# Real-Time Clustering of Large Geo-Referenced Data for Visualizing on Map

Mohammad REZAEI, Pasi FRANTI
*School of Computing, University of Eastern Finland, 80140, Joensuu, Finland*
*rezaei@cs.uef.fi*

***Abstract*—Displaying geo-referenced data in web mapping systems has become popular. However, most existing systems suffer from three annoying problems: (1) clutter when trying to visualize large amount of data; (2) slowness of transferring data over internet; (3) lack of support for dynamic queries. To solve these problems, we propose a real-time system using server-side clustering, transferring only the clustered data, and client-side visualization using existing map tools. As far as we know, there is no other scientific paper describing such real-time system that allows dynamic database queries without limiting to predefined queries. Experiments show that it can handle up to 1 million objects whereas all existing systems are either limited to pre-defined queries, or they support only a very small number of free parameters in the query whereas the proposed system has no such limitations.**

***Index Terms*—data visualization, clustering methods, web services, client-server systems, Internet.**

## I. INTRODUCTION

Rapid increase of cell phones and GPS devices has made it easy to collect huge amount of location-based or geospatial data. By location-based data, we mean photos or other data attached with their physical locations. Geo-visualization is a tool for better understanding, efficient search, and well-organized management of data. It has received considerable attention due to the rise of online maps and advances in graphics and display technology [1]. A *Web Mapping System* (WMS) is a tool for geo-visualization that standardizes the way of sharing geospatial information on the Internet using interactive web maps [2].

However, in case of visualizing large amount of data, most existing systems suffer from one of the following three problems: clutter, slowness, and restriction to pre-defined (static) queries. Fig. 1 shows a few examples found from web. Most present the data either as a set of points, or as small icons on the map. Only one web page shows the data (orienteering maps) as polygons. In all cases, the amount of data is relatively small and the selection of the data is predefined. The ones where data is several thousands (Jounin kauppa and Forenom) suffer from cluttering and sometimes also from slowness. To overome these issues, they allow user to select only a subset by filtering. Two pages (Way.fi and Wayne's coffee) use clustering provided by GoogleMaps API (the earth quake icon) but they are also limited to predefined data. None of the systems support dynamic queries.

The clutter problem appears when trying to show large number of items on the limited space of a display screen [3]. *Clutter reduction* aims at reducing the overlap for better information visualization. In cartographic generalization, a minimum distance between adjacent items is defined to avoid the overlap [4]. Zooming also helps to avoid information overload but at the cost of losing the overall view of the data [1],[4]. The data to be shown should therefore be reduced, but so that it still represents the entire selection. Filtering and sampling are other possible approaches for the data reduction. However, they can bias the distribution of the data, lose relevant data objects, and at the same time, present outliers [5], see Fig. 2. Clustering, on the other hand, aims at grouping the data based on a similarity (or distance) criterion between the objects. All the objects are still available by navigating through the groups.

Clustering consists of two design questions: which algorithm to use, and how to represent the clusters on the map. A cluster can be represented simply by an icon such as circles or point. For better usability, more information about the content of the cluster can also be given, such as the density or the distribution of the objects within the cluster. Several visualization techniques are shown in Fig. 3, in which we summarize six properties:

- Technique for data reduction
- Representation of cluster
- Showing density
- Showing distribution
- Opening (drilling down) a cluster
- Details-on-demand

*Flickr* and *Panoramio* filter data by random sampling. Panoramio uses two levels of sub-sampling based on popularity and proximity of the photos. All the other visualization approaches are based on clustering. Panoramio, *Mopsi* (http://cs.uef.fi/mopsi) and the system of Delort [6] use a selected photo to represent a cluster. *Tag maps* [7-8] represent a cluster by selected word whose size depends on the size of the cluster.

The visual aggregate should also include some information about the underlying data in the cluster [9]. Some systems show the cluster borders explicitly by Voronoi [2],[6],[10], convex hulls [11], or cells [12]. This helps to see the distribution of the data. However, the extra lines and shapes can overwhelm the view. Voronoi can also be misleading because it segments the entire area, also parts that do not contain any data [2]. Simple icon is still the most popular object to represent the clusters because of its simplicity. Google MarkerClusterer API uses an icon that

This work was supported in part by MOPSI projects 70052/09 and 70010/12.

Digital Object Identifier 10.4316/AECE.2018.04008

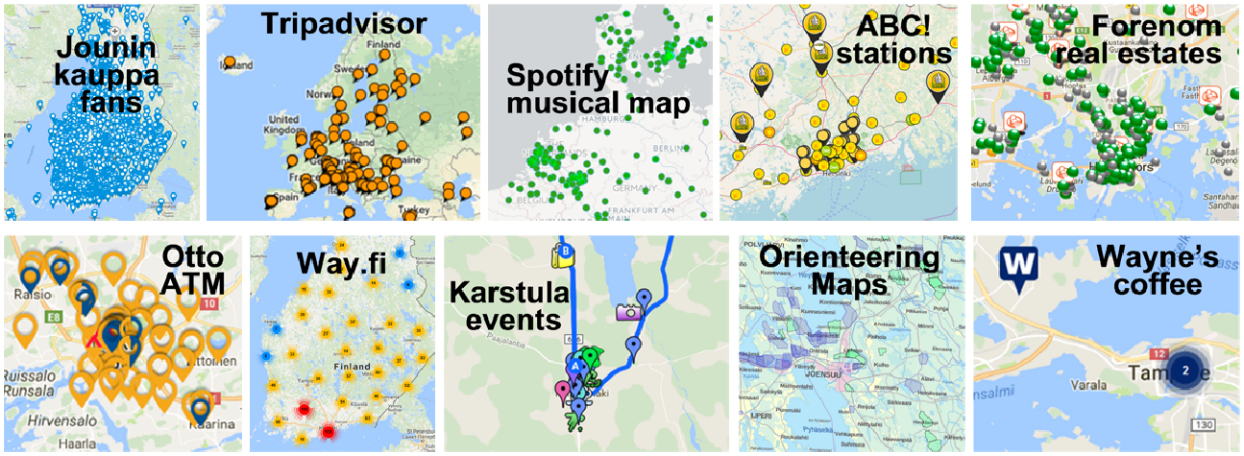



Figure 1. Typical examples of existing web systems and how they deal with the problem

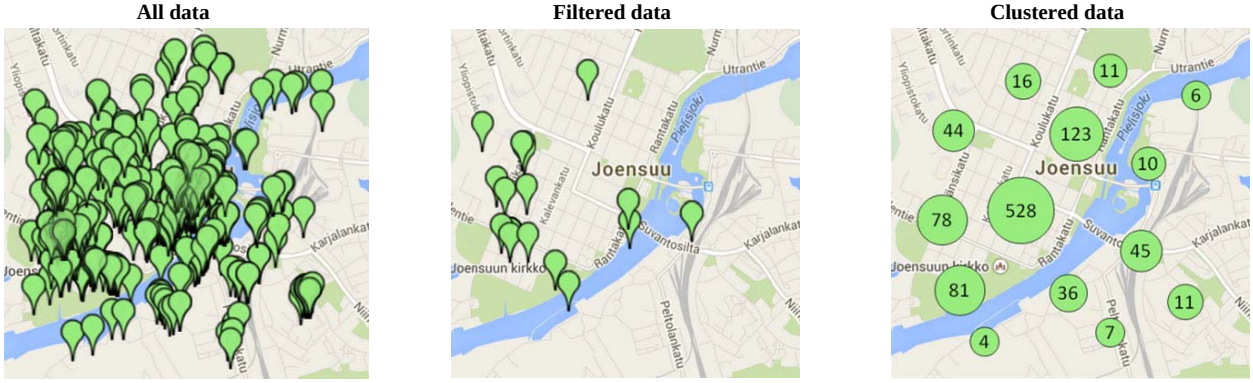



Figure 2. Clutter of 1000 geo-referenced data on the map (left), filtered by random selection of 20 objects (middle) and clustering (right)

indicates the density by showing number, color, and size of the icon. The data distribution and the covered region of the cluster cannot be shown directly by an icon [2],[13], but the area can be indirectly concluded from the overall distribution. Additional information can also be embedded into the icons [14-17]. For example, DICON uses a treemap style icon that includes statistical distribution of the data in the cluster [17]. Other representations include *heat maps* [5],[13],[18], and Splatterplots [19].

Opening a cluster [9] means that when the user clicks a cluster representative, an automatic zoom happens so that the objects in the cluster are displayed on the map view (the part of the map shown in a specified box in the interface). Most of the systems support this functionality. Details-on-demand indicates how the user can access the details of any object in a cluster without opening the cluster or zooming in [20]. In general, providing more information to help ordinary users for better understanding the data is not trivial [17].

For solving the clustering, a scalable algorithm is required that is capable of handling large data sets in real-time [4],[13],[21-22]. Downloading data and generating the clusters on the client is still possible but it would cause a high transmission load on the network, and therefore, is not suitable for low speed internet connections. Especially if the data contains images as in our case. To minimize data transfer, clustering should therefore be performed on the server, and send only the summary information of each cluster to the client.

Several server-side clustering approaches exist such as STING [23] and CLIQUE [24] but they pre-compute the clusters on the server. They apply clustering off-line to the entire data, and store the results on the server, which limits their use for static predefined queries only. However, we want to support also dynamic queries defined by the user in real-time. The results of such queries cannot usually be predicted beforehand because they are based on ad hoc query parameters such as free text keyword and time period, instead of just the location as in most existing systems [25]. To support dynamic queries, only a few approaches exist such as *imMens* and *nanocubes*, which have been recently proposed based on the idea of data cubes [5],[26]. A limited set of attributes (dimensions) of data are stored as aggregates on the server. The downside of this is the huge need for memory.

In this paper, we propose a solution that allows dynamic queries without any such limitations. We formulate the clutter removal of icons as a clustering problem. We propose a server-side approach, which clusters the objects dynamically on the server, and sends only the summary information of each cluster to the client. The algorithm has a grid-based structure, which provides high scalability. We provide client-side and server-side APIs for the method. The computational time of the clustering is less than one second for 1,000,000 objects, where the download size is limited to 15 Kbytes in a machine with Intel Xeon E7-4860 v2, 2.60 GHz. To represent the clusters, we support both photos and variable size user-defined icons. Clusters can be opened by clicking the cluster representative. When opening a cluster, the map is zoomed in automatically to cover the area where all the objects of the selected cluster are located. The new content is presented on the map, clustered again if the

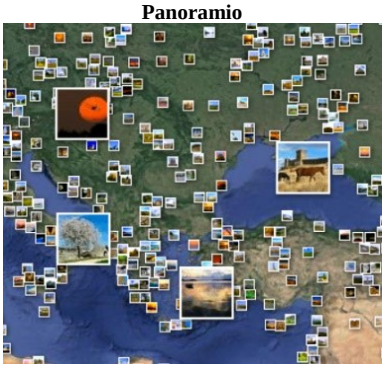
**Flickr**


(a) Filtering
(b) Circle icon
(c) -
(d) -
(e) -
(f) Yes

**Google maps**

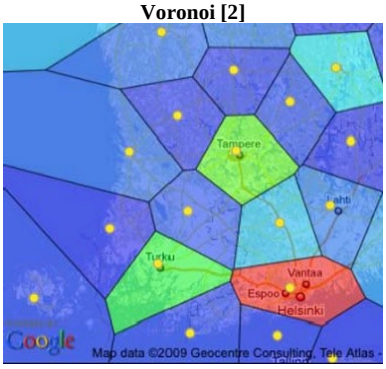

(a) Clustering
(b) Circle icon
(c) Color, number
(d) -
(e) Yes
(f) -

**Panoramio**



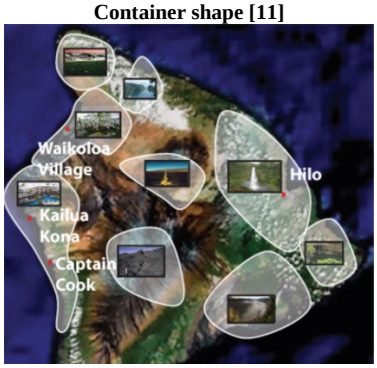
(a) Filtering
(b) Image icon
(c) -
(d) Yes
(e) -
(f) Yes

**Voronoi [2]**



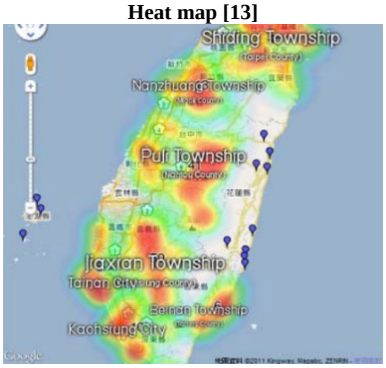

(a) Clustering
(b) Voronoi
(c) Color
(d) Yes
(e) Yes
(f) -

**Container shape [11]**




(a) Clustering
(b) Area, image
(c) -
(d) Yes
(e) Yes
(f) -

**Heat map [13]**

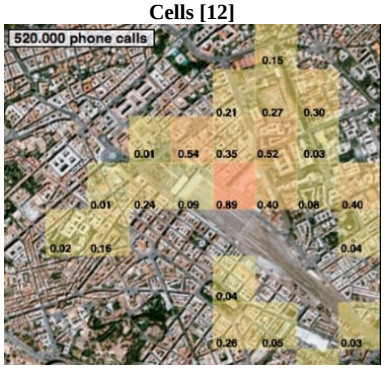



(a) Clustering
(b) Heat map
(c) Color
(d) Yes
(e) Yes
(f) -

**Mopsi**

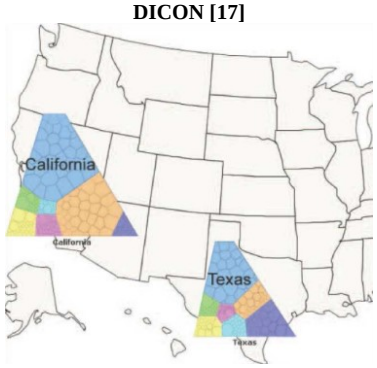
(a) Clustering
(b) Image icon
(c) Number
(d) -
(e) Yes
(f) Yes

**Cells [12]**



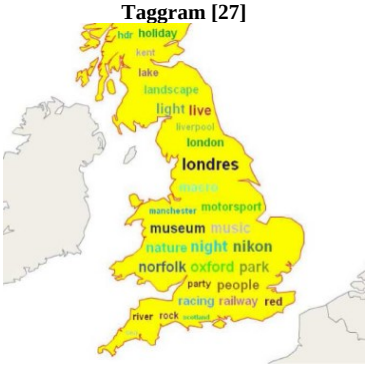

(a) Clustering
(b) Cell
(c) Color
(d) Yes
(e) Yes
(f) -

**DICON [17]**




(a) Clustering
(b) Icon
(c) Color & shape
(d) -
(e) Yes
(f) Yes

**Taggram [27]**




(a) Clustering
(b) Word
(c) Color & size
(d) -
(e) Yes
(f) -

Figure 3. Several representations of geo-referenced data on maps, and their properties: (a) Technique for data reduction (b) Representation of cluster (c) Showing density (d) Showing distribution (e) Opening a cluster (f) Details-on-demand


Figure 4. Different goals of normal clustering (middle) and clutter removal (right)

amount of data still causes clutter. Starting the view of the entire world, single photos are usually reached by 2-6 clicks.

The content of a cluster can also be accessible by *details-on-demand* principle [20]. Since our system contains merely photos, we open photo viewer where user can only go through the photos in the cluster one by one using kind of slideshow.

The rest of the paper is organized as follows. Clustering problem is formulated in Section II. In Section III, several clustering methods are analyzed for the purpose of the clutter problem, and the detailed procedure of the proposed grid-based clustering is presented in Section IV. Server-side approaches including our proposed method are studied in Section V. Experimental results are reported in Section VI, and conclusions are drawn in Section VII.

## II. Clustering Problem

Clustering for clutter removal differs from normal data clustering. Instead of finding real clusters, we aim at grouping data merely for visual clarity and better computer-human interaction. Any overlap causes difficulty and confusion when clicking an icon. For example, in Fig. 4, there are three well-separated clusters *A*, *B*, and *C*. The goal of normal data clustering would be to identify the three clusters, see Fig. 4 (middle). Some methods might consider the single object in cluster *C* as an outlier and identify only two clusters. However, a clutter removal method should be localized so that the sparse areas do not lose the details [4],[21]. Clustering distant objects together misleads the user about their real locations. The methods such as CLARA and CLARANS [28] that apply the clustering on a sample of data suffer from this problem. They assign non-selected objects to the clusters according to a criterion, for example to the nearest centroid. However, an object might be assigned to a distant cluster. To avoid this problem in clutter removal, it is therefore better to show more clusters as long as their representatives do not overlap. In this way, the time spent to access a photo would be shorter, see Fig. 4 (right).

### A. Objective of clustering

Given a data set $A \in R^2$ with $N$ objects, the problem of clustering is to group the objects into $K$ clusters. Each object must be assigned to a cluster, so that there are no outliers and missing data. The clustering has two objectives:

1. Maximize the number of clusters without overlap

To avoid overlap of clusters $i$ and $j$ with rectangular icons of sizes ($W_i$, $H_i$) and ($W_j$, $H_j$), see Fig. 5, one of the following conditions should be met:

$$\begin{aligned} |x_i - x_j| &> (W_i + W_j)/2 + T \\ |y_i - y_j| &> (H_i + H_j)/2 + T \end{aligned} \tag{1}$$

where ($x_i$, $y_i$) and ($x_j$, $y_j$) are the coordinates of the centroids $c_i$ and $c_j$. Value $T$=0 guarantees that there is no overlap, but a bigger value (we use here $T$=5 pixels) is preferred to have more space between the clusters. This leads to a more readable map, which is less covered with cluster representatives.

2. Minimize sum of squared error (SSE)

$$SSE = \sum_{i=1}^{N} \|a_i - c(a_i)\|^2 \tag{2}$$

where $c(a_i)$ is the centroid of the cluster that the object $a_i$ is placed in. Several possible clusterings satisfy the conditions in (1), from which the clustering that provides minimum SSE is the optimal. This condition indicates that the overall distance of the objects of a cluster to the centroid is minimized, which provides least confusion about the real location of the objects.

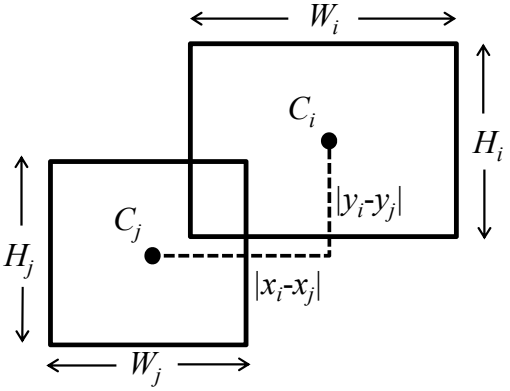



Figure 5. Overlap of the bounds of two icons

### B. Number of clusters

The number of clusters is unknown. However, it has an upper limit according to the average size of representative icons ($W$, $H$) and the size of map view ($W_0$, $H_0$). Assume that the map is filled with icons without any overlap. The maximum number of icons that can be drawn on the map is:

$$K_{\max} = \left\lfloor \frac{W_0}{W} \right\rfloor \times \left\lfloor \frac{H_0}{H} \right\rfloor \tag{3}$$

### C. Bounding box

We define the *bounding box* of a map view as the window with the top-left and bottom-right coordinates of the view: [($Lat_{max}$, $Lon_{min}$), ($Lat_{min}$, $Lon_{max}$)], see Fig. 6. The bounding box of a set of objects is defined in the same way but the boundaries are derived from the objects in the set. To display the set of objects on the map, a suitable map view should be first determined, which is the part of the entire map with the highest zoom level that contains the bounding box of the objects, see Fig. 6. The bounding box of the resulted map view can be wider than that of the objects because of the discrete change of zoom level.

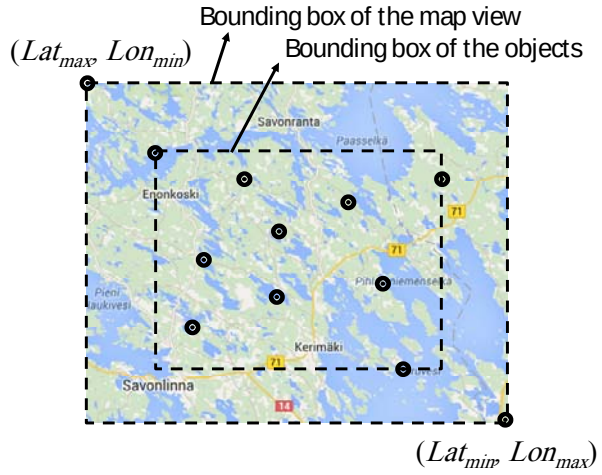



Figure 6. The bounding box of map view and the bounding box of objects. The map view for the given objects is corresponding to the highest possible zoom level that contains all the objects. Zooming in more will cause some of the objects move out of the display

## III. Clustering Methods

In this section, we study clustering approaches including divisive, density-based, agglomerative [28], and grid-based methods [29]. We show how some of them can be modified in order to be applicable to the clutter problem. We consider *K-means* (divisive), *DBSCAN* (density-based), and *centroid-linkage* (agglomerative) algorithms as examples of the first three categories, and analyze their suitability for the problem. A trivial *overlap-based clustering* algorithm is also considered because it is likely to be applied to the problem by many others due to its simplicity.

### A. K-means

*K*-means is a partitional clustering algorithm in which *K* centroids are initially selected in some way, for example randomly chosen data objects. Two steps of the algorithm are iteratively performed: assignment and update, for a fixed number of iterations or until convergence. In the first step, objects are assigned to their nearest centroid. In the second step, new centroids are calculated by averaging the objects in each cluster. Time complexity is O(*IKN*), where *I* is the number of iterations [30]. *K*-means is not suitable for the clutter problem as such because the number of clusters is unknown. Moreover, the representative icons might still overlap after clustering.

### B. Overlap-based clustering

The first cluster is created from the first data object, and all other objects within a given distance threshold are joined to this cluster. The process then continues similarly for the next object that has not yet joined to any cluster. The algorithm is given below:

```
overlapBasedClustering(X, N, TH)
k = 1
FOR  i=1 TO N
  visited[i] = FALSE
  label[i] = 0
FOR  i = 1 TO N
  IF NOT visited[i]
    visited[i] = TRUE
    createCluster(X, N, TH, label, visited, i, k)
    k = k + 1

createCluster(X, N, TH, label, visited, i, k)
label[i] = k
FOR  j=1 TO N
  IF (NOT visited[j]) AND (distance(i, j) < TH)
    label[j] = k
    visited[j] = TRUE
```

The time complexity is O(*KN*), because the function createCluster is called *K* times, where $K \leq N$ is the number of clusters. The main disadvantage is that the clustering result depends on the order of processing data.

### C. DBSCAN

DBSCAN is a density-based clustering algorithm that aims at finding arbitrary shape clusters. Its basic idea is to create clusters from points whose neighborhood within a given radius (*eps*) contain a minimum number (*minPt*) of other points [31]. Using every such a point, the algorithm grows a cluster by joining other points that are close to the cluster. Time complexity of the original DBSCAN is O($N^2$) but some efforts [32-33] have been made to reduce it close to O(*N*). In clutter removal of icons, the *minPt* must be set to 1 because a single separated object should also be considered as a cluster, and *eps* is set to the distance threshold that guarantees no overlap. The cluster growing is not needed because we do not aim at finding natural clusters. Therefore, DBSCAN is not a suitable choice for the clutter problem.

### D. Centroid-linkage

Agglomerative clustering is a bottom-up approach in which each object is initially considered as its own cluster. Two closest clusters are then iteratively merged [34]. Several criteria have been proposed for selecting the next two clusters to be merged such as *single-linkage*, *average-linkage*, *complete-linkage*, *centroid-linkage*, and *Ward's method*. Both *centroid-linkage* and *Ward's method* are applicable to the clutter removal problem because the overlap of representative icons is checked based on the distance between cluster centroids. The merging process continues until the distance between the centroids of the next two clusters to be merged exceeds a threshold that guarantees no overlap. The pseudo code of fast implementation of the centroid-linkage algorithm based on the solution introduced in [35] is shown in the next page.

Time complexity of the basic agglomerative clustering is O($N^3$) but the above solution reduces it to O($\alpha N^2$), where $\alpha << N$ in the above algorithm due to employing a nearest neighbor table that uses only O(*N*) memory. The algorithm can still be too slow for real-time applications. In [34], an algorithm based on k-nearest neighbor graph is proposed in order to improve the speed close to O(*N*log*N*) with a slight

decrease in accuracy. However, graph creation is the bottleneck of the algorithm, and should be solved. Otherwise, this step dominates the time complexity.

```
centroidLinkage(X, N, TH)
Set each object to its own cluster
k = N
iMin = createNNTable(X, N)
[i1, i2, dist] = findClosestClusters(iMin, k)
WHILE dist < TH
  mergeAndUpdate(iMin, k, i1, i2)
  k = k – 1
  [i1, i2, dist] = findClosestClusters(iMin, k)
createNNTable(X, N) → iMin
FOR i = 1 TO N
  iMin[i] = i
FOR i = 1 TO N
  FOR j = 1 TO N
    IF  distance(i, j) < distance(i, iMin[i])
      iMin[i] = j
findClosestClusters(iMin, k) → [i1, i2, dist]
i1 = 1
FOR i= 1 TO k
  IF distance(i, iMin[i])<distance(i1, iMin[i1])
    i1 = i
i2 = iMin[i1]
dist = distance(i1, i2)
mergeAndUpdate(iMin, k, i1, i2)
Merge cluster i2 in cluster i1
Update centroid of cluster i1
FOR i = 1 TO k
  IF iMin[i] = i2
    iMin[i] = i1
    FOR j = 1 TO k
      IF  distance(i, j) < distance(i, iMin[i])
        iMin[i] = j
Replace cluster i2 with the last cluster

distance(i, j) → dist
dist = Euclidean distance between X(i) and X(j)
IF i = j
  dist = MAX
```

### *E. Grid-based clustering*

*Grid-based clustering* consists of three main steps: grid-construction, initial clustering, and merge. The space containing the objects is first segmented by dividing each dimension into a predefined number of bins [29]. This provides rectangular grid cells, see Fig. 7. In the second step, initial clusters are formed by assigning each object to a cell simply by indexing without any need for distance calculation [36]. Each cell corresponds to one cluster. Centroids and other summary information such as the number of objects and density are then calculated for the clusters. The rest of the process is performed only on the non-empty cells that contain some objects.

In the third step, final clusters are formed by merging the neighboring cells according to some closeness criterion such as density or connectedness, see Fig. 7. Finding a suitable criterion for merging is not trivial because different criteria can lead to different clustering results [29]. Pseudo code of this overall algorithm is given below:

```
gridBasedClustering(X, N, cellSize)
// Step 1: Grid construction
region = bounding box of data X
Set grid for the region
Set indices of the cells
// Step 2: Initial clustering
K=0
FOR i=1 to N
  Find the cell index (m, n) for the object X[i]
  IF the cell[m, n] is empty
    K=K+1
    Create new cluster K
  j = cluster index of the cell (m, n)
  Update information of cluster j
// Step 3: Merge
FOR k=1 to K
  Check neighbors of cluster k and merge if
needed
```

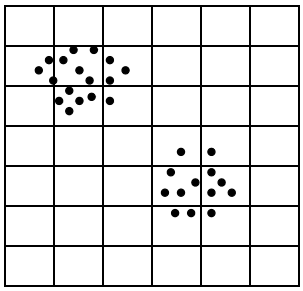


Figure 7. Some of neighboring cells should be merged to form natural clusters

The grid construction step contains the setting of the required parameters, which takes only O(1) time. The time complexity of the assignment step is O(*N*) because every object is processed. The third step is performed on the *K* initial clusters: the cells that contain objects. Since $K<<N$ (especially in 2-*D*), the overall time complexity is O(*N*). This makes the grid-based clustering a suitable choice for the real-time clutter removal problem. Memory complexity is O(*N*) and no distance calculation is required between the objects.

A few challenges have been reported for grid-based clustering methods such as finding clusters with variable densities, determining the size of grid cells, limitation of rectangular cells to fit the shape of clusters, and dimensionality problem [24],[29],[36]. However, most of the challenges are related to finding natural clusters or high dimensional data. Finding a suitable size for grid cell is not trivial because a small size leads to more cells, and therefore more computations, while a coarse cell size results in lower accuracy due to merging far away objects. However, in clutter removal of icons, the cell size is concluded directly from the size of the icons (for example maximum size). The only remaining issue is that two close objects might be clustered separately if located at the border of two cells. This misleads the user about the real locations of these objects.

TABLE I. COMPARISON OF CLUSTERING ALGORITHMS FOR CLUTTER REMOVAL OF ICONS

| Clustering algorithm | Item | Memory complexity | Supporting large data | Supporting parallel processing |
|---|---|---|---|---|
| K-means | O(IKN) | O(N) | No | No |
| Overlap-based | O(KN) | O(N) | Yes | No |
| DBSCAN | O(NlogN) | O(N) | No | No |
| Centroid-linkage | $O(N^2)$ | $O(N^2)$ | No | No |
| Grid-based | O(N) | O(N) | Yes | Yes |

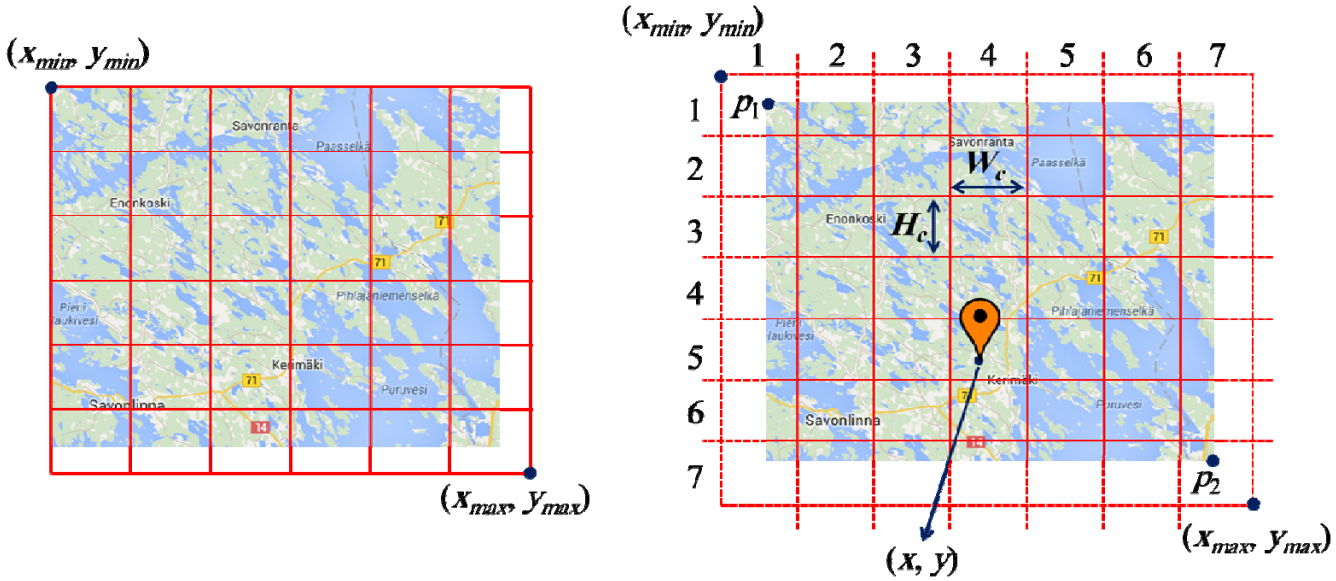



Figure 8. Starting grid from the beginning of the map view (left), and fixed grid starting from the beginning of the whole world but considering only the cells covering the map view

Grid-based clustering is simple to implement, and its time and memory complexities are better than those for other methods, see Table I. Moreover, parallel processing in order to increase the speed for large data can be perfectly applied to grid-based clustering. We present next the technical details of grid-based clustering for the clutter problem.

## IV. Grid-Based Clustering for Clutter Removal

This section presents the detailed procedure of grid-based clustering in order for clutter removal of icons on the map.

### A. Coordinates system

The location of a data object is represented by latitude and longitude, which are measured in degrees, minutes, and seconds of the globe sphere, or for computational purposes, simply in decimal degree. In Mercator projection, the areas far from the equator are exaggerated and it is not possible to find a fixed height for the grid cells, and a single distance threshold for avoiding overlap of icons, which have certain width and height in pixels. Therefore, we construct the grid in Cartesian coordinate system in pixel rather than degree, and convert the latitude and longitude of the objects to pixel for a certain zoom level as follows:

$$x = 128 + m \times 10^{-6} \times R \times lon$$
$$y = 128 + \frac{m \times R}{2} \times Ln\left[\frac{1+\sin(lat)}{1-\sin(lat)}\right] \quad (4)$$

where m=6.3952×10-6 is a scaling factor, R=6.371×106 is the earth radius, and lon and lat are in the range (-π, π) and (-π/2, π/2) respectively. The value (x, y) represents the coordinates of a point within a picture of size 256x256, which corresponds to the lowest zoom level (zero) in Google maps. For a higher zoom level z (up to 21), the coordinates are derived from x and y as follows:

$$x' = x \times 2^z \quad , \quad y' = y \times 2^z \quad (5)$$

### B. Grid construction

For a given map view, the grid is usually built starting from top-left corner, see Fig. 8 (left). However, this approach has a drawback. When the user pans the map, some new objects enter and some objects move out from the map view, and therefore, a new clustering is applied. Consider two clusters with 3 and 7 objects, respectively, in Fig. 9 (left); then, after horizontal panning to the right by the amount corresponding to 40% of the cell size, the objects will divide into two other clusters with 6 and 4 objects, respectively. This artifact happens because the new grid does not match with the old one, and objects might be assigned to different cells in the new grid. To avoid this problem, we set a fixed grid starting from the beginning of the whole world but consider only the cells which are completely or partly in the current map view, see Fig. 8 (right). This makes the grid invariant of panning.

Figure 9. Horizontal panning to the right causes different clusters when the grid is set according to the top-left of map view

The number of rows and columns in the map view are calculated as:

$$n_{row} = \left\lceil \frac{y_{\max} - y_{\min}}{H_c} \right\rceil \quad n_{column} = \left\lceil \frac{x_{\max} - x_{\min}}{W_c} \right\rceil \quad (6)$$

where $(W_c, H_c)$ is the size of a grid cell, and the points $(x_{min}, y_{min})$ and $(x_{max}, y_{max})$ are calculated according to the points $p_1$ and $p_2$ of the bounding box of the map view. A cell is then identified according to its row and column indices, which are in the range of $[1, n_{row}]$ and $[1, n_{column}]$, respectively, see Fig. 8.

### C. Initial clusters in cells

In this phase, the objects are assigned to the cells. We go through the objects one by one and calculate its corresponding cell. Row and column of the corresponding

cell of an object at the location $(x, y)$ is calculated as:

$$row = \left\lfloor \frac{y - y_{\min}}{H_c} \right\rfloor \quad column = \left\lfloor \frac{x - x_{\min}}{W_c} \right\rfloor \tag{7}$$

The cells that contain some objects become the initial clusters. The centroid of a cluster is calculated by averaging the locations of all the objects in the cluster. The initial clusters result in a fixed SSE for a certain input data. The cluster information that is used in the rest of the process includes the number of objects $n$, centroid $(x, y)$ and cluster representative. We calculate the size of the icon for each cluster $i$ from the size of the cluster $n_i$ using the following logarithmic function:

$$\begin{aligned} W_i &= W_{\min} + [\alpha \log_{10}(n_i)] \\ H_i &= H_{\min} + [\alpha \log_{10}(n_i)] \end{aligned} \tag{8}$$

where [.] is the rounding function, $W_{\min}$ and $H_{\min}$ are the minimum width and minimum height of the icons and $\alpha$ is the increase rate for the width and height. There is a trade off in the choice of $\alpha$. Large increase rate can lead to very big icons whereas small increase rate makes the difference of cluster sizes unnoticeable. In this work, we fix $\alpha$=8.

### D. *Merging overlapping clusters*

The representative icons of the clusters in neighboring cells may overlap when the distance between the centroids of the clusters is small, see Fig. 10. The overlap can be eliminated either by spatial distortion of representative icons or by merge. Spatial distortion is performed by moving the centroid location of a cluster away from the overlapping cluster [13],[21]. If the icon moves far, an arrow can be used to point from the icon to the original location. In case of many overlapping icons, the problem of finding good places for the icons becomes complicated. We therefore use the merge approach. After merging two clusters, their new centroid might place in anywhere within the two cells, or even move into a third cell, see Fig. 10. However, we keep the index of the first cluster for simplicity. This has no side effect to the clustering results, and it is needed for the server-side clustering; where the indices of initial cells are required for accessing the objects in a cluster.

We process as follows. First, we go through all clusters one by one to determine overlapping clusters according to (1). For every overlapping clusters $i$ and $j$, we calculate their merge cost as the increase in the total SSE:

$$\Delta SSE = \frac{n_i n_j}{n_i + n_j} \left\| c_i - c_j \right\|^2 \tag{9}$$

We select the clusters to be merged that result in minimum increase in the SSE, which is similar to the idea of Ward's criterion in agglomerative clustering [37]. The size of the representative icon of the new cluster is updated using (8). After the merge, the new cluster is checked for possible overlap to other clusters and the process then continues until no overlap remains. This merging approach does not guarantee the global optimal but it merely removes all overlaps by locally minimizing SSE in each step.

## V. Real-Time Clustering on Server-Side

Our goal is to apply clustering on server-side in order to limit the download size. Moreover, we want to support dynamic queries to the data without limiting to a small predefined set of queries. In this section, we first compare server-side and client-side approaches, and study existing server-side solutions and their limitations. We then propose a server-side approach based on grid-based clustering. We first need to define two types of queries that the user requests to see the desired results on the map: *spatial* and *non-spatial*.

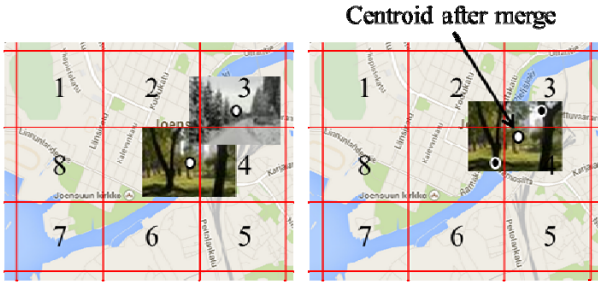



Figure 10. Overlap of two representative icons in neighboring cells (left) and merging clusters (right)

*Spatial query*: The user selects the map view and requests to see all data in this area. No other parameters are needed to specify which objects. The entire data can be clustered offline on the server if only this type of query is requested. The corresponding clusters in the region are then extracted from the pre-computed clustering. Zooming, panning, and opening a cluster are examples of how spatial query is initiated by the user. In the cases of zooming and panning, the map view is set directly using the map API tool. For opening a cluster, a new map view is calculated and set according to the bounding box of the objects in the cluster.

*Non-spatial query*: Instead of showing all the objects, the user selects a subset to be displayed based on other properties. For example, the user might want to see the pictures by a given person within a given time period, or the objects containing a given keyword. We refer to this as *non-spatial query* in contrast to spatial query. Pre-computing the clusters is possible but only for a predefined set of queries such as accessing all the data in the year 2015. However, in general, these types of queries are dynamic, and clustering must be performed real-time. This is because the set of data is dynamically retrieved based on the input parameters given by the user; the objects that match the query cannot be predicted in general.

### A. *Server-side vs. client-side*

The main advantage of server-side clustering approach is that it limits download size by sending only the summary information of clusters to the client. In client-side clustering, all the data are sent to the client, which provides two advantages. First, processing data on client relieves the server from overwhelming clustering requests. Second, no additional request to the server is needed for interactions such as zooming in the map or opening a cluster. However, obtaining the entire results from the server can cause high traffic load on the network. Suppose that 100,000 data objects are transferred to the client and 12 bytes are required for id, latitude, and longitude per object. The transmission load would sum up to 1.2 Mbytes, which is considerable amount for a low speed internet.

In server-side approach, the transmission load is limited by sending only the information of clusters. Since the

number of clusters is limited according to the sizes of display and icon, there is an upper limit for the transmission load, which is independent on the number of objects. Several server-side clustering APIs have been developed in recent years [25],[38].

### *B. Existing methods*

The existing methods that deal with visualization of large data on maps can be classified into two groups:

1. Hierarchical clustering on the server by pre-computing the clusters
2. Using data cubes

The first group [2],[13],[23],[25],[39] clusters entire data on the server by employing a hierarchical structure such as KD-tree or R-tree. Querying for a region is then performed in O(log*N*) time by finding the target clusters in the suitable level of the hierarchy. The hierarchical structure can also provide scalable visual representations [2],[9]. However, clustering of entire data does not support non-spatial queries. Elmqvist and Fekete [9] provide an overview on hierarchical aggregation of data to support visualization requirements such as panning, zooming, and opening a cluster.

To address non-spatial queries for large data sets and to support quick exploration, several researchers use data cubes [5],[26]. Data cubes are structures that build aggregations across every possible set of dimensions of data [26]. *imMens* [5] decomposes multi-dimensional data cubes into binned data tiles of reduced dimensionality and performs accelerated query processing and rendering on the GPU. For real-time interaction, the binned data tiles are pre-computed. *imMens* visualizes the aggregates on the map as geographic heatmaps which are 2-D binned plots. However, data cubes do not allow queries to individual record like traditional databases and they need considerable amount of memory. For example, in [26], after using the nanocubes for reducing memory, 45 Gbytes is needed for 210 million points (214 Mbytes for 1 million points). Moreover, this can be applied only to a limited (up to 5) number of data dimensions. As the number of dimensions increase, the required memory becomes quickly impractical. We note that data cubes can be used jointly with grid-based clustering if so wanted. These two approaches do not exclude each other.

### *C. Proposed approach*

Fig. 11 shows the flow of our server-side clustering approach both for non-spatial and spatial queries. The sizes of the grid cell and the map container box in the interface are sent to the server as parameters. In a non-spatial query, the map view that contains all the resulting data objects is obtained. In contrast, in a spatial query, the map view is specified by the user and sent to the server. The objects inside the map view resulted from the query are selected. The rest of the process is the same for both types of queries, where the objects and map view are inputs to the initial clustering. In a spatial query, we first apply the corresponding non-spatial query to retrieve the results from database. However, this is not needed if a faster approach is used to store the results of the last non-spatial query on the server so that the corresponding results for the given region specified by a spatial query are extracted. The following information of each cluster is collected:

1. Centroid of cluster: $(x,y)$
2. Number of objects: $(n)$
3. Bounding box: $(x_{min},y_{min})$ and $(x_{max},y_{max})$
4. Cluster representative

The information of each cluster representative is also sent to the client. For example, for photo collection, the filename of the representative photo is sufficient for displaying image thumbnail, see Mopsi cluster representation in Fig. 3.

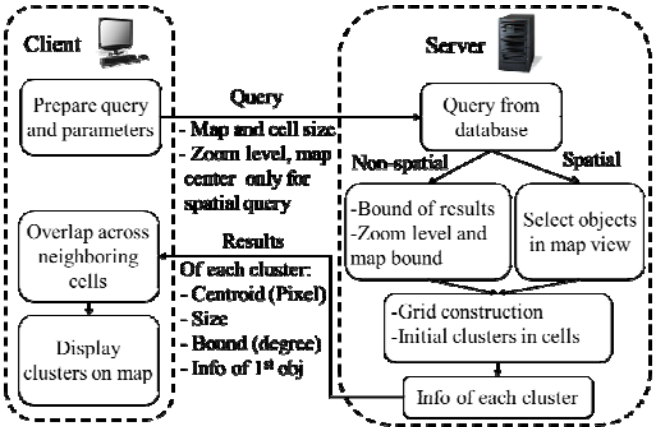



Figure 11. The proposed server-side clustering approach

### *D. Interactions using bounding box*

The bounding box of a cluster is needed for opening, and accessing the objects inside the cluster. To open a cluster, the map view is set using the bounding box of the cluster and a new spatial query is applied to retrieve the objects in the map view. To access the information of the $m^{th}$ object in the cluster, a query is applied to retrieve the identifiers (*id*) of the objects in the bounding box. We always sort the results in the same order so that the $m^{th}$ *id* in the list of results would correspond to the $m^{th}$ object. The object's *id* can then be used to retrieve its information. The same approach can be used to obtain the information of $k$ consequent objects of a cluster. This scenario applies only when the bounding box does not overlap with other clusters. However, overlap might happen when merging clusters. Next, we demonstrate the problem and explain our solution to solve it.

Consider the three clusters in Fig. 12. Clusters 1 and 2 should be merged because of the overlap of their representatives. To open the merged cluster when the user clicks on its representative, a spatial query is applied to the bounding box of the objects in the cluster. However, a problem appears that the bounding box of the merged cluster contains some objects from the cluster 3. We solve this by applying spatial queries for the initial cells of the merged cluster. We therefore send initial clusters in cells without any merge to the client. Merge step is performed on client, and the initial clusters and the order of merging are stored. To retrieve the information of an object $x$ in a cluster, its corresponding initial cluster $C$ and the index of $x$ in $C$ are determined. The *id* of $x$ is then obtained by applying a spatial query for the bounding box of $C$, and the information

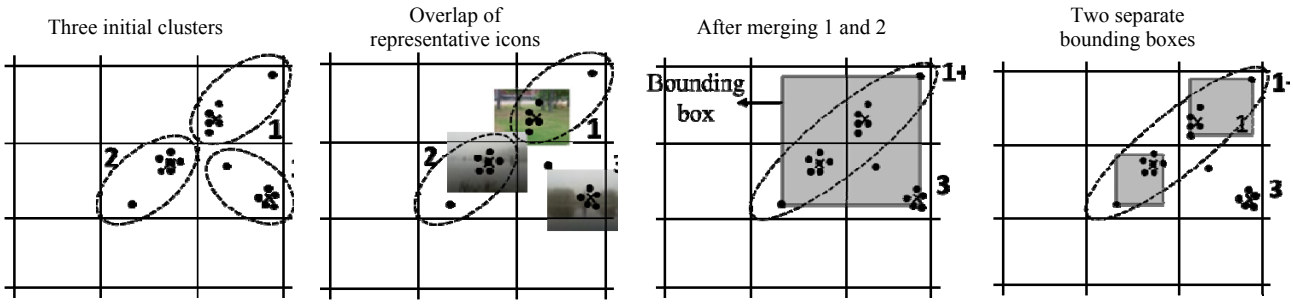



Figure 12. Bounding box of the merged clusters 1 and 2 overlaps cluster 3 and therefore two separated bounding boxes are used

of $x$ is retrieved by applying another query using the *id*.

### E. Scalability

In this section, we discuss the time and memory scalabilities of the proposed server-side approach. In general, grid-based clustering allows high scalability regarding both memory and time, and our method is no exception. The initial clustering is the most time consuming step with time complexity of O($N$), see Section III.

Initial clustering can be performed independently for every cell. We can query for the data inside a cell and calculate the information of the cluster. The data in each cell can be queried part by part if there is too much data in it. Two clusters resulted from two parts of data can iteratively be merged to provide the overall cluster for the cell. The information of the merged cluster is calculated from the information of two clusters. The number of objects is the sum of the number of objects of the two clusters, and the bounding box is the union of the two bounding boxes. The representative is taken from the first cluster, and the centroid is calculated as:

$$\begin{aligned} x &= (n_1 x_1 + n_2 x_2)/(n_1 + n_2) \\ y &= (n_1 y_1 + n_2 y_2)/(n_1 + n_2) \end{aligned} \tag{10}$$

where $(x_1,y_1)$ and $(x_2,y_2)$ are the centroids, and $n_1$ and $n_2$ are the number of objects of the two clusters. Therefore, there is no limitation regarding memory. However, in our current implementation, we load the whole data from the database at once, which limits the memory scalability with the current hardware to about 100 Mbytes.

The processing time on our server is 0.25 second for one Mbytes data. Assuming that 1 second is acceptable for a real-time interaction, the processing time limits the scalability of the current implementation to 4 Mbytes data. This excludes the time for loading data from database. There are many techniques to improve the interactions with the database but it is out of the focus of this paper.

Time scalability can be further improved according to the above-mentioned properties of the grid-based clustering by applying parallel or multi-thread processing, which require more investments on hardware. The subsets of the cells or even every cell can be processed in parallel.

## VI. Experiments

To evaluate the performance of the proposed server-side clustering and compare it with client-side clustering, we have provided a web page (http://cs.uef.fi/mopsi/markerClustering) that uses photos from Mopsi (http://cs.uef.fi/mopsi). We have implemented the server-side approach in *C* programming language. We have also implemented a client-side API in Javascript to compare with our server-side approach. Firefox 34.0.5 has been used as the web browser. The server and client specifications are as follow:

Client:
1. Windows 7, 64-bit
2. CPU: Inter(R) Core(TM) i3-2100, 3.10 GHz
3. Memory: 8 GB RAM

Server:
1. RedHat Enterprise Linux 7
2. Intel(R) Xeon(R) CPU E7-4860 v2 @ 2.60 GHz
3. Memory: 1000 GB RAM

To have 1,000,000 photos, we duplicated 20,000 of Mopsi photos, 50 times each, by randomly distributing their locations all over the world. For each photo, we have its id, location, title, and filename. We created 4 subsets (as txt files) containing 1000 (1K), 10,000 (10K), 100,000 (100K), and 1,000,000 (1M) photos. In the following experiments, we report the time taken for the clustering process only, excluding the time for reading the files. In practice, the photo data is retrieved from database and different techniques could be used to speed up the queries. However, they are out of the focus of this study. Note that for every task such as opening a cluster or accessing the information of the objects in a cluster, the files should also be read. We set the cell size equal to 60x50, and the distance threshold to $T$=5 pixels.

### A. Comparison of clustering algorithms

In this experiment, we compare the proposed grid-based clustering algorithm to overlap-based and centroid-linkage algorithms. The processing time of clustering is reported in Table II for several sizes of input data. Grid-based and overlap-based clustering provide reasonable time for real-time interaction, whereas centroid-linkage needs 15.5 seconds for clustering of only 10 Kbytes data, and it becomes impractical for 100 Kbytes and more. Overlap-based clustering is slightly slower than grid-based clustering as expected because their time complexities are O($KN$) and O($N$), respectively. Grid-based clustering is preferred because it is suitable for parallel processing, and it provides details on demand and opening cluster functionalities using the bounding box of a cluster. Google MarkerClusterer v3, which is a client-side clustering API, uses a method similar to overlap-based clustering, but for variable size icons, that makes it more time consuming. Table III reports the time for the clustering process and adding representative icons to the

map in the proposed client-side solution and Google MarkerClusterer. As the size of data increases, the proposed API outperforms MarkerClusterer, where for 1,000,000 items, it is almost 100 times faster.

We evaluate the clustering result by calculating *sum-of-squared errors* (SSE), i.e. the total (squared) distance between the data points and their corresponding cluster centroid, see Equation (2). We then normalize the values and report *mean squared error* (MSE):

$$MSE = SSE / N \quad (11)$$

where $N$ is the number of data objects. The value measures the average variance of the clusters. The smaller the number, the more compact and therefore better, is the clustering. The results are reported in Table IV for the subset of size $N$=1000.

The parameters for all the algorithms are set so that no cluster overlap appears. Each algorithm provides different number of clusters (K). For example, centroid-linkage results in K=54, overlap-based in K=47, and grid-based in K=45 clusters. Usually, the more clusters we have, the lower is the MSE-value. For fair comparison, we therefore tuned the parameters of the algorithms so that they all would result in exactly K=45 clusters. From the results we make the following observations.

Overlap-based and grid-based algorithms provided almost the same MSE-values (713 vs. 716), while centroid-linkage gave the best result (605). Considering all aspects such as quality, running time, memory requirement, and suitability for parallel processing, we conclude that grid-based clustering is the best overall choice for the problem.

TABLE II. PROCESSING TIME (SECONDS) OF THREE SLUSTERING ALGORITHM IN CLIENT-SIDE APPROACH

| **Data size** | 1K | 10K | 100K | 1M |
|---|---|---|---|---|
| Centroid-linkage | 0.09 | 15.5 | - | - |
| Overlap-based | 0.01 | 0.02 | 0.10 | 0.93 |
| Proposed Grid-based | <0.01 | <0.01 | 0.01 | 0.08 |

TABLE III. THE OVERALL PROCESSING TIME (SECONDS) IN THE PROPOSED CLIENT-SIDE API AND GOOGLE MARKERCLUSTERER API

| **Data size** | 1K | 10K | 100K | 1M |
|---|---|---|---|---|
| Proposed grid-based | 0.23 | 0.34 | 0.64 | 2.4 |
| Google maps API | 0.30 | 1.7 | 20 | 229 |

### *B. Server-side vs. client-side*

Running time of the client-side and server-side approaches has linear dependency on the size of data. The initial clustering and the merge step are very fast in both approaches. In the client-side approach, the time taken for downloading data is the bottleneck even with a high speed internet (400 Kbytes/sec). In the server-side approach, the download time is independent on the size of data. The overall time grows at a significantly slower rate in the server-side approach than in the client-side approach, see Table V. This makes it possible to use the server-side approach in real-time applications even with a large data of size 1,000,000 items. In the client-side approach, the clustering is run by the internet browser, which uses interpreted language such as Javascript. In the server-side approach, however, faster programming languages such as C and Java can be used.

In the client-side approach, the download size is proportional to the size of data set, see Table VI. In case of 1,000,000 data objects, the time needed to download data is around 26 seconds even using a high speed internet, which means that the client-side approach is not suitable for real-time applications of this magnitude.

The download size in the server-side approach is independent on the size of data, and it depends only on the number of the initial clusters in grid cells, which are produced by the grid-based clustering algorithm. This property makes the real-time interaction possible for the users with different internet speeds.

TABLE IV. CLUSTERING QUALITY (MSE) WITH THE SUBSET OF SIZE N=1000

| **Clustering algorithm** | **Same parameters** | | **Same number of clusters** | |
|---|---|---|---|---|
| | MSE | # Clusters | MSE | # Clusters |
| **Centroid-linkage** | 500 | 54 | 605 | 45 |
| **Overlap-based** | 643 | 47 | 713 | 45 |
| **Proposed Grid-based** | 716 | 45 | 716 | 45 |

TABLE V. PROCESSING TIME (SECONDS) OF CLUSTERING IN CLIENT-SIDE AND SERVER-SIDE APPROACHES

| **Data size** | | 1K | 10K | 100K | 1M |
|---|---|---|---|---|---|
| **Client-side** | Initial clustering | 0.000 | 0.003 | 0.012 | 0.077 |
| | Merge | 0.004 | 0.006 | 0.007 | 0.010 |
| | Downloading data | 0.019 | 0.062 | 1.6 | 26 |
| | Displaying representatives | 0.21 | 0.32 | 0.62 | 2.2 |
| | Total | 0.233 | 0.391 | 2.239 | 28.287 |
| **Server-side** | Initial clustering | 0.000 | 0.001 | 0.060 | 0.059 |
| | Merge | 0.004 | 0.006 | 0.007 | 0.010 |
| | Downloading data | 0.002 | 0.002 | 0.002 | 0.002 |
| | Displaying representatives | 0.19 | 0.33 | 0.60 | 2.15 |
| | Total | 0.196 | 0.339 | 0.669 | 2.221 |

TABLE VI. DOWNLOAD SIZE (KILOBYTES) IN CLIENT-SIDE AND SERVER-SIDE APPROACHES

| **Data size** | 1K | 10K | 100K | 1M |
|---|---|---|---|---|
| **Client-side** | 74 | 780 | 7,700 | 77,000 |
| **Server-side** | 13.4 | 14.7 | 14.8 | 14.8 |

## VII. CONCLUSION

We have proposed a novel web mapping system based on clustering. It allows users to make dynamic queries and

access the result in real-time. The system is unique, as we are unaware of any other similar server-side systems that allow presenting query results up to 1M objects. Most existing systems are limited to static predefined queries, or they only have client-side solution. For example, GoogleMaps can handle data real-time only up to few thousands only because of bandwidth limitation of the data transfer.

The proposed system consists of a server-side clustering algorithm, and client-side functionalities to allow real-time access to zoom in the clusters. The system is suitable for real-time applications even in low bandwidth environment. It is also highly scalable as it easily extends to parallel processing. The results can be verified using our freely available API, which includes both server-side and client-side implementations.

## REFERENCES


[1] M. Nöllenburg, "Geographic visualization," in Human-centered visualization environments, pp. 257-294, 2007. doi:10.1007/978-3-540-71949-6

[2] J. Delort, "Hierarchical cluster visualization in web mapping systems," 19th Int. Conf. World Wide Web, pp. 1241-1244, 2010. doi:10.1145/1772690.1772892

[3] J. K. Rayson, "Aggregate towers: Scale sensitive visualization and decluttering of geospatial data," IEEE Symposium on Information Visualization (Info Vis' 99), pp. 92-99, 1999. doi:10.1109/INFVIS.1999.801863

[4] J. Korpi, P. Ahonen-Rainio, "Clutter reduction methods for point symbols in map mashups," The Cartographic Journal, vol. 50, no. 3, pp. 257-265, 2013. doi:10.1179/1743277413Y.0000000065

[5] Z. Liu, B. Iiang, J. Heer, "imMens: Real-time visual querying of big data," Computer Graphics Forum, vol. 32, no. 3pt4, pp. 421-430, 2013. doi:10.1111/cgf.12129

[6] J.-Y. Delort, "Vizualizing large spatial datasets in interactive maps," Advanced Geographic Information Systems, Applications, and Services (GEOPROCESSING), pp. 33-38, 2010. doi:10.1109/GEOProcessing.2010.13

[7] A. Jaffe, M. Naaman, T. Tassa, M. Davis, "Generating summaries and visualization for large collections of geo-referenced photographs," 8th ACM Int. Workshop on Multimedia Information Retrieval, pp. 89-98, 2006. doi:10.1145/1178677.1178692

[8] S. Ahern, M. Naaman, R. Nair, J. H. Yang, "World explorer: visualizing aggregate data from unstructured text in geo-referenced collections," 7th ACM/IEEE-CS Conf. Digital Libraries, pp. 1-10, 2007. doi:10.1145/1255175.1255177

[9] N. Elmqvist, J.-D. Fekete, "Hierarchical aggregation for information visualization: Overview, techniques, and design guidelines," IEEE Trans. on Visualization and Computer Graphics, vol. 16, no. 3, pp. 439-454, 2010. doi:10.1109/TVCG.2009.84

[10] I. Peca, H. Zhi, K. Vrotsou, N. Andrienko, G. Andrienko, "Kd-photomap: Exploring photographs in space and time," IEEE Conf. Visual Analytics Science and Technology (VAST), pp. 291-292, 2011. doi:10.1109/VAST.2011.6102479

[11] M. Cristani, A. Perina, U. Castellani, V. Murino, "Content visualization and management of geo-located image databases," CHI'08 Extended Abstracts on Human Factors in Computing Systems, pp. 2823-2828, 2008. doi:10.1145/1358628.1358768

[12] F. Girardin, F. Calabrese, F. Dal Fiore, C. Ratti, J. Blat, "Digital footprinting: Uncovering tourists with user-generated content," IEEE Pervasive Computing, vol. 7, no. 4, 2008. doi:10.1109/MPRV.2008.71

[13] C. Lu, C. Chen, P. Cheng, "Clustering and visualizing geographic data using geo-tree," IEEE/WIC/ACM Int. Conf. Web Intelligence and Intelligent Agent Technology-Volume 01, pp. 479-482, 2011. doi:10.1109/WI-IAT.2011.171

[14] D. A. Keim, H. Kriegel, "VisDB: Database exploration using multidimensional visualization," IEEE Computer Graphics and Applications, vol. 14, no. 5, pp. 40-49, 1994. doi:10.1109/38.310723

[15] F. H. Post, F. J. Post, T. Van Walsum, D. Silver, "Iconic techniques for feature visualization," 6th IEEE Conf. Visualization'95, p. 288, 1995. doi:10.1109/VISUAL.1995.485141

[16] E. Keogh, L. Wei, X. Xi, S. Lonardi, J. Shieh, S. Sirowy, "Intelligent icons: Integrating lite-weight data mining and visualization into GUI operating systems," 6th Int. Conf. Data Mining, pp. 912-916, 2006. doi:10.1109/ICDM.2006.90

[17] N. Cao, D. Gotz, J. Sun, H. Qu, "Dicon: Interactive visual analysis of multidimensional clusters," IEEE Trans. on Visualization and Computer Graphics, vol. 17, no. 12, pp. 2581-2590, 2011. doi:10.1109/TVCG.2011.188

[18] D. Fisher, "Hotmap: Looking at geographic attention," IEEE Trans. on Visualization and Computer Graphics, vol. 13, no. 6, pp. 1184-1191, 2007. doi:10.1109/TVCG.2007.70561

[19] A. Mayorga, M. Gleicher, "Splatterplots: Overcoming overdraw in scatter plots," IEEE Trans. on Visualization and Computer Graphics, vol. 19, no. 9, pp. 1526-1538, 2013. doi:10.1109/TVCG.2013.65

[20] B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations," IEEE Symposium on Visual Languages, pp. 336-343, 1996. doi:10.1109/VL.1996.545307

[21] G. Ellis, A. Dix, "A taxonomy of clutter reduction for information visualisation," IEEE Trans. on Visualization and Computer Graphics, vol. 13, no. 6, pp. 1216-1223, 2007. doi:10.1109/TVCG.2007.70535

[22] J.-D. Fekete, C. Plaisant, "Interactive information visualization of a million items," IEEE Symposium on Information Visualization, INFOVIS, pp. 117-124, 2002. doi:10.1109/INFVIS.2002.1173156

[23] W. Wang, J. Yang, R. Muntz, "STING: A statistical information grid approach to spatial data mining," VLDB, vol. 97, pp. 186-195, 1997. doi: doi:10.1.1.106.7154

[24] R. Agrawal, J. Gehrke, D. Gunopulos, P. Raghavan, "Automatic subspace clustering of high dimensional data for data mining applications," ACM SIGMOD Int. Conf. Management of Data, vol. 27, no. 2, pp. 94-105, 1998. doi:10.1145/276304.276314

[25] J. Dabernig, "Geocluster: server-side clustering for mapping in Drupal based on Geohash," M.Sc. Thesis, Faculty of Informatics, TU Wien University, Austria, 2013.

[26] L. Lins, J. T. Klosowski, C. Scheidegger, "Nanocubes for real-time exploration of spatiotemporal datasets," IEEE Trans. on Visualization and Computer Graphics, vol. 19, no. 12, pp. 2456-2465, 2013. doi:10.1109/TVCG.2013.179

[27] D. Nguyen, H. Schumann, "Taggram: Exploring geo-data on maps through a tag cloud-based visualization," 14th Int. Conf. Information Visualisation (IV), pp. 322-328, 2010. doi:10.1109/IV.2010.52

[28] R. T. Ng, J. Han, "CLARANS: A method for clustering objects for spatial data mining," IEEE Trans. on Knowledge and Data Engineering, vol. 14, no. 5, pp. 1003-1016, 2002. doi:10.1109/TKDE.2002.1033770

[29] D. R. Edla, P. K. Jana, "A grid clustering algorithm using cluster boundaries," World Congress on Information and Communication Technologies (WICT), pp. 254-259, 2012. doi:10.1109/WICT.2012.6409084

[30] S. Na, L. Xumin, G. Yong, "Research on k-means clustering algorithm: An improved k-means clustering algorithm," Intelligent Information Technology and Security Informatics (IITSI), pp. 63-67, 2010. doi:10.1109/IITSI.2010.74

[31] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," KDD, vol. 96, no. 34, pp. 226-231, 1996. doi: 10.1.1.121.9220

[32] B. Liu, "A fast density-based clustering algorithm for large databases," Int. Conf. Machine Learning and Cybernetics, pp. 996-1000, 2006. doi:10.1109/ICMLC.2006.258531

[33] L. Zhao, J. Yang, J. Fan, "A fast method of coarse density clustering for large data sets," 2nd Int. Conf. Biomedical Engineering and Informatics, BMEI'09, pp. 1-5, 2009. doi:10.1109/BMEI.2009.5305132

[34] P. Franti, O. Virmajoki, V. Hautamaki, "Fast agglomerative clustering using a k-nearest neighbor graph," IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 28, no. 11, pp. 1875-1881, 2006. doi:10.1109/TPAMI.2006.227

[35] P. Franti, T. Kaukoranta, D. Shen, K. Chang, "Fast and memory efficient implementation of the exact PNN," IEEE Trans. on Image Processing, vol. 9, no. 5, pp. 773-777, 2000. doi:10.1109/83.841516

[36] M. Steinbach, L. Ertöz, V. Kumar, "The challenges of clustering high dimensional data," New Directions in Statistical Physics: Springer, pp. 273-309, 2004. doi:10.1007/978-3-662-08968-2_16

[37] J. H. Ward Jr, "Hierarchical grouping to optimize an objective function," J. American Statistical Association, vol. 58, no. 301, pp. 236-244, 1963. doi:10.1080/01621459.1963.10500845

[38] W. Meert, "Clustering maps," M.Sc. Thesis, Faculty of Engineering, University of Leuven, Belgium, 2006.

[39] T. Zhang, R. Ramakrishnan, M. Livny, "BIRCH: an efficient data clustering method for very large databases," ACM Sigmod Record, vol. 25, no. 2, pp. 103-114, 1996. doi:10.1145/235968.233324