_Port, hostname, f)

        f.write("Reply from controller ... \n")

        f.write(str(rep) + " \n")

        print "Reply from controller"

        print rep

        return rep


    def analyse_data(self, dataSet, segment, reply, f):

        print "Analyzing controller data.."

        f.write("Analyzing controller data.. \n")

        msg = reply.strip()

        words = msg.split(" ")

        l = len(words)

        print "Length: "+str(l)

        nAttrb = words[3]

        nIP = words[4]

        k = 5

        c = 0

        while k < (l-1):

            attr = str(words[k])

            k += 1

            minV = str(words[k])

            k += 1

            maxV = str(words[k])
            k += 1
            r = str(words[k]) # r is the range
            Vals.append([minV, maxV, r, attr])
            c +=1
            k += 1

        print "----------------------"
        print "Num of Attributes: "+str(nAttrb)
        print "Num of Resources: "+str(nIP)
        print "----------------------"
        print " Attributes | Upper Limit | Lower Limit | Range "
        k = 0
        for k in range(len(Vals)):
            print str(Vals[k][3])+" | "+str(Vals[k][1])+" |
                "+str(Vals[k][0])+" | "+str(Vals[k][2])
        print "----------------------"
        f.write("-------------------- \n")
        f.write("Num of Attributes: "+str(nAttrb)+" \n")
        f.write("Num of Resources: "+str(nIP)+" \n")
        f.write("-------------------- \n")
        k = 0
        f.write(" Attributes | Upper Limit | Lower Limit | Range \n")
        for k in range(len(Vals)):
            f.write(str(Vals[k][3])+" | "+str(Vals[k][1])+" |
                "+str(Vals[k][0])+" | "+str(Vals[k][2])+" \n")
```

```
f.write("------------------- \n")

numLines = dataSet.shape[0]

parts = numLines / segment

# print "Parts: ", parts

k = 0

initSeg = k

newSeg = parts

for k in range(segment):

    partn = [initSeg, newSeg]

    seg.append(partn)

    initSeg = newSeg + 1

    newSeg = parts * (k+2)

if initSeg < numLines:

    partn = [initSeg, numLines - 1]

    seg.append(partn)

    # print "Segments: ", seg

data = dataSet.values

CSp = np.array(data[:,0:1])

NCore = np.array(data[:,1:2])

CFree = np.array(data[:,2:3])

mLd1 = np.array(data[:,3:4])

mLd5 = np.array(data[:,4:5])

mLd15 = np.array(data[:,5:6])

MSize = np.array(data[:,6:7])

MFree = np.array(data[:,7:8])
```

```
DSize = np.array(data[:,8:9])

DFree = np.array(data[:,9:10])

Rx = np.array(data[:,10:11])

Tx = np.array(data[:,11:12])

Attrb['Csp'] = CSp

Attrb['NCore'] = NCore

Attrb['CFree'] = CFree

Attrb['1mLd'] = mLd1

Attrb['5mLd'] = mLd5

Attrb['15mLd'] = mLd15

Attrb['MSize'] = MSize

Attrb['MFree%'] = MFree

Attrb['DSize'] = DSize

Attrb['DFree'] = DFree

Attrb['Rx'] = Rx

Attrb['Tx'] = Tx

k = 0

for k in range(len(seg)):

    line = ''

    lowerLimit = seg[k][0]

    upperLimit = seg[k][1]

    r1 = float(min(np.array(data[lowerLimit:upperLimit,0:1])))

    r2 = float(max(np.array(data[lowerLimit:upperLimit,0:1])))

    line = line +" "+"CSp "+str(r2)+" "+str(r1)

    r1 = float(min(np.array(data[lowerLimit:upperLimit,1:2])))
```

```
r2 = float(max(np.array(data[lowerLimit:upperLimit,1:2])))

line = line +" "+"NCore "+str(r2)+" "+str(r1)

r1 = float(min(np.array(data[lowerLimit:upperLimit,2:3])))

r2 = float(max(np.array(data[lowerLimit:upperLimit,2:3])))

line = line +" "+"CFree "+str(r2)+" "+str(r1)

r1 = float(min(np.array(data[lowerLimit:upperLimit,3:4])))

r2 = float(max(np.array(data[lowerLimit:upperLimit,3:4])))

line = line +" "+"1mLd "+str(r2)+" "+str(r1)

r1 = float(min(np.array(data[lowerLimit:upperLimit,4:5])))

r2 = float(max(np.array(data[lowerLimit:upperLimit,4:5])))

line = line +" "+"5mLd "+str(r2)+" "+str(r1)

r1 = float(min(np.array(data[lowerLimit:upperLimit,5:6])))

r2 = float(max(np.array(data[lowerLimit:upperLimit,5:6])))

line = line +" "+"15mLd "+str(r2)+" "+str(r1)

r1 = float(min(np.array(data[lowerLimit:upperLimit,6:7])))

r2 = float(max(np.array(data[lowerLimit:upperLimit,6:7])))

line = line +" "+"MSize "+str(r2)+" "+str(r1)

r1 = float(min(np.array(data[lowerLimit:upperLimit,7:8])))

r2 = float(max(np.array(data[lowerLimit:upperLimit,7:8])))

line = line +" "+"MFree% "+str(r2)+" "+str(r1)

r1 = float(min(np.array(data[lowerLimit:upperLimit,8:9])))

r2 = float(max(np.array(data[lowerLimit:upperLimit,8:9])))

line = line +" "+"DSize "+str(r2)+" "+str(r1)

r1 = float(min(np.array(data[lowerLimit:upperLimit,9:10])))

r2 = float(max(np.array(data[lowerLimit:upperLimit,9:10])))
```

```
        line = line +" "+"DFree "+str(r2)+" "+str(r1)

        r1 = float(min(np.array(data[lowerLimit:upperLimit,10:11])))

        r2 = float(max(np.array(data[lowerLimit:upperLimit,10:11])))

        line = line +" "+"Rx "+str(r2)+" "+str(r1)

        r1 = float(min(np.array(data[lowerLimit:upperLimit,11:12])))

        r2 = float(max(np.array(data[lowerLimit:upperLimit,11:12])))

        line = line +" "+"Tx "+str(r2)+" "+str(r1)

        line = line[1:]

        queryAttrb[k+1] = line

    k = 0

    for k in range(len(seg)):

        line = ''

        lowerLimit = seg[k][0]

        upperLimit = seg[k][1]

        r1 = float(min(np.array(data[lowerLimit:upperLimit,3:4])))

        r2 = float(max(np.array(data[lowerLimit:upperLimit,3:4])))

        line = line +" "+"1mLd "+str(r2)+" "+str(r1)

        r1 = float(min(np.array(data[lowerLimit:upperLimit,4:5])))

        r2 = float(max(np.array(data[lowerLimit:upperLimit,4:5])))

        line = line +" "+"5mLd "+str(r2)+" "+str(r1)

        r1 = float(min(np.array(data[lowerLimit:upperLimit,5:6])))

        r2 = float(max(np.array(data[lowerLimit:upperLimit,5:6])))

        line = line +" "+"15mLd "+str(r2)+" "+str(r1)

        r1 = float(min(np.array(data[lowerLimit:upperLimit,8:9])))

        r2 = float(max(np.array(data[lowerLimit:upperLimit,8:9])))
```

```
            line = line +" "+"DSize "+str(r2)+" "+str(r1)
            r1 = float(min(np.array(data[lowerLimit:upperLimit,9:10])))
            r2 = float(max(np.array(data[lowerLimit:upperLimit,9:10])))
            line = line +" "+"DFree "+str(r2)+" "+str(r1)
            r1 = float(min(np.array(data[lowerLimit:upperLimit,10:11])))
            r2 = float(max(np.array(data[lowerLimit:upperLimit,10:11])))
            line = line +" "+"Rx "+str(r2)+" "+str(r1)
            r1 = float(min(np.array(data[lowerLimit:upperLimit,11:12])))
            r2 = float(max(np.array(data[lowerLimit:upperLimit,11:12])))
            line = line +" "+"Tx "+str(r2)+" "+str(r1)
            line = line[1:]
            querycrAttrb[k+1] = line
        return

    def userInterface(self, hostname, segment, f):
        while True:
            print "    ---------------------   "
            print "      1  ->  LOOKUP   "
            print "      2  ->  LOOKUP CR   "
            print "      3  ->  EXIT     "
            print "    ---------------------   "
            ch = int(raw_input("Enter your choice: "))
            if ch == 3:
                self.exitMessage(hostname, f)
            elif ch == 1:
```

```
            self.lookup_list(hostname, segment, f)

        elif ch == 2:

            self.lookup_list_cr(hostname, segment, f)

        else:

            print "Unknown choice, try again !"


def lookup_list_cr(self, hostname, segment, f):

    msg = "EXIT"

    msgl = len(msg)

    Msg = str(msgl)+" "+str(msg)

    client_without_recv(Msg, boostrap_port, hostname, f)

    print "Here are the look up options:"

    print "------------------------------------"

    print "Num of segments for query: ", len(seg)

    print "Parts: 1, 2, ... , "+str(len(seg))

    print "------------------------------------"

    print "Query format: Attr_1 <u_1 l_1> Attr_2 <u_2 l_2> ..
     Attr_n <u_n l_n>"

    tSent = cal_time()

    choice = int(raw_input("Enter your segment choice: "))

    print "Contacting controller for IP List"

    msg = "GET IPLIST 0 "+str(querycrAttrb[choice])+" "+str(tSent)

    msgL = len(msg)

    Msg = "0"+str(msgL)+" "+str(msg)

    print "Sending Msg: ", Msg
    client_without_recv(Msg, controller_Port, hostname, f)
    return


def lookup_list(self, hostname, segment, f):
    msg = "EXIT"
    msgl = len(msg)
    Msg = str(msgl)+" "+str(msg)
    client_without_recv(Msg, boostrap_port, hostname, f)
    print "Here are the look up options:"
    print "------------------------------------"
    print "Num of segments for query: ", len(seg)
    print "Parts: 1, 2, ... , "+str(len(seg))
    print "------------------------------------"
    print "Query format: Attr_1 <u_1 l_1> Attr_2 <u_2 l_2> ..
     Attr_n <u_n l_n>"
    tSent = cal_time()
    choice = int(raw_input("Enter your segment choice: "))
    print "Contacting controller for IP List"
    msg = "GET IPLIST 0 "+str(queryAttrb[choice])+" "+str(tSent)
    msgL = len(msg)
    Msg = "0"+str(msgL)+" "+str(msg)
    print "Sending Msg: ", Msg
    client_without_recv(Msg, controller_Port, hostname, f)
    return
    def exitMessage(self, hostname, f):
        msg = "EXIT"
        num = 2
        k = 0
        for k in range(num):
            msgl = len(msg)
            Msg = str(msgl)+" "+str(msg)
            port = int(port_alive[k])
            client_without_recv(Msg, port, hostname, f)
        print "Exiting myself !"
        f.write("Exiting myself ! \n")
        exit(0)
        return

s = search()

def main(argv):
    f.write('--- Shibayan: Thesis Multi-Attribute Query
    Resolution ---\n')
    f.write('--- Search Log --- '+time.strftime("%c")+' ---\n')
    global myPort
    global buff

    try:
```

```
    opts, args = getopt.getopt(argv,"ho:f:s:",["ifile=", "hostname=",
        "segment="])
except getopt.GetoptError:
    print 'python testmultiring.py -o hostname -f filename -s
        segments'
    sys.exit(2)
for opt, arg in opts:
    if opt == '-h':
        print 'python testmutliring.py -o <hostname> -f <filename> -s
            <segments>'
        sys.exit()
    elif opt in ("-o", "--hostname"):
        hostname = arg
    elif opt in ("-f", "--ifile"):
        filename = arg
    elif opt in ("-s", "--segment"):
        segment = arg

print "Filename: ", filename
segment = int(segment)
dataSet = pd.read_csv(filename, sep = '\t')
print "Dataset shape: ", dataSet.shape
nAttrb = dataSet.shape[1]
nAttrb = int(nAttrb)
reply = s.getlist(hostname, f)
```

```
    s.analyse_data(dataSet, segment, reply, f)

    s.userInterface(hostname, segment, f)


if __name__ == "__main__":

    main(sys.argv[1:])
```

The src code mentioned here provides test cases for generating queries related to correlated attributes only. These queries are resolved only by caching and overlapped ring architecture. The src code uses "Python2.7", along with "numpy" and "pandas" packages. The "filename" mentioned as input is the resource list obtained from "ResQue". To execute the src code:

```
sudo chmod +x testcorrelated.py

./python testcorrelated.py -o hostname -f filename -s segments
```

The hostname is the host name of the machine on which the simulation is to be executed. The entire list of resources are divided into multiple segments (value determined by the user), and the upper limit and lower limit for each individual attribute within a segment is calculated. These attributes along-with their upper limit and lower limit are combined together and passed on as a single query. For example a random query will look like:

```
0032 GET IPLIST Tx 1200 800 Rx 1100 700 DSize 200 173 DFree 100 53
```

The query generated will contain all the correlated attributes that is mentioned in the resource list. If the user wants to generate query for a some of the attributes not all that needs to be done manually.

- **Test-Cases for Caching and Overlapped Ring Architecture, contents for "testcorrelated.py"**

```
# This test case is used for Caching & Overlapped Ring Architectures
# The queries are generated at random for the same resource list used
# for the above mentioned mechanisms
import sys
import socket
import os
import time
import datetime
import getopt
import random
import pandas as pd
import numpy as np

os.system("clear")


f = open('search.log', 'w')
f.close()
f = open('search.log', 'a')


randserver_port = 10500
```

```
buff = 4096

controller_Port = 10000

Vals = []

lineNum = []

Attrb = {}

queryAttrb = {}

seg = []

portAlive = [10000, 10500]


def cal_time():

    ts = time.time()

    st = datetime.datetime.fromtimestamp(ts).strftime('%H:%M:%S')

    return ts


def client_without_recv(msg, port, hostname, f):

    port = int(port)

    ip = str(socket.gethostbyname(hostname))

    global buff

    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    sock.connect((ip, port))

    print "Contacting server... "

    sock.send(msg)

    time.sleep(3)

    print "Wait for the reply, in the output screen"

    print "Getting back to to user screen"
    sock.close()

    return


def exitMessage(hostname, f):

    msg = "EXIT"

    num = 2

    k = 0

    for k in range(num):

        msgl = len(msg)

        Msg = str(msgl)+" "+str(msg)

        port = int(portAlive[k])

        client_without_recv(Msg, port, hostname, f)

    print "Exiting myself !"

    f.write("Exiting myself ! \n")

    exit(0)

    return


def client(msg, port, hostname, f):

    port = int(port)

    ip = str(socket.gethostbyname(hostname))

    global buff

    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    sock.connect((ip, port))

    print "Contacting server... "

    sock.send(msg)
    time.sleep(3)

    print "Wait for the reply..."

    msg = sock.recv(buff)

    sock.close()

    return msg


def getlist(hostname, f):

    msg = "GET LIST"

    msgL = len(msg)

    Msg = "00"+str(msgL)+" "+str(msg)

    f.write("Contacting controller for getting attribute list ... \n")

    f.write(str(Msg) + " \n")

    print "Contacting controller for getting attribute list ..."

    print "Message: "+str(Msg)

    rep = client(Msg, controller_Port, hostname, f)

    f.write("Reply from controller ... \n")

    f.write(str(rep) + " \n")

    print "Reply from controller"

    print rep

    return rep


def analyse_data(dataSet, segment, reply, f):

    print "Analyzing controller data.."

    f.write("Analyzing controller data.. \n")

    msg = reply[:-1]
```

```
words = msg.split(" ")
l = len(words)
print "Length: "+str(l)
k = 4
while k < (l-1):
    attr = str(words[k])
    k += 1
    maxV = str(words[k])
    k += 1
    minV = str(words[k])
    k += 1
    r = float(maxV) - float(minV)
    Vals.append([attr, maxV, minV, r])

print "----------------------"
f.write("------------------- \n")
print "Num of Attributes: "+str(len(Vals))
f.write("Num of Attributes: "+str(len(Vals))+" \n")
print "----------------------"
f.write("------------------- \n")
print " Attributes | Upper Limit | Lower Limit | Range "
f.write(" Attributes | Upper Limit | Lower Limit | Range \n")
k = 0
for k in range(len(Vals)):
    print str(Vals[k][0])+" | "+str(Vals[k][1])+" |
        "+str(Vals[k][2])+" | "+str(Vals[k][3])
    f.write(str(Vals[k][0])+" | "+str(Vals[k][1])+" |
        "+str(Vals[k][2])+" | "+str(Vals[k][3])+" \n")
print "-----------------------"
f.write("------------------ \n")
numLines = dataSet.shape[0]
parts = numLines / segment
# print "Parts: ", parts
k = 0
initSeg = k
newSeg = parts
for k in range(segment):
    partn = [initSeg, newSeg]
    seg.append(partn)
    initSeg = newSeg + 1
    newSeg = parts * (k+2)
if initSeg < numLines:
    partn = [initSeg, numLines - 1]
    seg.append(partn)
# print "Segments: ", seg
data = dataSet.values
mLd1 = np.array(data[:,3:4])
mLd5 = np.array(data[:,4:5])
mLd15 = np.array(data[:,5:6])
```

```
DSize = np.array(data[:,8:9])

DFree = np.array(data[:,9:10])

Rx = np.array(data[:,10:11])

Tx = np.array(data[:,11:12])

Attrb['mLd1'] = mLd1

Attrb['mLd5'] = mLd5

Attrb['mLd15'] = mLd15

Attrb['DSize'] = DSize

Attrb['DFree'] = DFree

Attrb['Rx'] = Rx

Attrb['Tx'] = Tx

k = 0

for k in range(len(seg)):

    line = ''

    lowerLimit = seg[k][0]

    upperLimit = seg[k][1]

    r1 = float(min(np.array(data[lowerLimit:upperLimit,3:4])))

    r2 = float(max(np.array(data[lowerLimit:upperLimit,3:4])))

    line = line +" "+"mLd1 "+str(r2)+" "+str(r1)

    r1 = float(min(np.array(data[lowerLimit:upperLimit,4:5])))

    r2 = float(max(np.array(data[lowerLimit:upperLimit,4:5])))

    line = line +" "+"mLd5 "+str(r2)+" "+str(r1)

    r1 = float(min(np.array(data[lowerLimit:upperLimit,5:6])))

    r2 = float(max(np.array(data[lowerLimit:upperLimit,5:6])))

    line = line +" "+"mLd15 "+str(r2)+" "+str(r1)
```

```
        r1 = float(min(np.array(data[lowerLimit:upperLimit,8:9])))

        r2 = float(max(np.array(data[lowerLimit:upperLimit,8:9])))

        line = line +" "+"DSize "+str(r2)+" "+str(r1)

        r1 = float(min(np.array(data[lowerLimit:upperLimit,9:10])))

        r2 = float(max(np.array(data[lowerLimit:upperLimit,9:10])))

        line = line +" "+"DFree "+str(r2)+" "+str(r1)

        r1 = float(min(np.array(data[lowerLimit:upperLimit,10:11])))

        r2 = float(max(np.array(data[lowerLimit:upperLimit,10:11])))

        line = line +" "+"Rx "+str(r2)+" "+str(r1)

        r1 = float(min(np.array(data[lowerLimit:upperLimit,11:12])))

        r2 = float(max(np.array(data[lowerLimit:upperLimit,11:12])))

        line = line +" "+"Tx "+str(r2)+" "+str(r1)

        line = line[1:]

        # print "Query Line:", line

        queryAttrb[k+1] = line

    return


def lookup_list(hostname, segments, f):

    print "Here are the look up options:"

    print "-------------------------------------"

    print "Num of segments for query: ", len(seg)

    print "Parts: 1, 2, ... , "+str(len(seg))

    print "-------------------------------------"

    print "Query format: Attr_1 <u_1 l_1> Attr_2 <u_2 l_2> ... Attr_n
        <u_n l_n>"
    tSent = cal_time()

    choice = int(raw_input("Enter your segment choice: "))

    msg = "GET IPLIST 0 "+str(tSent)+" "+str(queryAttrb[choice])

    msgL = len(msg)

    Msg = "0"+str(msgL)+" "+str(msg)

    print "Sending Msg: ", Msg

    client_without_recv(Msg, controller_Port, hostname, f)

    return


def userInterface(hostname, segment, f):

    while True:

        print "    ---------------------    "

        print "      1  ->  LOOKUP    "

        print "      2  ->  EXIT      "

        print "    ---------------------    "

        ch = int(raw_input("Enter your choice: "))

        if ch == 2:

            exitMessage(hostname, f)

        elif ch == 1:

            lookup_list(hostname, segment, f)

        else:

            print "Unknown choice, try again !"


def main(argv):
```

```
f.write('--- Shibayan: Thesis Correlated-Attribute Query Resolution
    ---\n')
f.write('--- Search Log --- '+time.strftime("%c")+' ---\n')
global myPort
global buff

try:
    opts, args = getopt.getopt(argv,"ho:f:s:",["ifile=", "hostname=",
        "segment="])
except getopt.GetoptError:
    print 'python testcorrelated.py -o hostname -f filename -s
        segments'
    sys.exit(2)
for opt, arg in opts:
    if opt == '-h':
        print 'python testcorrelated.py -o <hostname> -f <filename> -s
            <segments>'
        sys.exit()
    elif opt in ("-o", "--hostname"):
        hostname = arg
    elif opt in ("-f", "--ifile"):
        filename = arg
    elif opt in ("-s", "--segment"):
        segment = arg
    print "Filename: ", filename

    segment = int(segment)

    dataSet = pd.read_csv(filename, sep = '\t')

    print "Dataset shape: ", dataSet.shape

    nAttrb = dataSet.shape[1]

    nAttrb = int(nAttrb)

    reply = getlist(hostname, f)

    analyse_data(dataSet, segment, reply, f)

    userInterface(hostname, segment, f)


if __name__ == "__main__":

    main(sys.argv[1:])
```

APPENDIX E

Bootstrap Server, Hash-Value Generation Source Code

This chapter presents the source code for Bootstrap Server, Hash-Value generation, and reading the resource list. The Bootstrap server provides the information regarding what all nodes are already present in the network for ROR architecture, Caching, and Overlapped Architecture to be formed. The Bootstrap server source code is executed as:

```
sudo chmod +x bootstrap.py

sudo ./bootstrap.py -n <num of nodes in network> -o <hostname>
```

- **Bootstrap Server src code, content for "bootstrap.py"**

```
# This Bootstraper Code is essential for all the below architectures:

# Multi-Attribute Query Resolution

# Caching Mechanism

# Over-lapped Architecture


import sys

import socket

import os

import time

import getopt

import random

os.system("clear")

f = open('boostrap.log', 'w')
```

```
f.close()

f = open('boostrap.log', 'a')

recordMR = []

recordSR = []

myPort = 11000

nodes = 0

buff = 1024

def client(msg, port, hostname, f):

    port = int(port)

    ip = str(socket.gethostbyname(hostname))

    # time.sleep(10)

    # Creating TCP Socket

    global buff

    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    sock.connect((ip, port))

    print "Contacting server..."

    sock.send(msg)

    data = sock.recv(buff)

    sock.close()

    return data


def updateMRecord(pName, attrType, portNum, mrID, srID):

    l = len(recordMR)

    if l == 0:

        #print "I am here !"
        time.sleep(2)
        recordMR.append([pName, attrType, portNum, mrID, srID])
        k = None
        #for k in recordMR:
        #    print k
        return "-1"
    else:
        k = len(recordMR)
        i = 0
        time.sleep(2)
        line = ''
        #print "I am here2 !"
        for i in range(k):
            line = line + str(recordMR[i][1])+" "+str(recordMR[i][2])+"
                "+str(recordMR[i][3])+" "
            #print "Line: ",line
        recordMR.append([pName, attrType, portNum, mrID, srID])
        return line


def updateSRecord(pName, attrType, portNum, srID):
    l = len(recordMR)
    k = 0
    line = ''
    for k in range(l):
        if str(recordMR[k][1]) == str(attrType):
            line = line + str(recordMR[k][1])+" "+str(recordMR[k][2])+"
                "+str(recordMR[k][4])+" "
            #print "Line at SR (MR): ", line
    l = len(recordSR)
    if l == 0:
        recordSR.append([pName, attrType, portNum, srID])
        return line
    else:
        k = 0
        for k in range(l):
            if str(attrType) == str(recordSR[k][1]):
                line = line + str(recordSR[k][1])+"
                    "+str(recordSR[k][2])+" "+str(recordSR[k][3])+" "
                #print "Line at SR (SR): ", line
        recordSR.append([pName, attrType, portNum, srID])
        return line


def send_query_to_clients(hostname, data, f):
    l = len(recordMR)
    x = random.randint(0, l)
    # pName, attrType, portNum, mrID, srID
    port = int(recordMR[x][2])
    ip = str(socket.gethostbyname(hostname))
    time.sleep(10)
    # Creating TCP Socket
    print "Creating socket to transfer data, address ->
        "+str(ip)+":"+str(port)
    f.write("Creating socket to transfer data ->
        "+str(ip)+":"+str(port)+" \n")
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    print "Connecting and sending query ..."
    f.write("Connecting and sending query ... \n")
    sock.connect((ip, port))
    sock.send(data)
    print "Query: "+str(data)+ " send to addr: "+str(ip)+":"+str(port)
    f.write("Query: "+str(data)+ " send to addr:
        "+str(ip)+":"+str(port)+" \n")
    return

def main(argv):
    f.write('--- Shibayan: Thesis Multi-Attribute Query Resolution ---\n')
    f.write('--- Boostrap Log --- '+time.strftime("%c")+' ---\n')
    global nodes
    global myPort
    global buff

    try:
        opts, args = getopt.getopt(argv,"hn:o:",["nodes=","hostname="])
    except getopt.GetoptError:
```

```
    print 'python bootstrap.py -n <num of nodes in network> -o
        <hostname>'
    sys.exit(2)
for opt, arg in opts:
    if opt == '-h':
        print 'python bootstrap.py -n <num of nodes in network> -o
            <hostname>'
        sys.exit()
    elif opt in ("-n", "--nodes"):
        nodes = arg
    elif opt in ("-o", "--hostname"):
        hostname = arg

nodes = int(nodes)
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
ip = str(socket.gethostbyname(hostname))
# Binding the socket to the port
server_address = (ip, myPort)
sock.bind(server_address)
print "Boostrap Server Address: "+str(server_address)
f.write('Boostrap Server Address: '+str(server_address)+' \n')
# Listening for incoming connections
sock.listen(nodes)
while True:
    conn, addr = sock.accept()
```

```
data = conn.recv(buff)

#print "Got Connection from: "+str(conn)

print "----- "+str(addr)+" --> Boostrap Server -----"

print " Message: "+str(data)

f.write("----- "+str(addr)+" --> Boostrap Server ----- \n")

f.write("Message: "+str(data)+" \n")

words = data.split(" ")

word = words[1]

if word == "REG":

    rtype = words[3]

    if rtype == "MR":

        pName = words[2]

        attrType = words[4]

        portNum = words[5]

        mrID = words[6]

        srID = words[7]

        val = updateMRecord(pName, attrType, portNum, mrID, srID)

        if val == "-1":

            msg = "REGOK FIRST"

            msgL = len(msg)

            Msg = "00"+str(msgL)+" "+str(msg)

            f.write("----- Boostrap Server -->

            "+str(addr)+" ----- \n")

            f.write("Message: "+str(Msg)+" \n")

            print "----- Boostrap Server -->
            "+str(addr)
            print Msg
            conn.send(Msg)
        else:
            msg = "REGOK"+" "+val
            msgL = len(msg)
            Msg = "00"+str(msgL)+" "+str(msg)
            f.write("----- Boostrap Server -->
            "+str(addr)+" ----- \n")
            f.write("Message: "+str(Msg)+" \n")
            print "----- Boostrap Server -->
            "+str(addr)
            print Msg
            conn.send(Msg)
    if rtype == "SR":
        pName = words[2]
        attrType = words[4]
        portNum = words[5]
        srID = words[6]
        val = updateSRecord(pName, attrType, portNum, srID)
        msg = "REGOK"+" "+val
        msgL = len(msg)
        Msg = "00"+str(msgL)+" "+str(msg)
        f.write("----- Boostrap Server -->
        "+str(addr)+" ----- \n")
```

```
                f.write("Message: "+str(Msg)+" \n")
                print "----- Boostrap Server -->
                "+str(addr)
                print Msg
                conn.send(Msg)
        elif word == "EXIT":
            print "Exiting simulation environment"
            f.write("Exiting simulation environment \n")
            time.sleep(2)
            exit(0)
        else:
            print "Request received: ", str(data)
            msg = "UNKNOWN REQUEST"
            msgL = len(msg)
            Msg = "00"+str(msgL)+" "+str(msg)
            f.write("----- Boostrap Server -->
            "+str(addr)+" ----- \n")
            f.write("Message: "+str(Msg)+" \n")
            print "----- Boostrap Server -->
            "+str(addr)
            print Msg
            conn.send(Msg)
    return


if __name__ == "__main__":
    main(sys.argv[1:])
```

- **Resource Hash Value generation src code, contents for "hashVal.py"**

```
# This file is used for Hash-Value Calculation of the resources for all
    architectures:
# Multi-Attribute Query Resolution
# Caching
# Over-lapped Ring


import hashlib


def hash_gen(m, k, ip, prime_num, f1):
    hVal = []
    ''' Need to change the parameters of u
        and l if range is not sufficient '''
    lP = len(prime_num)
    f1.write("Prime numbers generated are: " +str(prime_num)+ " \n")
    f1.write("Num of prime numbers: " +str(len(prime_num))+ " \n")
    if int(lP * 5) < int(k):
        print "ERROR: Can't generate hash values, Expand the range "
        f1.write("ERROR: Can't generate hash values, Expand the range \n")
        return -1
    count = 0
    prime_ind = 0
    while (count < k):
```

```
p = int(prime_num[prime_ind])

h = hashlib.md5()

h.update(ip)

p = int(p)

keyVal = int(h.hexdigest(), 32)

keyVal1 = int(keyVal % p)

k1 = str(keyVal1 % m)

hVal.append(k1)


h = hashlib.sha224()

h.update(ip)

p = int(p)

keyVal = int(h.hexdigest(), 32)

keyVal1 = int(keyVal % p)

k2 = str(keyVal1 % m)

hVal.append(k2)


h = hashlib.sha256()

h.update(ip)

p = int(p)

keyVal = int(h.hexdigest(), 32)

keyVal1 = int(keyVal % p)

k3 = str(keyVal1 % m)

hVal.append(k3)
```

```
    h = hashlib.sha384()

    h.update(ip)

    p = int(p)

    keyVal = int(h.hexdigest(), 32)

    keyVal1 = int(keyVal % p)

    k4 = str(keyVal1 % m)

    hVal.append(k4)


    h = hashlib.sha512()

    h.update(ip)

    p = int(p)

    keyVal = int(h.hexdigest(), 32)

    keyVal1 = int(keyVal % p)

    k5 = str(keyVal1 % m)

    hVal.append(k5)


    count = count + 5

    prime_ind = prime_ind + 2

l = len(hVal)

print "Hash values needed: "+str(k)+" and generated: "+str(l)+" for
    ip: "+str(ip)

f1.write("Hash values needed: "+str(k)+" and generated: "+str(l)+"
    \n")

f1.write("Hash Values: "+str(hVal)+" \n")

return hVal
```

```
class hashfunctions:


    def hash_values(self, m, k, ip, prime_num, f1):

        k = int(k)

        m = int(m)

        hval = []

        if k == 1:

            h = hashlib.md5()

            h.update(ip)

            keyVal = int(h.hexdigest(), 32)

            k1 = str(keyVal % m)

            hval = [k1]


        elif k == 2:

            h = hashlib.md5()

            h.update(ip)

            keyVal = int(h.hexdigest(), 32)

            k1 = str(keyVal % m)

            h = hashlib.sha224()

            h.update(ip)

            keyVal = int(h.hexdigest(), 32)

            k2 = str(keyVal % m)

            hval = [k1, k2]
```

```
elif k == 3:

    h = hashlib.md5()

    h.update(ip)

    keyVal = int(h.hexdigest(), 32)

    k1 = str(keyVal % m)

    h = hashlib.sha224()

    h.update(ip)

    keyVal = int(h.hexdigest(), 32)

    k2 = str(keyVal % m)

    h = hashlib.sha256()

    h.update(ip)

    keyVal = int(h.hexdigest(), 32)

    k3 = str(keyVal % m)

    hval = [k1, k2, k3]


elif k == 4:

    h = hashlib.md5()

    h.update(ip)

    keyVal = int(h.hexdigest(), 32)

    k1 = str(keyVal % m)

    h = hashlib.sha224()

    h.update(ip)

    keyVal = int(h.hexdigest(), 32)

    k2 = str(keyVal % m)

    h = hashlib.sha256()
    h.update(ip)
    keyVal = int(h.hexdigest(), 32)
    k3 = str(keyVal % m)
    h = hashlib.sha384()
    h.update(ip)
    keyVal = int(h.hexdigest(), 32)
    k4 = str(keyVal % m)
    hval = [k1, k2, k3, k4]

elif k == 5:
    h = hashlib.md5()
    h.update(ip)
    keyVal = int(h.hexdigest(), 32)
    k1 = str(keyVal % m)
    h = hashlib.sha224()
    h.update(ip)
    keyVal = int(h.hexdigest(), 32)
    k2 = str(keyVal % m)
    h = hashlib.sha256()
    h.update(ip)
    keyVal = int(h.hexdigest(), 32)
    k3 = str(keyVal % m)
    h = hashlib.sha384()
    h.update(ip)
    keyVal = int(h.hexdigest(), 32)
```

```
        k4 = str(keyVal % m)
        h = hashlib.sha512()
        h.update(ip)
        keyVal = int(h.hexdigest(), 32)
        k5 = str(keyVal % m)
        hval = [k1, k2, k3, k4, k5]

    elif k == 6:
        p = prime_num[0]
        h = hashlib.md5()
        h.update(ip)
        keyVal = int(h.hexdigest(), 32)
        k1 = str(keyVal % m)
        k6n = keyVal % p
        k6 = str(k6n % m)
        h = hashlib.sha224()
        h.update(ip)
        keyVal = int(h.hexdigest(), 32)
        k2 = str(keyVal % m)
        h = hashlib.sha256()
        h.update(ip)
        keyVal = int(h.hexdigest(), 32)
        k3 = str(keyVal % m)
        h = hashlib.sha384()
        h.update(ip)
        keyVal = int(h.hexdigest(), 32)
        k4 = str(keyVal % m)
        h = hashlib.sha512()
        h.update(ip)
        keyVal = int(h.hexdigest(), 32)
        k5 = str(keyVal % m)
        hval = [k1, k2, k3, k4, k5, k6]

    elif k == 7:
        p = prime_num[1]
        h = hashlib.md5()
        h.update(ip)
        keyVal = int(h.hexdigest(), 32)
        k1 = str(keyVal % m)
        k6n = keyVal % p
        k6 = str(k6n % m)
        h = hashlib.sha224()
        h.update(ip)
        keyVal = int(h.hexdigest(), 32)
        k2 = str(keyVal % m)
        k7n = keyVal % p
        k7 = str(k7n % m)
        h = hashlib.sha256()
        h.update(ip)
        keyVal = int(h.hexdigest(), 32)
        k3 = str(keyVal % m)
        h = hashlib.sha384()
        h.update(ip)
        keyVal = int(h.hexdigest(), 32)
        k4 = str(keyVal % m)
        h = hashlib.sha512()
        h.update(ip)
        keyVal = int(h.hexdigest(), 32)
        k5 = str(keyVal % m)
        hval = [k1, k2, k3, k4, k5, k6, k7]

    elif k == 8:
        p = prime_num[2]
        h = hashlib.md5()
        h.update(ip)
        keyVal = int(h.hexdigest(), 32)
        k1 = str(keyVal % m)
        k6n = keyVal % p
        k6 = str(k6n % m)
        h = hashlib.sha224()
        h.update(ip)
        keyVal = int(h.hexdigest(), 32)
        k2 = str(keyVal % m)
        k7n = keyVal % p
        k7 = str(k7n % m)
        h = hashlib.sha256()
        h.update(ip)
        keyVal = int(h.hexdigest(), 32)
        k3 = str(keyVal % m)
        k8n = keyVal % p
        k8 = str(k8n % m)
        h = hashlib.sha384()
        h.update(ip)
        keyVal = int(h.hexdigest(), 32)
        k4 = str(keyVal % m)
        h = hashlib.sha512()
        h.update(ip)
        keyVal = int(h.hexdigest(), 32)
        k5 = str(keyVal % m)
        hval = [k1, k2, k3, k4, k5, k6, k7, k8]

    elif k == 9:
        p = prime_num[3]
        h = hashlib.md5()
        h.update(ip)
        keyVal = int(h.hexdigest(), 32)
        k1 = str(keyVal % m)
        k6n = keyVal % p
        k6 = str(k6n % m)
        h = hashlib.sha224()
        h.update(ip)
        keyVal = int(h.hexdigest(), 32)
        k2 = str(keyVal % m)
        k7n = keyVal % p
        k7 = str(k7n % m)
        h = hashlib.sha256()
        h.update(ip)
        keyVal = int(h.hexdigest(), 32)
        k3 = str(keyVal % m)
        k8n = keyVal % p
        k8 = str(k8n % m)
        h = hashlib.sha384()
        h.update(ip)
        keyVal = int(h.hexdigest(), 32)
        k4 = str(keyVal % m)
        k9n = keyVal % p
        k9 = str(k9n % m)
        h = hashlib.sha512()
        h.update(ip)
        keyVal = int(h.hexdigest(), 32)
        k5 = str(keyVal % m)
        hval = [k1, k2, k3, k4, k5, k6, k7, k8, k9]

    elif k == 10:
        p = prime_num[4]
```

```
h = hashlib.md5()

h.update(ip)

keyVal = int(h.hexdigest(), 32)

k1 = str(keyVal % m)

k6n = keyVal % p

k6 = str(k6n % m)

h = hashlib.sha224()

h.update(ip)

keyVal = int(h.hexdigest(), 32)

k2 = str(keyVal % m)

k7n = keyVal % p

k7 = str(k7n % m)

h = hashlib.sha256()

h.update(ip)

keyVal = int(h.hexdigest(), 32)

k3 = str(keyVal % m)

k8n = keyVal % p

k8 = str(k8n % m)

h = hashlib.sha384()

h.update(ip)

keyVal = int(h.hexdigest(), 32)

k4 = str(keyVal % m)

k9n = keyVal % p

k9 = str(k9n % m)

h = hashlib.sha512()
```

```
            h.update(ip)

            keyVal = int(h.hexdigest(), 32)

            k5 = str(keyVal % m)

            k10n = keyVal % p

            k10 = str(k10n % m)

            hval = [k1, k2, k3, k4, k5, k6, k7, k8, k9, k10]


        else:

            if k > 10:

                hval = hash_gen(m, k, ip, prime_num, f1)

        return hval


    def prime_num_gen(self):

        u = 100000

        l = 50000

        prime_num = []

        lower = int(l)

        upper = int(u)

        for num in range(lower,upper + 1):

            # prime numbers are greater than 1

            if num > 1:

                for i in range(2,num):

                    if (num % i) == 0:

                        break

                else:
                    prime_num.append(num)
        return prime_num
```

- **Source code to read the resource list and calculate parameters, contents for "calParam.py"**

```
# This src code is used for reading resource list generated from ResQue
import numpy as np
import pandas as pd
import random
import math
import os
import time
import getopt
import sys
import socket

minVals = []
maxVals = []
rangeVals = []

def average(x):
    assert len(x) > 0
    return float(sum(x)) / len(x)

def pearson_def(x, y):
    assert len(x) == len(y)
    n = len(x)
    assert n > 0
    avg_x = average(x)
    avg_y = average(y)
    diffprod = 0
    xdiff2 = 0
    ydiff2 = 0
    for idx in range(n):
        xdiff = x[idx] - avg_x
        ydiff = y[idx] - avg_y
        diffprod += xdiff * ydiff
        xdiff2 += xdiff * xdiff
        ydiff2 += ydiff * ydiff

    return diffprod / math.sqrt(xdiff2 * ydiff2)

class calParam:

    def readFile(self, filename, Attrb, f):
        dataSet = pd.read_csv(filename, sep = '\t')
        print "Dataset shape: ", dataSet.shape
        nAttrb = dataSet.shape[1]
        nIP = dataSet.shape[0]
        print "Num of attribute: ", nAttrb
        f.write("Dataset shape: "+str(dataSet.shape)+" Num of attribute:
            "+str(nAttrb)+" \n")
        data = dataSet.values
        mLd1 = np.array(data[:,3:4])
        mLd5 = np.array(data[:,4:5])
        mLd15 = np.array(data[:,5:6])
        DSize = np.array(data[:,8:9])
        DFree = np.array(data[:,9:10])
        Rx = np.array(data[:,10:11])
        Tx = np.array(data[:,11:12])
        Attrb['mLd1'] = mLd1
        Attrb['mLd5'] = mLd5
        Attrb['mLd15'] = mLd15
        Attrb['DSize'] = DSize
        Attrb['DFree'] = DFree
        Attrb['Rx'] = Rx
        Attrb['Tx'] = Tx
        return (Attrb, nIP)

    def calCorrelation(self, Attrb, f):
        corr_attr = []
        f.write('\n')
        f.write('----------- Correlation Calculation --------- \n')
        f.write('# Attributes with correlation value more than 0.90 will
            be considered correlated \n')
    f.write('\n')
    keyList = Attrb.keys()
    i = None
    j = None
    for i in keyList:
        firstList = Attrb.get(i)
        for j in keyList:
            if j != i:
                secondList = Attrb.get(j)
                corCoeff = pearson_def(firstList, secondList)
                if corCoeff > 0.85:
                    f.write('Attribute #: '+str(i)+ ' Attribute #:
                        '+str(j)+' Correlation: '+str(corCoeff)+'\n')
                    print str(i)+" "+str(j)+" "+str(corCoeff)
                    corr_attr.append([i, j])
    return corr_attr
```

- **To assign locations to each individual nodes in the ROR architecture**

```
import numpy as np
import pandas as pd
import random
import math
import os
import time
import getopt
import sys

import socket


workAssgnMainID = []

workAssgnSubID = []


class locAssgn:


def bitspaceAlloc(self, spaceMain, spaceSubRing, nNodes, nAttr, f):

       diff = nNodes - nAttr

       idMainRing = random.sample(xrange(spaceMain), nAttr)

       idMainSubRing = random.sample(xrange(spaceSubRing), nAttr)

       idSubRing = random.sample(xrange(spaceSubRing), diff)

       f.write('------------------ Main Ring IDs

           ---------------------\n')

       f.write('Main Ring IDs: '+ str(idMainRing)+' \n')

       f.write('Main-Sub Ring Member IDs: '+ str(idMainSubRing)+' \n')

       f.write('Sub Ring IDs: '+ str(idSubRing)+' \n')

       f.write('------------------------------------------------------\n')

       return (idMainRing, idMainSubRing, idSubRing)


def workAllocation(self, idMainRing, idMainSubRing, idSubRing, Attrb, f):

       f.write('---------------- Work Allocation

           ---------------------\n')

       print "\n"
```

```
keyList = Attrb.keys()

i = 0

k = None

for k in keyList:

       workAssgnMainID.append([str(k), str(idMainRing[i]),

           str(idMainSubRing[i])])

       i = i + 1

l = len(idSubRing)

kl = len(keyList)


if l == kl:

       k = None

       i = 0

for k in keyList:

       workAssgnSubID.append([str(idSubRing[i]), str(k)])

       i = i + 1


print "------------------- Work Allocation ----------------"

print "Attribute -> Main Ring mc ID -> Main-Sub Ring mc ID ..."

f.write('Attribute -> Main Ring mc ID -> Main-Sub Ring ID \n')

k = 0

for k in range(kl):

       line = str(workAssgnMainID[k][0])+" ->

           "+str(workAssgnMainID[k][1])+" ->

           "+str(workAssgnMainID[k][2])
```

```
        print line
        f.write(line+' \n')

print "Attribute -> Sub Ring mc ID ..."
f.write('Attribute -> Sub Ring ID \n')
k = 0
for k in range(kl):
        line = str(workAssgnSubID[k][1]) + " ->
            "+str(workAssgnSubID[k][0])
        print line
        f.write(line+' \n')

if l > kl:
        k = None
        i = 0
        for k in keyList:
                workAssgnSubID.append([str(idSubRing[i]), str(k)])
        i = i + 1

if (i < l):
        diff = l - i
        k = 0
        val = []
        for k in range(diff):
                v = int(random.uniform(0, kl))
                val.append(str(keyList[v]))


#print "Val: ", val

k = 0

for k in range(diff):

workAssgnSubID.append([str(idSubRing[i]), str(val[k])])

i = i + 1


k = 0

print "----- Work Allocation ------"

print "Attribute -> Main Ring mc ID -> Main-Sub Ring mc ID ..."

f.write('Attribute -> Main Ring mc ID -> Main-Sub Ring mc ID \n')

for k in range(kl):

line = str(workAssgnMainID[k][0])+" ->

   "+str(workAssgnMainID[k][1])+" -> "+str(workAssgnMainID[k][2])

print line

f.write(line+' \n')


print "Attribute -> Sub Ring mc IDs ..."

f.write('Attribute -> Sub Ring mc IDs \n')

k = 0

for k in workAssgnSubID:

print k

f.write(str(k)+' \n')
```

```
print "---------------------------\n"
f.write('-------------------------\n')
return (workAssgnMainID, workAssgnSubID)


def generateIP(self, ipCount, IPAddress, f):
    k = 0
    for k in range(ipCount):
        ip1 = str(random.randint(100,255))
        ip2 = str(random.randint(100,255))
        ip3 = str(random.randint(100,255))
        ip4 = str(random.randint(100,255))
        ipLine = ip1+"."+ip2+"."+ip3+"."+ip4
        IPAddress.append(ipLine)


    l = len(IPAddress)
    print "Num of IP Addresses created: ", str(l)
    print str(IPAddress[0]) + " ..... " + IPAddress[l-1] + "\n"
    f.write('\n')
    f.write('------- IP Address creation -------\n')
    f.write('Num of IP Addresses created: '+str(l)+' \n')
    f.write(IPAddress[0]+"/"+IPAddress[1]+"/"+IPAddress[2]
    + "/ ..... /"+IPAddress[l-3]+"/"+IPAddress[l-2]+
    "/"+IPAddress[l-1]+' \n')
    f.write('-----------------------------------\n')
    return IPAddress
```

```
def finalDataset(self, Attrb, IPAddress, f):

        Attrb['IPAddress'] = IPAddress

        keysAttrb = Attrb.keys()

        f.write('\n')

        f.write('Updated Attributes: '+str(keysAttrb) + ' \n')

        createNewFile(Attrb, keysAttrb, f)

        return Attrb


def createNewFile(Attrb, keyAttrb, f):
'''
Creating a new file named derived-dataset.txt

for final cross verification, no need

but still creating
'''
        num_keys = len(keyAttrb)

        print "Number of keys present in the dataset:

            "+str(num_keys)

        f.write("Number of keys present in the dataset:

            "+str(num_keys)+" \n")

        df = open('derived-dataset.txt', 'w')

        df.write(str(keyAttrb)+ " \n")

        df.close()

        df = open('derived-dataset.txt', 'a')

        df.close()
```

```
              return
```

- **To calculate parameter for each individual attribute**

```
import numpy as np

import pandas as pd

import random

import math

import os

import time

import getopt

import sys

import socket


minVals = []

maxVals = []

rangeVals = []


def average(x):

assert len(x) > 0

return float(sum(x)) / len(x)


def pearson_def(x, y):

assert len(x) == len(y)

n = len(x)

assert n > 0
```

```
avg_x = average(x)

avg_y = average(y)

diffprod = 0

xdiff2 = 0

ydiff2 = 0

for idx in range(n):

xdiff = x[idx] - avg_x

ydiff = y[idx] - avg_y

diffprod += xdiff * ydiff

xdiff2 += xdiff * xdiff

ydiff2 += ydiff * ydiff


return diffprod / math.sqrt(xdiff2 * ydiff2)


class calParam:


def readFile(self, filename, numNodes, mainRSpace, Attrb, f):

dataSet = pd.read_csv(filename, sep = '\t')

header = list(dataSet.columns.values)

print "The headers are: ", str(header)

print "Dimension of dataset: "+str(dataSet.shape[0])+" X

    "+str(dataSet.shape[1])

f.write('\nDataset.shape: '+ str(dataSet.shape[0])+' X

    '+str(dataSet.shape[1])+ '\n')

nAttrb = dataSet.shape[1]
```

```
nIP = dataSet.shape[0]

print "Num of attributes: ", nAttrb

data = dataSet.values

f.write('Number of attributes: ' + str(nAttrb)+ '\n')

if nAttrb > numNodes:

f.write('Network topology doesnt have enough machines\n')

f.write('Exiting..!!')

return -1


if (mainRSpace < nAttrb) or (mainRSpace < numNodes):

print "Num. of attrb. are more than network capacity..!!"

print "Exiting..!!"

f.write('Num. of attrb. are more than network capacity..!! \n')

return -1


if (numNodes < 2*nAttrb):

print "At-least two nodes should be present for each attribute..!!"

print "Exiting..!!"

f.write('At-least two nodes should be present for each attribute..!! \n')

return -1


i = 0

f.write('---------------------------------------------\n')

for i in range(len(header)):

line = str(header[i])
```

```
Attrb.update({line:data[:, i: i+1]})


keyList = Attrb.keys()

f.write('Attrb-Type: '+str(keyList)+'\n')

line_params = " "


k = None

for k in keyList:

currentList = Attrb.get(k)

minVal = currentList.min()

minVals.append(minVal)

maxVal = currentList.max()

maxVals.append(maxVal)

rangeVal = maxVal - minVal

rangeVals.append(rangeVal)

f.write(str(k)+" [ "+str(minVal)+" "+str(maxVal)+" ] Range:

    "+str(rangeVal)+' \n')

line_params = line_params + str(k)+" "+str(minVal)+" "+str(maxVal)+"

    "+str(rangeVal)+ " "

f.write('---------------------------------------------\n')

return (nAttrb, nIP, line_params, header)


def calCorrelation(self, Attrb, f):

corr_attr = []

f.write('\n')
```

```
f.write('----------- Correlation Calculation --------- \n')
f.write('# Attributes with correlation value more than 0.90 will be
   considered correlated \n')
f.write('\n')
keyList = Attrb.keys()
i = None
j = None
for i in keyList:
firstList = Attrb.get(i)
for j in keyList:
if j != i:
secondList = Attrb.get(j)
corCoeff = pearson_def(firstList, secondList)
if corCoeff > 0.90:
f.write('Attribute #: '+str(i)+ ' Attribute #: '+str(j)+' Correlation:
   '+str(corCoeff)+'\n')
corr_attr.append([i, j])
print "Correlated attributes: "
f.write("Correlated attributes: \n")
k = None
for k in corr_attr:
print k
f.write(str(k)+" \n")
f.write('---------------------------------------------\n')
return corr_attr
```