Lopez-Novoa, U., Aguilera, U., Emaldi, M. et al. *Overcrowding detection in indoor events using scalable technologies*. **Pers Ubiquit Comput** 21, 507–519 (2017). This version of the article has been accepted for publication, after peer review (when applicable) and is subject to Springer Nature's AM terms of use, but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record is available online at: https://doi.org/10.1007/s00779-017-1012-6

Overcrowding Detection in Indoor Events using Scalable Technologies

Unai Lopez-Novoa · Unai Aguilera · Mikel Emaldi · Diego López-de-Ipiña · Iker Pérez-de-Albeniz · David Valerdi · Ibai Iturricha · Eneko Arza

Received: date / Accepted: date

Abstract The increase in the number of large scale events held indoors (i.e. conferences and business events) opens new opportunities for crowd monitoring and access controlling as a way to prevent risks and provide further information about the event's development. In addition, the availability of already connectable devices among attendees allows to perform non-intrusive positioning during the event, without the need of specific tracking devices. We present an algorithm for overcrowding detection based on passive Wi-Fi requests capture and a platform for event monitoring that integrates this algorithm. The platform offers access control management, attendees monitoring and the analysis and visualization of the captured information, using a scalable software architecture. In this paper, we evaluate the algorithm in two ways: first, we test its accuracy with data captured in a real event, and then we analyse the scalability of the code in a multi-core Apache Spark-based environment. The experiments show that the algorithm provides accurate results with the captured data, and that the code scales properly.

This work has been partially supported by the Basque Country Government under the Gaitek funding program (IG-2014/00172) and the Spanish Ministry of Economy and Competitiveness (grant number TIN2013-47152-C3-3-R).

U. Lopez-Novoa · U. Aguilera · M. Emaldi · D. López-de-Ipiña
Deusto Institute of Technology, DeustoTech, University of Deusto, Avda. de las Universidades 24, 48007, Bilbao, Spain
email{unai.lopez, unai.aguilera, m.emaldi, dipina}@deusto.es

I. Pérez-de-Albeniz · D. Valerdi
Fon Labs, Avda. Las Arenas 7, 48930, Getxo, Spain
email{iker.perez, david.valerdi}@fon.com

I. Iturricha · E. Arza
Aditium, Abadetxe Kalea 9, 48180, Loiu, Spain
email{i.iturricha, e.arza}@aditium.com

Keywords Overcrowding detection · Indoor location · Scalable data processing

1 Introduction

Conferences and business events are growing both in number and size in the last years. According to the latest Statistics Report of the International Congress and Convention Association (ICCA) [11], the number of meetings has grown exponentially by approximately 10% each year from 1963, reaching 173.432 in 2012. In 2014, Paris was the city holding the greatest number of events: 214 in the whole year. In addition, the number of participants in these events usually ranges from hundreds to thousands. According to the same ICCA report, there was an average of 424 participants per international conference in 2012.

These large scale meetings are usually held in buildings whose capacity is limited by several factors: the size of the building, the number and size of its doors, the availability of paths for impaired people, and so on. Exceeding the capacity of such buildings can be harmful if emergencies happen, and the usual way to prevent such issues is controlling the number of accesses through the doors.

Controlling the entries and exits to the building is a first step to prevent risks in large events, but it does not provide the organizer with further information on how the attendees are behaving. Even if the capacity is not exceeded, important risks are taken if most of the participants are located around a single spot, or if the accesses are saturated. Such information could be useful both while an event is being held and as an aid to plan future meetings better.

In this paper we present a platform aimed to support event organizers with real-time valuable information. In particular, it shows data from the access control system that registers the attendees, and the flow of persons within the building. This last feature is computed by an algorithm that, given tracking information from the attendees in the event, calculates the potential overcrowded spots. The tracking information that feeds the algorithm are wireless frames captured by standard Wi-Fi routers deployed along the monitored building.

We developed our software platform using the Cloudera[1] framework, a suite of scalable data processing tools that ensures reliability in scenarios with large data processing needs. On top of these tools, we implemented the overcrowding detection algorithm and the application that visualizes the processed information in an amenable way.

The remainder of the paper is structured as follows: Section 2 details the system used to capture data in an event, and Section 3 describes the algorithm to detect overcrowding. Section 4 describes briefly the Cloudera framework and Section 5 how we used it in this work. We describe in Section 6 how we evaluated the accuracy of our algorithm with a set of real data collected in the 4YFN 2015 conference and in Section 7 how we evaluated the scalability of the algorithm in an Apache Spark environment. Finally, we describe in Section 8 tools and algorithms similar to what we presented and we draw in Section 9 some conclusions and future lines of work.

2 Data capture systems

This section describes the mechanisms used by our platform to gather monitoring information in an event: wireless frame capture and attendee access control.

2.1 Wireless frame capture

The first step to device monitoring in our platform requires a minimal instrumentation of the building where an event takes place: Wi-Fi routers must be placed covering the meeting space prior to the beginning of the event. Then, these routers will capture any attempt of a Wi-Fi connection to them, e.g. from smartphones or tablets.

Wi-Fi enabled devices periodically send probe requests frames to find out what access points are in their vicinity. These frames include information such as its MAC address and its wireless connection capabilities

[1] Cloudera - http://www.cloudera.com

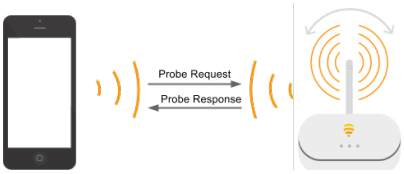

Fig. 1: Device detection based on probe request.

(e.g. whether 802.11b/g/n is supported) [17]. The frequency of (passive) sending depends on the operating system, applications, connectivity status, etc. of the device. In addition, active scans can be forced by the user manually.

Wireless access points reply to these frames with probe responses, which describe Wi-Fi capabilities of the access points and the networks they populate [17]. This exchange of information prepares the connection setup between the device and the access point. A representation of this interaction is depicted in Figure 1.

Our solution is based on the capture of probe requests by the access points. It captures the frame, extracts the MAC address (which uniquely identifies the Wi-Fi enabled device), and adds the timestamp and received power or Received Signal Strength Indication (RSSI). This data is stored by the access point during a pre-configured period of time. After this period expires, the access point groups it as per MAC address, and hashes it to protect anonymity. Metrics like maximum, minimum, averaged received power, timestamp of first and last frame,... are computed during collection time as well.

The collected information is periodically sent to the central server of our platform for its processing using a JSON description file. An HTTPS connection is used, as it provides additional security and prevents from content interception.

2.2 Attendee access control

Besides frames from wireless devices, our platform gathers data coming from the physical access controls to the buildings. These controls provide real time information about the attendees that are entering/exiting the event's area. Each attendee carries its own Near Field Communication (NFC) badge. When an attendee registers his/her badge is personalized, i.e. his/her information is linked to the badge. The personalization is both physical and logical. The badge is printed with attendees' personal information and the database is updated with the badge's unique ID.

Every time an attendee enters or exits the venue, the badge is validated using NFC enabled smartphones.

The software in the smartphone connects to a server that will be able to report who is inside the building at a given time. One of the main benefits of this approach is that all the information used to determine if a badge is allowed or not, is editable in real time. This way, if an attendee loses its badge, the staff can disable it.

The access control process is divided in three main modules: accreditation, validation and management.

- The *accreditation* module manages the life-cycle of an attendee, which can be in one of the following states: pendant, if the attendee has not received its access credential; identified, the attendee has received is credential that enables to access the event and, removed, if the attendee has been removed from the system. In addition, this module controls the NFC device (smartphone or contactless card) used by attendees to access the event.
- The *validation* module decides whether users can access the event's area through a turnstile or other controlled access with the NFC identifier.
- The *management* module orchestrates the access control system and obtains data about the attendees, which can be used to perform analytics by the platform.

Every time an attendee enters or exits the area of the event, the access control system sends a JSON object to the processing server that contains the information of the event. Multiple events occurring in some predefined timespan can be aggregated to reduce the volume of information sent to the platform.

The information provided in the JSON, for each access action, is the following: action *type*, which describes if the attendee enters or exits the area; an *identifier* of the attendee; the *name* of the company provided by attendee when registered into the event; the *group* that the attendee belongs to (e.g speaker, staff, or general attendee) and the *timestamp* of the action.

3 The overcrowding detection algorithm

This section describes the algorithm that, using the data described in the previous section, calculates the density of devices inside the area of the event. This process is performed in two steps: device tracking and density calculation.

3.1 Device tracking

The tracking algorithm takes the information periodically provided by the Wi-Fi routers and computes the 2D positions of the detected devices in the event area. To that end, it uses as input:

- Hashed MAC address of each device, which identifies it uniquely.
- Average signal level (RSSI) of all received transmissions during the last reporting period.
- *timestamp* and *frequency* of the latest detected transmission from the device.

We use Trilateration [6] as tracking algorithm, a popular positioning technique whose main drawback is that each device must be detected by three different sensors for its position to be estimated. The system uses a time window to aggregate the information coming from different sensors in order to avoid the requirement of simultaneous detection. Therefore, the information captured from the Wi-Fi sensors has a validity period and it is discarded when it exceeds that period.

When three or more sensors detect the same device within the configured time window, the system calculates its position in the defined area as follows: first, the distance between the Wi-Fi router and the device is computed using the Free Space Path Loss (FSPL) equation:

$$FSPL(dB) = 20\log d + 20\log f - 27.55 \tag{1}$$

where d represents the distance in meters between the emitter and the receiver, f is the frequency of the transmitted signal and $FSPL$ provides the difference in dB due to the propagation loss. This formula can be solved to obtain the distance in meters d, knowing the frequency of the signal and its strength, as:

$$d = 10^{\frac{27.55 - 20\log f + PL}{20}} \tag{2}$$

where PL provides the estimated loss of the signal between the emission and its reception, as measured by the Wi-Fi receivers (the RSSI).

After obtaining the distance in meters from the client device to each of the detecting sensors, trilateration is applied. It computes the position of the detected device relative the Wi-Fi routers. This process has been depicted in Figure 2, where $R1, R2, R3$ represent three routers and P the position to be computed.

The process begins with the following equations:

$$\begin{aligned} d_1^2 &= x^2 + y^2 + z^2 \\ d_2^2 &= (x-a)^2 + y^2 + z^2 \\ d_3^2 &= (x-c)^2 + (y-b)^2 + z^2 \end{aligned} \tag{3}$$

where d_1, d_2, d_3 represent the distances d from the device to each router using the FSPL formula, and a, b, c

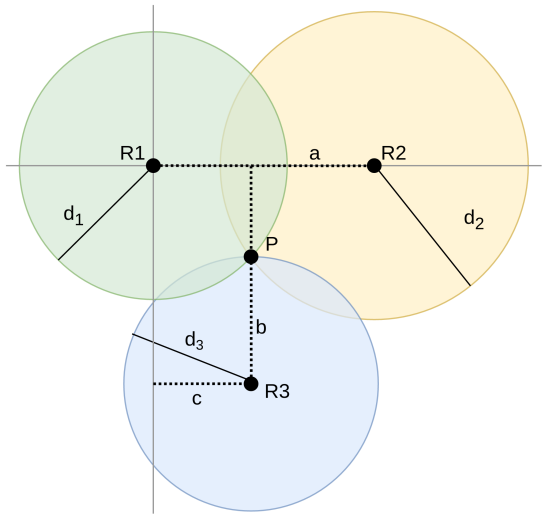

Fig. 2: Representation of the trilateration process

several distance metrics between the routers. Solving this equation system leads to obtaining x, y, z, i.e., the coordinates for the P point relative to the routers. Then, a transformation is applied to make the position relative to the origin of coordinates used by the event.

Each time the Wi-Fi routers collect new data from a device, it is sent to the platform and the device's position is updated. The platform stores the updated position of all detected devices. However, this information can be outdated if no new updates are obtained from three different sensors in the same time windows. Therefore, the system adds a timestamp to the position of each located device that can be used to know the antiquity of the tracking information for each detected device.

As a final note, we refer the interested reader to [20] for a deeper explanation on the trilateration algorithm and its limitations when used with signal strength measurement.

3.2 Device density calculation

Using the information obtained in the previous step we calculate the device density, which is used to estimate the existence of overcrowded zones in the area of the event.

The algorithm subdivides the area in a grid of cells that are used as the finest granularity for the density estimation (Figure 3a). If the density exceeds a predefined threshold inside a cell, the platform triggers an alarm to notify the event's administrators. The parameters that define the area of the event, i.e. dimensions and the size of a cell, are part of the configuration provided by the administrator for the event. The larger the cells, the less precision the algorithm will have detecting the position of the overcrowded zone. However, configuring smaller cells increases the number of calculations required to be performed by the algorithm, as the number of independent zones to be considered also increases.

The algorithm for calculating the device density uses two main working elements: First, the area of the event, which is divided into cells using the configuration defined by the administrators (the number of cells is determined by the size of a single cell). Second, a set of devices, each having a position and a associated radius of error. The error is determined by the applied tracking algorithm (trilateration in this case). This error defines a circle, with its center located at the device's estimated position. The device could be located anywhere inside the circle due to errors inherent to the tracking algorithm.

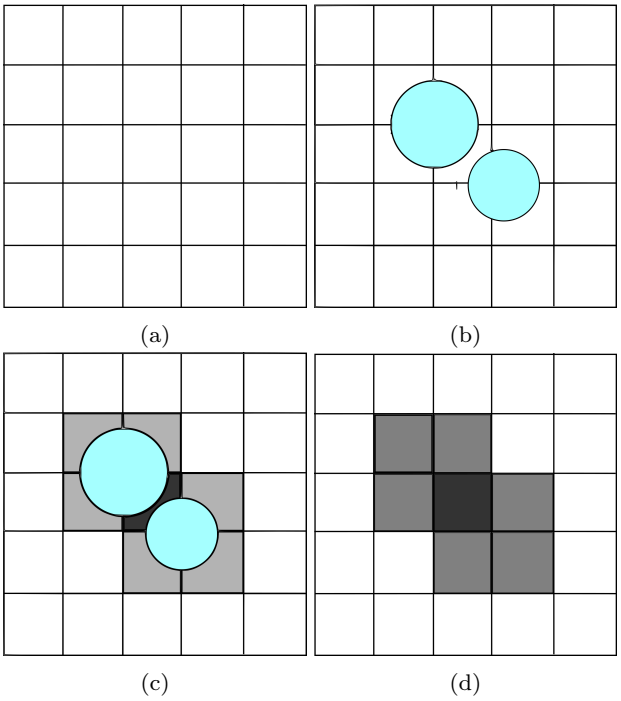

Fig. 3: Density calculation process.

Every time that the position of one or more devices change, due to the reception of new information from the Wi-Fi sensors, the algorithm recalculates the device density for each of the grid's cells. The process to calculate the density works as follows:

– Using the circle defined by each device's positions and its associated error, the algorithm obtains the list of intersecting cells. Each circle can intersect with one or more cells inside the event's area, mean-

ing that the device could be actually located inside any of these cells. Knowing the cells intersecting with each device's circle, we calculate how much of its area is shared with those these cells (Figures 3b and 3c).
Supposing an equal probability distribution of the device's position inside its circle of error, this step enables to obtain how the probability of the device tracking is distributed among the intersecting cells. It is more probable that the user is located in those cells that have a bigger shared area with its error circle, while the total probability always adds to one.
– The probability obtained for each intersection is accumulated in the corresponding cell. As this process is repeated by all the located devices, the result is a grid that contains the average occupation number (number of devices) per cell (Figure 3d).
– Finally, the average density of a cell is obtained by dividing the total occupation of a cell (obtained by adding the contributions of all devices calculated for all the devices in the previous step) by the area of the cell.

This way, the algorithm generates a density matrix that represents in which areas of an event the estimated device density could exceed some predefined threshold.

In order to speed up the computation of the device-cell intersection, R-tree[9] has been used. R-tree is a tree-like data structure that reduces the number of per-cell checks by using a binary search. If checks would be performed in a naive way, each device should be matched against each cell defined in the grid. In that case, the total number of intersection checks would be proportional to the number of cells and devices (N devices x M cells). The usage of R-tree prunes those parts of the area whose cells should not be taken into account, effectively reducing the search space.

3.3 Distributed implementation

We implemented our algorithm in a distributed way using the MapReduce paradigm [5]. MapReduce is a programming model targeting parallel/distributed systems, and amenable to process large datasets. The computation is specified in terms of *map* and *reduce* functions: *map* tasks perform parallel workload, such as filtering, and *reduce* tasks conduct consolidation or summary operations.

We consider two types of compute nodes: master and worker. The master node will be in charge of running sequential and management operations, and worker nodes will run the actual parallel workload.

The implementation of the algorithm has been structured in the following steps, and it has been depicted in Figure 4:

1. The device tracking algorithm computes the positions of the detected devices. This is a sequential operation and its results are scattered to the worker nodes.
2. Using the calculated device positions, each worker computes the device-cell interactions to create each a partial density matrix. This is run as a *map* task.
3. A *reduce* task is executed to merge the partial matrices. Their combination results in the final density, which represents the distribution of the devices across the event area. Due to the nature of MapReduce operations, part of the *Reduce* task is conducted in the worker nodes in a parallel way.

As a final note, the task of computing the device locations was initially developed as a parallel *map* operation. However, initial tests showed that it took less time to do it sequentially than in parallel, due to the data scattering overheads. Thus, it was left as a sequential operation.

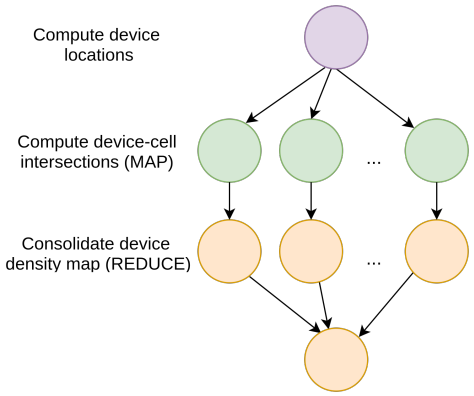

Fig. 4: Workflow of the distributed implementation.

4 The Cloudera framework

In this section we describe the software platform used to gather, process and visualize the data: Cloudera, a ready-to-use distributed computing platform that includes different Apache components on a linux system. The main goal of the Cloudera framework is to provide an out-of-the-box scalable analysis and visualization system, without the need to install and configure several tools required by a Hadoop/Spark environment from scratch. From all the tools provided by Cloudera, we have used the following to implement our solution:

– Flume [10] is a service that provides efficient gathering and aggregation of data. It enables to process a high volume of input data from different sources and write them into multiple custom or predefined consumers using a scalable and fault-tolerant process. Flume is used in the proposed solution as the entry point for all the data captured by the access system and the Wi-Fi sensors.
– Hbase [8] and HDFS are respectively the database and distributed file system used by Hadoop, which allow the storage and management of high volumes of data in a scalable way. In the proposed platform, these tools are used to store the raw data captured during the event and the results of the analytics obtained after processing this data.
– Spark [24] is a framework for large-scale data processing. It has been widely adopted to accomplish *Big Data* processing tasks due to its ease of use, flexibility and wide user community. It is used in our platform to run the distributed implementation of the overcrowding detection algorithm.
– SolR [19] is a search engine prepared to index high volumes of data. It allows to search using text and No-Sql based queries. SolR is used by many large scale companies like Netflix, DuckDuckGo or eBay to provide advanced search capabilities. In our platform it is used to index raw data captured in an event prior to its processing, and to feed the graphs in the Hue visual front-end.
– Hue[2] (Hadoop User Experience) is a Django[3]-based front-end for the visual analysis of data. It is able to load data from common SQL RDBMSs (e.g. MySql or Oracle) or SolR indexing engine, and allows the creation of dynamic dashboards. These dashboards include interactive graphs that ease the analysis of large scale data. It is used in our platform to visualize the monitoring information captured in an event.

5 The Sicafe application

On the top of the Cloudera platform, we developed a software application called Sicafe[4]in charge of the management and visualization of the received information. The use of the Cloudera platform provides us with scalability at platform level, as described in the previous section. We depict in Figure 5 the architecture of Sicafe using the described tools. In short, we coded a personalized Flume configuration and a separate application using some tools inside Cloudera.

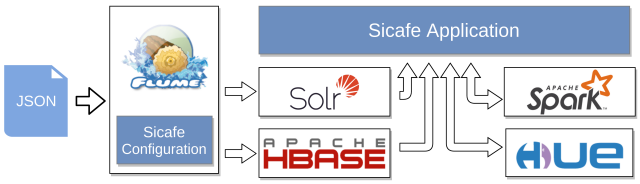

Fig. 5: Sicafe application architecture.

The flume configuration consists of custom *Handler* and *Sink*. The *Handler* is in charge of capturing the JSON files sent to the listening Flume port and creating a new Flume event with each one. The *Sink* takes each of these events, analyzes it, and depending on its type, writes it to a Hbase table or to a SolR index. Both *Handler* and *Sink* are coded in Java and access to Hbase and SolR using the Cloudera API and the SolR Rest API respectively.

Each time a JSON file is received in the Flume listening port, the following events take place:

1. Data is read from the HTTP connection and parsed. Both the wireless frame and access control data will come in JSON format.
2. Data is analyzed and stored in Hbase/SolR.
3. Data is consumed by the Sicafe Application, processed using the overcrowding detection algorithm and visualized using Hue.

We created an application inside Hue that provides the user with a visual way to manage all the configuration and monitoring of an event. This application allows the creation, modification and deletion of events, and the customization of their parameters.

On the data visualization side, the Hue-based application shows two main types of graphs. First ones are interactive dashboards displaying statistics, such as the historical entries or exits through the physical access control. An example of these dashboards is shown in Figure 7. The second graph is a map displaying the position of (1) wireless frame capturing routers in the event, (2) detected wireless devices and (3) areas surpassing a certain occupation threshold, i.e. a potential overcrowding. An example of this map is shown in Figure 6, where (1) is shown as blue map markers, (2) as blue dots and (3) as red circles. Using this web console the administrator of an event can make real time monitoring of what is happening in the area, and once the event has finished, analyze the occupancy data to detect hotspots and trends.

[2] Hue - http://gethue.com
[3] Django - https://www.djangoproject.com
[4] Sicafe stands for *SIstema de Control de Acceso y Flujo a Eventos*, the Spanish for *Event flow and access control system*

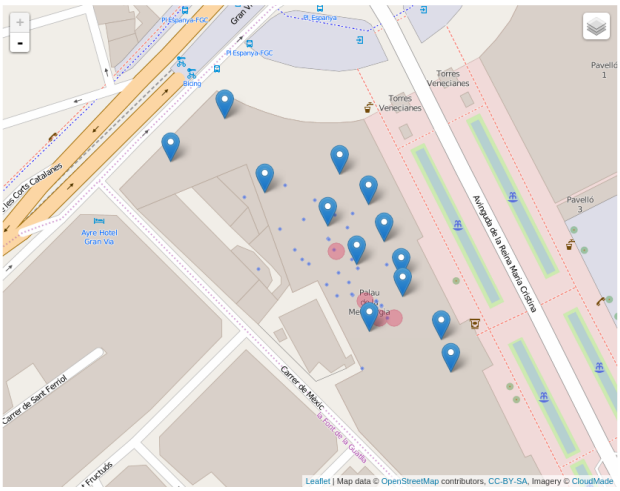

Fig. 6: Example map with attendees and alerts.

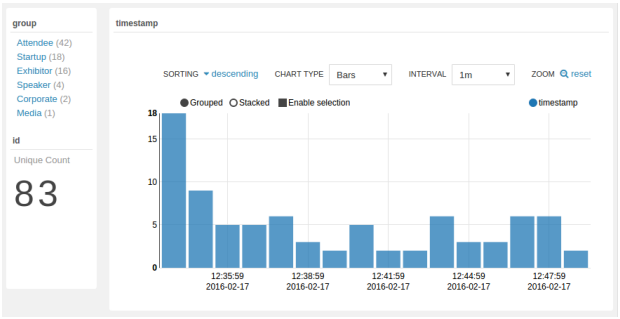

Fig. 7: Example dashboard with entries data.

On the implementation side, the algorithms described in Section 3.2 have been implemented using R-tree[5] and Shapely[6] Python libraries. The Hue-based code is a custom Django application with Hue v3.9, running on Python 2.6.6. The map is shown using Leaflet[7] and the heat-map is drawn with the MaskCanvas[8] plugin, which provides scalable plotting for large amounts of points.

6 Evaluating the accuracy of the algorithm

In this section, we analyze the accuracy of our algorithm with data gathered from a real world scenario. In particular, we use the data collected in *4 Years From Now* (4YFN)[9] business event that took place in Barcelona the 2nd, 3rd and 4th of March 2015. 4YFN is a conference located within the *Mobile World Congress* (MWC)[10], whose aim is to provide a common place for businessmen of start-up enterprises to connect between themselves. The 2015 edition was located in the *Montjuic* building of the Barcelona exhibition center, which has usable space of around 19,000 m^2 space.

The conference area was set-up with the monitoring features described in Section 2. The capture routers were located as depicted in Figure 6, and access control spots were located in the main door of the building. We provide a video on how this system worked in 4YFN[11].

In order to validate the wellness of our device tracking algorithm, we compared the number of localized devices with the number of attendees inside the building (registered by the NFC system) along each day. Results are depicted in Figures 8 and 9 for Monday and Tuesday respectively. Data was collected for Wednesday as well but results were similar and they were left out of the paper due to space constraints.

These results show that the number of detected devices by our algorithm is close to the number of real attendees most of the time. In order to assess the relationship between these two metrics, we calculated the statistical correlation between them using the Pearson correlation coefficient. This coefficient provides the degree of linear dependence between two variables, represented in a range between -1 and 1. In this case, the resulting *r-values* are 0.782 and 0.786 for the data on Monday and Tuesday respectively, which states a high positive correlation between the number of devices and attendees.

After this initial assessment, we must highlight an issue derived from the comparison of detected devices to actual attendees: some attendees might carry more than one device with Wi-Fi switched on and some might carry none. However, we calculated that on the assumption of *one wireless enabled device per assistant*, the trilateration algorithm would detect on average a 73.96% of the attendees counted by the access control system.

In a step further, we note a number of devices detected by the algorithm prior to the entry of the first attendees. To our knowledge, those correspond to the mobile devices of the organizing staff and other Wi-Fi enabled devices that were located in the building for exhibition purposes, such as Smart TVs. We also note some outliers in the graphs depicted as peaks, the most notable one being in the Monday graph at 4:40 PM. These outlier peaks are due to minimal failures in the capturing systems. In the case of the Monday-4:40 PM peak, the routers were unable to capture frames for two minutes, and the frames for the next two minutes were

[5] Rtree - https://pypi.python.org/pypi/Rtree
[6] Shapely - https://pypi.python.org/pypi/Shapely
[7] Leaflet - http://leafletjs.com
[8] MaskCanvas - https://github.com/domoritz/leaflet-maskcanvas
[9] 4YFN - https://4yfn.com
[10] MWC - https://www.mobileworldcongress.com
[11] http://aditium.com/en/success-cases/4yfn-2015-2

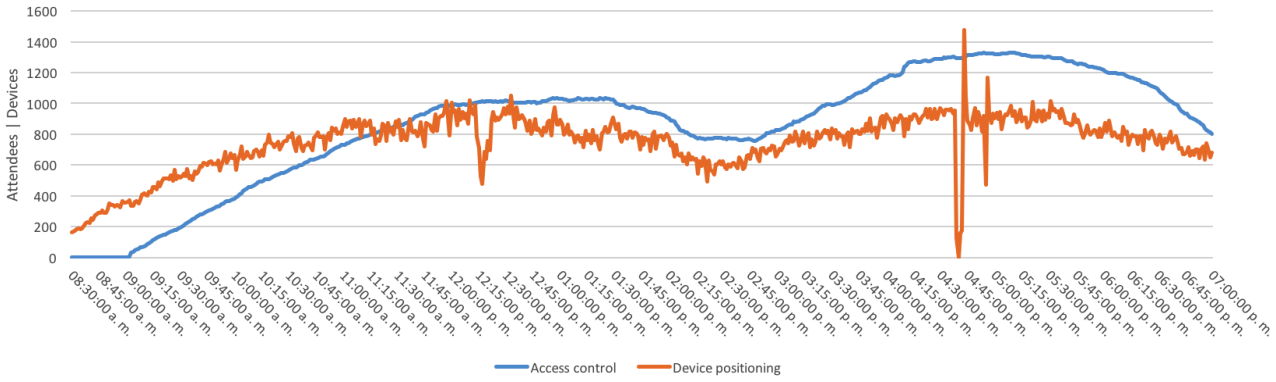

Fig. 8: Comparison of number of attendees to detected wireless devices. Data from 4YFN 2015 Monday.

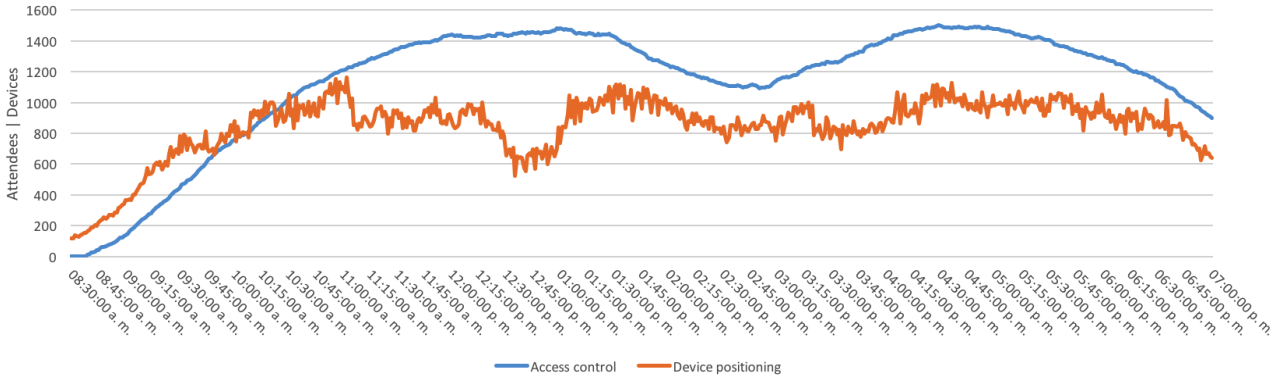

Fig. 9: Comparison of number of attendees to detected wireless devices. Data from 4YFN 2015 Tuesday.

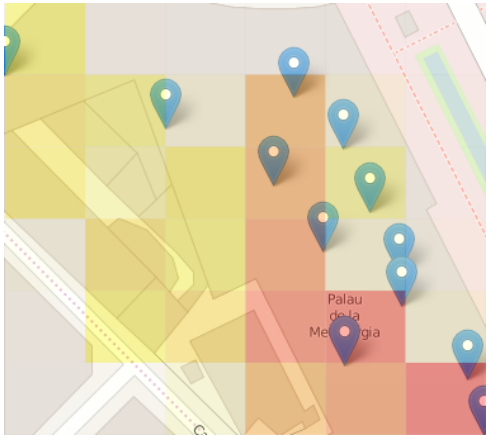

Fig. 10: Average density in 4YFN 2015. Monday.

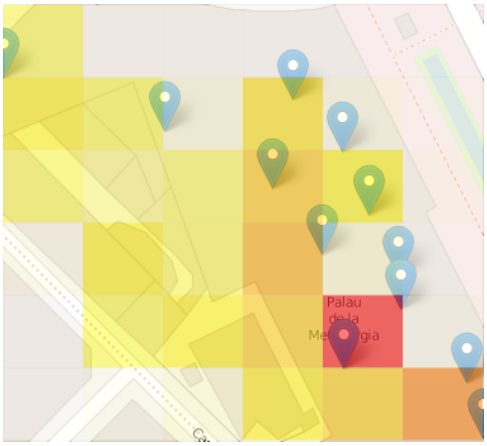

Fig. 11: Average density in 4YFN 2015. Tuesday.

sent in a single JSON to the processing server. After this issue, the system continued working properly.

We must state that, far from obtaining an exact count of the participants, the main aim of the platform is to detect overcrowding. An estimate of this can be done by detecting the coarse movements of the attendees, thus, an exact count is not required. As a proof of concept, we computed the average density distribution of the detected devices in the event for Monday and Tuesday. Results are shown in Figures 10 and 11 as a heatmap were colors closer to red indicate higher density. We observe how both days the most visited places

were located on the most southern part of the building. To our knowledge, this matches the trend of the attendees, as the most popular stands were located in those positions.

7 Evaluating the scalability of the distributed implementation

In this section we carry out a performance evaluation of the distributed implementation of the overcrowding detection algorithm. The algorithm has been implemented using Spark, and we want to find out whether it scales with problem sizes larger than the one presented in the previous section. To this end, we isolated the distributed implementation from the entire platform and ran it as a separate application.

7.1 Settings used in the experiments

For this part of the work we created several synthetic datasets that enabled us to conduct a wider evaluation than the 4YFN 2015 dataset. We created this dataset using the Random Waypoint algorithm [3], a popular mobility model in ad-hoc networking research field that simulates the movement of several entities in a defined space, including location, velocity and acceleration constraints. In this experimentation we set that each entity (wireless device) would move in a speed between 0.1 and 1.4 meters/second, and make random stops of up to 10 seconds. These values represent the standard walking pattern for a human. We used the implementation of this algorithm in the Pymobity[12] library.

There are two main parameters in our evaluation that impact directly the execution time: the dataset size (the number of wireless frames to be processed) and the number of cells in the grid. The larger the number of frames or the smaller the cell size, the larger the problem size.

Regarding the grid size, we defined an area of 138x138 meters (approximately the same area of the Fira Montjuic building where 4YFN 2015 conference took place) and defined different configurations for the evaluation grid: 6x6, 12x12, 24x24, 48x48 and 96x96 cells, which lead to spaces with 36, 144, 576, 2304 and 9216 cells respectively.

Regarding the input size, we created different datasets with 100, 1000, 10000, 100000 and 1000000 devices using the Random Waypoint algorithm. In a real conference, it is unlikely to spot up to 100000 and 1000000 wireless-enabled working devices. We created these large ones to test the scalability of the implementation.

On the hardware side, we have used a single dual-socket server with two Intel Xeon E5640 CPUs, 49 GB of DDR3 RAM memory and 6 TB of storage. Each CPU has 4 cores @ 2.67GHz with Hyper-Threading enabled, leading to a total of 16 physical threads. This server runs Ubuntu Server 16.04 and Apache Spark version 2.0.1. In these tests we configured the server as a single Spark worker with multiple cores.

7.2 Performance assessment

We initially conducted some scalability tests, running the code with the configuration described in the previous section in order to measure its execution time. First, we fixed the grid size to 96x96 cells and variated the number of devices, and then, we fixed the input size to 1000000 devices and variated the grid size. Results for these two tests have been depicted in Figures 12 and 13 respectively.

[12] Pymobility - https://github.com/panisson/pymobility

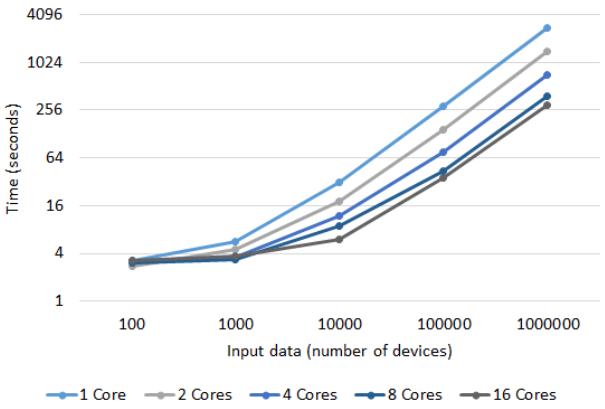

Fig. 12: Execution times for different dataset sizes.

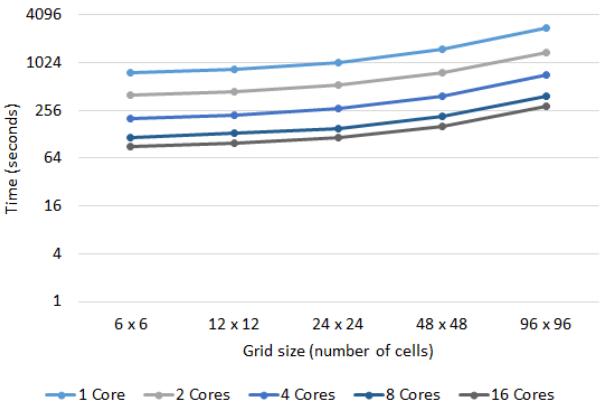

Fig. 13: Execution times for different grid sizes.

Table 1: Shuffle Read and Writes for different input sizes and core configurations. All values are reported in kB.

Input devices	Input data size	1 Core		2 Cores		4 Cores		8 Cores		16 Cores	
		Read	Write	Read	Write	Read	Write	Read	Write	Read	Write
100	7.71	8.84	8.84	8.72	8.72	10.14	10.14	12.49	12.49	16.74	16.74
1000	78.13	44.72	44.72	59.20	59.20	72.29	72.29	79.41	79.41	88.98	88.98
10000	791.02	69.25	69.25	132.22	132.22	238.40	238.40	376.35	376.35	548.13	548.13
100000	8007.81	72.24	72.24	144.35	144.35	286.31	286.31	563.09	563.09	1083.11	1083.11
1000000	81054.69	72.24	72.24	144.47	144.47	288.95	288.95	577.84	577.84	1155.25	1155.25

First, we note that the obtained execution times in these tests are acceptable for the purpose of overcrowding detection. In the case of 10000 or 100000 devices with the largest grid sizes, a delay of several minutes seems reasonable. In addition, we can observe how the algorithm benefits from the use of many cores and that every core configuration scales well. We report in Table 2 the speed-up values for the largest problem size (1000000 devices and 96x96 cells), compared to the execution time using 1 core. We can be satisfied for the values with 2, 4 and 8 cores, where the number of cores nearly matches the obtained speed-up. This is not so for the 16-core execution, but we must note that the extra power comes from the use of 8 HyperThreaded cores, and not 16 full ones.

After this black-box assessment, we go deeper into the parameters of our code that affect performance. Our application is configured as a single Spark job with 2 stages. The first stage runs the compute-intensive part of the algorithm (including *map* and *reduce* operations), and the second one reads the result of the previous one and generates the output matrices. We measured that the first stage covered around 99% of the execution time in every test.

The information exchange between stages is done through Shuffling operations, handled by Spark[13]. The larger the number of bytes in shuffle operations, the more overheads in the application. We show in Table 1 the number of bytes that are read and written in these Shuffle operations for several runs with the largest grid size (96x96 cells) and different input sizes. The values have been taken from the logs generated by Spark.

From the numbers in the table we can see how processing small datasets requires shuffling an amount of data larger than the input. However, this only happens for datasets of few kilobytes. In the case of larger inputs, the shuffled data turns out to be proportionately smaller, which benefits the scalability of the code. In particular, processing the largest dataset (81054.68 kB $\sim$ 79.1 MB) requires shuffling 144,48 kB (72.24 read kB + 72.24 write kB) for 1-core execution, and 2.25 MB (1155.25 read kB + 1155.25 Write kB) for the 16-core execution. These overheads seem acceptable, as the size of shuffled data is in every case less than an order of magnitude smaller compared to the input data.

Table 2: Speed-up values over the 1 core configuration for the largest problem size.

2 Cores	4 Cores	8 Cores	16 Cores
1.99x	3.87x	7.28x	9.58x

Besides the amount of shuffled data, there is another Spark parameter that affects performance: the number of partitions that the data is split into. Spark assigns one or several partitions to each worker core, and the incorrect configuration of this parameter could incur in a performance degradation or a misuse of the computing resources (e.g. less partitions than available cores). In our code, we set this parameter to the default value provided by Spark: a number of partitions equal to the number of cores. This provided a sufficient performance in our tests, and we consider that a deep exploration of the effects of this parameter falls out of the scope of this paper. More on how to tune the partitioning can be found in [13].

Next, we measured the hardware resource consumption of our code in the testing server. To that end we used a combination of collectd[14] to retrieve performance metrics, Prometheus[15] as a lightweight database to store these metrics and Grafana[16] to create a dashboard that visualizes this information. We ran the performance monitoring tools for the tests with different input sizes and core configurations, and all the executions exhibited similar patterns. We report the results just for the largest problem size (1000000 devices input dataset and 96x96 cells grid) in Figures 14, 15, 16 and 17 for CPU usage, memory usage, disk read throughput and write throughput respectively.

In the first of the performance graphs, we can observe that the CPU consumption exhibits normal trends. In the case of memory consumption, we observe how it

[13] http://blog.cloudera.com/blog/2015/03/how-to-tune-your-apache-spark-jobs-part-1

[14] collectd - https://collectd.org

[15] Prometheus - https://prometheus.io

[16] Grafana - http://grafana.org

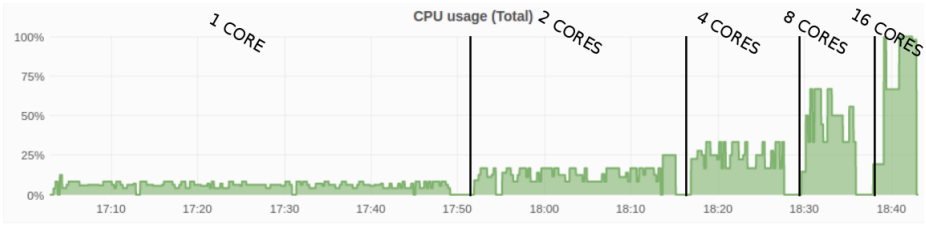

Fig. 14: CPU usage for the largest problem size and different core configurations.

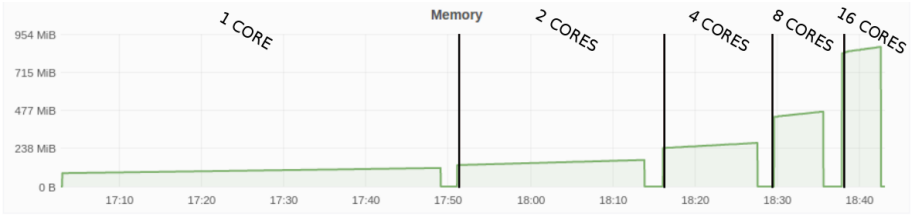

Fig. 15: Memory usage for the largest problem size and different core configurations.

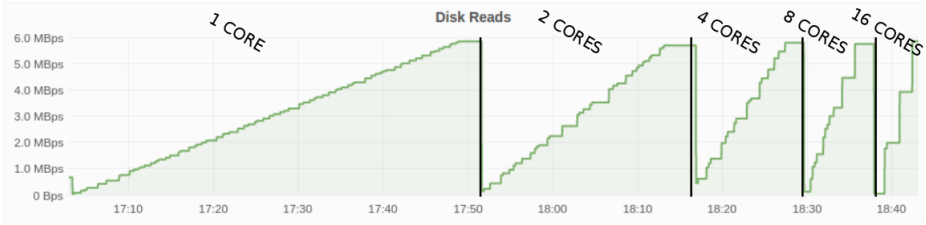

Fig. 16: Disk read throughput for the largest problem size and different core configurations.

increases as the application is running, but within an acceptable range. We see how it gets below 100 MB for the 1-core execution, and it reaches nearly a 1GB of occupied RAM for the 16-core execution. These values seem acceptable for the largest problem size. Regarding the I/O throughput (Figures 16 and 17), we can observe how the pressure on the disk increases almost linearly for all the core configurations, both for read and write operations. In every core configuration, peak values reach 6 Mbps and 2 Mbps for read and write operations respectively. Given that every modern hard drive provides a minimum of 50 Mbps of throughput for read and write tasks [12], we can claim that I/O does not seem to be a bottleneck nor a scalability limiting factor.

As a conclusion for this section, we can state from these tests that our code exhibits good scalable performance. The source code of the distributed implementation is publicly available in [17].

[17] Overcrowding algorithm source code - https://github.com/morelab/overcrowd-simulator

8 Related work

The literature shows many works that have applied Wi-Fi technologies to estimate people movements and density in some area, and in this section we will mention the most relevant ones from our point of view. In [16] authors use a passive Wi-Fi tracking system to estimate people's trajectories by using routers distributed along the area of interest. The users' movements is inferred from their connection to different access points along their movement and not by trying to positioning them in specific location. Barbera et al. [2] apply an approach based on *probe request* gathering, similar to ours, in order to estimate the social relations among participants in a 3-months long campaign. In [18], Schauer et al. use sensing technologies to estimate the crowd density inside some specific areas of an airport and the flux of travelers from one region to the other. However, in this last work the density estimation is performed in two wide areas, without the fine granularity that our approach tries to achieve.

Besides Wi-Fi, other near-range technologies, such as iBeacons [7] based on Bluetooth Low Energy (BLE), enable to improve the location accuracy with other con-

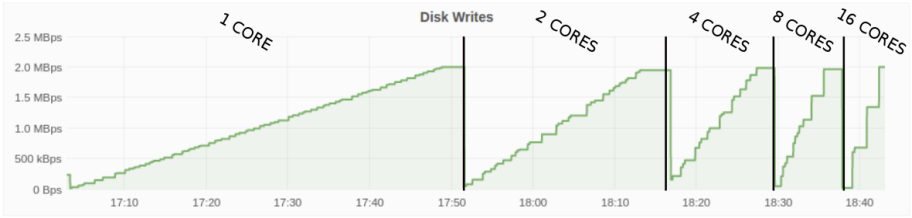

Fig. 17: Disk write throughput for the largest problem size and different core configurations.

straints. For example, [22] describes the usage of Blueetoth technology to track a visitor flow map during its visit to an outdoor fair in Ghent, while [23] also uses Bluetooth technology to estimate crowd density in a large area by using scanning devices distributed across the place of the event. Using Bluetooth for the development of our work would have been possible but the choice of Wi-Fi was done for two main reasons: first, the wider range of space that a single Wi-Fi hotspot covers compared to the range using Blueetooth, and second, the ease to obtain and deploy Wi-Fi enabled routers, compared to Bluetooth ones.

Related to these last works, we would like to point the interested reader to the contribution by Liu et al. [14], were they conduct an excellent survey on the many ways of indoor positioning using different techniques that can be adapted to improve the estimation of attendees location.

In addition, there are some other approaches in the literature that use Computer-based vision to estimate the density of people in some area by obtaining data from CCTV systems [15],[21]. However, these approaches require the installation of costly camera monitoring equipment that is not always available in all the infrastructures. Our approach uses the information provided by more affordable devices that are already part of the communications infrastructure.

On the data processing side, the platform proposed in this work is constructed using a distributed architecture based on Cloudera. We have selected it because of its maturity and our previous expertise on managing big data clusters with this distribution. We found other works in the literature that use the Cloudera toolset with different purposes: Cassavia et al. [4] use the Flume-HBase-SolR-Hue pack to gather Tourism related data and then conduct fast faceted text search and visualize the results using Hue. In [1], the authors state how HDFS-Hue is used to fast index and search information in the University of Maryland Library.

There are also some alternatives to the Cloudera toolset to be considered for data processing. Hortonworks HDP[18] or MapR[19] are ready-to-use Hadoop distributions which provide a full stack of Hadoop tools out of the box, similar to Cloudera. Another interesting software toolset is the Elastic stack[20], which provides the Logstash tool to manage logs (analog to Flume), the Elasticesarch JSON based indexer (analog to SolR) and the tool for creating dashboards, Kibana (analog to Hue).

9 Conclusions and future work

In this paper we have presented an algorithm for overcrowding detection, integrated into a platform aimed at localizing attendees in large indoor events. The localization process is non intrusive for the attendees, as we set a series of wireless routers along the monitored building to capture wireless frames from Wi-Fi enabled devices. These captures are sent to a processing server, which estimates the position of wireless devices using the algorithm described in Section 3. The processing software platform is built on top of the Cloudera platform, which provides a set of scalable tools. On top of these tools, we built an application that displays the processed information in a user friendly way in real time.

We have evaluated the wellness of the overcrowding detection algorithm in two ways: first, we have measured the accuracy of the system with real data gathered from an indoor business event in 2015 that attracted several hundreds of attendees. Some routers were placed along the building and collected information from the wireless devices. We calculated that the relationship between the number of localized devices and the number of attendees is statistically significant, and that under the assumption of one wireless device per attendee, our algorithm locates a 73.96% of the attendees present in the building. We generated a heatmap representing the most crowded points of the event as well. Second, we tested the scalability of the algorithm in an Apache Spark environment. We observed that the

[18] HDP - http://hortonworks.com/hdp
[19] MapR - https://www.mapr.com
[20] Elastic Stack - https://www.elastic.co

code scaled close to linearly, reaching up to 7.28x of speed-up in a 8-core execution with a large dataset. We analyzed the data access patterns of the code and we measured the usage of memory and disk I/O, and found no significant bottleneck. After these tests, we concluded that the code shows a proper scalable behaviour.

Our first step in a future work will be to test other indoor positioning algorithms. Despite its simplicity, the trilateration algorithm gives good performance for our purposes. However, other approaches, such as bilateration, could provide a higher rate of detected devices and more accurate results. In a further step, we would like to test our platform in new indoor events and conduct deeper analytics on collected data. Being built on top of scalable technologies, we could face higher data input rates, process them in real time and try to link them with data from social media, such as Twitter.

Acknowledgements The authors would like to thank Dr. Carlos Pérez-Miguel for his aid in this work and the anonymous reviewers for their insightful comments.

References

1. Abdul Rasheed MM (2013) Fedora commons with apache hadoop: A research study. Code4Lib Journal 22
2. Barbera MV, Epasto A, Mei A, Perta VC, Stefa J (2013) Signals from the crowd: Uncovering social relationships through smartphone probes. In: Proceedings of the 2013 Conference on Internet Measurement Conference, ACM, New York, NY, USA, IMC '13, pp 265–276
3. Bettstetter C, Resta G, Santi P (2003) The node distribution of the random waypoint mobility model for wireless ad hoc networks. IEEE Transactions on Mobile Computing 2(3):257–269
4. Cassavia N, Dicosta P, Masciari E, Sacca D (2015) Improving tourist experience by big data tools. In: 2015 International Conference on High Performance Computing Simulation (HPCS), pp 553–556
5. Dean J, Ghemawat S (2008) Mapreduce: simplified data processing on large clusters. Communications of the ACM 51(1):107–113
6. Fang BT (1986) Trilateration and extension to global positioning system navigation. Journal of Guidance, Control, and Dynamics 9(6):715–717
7. Fard HK, Chen Y, Son KK (2015) Indoor positioning of mobile devices with agile ibeacon deployment. In: 2015 IEEE 28th Canadian Conference on Electrical and Computer Engineering (CCECE), pp 275–279
8. George L (2011) HBase: the definitive guide. O'Reilly Media, Inc.
9. Guttman A (1984) R-trees: A dynamic index structure for spatial searching. In: Proceedings of the 1984 ACM International Conference on Management of Data, ACM, New York, NY, USA, SIGMOD '84, pp 47–57
10. Hoffman S (2013) Apache Flume: Distributed Log Collection for Hadoop. Packt Publishing Ltd
11. International Congress and Convention Association (2015) ICCA Statistics Report: The International Association Meetings Market 2014. Tech. rep., ICCA
12. Kasavajhala V (2011) Solid state drive vs. hard disk drive price and performance study. Tech. rep., Dell PowerVault Storage Systems
13. Li M, Tan J, Wang Y, Zhang L, Salapura V (2015) Sparkbench: A comprehensive benchmarking suite for in memory data analytic platform spark. In: Proceedings of the 12th ACM International Conference on Computing Frontiers, ACM, New York, NY, USA, CF '15, pp 53:1–53:8
14. Liu H, Darabi H, Banerjee P, Liu J (2007) Survey of wireless indoor positioning techniques and systems. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 37(6):1067–1080
15. Lo BPL, Velastin SA (2001) Automatic congestion detection system for underground platforms. In: 2001 International Symposium on Intelligent Multimedia, Video and Speech Processing, pp 158–161
16. Musa ABM, Eriksson J (2012) Tracking unmodified smartphones using wi-fi monitors. In: Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems, ACM, New York, NY, USA, SenSys '12, pp 281–294
17. O'hara B, Petrick A (2005) IEEE 802.11 handbook: a designer's companion. IEEE Standards Association
18. Schauer L, Werner M, Marcus P (2014) Estimating crowd densities and pedestrian flows using wi-fi and bluetooth. In: Proceedings of the 11th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, ICST, Brussels, Belgium, Belgium, MOBIQUITOUS '14, pp 171–177
19. Shahi D (2015) Apache Solr: A Practical Approach to Enterprise Search, Apress, Berkeley, CA
20. Shchekotov M (2014) Indoor localization method based on wi-fi trilateration technique. In: Proceedings of the 16th Conference of Open Innovations Association FRUCT, pp 177–179

21. Velastin SA, Boghossian BA, Lo BPL, Sun J, Vicencio-Silva MA (2005) Prismatica: toward ambient intelligence in public transport environments. IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans 35(1):164–182
22. Versichele M, Neutens T, Delafontaine M, de Weghe NV (2012) The use of bluetooth for analysing spatiotemporal dynamics of human movement at mass events: A case study of the ghent festivities. Applied Geography 32(2):208 – 220
23. Weppner J, Lukowicz P (2013) Bluetooth based collaborative crowd density estimation with mobile phones. In: 2013 IEEE International Conference on Pervasive Computing and Communications (PerCom), pp 193–200
24. Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I (2010) Spark: Cluster computing with working sets. In: Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing, USENIX Association, Berkeley, CA, USA, HotCloud'10, pp 10–10