MAD-C: Multi-stage Approximate Distributed Cluster-Combining for Obstacle Detection and Localization

Downloaded from: https://research.chalmers.se, 2024-04-28 09:59 UTC

Citation for the original published paper (version of record):
Keramatian, A., Gulisano, V., Papatriantafilou, M. et al (2019). MAD-C: Multi-stage Approximate Distributed Cluster-Combining for Obstacle Detection and Localization. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 11339: 312-324.
http://dx.doi.org/10.1007/978-3-030-10549-5_25

N.B. When citing this work, cite the original published paper.

research.chalmers.se offers the possibility of retrieving research publications produced at Chalmers University of Technology.
It covers all kind of research output: articles, dissertations, conference papers, reports etc. since 2004.
research.chalmers.se is administrated and maintained by Chalmers Library

(article starts on next page)

MAD-C: Multi-stage Approximate Distributed Cluster-Combining for Obstacle Detection and Localization

Amir Keramatian, Vincenzo Gulisano, Marina Papatriantafilou(✉), Philippas Tsigas, and Yiannis Nikolakopoulos

Chalmers University of Technology, Gothenburg, Sweden
{amirke,vinmas,ptrianta,tsigas,ioaniko}@chalmers.se

Abstract. Efficient distributed multi-sensor monitoring is a key feature of upcoming digitalized infrastructures. We address the problem of obstacle detection, having as input multiple point clouds, from a set of laser-based distance sensors; the latter generate high-rate data and can rapidly exhaust baseline analysis methods, that gather and cluster all the data. We propose MAD-C, a distributed approximate method: it can build on any appropriate clustering, to process disjoint subsets of the data distributedly; MAD-C then distills each resulting cluster into a data-summary. The summaries, computable in a continuous way, in constant time and space, are combined, in an order-insensitive, concurrent fashion, to produce approximate volumetric representations of the objects. MAD-C leads to (i) communication savings proportional to the number of points, (ii) multiplicative decrease in the dominating component of the processing complexity and, at the same time, (iii) high accuracy (with RandIndex > 0.95), in comparison to its baseline counterpart. We also propose MAD-C-ext, building on the MAD-C's output, by further combining the original data-points, to improve the outcome granularity, with the same asymptotic processing savings as MAD-C.

Keywords: Point cloud processing · Approximations · Fog computing

1 Introduction

LIDAR (LIght Detection And Ranging), used in e.g. autonomous vehicles and production environments, is a 3D scanning method to measure ranges with rotating pulsed lasers. A LIDAR sensor produces hundreds of thousands of points (*point clouds*) per rotation, at rates of several MBps. In the presence of occlusions, multiple such sensors could join local views from various angles into a consistent global view, an overlooked benefit, to the best of our knowledge, that can enhance resiliency and availability.

Challenges. Single-source point cloud object detection can be achieved with clustering methods [13]. With multiple LIDAR sensors, a *baseline* approach of

© Springer Nature Switzerland AG 2019
G. Mencagli et al. (Eds.): Euro-Par 2018 Workshops, LNCS 11339, pp. 312–324, 2019.
https://doi.org/10.1007/978-3-030-10549-5_25

clustering the union of the sources' point clouds is impractical due to its cumulative data volumes and rates resulting in prohibitive (i) processing costs and latency (at least linear in the number of point-clouds' sizes) even for parallel clustering approaches [9,11], and (ii) communication bandwidth requirements. *Edge/fog* continuous data processing (i.e., distributed clustering local to each LIDAR) could overcome these limitations. However, two *opposing goals* make such an approach challenging: sharing fine-grained data (to maximize the accuracy) versus coarse-grained data (to minimize communication overheads).

Contributions. We propose MAD-C, a multi-stage approximate distributed cluster-combining method for obstacle detection and localization. First, it clusters each point cloud at the edge, i.e. at each LIDAR sensor. Then, it computes a local constant size geometric summary of each object and combines it with those of other LIDARs (in time depending only on the number of objects and sensors, not on the point-clouds' sizes). We show that MAD-C's summaries are computable in a continuous way and can be combined in an order-insensitive concurrent fashion, exploiting data parallelism. Our extensive experimental study covers a wide spectrum of scenarios, including very demanding cases, showing that the common view produced by MAD-C is very close to that of the aforementioned baseline. We also observe significant improvements in processing and communication efficiency, which is all the more important for edge/fog architectures and use of the algorithm in time-sensitive applications.

In the following, Sect. 2 describes the system model, problem and preliminary concepts; Sect. 3 and Sect. 4 introduce MAD-C, its properties and its algorithmic implementation. Experimental evaluation is presented in Sect. 5, related work discussion in Sect. 6 and conclusions in Sect. 7.

2 Preliminaries

System Model. We consider K (≥ 1) asynchronous, interconnected nodes, each being at a known location and associated with a LIDAR and a processing unit (i.e. nodes are *edge/fog* devices). We assume the existence of a *spanning tree* for nodes to communicate and aggregate data. Each node knows its children and its parent. Let $\mathbb{S}$ denote the *sink* of the network (i.e., the tree-root), in charge of generating a global view from data from the other nodes. We first present our methods under the spanning tree and no-message-loss assumptions, for ease of the presentation. Later, we generalize using known results in distributed systems.

Each LIDAR, in each rotation, collects a *point cloud* centered at its location. The node can process the point-cloud locally, as well as communicate raw or processed data to others. Let $ptCloud_i$ be the point cloud from a full rotation of LIDAR L_i, consisting of n_i data points, as *node i's view*. A *(local) view* refers to an individual $ptCloud_i$ while a *merged point cloud* is the union of point clouds.

(a) 3 nodes (b) N1's view (c) N2's view (d) N3's view (e)merged view

Fig. 1. A scene with three LIDAR nodes

For simplicity and w.l.o.g we assume point clouds be obtained at the same time and views are expressed in the same coordinate system[1].

Problem Description. Using point clouds from K LIDARs, We want to detect objects, with low communication cost, while ensuring high quality of detection, data parallelism, as well as continuous, stream-compliant processing. The goal is to find a *map* that: (i) enumerates the objects and (ii) for each object, provides a representation (e.g. volumetric, or expressed as clusters of points). Besides detection and localization, this map can be used in scenarios with e.g. geo-fences.

Evaluation Criteria: (i) complexity in time, communication overhead and *(ii) accuracy* of the outcome. For the former we estimate the number of processing steps and the amount of information that needs to be communicated among the nodes. For the latter we use *Rand Index*, which is a similarity measure between two clusterings [15].

Example. *Figure 1(a) is to introduce running example to illustrate the problem and the functionality of our proposed methods. Parts 1(b–d) respectively visualize the local views of the 3 LIDARs. Figure 1(e) shows the merged point cloud. Notice that (i) there is at least one object missing in each local view and (ii) the views are complementary regarding the objects that are not occluded; e.g. they display almost non-overlapping segments of the car. Therefore, engaging more nodes to collect point clouds can result in higher accuracy.*

Background. Given a point cloud, there are several algorithms that segment the data points in it into *scene objects* [4,13], that our proposed methods can build on. Taking, e.g. Euclidean clustering, a point cloud would be partitioned into a set of clusters that correspond to objects and noise-points. To describe our methods we use the latter and for self-containment we paraphrase the definition from [13] (Ch. 4): Given n points in 3D space, a *Euclidean clustering* is a partitioning of them into some (unknown) number of disjoint sets (i.e. clusters),

[1] Else, pre-processing can transform them into a canonical system: depending on each LIDAR's disposition, a rotation matrix and a translation can be applied on its point-cloud, in constant time, in conjunction with the data-reading, along with filtering away ground points, a common pre-processing phase [8].

each containing at least a predefined number of points ($minPts$), so that pairs of points p_i and p_j are clustered together if $||p_i - p_j||_2 < \epsilon$, a predefined threshold. Points that don't belong to any cluster are characterized as *noise*.

3 The MAD-C Algorithm

We now describe MAD-C and how it meets the challenges described in Sect. 1. Due to space limitations, the proof arguments are briefly sketched. We consider a *baseline* that gathers all point-clouds and performs Euclidean clustering of these n points, with complexity $O(n \log n)$ expected processing steps [4,13].

In a nutshell, each node L_i in MAD-C locally detects objects in $ptCloud_i$ and forwards compact summaries of the local objects. The summaries get merged with the ones of other nodes along a spanning tree, up to $\mathbb{S}$, which then can deliver the set of global objects. Compared to the *baseline*, MAD-C drastically reduces data communication, while it pipelines and distributes the analysis.

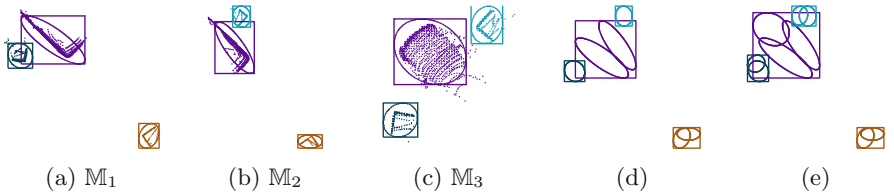
In the following we address how to efficiently (i) generate local maps, i.e. summaries of the local clusters in the local views; and (ii) gradually merge the maps in a deterministic fashion, despite network asynchrony.

Efficient Maps and Summarization of Local Clusterings. Consider two local clusters c_1 and c_2 from two views. How can we determine whether to *merge* them without having to calculate pairwise distances of points in c_1 and c_2? Simply considering distances between their centroids doesn't work, as the size and shape of clusters matter. Hash-based similarity checks don't apply either, since point clouds have different elements. To address these issues efficiently, MAD-C works on summaries of local clusters.

A summary of a cluster c should ideally (i) use small space (independent of $|c|$), (ii) be built incrementally as new points are added, (iii) be shared with peers as soon as all c's points are found and (iv) express the *volume* that c occupies, to allow comparisons and merging with close/overlapping clusters.

We noticed that *bounding ellipsoids* satisfy these requirements. With this in mind, and inspired by contour surfaces of a three-variable Gaussian distribution, which form 3D ellipsoids, we propose to fit Gaussian distributions to local clusters and represent them as *bounding ellipsoids*.

A Gaussian distribution is characterized by a mean vector $\mu \in \mathbb{R}^3$ (center of the distribution) and a covariance matrix $\Sigma \in \mathbb{R}^{3\times3}$ (spread of the distribution). The family of ellipsoids corresponding to the surface plots of a three-variable Gaussian distribution are characterized through $(x-\mu)^T \Sigma^{-1}(x-\mu) = \alpha^2$, where α is a constant (i.e. a parameter of MAD-C) which we call the *confidence step*. The unit eigen-vectors of Σ define the directions of the principal axes of the ellipsoid centered at μ [7]. The Gaussian fit through maximum likelihood estimation [7], allows to calculate a bounding ellipsoid incrementally by calculating N (c's number of points), $S = \sum_1^N p_i$ (cumulative vector sum of c's points) and $\tilde{\Sigma} = \sum_1^N p_i p_i^T$ (cumulative sum of outer products of c's points). As soon as c is complete, μ and Σ of the bounding ellipsoid $\mathbb{E}$ can be calculated through S/N and $\tilde{\Sigma}/N - \mu\mu^T$ respectively (Algorithm 1, l. 10).



(a) $\mathbb{M}_1$ (b) $\mathbb{M}_2$ (c) $\mathbb{M}_3$ (d) (e)

Fig. 2. (a,b,c) are local maps. (d) $\mathbb{M}_w = \mathrm{C}\ (\mathbb{M}_1, \mathbb{M}_2)$, (e) $\mathbb{M}_w = \mathrm{C}\ (\mathbb{M}_w, \mathbb{M}_3)$

Example. *Figure 2a, b, and c respectively show the local maps corresponding to Fig. 1b, c, and d. Ellipses symbolically illustrate the bounding ellipsoids. The delimiting boxes are explained later in this section.* We need some definitions to introduce next steps and properties of MAD-C.

Definition 1. *A map* $\mathbb{M}$ *is a set of objects. An object* $\mathbb{O}$ *is a set of ellipsoids.* $\|\mathbb{M}\|$ *denotes the number of ellipsoids in* $\mathbb{M}$.

In MAD-C, a node L_i produces a *local map* $\mathbb{M}_i$, i.e. a set of singletons, each containing a bounding ellipsoid approximating a local cluster in L_i's view (excluding noise points). The calculation of each ellipsoid's parameters can be embedded in the calculation of the clustering, at constant overhead per point.

Observation 1. *The representation of a bounding ellipsoid of cluster* c *is of size independent of* $|c|$. *The cost of calculating its parameters* μ *and* Σ *is constant per point in* c. *The representation of a map* $\mathbb{M}_i$ *is of size linear in* $|\mathbb{M}_i|$.

Algorithm 1. GenLocalMap(i)

1: $\mathbb{A}$: Euclidean clustering algorithm
2: α: confidence step in MAD-C
3: **while** $\exists\ p$ just clustered by $\mathbb{A}$ **do**
4: c : local cluster where p belongs
5: **if** c is new **then**
6: $c.N = 0; c.S = 0_{[3\times1]}; c.\tilde{\Sigma} = 0_{[3\times3]}$
7: $c.N = c.N + 1$; $c.S \leftarrow c.S + p$;
8: $c.\tilde{\Sigma} = c.\tilde{\Sigma} + p * p^T$
9: **for** $c \in$ detected clusters at L_i **do**
10: $\mu = c.S/c.N$; $\Sigma = c.\tilde{\Sigma}/c.N - \mu\mu^T$;
11: $\mathbb{E}$: an ellipsoid with a unique id
12: $\mathbb{E}.\mu \leftarrow \mu; \mathbb{E}.\Sigma \leftarrow \alpha^2\Sigma$;
13: Initialize $\mathbb{O}$ to contain $\mathbb{E}$
14: **for** $d \in \{x, y, z\}$ **do**
15: $\mathbb{O}.b_d = [\min proj_d\mathbb{E}, \max proj_d\mathbb{E}]$
16: $\mathbb{M}.addSingleton(\mathbb{O})$

Algorithm 2. UnifyChildren(i)

1: M_w = GenLocalMap(i)
2: **for all** Child $\mathbb{C}$ **do**
3: get($\mathbb{M}_\mathbb{C}$); M_w = Merge($M_w, \mathbb{M}_\mathbb{C}$)
4: send M_w to parent (if any)
5: **Function** Merge($M_w, \mathbb{M}_\mathbb{C}$)
6: $\mathbb{M}_r \leftarrow M_w \cup \mathbb{M}_\mathbb{C}$
7: **for all** $\mathbb{O}_i \in M_w$,$\mathbb{O}_j \in \mathbb{M}_\mathbb{C}$ **do**
8: **if** overlap($\mathbb{O}_i.b, \mathbb{O}_j.b$) **then**
9: **if** $\exists\mathbb{E} \in \mathbb{O}_i \wedge \exists\mathbb{E}' \in \mathbb{O}_j | \mathbb{E} \cap \mathbb{E}'$ **then**
10: $\mathbb{M}_r$.Merge($\mathbb{O}_i, \mathbb{O}_j$) *with:*
11: $\mathrm{b}_d = \mathbb{O}_i.\mathrm{b}_d \cup \mathbb{O}_j.\mathrm{b}_d, d \in \{x, y, z\}$
12: **RETURN** $\mathbb{M}_r$

Combining Ellipsoids and Maps from Multiple Nodes. While passing maps along the tree, each node merges its *working map* M_w (initially its local

map), with maps from its children, then it forwards the result to its parent (cf. Algorithm 2; shadowed lines are explained later in this section).

If merging is performed on the local point clouds rather than summaries, two local clusters become one if at least a pair of points (one from each) are within ϵ distance. Similarly, objects in the M_w and each child map $\mathbb{M}_{\mathbb{C}}$ are compared to detect if they contain ellipsoids satisfying such *matches*. If so, those objects are *merged*; i.e. the union of their ellipsoids is recognized as one object in M_w. In Sect. 4 we explain how (i) to integrate ϵ in an ellipsoid's representation, (ii) to check if two ellipsoids intersect and (iii) merge two objects, all in constant time.

If the baseline is performed on the merged point cloud excluding noise, then it generates clusters consisting of one or more local clusters because local clusters do not break into smaller pieces in the merged point cloud. Hence:

Lemma 1. *Applying the baseline on* $\cup_{i=1}^{K} ptCloud_i$ *results in clusters, each containing local clusters from local views. Likewise, the objects returned by* $\mathbb{S}$ *are sets of ellipsoids, each of the latter corresponding directly to a local cluster.*

Example. *Figure 2d shows the result of* MERGE $(\mathbb{M}_1, \mathbb{M}_2)$. *Figure 2e shows the* MERGE *result of the latter and* $\mathbb{M}_3$.

Lemma 2. *Operation* MERGE *on maps containing ellipsoids with unique identities, satisfies the reflexive, symmetric and associative properties.*

This follows through line 7 of Algorithm 2: if $\mathbb{O}_i$ and $\mathbb{O}_j$ have intersecting ellipsoids, they will be merged regardless of the order of execution, implying that MERGE satisfies properties of *conflict-free replicated data types* [12].

Corollary 1. *The network topology and timing asynchrony does not affect the final map at* $\mathbb{S}$. *Moreover, the* MERGE *operations can be executed using non-atomic multicasting, similar to gossiping or selective flooding, guaranteeing* eventually consistent *final outcome and inherent fault-tolerance properties.*

Corollary 1 implies the *spanning tree assumption can be lifted* and besides the sink node, any other node can construct the global map, if nodes broadcast their views in the network. We now study the processing and communication overhead of MAD-C, with a single sink.

Observation 2. $\|\mathbb{M}\|$ *equals* $\|\mathbb{M}_1\|+\|\mathbb{M}_2\|$ *if* $\mathbb{M}$ *is the result of* MERGE$(\mathbb{M}_1, \mathbb{M}_2)$.

Lemma 3. *Comparing objects* $\mathbb{O}_1$ *and* $\mathbb{O}_2$ *needs at most* $\theta(|\mathbb{O}_1| \times |\mathbb{O}_2|)$ *comparisons.* $O(\|\mathbb{M}_1\|\|\mathbb{M}_2\|)$ *processing steps is an upper bound on the computational cost of merging maps* $\mathbb{M}_1$ *and* $\mathbb{M}_2$.

This is because the number of comparisons for merging two maps is at most: $\left(\Sigma_{i=1}^{|\mathbb{M}_1|}\Sigma_{j=1}^{|\mathbb{M}_2|}|\mathbb{M}_1(i)||\mathbb{M}_2(j)|\right) \leq (\Sigma_{i=1}^{|\mathbb{M}_1|}|\mathbb{M}_1(i)|)(\Sigma_{j=1}^{|\mathbb{M}_2|}|\mathbb{M}_2(j)|) = \|\mathbb{M}_1\|\|\mathbb{M}_2\|$, while the cost of comparison and merging is constant (see Sect. 4). This bound is an overestimation of a worst-case because it counts unnecessary comparisons as well. The exact bound is data-dependent and hence harder to estimate in a

data-agnostic way, yet we experimentally study the number of comparisons in Sect. 5. In the following we study the role of topology in the above (still worst-case estimations), while later in this section we explain how to avoid unnecessary comparisons.

Let γ be the number of actual objects and K be the number of LIDARs. In each local view, while some objects might be entirely occluded, others might split into smaller ones, though not changing the order of magnitude of objects $O(\gamma)$ detected in the view, for the same ϵ and $minPts$ (cf. Sect. 2) as the baseline.

Lemma 4. MERGE*'s worst-case complexity is* $O(\gamma^2 K^2)$ *with a star or non-balanced tree topology and* $O(\gamma^2 K \lg K)$ *with a balanced binary tree.*

Recall that the expected cost of Euclidean clustering of n points is $O(n \log n)$ processing steps [4,13]. Let n_i be the size of $ptCloud_i$.

Corollary 2. *The overall computation cost of MAD-C is the sum of (i) the local clustering steps,* $\sum_{i=1}^{K} O(n_i \log(n_i))$*, (ii)* MERGE *operations steps, (Lemma 4, Lemma 3) and (iii) bounding ellipsoids calculation steps,* $\sum_{i=1}^{K} O(n_i)$ *(Observation 1).*

Lemma 5. *The total volume of data (e.g. in bytes) to be transferred between pairs of nodes in MAD-C is* $O(\gamma K)$*,* $O(\gamma K^2)$*, and* $O(\gamma K lg K)$ *under star, non-balanced tree, and balanced binary tree topologies, respectively.*

The above are determined through the ellipsoids to be transferred, using Observation 2 to find the number of ellipsoids that any node transfers to its parent.

Considering that (i) MAD-C relies on local clustering and assuming the latter is performed in parallel, and (ii) in the worst case, no MERGE operation takes place until the latest local clustering is completed, we have:

Corollary 3. *Completion time of MAD-C is determined by* $\max_{i=1}^{K} O(n_i \log n_i)$*, plus the time to complete* MERGE *operations and the time to transmit the maps.*

Avoiding unnecessary comparisons To avoid unnecessary one-to-one comparisons (e.g. when two objects occupy completely different parts of the scene), we propose *delimiting boxes* as a way of distinguishing objects, so that those that don't need to be compared, get grouped separately. An object's delimiting box is an axis-aligned rectangular shape that encapsulates all the ellipsoids corresponding to that object (Algorithm 2, l. 11). An ellipsoid's *delimiting box* is the smallest axis-aligned circumscribed rectangle encapsulating that ellipsoid, i.e. one closed interval for each axis (Algorithm 1, l.14).

Lemma 6. *If the delimiting boxes of* $\mathbb{O}_i$ *and* $\mathbb{O}_j$ *do not overlap, the two objects do not have overlapping ellipsoids.*

This follows from the definition of delimiting boxes and it helps to reduce the comparison costs, while the other properties shown in the analysis still hold.

MAD-C-ext: Delivering Data Point Labels Rather than Ellipsoids. The baseline determines a labeling/clustering tag for each data point in the merged point cloud. MAD-C too can be modified so that, as well as maintaining a M_w, each parent node combines point clouds from its children and its own, and it determines a labeling for the latter and forwards both to its parent.

4 Algorithmic Implementation of MAD-C

Ellipsoidal Overlap. Given a pair of ellipsoids $\mathbb{E}_a, \mathbb{E}_b$, the method described in [1] determines in constant time if they intersect. It characterizes $\mathbb{E}_a, \mathbb{E}_b$ respectively as $X^T AX = 0$ and $X^T BX = 0$, where A and B are 4×4 matrices derived from their centroids and covariance matrices by extending with a default row and column. $\mathbb{E}_a, \mathbb{E}_b$ overlap if there is at least an admissible eigenvector (one without a zero in the fourth dimension) of $A^{-1}B$ that satisfies both equations.

Aura: Integrating ϵ in Ellipsoids. If the minimum distance of pairs of points from two objects is less than ϵ, then they are grouped together by the Euclidean clustering algorithm. We target the same behaviour with the ellipsoidal models, adding an *aura* $\delta = \epsilon/2$ around them, simply by increasing lengths of the main axes by δ. This is achieved by manipulating the covariance matrix of the ellipsoid to be expanded. Suppose $V \Lambda V^T$ is the singular value decomposition of the covariance matrix. Since the lengths of the main axes of the ellipsoid are the entries in the diagonal matrix Λ, it suffices to update Λ to $\left(\Lambda^{0.5} + \delta.I\right)^2$.

Data Structure for Maps. Implementation of MAD-C requires a data structure supporting *map*s. As described in Sect. 3, a map is a set of objects, each being a set of ellipsoids. We employed a variant of disjoint-set data structure with path compression technique. In our implementation, ellipsoids are initially elements of a disjoint-set forest and objects are merged by merging their corresponding trees through a simple pointer operation, hence the merging cost is constant.

5 Experimental Evaluation

We study (i) how well the ellipsoids represent local objects, (ii) the quality of MAD-C's approximate clustering and (iii) the quality of the clustering from all the LIDAR nodes for both the baseline and MAD-C-ext. To complement MAD-C's MERGE and communication worst case costs (Lemmas 3, 4 and 5) we also empirically measure (i) the computational costs of the former (including that of maintaining maps on local nodes) and (ii) the communication costs of the latter.

Evaluation Data. Public LIDAR datasets are usually gathered by a single source. Therefore, we only use them to study how well the ellipsoids represent local objects. To that end, we use 30 randomly chosen point clouds from the KITTI dataset [5], collected by a Velodyne laser scanner in urban driving (Fig. 3).

Fig. 3. KITTI-dataset scene.

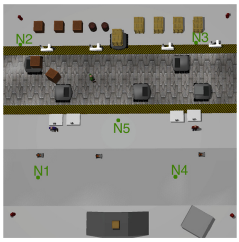


Fig. 4. Factory scene.

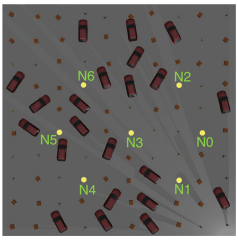


Fig. 5. Random scene.

We also use datasets generated by the Webots simulator (https://www.cyberbotics.com/overview), which simulates real-world LIDARs (VelodyneHDL-32e, in our case) and 3D scenes. One such scene resembles a factory environment (with Automated Guided Vehicles, lifting arm cranes and related objects) with four LIDARs placed at the corners of the scene and one in the middle (Fig. 4). Other scenes define random objects, as small as cubic boxes (with lengths of 80 cm) to objects as big as cars, over an area of $50 \times 50m^2$ with LIDARs placed at up to seven spots. Each object is randomly rotated around its vertical axes to vary the angle with which it is exposed to LIDARs (e.g. Fig. 5). To study MAD-C's operational costs, which depend on the number of scene's object and LIDARs (Lemmas 4 and 5), random scenes have a variable number of objects. We define 10 scenes for 10, 50 and 100 objects, for a total of 30 scenes. We use the notation Λ_i for any scene to specify it contains i LIDAR nodes. We exclude the point cloud portions falling outside the scenes' area.

Evaluation Setup. We implemented MAD-C in C++ and used GNU scientific library and Eigen for matrix algebra. For the baseline and local clusterings, we employed Euclidean clustering (cf. Sect. 2) algorithm in Point Cloud Library [14], with ϵ and $minPts$ respectively set to 0.35 and 10. With these values, the baseline reasonably detects all objects in the scenes and provides a reliable ground-truth. All experiments were run on an Ubuntu 14.04 virtual machine with one 3.1 GHz core and 4 GB of memory. We assume a *star topology*, i.e. $K-1$ nodes communicating with a *sink*. Execution of fog/edge devices was emulated by individually running them on the virtual machine and profiling the intermediate results (i.e. local maps) and the performance measurements. The MERGE was performed afterwards. Corollary 3 suggests why this approximations hold.

We estimate *running times* by dividing the *rdtsc* [10] count, the number of *CPU cycles*, by the CPU frequency clock rate. To approximate the *communication times*, we divide the communication volume (sum of local point clouds' volumes for the baseline and sum of the maps' volumes in MAD-C) by the available bandwidth. Despite the latter being a coarse-grained approximation that favours the baseline (since the latter transfers about two orders of magnitude more data, which causes even higher communication overheads and possibly

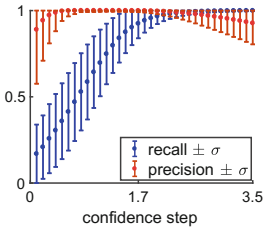


Fig. 6. Performance of the bounding ellipsoid (KITTI-dataset scenes)

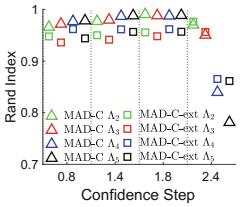


Fig. 7. Accuracy of MAD-C and MAD-C-ext (factory scene)

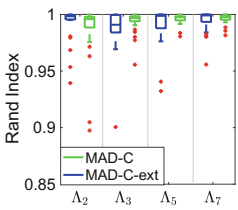


Fig. 8. Accuracy of MAD-C and MAD-C-ext (random scenes)

retransmissions, especially in multi-hop networks), we show that MAD-C still has better performance. We also count the *ellipsoid comparisons* in Algorithm 2 to see how effective the delimiting-box method is.

Estimating the Confidence Step α: Large α (i.e., large bounding ellipsoids) leads to high coverage of local objects. Yet, excessively large α, can lead to ellipsoids erroneously covering other objects' points. To study the trade-off, we employ *precision* and *recall*. For a local object and its bounding ellipsoid, they measure the ratio of the correctly covered points to all covered points and the ratio of the correctly covered points to the size of the point-set of the object, as detected by the baseline, respectively. As shown in Fig. 6 for the KITTI dataset (similar is the behaviour for the Webots simulations), when α is too small, local objects are partly covered (i.e. low average recall) or not covered at all (i.e. low average precision). This is not the case for higher values of α, until the precision decreases again when the bounding ellipsoids erroneously start overlapping other objects. We take $[0.8, 2.4]$ as the desirable range for α.

Accuracy of MAD-C and MAD-C-ext. As noted in Lemma 1, objects identified by the baseline contain one or more local objects from different views. Objects returned by the sink node in MAD-C, likewise, are composed of ellipsoids which in turn relate to local objects. Therefore, we take local objects as the basic elements on which MAD-C and the Euclidean clustering algorithm are executed and compare them using the *RandIndex* measure (cf. Sect. 2). Figure 7 presents the accuracy of MAD-C and MAD-C-ext, respectively, for two, three, four, and five nodes with α values 0.8, 1.4, 1.8, and 2.4 for the factory scene. Figure 8 shows their accuracy for $\alpha = 1.5$ for the random scenes; we use box plots to present accuracy for all the 30 scenes. As shown, both MAD-C's and MAD-C-ext's clustering outcomes are close to the baseline ones. In the remainder, the experimental study of processing and communication costs assumes $\alpha = 1.5$.

Execution cost of MAD-C. Figure 9 (left) shows MAD-C's and baseline's execution costs (Corollary 3) for the random scenes while Fig. 9 (middle) distinguishes MAD-C's costs for local clustering - $\mathbf{C}_1$ - and for the MERGE operation (including the calculations of the bounding ellipsoids) - $\mathbf{C}_2$. Notice

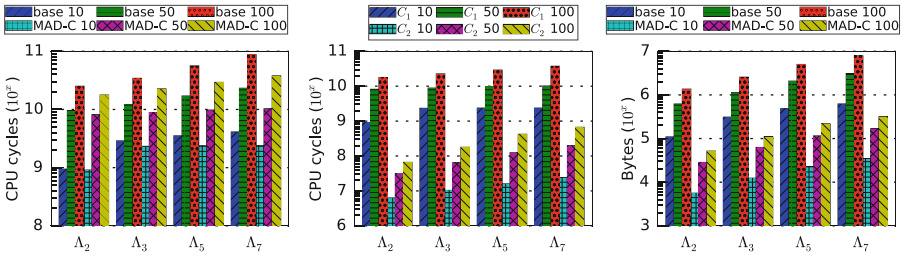


Fig. 9. MAD-C and baseline - avg. execution cost, MAD-C's execution costs decomposition and MAD-C vs baseline - avg. communication cost (random scenes).

Table 1. Average number of ellipsoid comparisons with/without the delimiting-box method.

	Λ_2		Λ_3		Λ_5		Λ_7	
10 obj.	16	168	46	599	92	1827	164	2951
50 obj.	72	3610	218	12738	487	38858	804	56477
100 obj.	105	12618	390	42481	1094	140380	2049	203423

Table 2. Execution times in seconds (100 objects).

	Λ_2	Λ_3	Λ_5	Λ_7
baseline	14	19	32	50
MAD-C	9	11	15	19

the logarithmic-scale y-axis, showing order(s) of magnitude difference between MAD-C and the baseline. Table 1 quantifies the effectiveness of the delimiting-box heuristic (Lemma 6), showing the average number of comparisons with (highlighting) and without the heuristic.

Communication Cost of MAD-C. Figure 9 (right) contrasts the required average volume of communication for both MAD-C (see Lemma 5) and the baseline for the random scenes. MAD-C improves by two orders of magnitude the average communication cost compared to that of the baseline.

Summary. In Table 2 we estimate the total execution time for 100 objects of MAD-C versus the baseline, assuming CPU frequency of 2 Ghz and communication bandwidth of 10 Mbps (similar to specification of devices in edge and fog computing). As observed, MAD-C offers a considerable improvement over the state-of-the-art, with a gap increasing accordingly to the number of LIDARs.

6 Related Work

Relevant clustering-based object detection algorithms for point clouds found in the literature are [4,13]. To cope with point clouds' large data volumes, parallel analysis techniques are given in [9,11]. All these can be leveraged by MAD-C since, as discussed, it integrates on top of *any* clustering algorithm. Variants of Octrees [3], voxel grids [13], and bounding boxes [6] are efficient tools for processing point clouds. MAD-C offers new opportunities due to the compact

representation of bounding ellipsoids and their properties. ICP [2] performs geometric alignment of point clouds when the relative location and pose of sources is unknown, yet, in our work, we know this information.

7 Conclusions and Future Work

MAD-C is a multi-stage method to distributedly approximate detection and localization of objects with multiple LIDARs. Its core phase clusters disjoint subsets of data in a distributed and parallel fashion. Through summarization, it drastically reduces the volume of transmitted data while approximating efficiently the outcomes obtained by clustering all the raw data as a whole. The summaries, computable in a continuous way and with constant time and space overhead, can be combined in an order-insensitive concurrent fashion, allowing for more general-purpose uses of MAD-C. Future work will focus on the deployment of a MAD-C prototype on an IoT test-bed.

Acknowledgements. Work supported by SSF grant "FiC: Future Factories in the Cloud" (GMT14-0032) and VR grants "HARE: Self-deploying and Adaptive Data Streaming Analytics in Fog Architectures" (2016-03800) and "Models and Techniques for Energy-Efficient Concurrent Data Access Designs" (2016-05360).

References

1. Alfano, S., Greer, M.L.: Determining if two solid ellipsoids intersect. J. Guid. Control Dyn. **26**(1), 106–110 (2003)
2. Besl, P.J., McKay, N.D.: Method for registration of 3-D shapes. In: Sensor Fusion IV: Control Paradigms and Data Structures, vol. 1611, pp. 586–607. International Society for Optics and Photonics (1992)
3. Elseberg, J., Borrmann, D., Nüchter, A.: One billion points in the cloud-an octree for efficient processing of 3D laser scans. ISPRS J. Photogramm. Remote. Sens. **76**, 76–88 (2013)
4. Ester, M., Kriegel, H.P., Sander, J., Xu, X., et al.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: KDD, vol. 96, pp. 226–231 (1996)
5. Geiger, A., Lenz, P., Stiller, C., Urtasun, R.: Vision meets robotics: the KITTI dataset. Int. J. Robot. Res. (IJRR) **32**(11), 1231–1237 (2013)
6. Geiger, A., Lenz, P., Urtasun, R.: Are we ready for autonomous driving? The KITTI vision benchmark suite. In: CVPR, pp. 3354–3361. IEEE (2012)
7. Hansen, N.: The CMA evolution strategy: a comparing review. In: Lozano, J.A., Larrañaga, P., Inza, I., Bengoetxea, E. (eds.) Towards a New Evolutionary Computation. STUDFUZZ, vol. 192, pp. 75–102. Springer, Heidelberg (2006). https://doi.org/10.1007/3-540-32494-1_4
8. Himmelsbach, M., Hundelshausen, F.V., Wuensche, H.J.: Fast segmentation of 3D point clouds for ground vehicles. In: Intelligent Vehicles Symposium, pp. 560–565. IEEE (2010)
9. Kumari, S., Goyal, P., Sood, A., Kumar, D., Balasubramaniam, S., Goyal, N.: Exact, fast and scalable parallel DBSCAN for commodity platforms. In: 18th International Conference on Distributed Computing and Networking, p. 14. ACM (2017)

10. Paoloni, G.: How to benchmark code execution times on Intel IA-32 and IA-64 instruction set architectures. Intel Corporation, p. 123 (2010)
11. Patwary, M.A., Palsetia, D., Agrawal, A., Liao, W.k., Manne, F., Choudhary, A.: A new scalable parallel DBSCAN algorithm using the disjoint-set data structure. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, p. 62. IEEE Computer Society Press (2012)
12. Preguica, N., Marques, J.M., Shapiro, M., Letia, M.: A commutative replicated data type for cooperative editing. In: 29th IEEE International Conference on Distributed Computing Systems, ICDCS 2009, pp. 395–403. IEEE (2009)
13. Rusu, R.B.: Semantic 3D object maps for everyday manipulation in human living environments. KI-Künstliche Intelligenz **24**(4), 345–348 (2010)
14. Rusu, R.B., Cousins, S.: 3D is here: point cloud library (PCL). In: IEEE International Conference on Robotics and automation (ICRA), pp. 1–4. IEEE (2011)
15. Wagner, S., Wagner, D.: Comparing clusterings: an overview. Universität Karlsruhe, Fakultät für Informatik Karlsruhe (2007)