Shukla *et al. J Big Data (2018) 5:24*
https://doi.org/10.1186/s40537-018-0134-7

Journal of Big Data

RESEARCH Open Access

DIMPL: a distributed in-memory drone flight path builder system

Manu Shukla[1*], Zhiqian Chen[2] and Chang-Tien Lu[2]

*Correspondence:
mashukla@vt.edu
[1] Omniscience Corporation, Palo Alto, CA, USA
Full list of author information is available at the end of the article

Abstract

Drones are increasingly being used to perform risky and labor intensive aerial tasks cheaply and safely. To ensure operating costs are low and flights autonomous, their flight plans must be pre-built. In existing techniques drone flight paths are not automatically pre-calculated based on drone capabilities and terrain information. Instead, they focus on adaptive shortest paths, manually determined paths, navigation through camera, images and/or GPS for guidance and genetic or geometric algorithms to guide the drone during flight, all of which makes flight navigation complex and risky. In this paper we present details of an automated flight plan builder *DIMPL* that pre-builds flight plans for drones tasked with surveying a large area to take photographs of electric poles to identify ones with hazardous vegetation overgrowth. The flight plans are built for subregions allowing the drones to navigate autonomously. DIMPL employs a distributed in-memory paradigm to process subregions in parallel and build flight paths in a highly efficient manner. Experiments performed with network and elevation datasets validated the efficiency of DIMPL in building optimal flight plans for a fleet of different types of drones and demonstrated the tremendous performance improvements possible using the distributed in-memory paradigm.

Keywords: Distributed system, In-memory distribution, Autonomous drones, Flight path of drones

Introduction

As use of drones rapidly expands, it is aided by improvements in technology such as high speed cameras, sensors, and processors able to analyze the data rapidly and efficiently on the drone. Better scalability in processing image, terrain, weather and surface data and using it to aid in navigation has also allowed them to become increasingly autonomous during flight. Drones have proven very useful in both military battlefield and civilian tasks. Along with common civilian tasks in education [30], studying natural phenomena [45], reconnaissance [36] and conservation [28] they have been used increasingly in surveying farms [41], forests [10] and borders [27]. Drones can either be controlled manually by an operator or fly autonomously, but as the use for an operator increases the cost considerably, it is clearly preferable for them to operate autonomously. Autonomous flight presents challenges in terrain navigation, however, as a multitude of flight path scenarios such as variations in altitude and the density of objects to be surveyed must be taken into account. Flight planning

must also account for drone hardware limitations. Adding complexity to the fight path building process also decreases the performance of the algorithms used to pre-build them. The speed at which these flights can be planned and re-planned with changes in weather and terrain data and updates in drone battery and hardware becomes crucial.

The complexity involved in covering an area with automated flights increases when there are multiple types of drones available with different capabilities. There are two main types: conventional drones that perform conventional take offs and landings and quadcopters with vertical take offs and landings. Along with the challenge encountered in building flight paths for drones optimizing the use of these different types of drones to cover the entire region, the flight path construction scaling needs to stay efficient with additional constraints on types of drones. Each drone type suffers from specific limitations and the cost of operating the different types of drones can also vary considerably. These limitations and costs also change significantly over a period of time with improvements in battery and drone hardware. The optimization problem then becomes multi-pronged: not only is it necessary to cover the entire region with multiple flights of a drone type that can navigate varying terrain, but the drones that are cheaper to operate should be used as frequently as possible to minimize cost and the entire process should be rapidly repeatable with updates in batteries and drone hardware, adding terrain to task and changes in weather conditions. The distributed processing of larger overall areas is generally split into smaller sections or subregions. Doing this interactively with in-memory based distribution reduces the operator time. This reasoning motivated us to create DIMPL. Covering very large areas using multiple drones, each with its own specific flight path, creates a number of issues, however: *Challenge 1: Process terrain and network data in memory*. Terrain data includes geo-coordinates, elevation, humidity and more and needs to be processed on a cluster. Processing them in memory and thus minimizing disk accesses is crucial for good performance. *Challenge 2: Distributed computation of flight paths.* Efficient computations of flight plans in a short amount of time allows for rapid turn-around times after adjustments have been made. The efficient computing of flight paths also allows them to be fed into each drone manually and adjusted if the scope of the work changes. *Challenge 3: Optimize for different types of drones.* Optimize the constraints for different types of drones. Focus on the most efficient use of the drone fleet based on operating costs, the limitations of each drone type and other constraints.

DIMPL accounts for terrain factors such as power lines length and elevation to optimally divide an area into subregions and then build a flight plan for each subregion almost interactively in memory. A set of *Flight Plans* are generated for surveying the entire power line network for an aviation organization. The goal is to minimize the number of drone flights and optimize the distribution of processing terrain and network line data and hence the overall cost. This is achieved by optimizing the coverage of each flight across a subregion and assigning the most appropriate type of drone for that subregion. Knowledge regarding network lines and elevation of waypoints within each subregion are needed to satisfy the rules that govern the *autonomy* and *climbing angle* constraints. These constraints determine whether the subregion needs to be shrunk or expanded or split between multiple drone types. Our new technique uses

a section-wise or subregion based distribution of network line processing, linear inequalities, and spatial indexes to query the elevation around waypoints. This powerful approach can automate flight path building utilizing only terrain data and pre-known drone hardware limitations. This project designed algorithms to support drone flights that will take pictures of vegetation along electricity pole networks and was carried out for a drone company under contract from a large utility company in Europe. Image analysis during post-processing determines whether the vegetation has overgrown the poles and needs trimming. The resulting application can be used to automate flights for many such tasks however, including determining the extent of flood damage, deforestation, pollution and agricultural activities. As the volume of terrain and network line data for large areas increases rapidly, in order to scale to larger terrain and network line datasets, DIMPL uses an in-memory distributed paradigm. With increasing amounts of data, scaling can be extended indefinitely by simply adding nodes to the cluster where data is stored and computations are performed. The contributions of this paper are:

- *Develop a framework to dynamically create and resize subregions for single drone flights* The proposed techniques dynamically divide a large area into subregions that can be covered by a single drone flight and collectively resize and adjust them for optimal coverage. Processing terrain data by subregion provides flexibility in deciding which type of drone to assign to a particular subregion.
- *Model different terrain scenarios taking into account multiple drone types and their hardware limits* This work combines terrain and network data with drone flight constraints for multiple drone types, applied as linear inequalities. It maximally exploits the capability of each drone type based on the subregion size and the type of drone needed to cover it.
- *Distributed flight plan creation* Flight plan construction is distributed in-memory or on disk using standard frameworks to avoid any limitations due to the size of the spatial index on a single node by applying novel key-value pair based joins. It also scales horizontally to larger terrains datasets.
- *Design algorithms for optimized levels of distribution* Explore multiple levels of distribution by dividing the entire region into subsections or subregions and processing them in parallel. Section-wise and subregion-wise distribution each have their tradeoffs and require different reconciliation steps.
- *Conduct extensive experiments on real world data and scenarios* The proposed techniques are validated using experiments on two datasets, one real world and other simulated, to test the scaling effectiveness of the distributed techniques. The flight plans generated were used to solve an actual real world problem.

The rest of the paper is organized as follows. "Related work" section explores research related to this work and "Preliminaries" section presents the preliminary design considerations for the flight plan builder. In "Distribution techniques" section, we present an overview of the disk based distribution framework utilized in DIFPL implementation and the in-memory

implementation DIMPL, followed by a description of the experiments conducted in "Experiments" section. Performance results are discussed in "Results" section and conclusions and possible future applications of the algorithms are presented in "Conclusions" section.

Related work

Earlier research in the area can be split into three categories, namely approaches designed to automate the flight of drones, routing problems and distributed platforms for general spatial data processing that include processing specific to drones.

Automated flights Automated drone flights in natural and man made environments are becoming ubiquitous [22]. Automated flights in university environments have been explored [7], as has automation of flights in man made environments including those utilizing images in indoor contained environments [9] and the combination of image, sonar and odometry inputs [39].

Several ways to automate the flight of drones have been proposed, including using sensors [42], camera images [9], inputting waypoints as a file [5] or automating the controls [33]. Image based autonomous flights have been described [18] and the use of a platform for the verification of image based navigation explored [35]. Landmark based visual navigation for use in cases of GPS failure has also been investigated [6], along with using network connections to control fleet of drones [46]. Reactive route selection from a bank of optimal trajectories based on changes in operating environments have been proposed [25], and an A* algorithm to plan shortest flight paths for drones adaptive to different flight conditions [37].

Pre-planning the paths of drones has been the focus of a lot of research. Genetic algorithms have been used to trace flight paths [15], and the use of ant colony algorithms for 3D route planning has also been considered [17]. UAV path planning using Particle Swarm Optimization and digital pheromones incorporating vehicle mechanics to generate multiple 3-d paths for operators have been proposed [24], as have optimization algorithms for multi-objective drone route planning [32]. Fast graph search algorithms that satisfy constraints on flights have been used to determine the optimum path of a drone traveling between two given locations [13] and UAV flight path planning in time-space varying wind fields using a kinematic tree path planner have been developed [12]. None of these techniques have sought to pre-build the flight paths of drones and then allow them to navigate along the path.

Vehicle routing Vehicle routing problem (VRP) has been studied extensively [40]. VRP is NP hard and is a variant of traveling salesman problem (TSP). It entails delivering goods from a central depot to customers who have ordered them efficiently. Deploying mix fleet to reduce energy consumption for distance or time optimizations is explored [29]. Vehicle routing for drones for delivery by experimentally validating energy consumption model is proposed [19] and by examining worst case scenarios has been explored [44]. The problem of distributed vehicles needing to compute their routes in decentralized manner using duality theorem of linear programming has been studied [31]. Scenario based parallelization integrating Monte Carlo simulation inside a heuristic-randomization process for real time VRP is described [26].

Distributed spatial operations The increased availability of spatial data distributed approaches has led to a surge in their popularity. Spatial data processing techniques

using MapReduce have been implemented [11], along with accelerated processing with MapReduce [43]. Distributed spatial operations with Hadoop and the use of SpatialHadoop as a Hadoop extension for spatial operations have both been explored [21] and computational geometry algorithms have been distributed using SpatialHadoop [20]. Approaches based on distribution of drone data analysis and multiple drone flight coordination tasks has begun to gather momentum [14] and Hadoop based platforms that support spatial queries with MapReduce have been developed [1], as well as in-memory spatial operation framework based on Apache Spark [47]. As yet however, none of these techniques are capable of building efficient paths that will enable fleets of drones to cover a large area while taking into account multiple terrain and task datasets.

As far as we can determine, previous researchers have not considered ways to combine multiple drone types and variations in terrain simultaneously when automating flight paths. VRP solutions have proposed optimization of deliveries for a mix fleet, some even in a distributed paradigm but they are not suited for coordinated survey of an area in a manner that is easy to distribute computationally. None of the techniques proposed so far has the capacity to build coordinated flight paths for fleet of drones that will enable them to perform a complex survey task in a distributed co-ordinated performant manner. DIMPL, like an earlier prototype DIFPL [38], does not rely on images or video to navigate. Instead flight paths are built offline using terrain and network line data and do not need to be adjusted dynamically as all the constraints are applied when the program is run. Our distributed technique performs all operations in memory. Unlike DIFPL, which makes 2 passes through the data using standard Hadoop constructs and avoids building a large spatial index on a single node by splitting the data into sections or subregions that can be processed independently, the new model builds all data structures in its memory in a distributed fashion. This eliminates disk I/O from one stage to the next, further improving performance.

Preliminaries

In this section we describe the fundamental principles of the new DIMPL algorithms. "Data" section describes the input and output data; "Constraints" section explores the constraints governing the hardware, subregions and waypoint construction. "Problem formulation" section formulates the problem after which "Flight path builder" section details the base flight plan builder that serves as the basis for both the DIMPL and DIFPL distribution techniques.

Data

DIMPL utilizes (x, y) geo-coordinate positions of the network line endpoints provided by the commissioning aviation organization and elevation data (x, y, elevation) provided by a geographic agency. The elevation points are spaced evenly 25 m apart. The output of the program consists of a set of flight plans, each composed of a set of waypoints (x, y, altitude) and one landing point in KML format, which are fed directly into the drone. An output file containing the flight plan is written for each subregion. The provided 7800 km^2 power line network is shown in Fig. 1. The elevation point data provided for the entire European country and is filtered for the region for which network lines data is available as part of the pre-processing.

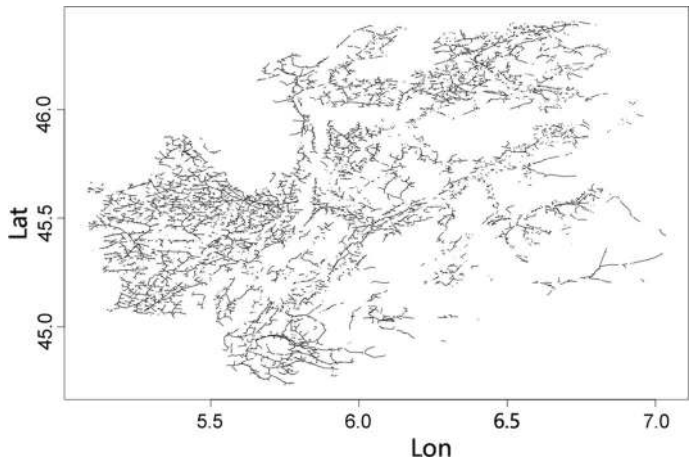

Fig. 1 Entire line network in the study area

Constraints

The drone can be one of several types. The characteristics for each type of drone are features such as its speed, autonomy, turning radius, max slope, and flight height. To allow handling of multiple drone types, these characteristics are kept configurable and the drone company can adjust them easily through the configuration files. Every pole in the network must be photographed at least 4 times. The drone performs 2 passes from each side of the lines, with the first pass in one direction and the return pass in the other. Each drone is equipped with a NEX7 24 Mega pixel camera with a 50 mm optical lens. The camera takes one image every second. The images of each pole from each side and multiple angles are used to perform a 3D image reconstruction of each pole in order to determine whether vegetation has overgrown that pole. Every attempt is made to maximize the use of conventional drones as they are both cheaper and more plentiful. The primary limitations of the drone hardware are:

- *Number of waypoints* The hardware in conventional drones can be programmed with up to 200 waypoints while a quadcopter can only be programmed with 50 waypoints.
- *Climbing angle* The maximum slope of ascent for conventional drone is 12°; for descent this is −16°. For a quadcopter the maximum slope for ascent is 90° and for descent is −90°.
- *Autonomy* The maximum distance a conventional drone can fly in a single flight is 30 km and for a quadcopter it is 3 km.

These constraints on the flight path of the different drone types are modeled as inequalities. The inequalities are applied to each subregion for the various types of drones and defined as follows.

For climbing angle:

$$Max(c_p) \leq C_{type}$$

where c_p is the angle the drone has to climb to fly from one waypoint to the next along the network line and is calculated based on the recommended drone flying altitude and the elevation at the waypoints. The recommended altitude for conventional drones is 100 m and for quadcopters is 50 m. The maximum weight of the drone can be 2200 g. For the waypoints along the sides of the network lines, the elevation is calculated by querying the k nearest neighbor elevation points with a kNN spatial index query and taking their average, thus ensuring it satisfies the climbing angle constraint. The k in KNN query was set to 4 and the option was provided to take the maximum elevation of kNN results instead of average for irregular terrains.

For autonomy:

$$\sum_{l}(2 * d_l + i_l) + 3 * l * 2 * \pi * r + t + n + \epsilon_{type} \leq A_{type}$$

where d is the distance along each network line, t is the takeoff distance to reach the required elevation over the first network pole for the climbing angle of each drone type, n is the landing distance for the descent angle for that drone type, i is the distance between two network lines, r is the turn distance for the drone type for l lines and ϵ_{type} is the distance added to buffer against unforeseen conditions encountered by the drone. The turning radius of a conventional drone is 150 m, while that of a quadcopter is 0 m. The distance i is calculated by ordering network lines in the subregion based on their x_{start} and then calculating the distance between one line and the next. Since there are 3 turns needed for a drone to cover a line segment twice, after which it proceeds to the next line segment, 3 turning circumferences have to be added to this equation. Every network line must be covered by one of the available drone types. The requirement to photograph each pole 4 times is satisfied by setting the camera to take an image every second. The number of waypoints in the output is determined by collecting waypoints along the network lines at 200 m intervals for conventional drones and 100 m intervals for quadcopters; these spacings can be increased if the number of waypoints exceeds the maximum allowed by the drone. The drone can be placed anywhere in the area they are required to survey, hence operators were recommended to keep them close to the first waypoint, giving conventional drone enough distance to climb to the elevation of the first waypoint. The drones are programmed to land after the last waypoint and the drone operator is expected to pick them up.

Problem formulation

We now formulate the problem the system is solving with in-memory distribution. We begin by formulating the problem that fleet of drones is solving and then proceed to the distribution problem.

(A) *Drone survey problem*

Cover a region $\mathcal{R}$ *containing* $\mathcal{P}$ *electric poles with drones* $\mathcal{Q} \oplus \mathcal{C} \in \mathcal{D}$
where $\mathcal{Q}$ *is the set of quadcopters and* $\mathcal{C}$ *is the set of conventional drones such that*:

(i) $\forall P_i \in \mathcal{P}$ *photograph with photos* $H_i \in \mathcal{H}$ *from all desired angles*
(ii) $\forall D_i \in \mathcal{D}$ *follow autonomy and the number of waypoints constraint*
(iii) $\forall D_i \in \mathcal{D}$ *the drone type satisfies the climbing angle constraint*
(iv) *Minimize* $||Q||$
(v) *Cover entire region* $\mathcal{R}$ *by* $\mathcal{D}$

(B) *Distribution problem*

Distribute processing of parts of region $\mathcal{R}$ *with drones* $\mathcal{D}$ *such that*:

(i) *Reconcile all* R_i *with no overlaps or uncovered spaces*
(ii) *Minimize* $||\mathcal{R}||$
(iii) *Meet requirements for overall region* $\mathcal{R}$ *in* (A)

Flight path builder

The algorithms utilized in DIFPL for subregion-wise and section-wise distributions are implemented in DIMPL for the in-memory paradigm. They include a base flight plans builder algorithm that performs queries and applies constraints on the results within a section or subregion. Two levels of distribution are built, one based on the flight path algorithm and another parallelized on subregions.

Flight path building within a section Algorithm 1 shows the process utilized to build the flight paths. The sections are illustrated in Figs. 2 and 3. The flight path builder

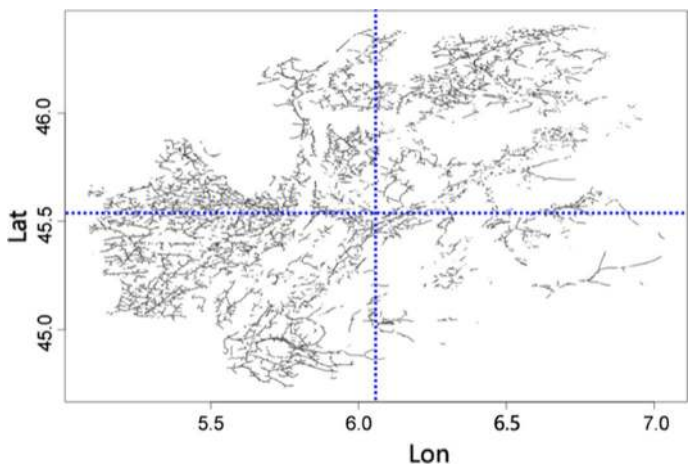

Fig. 2 Overall region divided into 4 subsections

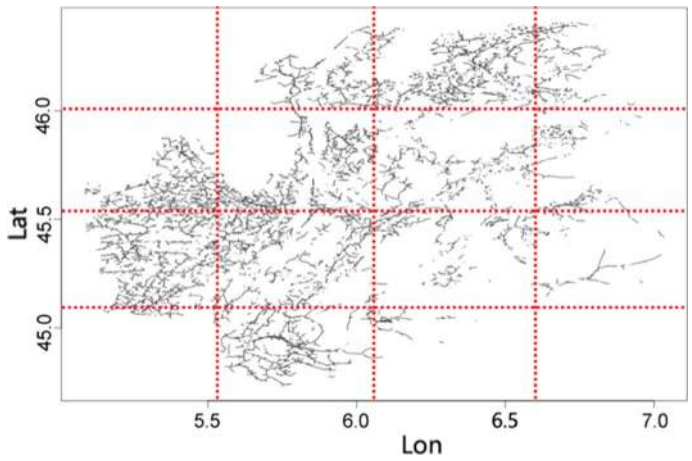

Fig. 3 Overall region divided into 16 subsections

algorithm accepts as input network and elevation line data from disk files in its first step. It then checks for existence of a spatial index on disk. If not present, it proceeds to create and insert the network lines and elevation data into the index Si else simply loads the index from disk into memory. It then proceeds to query the spatial index for network lines and elevation data by subregion starting from the bottom left of the minimum bounding rectangular area containing the region. The default conventional drone sized rectangles are denoted as D_c and the range queries against the spatial index for D_c sized subregions are q_i. The spatial index Si is an R* tree. The flight path algorithm then computes the waypoints along the network lines returned in q_i response with elevation along the waypoints computed by averaging elevation of neighboring points obtained with kNN query against spatial index.

If the network lines satisfy the climbing angle constraint C_c and autonomy constraint A_c, waypoints are built along network lines for conventional drone in second step. If it fails the autonomy constraint the region is shrunk or expanded by γ sized slices until the autonomy constraint is satisfied. The default for γ is 10% of D_c. If the network lines and waypoints fail the constraint C_c, then the spatial index is queried for the network lines and elevation points in quadcopter sized subregions D_q with queries qq_i from left to right in the conventional drone subregion. Each consecutive quadcopter size subregion that satisfies the climbing angle constraint is merged with the previous one. Those that that do not satisfy C_c are deemed to require the quadcopter. The default queries go left to right and top to bottom until they reach the top right of the overall rectangular region. The output of each finalized subregion O_i is written to disk as flight paths consisting of waypoints and landing point.

Algorithm 1 Flight Plans Builder

Input: $\{network_i\}$,$\{elevation_i\}$ {network lines and elevation data}
Output: $\{subregion_i, waypoint_i\}$ {each subregion and waypoints}
{**step 0: setup index and subregion iterations**}
if index spatial index file does not exist **then**
  Si $\leftarrow$ $\{network_i, elevation_j\}$ {create spatial index with network lines as 2D objects and elevation points}
else
  read Si from disk {read in spatial index from disk}
end if
$\forall$ query q_i for subregions $subregion_i$ starting from bottom left of region with size D_c
while !at top right of region **do**
  regionDone=false
  $r_i \leftarrow$ Si(q_i) {query Spatial index Si with query q_i to generate result set}
  generate $waypoint_i$ in r_i
  for all $waypoint_i$ in r_i **do**
    $elevation_{wi} \leftarrow \frac{\sum kNN_{waypointi}}{k}$
  end for
  if max C(r_i) $\leq C_c$ **then**
    {**step 1: Subregion can be covered with conventional drone**}
    {retrieved objects elevations satisfy conventional drone climbing angle constraint}
    while $\sum\{|\ l_r\ |\}$ does not satisfy A_c and subregion right query window within overall region right boundary and right of subregion left boundary **do**
      {retrieved objects network lines do not satisfy conventional drone autonomy constraints}
      Si $\leftarrow q_i \pm \gamma$
      {reduce or expand window size and re-query}
    end while
    if $\sum\{|\ l_r\ |\}$ satisfies A_c **then**
      {conventional drone constraints satisfied}
      $\{waypoint_i\} \leftarrow C_q$ {create waypoints for conventional drone subregion}
      write output O_i
      $q_i \rightarrow q_j$ {move to next window}
    end if
  else
    {**step 2: Subregion needs quadcopter**}
    for all $qq_i \in q_i$ **do**
      $r_i \leftarrow$ Si(qq_i)
      {requery with default quadcopter subregion sizes}
      if !maxC(r_i) $\leq C_c$ **then**
        {region fails conventional drone climbing angle constraints}
        $\{waypoint_i\} \leftarrow Q_q$
        {build waypoints for quadcopter}
      else
        {merge waypoints into previous conventional subregion if possible or start new conventional subregion}
        $O_i \cup q_{i-1}$
      end if
      Write Output O_i as {waypoints} of subregion if complete
      $qq_i \rightarrow qq_j$
      {move to next window}
    end for
  end if
end while

Distribution techniques

This section provides details of the DIMPL algorithms and architecture. "Distributed system *DIFPL*" section describes the algorithms that are first implemented in DIFPL using on-disk distribution that are now implemented in the in-memory paradigm in

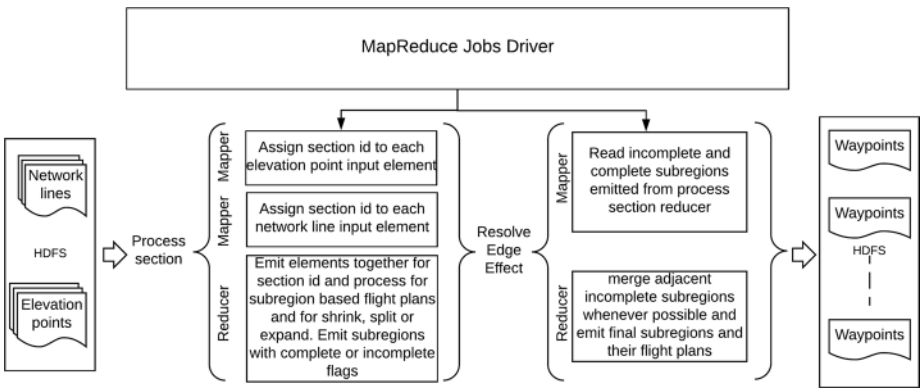

Fig. 4 Distributed disk based system architecture

DIMPL. "In-memory distribution in DIMPL" section goes on to provide a description of the in-memory distribution framework used and "DIMPL algorithms" section details the section-wise and subregion-wise in-memory distribution algorithms.

Distributed system *DIFPL*

This section describes the distributed system DIFPL that builds flight plans based on a distributed paradigm. It provides an overview of the architecture of the system in "Architecture" section, the algorithms used in "DIFPL algorithms" section and the optimized distribution for maximum parallelization in "DIFPL algorithms" section.

Architecture

The architecture for DIFPL is based on a distributed paradigm. The distribution approach in DIFPL was implemented using the Apache Hadoop MapReduce framework [16]. MapReduce programming model allows users to develop scalable, fault tolerant applications. It provides a key-value pair based paradigm that applications can utilize to run in a parallelized manner on a cluster in a shared nothing environment. A master node in the cluster orchestrates data storage and computation distribution over slave nodes. Each MapReduce job consists of three phases, Map, Shuffle and Reduce. The input data is stored in a distributed file system HDFS. The three phases then transform as follows:

1. *Map* This stage reads the input as key-value pairs $< k_1, v_1 >$ and transforms them into another set of key-value pairs $< k_2, v_2 >$ that are handed to the shuffle on each node.
2. *Shuffle* The key-value pairs $< k_2, v_2 >$ are taken from the map phase and distributed across all machines. This stage guarantees sorting on keys and all values for a key combined together before it hands them over to the reduce stage.
3. *Reduce* It receives each key and all its values grouped together as $< k_2, < v_2, v_2', v_2'', \ldots >>$ from all the mappers across machines and allows user to emit another set of key-value pairs $< k_3, v_3 >$ to be processed in the next job.

The user can implement application specific operations in the map and reduce stages.

The inability of limited memory on a single node to index increasing elevation and network lines datasets as survey area increases necessitates distribution of flight path builder. Several options for the distribution of the flight plan builder process are available. The identification of quadcopter subregions in one part of survey area and the shrinking and expansion of quadcopter and conventional drone subregions in other parts can be performed in parallel. An overview of the distributed system architecture is shown in Fig. 4. The network lines and elevation data is read in from a distributed file system and a series of MapReduce jobs are run on it. The first set of jobs assign elevation and network line data by section id to the same reducer in Mappers. The reducer combines all data by section id and builds subregions within the section and emits flight plants for complete and incomplete subregions. The next job resolves subregions at the edge of sections by merging whenever possible and emits updated flight plans for those subregions while emitting the rest unchanged. Final set of flight plans are written back to the distributed file system. A jobs driver orchestrates the execution of MapReduce in sequence. The distributed application runs as a cluster on Amazon Web Services (AWS). MapReduce jobs are run on AWS Elastic MapReduce (EMR) and data is read from and written to S3 buckets similar to HDFS.

DIFPL algorithms

These algorithms utilize the flight path algorithm and implement the distribution algorithms utilizing it in MapReduce framework.

Sectionwise distribution The first task in this distribution is to split the entire area into sections and query for subregions in a spatial index containing elevation and network lines for each section. This allows for indexing smaller sections in the spatial index instead of the entire area.

The distribution is broken down into 2 phases, each phase making a pass over the data. Each pass incrementally identifies subregions for a quadcopter or conventional drone, with the second and final pass resolving boundary issues and outputting the final flight paths for each subregion.

Assign and process sections The first pass labels network and elevation data with the section they belong to. The details are described in Algorithm 2. The data is processed by two mappers. The first mapper reads network lines data $\{network_i\}$ as text and calculates the section id based on the coordinates of the network lines. The key value pairs emitted from the mapper are $< sectionID_j, networkline_l >$. The second mapper similarly reads elevation data $\{elevation_j\}$ and calculates the section id and emits key-value pairs $< sectionID_j, elevation_i >$. The reducer reads the data and aggregates all elevation and network line observations for the section id and applies the flight plans builder algorithm for the section. It first builds a spatial index for all the elevation points for the subregion based on the network and elevation data in memory. It then identifies subregions as quadcopter and conventional subregions based on the elevation and autonomy constraints, shrinking, expanding and merging subregions as needed. The final key-value pairs emitted in the reducer consist of the details of each subregion within a section, namely the subregion id $subregion_j$, the subregion extent xsr_j,ysr_j,xer_j,yer_j which represents the diagonal coordinates of the subregion starting from bottom left edge xsr_j, ysr_j to top right edge xer_j, yer_j. It also includes a flag indicating if it is a quadcopter or conventional drone subregion, the network lines and elevation points and waypoints $waypoint_k, \ldots, waypoint_l$ for

a quadcopter or conventional drone to follow along the network lines in the subregion. If there are no network lines in the subsection emitted by mapper the reducer simply exits.

Algorithm 2 Assign and Process Sections

Input: $\{elevation_i, networkline_i\}$
Output: $subregion_j \rightarrow \{xsr_j, ysr_j, xer_j, yer_j$, quadcopter|conventional,$\{network_j\}, \{elevation_j\}, \{waypoint_j\}$

mapper1:
calculate $sectionID_j$ based on each elevation data point based on their coordinates
emit:$sectionID_j \rightarrow elevation_j$
mapper2:
calculate $sectionID_j$ for each network data point based on their coordinates
emit:$sectionID_j \rightarrow networkline_j$
reducer:
for all observations **do**
    build $section_j$ that incorporates all the $networkline_j$ and $elevation_j$ points for the section
end for
APPLY Flight Plans Builder Algorithm
emit:$subregion_j \rightarrow \{xsr_j, ysr_j, xer_j, yer_j$, quadcopter|conventional,$\{network_j\}, \{elevation_j\}, \{waypoint_j\}$

Resolve edge effects The second pass resolves edge effects between sections, as described in Algorithm 3. The input to the mapper consists of the subregions $subregion_j$ with their extent, the drone type needed, network lines and elevation points in the subregion and the waypoints. For the subregions located along the vertical edge of the sections associated with their section id, reducer pairs the corresponding left and right subregions. The subregions can thus be merged if they are covered by the same drone type. The output from the reducer consists of the entire set of subregions, including the merged subregions. All non boundary subregions are emitted as is.

Algorithm 3 Resolve Edge Effects

Input: $\{subregion_j \rightarrow \{xsr_j, ysr_j, xer_j, yer_j,$ quadcopter|conventional,$\{waypoint_j\}, \{network_j\}, \{elevation_j\}\}\}$
Output: $subregion_k \rightarrow \{xsr_j, ysr_j, xer_j, yer_j$,quadcopter |conventional, $\{waypoint_k\}\}$

mapper:
emit:"verticalborder|notverticalborder" $\rightarrow subregion_j$, $\{xsr_j, ysr_j, xer_j, yer_j, \{network_j\}, \{elevation_j\}$ $\{waypoint_k\}$, $\}$
reducer:
if border **then**
    if $subregion_i$ && $subregion_j$ are adjacent **then**
        if they can be merged with combined subregion network length $<\beta\%$ **then**
            merge subregions
            emit:$subregion_k \rightarrow \{xsr_k, ysr_k, xer_k, yer_k$, quadcopter|conventional,$\{waypoint_k\}\}$
        else
            emit:$subregion_i \rightarrow \{xsr_i, ysr_i, xer_i, yer_i$, quadcopter|conventional,$\{waypoint_i\}\}$
            emit:$subregion_j \rightarrow \{xsr_j, ysr_j, xer_j, yer_j$, quadcopter|conventional,$\{waypoint_j\}\}$
        end if
    end if
else
    emit:$subregion_j \rightarrow \{xsr_j, ysr_j, xer_j, yer_j$, quadcopter|conventional,$\{waypoint_j\}\}$
end if

Optimized distribution A more scalable approach that avoids the need to build spatial indexes in each reducer in order to build waypoints in a subregion is now discussed. Network lines are input in their raw form and elevations inserted in the spatial index.

Assign and process subregions This algorithm receives as input the network and elevation data as $\{elevation_i\}$, $\{networkline_i\}$ and the mapper emits them with the key $subregion_j$. The resulting mapper emitted pairs $subregion_j$, $networkline_j$ and

$subregion_j$, $elevation_j$ ensure that network lines and elevation points land together in the reducer. In the reducer the waypoints are built along the network lines and the climbing angle constraint C_c is checked by querying the elevation points from subregions near each waypoint from the spatial index using a kNN query. All subregions with waypoints are then emitted, along with a flag indicating whether they are complete or incomplete. The details are shown in Algorithm 4. If the autonomy constraint is not satisfied because the network lines are too long or less than the threshold β% of the drone type autonomy, the subregions are marked as incomplete when they are emitted from the reducer. At this point an attempt is made to merge the adjacent incomplete subregions.

Algorithm 4 Assign and Process Subregions

Input: $\{elevation_i, networkline_i\}$
Output: $subregion_j \rightarrow xsr_j, ysr_j, xer_j, yer_j$, $\{waypoint_j\}$, quadcopter |conventional, complete |incomplete, $\{elevation_j\}$,$\{networkline_j\}$

```
mapper1:
emit:subregion_j → elevation_j
mapper2:
emit:subregion_j → networkline_j
reducer:
index {elevation_i} in spatial index
for all observations calculate waypoints_j based on its network lines do
    for all waypoint_i calculate elevation with kNN query do
        compute climbing angles and check if they satisfy constraint C_c
        if climbing angle fails constraint then
            split subregion into quadcopter subregions
            progressively apply C_c on each subregion of size D_q
            if satisfied then
                if autonomy constraint A_c satisfied then
                    merge with previous if autonomy constraint satisfied
                else
                    mark as quadcopter
                end if
            end if
            emit:subregion_j → xsr_j,ysr_j,xer_j,yer_j, {waypoint_j}, quadcopter |conventional, complete
            |incomplete, {elevation_j},{networkline_j}
        else if autonomy constraint A_c not satisfied then
            shrink subregion_j
            emit:subregion_j → xsr_j,ysr_j,xer_j,yer_j, {waypoint_j}, quadcopter |conventional, complete
            |incomplete, {elevation_j},{networkline_j}
        end if
    end for
end for
```

Reconcile adjacent subregions Subregions in a sparse area that needs to be expanded, or those generated after shrinking a dense subregion are emitted as incomplete. All subregions that are split result in a set of quadcopter and conventional drone subregions. Those that are unable to satisfy the autonomy β% constraint are also emitted by the reducer as incomplete so that they can be merged with adjacent incomplete subregions. The details of this process are shown in Algorithm 5. The algorithm accepts all subregions and the mapper emits those that are incomplete. The reducer then aligns those that are adjacent and checks whether the adjacent subregions can be merged together. Every time a merged subregion satisfies the β% threshold and autonomy constraint, reducer emits it as $subregion_i \rightarrow \{xsr_i, ysr_i, xer_i, yer_i$,$\{waypoint_i\}$,quadcopter|conventional} and then proceeds to the next subregion.

Algorithm 5 Reconcile Adjacent Subregions

Input: $subregion_j \rightarrow xsr_j, ysr_j, xer_j, yer_j$, $\{waypoint_j\}$, quadcopter |conventional, complete |incomplete,$\{elevation_j\}$,$\{networkline_j\}$
Output: $subregion_i$,$xer_i, yer_i, xsr_i, xsr_i$,$\{waypoint_i\}$, conventional|quadcopter
1: *mapper*:
2: emit:"incomplete" $\rightarrow$ $subregion_i$,$xer_i, yer_i, xsr_i, xsr_i$, $\{elevation_i\}$,$\{networkline_i\}$,incomplete,$\{waypoint_i\}$, conventional|quadcopter
3: emit:$subregion_j \rightarrow xer_i, yer_i, xsr_i, xsr_i$, $\{elevation_i\}$,$\{networkline_i\}$,complete,$\{waypoint_i\}$, conventional|quadcopter
4: *reducer*:
5: **for all** incomplete subregions calculate new subregions based on its coordinates **do**
6: **if** $subregion_i$ && $subregion_j$ are adjacent **then**
7: **if** they can be merged with combined subregion network length <β% **then**
8: merge subregions
9: emit:$subregion_k \rightarrow \{xsr_k, ysr_k, xer_k, yer_k$, quadcopter|conventional,$\{waypoint_k\}\}$
10: **else**
11: emit:$subregion_i \rightarrow \{xsr_i, ysr_i, xer_i, yer_i$, quadcopter|conventional,$\{waypoint_i\}\}$
12: emit:$subregion_j \rightarrow \{xsr_j, ysr_j, xer_j, yer_j$, quadcopter|conventional,$\{waypoint_j\}\}$
13: **end if**
14: **end if**
15: **for all** complete subregions **do**
16: emit:$subregion_i \rightarrow \{xsr_i, ysr_i, xer_i, yer_i$, quadcopter|conventional,$\{waypoint_k\}\}$
17: **end for**
18: **end for**

In-memory distribution in DIMPL

This section describes the algorithms used to perform flight path generation using in-memory techniques in DIMPL. The in-memory distribution implementation, unlike the DIFPL implementation, does not need to write the intermediate output to disk, it performs all operations in-memory. Apache Spark is an in-memory based framework that allows computations to be distributed in-memory over a large number of nodes in a cluster [48]. The programming constructs available in Spark transform the data on a disk into RDDs (resilient distributed datasets) and then apply appropriate actions to the RDDs on subsets of data in processes called *executors* on cluster nodes to generate values that can be returned to the application. RDDs provide fault tolerance in case one or more nodes of the cluster fail. The algorithms typically utilized by Spark are ML and statistical functions that are highly iterative in nature. Performing highly distributed operation in a disk based distribution framework such as MapReduce would be expensive computationally due to the need to write data to the disk during each iteration. There are several other distributed in memory frameworks with Apache Flink [3], Message Passing Interface (MPI) [23] and Apache Storm [4] being the popular ones. Spark allows for highly efficient iterative operations in batch on large datasets in memory, has an easy programming interface and is readily available on popular cloud services such as AWS and hence was chosen as the framework.

The architecture of DIMPL is shown in Fig. 5. The in-memory distributed application runs as a cluster on AWS. Spark jobs are run on AWS EMR and data is read from and written to S3 buckets which is similar to HDFS. As the figure shows, the various in-memory RDD transforms that are performed allow the construction of flight plans by subregion with section-wise and subregion-wise distributions. A Spark jobs driver orchestrates the sequence of RDD transformations and actions.

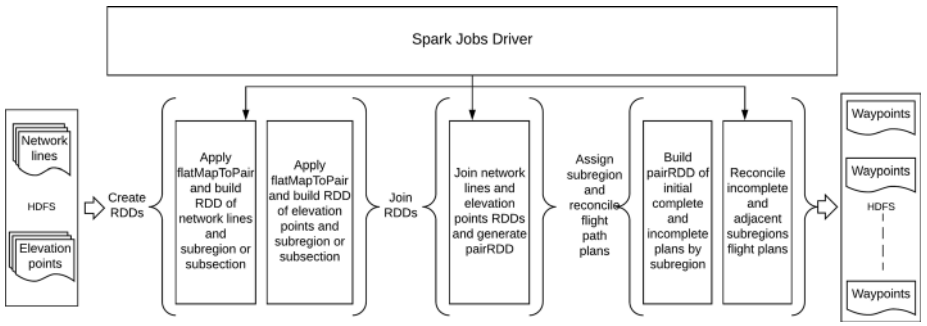

Fig. 5 In-memory distributed system architecture

DIMPL algorithms

We now describe the algorithms that perform the key-value pair based RDD transforms. The < Key,Value > based RDDs are modeled as PairRDD.

Algorithm 6 Section Wise Distribution and Reconciling Vertical Boundary Subregions

Input: $subregion_j \rightarrow xsr_j, ysr_j, xer_j, yer_j$, $\{waypoint_j\}$, quadcopter |conventional, complete |incomplete,$\{elevation_j\}$,$\{networkline_j\}$

Output: $subregion_i$,xer_i,yer_i,xsr_i,xsr_i,$\{waypoint_i\}$, conventional|quadcopter

1: *PairRDD elevationRDD* = elevationData.flatMapToPair
{Convert elevationData to subsection→{elevationPoints}}
2: *PairRDD networklinesRDD* = networklinesData.flatMapToPair
{Convert networklinesData to subsection→{networkLines}}
3: PairRDD<sectionID,Tuple2 >joinOutput = elevationRDD.join(networkRDD)
{join elevationRDD and networklinesRDD by section }
4: PairRDD <sectionID,Tuple2 <subregionID,FlightPlan>>flightPlans = joinsOutput.mapToPair
{Build flight plans of subregions in a subsection }
5: PairRDD <sectionID,Tuple2<subregionID,FlightPlan >>resolveBoundaryPlans = flightPlans.mapToPair
{Resolve boundary flight plans of subregions in a subsection vertical boundary}

Algorithm 6 provides the in-memory section by section flight path implementation in Spark using RDDs. Since all operations are performed in-memory, the algorithm simply transforms the input data into RDD elements that represent all elevation data in step 1 and the network line data in step 2 within each section. The *flatMapToPair* function flattens the data elements for a key into a single value in the transformed RDD. Step 3 joins the two RDDs with their section id and then applies the flight plans builder to the data points in each section. The operation in step 4 allows flight plans to be created by subregion in parallel for each subsection in memory, and finally step 5 resolves the subregions based on the section boundary.

Algorithm 7 SubregionWise Distribution and Reconciliation of Adjacent Incomplete Subregions

Input: $subregion_j \rightarrow xsr_j, ysr_j, xer_j, yer_j$, $\{waypoint_j\}$, quadcopter |conventional, complete |incomplete,$\{elevation_j\}$,$\{networkline_j\}$
Output: $subregion_i$,xer_i, yer_i,xsr_i,xsr_i,$\{waypoint_i\}$, conventional|quadcopter
1: *PairRDD elevationRDD* = elevationData.flatMapToPair
{function to convert elevationData to subregion→{elevationPoints}}
2: *PairRDD networklinesRDD* = networklinesData.flatMapToPair
{function to convert networklinesData to subregion→{networkLines}}
3: PairRDD<subregionID,Tuple2<elevationID,networkID>>joinOutput = elevationRDD.join(networkRDD)
{join elevationRDD and networklinesRDD by subregion }
4: PairRDD <sectionID,Tuple2 <subregionID,FlightPlan>>flightPlans = joinsOutput.mapToPair
{Build flight plans of subregions in parallel }
5: PairRDD <sectionID,Tuple2<subregionID,FlightPlan>>resolveBoundaryPlans = flightPlans.mapToPair
{Resolve flight plans for adjacent incomplete subregions}

Algorithm 7 gives the in-memory subregion by subregion waypoints construction for each flight path in Spark using RDDs. The algorithm transforms the input data into RDD elements that represent all the elevation and network lines data within a subregion and then applies the flight plans builder to the data points in each subregion and to enable flight plans for each subregion to be created in parallel in the drone's memory, with step 5 resolving the incomplete subregions that are adjacent to each other.

Experiments

This section explores a set of scenarios designed to identify appropriate subregions and the type of drone flights needed to cover each. The scenarios are divided into 3 categories, namely within a section, including the entire area on a single node, or distributed by subregion; "Scenarios within a single section" section looks at single section scenarios, "Distributed scenarios" section considers results for distributed scenarios and "Optimized distributed scenarios" section for optimized distributed scenarios.

Scenarios within a single section

These scenarios are how the new model functions when applying queries and constraints within a single section, which can either be the whole area or an individual section within it.

Use of conventional drone A typical flight path built by flight plans builder is shown in Fig. 6. The waypoints for the subregion are highlighted, along with the electric pole network lines they must cover, which are shown in red. The distance δ of the waypoints path along the network line is constant and configurable. The turning radius of the drone, along with its flight from one network line to another, is taken into account in the *autonomy* constraint. It is not shown in the figure or entered as a waypoint as the drone automatically determines how it will navigate from one point to the next. The value of ϵ_{type} is set to 0 for the experiments but operator is given the option to change it if necessary. The circles represent the turns a conventional drone has to make to fly along both sides of a network line.

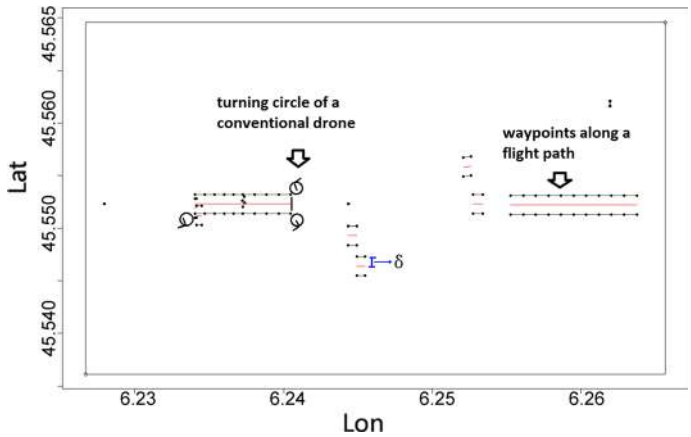

Fig. 6 Waypoints along network lines. δ represents the distance from the line for a subregion covered by a conventional drone

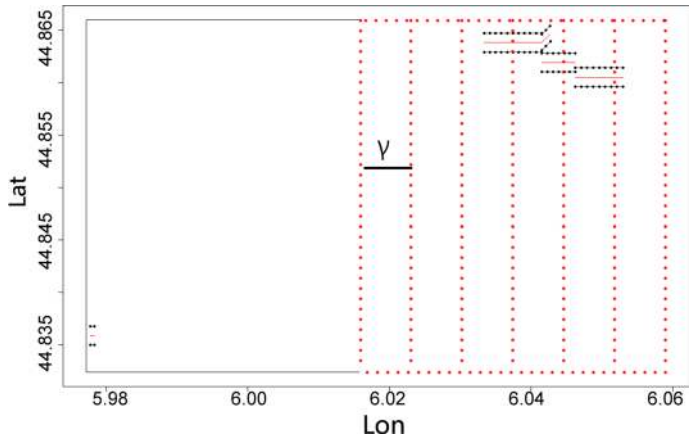

Fig. 7 Expanded view of subregion with network line lengths below the threshold of conventional drone autonomy

Expanding a subregion for a conventional drone or quadcopter Figure 7 shows a subregion which has fewer network lines than β% conventional drone autonomy threshold. The size of the subregion is incrementally expanded by length γ until autonomy reaches β% or higher. This parameter is configurable and defaults to 80%. The dotted lines in the figure represent the incremental expansion of the subregion.

Splitting a subregion between conventional drone and quadcopter flights If a subregion fails to satisfy the climbing angle constraint for a conventional drone, it is split to separate out the segments where the constraint fails and these portions assigned to a quadcopter. Figure 8 shows the flight path waypoints for a subregion that cannot be covered by a conventional drone and thus require a quadcopter. A larger subregion with this split is shown in Fig. 9. Quadcopter default size subregions are incrementally applied to determine where the conventional drone climbing angle constraint fails. If that subregion lies in the middle of the original subregion, it is covered with a quadcopter and all the other subregions before it in the sequence are assigned to a minimal number of conventional

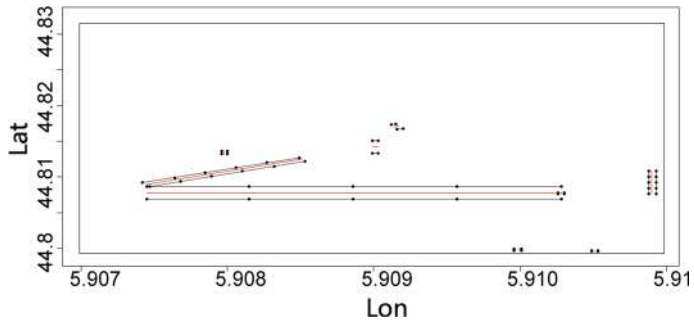

Fig. 8 Waypoints along network lines. δ represents the distance from a line for subregions covered by a quadcopter

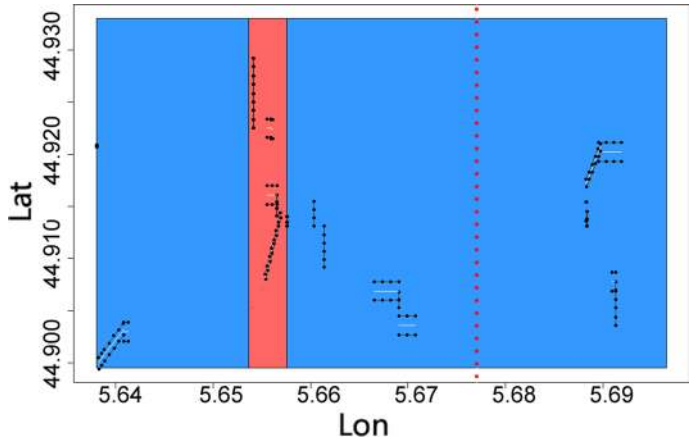

Fig. 9 Subregions that fail to satisfy the climbing angle constraint and are thus split between conventional drones (in blue) and a quadcopter (in red)

drone flights. Subregions following the final quadcopter segment are merged with subsequent conventional drone subregions, as indicated by dotted line. Only one subregion that required quadcopter coverage was found in the example shown here, so after that subregion the algorithm resumes the query process by once again applying the default conventional drone subregion size.

The quadcopter default sized subregions before the quadcopter assigned subregion are merged together to create a single conventional drone subregion.

Shrinking subregions for conventional drones or quadcopter Subregions containing network lines that are too long for a conventional drone or quadcopter to cover in a single flight necessitate shrinking that subregion. Figure 10 shows the process involved in incrementally shrinking a subregion by horizontal length γ until a size that satisfies the autonomy constraint for a conventional drone is reached. This reduces the length of the network lines until they can be covered by a conventional drone. The query process then resumes from the end of the new, smaller subregion. The dotted lines represent incremental changes in the subregions.

After experimenting with several default subregion sizes for both types of drones, intial default subregion sizes of 3.7 km $\times$ 2.9 km for conventional drones and 3.7 km $\times$ 0.29 km

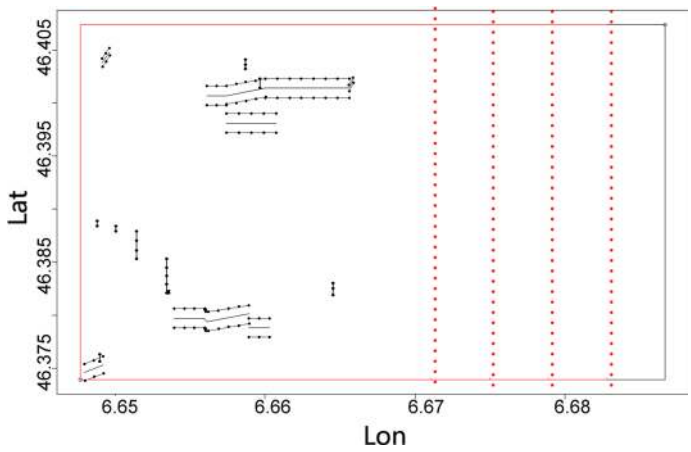

Fig. 10 Subregion containing network lines that exceed conventional drone autonomy and must be shrunk

for quadcopters were found to be good default sizes. These default values are an optimal size that simultaneously maximizes the autonomy of the conventional drones and quadcopter and minimizes the processing time. Figure 11 shows the subregions in the study area with some subregions covered by conventional drones and the others by quadcopters. Some subregions have been shrunk for the conventional drones and the others split for quadcopters and the remaining segments merged into an adjoining subregion. The subregions in the figure provide a good match for the network lines and their density, as shown in Fig. 1. Areas with dense network lines have more default sized and smaller conventional drone subregions. Areas with more quadcopter subregions indicate hilly areas with substantial altitude variations, while areas with sparse coverage of network lines contain a greater number of expanded conventional drone subregions.

The optimizations process significantly reduces the number of drone subregions overall and minimizes the number of quadcopter flights, which is important as conventional drones are both cheaper and more plentiful. The impact of incremental optimizations is shown in Fig. 12. The incremental savings gained due to expansions and splitting of

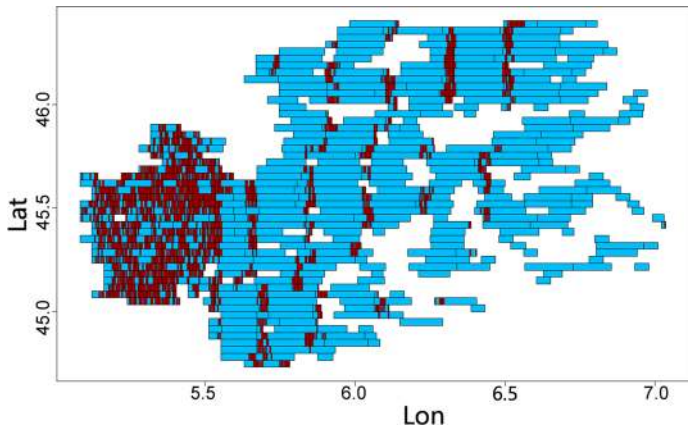

Fig. 11 All the subregions in the study area, conventional drone areas shown in blue and quadcopter areas in red

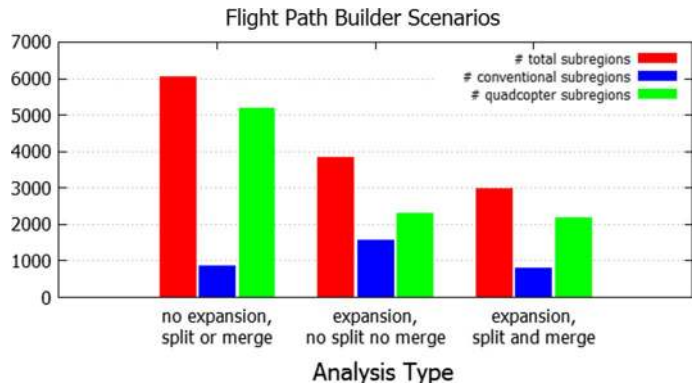

Fig. 12 All the subregions for both conventional drones and quadcopters broken down by analysis type

conventional default sized subregions into a minimal number of quadcopter and conventional drone subregions is very effective in minimizing the number of subregions, particularly those requiring quadcopter subregions. The bulk of the efficiency savings come from expanding subregions and thus reducing the number of conventional drone subregions from 1553 to 809. Splitting a conventional drone subregion into pinpoint quadcopter areas and efficiently merging conventional drone subregions minimizes the number of quadcopter subregions required.

Distributed scenarios

In a distributed paradigm, the scenarios change as subregions on the edges of sections are unable to expand and thus must be resolved in an additional step.

Shrink, expand, merge or split subregions The subregions are shrunk and expanded based on the same threshold β% for autonomy as that used in the sequential algorithm for each section. This invariably means several subregions on the extreme edge of each section fail to meet the threshold as they lack room to expand. This issue is mitigated to some extent by the edge effect task.

Merge section edge subregions Edge subregions for each section can be merged together if feasible. The result after merging for a 4-section distribution are shown in Fig. 13. If the subregions are being covered by the same drone type on either side of an edge and they have not been expanded, they are likely to be merged together. Experiments identified 4 such subregions in the study area.

Optimized distributed scenarios

Optimized distribution focuses on processing each subregion in parallel. The subregions are then reconciled by merging them with adjacent subregions where necessary.

Splitting or shrinking subregion Since a subregion by itself cannot be expanded, the length of its network lines can only be shrunk if its network lines exceeds the autonomy of a conventional drone or quadcopter. Subregions that do not satisfy the β% network line length constraint after splitting or shrinking are marked as incomplete. Subregions that satisfy the length constraint are emitted as complete.

Merging subregions All incomplete subregions are then ordered and those adjacent to each other that can be merged are emitted as new subregions. Figure 14 shows the

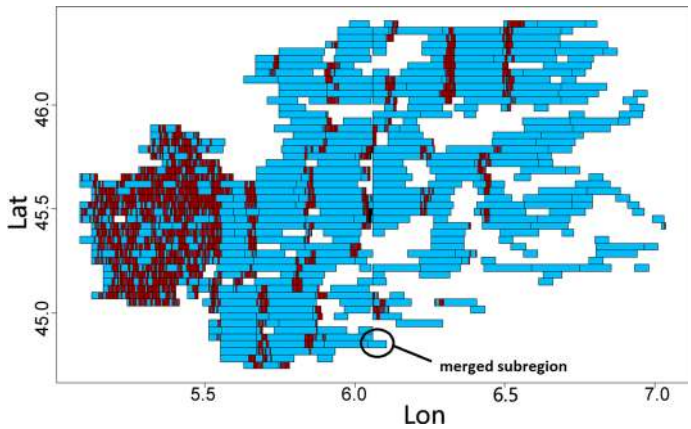

Fig. 13 The subregions in overall area with conventional drone areas shown in blue, and quadcopter areas in red after merging subregions across the vertical section boundaries

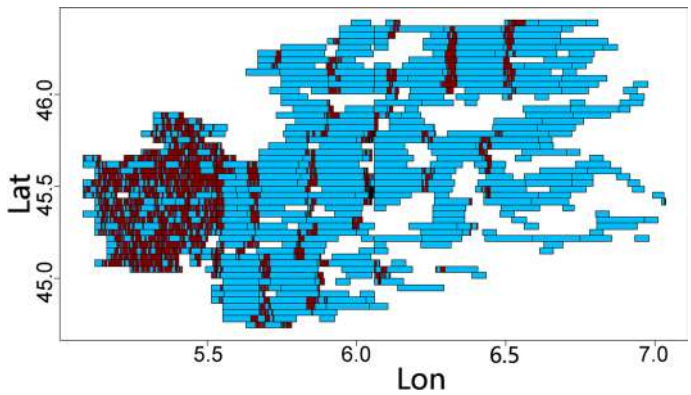

Fig. 14 The subregions in the study area, with conventional drone areas shown in blue and quadcopter areas in red after subregion-wise distributed processing and merging

subregions created by this subregion parallelization process. The entire set of subregions that can be processed together are merged if possible in a single pass. This is optimal for when large number of processor nodes are available in distributed cluster and the jobs can complete deterministic calculation of subregions and flight paths of each while taking full advantage of parallelization possible. Running the scenarios in the distributed paradigm produces similar number of subregions for both quadcopter and conventional drone types to those obtained with sequential processing. Merging reduces the quadcopter subregions by 627 and conventional subregions by 385.

Comparisons of single node sequential results with distributed and optimized distributed results are shown in Fig. 15. The figure clearly shows how merging reduces the count of quadcopter and conventional drone subregions in an optimized distribution. After merging, there are 1239 conventional drone subregions and 1839 quadcopter subregions, which compares well with the 809 conventional drone subregions and 2181 quadcopter subregions obtained using a single node execution process.

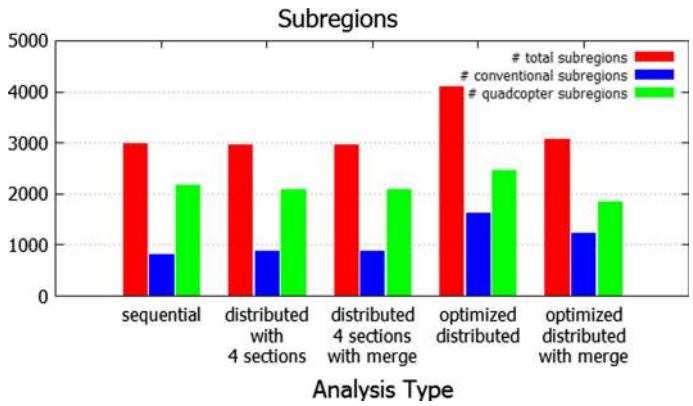

Fig. 15 All the subregions for conventional drones and for quadcopters broken down by analysis type

Results

This section presents results on user provided and synthetic datasets. Performance results for a user provided dataset are presented in "Performance on the user provided dataset" section and those for an extended synthetic dataset in "Performance on larger synthetic dataset" section. The results are analyzed in "Discussion" section. The Spark and MapReduce experiments were run on AWS using Elastic MapReduce clusters. Cluster nodes are of type m3.2 × large with 8 vCPU processors and 30GB of RAM.

Performance on the user provided dataset

A comparison of the performance of sequential execution against distributed execution in DIFPL and DIMPL across the 7800 km^2 study area for section wise distribution in clusters of various sizes for an overall area divided into 4 and 16 sections is shown in Fig. 16. Hadoop 2.5.1, MapReduce2 and Spark1.5.1 were utilized and experiments are performed on 3, 5 and 9 node Hadoop clusters with 1 master and 2, 4 and 8 slave nodes.

The results show that although with increasing cluster size, although due to the relatively small data size in this example the performance improvement from 4 to 16 sections is not significant. The results of subregion based distribution in MapReduce for DIFPL and Spark for DIMPL are shown in Fig. 17. Here again the section-wise distribution

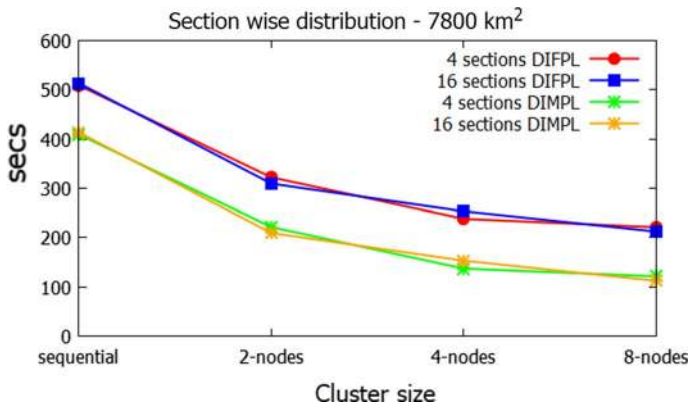

Fig. 16 Performance of section-wise distribution in MapReduce and Spark for various sized clusters

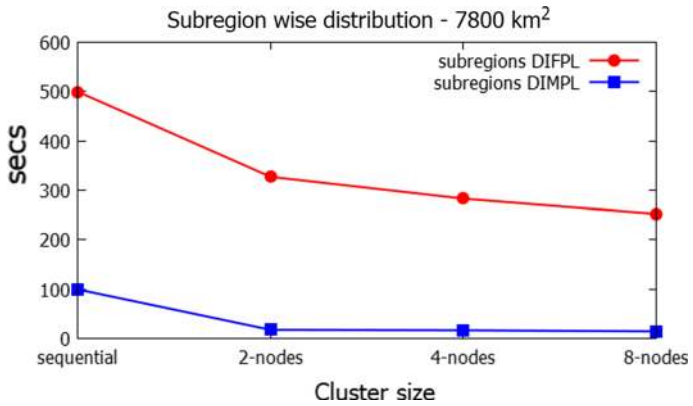

Fig. 17 Performance of subregion-wise distribution in MapReduce and Spark for various sized clusters

performance improves with increasing cluster size. With less disk I/O, the performance of DIMPL is significantly better due to all the operations being performed in memory.

Performance on larger synthetic dataset

In order to test the performance of the algorithms on a much larger data set, we created a synthetic dataset 100 times the size of the user provided dataset. The larger dataset has the same distribution and density of elevation points and network lines as the original dataset. It was created by tiling the original dataset 100 times in a 10×10 grid. This made sure we did not alter the data in any way but performed experiments that did not assume anything about the network or elevation points and algorithms operated on a much larger region. Experiments on the larger dataset confirm the scalability of the algorithms.

A comparison of the performance of sequential execution and that of distributed execution in DIFPL MapReduce and DIMPL in Spark for section-wise distribution in clusters of various sizes for the overall area divided into 4 and 16 sections is shown in Fig. 18. These results clearly show the improvement in performance as the number of subsections increases from 4 to 16. A comparison of the results of subregion based distribution in MapReduce and Spark is shown in Fig. 19. Here again the performance of

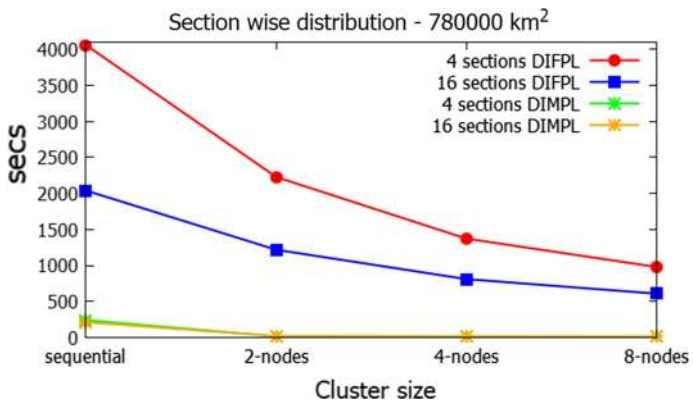

Fig. 18 Performance of section wide distribution in MapReduce and Spark for various sized clusters and a larger dataset

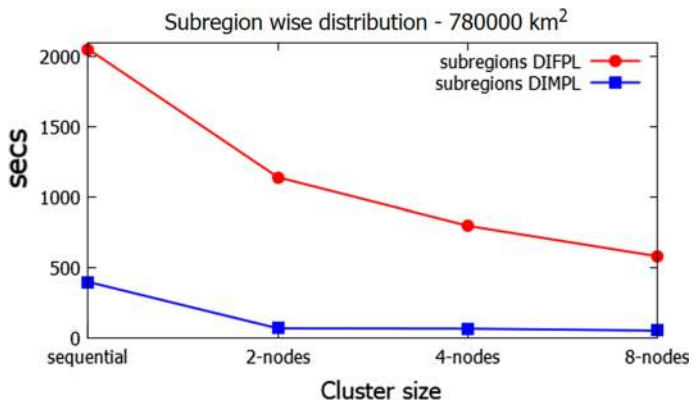

Fig. 19 Performance of subregion wise distribution in MapReduce and Spark for various sized clusters and a larger dataset

DIMPL is much better than that achieved by DIFPL due to all operations being performed in memory on clusters with multiple processors and a large amount of memory. These results clearly show the performance improvement achieved by using in-memory distribution in Spark in DIMPL compared to the disk based distribution in MapReduce in DIFPL. These results confirm that the performance on larger datasets mirrors the performance on the full original dataset provided by the aviation agency. This proves our technique is scalable to larger sized datasets.

Discussion

The performance results show that the algorithms are scalable in both DIFPL and DIMPL. The experimental results verified following observations: *(1) Improvements in distributed and optimized distributed scenarios* Distribution and optimized distribution scenarios reduce the number of subregions overall and minimize the number of quadcopter regions. The results from distribution and optimized distribution are deterministic and do not fluctuate from one run to the next. Optimized distribution is able to leverage large number of nodes in a MapReduce or Spark cluster if available to run the analysis speedily and in case of DIMPL in near real time to allow operator tremendous flexibility in building flight paths. *(2) Scaling efficiency* The algorithms scale well with larger datasets. The scaling with both DIFPL and DIMPL is efficient and we further validate that with strong and weak scaling [34] results as shown in Fig. 20 for DIFPL algorithm and Fig. 21 for DIMPL. In strong scaling the problem size stays the same while we measure run time by gradually increasing the number of nodes. In weak scaling we keep the workload on each processor the same while gradually increasing the size of problem along with number of processors. The scaling charts show that with increasing number of processors the performance keeps improving, and stays stable for same amount of data per node for increasing data sizes. The scaling charts demonstrate that the algorithms can be scaled indefinitely with increasing survey area in a horizontal manner by simply adding nodes to the cluster. During the running of DIFPL MapReduce and DIMPL Spark distribution jobs the memory of the nodes increases rapidly as more mappers and reducers in case of MapReduce and executors in case of Spark are started. It stabilizes after a while and processes proceed to release it upon completion. *(3) kNN*

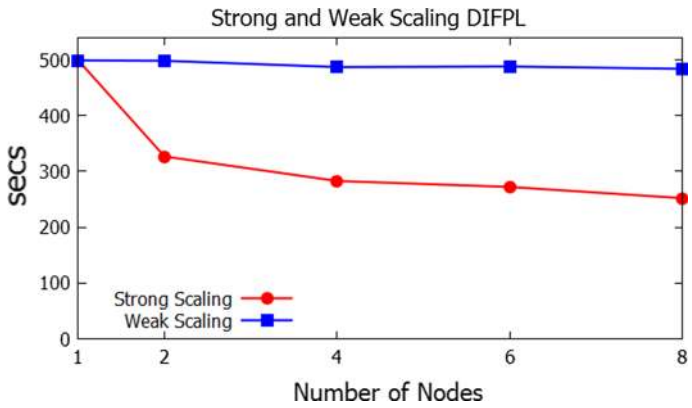

Fig. 20 DIFPL strong and weak scaling

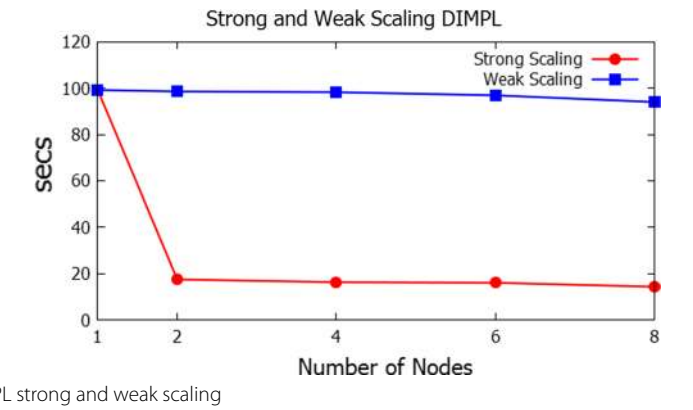

Fig. 21 DIMPL strong and weak scaling

query impact The invocation of kNN queries over elevation points with maximum elevation instead of average in results used to determine elevation of waypoint increased the number of quadcopter regions by 54 from 2181 to 2235 while number of conventional drone regions decreased by 1 from 809 to 808 for the standard size data set in sequential execution. The increase in quadcopter regions was due to several conventional drone subregions split into quadcopter subregions with some of the smaller split subregions in between assigned to conventional drones. This proves the usefulness of providing this option to the drone operator to build flight plans suitable to the terrain.

Conclusions

This study used a novel approach to flexibly divide a large area into subregions and dynamically adjust them to optimally cover each with a single drone flight. The algorithms combine spatial data and drone limitations or constraints, which are modeled as linear inequalities, to automate the flight paths of the drones. The distributed implementation provides an excellent way to handle large datasets that cannot be processed on a single node. Utilizing in-memory distribution significantly speeds up the flight paths building process compared to disk based distribution and this subregion level distribution allows horizontal scalability. The flight plans produced by the new distributed

model are similar in numbers to those obtained by single node implementations but are generated more efficiently. Their construction can also be adjusted for dynamic evasion of unforeseen obstruction. The technique applied here is not only useful for the assigned task of surveying power lines but can easily be extended to a host of other drone applications such as surveying coastlines for hurricane damage, forest surveys for logging, farm surveys for fertilization, insecticide spraying and watering, and many others.

Abbreviations
DIFPL: DIstributed Flight Path buiLder; GPS: global positioning system; DIMPL: Distributed, In-Memory flight Path buiLder; VRP: vehicle routing problem; TSP: traveling salesman problem; EMR: elastic map reduce; AWS: Amazon Web Services; EC2: elastic compute cloud; HDFS: Hadoop distributed file system; RDD: resilient distributed dataset; RAM: random access memory; UAV: unmanned aerial vehicle; KML: Keyhole Markup Language; MPI: Message Passing Interface.

Authors' contributions
MS was the primary author with ZC and C-TL contributing with running experiments and refining the concepts discussed and format of the paper. All authors read and approved the final manuscript.

Author details
[1] Omniscience Corporation, Palo Alto, CA, USA. [2] Virginia Tech, Falls Church, VA, USA.

Acknowledgements
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Availability of data and materials
The data and material for paper are not being made available.

Consent for publication
All authors have consented for publication of this paper.

Ethics approval and consent to participate
All authors give ethics approval and consent to participate in submission and review process.

Funding
Not applicable.

Publisher's Note

Received: 28 January 2018 Accepted: 4 July 2018
Published online: 12 July 2018

References
1. Aji A, Wang F, Vo H, Lee R, Liu Q, Zhang X, Saltz J. Hadoop GIS: a high performance spatial data warehousing system over mapreduce. Proc VLDB Endow. 2013;6(11):1009–20.
2. Apache and Hadoop. 2014. http://hadoop.apache.org. Apache Hadoop.
3. Apache Foundation. 2018. https://flink.apache.org/. Apache Flink.
4. Apache Foundation. 2018. http://storm.apache.org/. Apache Storm.
5. Babaei AR, Mortazavi M. ree-dimensional curvature-constrained trajectory planning based on in-flight waypoints. J Aircraft. 2010;47:1391–8.
6. Babel L. Flight path planning for unmanned aerial vehicles with landmark-based visual navigation. Robot Autonom Syst. 2014;62(2):142–50.
7. Band R, Pleban J, Schn S, Creutzburg R, Fischer A. Concept for practical exercises for studying autonomous flying robots in a university environment: part i. Proc SPIE. 2013;8667:86670P.
8. Bills C, Chen J, Saxena A. Autonomous MAV flight in indoor environments using single image perspective cues. In: IEEE international conference on robotics and automation (ICRA). 2011. p. 5776–83.
9. Bills C, Chen J, Saxena A. Autonomous MAV flight in indoor environments using single image perspective cues. In: 2011 IEEE international conference on robotics and automation (ICRA). 2011.
10. Banu TP, Borlea GF, Banu C. The use of drones in forestry. J Environ Sci Eng B. 2016;5(2016):557–62.
11. Cary A, Sun Z, Hristidis V, Rishe N. Experiences on processing spatial data with MapReduce. In: Proceedings of the 21st international conference on scientific and statistical database management, SSDBM. Berlin: Springer; 2009. p. 302–19.
12. Chakrabarty A, Langelaan J. UAV flight path planning in time varying complex wind-fields. In: Proceedings of the American control conference. 2013.

13. Chang W-Y, Hsiao F-B, Sheu D. Two-point flight path planning using a fast graph-search algorithm. J Aerospace Comput Inf Commun. 2006;3:453–88.
14. Chmaj G, Selvaraj H. Distributed processing applications for UAV/drones: a survey. In: Selvaraj H, Zydek D, Chmaj G, editors. Progress in systems engineering. Advances in intelligent systems and computing, vol. 1089. Berlin: Springer; 2015. p. 449–54.
15. De Paula Santos G, Garcia Marques L, Miranda Neto M, Cardoso A, Lamounier E, Yamanaka K. Development of a genetic algorithm to improve a UAV route tracer applied to a man-in-the-loop flight simulator. In: 2013 XV symposium on virtual and augmented reality (SVR). 2013. p. 284–7.
16. Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Commun ACM. 2008;51(1):107–13.
17. Deng T, Xiong ZM, Liu YJ, Meng QZ. Research on 3d route planning for UAV in low-altitude penetration based on improved ant colony algorithm. Appl Mech Mater. 2013;442:556–61.
18. Do T, Carrillo-Arce LC, Roumeliotis SI. Autonomous flights through image-defined paths. In: Bicchi A, Burgard W, editors. Robotics research. Springer proceedings in advanced robotics, vol. 2. Cham: Springer; 2018.
19. Dorling K, Heinrichs J, Messier GG, Magierowski S. Vehicle routing problems for drone delivery. IEEE Trans Syst Man Cybernet. 2017;47(1):70–85.
20. Eldawy A, Li Y, Mokbel MF, Janardan R. CG_Hadoop: computational geometry in MapReduce. In: ACM SIGSPATIAL. 2013. p. 294–303.
21. Eldawy A, Mokbel MF. A demonstration of SpatialHadoop: an efficient MapReduce framework for spatial data. Proc VLDB Endow. 2013;6(12):1230–3.
22. Floreano D, Wood RJ. Science, technology and the future of small autonomous drones. Nature. 2015;521:460–6.
23. Gabriel E, Graham EF, George B, Thara A, Jack JD, Jeffrey MS, Vishal S, Prabhanjan K, Brian B, Andrew L, Ralph HC, David JD, Richard LG, Timothy SW. Open MPI: goals, concept, and design of a next generation MPI implementation. In: Proceedings, 11th European PVM/MPI users' group meeting, Budapest, Hangary; 2004. p. 97–104.
24. Holub JS. Improving particle swarm optimization path planning through inclusion of flight mechanics. Graduate theses and dissertations. Paper 11741. 2010.
25. Hall J, Anderson D. Reactive route selection from pre-calculated trajectories—application to micro-UAV path planning. Aeronaut J. 2011;115(1172):635–40.
26. Juan AA, Faulin J, Jorba J, Caceres J, Marquès JM. Using parallel & distributed computing for real-time solving of vehicle routing problems with stochastic demands. Ann Operat Res. 2013;207(1):43–65.
27. Kim S, Lim G. Drone-aided border surveillance with an electrification line battery charging system. J Intell Robot Syst. 2018;2018:1–14.
28. Koh LP, Wich SA. Dawn of drone ecology: low-cost autonomous aerial vehicles for conservation. Trop Conserv Sci. 2012;5(2):121–32.
29. Kopfer H, Benedikt V, Jan D. Vehicle routing with a heterogeneous fleet of combustion and battery-powered electric vehicles under energy minimization. In: International conference on computational logistics. Cham: Springer; 2017.
30. Krajnik T, Vonasek V, Fiser D, Faigl J. *AR* drone as a platform for robotic research and education. Commun Comput Inf Sci. 2011;161:172–86.
31. Krishnan A, Markov M, Distributed BB. Approximation vehicle routing. In: IEEE international parallel and distributed processing symposium. Orlando: IPDPS; 2017. p. 503–12.
32. Li L, Liu X, Peng ZR, Xu X. Multi-objective optimization model and evolutionary algorithm to plan UAV cruise route for road traffic surveillance. In: Transportation research board 92nd annual meeting. 2013.
33. Lugo JJ, Zell A. Framework for autonomous on-board navigation with the AR drone. J Intell Robot Syst. 2014;73(1–4):401–12.
34. Moreland K, Ron O. Formal metrics for large-scale parallel performance. In: International conference on high performance computing. Cham: Springer; 2015. p. 488–96.
35. Saska M, Krajnik T, Faigl J, Vonasek V, Preucil L. Low cost MAV platform AR-drone in experimental verifications of methods for vision based autonomous navigation. In: 2012 IEEE/RSJ international conference on intelligent robots and systems (IROS). 2012. p. 4808–9.
36. Segor F, Bürkle A, Kollmann M, Schönbein R. Instantaneous autonomous aerial reconnaissance for civil applications. In: ICONS 2011, the sixth international conference on systems. St. Maarten, The Netherlands Antilles. 2011.
37. Shiun JS, Hsiang LY. Integrated flight path planning system and flight control system for unmanned helicopters. Mol Divers Preserv Int (MDPI). 2011;11:7502–29.
38. Shukla M, Chen Z, Lu CT. DIFPL—distributed drone flight path builder system. In: Proceedings of the 1st international conference on geographical information systems theory, applications and management, GISTAM, Barcelona, Spain. 2015, p. 17–26.
39. Ta DN, Ok K, Dellaert F. istas and parallel tracking and mapping with wall–floor features: enabling autonomous flight in man-made environments. Robot Autonom Syst. 2014;62(11):1657–67.
40. Toth P, Daniele V, eds. The vehicle routing problem. In: Society for industrial and applied mathematics. 2002.
41. Tripicchio P, Satler M, Dabisias G, Ruffaldi E, Avizzano CA. Towards smart farming and sustainable agriculture with drones. IN: 2015 international conference on intelligent environments, Prague. 2015. p. 140–3. https://doi.org/10.1109/IE.2015.29.
42. Visse A, Dijkshoorn N, van der Veen M, Jurriaans R. Closing the gap between simulation and reality in the sensor and motion models of an autonomous AR drone. In: Proceedings of the international micro air vehicles conference. 2011.
43. Wang K, Han J, Tu B, Dai J, Zhou W, Song X. Accelerating spatial data processing with MapReduce. In: Proceedings of the 2010 IEEE 16th international conference on parallel and distributed systems, ICPADS '10, Washington, DC, USA. 2010. p. 229–36.
44. Wang X, Poikonen S, Golden B. The vehicle routing problem with drones: several worst-case results. Optim Lett. 2017;11:679.

45. Williams S. Studying volcanic eruptions with aerial drones. Proc Natl Acad Sci USA. 2013;110:10881.
46. Yoo S, Kim K, Jung J, Chung AY, Lee J, Lee SK, Lee HK, Kim H. Poster: a multi-drone platform for empowering drones' teamwork. In: Proceedings of the 21st annual international conference on mobile computing and networking, MobiCom '15. New York: ACM; 2015. p. 275–7.
47. Yu J, Wu J, Sarwat M. GeoSpark: a cluster computing framework for processing large-scale spatial data. In: Proceedings of the 23rd SIGSPATIAL international conference on advances in geographic information systems, SIGSPATIAL '15. Seattle, Washington: ACM; 2015. p. 1–70.
48. Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I. Spark: cluster computing with working sets. In: Proceedings of the 2Nd USENIX conference on hot topics in cloud computing, HotCloud'10. Berkeley: USENIX Association; 2010. p. 10.