# XtreemOS WP3.2 - T3.2.3
# Scalable Directory Service Design
# State of Arts and Proposals


**Martina Baldanzi, Massimo Coppola, Domenico Laforenza, Laura Ricci**

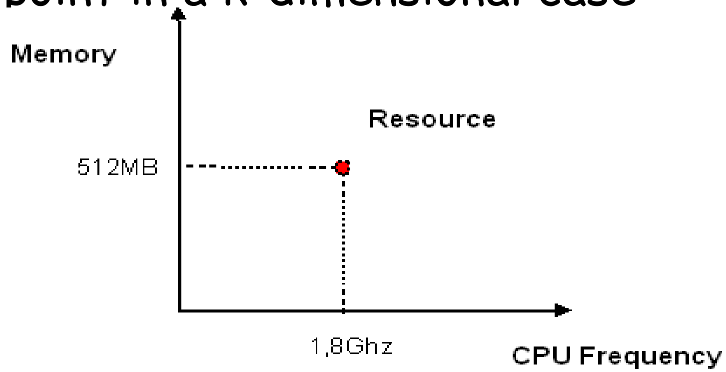
**ISTI/CNR Pisa**



XtreemOS IP project is funded by the European Commission under contract IST-FP6-033576

# DISTRIBUTED DIRECTORY SERVICES

Consider a system where any resources R is defined by k attributes. R corresponds to a point in a k-dimensional case




Directory services (DS) = A service returning the name(s) or address(es) of all the items (resources) characterized by a k given values of the attributes.

Some functionalities of the DS:

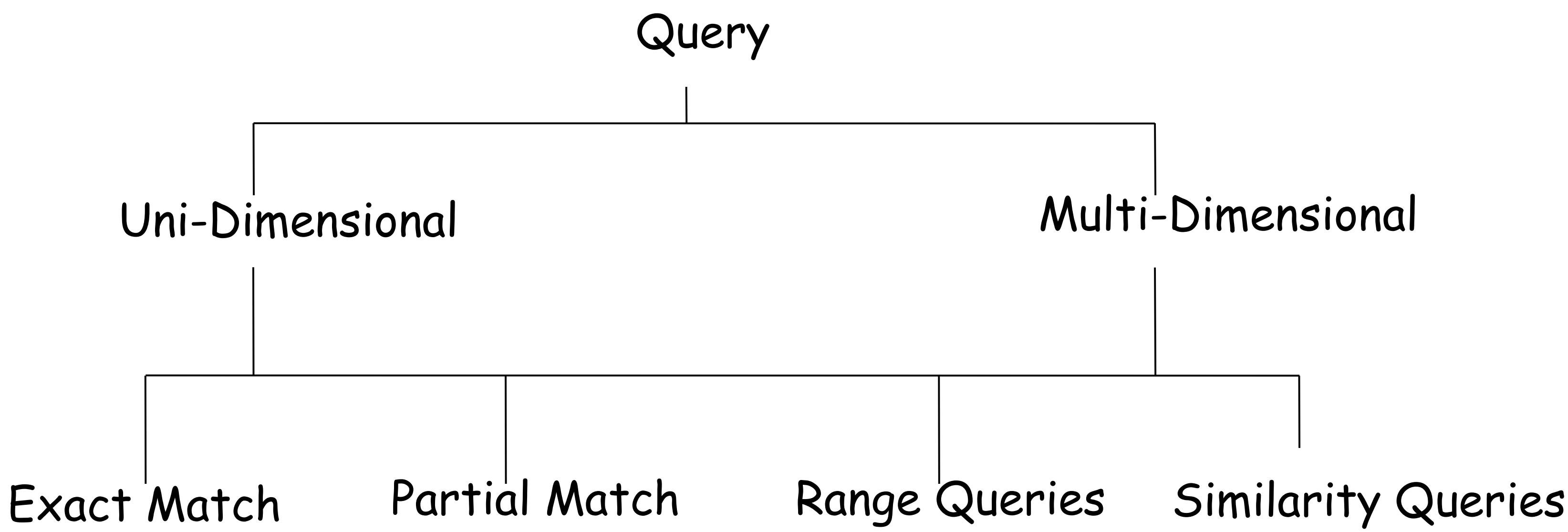
- indexing of distributed resources
- complex queries requiring resources satisfying a set of constraints
- dynamic attributes
- pub/sub functionalities. notification of a resource state updates

# COMPLEX QUERIES: CLASSIFICATION

Queries may be classifed according to

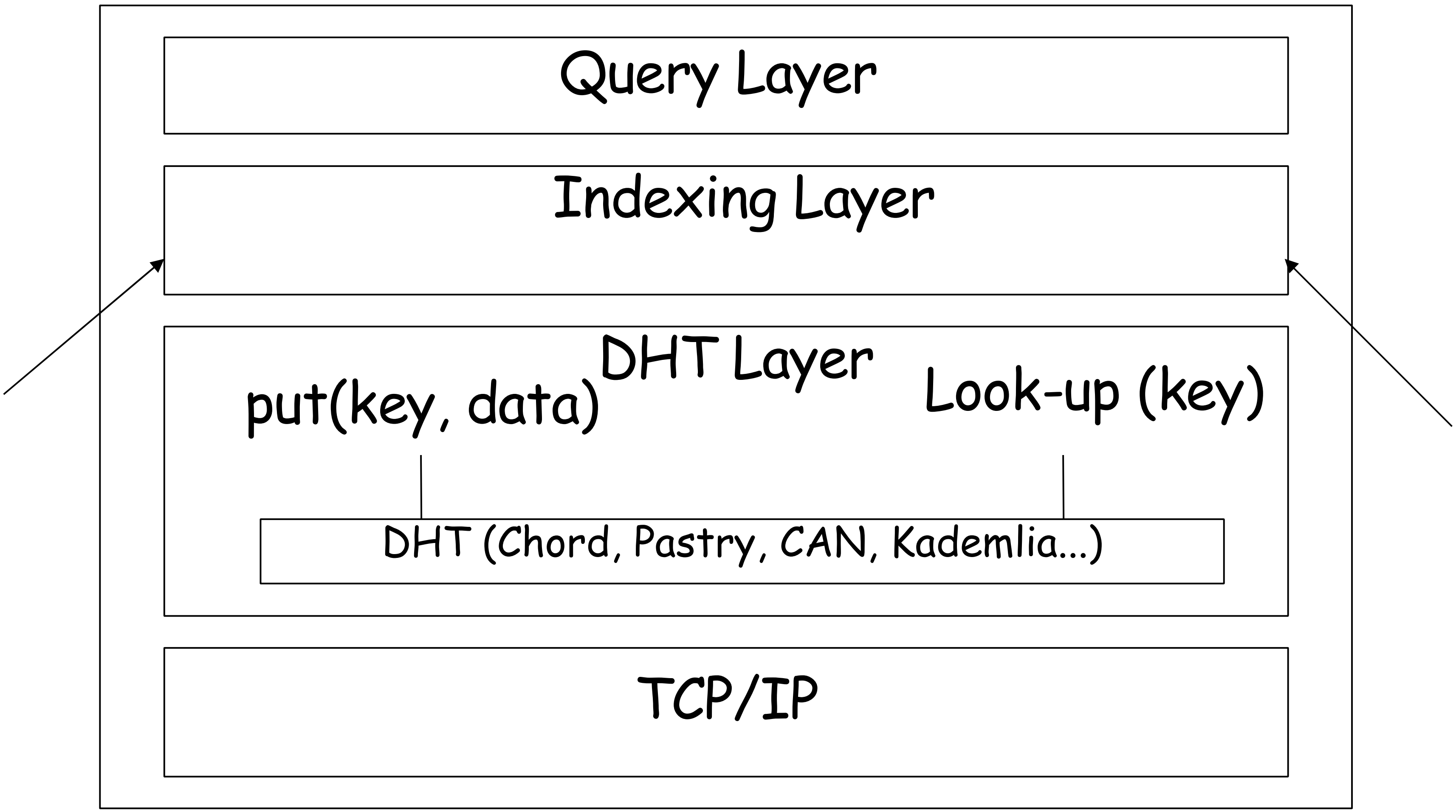
- the number of attributes considered by the query
  - k-dimensional queries, k, k≥2 (DHT support 1-dimensional queries only)
- type of attributes: static, dynamic, semi-dynamic
- constraint defined on each attribute
  - exact match query: Arch.='x86' **and** CPU-Speed='3 Ghz' **and** RAM='256MB'
  - partial match queries: CPU-Speed='3 Ghz' **and** RAM='256MB' (**and** Arch.=*)
  - range queries 1Ghz<CPU-Speed<'3Ghz' and 512MB<RAM<1Gb
  - similarity queries (o nearest neighbour queries)
    - require the definition of a metric in the attribute space
    - the user submit an exact match query, which defines a point P in the attribute space. P may not correspond to any resource.
    - Output: k resources nearest to P, according to the defined metric

# COMPLEX QUERIES: CLASSIFICATION




- Data  returned by these queries are close in the attribute space
- Data locality is destroyed by the hash mapping defined in DHT. Points, i.e. resources, close in the attribute space may be mapped to nodes far from each other on the overlay
- This is due to the uniform mapping defined by the hash function
- Definition of an indexing layer above the DHT to recover loss of locality

# NODE ARCHITECTURE



# EXISTING APPROACHES

DHT based approaches

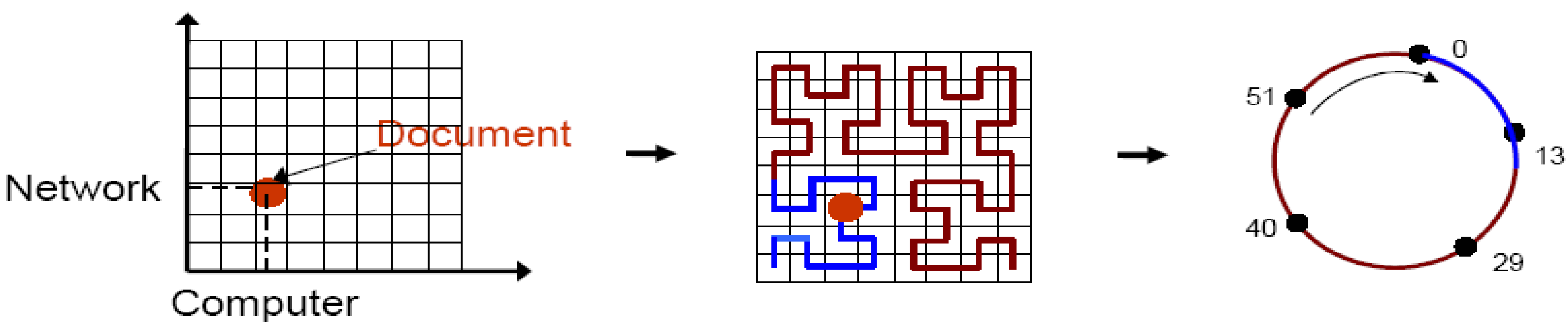
- Locality preserving hash functions
  - 1-dimensional-range queries (MAAN,CHORD#)
  - k-dimesional range queries: locality preserving mapping from a k-dimensional space to a 1-dimensional space based on space filling curves (Squid)
- Space partitioning based approaches
  - The k-dimensional attribute space is partitioned into a set of zones
  - The granularity of the domain of the hash function is increased
  - Mapping of zones to peers (Gao-Steenkiste, Ratnasamy,.....)
- Other approaches (not DHT based)
  - Voronoi based overlays definition
- Important Remark: No proposal defines a single framework for range queries, multidimensional queries, dynamic attributes

# K-DIMENSIONAL SPACES ON DHT

- Red-blue Line= Space filling curve
- Space linearization: Each point of the k-dimensional space is mapped to a point of the blue-red line
- Points on the blue-red line are indexed by integer numbers.
- Mapping of the points of the line on the DHT: keys= indexes of points

Example: Point (010, 010) (red point ) is mapped on Successor(001000).

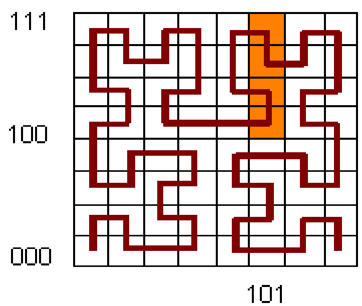
- Points close on the blue-red line
  - are close in the k-dimensional space
  - are mapped to the same node of the DHT or to close nodes



# SPACE FILLING CURVES: QUERY RESOLUTION



- Range Query Resolution
  - points which are close on the red line are also close in the k-dimensional space
  - Points close in the k-dimensional space may be not close on the red line
- Cluster = set of points belonging to the same segment of the red line
- Range Query Resolution
  - Detection of clusters covering data covered by the query
  - For each cluster, the query is sent to the nodes storing that cluster
  - A single message for each cluster.

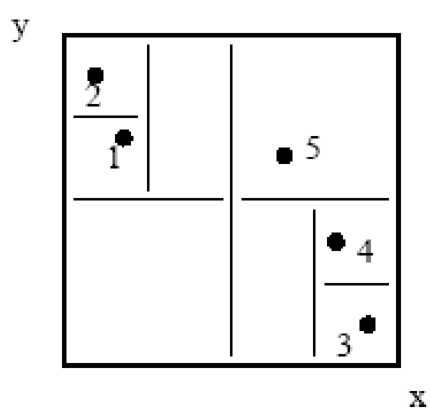


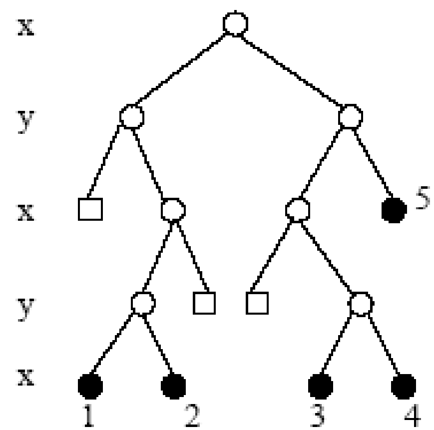

Range query (101, 100-111)

- covers the orange zone
- defines two different clusters

# SPACE PARTITIONING APPROACHES

- Attributes space is partitioned into zones
- Each zone is assigned to a different peer
- Hash functions maps zones, instead that single points.
- Attribute space partitioning is described by a tree-like index structure which is distributed to the nodes of the system
- Given a k-dimensional query corresponding to a point P in the k-dimensional space, the tree is exploited to detect the zone Z including P
- The querying node exploits the hash function to map Z to a node

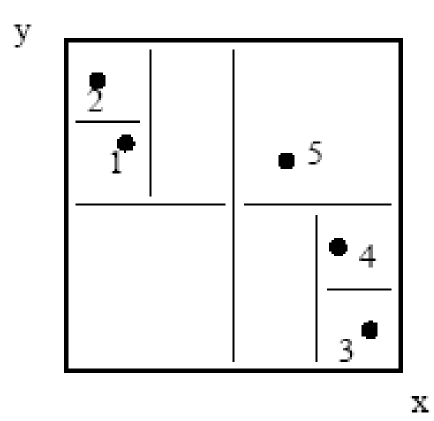


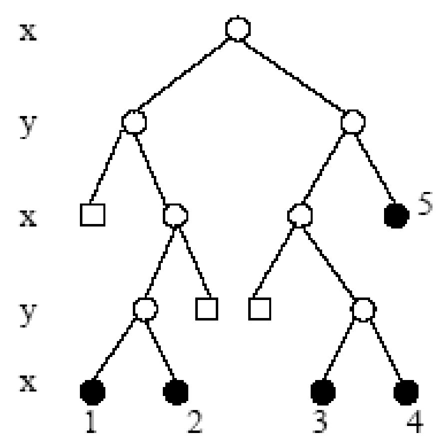



# SPACE PARTITIONING APPROACHES

Open research problems:

- Range query support
- Definition of highly distributed data structures
- Replication/consistency of the data structure
- Dynamic indexes
- Load Balancing techniques



# FIRST PERIOD PROTOTYPE

- The prototype developed in the first period of the project must integrate k-dimensional and range queries within a single framework
  - based on locality preserving hash functions
  - dynamic attributes?
  - experiments on a real distributed platform (GRID 5000)
- Afterwards, investigate more complex solutions
  - Tree based indexes
  - Space filling curves

# FIRST PERIOD PROTOTYPE



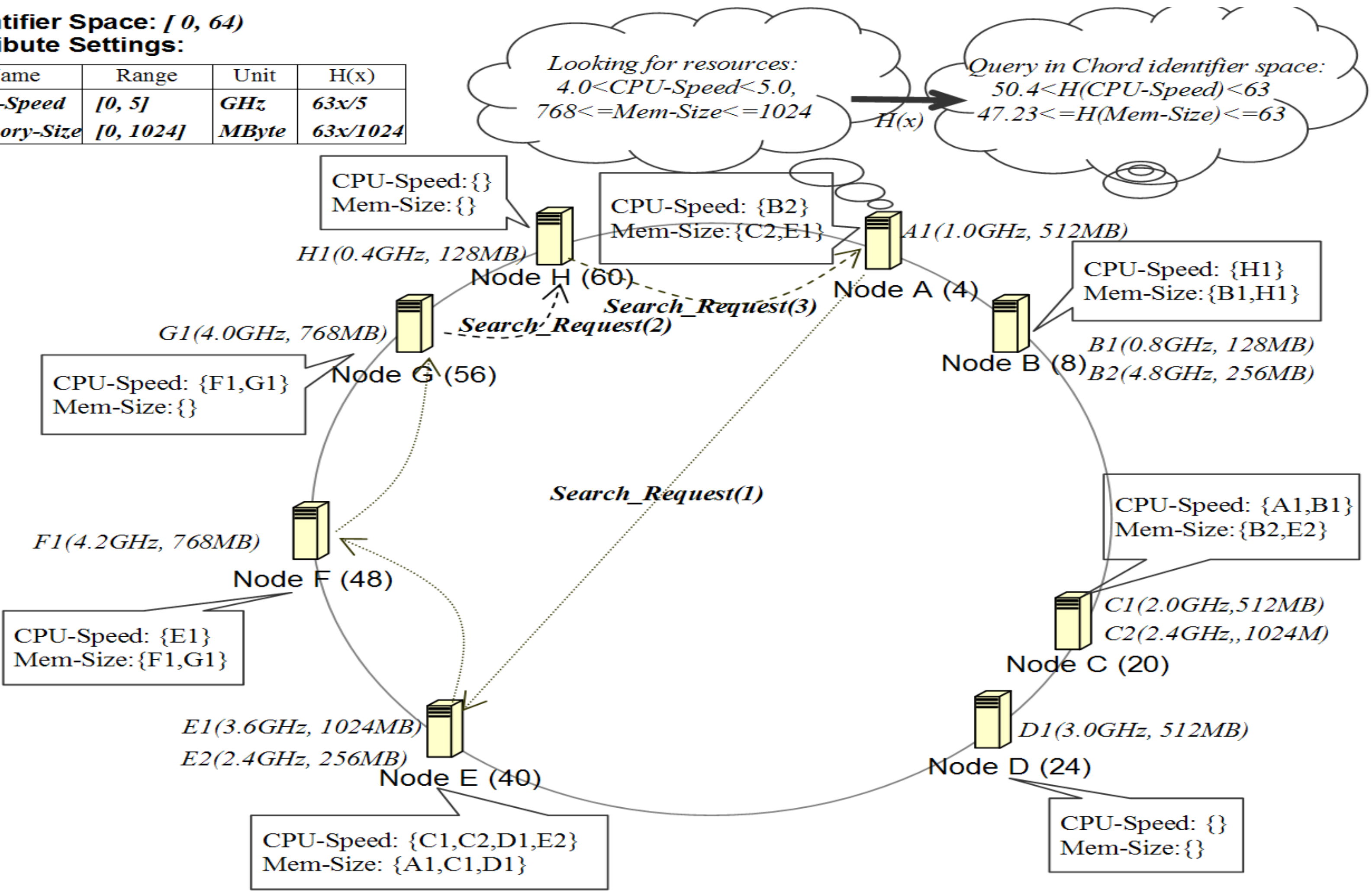
A directory service supporting k-dimensional range queries:

- based on DHT
- exploiting locality preserving hash functions (Locality Preserving Bamboo, Chord#,...?)
- load balancing: insertion of new nodes in 'crowded regions'
- k-dimensional range queries
  - replication. A resource R defined by k attributes is registered under k different keys. One key for each attribute value
  - k-dimensional Range Queries
    - simple, but inefficient solution: intersection of k 1-dimensional range sub-queries (one for each attribute).
    - improvement:
      - definition of a dominant attribute. decreasing the size of the search space
      - A query for the dominant attribute, sub-query managed by each detected node

# FIRST PERIOD PROTOTYPE

**Identifier Space:** *[ 0, 64)*
**Attribute Settings:**

| Name | Range | Unit | H(x) |
|---|---|---|---|
| *CPU-Speed* | *[0, 5]* | *GHz* | *63x/5* |
| *Memory-Size* | *[0, 1024]* | *MByte* | *63x/1024* |



# DYNAMIC ATTRIBUTES

- detect static and dynamic attributes
- define groups of resources characterized by the same values of a static attribute
  - Ex: all the host with CPU=2.5GHz
- define a multicast group G for each group of resources
- Hashing is applied to the static attribute. The resulting node returns a node R acting as the root of the a multicast tree associated to G.
- R forward the query to any peer P belonging to the multicast group. Each P checks the value of the dynamic attributes
- Exploit the DHT routing level (ex: Scribe application level multicast on Pastry) to define an application level multicast