
ulm university universität
**u**ulm

**Universität Ulm** | 89069 Ulm | Germany

**Fakultät für Ingenieurwissenschaften, Informatik und Psychologie**
Institut für Datenbanken und Informationssysteme

# Path Recognition with DTW in a Distributed Environment

Masterthesis at Ulm University

**Submitted by:**
Yu Tong
yu.tong@uni-ulm.de

**Evaluator:**
Prof. Dr. Manfred Reichert
Dr. Rüdiger Pryss

**Supervisor:**
Burkhard Hoppenstedt

2018

Version February 5, 2018



Satz: PDF-LaTeX 2$_\varepsilon$

# Abstract

The Internet of Things is a concept, where various devices are connected in a network and data is exchanged between them. With the help of Internet of Things applications, it is possible to access sensors remotely to collect data from the physical world. The collected data contains potential knowledge, which could be revealed by applying machine learning techniques. Due to the rapid development of Internet of Things applications, the amount of collected data increases enormously. In order to perform computations on large datasets, distributed computing technologies are used.

Recognizing people's movements is a popular topic in the context of the Internet of Things. Movement patterns are usually sequential and continuous, and can be therefore encoded in the form of time series. Since the Dynamic-Time-Warping (DTW) is an established algorithm for processing time series data, it is chosen as a similarity measure for different movement patterns. Moreover, based on the DTW results, the movements are classified.

In this thesis, we provide an implementation for the recognition of movement patterns. The prototype is built on Apache Spark and Apache Hadoop and uses their distributed computation possibilities. In an experiment, data from probands is collected and evaluated. Finally, the algorithm performance and accuracy are measured.

## Acknowledgments

First I would like to thank my supervisor Burkhard Hoppenstedt, his feedback and support are priceless to me.

Second I would like to thank Prof. Dr. Hans A. Kestler and Dr. Axel Fürstberger. Prof. Dr. Hans A. Kestler gives me a lot of valuable advice for this thesis, and Dr. Axel Fürstberger helps me with establishing the cluster.

Last but not least, I would like to thank my evaluators Prof. Dr. Manfred Reichert and Dr. Rüdiger Pryss.

# Contents

# 1

# Introduction

Nowadays, machines, along with the installed software applications, become an indispensable part of the daily life. Furthermore, these machines become more intelligent by increasing possibilities fueled e.g. by data analytics. They help humans to make decisions by extracting knowledge from raw data, or even take actions themselves according to the learned patterns. For example, a smart building system controls the automatic lighting and air conditioning. Mark Weiser, the man who coined the name Ubiquitous Computing, envisioned, the third wave of computing has begun, in which "many computers serve each person everywhere in the world" [1] and "people will simply use them unconsciously to accomplish everyday tasks" [2].

Due to the development of Internet of Things (IoT), the physical objects are connected to build the network, and data can be exchanged in it. This brings more options for data collection. The potential knowledge that is contained in the collected data can be revealed by applying machine learning techniques. Furthermore, the distributed computing technologies promise to improve the computation performance on the large datasets.

## 1.1 Statement Of The Problem

Recognizing people's movements (path recognition) is a very popular topic in the context of Internet of Things. Previous work revolves around classifying the movement patterns via image processing, or applying classification on accelerometer data collected from mobile devices. In this thesis, we try to find another approach to encode the movement

patterns and apply the classification. Moreover we are interested in the scalability of our approach with a growing dataset.

## 1.2 Objective

The goal of this thesis contains three aspects: first we want to evaluate the feasibility of our approach for path recognition, which encodes the movement patterns (paths) into sequential distance information and applies the classification with the Dynamic-Time-Warping (DTW) algorithm.

Second, we provide an implementation of the whole system. The distance information are collected with the beacons and the computation is performed in the distributed computing environment. We establish a cluster, and install Apache Hadoop and Apache Spark. Apache Hadoop is used for the data storage, while Apache Spark is used for the practical computation.

Last, we carry out the experiments to collect movement data with test persons. The collected data are used for evaluating the algorithm performance and accuracy.

## 1.3 Structure Of The Thesis

Chapter 2 gives an overview of the related work. Chapter 3 introduces the fundamentals of our approach, which describe the details of the DTW algorithm, and the essentials of Apache Spark as well as Apache Hadoop technologies. Chapter 4 provides a sketch of the prototype for validating this recognition approach. Chapter 5 presents the design and implementation of the application. The procedure of the experiment, and analysis results of it are explained in Chapter 6. Chapter 7 elucidates the evaluation on algorithm performance and accuracy. The thesis closes with Chapter 8, which gives a short summary and a brief discussion on the further research issues.

# 2

# Related Work

Apache Hadoop is an open source framework for processing large data sets in a distributed computing environment, which is mostly developed by Yahoo! ([3], [4], [5]). Hadoop consists of HDFS (Hadoop Distributed File System), YARN (Yet Another Resource Negotiator) and the implementation of Map-Reduce paradigm. For this thesis, the most interesting part is HDFS, which makes it possible to store very large files across clusters of computers and is highly fault-tolerant. The HDFS cluster at Yahoo! includes 3500 nodes, each node has 2 cores and 16 GB memory [3]. HDFS is designed to provide high IO performance with large data sets. Yahoo! evaluates the IO performance with several operations such as read, create, delete and so on. Kavulya et al. analyse the 10-months trace data from the M45 cluster to provide a detailed insight of the performance and failure characteristics of Hadoop jobs [6]. Most important aspects they evaluate include resource utilization, failure characteristics, and job characteristics.

Apache Spark is a cluster-computing framework developed by University of California Berkeley several years ago [7]. It realizes the fast computation on very large data by applying in-memory computation and introducing an abstract programming model the RDDs (Resilient Distributed Datasets) [8]. There are various papers evaluating the scalability of Apache Spark. Venkataraman et al. evaluate the scalability by deploying a cluster of 32 r3.xlarge machines on Amazon EC2, each machine has 2 physical cores with 30GB memory and 80GB of SSD storage. Their experiments observe a near liner scaling. Zaharia et al. compare the performance of Spark MLLib and Hadoop Map-Reduce with the Logistic Regression algorithm [9]. They use a cluster of 20 “m1.xlarge” EC2 nodes with 4 cores each and a dataset of 20 GB. The results indicate using Spark MLLib is up to 10x faster. Also they evaluate the broadcast variables for iterative jobs in

Spark by applying the Alternating Least Squares algorithm. The evaluation is applied on a 30-node EC2 cluster and shows that using a broadcast variable improved performance by 2.8x. Ayyalasomayajula compares the performance of Spark MLLib with Mahout on K-Means algorithm in his PHD thesis [10]. He uses the Shark cluster for his experiment, which is at the University of Houston. Shark cluster has 17 SUN X2100 nodes, each node has 2 cores and 2-4 GB memory; and 3 three SUN X2200 nodes with 8 cores and 16 GB memory. His results show that using Spark MLlib is about 4 times faster than using Mahout.

Raspberry Pi has become very popular IoT device recently because it suits the dynamic and radically distributed networked environment very well. Maksimovi et al. evaluates the advantages and disadvantages of Raspberry Pi as IoT device [11], they compare Pi with other IoT devices: Arduino (Uno), BeagleBone Black, Phidgets and Udoo in the aspects: cost and size, power and memory, flexibility, communication, operating system. Vujovi et al. propose a home automation implementation based on Pi as a sensor web node, and demonstrate specifically a fire detection and alarm system by combining temperature sensor with Pi [12]. Ansari et al. appply the motion detection based on the video stream taken by camera which is set on Pi [13]. Chowdhury et al. implement an application of Access control and Home Security using Passive Infra Red sensor on Pi [14]. In this thesis, we tried to build the computing cluster with Raspberry Pi computers. However, due to the insufficient computation power of Pi and the slow speed of the associated network switches, the established cluster is not so suitable for the classification task. Therefore, the classification task is performed by the remote cluster. The Pi machines are only used in the experiment with data collection.

# 3

# Fundamentals

## 3.1 DTW

### 3.1.1 Background

In this thesis, the core algorithm is Dynamic time warping (DTW) [15]. We use this algorithm as the kernel of the classifier for the predication task.

DTW is an algorithm designed to compute the similarity between time series. It was introduced in 1960s and has been proven to be hard to beat and very efficient especially for shifted time series. The DTW algorithm is frequently used in the fields of speech recognition, handwriting recognition and image processing. In recent years, it is commonly applied in various fields, such as data mining. As a matter of fact, it is able to process any data which can be converted into a linear sequence. Prime examples of linear sequences can be found in natural language processing [16].

It is worth noting that the DTW algorithm is applied in Internet of Things (IoT) applications more and more often. Since time series are a common data form in the ubiquitous computing. For instance, [17] presents a health monitoring system called iCarMa, which applies the DTW algorithm for processing photoplethysmogram signals. [18] presents a solution of appliance recognition. In their solution, the classification is performed by using the DTW algorithm with the generated waveform extraction of electronic energy signal.

#### Time Series

Time series are sequences of data points ordered in time. Different from feature vectors [19], time series data is sequential, and usually has different lengths. Common examples of time series data include audio and gesture patterns.

In this thesis, we use the DTW algorithm to recognize motion patterns. We collect information of the motion pattern with a constant sample rate and receive therefore ordered, equidistant data points. Each data point involves the information of the motion pattern at a timestamp, all combined data points can be interpreted as a time series.

#### Time Series Similarity

The naive way to compute the similarity between two time series is to apply the one-to-one mapping. This algorithm maps each data point from the first time series to the corresponding data point in the second time series (cf. Figure 3.1), and computes the similarity of the mapping pair using e.g. the Euclidean distance. Afterwards, the similarity of the two time series can be expressed by combining the Euclidean distances of all mapping pairs to a new feature, e.g. using a sum. This approach is straightforward and can be applied for time series with the same length.

The DTW algorithm can be considered as the extension of the one-to-one mapping. It supports one-to-many mapping and can directly process time series with different lengths (cf. Figure 3.1).

### 3.1.2 Theory

The DTW algorithm computes the similarity between two time series by finding an optimized mapping path, which maps data points from the first time series to those in the second time series. Optimizing along the mapping path means minimizing the accumulated distance of each mapping pair.

For a better illustration, we introduce the algorithm with an example of two concrete time series, further named as $q$ and $c$. These two time series are chosen having different

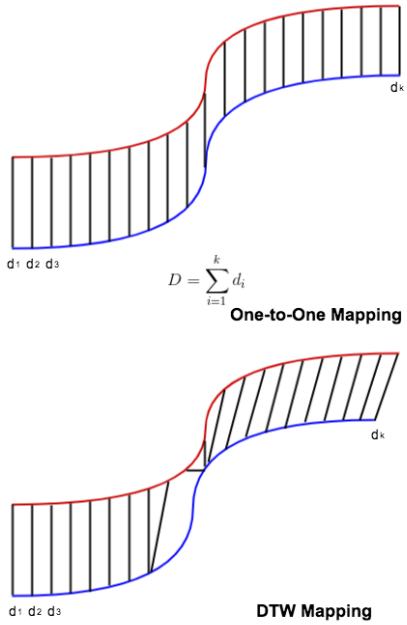


Figure 3.1: One-to-One Mapping vs. DTW Mapping

lengths, in order to show the flexibility of the DTW algorithm. They are plotted in Figure 3.2, where the x-axis shows the number of data points in each time series and the y-axis shows the corresponding value.

In order to find the optimized mapping path, the distance between each data point in the time series is calculated using the Euclidean distance (cf. Equation 3.1). This distance is denoted as local distance.

$$w = \sqrt{(c_i - q_j)^2} \tag{3.1}$$

We can use a matrix of the size $[m \times n]$ to show the corresponding local distances, because there are *m* points in the first time series and *n* points in the second time series. The accumulated distance *r* can be expressed with Equation 3.2, where the *k* is the number of the mapping pairs in the mapping path. Typically, the accumulated distance is also called global distance. Finding the optimized mapping path can be represented through Equation 3.3.

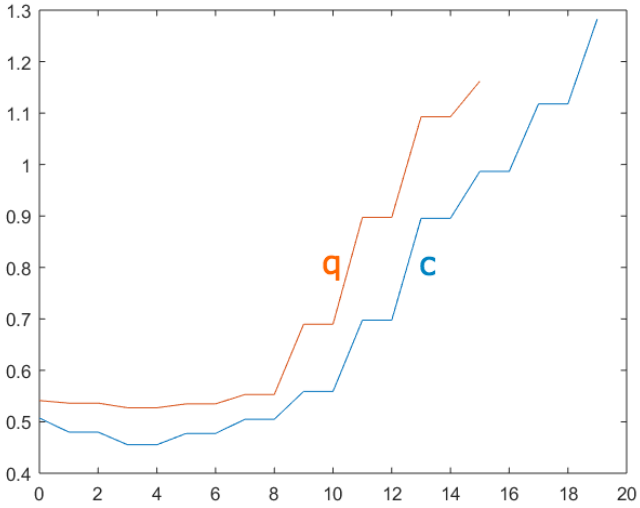


Figure 3.2: Plot of Time Series

$$r(q,c) = \sum_{k=1}^{K} w_k \tag{3.2}$$

$$MIN(r(q,c) = \sum_{k=1}^{K} w_k) \tag{3.3}$$

The global distance matrix can be computed using the local distance matrix. Each point *(i,j)* in the global distance matrix is calculated as the sum of the local distance in this point and the minimal global distance from the previous neighbors. The previous neighbors are defined at the positions:

1. *(i-1,j)*, left neighbor
2. *(i,j-1)*, upper neighbor
3. *(i-1,j-1)*, diagonal neighbor

The computation of the next mapping pair is represented by Equation 3.4. The whole process of finding the DTW distance between *q* and *c* is illustrated by the PseudoCode 1.

$$r(i,j) = w(q_i, c_j) + Min\{r(i-1,j), r(i,j-1), r(i-1,j-1)\} \quad (3.4)$$

**Algorithm 1** DTW Naive Algorithm

```
1:  procedure COMPUTE DTW
2:      declare double DTW[m,n], double localDistance
3:      declare int i, j
4:
5:      for i:=0...m-1 do
6:          for j:=0...n-1 do
7:              localDistance(i, j):=w(q[i], c[j])
8:              DTW[i, j]:=localDistance(i, j)+Min{DTW[i-1, j],DTW[i, j-1],DTW[i-1, j-1]}
9:
10:     reutrn DTW[m-1,n-1]
```

Back to the example with *q* and *c*, we first compute the local distance matrix (cf. Figure 3.3). Figure 3.4 visualizes the local distance matrix with a surface plot in order to give an intuitive demonstration. Speaking in a metaphor, the DTW path can be seen as hiking in a mountain using the least exhausting way.

The mapping is conducted through the whole time series, from the first to the last point in both time series. In our example, the mapping path though the global distance matrix starts at (1,1) and ends at (16, 20). Therefore, we have to compute the global distance for every point. For the point (1,1), the global distance is exactly the local distance, because it has no previous points. For the points in the first row, the global distance can be computed using $w(q_i, c_j) + r(i-1, j)$. It includes the fact that the upper neighbor and the diagonal neighbor are missing. The computation is done analogously for the points in the first column, since they do not have a left neighbor nor a diagonal neighbor. Their global distance is computed as $w(q_i, c_j) + r(i, j-1)$ (cf. Figure 3.5).

For the point *(2,2)*, the global distance is the local distance at this point plus the minimal global distance from its left neighbor *(1,2)*, upper neighbor *(2,1)* and diagonal neighbor *(1,1)*. Since the global distance at *(1,1)* is minimal, the global distance for the point *(2,2)* is computed as $w(q_2, c_2) + r(1,1)$, which is *0.03+0.06=0.09*. The global distance for other points can be computed in the same way. Next, we obtain the global distance

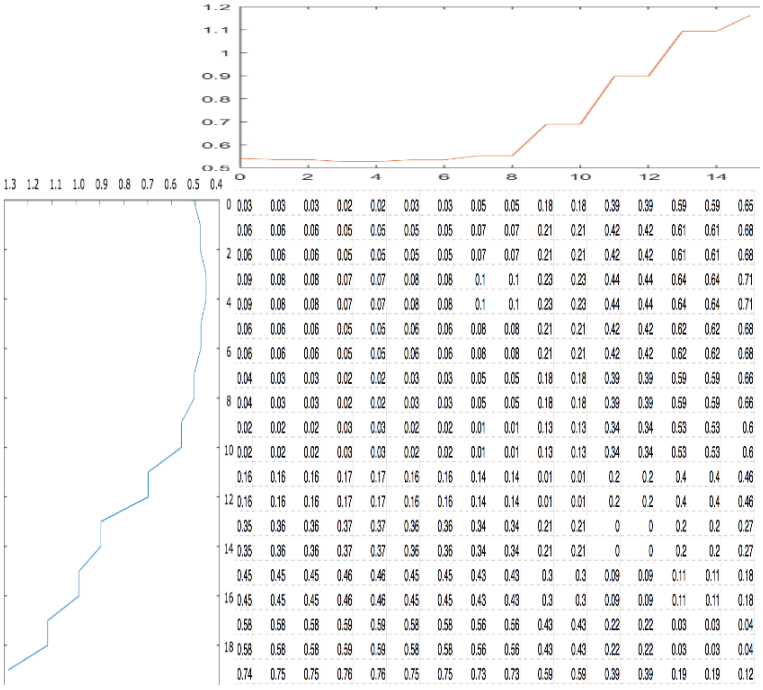

Figure 3.3: Matrix of Local Distance Values

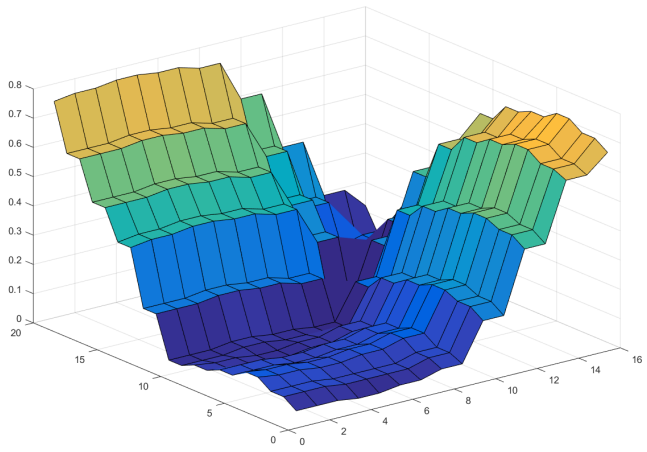

Figure 3.4: Visualization of Local Distance Matrix

matrix (cf. Figure 3.6). It is visualized by a heat map (cf. Figure 3.7) and a surface plot (cf. Figure 3.8).

The similarity between *q* and *c* is exactly the global distance at the end point (16,20) - 0.85, it is also called the DTW distance between the two time series. The mapping path through the global distance matrix is shown in Figure 3.9. From the mapping path, we can obtain the DTW warping path, which characterizes how the two time series are aligned in time. Figure 3.10 illustrates the DTW warping path for *q* and *c* along the mapping path.

### 3.1.3 DTW issues

We described the naive version of the DTW algorithm, which satisfies the properties: *continuity*, *monotonicity*, and *boundary condition*.

#### Continuity

Continuity ensures the mapping pairs along the mapping path are successive. It means the mapping path must be extended one step at a time. Therefore, the increment of *i* and *j* is 1 each time. Continuity can be seen as the core of the DTW algorithm, and all variants of DTW must satisfy it.

#### Monotonicity

Monotonicity guarantees the mapping path is not possible to go back in algorithm iterations. It means along the mapping path, the indices *i* and *j* can increase or stay the same value, but cannot decrease. Regarding one mapping *k* with its global distance $r_k(i, j)$ and the previous mapping *k-1* with the global distance $r_{k-1}(i', j')$, the monotonicity constraints $i - i' \geqslant 0$ and $j - j' \geqslant 0$ must be fulfilled.

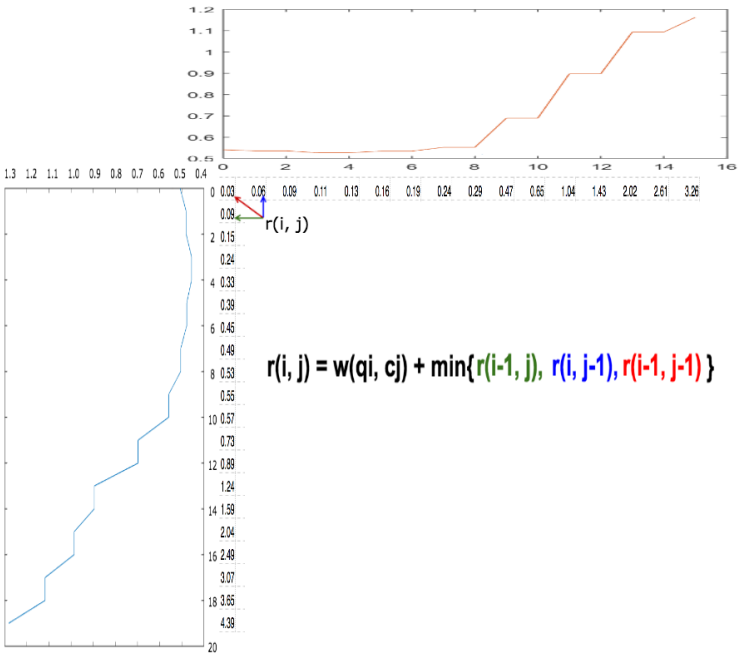


Figure 3.5: Matrix of Global Distance (part 1)

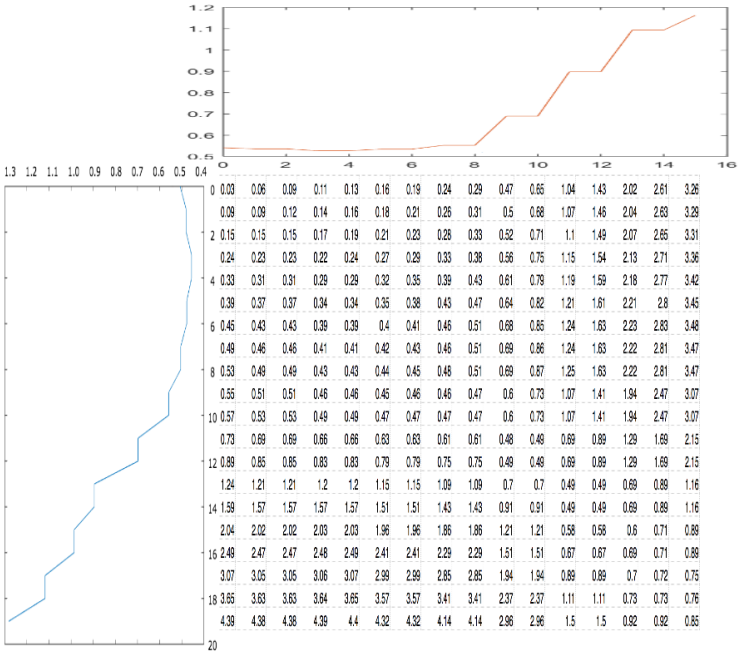

Figure 3.6: Matrix of Global Distance (part 2)

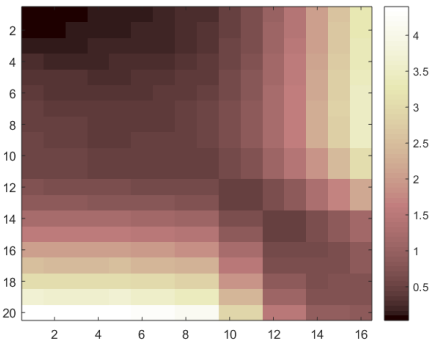


Figure 3.7: HeatMap of Global Distance Matrix

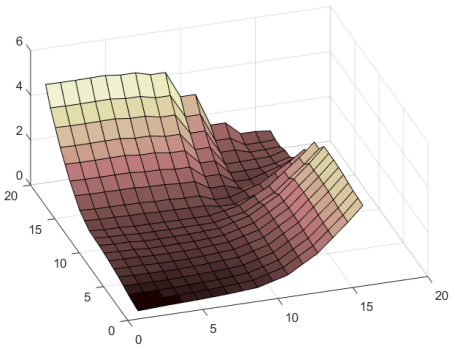


Figure 3.8: Surface plot of Global Distance Matrix

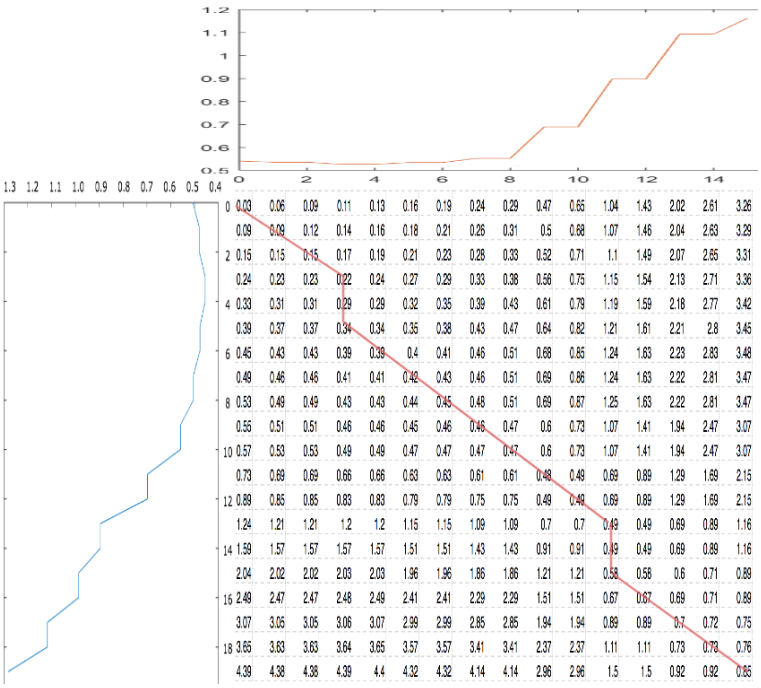


Figure 3.9: Optimal Path in Global Distance Matrix

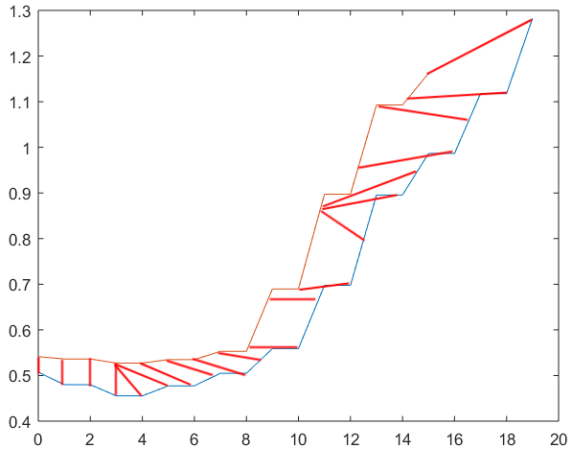


Figure 3.10: Matching Points in Time Series

**Boundary Condition**

Boundary Condition determines the range of the mapping path. When the boundary condition is switched on, the mapping path must start from the first points of both time series and finish at the last points.

### 3.1.4 Classifier

In this thesis, we use the DTW algorithm as the core of the classifier in our classification task. It follows the manner of supervised learning, which teaches the classifier algorithm by preparing labels of all samples in the dataset. In contrast, the unsupervised learning extracts labels and features by asking the algorithms to find the hidden structure of the dataset. Further on, we denote the movements as prototypes.

We combine the DTW algorithm with a decision algorithm for the classification task, e.g. 1- Nearest Neighbour. During the process of the classification, the new samples must be compared to each prototype for computing the DTW distance. The class for the new sample is assigned as the label of the closest neighbour.

The classification process combined with 1-Nearest Neighbour is expressed through the PseudoCode 2.

**Algorithm 2** DTW combined with 1-NN

```
1:  procedure CLASSIFICATION
2:      declare AssociativeArray result
3:      double q[m], double prototype[n,m]
4:
5:      for i:=0...n-1 do
6:          result.add(prototype[i], computeDTW(q, prototype[i]))
7:
8:      sort result by value
9:
10:     reutrn result[0].key
```

## 3.2 Spark

In case of a large number of prototypes and a classification in real time, we need an enormous amount of computations. Therefore, we implement the software archetype with the distributed computing technologies.

As the dataset size becomes fairly large, the computation power of one single computer may not fulfill the job requirements. Distributed computing frameworks aim to connect several machines to horizontally increase the combined computation power.

Apache Spark [7] is one of those frameworks, we choose it as the computing platform in this thesis. Disparate from Hadoop Map-Reduce, Spark applies the computing following the in-memory manner. During the computation, the data is stored in the random access memory of the connected machines as much as possible (cf. Figure 3.11). Spark keeps track of this data between computation operations so that they can be reused immediately.

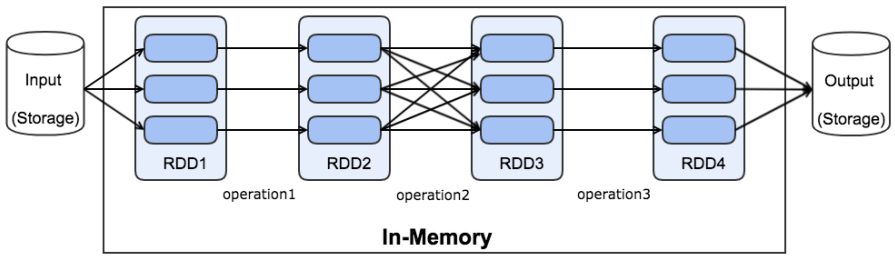


Figure 3.11: Spark In-Memory Computing

### 3.2.1 Spark Application

Spark actualizes in-memory computation by introducing a collection of objects, called Resilient Distributed Datasets (RDD) [9]. RDDs can be computed in many Java Virtual Machines (JVM) concurrently. In a Spark application, RDDs are defined by the Spark driver program.

When a Spark application is submitted to the cluster, the driver program is initialized and runs the user's main function. Each Spark application must have a Spark Context object, which is responsible for splitting the application codes into detailed instructions. The detailed instructions performed on a RDD are called Spark Tasks. One Spark Task is corresponded to a Spark operation in the user's main function. The Spark Tasks are executed by the executors in parallel. One executor will be created on one machine (worker node). Figure 3.12 illustrates this process.

The Spark operations can be further divided into the categories *transformation* and *action*. When a *transformation* is performed, new RDD(s) will be created from the existing one; in addition, perform *actions* will send the computed result back to the driver program. Moreover, all transformations are lazy, which indicates the computation of the transformation will be triggered only when the resulted RDD is computed by an action. When an action is invoked, a Spark Job is created. Table 3.1 summarizes the common Spark operations.

Table 3.1: Common Spark Operations

| Name | Type | Description |
|---|---|---|
| map | trans | apply function on each element in RDD, return a new RDD |
| filter | trans | return a new RDD only containing elements satisfying a filter condition |
| flatMap | trans | apply function on each element in RDD, return a new RDD (may have different size) |
| union | trans | return a new RDD containing elements in either of the RDDs |
| intersection | trans | return a new RDD containing elements in both RDDs |
| groupByKey | trans | apply on (K,V) RDD, group RDD by K, return (K, Iterable<V>) |
| collect | action | return an Array of all elements in RDD to the driver program |
| foreach | action | iterate each element in RDD to apply function |
| reduce | action | aggregate the elements in RDD by applying function |
| count | action | return the number of the elements in RDD |
| first | action | return the first element in RDD |
| takeOrdered | action | return the first n elements in RDD |
| saveAsTextFile | action | write elements in RDD as text file in a file system, e.g. HDFS, local file system |

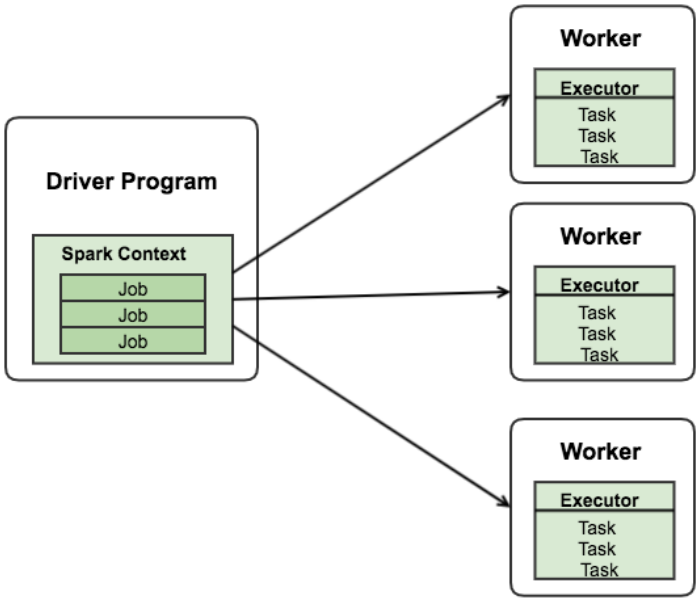


Figure 3.12: Process of Spark Application

### 3.2.2 Spark DAG pattern

Besides the RDD, the high performance of Spark is achieved by introducing the Directed Acyclic Graph pattern (DAG).

For a Spark Job, a directed acyclic graph of consecutive computation units is generated. In this graph, Spark operations are represented by vertices, which edges indicate the processing sequence. Based on the DAG, Spark will generate an optimized execution plan for the Spark Job. Figure 3.13 shows the visualization of the DAG of a Spark Job. This visualization can be accessed through the Spark monitor Web UI when the Spark application is running.

When an action is invoked, the driver program will send the Spark Job to the DAG scheduler. DAG scheduler is in charge of separating the Spark Job into Spark Stages according to the shuffle boundary. The Spark Stage contains the related Spark operations, the operations within one stage can be computed in a pipeline. The separation of the stages is the optimized execution plan. Next, the Spark Stages are sent to a Task scheduler, which is responsible for converting the Spark Stages into Spark Tasks for

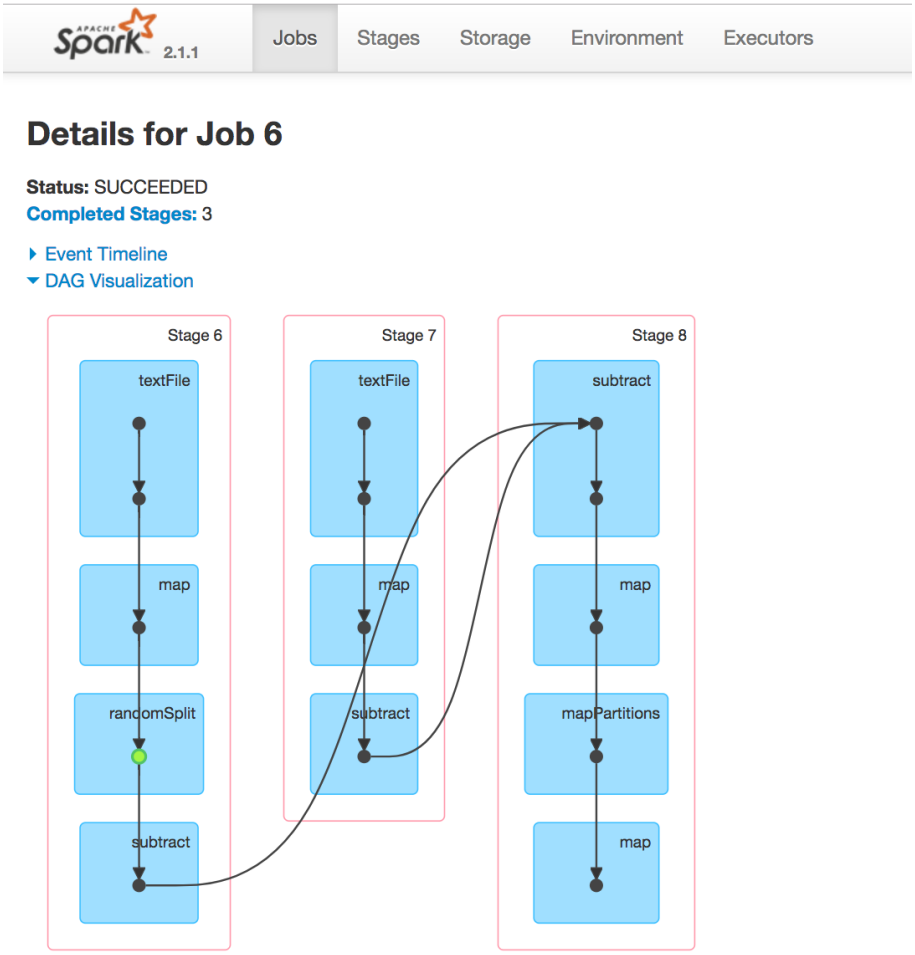


Figure 3.13: Spark Stages

different partitions. The Spark Tasks are then dispatched to the executors by a cluster manager. The detailed process is shown in Figure 3.14.

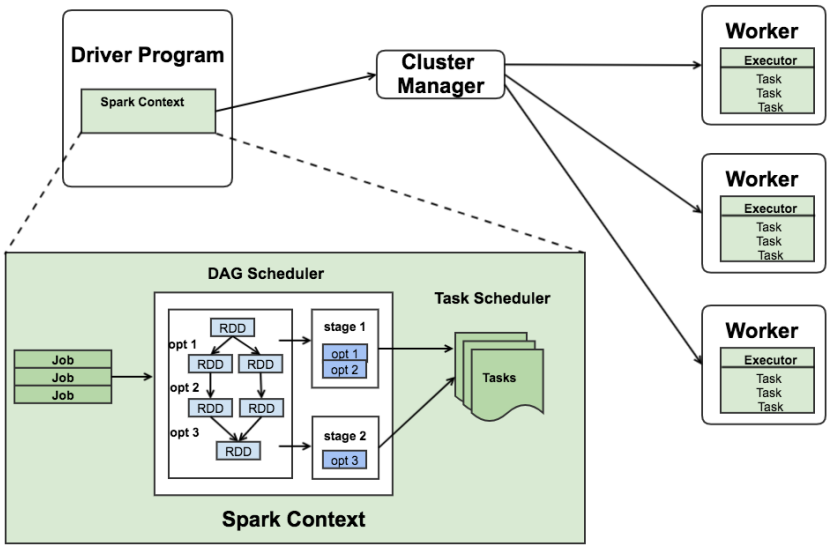


Figure 3.14: Spark Job Scheduling Process

With the help of the DAG pattern, Spark supports the multi-step computation pipelines. This improves the computation performance significantly compared to Map-Reduce paradigma. In addition, it also enhances the fault tolerance since the lost RDDs can be recovered easily with the DAG pattern.

## 3.3 Hadoop

### 3.3.1 DFS

In a distributed environment, a distributed computing platform can only interact with the distributed storage. Therefore, the data should be stored in the distributed file system. In this thesis, we use Hadoop Distributed File System (HDFS) [3] as the distributed storage.

The basic idea of the Distributed File System (DFS) is to create a virtual file system, which enables the user to access the files as if they are stored locally. The virtual file system is deployed in the distributed environment. In other words, the DFS connects several

machines via network, and is able to take the storage power of all the machines. Very important issues related to the DFS are name space, sharing semantic, synchronization, replication and consistency. Generally, there are two patterns for designing the DFS [20]:

- Master-Client pattern
- Client-Client pattern

#### Master-Client pattern

With Master-Client pattern, there is at least one master node and several client nodes. Each master node is responsible for storing the meta-data and maintaining the system. Client nodes are responsible for data storage and processing. When the user reads or writes data, the request is processed by the master node(s) to obtain the location of the file. Afterwards, the user reads data or writes data to client nodes. Examples of this DFS architecture are Google File System (GFS) and Hadoop Distributed File System (HDFS).

#### Client-Client pattern

With Client-Client pattern, there is no global information in the DFS. All nodes are equal and each of them is responsible for processing data and maintaining the system. Examples of this DFS pattern includes Network File System (NFS) and Andrew File System (AFS).

### 3.3.2 HDFS

HDFS is the implementation of DFS from Apache Hadoop. It represents the storage layer in the Hadoop Eco-System, and integrates with other distributed computing system as well, for instance Apache Spark and Apache Storm.

HDFS follows the Master-Client pattern, there is one single master node (NameNode) and several client nodes (DataNode) [21]. Files are stored as chunks of size up to 64 MB, while large files are split into several chunks, each chunk is identified by a unique

128 bits chunk-id. The NameNode stores the locations of the chunks, and the chunks are stored on the DataNodes. The workflow of reading and writing data to HDFS is shown in Figure 3.15 and Figure 3.16.

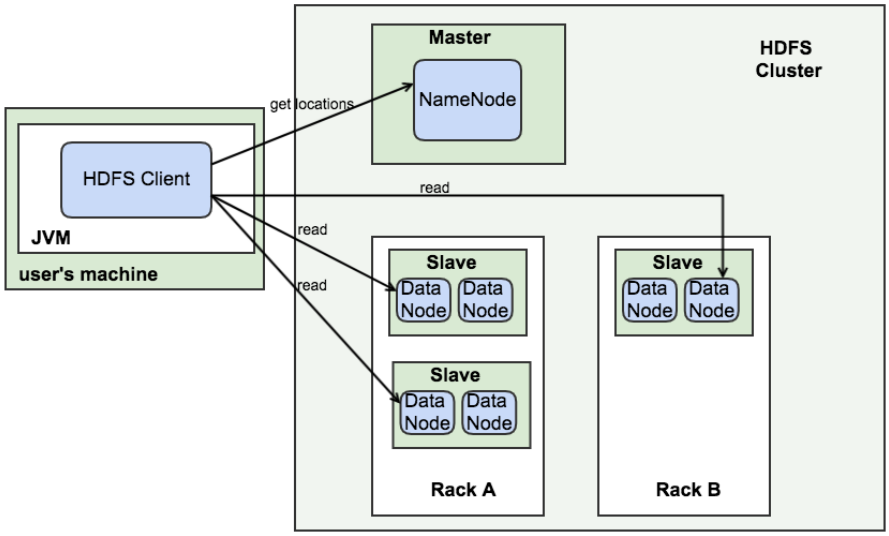


Figure 3.15: Read from HDFS

The sharing semantic of Hadoop is *Write-Once-Read-Many*. After the creation of a file, the writing should be done immediately. Finally, the file is closed. Normally, the file should not be changed except appending new content. This semantic simplifies coherency issues and increase the readability of data. It is worth mentioning, that with the previous versions of Hadoop appending was even not possible. Once the file is created and saved in HDFS, it stays immutable.

The robustness of HDFS is guaranteed, as it can detect disk failures automatically. Each DataNode sends heartbeat messages to the NameNode periodically. If one DataNode is not seen for a certain period of time, it will be marked as dead. No further data will be written to it, reading requests will not be sent to it neither. On the other hand, data which is stored on the dead nodes is not available to HDFS any more. This would be one reason that the distributed file system should keep multiple replications of the data.

Replication is the approach to improve the fault tolerance of the distributed file system. It increases the availability, yet brings risk to consistency. All the files and chunks in HDFS

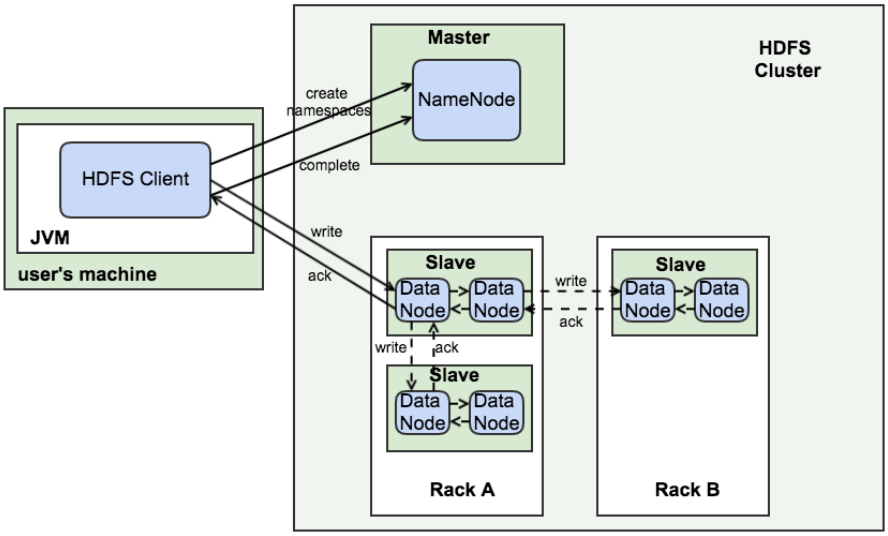


Figure 3.16: Write to HDFS

can be replicated by configuring the replication factor. By default, the replication factor is three. It means the HDFS stores three copies of the same data following the default replication strategy:

1. One copy is stored on the same DataNode;
2. One copy is stored on a different DataNode but on the same rack;
3. One copy is stored on a DataNode on a different rack.

In summary, HDFS is designed to store large files and for streaming data access. However, it is not designed for the single update to the files nor storing many small files.

### 3.3.3 Hadoop Map-Reduce

The basic computation paradigma of Hadoop is Map-Reduce pattern [22].

The Map-Reduce pattern involves the phases map and reduce. In the map phase, an one-to-one function is applied on each key-value pair. In the reduce phase, a many-to-one function is applied, which iterates the key-value pairs under the same key and apply some aggregation to produce one result for each key.

In this thesis, we do not use the Map-Reduce programming pattern. We include this short introduction of Map-Reduce pattern because the Spark DAG pattern is considered as the generalization of Map-Reduce pattern and they are compared to each other very often.

# 4

# Prototype

## 4.1 Use Case

The aim of this thesis is to classify different motion patterns in an indoor environment. In our use case, the motion patterns refer to the walking paths, the indoor environment refers an experiment area inside a room. On one side, as people walk the same path, the encoded data share the potential similarities. On the contrary, different people have different walking behaviors, such as different walking speeds and habits. We would like to see among those walking behaviors, whether the DTW algorithm is able to classify the walking paths correctly.

The walking paths are encoded with distance information. To obtain the distance information, we place several sensors in the experiment area. During the experiment, the sensors collect the distance information between the experiment object and themselves periodically. One distance value is a data point in the corresponding time series, we will gather one time series for each sensor. Figure 4.1 illustrates the procedure of data encoding. The experiment object walks in the experiment area shown as the light blue area. The blue squares represent the sensors. In our setting, there are five sensors. The blue line shows a certain path. The length of black dashed line represents the distance value between the experiment object and one sensor. This would be one data point in the encoded time series shown at the bottom.

We have designed six different walking paths in a particular experiment area (cf. Figure 4.2, Figure 4.3 and Figure 4.4). The green boxes in the diagrams represent some objects, for instance they could be some machines in the practical scenario. The small grey squares represent the sensors placed around the objects or on the objects. The

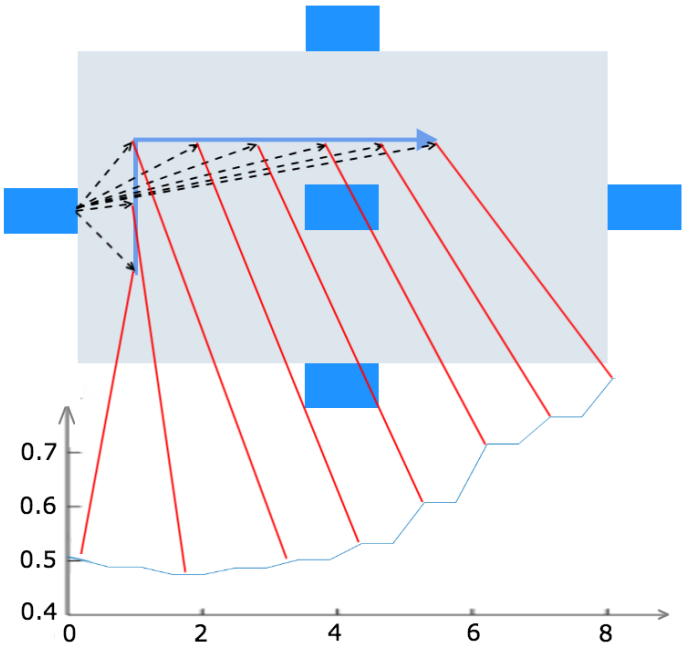


Figure 4.1: General Process of Use Case

yellow lines are the designed walking paths. The arrowheads indicate the directions of the paths. The yellow star-sign marks the start and end point of each designed path. The labels of the paths are shown by the text in the diagrams respectively. For one experiment, the experiment object starts walking at the star-sign, along one designed path in the arrowhead direction, and finishes the path when he returns to the star-sign. The data collected withing one experiment consist of one sample.

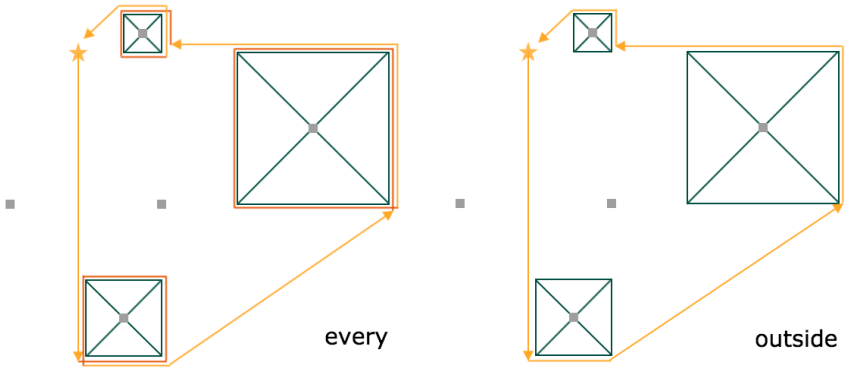


Figure 4.2: Designed Paths (a)

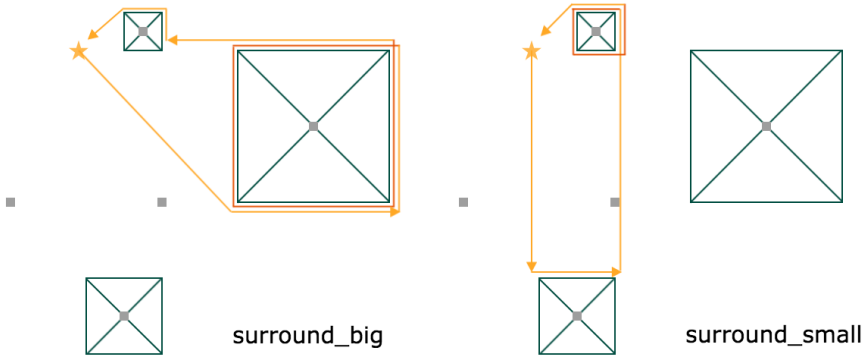


Figure 4.3: Designed Paths (b)

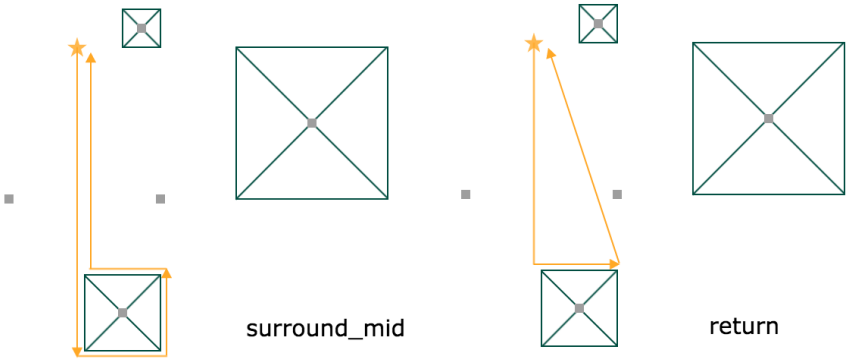


Figure 4.4: Designed Paths (c)

## 4.2 GUI Simulator

Before conducting the experiment in the concrete scene, we implement a GUI program as prototype. With the help of the prototype, we can simulate the whole experiment process. By evaluating the simulated data, we can assess whether the DTW algorithm is appropriate for our approach, which encodes the motion patterns with the distance information.

With the prototype, user can move the mouse to emulate walking paths. As the user moves the mouse, we track the position of the mouse, it can be denoted as a two dimensional coordinate $(x_m, y_m)$. Since the simulated sensors are the fixed points on the screen, we can obtain the coordinates $(x_{sensor}, y_{sensor})$ of them easily. The relative distance can be acquired by calculating the Euclidean distance using the coordinates (cf. Equation 4.1).

$$Distance = \sqrt{(x_m - x_{sensor})^2 + (y_m - y_{sensor})^2} \tag{4.1}$$

Figure 4.5 shows the screenshot of the prototype when it launches. The light blue area is the experiment area, the layout is quite similar as the diagrams of the designed paths. The green boxes are the concrete objects, the small blue squares are the sensors, the star-sign is the starting point. We take the position of the upper-left angles of the small blue squares (shown as red dots in Figure 4.6) for computing the relative distance.

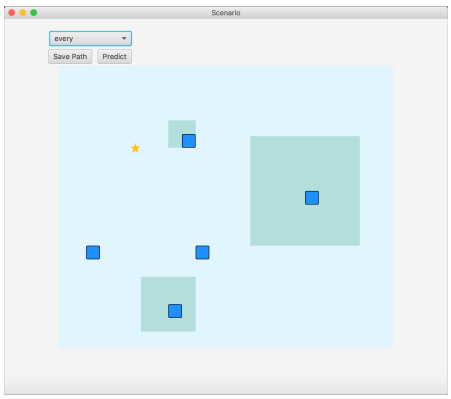


Figure 4.5: Prototype - Screenshot

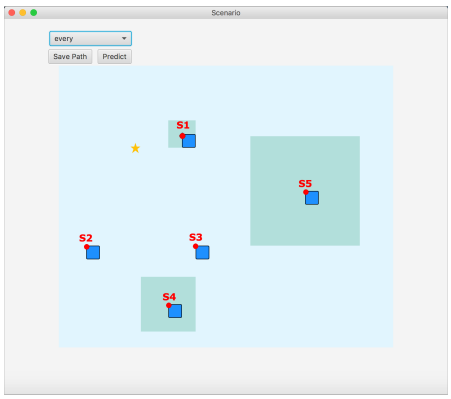


Figure 4.6: Prototype - Referenced

When the mouse enters the experiment area, the tracking of the mouse positions starts automatically. This refers to the start of recording one sample. The current coordinate of the mouse will be shown above the experiment area (cf. Figure 4.7). During the recording, the program computes the distances between the mouse and each simulated sensors every second. The computed distances information are kept in five arrays, one for each simulated sensor. When the mouse exists the experiment area, the tracking ends. This indicates that the recording of one sample is finished.

We include a combo list in the prototype, which contains the labels of all designed paths. Before recording, the user has to select the label of the path to be recorded (cf. Figure 4.8). After the recording, the user has two options:

1. Save the recorded data

2. Apply prediction for the recorded data

By clicking the button *Save Path*, the selected label and the arrays of computed distances will be encoded as a string. The encoded string represents one sample and it will be appended to a local file. By clicking the button *Predict*, the program will apply the classification. All samples stored in the local file are used as the prototypes in the training set. The newly recorded data is taken as the query sample. After the classification, the predicted label will be displayed above the experiment area.

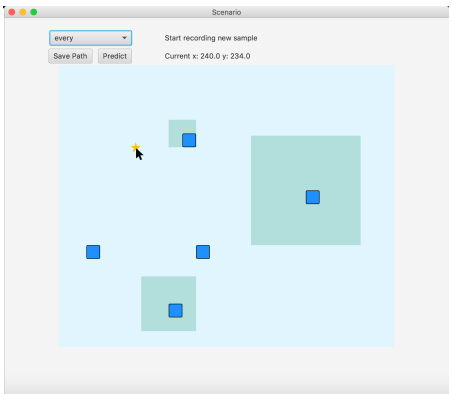


Figure 4.7: Prototype - Recording

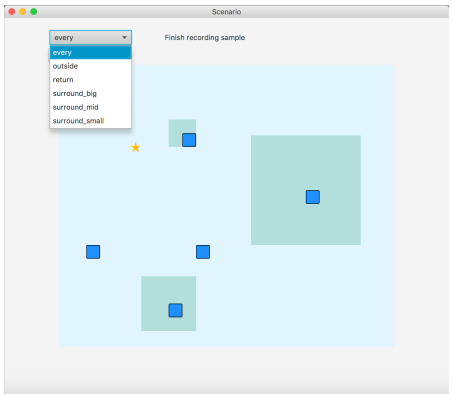


Figure 4.8: Prototype - Labels

## 4.3 Classification Validation on Simulated data

### 4.3.1 General

In this section, we apply the classification validation to evaluate the performance of our predictive model.

We recorded 10 samples for each designed path using the prototype. Our dataset contains 60 samples in total. To ensure that the comparison is only done in the walking paths part, the mouse enters and exits the experiment area approximately at the same point.

We choose the Leave-One-Out strategy as the validation technique [23], because the size of our dataset is not large. By Leave-One-Out, the classification is performed for every sample. In each iteration, the current sample is the testing set and the other samples consist of the training set. The accuracy of the classification is calculated as $\frac{N_{correct\_prediction}}{N_{all\_prediction}}$.

We conduct the classification validation on the whole data, as well as on the data collected by each sensor. Therefore, we can be conscious of the prediction accuracy of each sensor.

### 4.3.2 Decision Algorithms

We take the DTW algorithm as the kernel of the classifier, and combine it with several decision algorithms [24]:

- 1-Nearest Neighbour
- K-Nearest Neighbour
- Weighted K-Nearest Neighbour

#### 1-Nearest Neighbour

1-NN (1-Nearest Neighbour) is described with the PseudoCode 2 (cf. Section 3.1.4). The prediction is the label of the training sample with the minimal DTW distance.

#### k-Nearest Neighbour

k-NN (k-Nearest Neighbour) is based on 1-NN. Unlike 1-NN, this decision algorithm keeps k training samples with the minimal DTW distances. Then the prediction is decided by the majority vote of the k samples' labels. In our evaluation, we set k to the value of five.

#### Weighted k-Nearest Neighbour

Weighted k-NN (weighted k-Nearest Neighbour) is an extension of k-NN. This decision algorithm takes the ordering of the nearest neighbor into consideration as well. It is based on the concept that, the closer the neighbor is, the higher its weight should be. In our evaluation, we set k=5. The weight for each neighbor is calculated by Equation 4.2.

$$-0.9 * \frac{order}{k} + 1 \tag{4.2}$$

### 4.3.3 Results

Table 4.1 shows the accuracy and run time for each classification validation. The results indicate the accuracy of our predictive model is pretty optimized. The accuracy of the prediction is over 95% using all sensors. The accuracy results are above 90% using a single sensor.

Table 4.1: Classification Results on Simulated Data

| | 1-NN | | k-NN | | weighted k-NN | |
|---|---|---|---|---|---|---|
| all sensors | 98,33% | 26,9 s | 95% | 22,9 s | 96,67% | 20,6 s |
| s1 | 90% | 11,7 s | 93,33% | 10,1 s | 91,67% | 10,4 s |
| s2 | 100% | 12,5 s | 95% | 10,9 s | 98,33% | 10,9 s |
| s3 | 95% | 11,8 s | 90% | 11,1 s | 93,33% | 11,0 s |
| s4 | 96,67% | 13,3 s | 93,33% | 10,6 s | 91,67% | 11,9 s |
| s5 | 96,67% | 12,3 s | 91,67% | 10,5 s | 91,67% | 11,1 s |

From the result we can see that, it is meaningful to place multiple sensors in the experiment and collect distance information from different angles.

# 5 Application

## 5.1 General

In order to carry out the experiment in the concrete scene, we need a software system providing the following functions:

- Collect distance data with sensors
- Save the collected data in a cluster
- Apply the prediction in the cluster

Figure 5.1 describes the tasks that the users can complete with the software system. There is only one user role in our system and two main tasks: *save data* and *apply prediction*. The task *record distance information* is the basic step of both main tasks.

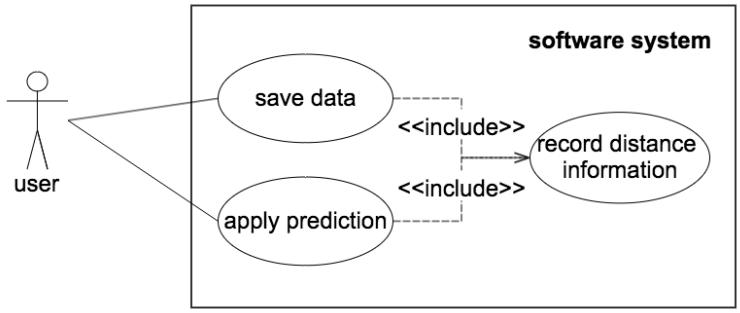


Figure 5.1: Use Cases

According to the use cases, we implement a software archetype that consists of several components. Those components are *Collection Component*, *Proxy Component*, *Save*

*Component* and *Prediction Component* (cf. Figure 5.2). Each component is responsible for different jobs:

- *Collection Component*: It is responsible for recording samples and interacting with sensors. It is the interface of the system for users.
- *Proxy Component*: It is in charge of connecting other components together.
- *Save Component*: It takes care of saving recorded samples in HDFS.
- *Prediction Component*: It applies the prediction in the cluster.

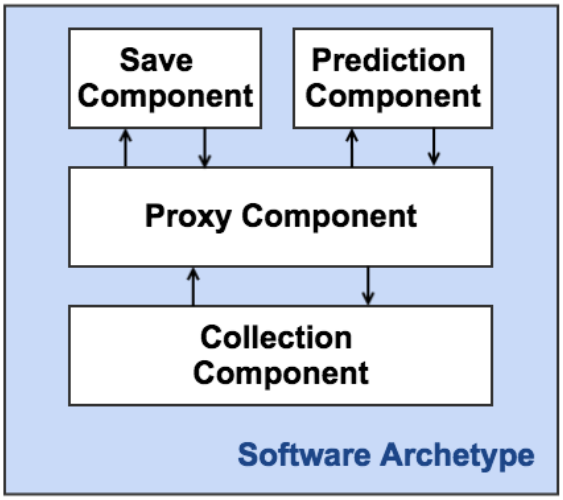


Figure 5.2: Overview of Components in the Archetype

In the following sections, we describe the functional requirements, the general design of the software archetype, and the implementation along with the selected technologies.

## 5.2 Functional Requirements

**FR1**

**Title:** Save recorded motion pattern in HDFS

**Description:**

The user should be able to use a mobile application to record the motion pattern when he walks. After recording, the user can save the encoded sample in a file in HDFS.

**Process:** The mobile application is the implementation of the *Collection Component*. The process of FR1 includes (cf. Figure 5.3):

1. User collects distance information for recording sample using *Collection Component*
2. *Collection Component* encodes the recorded data as a sample
3. *Collection Component* sends the encoded sample to *Proxy Component*
4. *Proxy Component* asks *Save Component* to append the encoded sample in HDFS
5. *Proxy Component* sends the response information to the *Collection Component*
6. *Collection Component* shows the response information to the user

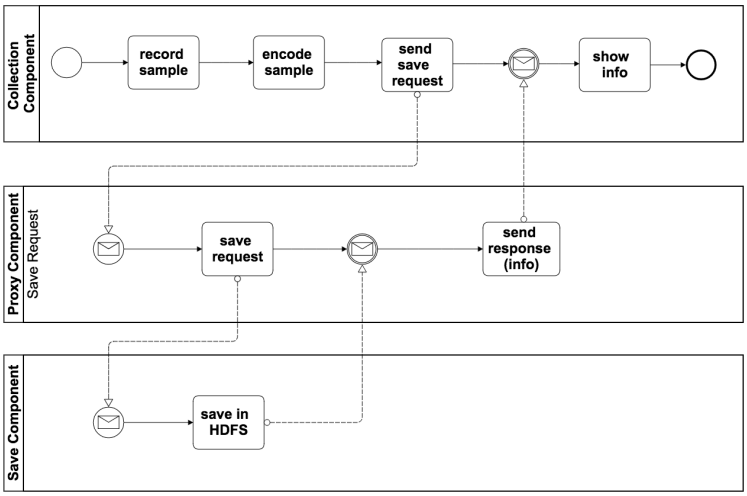


Figure 5.3: Process of FR1

**FR2**

**Title:** Apply prediction on recorded motion pattern

**Description:**

The user should be able to use a mobile application to record the motion pattern when he walks. Afterwards, the user can request the system to apply the prediction on the recorded sample. When the prediction is finished, the predicted label should be displayed to the user.

**Process:** The mobile application is the implementation of the *Collection Component*. The process of FR2 includes (cf. Figure 5.4):

1. User collects distance information for recording sample using *Collection Component*
2. *Collection Component* encodes distance information as a testing sample
3. *Collection Component* sends the testing sample to the *Proxy Component*
4. *Proxy Component* prepares the arguments
5. *Proxy Component* submits the Spark Application with the arguments to *Prediction Component*
6. *Prediction Component* executes the Spark application in the cluster
7. *Prediction Component* returns the predicted label to *Proxy Component*
8. *Proxy Component* sends the response with the predicted label to *Collection Component*
9. *Collection Component* displays the result to the user

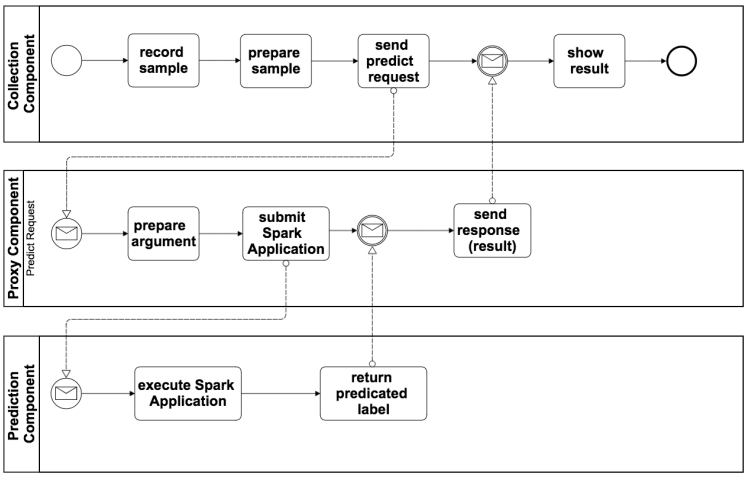


Figure 5.4: Process of FR2

## 5.3 System Design

The *Collection Component* is designed as a mobile application, which can be installed on a mobile device e.g. a small phone or a tablet. It is the interface of the system, which takes the user's instructions and displays the information to the user. During the recording, the user holds the mobile device and walks. We choose the iBeacon device as the sensor to collect the distance information.

The *Proxy Component* plays the role of the middleware, which interacts between the mobile application and the cluster. It is in charge of converting the encoded sample to the Spark Application's argument and submitting the Spark Application to the cluster. After the prediction, *Proxy Component* returns the predicted result to the mobile application. We choose HTTP as the protocol to transfer data among the mobile application, proxy application, and the cluster. Therefore, the *Proxy Component* is a proxy web server, which is listening to the mobile application's requests. In addition, the *Save Component* is designed as a module in the proxy web server.

The cluster is built on several machines and is remote. The Hadoop and Spark are installed on it and both started. Because the cluster is built with the internal network, the Hadoop and Spark APIs cannot be accessed outside the cluster. This internal network has a certain entry point for external accesses. Therefore, the *Proxy Component*

is required for the archetype, and it is deployed on the entry point machine. The *Prediction Component* is a Spark Application, which applies the prediction using the Spark framework. This Spark Application is installed on the cluster as well.

The general architecture of the software archetype is shown in Figure 5.5.

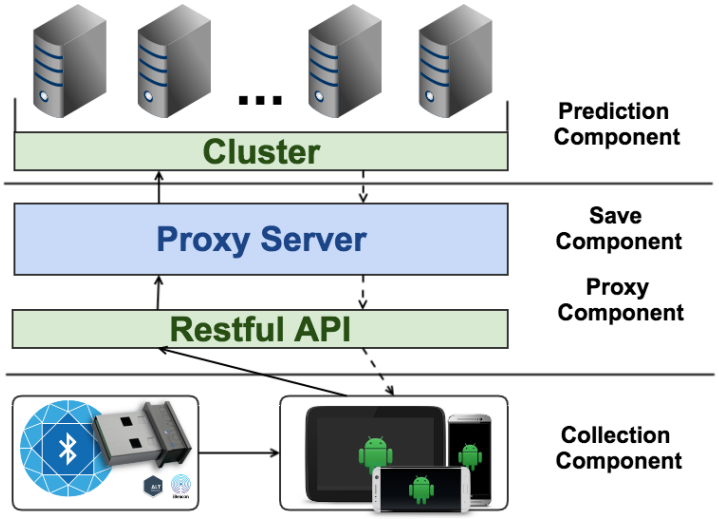


Figure 5.5: Architecture of Archetype

## 5.4 Implementation

### 5.4.1 Mobile Application

The mobile application is responsible for the user interaction, and communication with the proxy web server by sending the HTTP requests. We implement it as an Android App. In this App, transmitting the HTTP data is implemented using the Google library Volley. Besides, the mobile application is responsible for collaboration with iBeacon devices for collecting distance information. This is implemented using AltBeacon.

### Volley

Volley is an open source library developed by Google, which aims to make the networking easier and faster for Android applications. Volley brings the following advantages:

1. It executes requests asynchronously and will not block the main thread.
2. It affords a robust Request queue and schedules the requests automatically.
3. It provides high level API for Restful HTTP requests.
4. It offers rich customization for retry, backoff, etc.
5. It is extensible for custom request and response handling.

Listing 5.1 shows an example of sending a HTTP GET request to "http://github.com/google/volley". When it receives the response, it prints the response body in the console.

```
1  RequestQueue queue = Volley.newRequestQueue(this);
2  String url ="http://github.com/google/volley";
3  StringRequest stringRequest = new
       StringRequest(Request.Method.GET, url,
4          new Response.Listener<String>() {
5              @Override
6              public void onResponse(String response) {
7                  Log.i("volley", response);
8              }
9          },
10         new Response.ErrorListener() {
11             @Override
12             public void onErrorResponse(VolleyError error) {
13                 Log.i("volley", "Error with sending request");
14             }
15 });
16 queue.add(stringRequest);
```

Listing 5.1: Example of Volley Simple Reques

### iBeacon & AltBeacon

iBeacon is a protocol developed by Apple and was introduced in 2013. It is based on the *Bluetooth Low Energy (BLE)* technique. The compatible hardware transmitters of iBeacon are called beacons. The beacon can broadcast its identifier (iBeacon packet), so that the nearby BLE devices are able to receive it. iBeacon is originally designed for detecting user's close proximity. The reasons we choose iBeacon as the sensor include:

1. It broadcasts the iBeacon packet constantly, so that we can generate the time series for DTW computation.
2. Distance information can be interpreted using the iBeacon signal power.
3. Distance information can be interpreted for different users individually.
4. It is easy to interact with the mobile applications.
5. iBeacon devices are affordable.

Android does not have the native support for iBeacon. Therefore, we use the AltBeacon to communicate with beacons. AltBeacon is an open source library developed by Radius Network. It can detect beacons which meet open AltBeacon standard and provides the common API for interacting with beacons.

The standard iBeacon packet contains information about the signal power, therefore, the *Received Signal Strength Indication (RSSI)* can be measured at the receive devices. Based on the RSSI, the proximate relative distance can be computed.

AltBeacon prepares a known table for *distance - RSSI value*, and applies a power regression to find the best matching curve. This curve can be expressed through Equation 5.1, where $r$ is the measured RSSI value; $t$ is the reference RSSI value at the distance of one meter; $A$, $B$ and $C$ are the constant weights learned by the power regression.

$$d = A \cdot \left(\frac{r}{t}\right)^{B} + C \tag{5.1}$$

Listing A.1 shows an example of scanning the beacons nearby. In this example, we only scan the AltBeaon packets and iBeacon packets (cf. Line 10 to Line 13). Since the person is moving during the experiment, the beacons should be scanned periodically for computing the current distances. Therefore, we scan the beacons every 200 miliseconds, each round the scanning lasts 800 miliseconds. In other words, the scanning is completed every second (cf. Line 19 to Line 20). Each round, the example program prints the number of the beacons nearby and the distance to every beacon in the console (cf. Line 25 to Line 28).

#### Android App

We implement the Android App containing two *Activities*: MainActivity and SettingActivity.

The MainActivity is responsible for scanning beacons, displaying distance information and offering the operations related to recording samples. We use the *SectionsPagerAdapter* in the MainActivity, so that the user can slide the screen to see different views. There are three views in our Android App (cf. Figure 5.6):

1. *Scan View* shows scanning results.
2. *Operation View* shows the operations for recording samples.
3. *Distance View* shows distance information.

When the application launches, it starts scanning beacons automatically, the nearby beacons will be shown in the *Scan View*. The user can click the checkbox to add the beacon into monitor list, because there might be other BLE devices in the environment. When the user click again the selected checkbox, the beacon is removed from the monitor list. The distance information of the beacons in the monitor list will be shown in the *Distance View*.

The *Operation View* shows the user operations, which are very similar to the prototype (cf. Section 4.2). It contains a combo list for different labels (cf. Figure 5.7) and a small

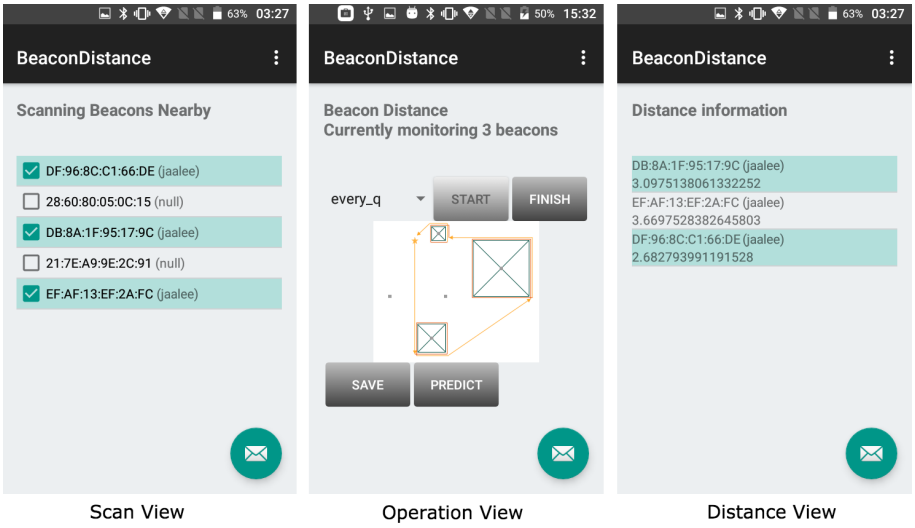


Figure 5.6: MainActivity - Views

diagram shows the path for the selected label. There are two buttons above the diagram: *START* for starting the recording and *FINISH* for stopping the recording. There are two buttons below the diagram: *SAVE* for sending the save request, *PREDICT* for sending the prediction request.

The SettingActivity provides the possibility to modify the App features and preferences. The settings menu will be displayed when *Settings* is clicked in the App (cf. Figure 5.8). There are two categorizes of settings in the App: *General* and *Advance* (cf. Figure 5.9). In the *General* Setting, user can:

- Modify the URI of the HDFS cluster
- Change the name of the file in HDFS

In the *Advance* Setting, user can:

- Enable and disable the automatic scanning for beacons
- Change the accuracy of the encoded sample

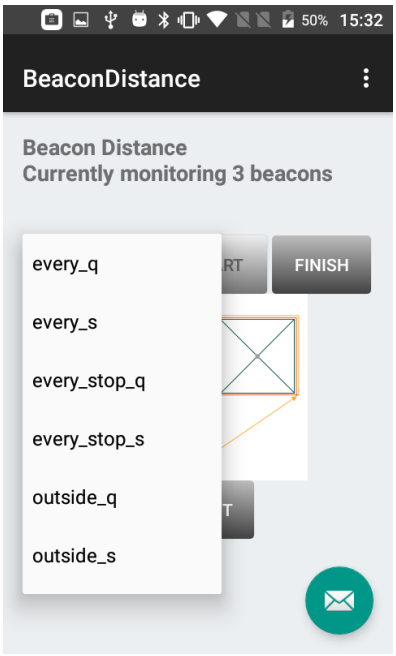


Figure 5.7: Operation View

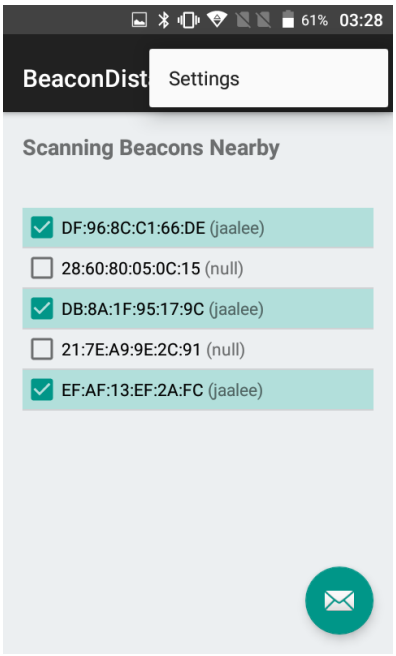


Figure 5.8: Settings Menu

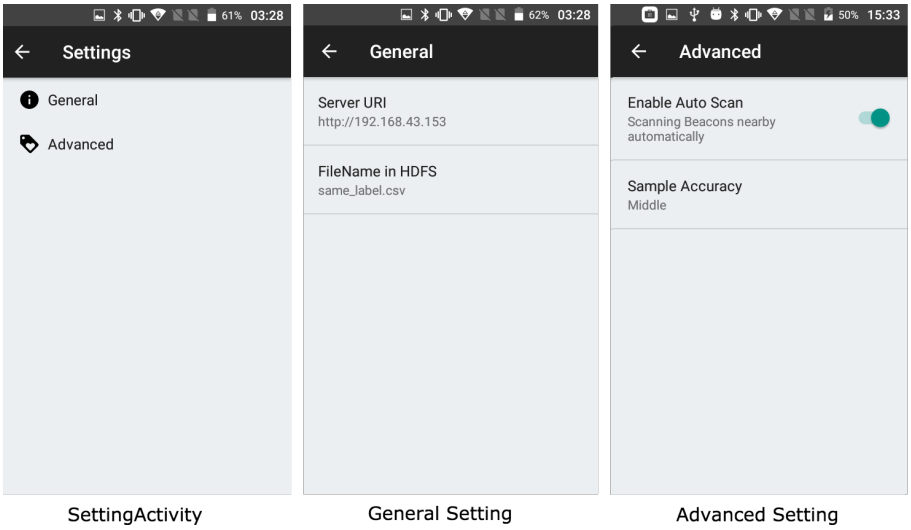


Figure 5.9: SettingActivity - Views

### 5.4.2 Proxy Web Server

The proxy web server is in charge of listening for the HTTP requests from the mobile application and invokes different actions for different requests. For the prediction request, it sends the HTTP request to Spark Server in the cluster for submitting the Spark Application. In this case, the proxy web server acts as the HTTP Client to the Spark Server. For the save request, it appends the encoded sample in HDFS. This is done by using HDFS Client API. We choose Vert.x to implement the web server.

#### Vert.x

Vert.x is an open resouece toolkit for building a reactive and non-blocking application, which runs in a JVM. It was originally developed by Tim Fox and now is maintained by the Eclipse Foundation. Vert.x provides an event-based programming model, therefore all I/O requests are treated as events. It uses the Event Loop method [25], which checks if there is a new event in the infinite loop. When a new event arrives, Vert.x will call an event handler to process it asynchronously, which never blocks the threads. Hence, it enables the application to handle a lot of concurrency by using a small number of threads. Vert.x application consists of Verticles, which are chunks of codes. Verticles can communicate with each other by sending and receiving message via the Event Bus.

Vert.x Web is a module based on Vert.x core, which is designed for building the web applications easily and fast. Listing 5.2 shows an example of building a simple web application. It deploys a Verticle called *ServerVerticle* (cf. Line 4 to 5). This Verticle creates a HTTP Web Server listening at the port *1234* (cf. Line 11 to 18). The *router* object takes care of routing HTTP requests according to the relative URI. For instance, the requests with the relative URI */home* will be routed to the method *home(RoutingContext context)*. This method writes *"This is homepage"* to the response body (cf. Line 20 to 22).

```
1 public class ServerVerticle extends AbstractVerticle {
2    private Logger logger =
        LoggerFactory.getLogger(ServerVerticle.class.getName());
```

```
3     public static void main(String[] args) {
4        Vertx vertx = Vertx.vertx();
5        vertx.deployVerticle(ServerVerticle.class.getName());
6     }
7     @Override
8     public void start() throws Exception {
9        Router router = Router.router(vertx);
10       router.get("/home").handler(this::home);
11       vertx.createHttpServer().requestHandler(router::accept)
12       .listen(1234, hr -> {
13          if (hr.succeeded()) {
14             logger.info("Web Server is listening at 1234");
15          } else {
16             logger.error(hr.cause());
17          }
18       });
19    }
20    public void home(RoutingContext context) {
21       context.response().end("This is homepage");
22    }
23 }
```

Listing 5.2: Example of Vert.x WebServer

Furthermore, Vert.x WebClient helps to do the HTTP request/response interactions conveniently. It provides rich features with sending HTTP request, e.g. request parameters, encoding and decoding Json. Listing 5.3 shows an example of sending a HTTP GET request to "http://vertx.io/" and prints the response body when the response arrives.

```
1     public void client(){
2        WebClient client=WebClient.create(vertx);
3        client.getAbs("http://vertx.io/").send(hr->{
```

```
4           logger.info(hr.result().bodyAsString());
5       });
6   }
```

Listing 5.3: Example of Vert.x WebClient

### HDFS Client

In order to interact with HDFS, we can use the File System shells, which include various commands. The syntax of the most commands are similar to the corresponding Unix commands. Listing 5.4 includes two HDFS commands. The first command returns a list of the direct children of the HDFS home directory; the second command copies the local file *myfile.txt* to the HDFS home directory.

```
1  $ hdfs dfs -ls
2  $ hdfs dfs -put myfile.txt
```

Listing 5.4: HDFS Commands

Additionally, HDFS provides the Client API, so that the user can interact with HDFS programmatically. Listing 5.5 illustrates a function to create a file in the HDFS home directory.

```
1   public static void create(String fileName) throws
        IOException {
2       FileSystem hdfs = null;
3       try {
4           Configuration conf = new Configuration();
5           hdfs = FileSystem.get(new URI("hdfs://master:9000"),
                conf);
6           String path = "hdfs://master:9000/user/tongyu/" +
                fileName;
7           Path file = new Path(path);
```

```
8           if (hdfs.exists(file)) {
9              System.out.println("File already exits");
10             return;
11          }
12          hdfs.create(file);
13          System.out.println("File created successfully");
14       } catch (Exception e) {
15          e.printStackTrace();
16       } finally {
17          hdfs.close();
18       }
19    }
```

Listing 5.5: HDFS Client: Create file

### 5.4.3 Spark Application

The Spark Application applies the prediction to the testing sample, which is passed as one of the application argument. When the prediction is completed, the Spark Application is also responsible for sending the predicated label to the web proxy server by HTTP request. The Spark Application is built as a Jar file and is stored in the local file system on all worker nodes in the cluster. Listing 5.6 shows the main steps of the DTW Spark Application, the detailed implementation of the functions are left out for the sake of brevity.

```
1  package com.tongyu
2  object DTW_1NN {
3    type pairRDD = (String, Array[Array[Double]])
4    def main(args: Array[String]): Unit = {
5       /*
6       * args(0) - testing sample
7       * args(1) - token
```

```
8       * args(2) - spark master uri
9       * args(3) - hdfs file path
10      * args(4) - web host
11      */
12    val conf = new
          SparkConf().setAppName("DTW_Cluster").setMaster(args(2))
13    val sc = new SparkContext(conf)
14
15    val dataset = readAndParse(args(3), sc)
16    val testSample = parse(args(0))
17    val prediction = runDTW(newSample, dataset)
18    new WebClient().response(args(4), prediction + "," +
          args(1))
19  }
20
21  // detailed implementation
22  // ...
23 }
```

Listing 5.6: HDFS Client: Create file

### Spark REST

Spark REST is comprised in the Spark framework, it offers the possibility to interact with Spark besides the command line. Spark REST contains the Restful API to execute the operations on the Spark Applications, for instance, get the status of a Spark Application and kill a Spark Application.

In this thesis, we use Spark REST for submitting Spark Application programmatically. By the time of submitting the Spark Application, several parameters must be passed to the Spark Server. Those parameters include the location of the Jar file, the full name of main class, the URI of Spark master, etc. Besides those parameters, it is also possible to pass

the application arguments. The application arguments will be the *args: Array[String]* of the main function. Listing 5.7 shows an example of submitting Spark Application using cURL command line.

```
curl -X POST http://master:6066/v1/submissions/create --header
   "Content-Type:application/json;charset=UTF-8" --data '{\
   "action" : "CreateSubmissionRequest",\
   "appArgs" : [ "someAppArgument" ],\
   "appResource" : "file:/home/yu/SparkDTW.jar",\
   "clientSparkVersion" : "1.5.0",\
   "environmentVariables" : {\
   "SPARK_ENV_LOADED" : "1"\
   },\
   "mainClass" : "com.tongyu.DTW_1NN",\
   "sparkProperties" : {\
       "spark.jars" : "file:/home/yu/SparkDTW.jar",\
       "spark.driver.supervise" : "false",\
       "spark.app.name" : "DTW",\
       "spark.eventLog.enabled": "true",\
       "spark.submit.deployMode" : "cluster",\
       "spark.master" : "master:6066"\
    }\
  }'
```

Listing 5.7: Spark REST - Submit Application

# 6

# Real Data

## 6.1 Experiment

To evaluate our approach in the practical scene, we invite 22 people for collecting data. During the experiment, we do not keep track of any personal information, only the distance information for the motion patterns are recorded.

We build the experiment area in accordance with designed prototype (cf. Figure 6.1). The experiment consists of two sessions since the invited people belong to two different groups. Moreover, the two sessions are carried out in two different places. We endeavour to build the experiment areas for the two sessions to be the same, but it is impossible to reproduce them identically. Due to different sizes and layouts of the experiment rooms, there is a certain error between the two experiment areas. For this reason, we do not merge the collected data of two sessions together. Instead, we apply the classification validation on each of them separately.

As discussed in section 4.1, we have designed six different paths. In the experiment, we add three extra: *every_stop*, *outside_stop* and *surround_big_stop*(cf. Figure 6.2, Figure 6.3 and Figure 6.4). The three new paths can be considered as the variants of *every*, *outside* and *surround_big*, which include several pauses during the walk. The test person is required to take a short stay around the certain positions shown as red S in the corresponding diagrams. The idea of having those variants is to challenge the DTW algorithm. We would like to see whether the DTW algorithm is capable to distinguish very similar paths.

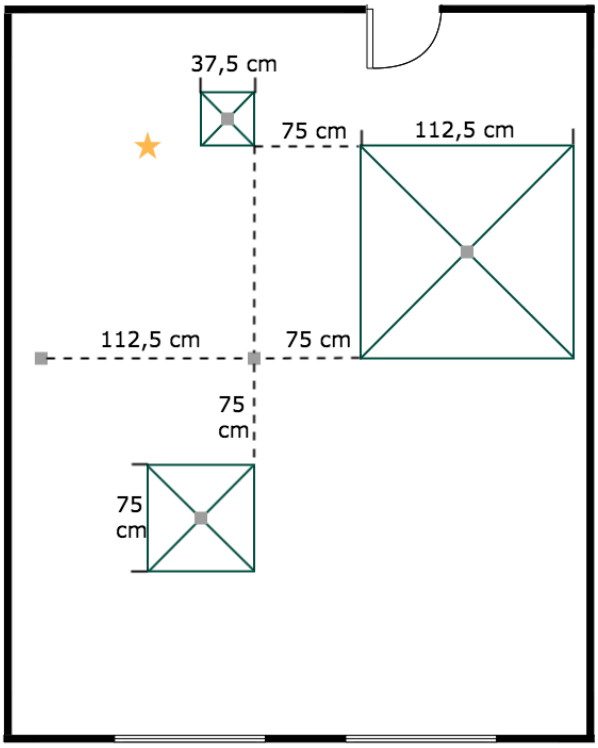


Figure 6.1: Experiment area

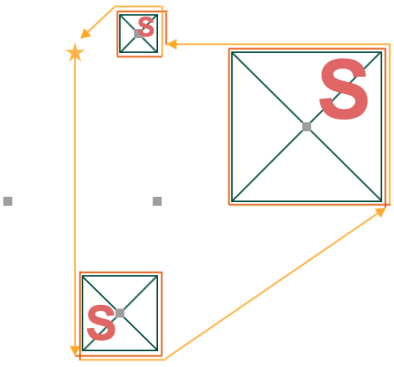


Figure 6.2: Path every_stop

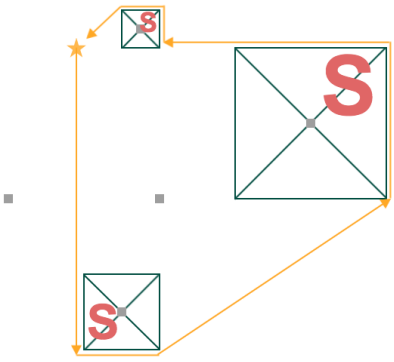


Figure 6.3: Path outside_stop

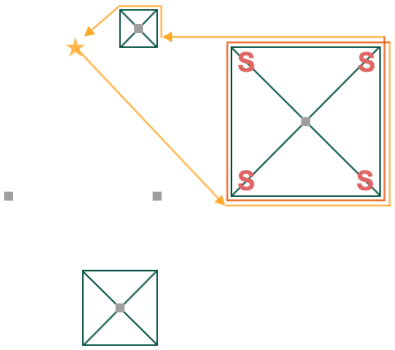


Figure 6.4: Path surround_big_stop

In addition, the test person is required to record twice for each designed path. One time with relative slow speed, one time with relative fast speed. The original intention is to challenge the DTW algorithm more forward - assess its ability of differentiating speed. During the experiment, we have observed that the walking speeds of the invited person are highly diverse as the demand of slow speed and fast speed is pretty vague. Everybody has his own definitions of slow and fast. However, if we quantify the slow speed and fast speed for the test persons in general, those demands are unable to meet. For instance the fast speed is 2,5 m/s and the slow speed is 1 m/s. Then the experiment becomes impossible to carry out. Therefore the two versions of the same path will be taken as the same label.

We use the implemented software archetype to collect data. When the recording for one path is finished, the recorded sample is saved in a file in HDFS. Due to the immaturity of the archetype, some samples are lost during the saving process, because of the network transmission error. Table 6.1 shows the numbers and sizes of the recorded data for each label.

Table 6.1: Details of Recorded Samples

| | session 1 | session 2 |
|---|---|---|
| every | 21 | 20 |
| every_stop | 22 | 22 |
| outside | 22 | 22 |
| outside_stop | 22 | 22 |
| return | 22 | 22 |
| surround_small | 22 | 21 |
| surround_mid | 20 | 21 |
| surround_big | 22 | 22 |
| surround_big_stop | 22 | 21 |
| in total | 195 | 193 |
| size | 1,1MB | 1,7MB |

## 6.2 Classification Validation on Collected Data & Results

As discussed in section 4.3.2, we combine the DTW algorithm with the decision algorithms: 1NN, kNN and weighted kNN for the classification task. We use the same strategy to apply the classification validation on the whole data, on the data collected by each sensor and on the data collected for each person. Table 6.2 shows the accuracy and run time for the classification validations using all sensors and using each sensor.

Table 6.2: Classification Results on Collected Data

| Session One | | | | | | |
|---|---|---|---|---|---|---|
| | 1-NN | | k-NN | | weighted k-NN | |
| all sensors | 32,82% | 151,2 s | 29,74% | 134,9 s | 32,82% | 132,4 s |
| s1 | 20,00% | 61,6 s | 16,41% | 58,2 s | 17,44% | 58 s |
| s2 | 20,51% | 60,2 s | 26,15% | 52,5 s | 24,62% | 58,1 s |
| s3 | 26,67% | 61,3 s | 23,59% | 51,7 s | 26,15% | 60,8 s |
| s4 | 23,08% | 60,6 s | 21,03% | 55,5 s | 20,51% | 57,1 s |
| s5 | 28,72% | 56,6 s | 19,49% | 54,1 s | 22,05% | 52,7 s |
| Session Two | | | | | | |
| | 1-NN | | k-NN | | weighted k-NN | |
| all sensors | 29,02% | 333,4 s | 24,35% | 354,6 s | 27,46% | 369,3 s |
| s1 | 24,35% | 107,7 s | 22,80% | 103,2 s | 20,72% | 105,2 s |
| s2 | 23,32% | 109,9 s | 21,76% | 108,2 s | 21,76% | 105,1 s |
| s3 | 21,24% | 109,7 s | 21,24% | 104,1 s | 20,73% | 108,1 s |
| s4 | 12,95% | 110,5 s | 16,06% | 103,4 s | 14,51% | 106,8 s |
| s5 | 22,80% | 107,2 s | 13,99% | 108,5 s | 16,58% | 108,3 s |

We also apply the classification validation for every person (cf. Table 6.3). It is pretty interesting that the classification accuracy is quite diverse from each other among different test persons. The reason might be that a few of the test persons have similar walking habits while the other test persons have relatively distinct walking habits. The run time of the classification validation for each people is varying as well, since the lengths of the recorded samples are different, which are correlated to the walking speeds.

Table 6.3: Classification Results Per Person

| Session One | | | | | | | |
|---|---|---|---|---|---|---|---|
| | sample | 1-NN | | k-NN | | weighted k-NN | |
| p1 | 18 | 27,78% | 16,3 s | 44,44% | 13,0s | 33,33% | 12,6 s |
| p2 | 17 | 58,82% | 16,5 s | 35,29% | 16,0 s | 47,06% | 15,9 s |
| p3 | 18 | 44,44% | 11,9 s | 38,89% | 10,3 s | 44,44% | 11,3 s |
| p4 | 16 | 43,75% | 19,3 s | 37,50% | 19,3 s | 31,25% | 19,2 s |
| p5 | 18 | 27,78% | 13,5 s | 33,33% | 11,8 s | 33,33% | 12,5 s |
| p6 | 16 | 27,78% | 15,7 s | 43,75% | 14,4 s | 43,75% | 15,7 s |
| p7 | 18 | 27,78% | 19,5 s | 16,67% | 18,0 s | 33,33% | 17,2 s |
| p8 | 18 | 33,33% | 16,5 s | 22,22% | 14,0 s | 33,33% | 15,1 s |
| p9 | 18 | 11,11% | 10,9 s | 11,11% | 9,4 s | 11,11% | 10,2 s |
| p10 | 18 | 11,11% | 21,9 s | 22,22% | 18,5 s | 16,67% | 20,4 s |
| p11 | 18 | 27,78% | 37,5 s | 33,33% | 34,7 s | 33,33% | 31,8 s |
| Session Two | | | | | | | |
| | sample | 1-NN | | k-NN | | weighted k-NN | |
| p12 | 18 | 22,22% | 33,1 s | 33,33% | 32,3 s | 22,22% | 31,8 s |
| p13 | 18 | 27,78% | 36,6 a | 16,67% | 32,2 s | 27,78% | 30,4 s |
| p14 | 18 | 33,33% | 13,7 s | 22,22% | 12,6 s | 22,22% | 12,9 s |
| p15 | 18 | 11,11% | 16,4 s | 11,11% | 14,9 s | 11,11% | 14,7 s |
| p16 | 18 | 44,44% | 18,8 s | 22,22% | 17,0 s | 38,89% | 17,2 s |
| p17 | 18 | 27,28% | 28,5 s | 16,67% | 27,2 s | 33,33% | 30,8 s |
| p18 | 18 | 44,44% | 39,4 s | 38,89% | 36,4 s | 44,44% | 36,3 s |
| p19 | 15 | 6,67% | 50,5 s | 0,00% | 54,8 s | 6,67% | 45,6 s |
| p20 | 18 | 16,67% | 40,1 s | 22,22% | 40,0 s | 16,67% | 46,0 s |
| p21 | 18 | 33,33% | 72,6 s | 22,22% | 65,9 s | 27,78% | 70,0 s |
| p22 | 16 | 50,00% | 23,8 s | 56,25% | 22,0 s | 56,25% | 21,9 s |

## 6.3 Threats to Validity

As shown in Table 6.3, the classification accuracy on the real data is not so optimized. The classification accuracy is about 30% using all sensors, which is far lower than result on simulated data. The result of classification validation on simulated data indicates that, our approach with the DTW algorithm is appropriate for recognizing motion patterns theoretically. The classification would be fairly optimized with high quality distance information. However, in the practical experiment, the quality of the distance information, which we collected with beacons, is not so good. In this section, we look into several noise sources of the collected data.

### 6.3.1 Beacon Precision

The first conjecture is the precision of the distance collected by beacons. Thus we carry out a small evaluation on the beacon's precision. We statically record the distance information for certain distances: 0.5 meter, 1 meter, 2 meter and $\sqrt{5}$ meter (cf. Figure 6.5) for a certain period. The recorded distance information are plotted in Figure 6.6.

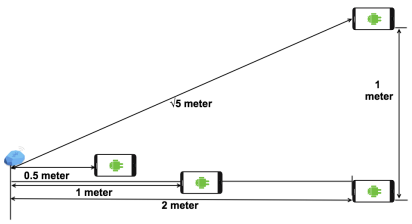


Figure 6.5: Beacon Precision

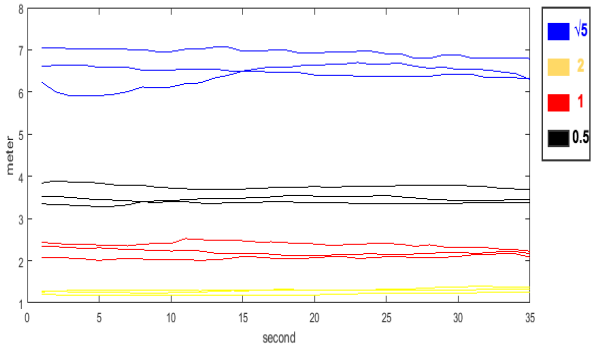


Figure 6.6: Plot of Beacon Precision

From the evaluation, it is evident that the distance information we collected are not accurate. One possible reason for the low quality would be the AltBeacon computing mechanism. As we explained in section 5.4.1, the AltBeacon library computes the

distances based on the RSSI method. Unlike iOS devices, there are many manufacturers for Android devices, different mobile devices have various Bluetooth radios and antennas.

As stated in section 5.4.1, the AltBeacon library learns a curve for distance computation. The learned curve is based on the prepared known table, and it can be only applied to a particular mobile device. Even though AltBeacon provides several models learned for different mobile devices [26], the mobile device used in our experiment is not in the list. As the result, the default model (learned for Nexus 5) will be used, whose learned curve might be not applicable to our device.

During the evaluation, we noticed another interesting phenomenon: the angle between the mobile device and the beacon influences the measured distance as well. We conduct another evaluation to expound. We use the mobile device to record the distance information for one meter from four different positions (cf. Figure 6.7). The recorded distance information are plotted in Figure 6.8.

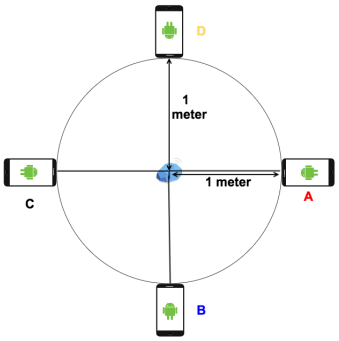


Figure 6.7: Beacon Precision in Angle

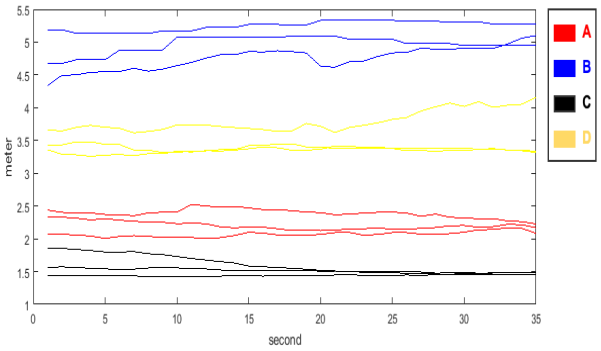


Figure 6.8: Plot of Beacon Precision in Angle

From the evaluation, it is indisputable that the angles between mobile device and beacon have a rather great impact on the measured distance. This could be another reason for the low quality of the collected distance information.

### 6.3.2 Other factors

#### Signal Interference & Reflection

In addition to the precision of beacon, we believe the signal interference and the reflection would influence the measured distance as well. In the experiment, we used five beacons, some of them are placed quite close to each other. As they broadcast the beacon packets at the same time, there must be signal interference. Moreover, the experiment area is built in a narrow room, some beacons are placed near the walls. The beacon signals are reflected by the walls and other objects in the room for certain. Since AltBeacon computes the distance based on the RSSI merely, the signal interference and reflection could be the sources of noise as well.

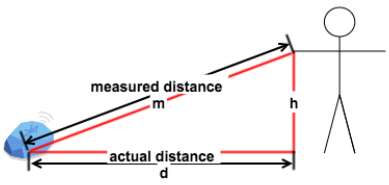


Figure 6.9: Distance with Height Influence

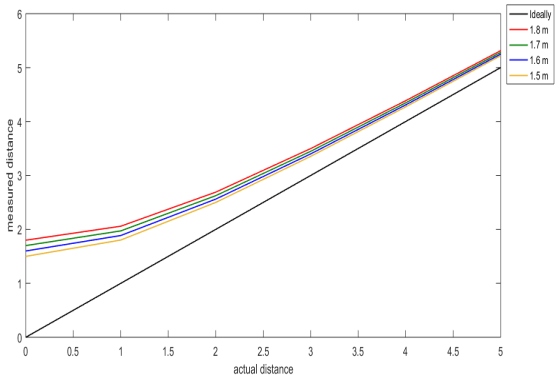


Figure 6.10: Plot of Height Influence

#### Height of Person

Another issue of our approach is the height of the test person. In the experiment, the measured distance is the direct distance between the beacon and the mobile device (m). Since the mobile phone is held in the hand of the test person, the computed distance are not the absolute two-dimension distance (d) between the beacons and the mobile device (cf. Figure 6.9). As the test persons have different heights, the mobile device cannot be held at the same height. However, the heights of test persons cannot be recorded for the ethical reason, therefore it is impossible to remove the noise of test persons' heights

after recording. Figure 6.10 illustrates the differences among the distances with heights compared to absolute two dimension distances.

# 7

# Evaluation

## 7.1 Remote Cluster

We use the remote cluster from the Institute of Medical Systems Biology for all the evaluations. This cluster is built for the goal of providing great computation power. It consists of only two machines *phi1* and *phi2*, and can be accessed through another machine *hopper*. In the cluster, Hadoop 2.8.1 is installed and started, it has one master node (on phi1) and one worker node (on phi2). Spark 2.2.0 with Scala 2.11.8 is installed and started as well, it has one master node (on phi1) and one worker node (on phi2). The Spark cluster consists of 271 CPU cores and 109 GB memory in total. Figure 7.1 shows the structure of the remote cluster.

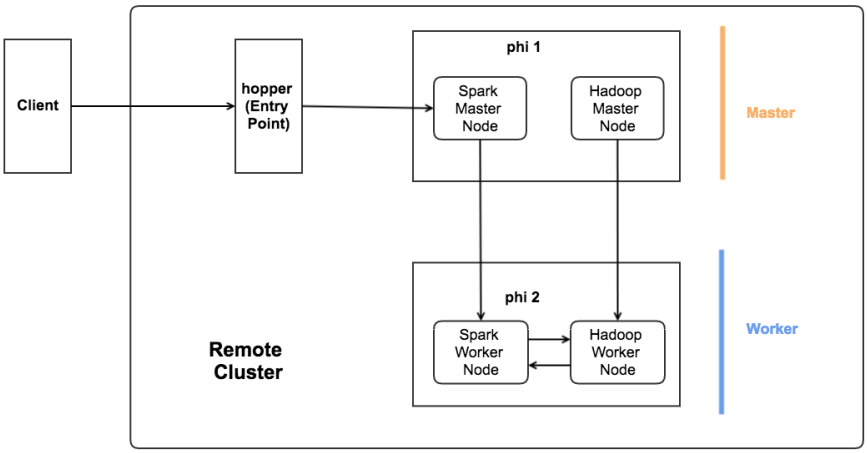


Figure 7.1: Structure of Remote Cluster

## 7.2 Synthetic Data

Since the dataset collected from the experiment sessions are small, it is not possible to apply the evaluations directly on them. Therefore, we take the recorded data as the basis, and generate the synthetic datasets.

The basic idea of the synthetic generation is to extend the length of each sample rather than increasing the number of the samples. By this operation, the generated samples are still valid, and the classification results on the synthetic datasets stay the same. We use the *resample()* function from the MATLAB framework, which resamples an input sequence at *p/q* times the original rate. By using this function, we can extend a time series to be *q/p* times as long as the original length. Figure 7.2 shows an example of employing this function on time series *t*, which resamples it at 1/2 and 1/5 times of the original rate. The resulted time series *t1* is two times long as *t*, and *t2* is five times long.

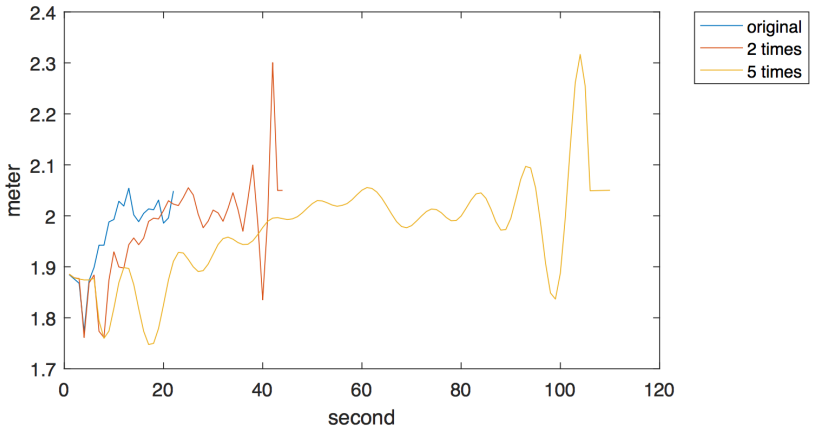


Figure 7.2: Plot of Resample

When generating the synthetic datasets, we keep the format of the samples stay the same as the original datasets, in which all samples are constitutive of the original label and five extended time series. We take the collected data from experiment session 1 as the generation basis, and resample it at the rate of 1/2, 1/5, 1/10, 1/20, 1/50 and 1/100, which extends the samples to be 2 times, 5 times, 10 times, 20 times, 50 times and 100 times as long as original lengths. Since the default precision of MATLAB implementation

is one ten thousandth, we also resample the original data at the rate of 1/1. Detailed information of the synthetic datasets are shown in Table 7.1.

Table 7.1: Details of Synthetic Datasets

| FileName | Rate | Times of Length | Size |
|---|---|---|---|
| original_data | / | / | 1,1 MB |
| 1times.txt | 1/1 | 1 | 406 KB |
| 2times.txt | 1/2 | 2 | 808 KB |
| 5times.txt | 1/5 | 5 | 2 MB |
| 10times.txt | 1/10 | 10 | 4 MB |
| 20times.txt | 1/20 | 20 | 8,1 MB |
| 50times.txt | 1/50 | 50 | 20,2 MB |
| 100times.txt | 1/100 | 100 | 40,3 MB |

## 7.3 Evaluation on Algorithm Performance

The DTW algorithm we have implemented is the standard version. As stated in the section 3.1.2, we have to traversal every element in the local distance matrix for computing the DTW distance between two time series. Therefore, the complexity of the standard DTW algorithm is $O(N^2)$ theoretically. In this section, we evaluate the performance of the DTW implementation using the synthetic datasets.

We submit the Spark application to the remote cluster, which applies the prediction on a certain test sample against different synthetic datasets. We note down the run time of the Spark application with different datasets. The Spark application runs with the default configurations in the remote cluster, which uses 271 cores in total and 1 GB memory on each executor. The evaluation results are shown in Figure 7.3, the x axis displays the size of the synthetic datasets in kilobyte, the y axis shows the run time in second. The blue points represent different Spark submits, and the red line illustrates the fitted curve against the plotted points. Details of learned parameters for the fitted curve are shown in Figure 7.4.

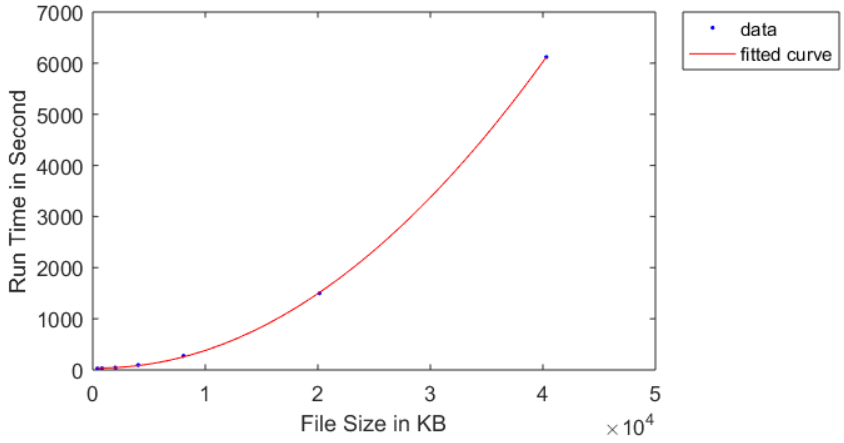


Figure 7.3: Evaluation on the Algorithm

```
f =

     Linear model Poly2:
     f(x) = p1*x^2 + p2*x + p3
     Coefficients (with 95% confidence bounds):
       p1 =   3.849e-06  (3.732e-06, 3.966e-06)
       p2 =   -0.004339  (-0.009115, 0.0004359)
       p3 =       40.23  (14.8, 65.66)
```

Figure 7.4: Parameters of Fitted Curve

## 7.4 Evaluation of Scalability

Scalability refers to the capability of the system, which can function well when the computation resources and task size are enlarged. Generally there are two types of scalability: *scale-up* and *scale-out*. *Scale-up* means adding more computing resources to a single node, e.g. the CPU resource and memory. It is also known as *scale vertically*. *Scale-out* means adding more computing nodes in the system, it is also called *scale horizontally*.

Distributed computing systems are designed to be able to scale easily. In this thesis, we implemented the *Prediction Component* in the software archetype with Spark. The run time of executing the prediction task is the critical metric of the software performance. As Spark applies the computation following the in-memory manner, the performance of the computation can be influenced by many resource factors like CPU cores, memory, network bandwidth. In this section, we evaluate the performance of the prediction task against different allocations of CPU cores and memory.

Since the remote cluster consists of two machines only: 1 master node and 1 worker node, it is not practical to evaluate the horizontal scalability. However, Spark provides the possibility to configure the allocated resources when submitting the Spark application. Hence, it is possible to evaluate the vertical scalability by tuning the submit configurations. The allocated memory for running the Spark application can be configured by the parameter *–executor-memory*, which sets the maximal memory usage for each executor. The CPU resource can be configured by the parameter *–total-executor-cores*, which sets the CPU cores used in total.

The first step of the evaluation is to double the used CPU cores and the maximal memory usage at the same time. From the aspect of resource usage, it equals to doubling the number of workers in the cluster. We use the synthetic datasets 10times.txt (4 MB) and 20times.txt (8,1 MB) for the evaluation. Table 7.2 shows the allocated resources and the run time for executing the Spark application. We plot the scaling results for better illustration (cf. Figure 7.5 and Figure 7.6).

Table 7.2: Scaling Resource Allocations

| Memory | CPU Cores | Factor of Scaling Resource | Run Time (10times.txt) | Run Time (20times.txt) |
|---|---|---|---|---|
| 2g | 5 | 1 | 84 sec | 234 sec |
| 4g | 10 | 2 | 72 sec | 198 sec |
| 8g | 20 | 4 | 66 sec | 162 sec |
| 16g | 40 | 8 | 63 sec | 147 sec |
| 32g | 80 | 16 | 59 sec | 138 sec |
| 64g | 160 | 32 | 58 sec | 135 sec |

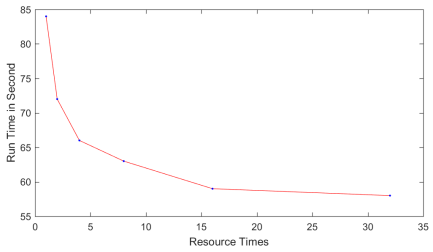


Figure 7.5: Scale Vertically on 10times

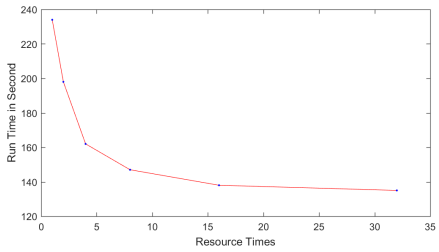


Figure 7.6: Scale Vertically on 20times

The evaluation results demonstrate clearly, as we add more resources for the computation, the run time declines. However, there is limitation on the decrease. When the allocated resources are more than 16 times as the first submit, the run time goes stable.

In addition to doubling the allocated resources for the computation, it is interesting as well to find out which parameter is more significant for speeding up the prediction. Therefore, we apply the evaluation for tuning the CPU cores and memory separately. The synthetic dataset used for the evaluation is 20times.txt (8,1 MB).

When apply the evaluation on the allocated memory, we keep the number of allocated CPU cores the same and double the maximal memory usage for the computation each time. We set the CPU cores to 80 as the situation of having abundant CPU resource, and set it to 5 as the CPU resource is lacking. The evaluation results are shown in Table 7.3 and are plotted in Figure 7.7.

Table 7.3: Tuning Memory

| Memory | Cores | Run Time | Cores | Run Time |
|---|---|---|---|---|
| 2g | 80 | 240 sec | 5 | 242 sec |
| 4g | 80 | 198 sec | 5 | 195 sec |
| 8g | 80 | 168 sec | 5 | 168 sec |
| 16g | 80 | 150 sec | 5 | 151 sec |
| 32g | 80 | 138 sec | 5 | 137 sec |
| 64g | 80 | 136 sec | 5 | 135 sec |

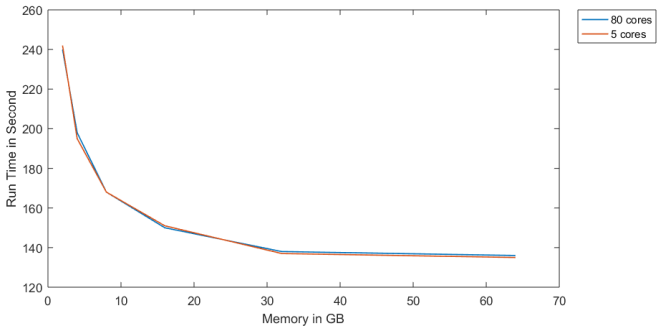


Figure 7.7: Plot of Tuning Memory

When apply the evaluation on the allocated CPU cores, we keep the maximal memory usage the same and double the number of used CPU cores each time. We set the allocated memory to 32 GB as the situation of having rich memory resource, and 2 GB as the memory resource is plain. The evaluation results are shown in Table 7.4 and are plotted in Figure 7.8.

Table 7.4: Tuning CPU Cores

| Cores | Memory | Run Time | Memory | Run Time |
|---|---|---|---|---|
| 5 | 32g | 138 sec | 2g | 236 sec |
| 10 | 32g | 139 sec | 2g | 234 sec |
| 20 | 32g | 138 sec | 2g | 234 sec |
| 40 | 32g | 141 sec | 2g | 237 sec |
| 80 | 32g | 138 sec | 2g | 235 sec |
| 160 | 32g | 137 sec | 2g | 234 sec |

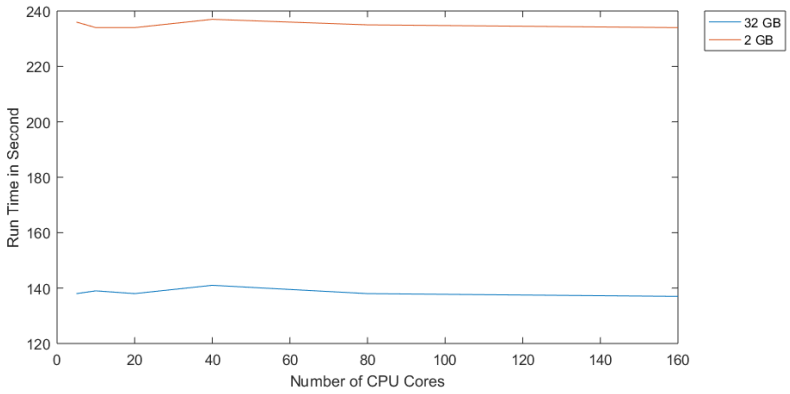


Figure 7.8: Plot of Tuning CPU Cores

The evaluation results indicate the run time can be shorten prominently by employing more memory in the computation. The run time tends to be steady when using more than 30 GB memory for the dataset 20times.txt. The number of involved CPU cores in the computation has a relatively tiny impact on the performance.

# 8

# Future work

In this thesis, we evaluate an approach to classify the motion patterns, which applies the DTW algorithm with the beacon-based distance data. We also provide an implementation of the software archetype, from collecting distance to performing the classification in the distributed environment.

As shown in section 6.2, the classification results on the real data are not optimized, there are several issues can be improved. In this chapter, we present a short discussion on those issues. Generally the further research issues involve four aspects:

1. Indoor positioning techniques
2. Indoor positioning strategies
3. Experiments with other scenarios
4. Horizontal Scalability

## 8.1 Indoor Positioning Techniques

The most critical issue as the next step is to validate whether AltBeacon is the appropriate technology for measuring distance. It is impractical to apply classification on the data containing large deviation. In order to remove, or at least reduce the bias in the collected distance information, we can switch to the experiment device which is supported by AltBeacon. Or we can prepare the known table of distance/RSSI for the used device. Then evaluate the accurateness of the measured distances as presented in section 6.3.1.

However, the problem of inaccurate measurements indicates, that the AltBeacon is not universally compatible for Android devices. Therefore, it might not be a good option for the software implementation in the production. Another idea for the next step is to find the alternatives of AltBeacon. For instance to implement the mobile application on iOS device with iBeacon. Besides, it is also meaningful to try other *Indoor Positioning System* (IPS) for locating rather than adhering to beacon related techniques. For example, collecting the location information with Wi-Fi-based IPS.

## 8.2 Indoor Positioning Strategies

The second aspect of the future work is to improve the positioning strategies. GPS is significantly effective for positioning outdoors, but it cannot provide accurate location information inside the building. That is the reason why we applied several sensors to collect the distance information, instead of obtaining the location coordinates directly. The distance information can be considered as the encoding of location coordinate in the form of a multi-dimensional metric.

It is yet pretty interesting to apply the DTW classification directly on the location coordinates. With the help of the geometry methodology such as *Triangulation* [27] and *Trilateration* [28], the location coordinates can be computed using several relative distances. Moreover, the location coordinates can be computed in the three dimensional format. In that case, it is possible to eliminate the noise of the height.

Furthermore, some sensors provide reliable distance measurements when the object is in the neighboring area. For example, the measured distance by iBeacon is fairly accurate within two meters. Therefore, another idea of improving the location accuracy is to select the collected distances dynamically. Then the location coordinate can be computed with the distances collected by the closest $N$ sensors instead of by all sensors.

As stated in section 5.4.1, beacon is originally designed for proximity detection. Other sensors, like passive *Radio-Frequency Identification* (RFID), are designed for this purpose as well. Therefore, further idea of the positioning strategy is to encode the distance information in the binary format. That is to say, when the person is clearly close to the

sensor, for example within 50 centimeters, the distance is encoded as 1; otherwise the distance is encoded as 0. The precision of the encoded distance gets enhanced as the number of applied sensors increases.

## 8.3 Experiments with Different Scenarios

Another aspect of the future work is to repeat the experiment with different scenarios and invite more people as the experiment objects.

In our experiment, we designed only six paths and they are quite similar to each other. The results on the simulated data suggests, those six paths can be distinguished with the DTW algorithm (cf. section 4.3.3). It is interesting to know, whether our approach is also adequate for other paths with different layouts of the sensors.

Furthermore, we have invited 11 people for each session in the experiment. Classification results on such small sample space is not convincing. Besides, the size of the collected data is too small to employ the distributed computing technologies. Therefore, it is significant to invite more people for conducting the experiment.

## 8.4 Horizontal Scalability

As stated in section 7.4, we only evaluate the scalability by allocating more resource in the cluster. More interesting as the further step, is to evaluate the scalability of adding more machine nodes in the cluster. Considering the mechanisms of shuffling and sorting of Spark, it may reveal different scaling patterns.

In addition, it is also interesting to build the computing cluster with Raspberry Pi machines, with the fitting Spark parameters and faster network switches.

# Bibliography


[1] Weiser, M., Brown, J.S.: The coming age of calm technology. In: Beyond calculation. Springer (1997) 75–85

[2] Weiser, M.: The computer for the 21 st century. Scientific american **265** (1991) 94–105

[3] Hadoop, A.: Apache hadoop (2011)

[4] Borthakur, D.: The hadoop distributed file system: Architecture and design. Hadoop Project Website **11** (2007) 21

[5] Vavilapalli, V.K., Murthy, A.C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., et al.: Apache hadoop yarn: Yet another resource negotiator. In: Proceedings of the 4th annual Symposium on Cloud Computing, ACM (2013) 5

[6] Kavulya, S., Tan, J., Gandhi, R., Narasimhan, P.: An analysis of traces from a production mapreduce cluster. In: Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on, IEEE (2010) 94–103

[7] Spark, A.: (Apache spark)

[8] Zaharia, M., Xin, R.S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M.J., et al.: Apache spark: A unified engine for big data processing. Communications of the ACM **59** (2016) 56–65

[9] Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: Cluster computing with working sets. HotCloud **10** (2010) 95

[10] Ayyalasomayajula, H.: An Evaluation of the Spark Programming Model For Big Data Analytics. PhD thesis (2015)

[11] Maksimović, M., Vujović, V., Davidović, N., Milošević, V., Perišić, B.: Raspberry pi as internet of things hardware: performances and constraints. design issues **3** (2014) 8

[12] Vujović, V., Maksimović, M.: Raspberry pi as a sensor web node for home automation. Computers & Electrical Engineering **44** (2015) 153–171

[13] Ansari, A.N., Sedky, M., Sharma, N., Tyagi, A.: An internet of things approach for motion detection using raspberry pi. In: Intelligent Computing and Internet of Things (ICIT), 2014 International Conference on, IEEE (2015) 131–134

[14] Chowdhury, M.N., Nooman, M.S., Sarker, S.: Access control of door and home security by raspberry pi through internet. Int. J. Sci. Eng. Res **4** (2013) 550–558

[15] Berndt, D.J., Clifford, J.: Using dynamic time warping to find patterns in time series. In: KDD workshop. Volume 10., Seattle, WA (1994) 359–370

[16] Ratanamahatana, C.A., Keogh, E.: Everything you know about dynamic time warping is wrong. In: Third Workshop on Mining Temporal and Sequential Data, Citeseer (2004) 22–25

[17] Puri, C., Ukil, A., Bandyopadhyay, S., Singh, R., Pal, A., Mandana, K.: icarma: Inexpensive cardiac arrhythmia management–an iot healthcare analytics solution. In: Proceedings of the First Workshop on IoT-enabled Healthcare and Wellness Technologies and Systems, ACM (2016) 3–8

[18] Chen, S.Y., Lai, C.F., Huang, Y.M., Jeng, Y.L.: Intelligent home-appliance recognition over iot cloud network. In: Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International, IEEE (2013) 639–643

[19] Bertsekas, D.P., Tsitsiklis, J.N.: Neuro-dynamic programming: an overview. In: Decision and Control, 1995., Proceedings of the 34th IEEE Conference on. Volume 1., IEEE (1995) 560–564

[20] Ghemawat, S., Gobioff, H., Leung, S.T.: The google file system. In: ACM SIGOPS operating systems review. Volume 37., ACM (2003) 29–43

[21] Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The hadoop distributed file system. In: Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on, IEEE (2010) 1–10

[22] Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Communications of the ACM **51** (2008) 107–113

[23] Efron, B.: Bootstrap methods: another look at the jackknife. In: Breakthroughs in statistics. Springer (1992) 569–593

[24] Wu, X., Kumar, V., Quinlan, J.R., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G.J., Ng, A., Liu, B., Philip, S.Y., et al.: Top 10 algorithms in data mining. Knowledge and information systems **14** (2008) 1–37

[25] Van Cutsem, T., Mostinckx, S., De Meuter, W.: Linguistic symbiosis between event loop actors and threads. Computer Languages, Systems & Structures **35** (2009) 80–98

[26] GARCIA, C.: Altbeacon. `https://github.com/AltBeacon/android-beacon-library/blob/master/src/main/resources/model-distance-calculations.json` (2016)

[27] Hartley, R.I., Sturm, P.: Triangulation. Computer vision and image understanding **68** (1997) 146–157

[28] Manolakis, D.E.: Efficient solution and performance analysis of 3-d position estimation by trilateration. IEEE Transactions on Aerospace and Electronic systems **32** (1996) 1239–1248

# A

# Code

In this chapter, the critical codes of using AltBeacon library for scanning beacons nearby are listed, which are discussed in section 5.4.1.

```
1  public class AltBeaconActivity extends Activity implements
       BeaconConsumer {
2      private static final String INFO = "Beacon_Info";
3      private BeaconManager beaconManager;
4      @Override
5      protected void onCreate(Bundle savedInstanceState) {
6          super.onCreate(savedInstanceState);
7          setContentView(R.layout.activity_altbeacon);
8          beaconManager =
               BeaconManager.getInstanceForApplication(
9                  MainActivity.this);
10         beaconManager.getBeaconParsers().add(new
               BeaconParser().setBeaconLayout("m:2-3=0215," +
11             "i:4-19,i:20-21,i:22-23,p:24-24"));
12         beaconManager.getBeaconParsers().add(new
               BeaconParser().setBeaconLayout("m:2-3=beac," +
13             "i:4-19,i:20-21,i:22-23,p:24-24,d:25-25"));
14         beaconManager.bind(AltBeaconActivity.this);
15     }
16     @Override
17     public void onBeaconServiceConnect() {
```

```
18          Log.i(INFO, "onBeaconServiceConnect");
19          beaconManager.setBackgroundBetweenScanPeriod(200l);
20          beaconManager.setBackgroundScanPeriod(800l);
21          beaconManager.setBackgroundMode(true);
22          beaconManager.addRangeNotifier(new RangeNotifier() {
23              @Override
24              public void
                    didRangeBeaconsInRegion(Collection<Beacon>
                    beacons, Region region) {
25                  Log.i(INFO, new Date() + " ranging in region,
                        size: " + beacons.size());
26                  for(Beacon beacon: beacons){
27                      Log.i(INFO, beacon.getDistance()+"");
28                  }
29              }
30          });
31          try {
32              beaconManager.updateScanPeriods();
33              Region region = new
                    Region("com.easibeacon.demos.demo1", null,
                    null, null);
34              beaconManager.startRangingBeaconsInRegion(region);
35          } catch (RemoteException e) {
36              e.printStackTrace();
37          }
38      }
39  }
40  //...
```

Listing A.1: Example of AltBeacon Ranging

# List of Figures

# List of Tables

Name: Yu Tong Matrikelnummer: 900846

**Erklärung**

Ich erkläre, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Yu Tong