VoRaQue: Range Queries on Voronoi Overlays

Michele Albano and Laura Ricci
Department of Computer Science
University of Pisa
Largo B. Pontecorvo, 3, 56127,
Pisa, Italy
{michele, ricci}@di.unipi.it

Martina Baldanzi and Ranieri Baraglia
Institute for Science and Technology of Information
National Research Council
Via G. Moruzzi, 1, 56124,
Pisa, Italy
{martina.baldanzi, ranieri.baraglia}@isti.cnr.it

Abstract—This paper presents VoRaQue, a software layer supporting range queries on Voronoi P2P overlays. VoRaQue maps data in a 2-dimensional space. The P2P overlay is defined by links connecting nodes that are close in the 2-dimensional space and by a set of long-range links which guarantee a logarithmic routing. When a query is submitted, VoRaQue finds out a node belonging to the region defined by the query. A multicast spanning tree covering that region is then built by applying compass routing, a distributed protocol to embed a spanning tree into a Delaunay Triangulation. The paper presents the basic VoRaQue protocol, then introduces a set of optimizations and finally presents some experimental results.

Index Terms—peer-to-peer, multicast, compass routing, delaunay triangulation

I. INTRODUCTION

The wide diffusion of P2P applications has recently produced a large set of proposals for the distribution and the retrieval of large data sets on a distributed setting. Distributed Hash Table, DHT[1] is the approach which has recently received most attention, because it does not require any centralized support to coordinate the distribution of data on the P2P nodes. The main goal of a DHT is to achieve a uniform distribution of data objects in order to obtain a good load balancing and an high degree of robustness. Furthermore, the most popular DHTs, like Chord[2] and Pastry[3], guarantee logarithmic bounds both for the size of the routing tables and for the number of routing hops for query resolutions. There is a number of different approaches based on DHT, and a thorough survey is presented in [4]. Despite their several merits, DHTs have limited capabilities for the support of complex queries, like multi-attribute range, k-neighbours or similarity queries [4], while single attribute exact queries are easily supported through the standard get and put DHT interface. The main problem with complex queries is that most current DHT approaches distribute data objects uniformly within the system and destroy any locality of source data. Several applications, like Geographic Information Systems[5], need to retrieve data through complex queries. For instance, queries for spatial data in a Geographic Information System (GIS) generally concern a certain area, hence the spatial relationship of objects, and therefore the spatial locality needs to be preserved when data objects are distributed. It is possible to exploit spatial locality to realize a look-up service for items with similar properties. Two numerical properties of the items (e.g. memory capacity and processor power to find computers with similar characteristics in a grid) can be used like coordinates to manage the queries like in a GIS. A query for similar items can be implemented like a ranged spatial query.

Several approaches have recently been proposed to overcome the limitations of DHT. Some of them are based on the definition of locality preserving hashing functions[4]. On the other hand, if a non-uniform hash function is exploited, load balancing may be not guaranteed. Another class of proposals define a indexing layer[6] based on the construction of a tree-shaped data structure, which may be defined on the top of the DHT layer or be exploited to directly define the overlay network.

In this paper we present an alternative approach based on the definition of a Voronoi based overlay network. The main idea of this approach is to consider the 2-dimensional space of attributes characterizing the shared data and map each data into this space, according to the values of its attributes. In this way, data characterized by nearby values of the attributes are mapped to close sites in the space. In the simplest case a one-to-one mapping between the data and the hosts is considered. In the more general case, subsets of the 2-dimensional space including close data may be mapped on the same host. The definition of an overlay network preserving the data locality exploits Voronoi tessellations [7] of the data space. A Voronoi tessellation is a partition of the space which assigns to each site $S$ the set of points of the 2-dimensional space which are closer to $S$ than to any other data site. The overlay network is defined by connecting the sites whose Voronoi regions are adjacent. These connections correspond to the edges of the Delaunay Triangulation defined by the Voronoi tessellation. To guarantee a small world overlay [8], a set of long-range links have to be added to the overlay. This paper presents VoRaQue, a Voronoi based overlay supporting range queries. When receiving a range query $Q$ from a node belonging to the overlay, VoRaQue forwards $Q$ toward a node belonging to the subspace defined by $Q$, afterwards $Q$ is propagated to any node belonging to this subspace. VoRaQue

Work funded in part by the European Commission in the framework of the XtreemOS project (FP6-IP-IST-033576) and of the Coregrid project (FP6-NoE-IST-004265).
embeds a multicast spanning tree into the Delaunay based overlay by exploiting Compass Routing [9], a technique for the distributed construction of the spanning tree which exploits a set of properties of Delaunay triangulations. The basic protocol has been improved by a set of optimizations. The paper is organized as follows. Sect. II describes some related work. The main design choices of VoRaQue are described in Sect. III, while Sect. IV describes the protocol in more details. Some experimental results are described in Sect. V, while section VI reports some conclusions.

II. RELATED WORK

The idea of exploiting Voronoi tessellations to build P2P overlays was presented in [10], that introduced Voronet. This P2P system does not exploit hash functions to distribute data objects. Instead, it specifies a 2-dimensional attribute space where the coordinates of each object are the values of its attributes. To define a Small World overlay, Voronet maintains two sets of neighbours, the Voronoi Neighbours corresponding to the edges of the Delaunay Triangulation and the Long-Range Neighbours providing the small-world characteristics, according to Kleinberg’s model[8]. Finally, each object maintains a set of close neighbours whose distance from the object is smaller than a predefined threshold depending on the number of objects on the overlay. These neighbours are required to ensure a poly-logarithmic routing cost, even in presence of ‘crowding phenomena’, i.e. situations where a lot of objects are located in the same region of the space. Voronet’s paper describes the Voronet routing algorithm and proves complexity bounds both for the routing tables size and for the routing steps, but does not propose any support for range queries.

[11] proposes a family of distributed access methods, the Small World Access Methods (SWAM) for the efficient execution of different kinds of complex queries like range and k-nearest neighbours queries. These methods guarantee that the time necessary to find the object defined by an exact query is proportional to the logarithm of the size of the network. Furthermore, all the similar objects would be located in neighbouring nodes. This work proposes a Voronoi based SWAM where range queries are solved in two steps. First, the range query is considered as an exact-match query for a point chosen in the query region. Then flooding is exploited to propagate the query to all the nodes of the region. Our approach differs from that of SWAM because we construct a multicast spanning tree on the nodes of the region defined by the query, and then we propagate the query on this tree.

[12] exploits a metric-space approach, firstly introduced in [13], and a Voronoi-based heuristics is applied to develop a search tree. As other tree-based approaches, this one fails to distribute the query execution load among the nodes of the overlay.

[14] applies Delaunay Triangulation to the processing of spatial queries. It employs a two-level routing (at the first level the query is routed to the region where results are located, then results are collected using a distributed visit in unicast). The main differences with our approach are in the collection phase. In fact, VoRaQue uses a Voronoi-based broadcast, that parallelizes communication and that results in a faster resolution of the query.

III. VOQAQUE: DESIGN CHOICES

This section describes the general design of VoRaQue. In VoRaQue each node is represented by two attribute-value pairs. The attribute is numerical and therefore the data can be described as a point in a 2-dimensional space. In the following, the terms data and node will be used in an interchangeable way, since a one-to-one mapping between data and nodes is considered.

VoRaQue supports multi-attribute range queries, i.e. queries defined as a conjunction of predicates where each predicate has the form (attribute operator value). VoRaQue supports the following operators: >, >, =, ≤, ≥. A disjunction may be simply implemented by multiple distinct queries. Each query \( Q \) defines a region \( R(Q) \) in the 2D-attribute space. The 2-dimensional space of the nodes is partitioned into a set of regions by considering a Voronoi tessellation of the space. We recall that, given a set of sites \( S \) in the plane, a Voronoi tessellation is a partition of the plane which associates a region \( Voro(s) \) with each site \( s \in S \) so that all the points in \( Voro(s) \) are closer to \( s \) than to any other site in \( S \). In our case, the points correspond to the nodes of the overlay. A Delaunay triangulation of a set of sites is a collection of edges satisfying an empty circle property. For each edge it is possible to find a circle containing the endpoints of the edge, but not containing any other site. The Delaunay triangulation is the dual structure of the Voronoi diagram and can be obtained by drawing a line segment between two sites \( s_1, s_2 \) if and only if \( Voro(s_1) \) is adjacent to \( Voro(s_2) \), i.e. they share a common edge. The Voronoi Overlay includes all the connections corresponding to the Delaunay graph defined by the Voronoi partition of the space.

VoRaQue is built on the top of Nomad[15], a lower level layer which defines a set of functionalities for Voronoi Overlays, including the management of Voronoi, long-range and close neighbours links. The long-range neighbours are chosen and managed like in [10]. The description of this level is outside the scope of this paper.

Let us now discuss the strategy defined by VoRaQue to support range queries. In the following, given a query \( Q \), each node of the 2D space belonging to \( R(Q) \) will be referred as an exact match of the query, while the term border match will be exploited to denote a node \( n \) such that \( Voro(n) \) intersects \( R(Q) \), but \( n \) does not belong to \( R(Q) \). A node is a match for a query \( Q \) if and only if it is a border or an exact match for \( Q \). A node \( n \) may submit a query only after it has joined the underlying Voronoi Overlay. When a range query \( Q \) is submitted, VoRaQue exploits the following two steps strategy to find all the exact matches of \( Q \).

- \( Q \) is forwarded to a node \( T \) belonging to \( R(Q) \) by exploiting a greedy routing approach. To implement a greedy strategy, VoRaQue exploits all the neighbours (the
Voronoi, the close and the long-range ones) of each node. Section IV will show that a proper choice of $T$ may influence the performance of the query resolution.

- starting form $T$, $Q$ is propagated to any other node in $R(Q)$.

The choice of the protocol exploited in the second step is fundamental to guarantee a good scalability. [11] exploits flooding to propagate the query. While the implementation of this strategy is straightforward, the resulting amount of messages is too high. VoRaQue decreases the number of messages by an application level multicast based on the distributed computation of a spanning tree embedded in the overlay. [9] proposes to exploit the Delaunay Triangulation, corresponding to the Voronoi partition, to build a spanning tree where the root of the tree is the sender of the multicasted message. The properties of Delaunay Triangulation guarantee a distributed construction of the tree by the application of compass routing, a distributed routing strategy which may be exploited by each node to determine locally its children in the spanning tree. Section III-A describes this approach in more detail. On the other hand, a direct application of this approach is not possible in our case, since we must take into account that the spanning tree should be constrained within the $R(Q)$. The trivial solution which considers only the nodes belonging to $R(Q)$ is not correct. Consider, for instance, the rectangular region, shown in Fig. 1, that is the region defined by a query. Furthermore, the theorem suggests that some paths of the spanning tree may be, in general, disconnected.

To guarantee that the spanning tree covers each node belonging to $R(Q)$, VoRaQue exploits all the query matches, i.e. both exact and border matches. The following theorem shows that the graph whose nodes correspond to the matches $M$ of a query and whose edges correspond to Voronoi connections between nodes in $M$ is connected.

**Theorem 1**: Given a 2D region $V$ including a set $S$ of sites, let us consider the Voronoi tessellation $Vo$ of $V$ defined by $S$. Let $R = \{(x, y) : x1 \leq x \leq x2, y1 \leq y \leq y2, \}$ where $R \subseteq V$. The graph $G = (N, E)$ where $N = \{s \in S : Voronoi(s) \cap R \neq 0 \}$ and $E = \{(n1, n2) : n1 \text{ is Voronoi neighbour of } n2 \text{ in } Vo\}$, is connected.

**Proof**: Let us call $n1$ and $n2$ a pair of nodes of $G$. Firstly, let us consider the case in which both $n1$ and $n2 \in R$. Let us consider the segment connecting $n1$ and $n2$. Let $C = \{Cr : \exists x \in N, Cr \subseteq Voronoi(x)\}$ be the set of contiguous regions that are intersected by the segment. One possible path to connect $n1$ and $n2$ is defined by the edges of $G$ corresponding to the Voronoi connections between each pair of regions in $C$ which are contiguous. Since the segment is completely $\in R$, all the regions $\in C$ share at least the segment’s points with $R$ and hence the sites of the regions $\in C$ must be $\in N$.

Let us now consider a pair of nodes $n1, n2 \in N$, where $n1 \in R$, $n2 \notin R$, i.e. $n1$ is an exact match for the query, while $n2$ is a border match. $Voronoi(n2)$ includes at least one point $p_{n2} \in R$. The segment connecting $p_{n2}$ and $n1$ is $\in R$ and we can apply the previous argument to tell that the contiguous regions selected by the segment must describe a connected subset of $G$. But $p_{n2}$ and $n1$ belong to the same region $\in N$. Hence the path, that is a subgraph of $G$, shows that $n1$ and $n2$ are connected in $G$.

The case where $n1$ and $n2$ are $\notin R$ is analogous.

Hence, VoRaQue guarantees packet delivery to each peer belonging to $R(Q)$. The theorem suggests that all the query matches must be considered to define a spanning tree which covers all the exact matches for the query. Furthermore, the theorem suggests that some paths of the spanning tree may zigzag around the borders of $R(Q)$. It is possible to show that the number of zigzag paths may be minimized by choosing a site near the center of $R(Q)$ as the root of the spanning tree.

**A. Spanning Tree Construction**

The distributed algorithm for the embedding of a spanning tree into a Delaunay triangulation is based on Compass Routing[16][17]. According to this approach, each node is able...
to determine its children in the spanning tree by exploiting its coordinates, the coordinates of its neighbors and of the root of the tree only. Compass routing is based on a simple observation. Let us consider a connected graph and suppose to be located at a node $n$ of $G$ with the goal to reach a destination node $d$. [9] shows that the best strategy is to look at the edges incident in $n$ in order to choose the edge $e$ whose slope is minimal with respect to the segment connecting $n$ and $d$. Based on this observation, a spanning tree where the root corresponds to $d$ may be built by a bottom up procedure allowing each node to detect its parent in the spanning tree. On the other way, VoRaQue requires that the tree is spanning starting from the root, which corresponds to the node propagating the query. Each node $n$ can find out if a neighbour $m$ is one of its sons by detecting if $n$ belongs to the path from the $m$ to the root. For this reason, $n$ must be aware of the neighbours of its neighbours in order to apply compass routing. In Fig. 2 node $R$ is the root of the spanning tree. Let us suppose that the spanning tree construction has reached $A$ and that $B$ and $C$, which are neighbours of each other, are also neighbours of $A$. While the parent of $B$ in the spanning tree is $A$ because $RBA$ is smaller than $RBC$, the parent of $C$ is $B$ because $RBC$ is smaller than $RCB$. The bold lines shows the edges in the spanning tree, while the thin lines are drawn to show the angles involved in the computation.

IV. VoRaQue: the protocol

This section describes the protocol defined by VoRaQue in more details.

A node must belong to the Voronoi overlay to be able to submit a query to VoRaQue. Firstly, VoRaQue forwards each query $Q$ using greedy routing. At each forwarding step, each node chooses among its neighbours, i.e. its Voronoi, Close or Long-Range neighbours, the one whose euclidean distance from the middle point of $R(Q)$ is minimal. The user can ask the system to start the data collection from $P$, the center of $R(Q)$, or from the first point of $R(Q)$ that the query reaches.

VoRaQue defines the following set of conditions to detect if a node $n$ may be considered a match for a query $Q$. These conditions are not mutually exclusive. If at least one of these conditions holds, $n$ is a match for the query.

1) $n$ is an exact match for the query
2) a vertex of $\text{Voro}(n)$ belongs to $R(Q)$
3) the vertex of $R(Q)$ whose euclidean distance from $n$ is maximal, falls within $\text{Voro}(n)$
4) a border of $\text{Voro}(n)$ intersects a border of $R(Q)$.

These conditions are sorted with respect to their computational complexity. VoRaQue checks them following this order to avoid, when possible, expensive computations. The first and the second condition are the easiest, since they require the check of a set of linear restraints only. The third condition takes into account the case where $R(Q)$ is a subset of $\text{Voro}(n)$, but $n$ does not belong to $R(Q)$, i.e $R(Q)$ does not include any exact match. The last condition requires the computation of a set intersections between pair of segments and it is taken into account if and only if any other condition fails.

The target node $n$ of the greedy routing acts as the root of the spanning tree by starting its construction through compass routing. To detect its children in the spanning tree, $n$ first asks its neighbours which are matches for the query to find their neighbours which are matches as well, then it applies compass routing. A VoronoiNeighMSG message tagged by a unique identifier is built and sent to each different neighbour. The identifiers are exploited to retrieve the corresponding RangeMSG and continue the computation of compass routing. This is required because each node may receive and process further queries before the computation of the previous one is completed. Each neighbour sends a ReplyVoronoiNeighMSG including the same identifier received in the corresponding VoronoiNeighMSG, its coordinates and the coordinates of all its neighbours which are matches for the query. As soon as $n$ receives a ReplyVoronoiNeighMSG from a neighbour $m$, it may apply compass routing to detect if $m$ is one of its children. In this case, it propagates a RangeMSG which comprises the coordinates of the region defined by the query and a user-defined $TTL$ value. This last field is used to define incremental searches, as described in section IV-A. Furthermore, $n$ detects if it is an exact match for the query and, in this case, it inserts its coordinate in the RangeMSG sent to one of its children. In this way, exact matches are added incrementally to the RangeMSG as it propagates down the spanning tree. When a RangeMSG reaches a leaf of the spanning tree or the value of $TTL$ becomes 0, the exact matches are returned to the node which has submitted the query, through a ReplyRangeMSG. To avoid duplicated messages, each exact match is sent along a single path in the spanning tree. The behaviour of each node receiving a RangeMSG is similar to that of the root.

A. Incremental Search

The user may associate a $TTL$ to each range query. This way, he defines the maximum number of hops for the query, i.e. the maximum depth of the spanning tree. At the use-case level, the user may send a first query with a certain $TTL$. Afterward, if the amount of exact matches is not adequate, the user may increment this value and re-submit the query.

VoRaQue does not repeat the search from scratch: it starts from the root of the spanning tree. For this reason, when a node $n$ sends a ReplyRangeMSG, it inserts in the message a boolean flag which suggests if the search may proceed from $n$. For instance, nodes near the border of $R(Q)$ may signal that there is no further exact matches in that direction. When the user requests further matches, VoRaQue avoid to contact these nodes, while it asks for other exact matches to other nodes.

B. Caching Strategies

Caching is exploited by VoRaQue to optimize the basic protocol. Each node caches any information received by its neighbours regarding the topology of the overlay. In this way the amount of VoronoiNeighMSG transmitted to implement compass routing is minimized. The cache is periodically
reflected. The rate of refresh depends upon the degree of churning of the overlay. Caching is exploited by a node \( n \) also to record the structure of the spanning tree with respect to a specific range query, i.e. its children in the spanning tree with respect to that query.

In the current implementation of VoRaQue, similar queries are supposed to be repeated in a short time span. Hence, all the topology data will already be into the local memory. A realistic caching would have to deal with the finite size of caching memory and with aging politics to substitute old data, but this topic will be addressed in future works.

V. EXPERIMENTAL RESULTS

In this section we describe the experiments that demonstrate the scalability of VoRaQue. We have used Grid'5000 [18] as testbed, that is a reconfigurable, controllable, monitorable Grid composed by heterogeneous clusters. Grid'5000 is composed by 9 sites and each one of them hosts one or more clusters. Grid’5000 is a real-life testbed because the sites are geographically distributed – one site is on Rennes, another is on Lyon, etc. – and it is not dedicated to our experiments only, but it is available for tests conducted by other registered users.

As we said, Grid’5000 is composed by 9 sites and our tests have been performed on 4 subsets of them:
- Rennes, Sophia-Antipolis;
- Bordeaux, Rennes;
- Nancy, Rennes, Sophia-Antipolis;
- Bordeaux, Nancy, Rennes, Sophia-Antipolis.

The maximum size of the set of nodes that was reserved to out experiment was 300 out of 1249. On each node involved in the testing phase, we run a VoRaQue instance.

Traffic evaluation. The goal of the first set of experiments was to evaluate the average number of messages sent by each node for different sizes of the overlay, and to show the effects of the caching strategies. The experiment consisted in initializing a VoRaQue overlay within each node exploiting an empty local cache, then submitting twice the same query. The area covered by the query was the same for all the experiments. The second query could exploit the information gathered by the compass routing of the first one and stored in the local cache of each node.

For the first query, the number of messages includes any message described in section IV, i.e. the messages supporting compass routing and the ones exploited to forward the query and to gather the results. For the second query, only query forwarding and gathering messages are considered, since the local cache includes information related to the topology of the overlay. The average number of messages per node was computed by dividing the total number of messages by the number of nodes on the overlay. Fig. 3 shows the average number of messages for each overlay size and the variance when no caching is exploited. Fig. 4 shows the same results when caching is exploited. The results show both the scalability of VoRaQue and the efficiency of the optimization.

Latency. The goal of the second set of experiments was to evaluate the milliseconds necessary to resolve a range query for different query areas and to show the efficiency of the caching strategies (figure 5, 6). The overlay network was fixed on 300 nodes. Figures 6 shows good results thanks to our protocol and caching strategies.

Hop number evaluation. The goal of the third set of experiments was a comparison of the number of exact matches of a range query detected at each hop from the root of the spanning tree when different initial positions of the spanning tree root were considered. The queries were submitted by exploiting the incremental protocol, increasing the value of TTL with each message in order to collect the number of
matches at each hop. The overlay included a fixed number of nodes and each node exploited caching. Figures 7, 8 show the number of matches as a function of the depth of the spanning tree, i.e. the maximum number of hops, with respect to 5 different queries, which are reported in the figure. Figure 7 shows the results when the root of the spanning tree was located almost in the middle of the region defined by the query, while in Figure 8 the root of the tree was located at the border of the region. The number of matches assumes a 'bell-shaped' behaviour, i.e., it increases until its maximum height, which is reached in correspondence of DepthMax. After DepthMax, the number of matches decreases, because the borders of the query are reached and the number of directions where the compass routing can forward the message decreases. The experiment shows that DepthMax is reached earlier if the root of the spanning tree is located near the middle point of the region.

VI. CONCLUSIONS AND FUTURE WORKS

This paper presents VoRaQue, a software layer designed to support exact and range queries on P2P Voronoi based overlays. We plan to improve the first prototype in several ways, e.g. investigating the behaviour of VoRaQue when considering highly dynamic attributes. VoRaQue currently manages this issue by 'moving' the object in the space and by locally reconstructing the Voronoi diagrams. We plan to investigate this issue more deeply and to evaluate the cost of the overlay management both in the case of a high degree of churning and when objects are characterized by highly dynamic attributes. Another interesting research direction is the analysis of caching strategies against realistic query distributions. We plan to experiment with different algorithms for the substitution of old topology information in the cache.

REFERENCES