UNIVERSITÉ LIBRE DE BRUXELLES

INFOH-515 - BIG DATA

BIG DATA MANAGEMENT & ANALYTICS

Scalable distributed online forecasting system for missing error measurements - Phase II

Authors:
Anass Denguir
Keneth Ubeda
Christian Frantzen

Professor:
Stijn Vansummeren
Kyriakos Efthymiadis

June 10, 2019

Contents

1 Introduction

Dealing with noisy, missing and erroneous values is one of the hardest challenge when working Internet of Things data. An efficient way to handle these issues is to rely on machine learning prediction models. Specifically we can threat time series forecasting as a Supervised Machine Learning problem. The features we can use for these kind of problems are sensor streams of measures and device properties (e.g. geo-localization, measure time), among others. Some of the most relevant challenges are: large dimensionality data, non-stationary data, real-time processing, irregular and non structured data management, anomaly detection, spatio-temporal analysis and noise.

The phase I of this project consisted in the design and development of Big Data Management & Analytics pipeline. The BDMA pipeline should be able of efficiently handle the flow of sensors data as well as processing it in a effectively distributed manner. In this phase (II), we were requested to deliver an online scalable distributed prediction system for missing error measurements. The idea is to implement 3 different Machine Learning models that uses spatio-temporal data to deal with missing and erroneous measurements. In the following section we describe the Data set characteristics (section 2), The system architecture (section 3), Details about the utilized algorithms with their respective results (section 4), Future work (section 5) and overall conclusions regarding the design and implementation of the system (section 6).

2 Data set

2.1 Sensor 1 and 24

We try to predict the temperature measurements of sensors 1 and 24 in the data set for the week of 1/03/2017 to 7/03/2017 with the 28/02/2017 being the date from which we will use data to train our models initially. In the following you will see multiple graphs of the readings from sensors 1 and 24, just like from some of their adjacent sensors. First, we shall look at the readings of sensor 1 and 24 for the first day, 28/02/2017:

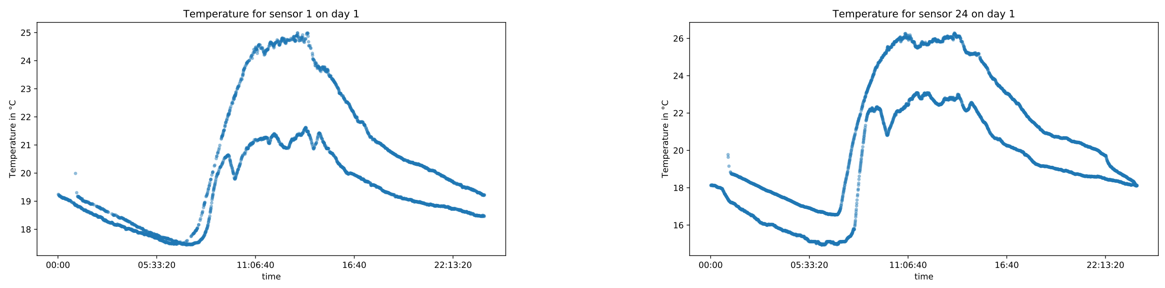


Table 1: Readings for sensors 1 and 24 for the first day

As you can see from the measurements of sensor 1 and 24, their values vary quite a lot in-between subsequent readings. While the difference between readings is bounded by around 1 degree Celsius in the beginning of the day, the difference raises up to 2 to 3 degrees during midday. The day starts around 19 degrees, cools down to around 15 degrees before it starts to get hotter again at 7am. The maximum temperature for each sensor seems to be reached at 12pm, which keeps to be the temperature until the end of the afternoon at which point the temperature starts to decline slowly until midnight. While one can see some gaps between data points, in general there seems to be continuity in readings received.

Now, let us look at the readings for the second day, 01/03/2017 :

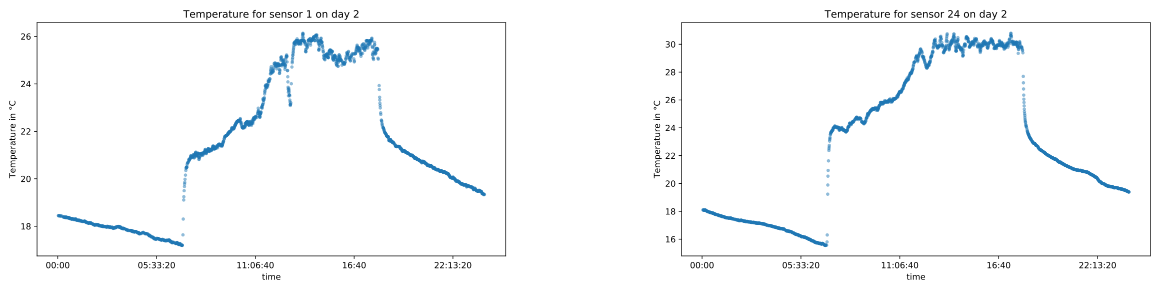
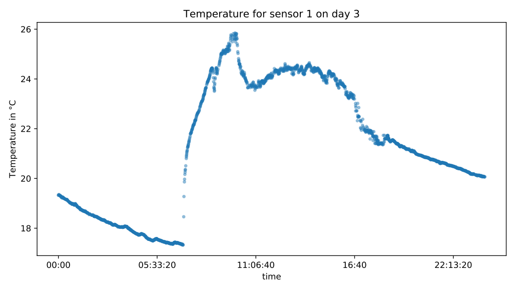


Table 2: Readings for sensors 1 and 24 for the second day

Unlike the first day, there do not seem to be as many variations between subsequent readings. Just like for day 1, the minimum temperature is reached around 7am, however then the temperature climbs far quicker than the previous day and there are a lot of gaps between 7 and 8 am. It also stays hotter for a longer period of time, until 5pm, at which point gaps in-between subsequent readings are noticeable again. Notice also that the maximum temperature for sensor 24 rose from 26 degrees to 30 while the one for sensor 1 stayed close to the previous day's with 26 degrees.



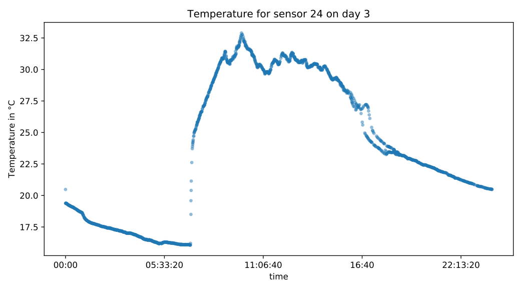
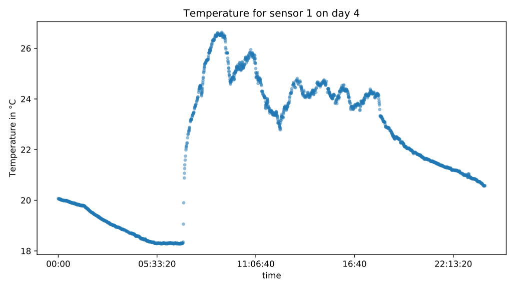


Table 3: Readings for sensors 1 and 24 for the third day

For the third day, 02/03/2017, the gaps of readings in the morning hours when the temperature starts rising is the same than the day before. However, the temperature rises even faster and peaks even before 11am. The global decline in temperature is smoother than day 2, however there is a small decline and than rise in temperature at 11am, also there appear variations in subsequent readings again around 5pm for sensor 24.



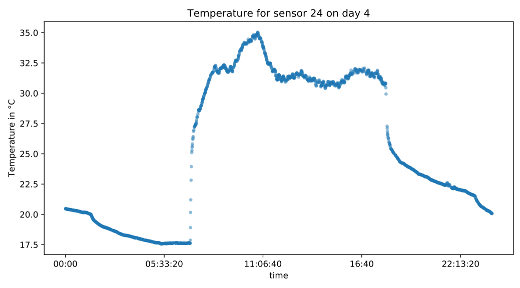


Table 4: Readings for sensors 1 and 24 for the day 4

For day 4, the pattern in the morning of the previous days applies again: at 7am temperature starts rising, with gaps in-between subsequent readings. Sensor 1 diverges from sensor 24 on that day in multiple ways: the peak temperature is before 11am and then the temperature declines in a constant fashion but with saw-like pattern until the end of the afternoon where it declines smoothly again. Sensor 24 peaks around 11am, with a subsequent decline but then stays at a constant temperature before the end of the afternoon where it declines abruptly. Again gaps between data points.

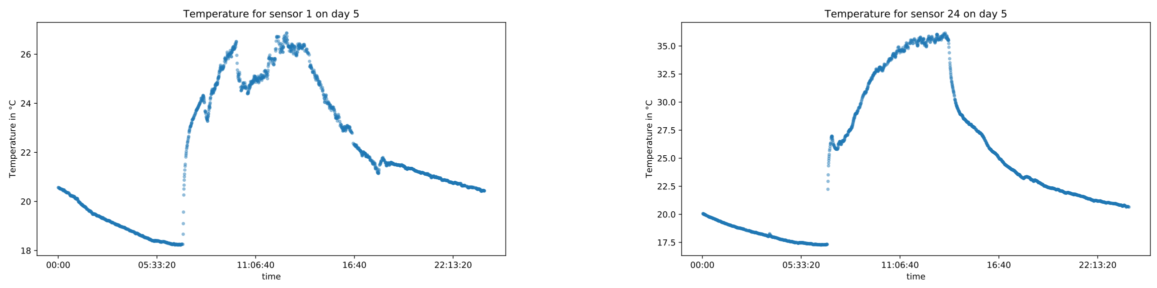


Table 5: Readings for sensors 1 and 24 for the day 5

For day 5, after the usual lack of data points at the beginning of the temperature rising, sensor 1 peaks early before 11am then declines by about 2 degrees and then returns to its maximum temperature again before it declines smoothly at the end of the afternoon. Sensor 24 registers a constant climb in temperature until the end of the afternoon before it declines fast in a first instance and then smoothly again.

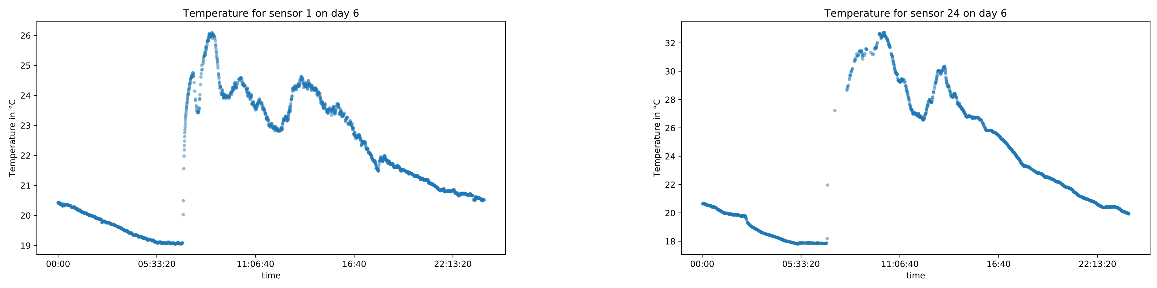


Table 6: Readings for sensors 1 and 24 for the day 6

For day 6, sensor 24 has a big amount of readings missing in the morning leading up to the peak temperature. Both senors record a decline after the peak temperature followed by a small climb again. The day cools off smoothly.

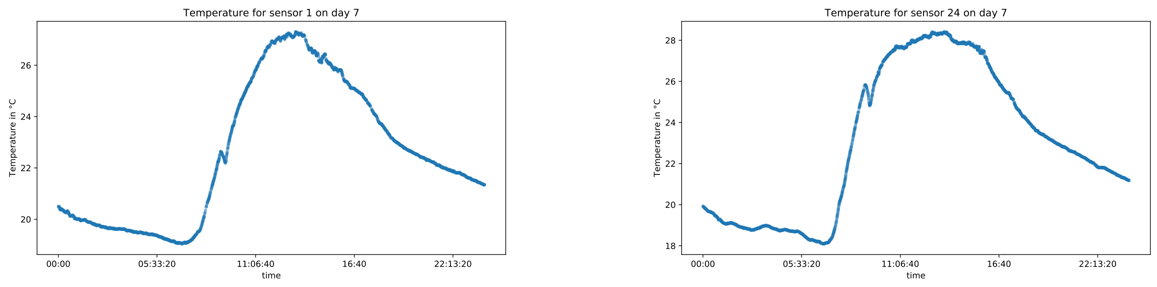


Table 7: Readings for sensors 1 and 24 for the day 7

Day 7 shows a bell-like shape for the temperature graphs, no major lack of readings over the course of the day. Noticeable is only a small fold before 11am for both sensors.

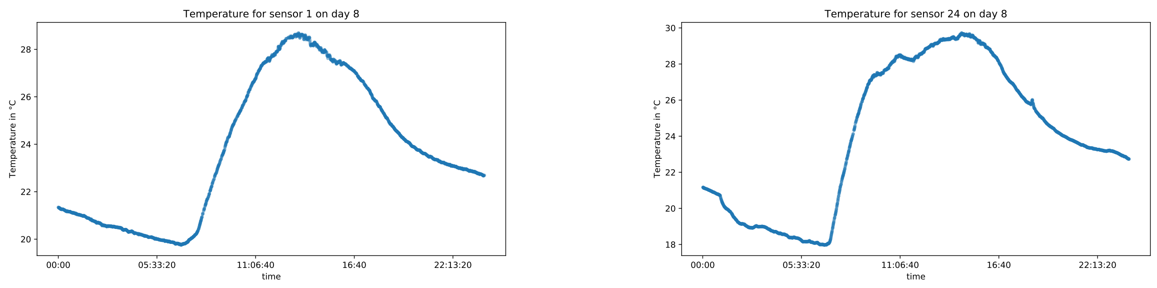


Table 8: Readings for sensors 1 and 24 for the day 8

The last day shows a similar development as the day before.

To represent the different days and sensors in a more readable fashion we list their characteristics in a table where we show the number of readings, minimum and maximum temperature, their average and the first and last reading of each day:

Day	Sensor	# Readings	Max	Min	Avg	First reading	Last reading
1	1	3045	24.99	17.44	20.02	19.24	19.22
2	1	1921	26.13	17.20	21.48	18.45	19.34
3	1	2008	25.85	17.32	21.31	19.32	20.07
4	1	1868	26.63	18.27	22.08	20.06	20.58
5	1	1741	26.86	18.22	22.08	20.57	20.42
6	1	1824	26.10	19.05	21.88	20.42	20.51
7	1	2049	27.30	19.04	22.55	20.50	21.34
8	1	2136	28.69	19.75	23.67	21.34	22.69
1	24	4287	26.29	14.93	20.05	18.14	18.13
2	24	1959	30.79	15.56	23.12	18.11	19.39
3	24	1886	32.90	16.07	23.80	20.48	20.48
4	24	1743	35.03	17.56	25.56	20.48	20.07
5	24	1786	36.16	17.26	24.59	20.07	20.65
6	24	1573	32.75	17.80	23.47	20.65	19.93
7	24	2266	28.41	18.09	23.14	19.93	21.17
8	24	2367	29.72	17.97	24.12	21.16	22.73

Table 9: Statistics for sensors 1 and 24 for the 8 days

The main takeaways from table 9 are that for sensor 1, temperatures are rising over the course of the week, with day 6 being a small exception. For sensor 24, the main observation is that while temperatures rise at the first half of the week they decline again over the second half but stay above the initial temperature values from day 1. While we were able to observe those traits in the figures before, the table makes easier to put things into perspective. Noticeable is also the fact the last reading and the first reading of subsequent days are compatible, meaning that there are no huge amount of differences we have to investigate. The only thing to investigate is the number of readings per day and the numerous oscillations of subsequent readings for day 1 and some other days. Since the sensors are supposed to send a reading every 30 seconds, the total number of readings per day should be around 2880. Both sensor 1 and 24 have more than said number of readings, with sensor 4 having 50% more readings than it should. When looking at the raw data, one observes that whenever there is a reading which is 'out of line' with the previous reading, there usually is a second reading recorded within a few seconds of that presumably faulty reading. This explains the 2 different curves for both sensors on day 1 and it also leads to the impression that a sensor may be able to tell when it sent out an incorrect reading and then sends out a new, correct value. Another possibility is that the default update period is every 30 seconds but whenever it detects a change of temperature above a certain threshold it sends out an update. Since we do not know the setting of the sensor we can only make assumptions about its behaviour. When grouping readings received in the same minute, we find that for sensor 1 there are 652 readings more than there should be, for sensor 24 has 1509 of those excessive/faulty readings.

2.2 Adjacent Sensors

Since we know that there will be readings missing, one might resort to readings of adjacent sensors to infer the current temperature of a sensor. This reasoning is based on the intuition that, given that all sensors are in the same space, they should have similar temperatures, especially those which are close to each other. Of course depending on the environment, some sensors might be exposed to conditions which make them unfit to compare them to their adjacent sensors, such as being exposed to sunlight because of a nearby window, being close to an air conditioning unit or heater. To check the viability of this method we shall have a look at the map for the space we are working with and see which sensors we might useful to improve our system.

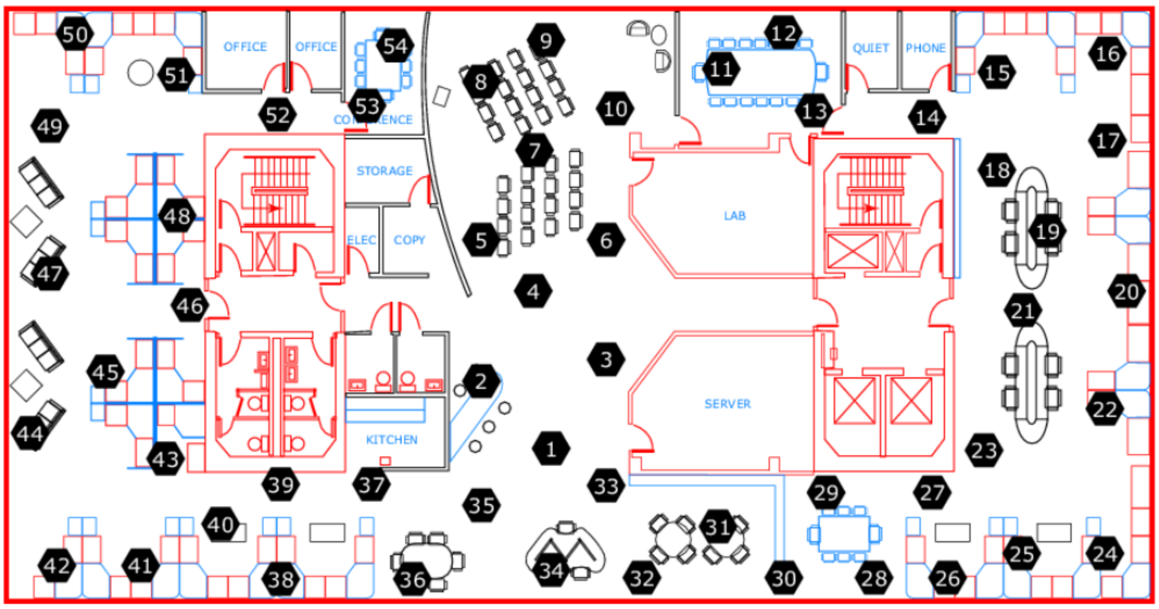


Figure 1: Placements of sensors in the space

3 Architecture

Dealing with IoT data as stream is a big challenge because the data can rapidly grow in volume meaning that lots of devices can be added. In this project adding more sensors to the system affects the velocity in which the system can process the data, if this is not able of scale horizontally. In the first phase of the project we studied some architectures and tools that allows us handling Big Data in a proper way. We learned from the phase I that in order to manage Big Data efficiently we must use distributed systems that are aware of their distributed nature. This characteristic is important because the system should be aware of the available resources and use them effectively in case that the incoming data increases.

The architecture we built, is aiming to solve some of the challenges when processing streaming data; receive continues data (no start, no end), Scalability (adding more sensors to the system), Data receiving rate, apply ML algorithms over streams and data visualization in real-time. We will now describe the architecture of the system (Figure 2) and explain how, with these selected tools, we can properly handle the data generated by sensors coming from smart cities, in this case from Brussels. The architecture follows the guidelines of a κ-architecture which is a software architecture pattern to work with massive streaming data. We use Apache Kafka to deal with the acquisition of the sensor's data, Spark streaming as streaming processing framework, HDFS (Hadoop) + OpenTSDB(HBase for time series data) as distributed storage, and Grafana as visualization tool. To access the data stream we use Jupyter notebooks that already communicated with the Spark nodes to make the computations.

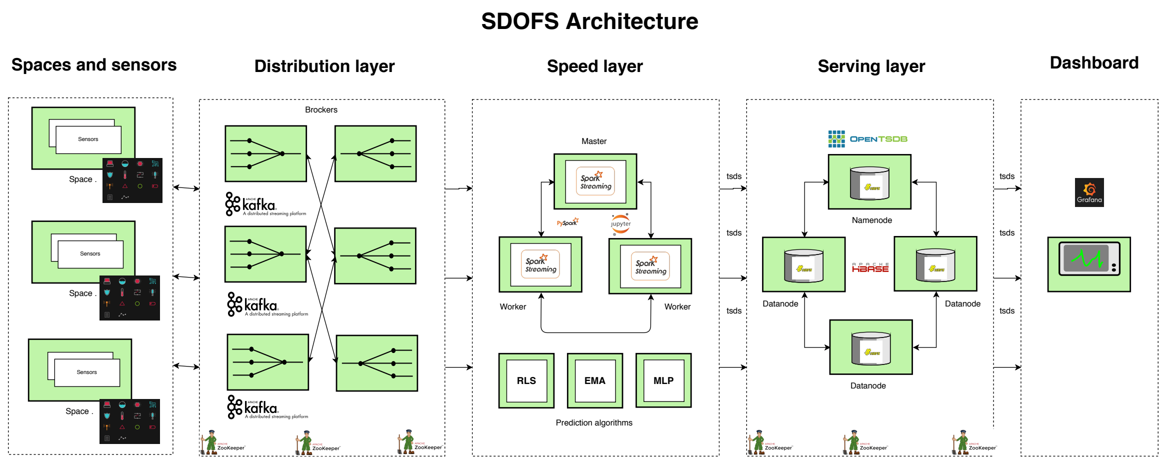


Figure 2: SDO forecasting system for missing error measurements architecture. This is a distributed architecture, with no single point of failure, that is able to work in elastic hardware environments (e.g. AWS) by adding more nodes or more processing units to each node.

3.1 Scalability

The most common issues when handling big data systems are: volume, velocity, variety, performance (e.g. throughput, latency), distributed computation. All those issues are directly related to the scalability of the system. However, nowadays there are some tools that allow us to handle these scalability with some level of facility, such as distributed file systems, distributed queue systems, distributed computation frameworks, Hardware on demand (Elastic cloud). Systems are shifting to horizontal scaling (multiple computation units) rather than vertical scaling (make faster a computation unit).

Big data Systems should be deployed in a parallel and distributed environment but also effectively use the resources. For that to happen systems should be conceived with a parallel distributed architecture since the beginning. That is a challenge because, many assumptions made in single-node systems are not valid anymore. For example, data must be partitioned across many nodes in a cluster and there are some algorithms with wide data dependencies that will suffer from the fact that network transfer rates are orders of magnitude slower than memory accesses. Other examples are: when the number of machines increases the probability of failure increases, The need of a programming paradigm to work in a highly parallel manner and the fact that no general approach for distributing machine learning is found in literature. In this specific system we found some important data management challenges and computational challenges to deal with, in order to achieve scalability. We will go through them by describing the problem and giving it a solution by means of a tool or a technique that works in a parallel and distributed environment.

3.1.1 Data management challenges

The first challenge is collecting the data and treating it as stream. For that purpose we use Apache Kafka, which is a distributed streaming platform with 3 principal capabilities: publish and subscribe to stream of records (message queue or messaging system), store streams in a fault-tolerant way and process streams records as they occur. Apache Kafka runs as a cluster on one or more servers (distributed nature), and it uses categories called topics to store the records. The use of topics allow us to organize the incoming data according to our needs. For example for this project we can create topics for groups of spaces or even spaces, that we want to have together to process independently and in parallel. Another scenario in which that is useful is to group incoming data with similar characteristics (temperature sensors, motion sensors, similar devices, similar Geo-localization, etc), this is useful specially for Machine Learning processes because we need similar data to make the models working better. To horizontally scale with Apache Kafka we can add more Kafka servers in the cluster and workload will be distributed through them.

The second challenge is processing the data in such a way that is useful as inputs to the prediction algorithms. For instance, we receive the data as logs, in which we could have information that is useful to know the context, but not interesting for the algorithms. We should then, extract the information that is useful for us, such as : measurements that are close to each other, measurements that are in similar hours but different days, or even measurements of same time but taken by other devices of the same type (neighborhood). These can be a really hard computational task when working with massive data, it can be infeasible to tackle with sequential approaches. Fortunately, There are frameworks that allow us divide these tasks in independent tasks and thus process them in parallel. The framework we use is Apache Spark, which is a fast and general-purpose cluster computing system that uses the Map-Reduce programming model. We use the streaming tool that it provides, called Spark streaming that brings Apache Spark's language-integrated API to stream processing, it transforms datastreams from any service such as HDFS, Kafka or Twitter into a sequence of Resilient Distributed Datasets (RDD), namely a DStream. We can scale spark streaming by adding more computation nodes to the cluster which.

The last data management challenge is to be able of querying and visualizing the processed data in real time. For this purposes we use OpenTSDB a Database that uses Hadoop as distributed file system to store the data. OpenTSDB is specialized in handling Time series data for what is perfect for this project since IoT is time series data. It provides a high writ ting rate and also an API to make temporal queries. OpenTSDB can be easily connected to Grafana an open source visualization and monitoring platform. Grafana is able to program queries in time (e.g. querying every 1 second) and it provides a friendly user interface to build different kind of plots and them to a persistence Dashboard.

3.1.2 Computation challenges

So far, we have the tools that already provide us an scalable environment, however this isn't enough for the forecasting system to be scalable. This isn't enough because the prediction models algorithms should be compatible with with the distributed and parallel nature of the environment. For that to happen we have two possible solutions, which is a Machine Learning platform such as MLLib that works on top of Spark or redesign or adapt the algorithms in a scalable manner (Map-Reduce paradigm). We have selected models that can be adaptable as Recursive Least Squares (RLS) and Exponential Moving Average (EMA) and we explain in detail in the following sections.

4 Prediction models

In this section, we will present different possible prediction models to estimate the future temperature of a given set of sensors. We will then compare their performance in terms of Mean Squared Error (MSE). At first, we will present the simplest possible model, which is the Persistence model. This model will serve as baseline for the following ones. The second model is a time-series model, which models the temperature evolution as a Linear Time Invariant (LTI) System whose parameters are optimized in the least-square sense by using the Recursive Least Square (RLS) algorithm. The third model exploits spatial correlation between the sensors to predict future temperature. To do that, the model computes an Exponential Moving Average (EMA) of the neighboring sensor temperatures. Finally, the last model is based on a Multi Layer

Perceptron (MLP) which uses data of neighbouring sensors to predict the temperature of sensors we're interested in.

4.1 Persistence

4.1.1 Model

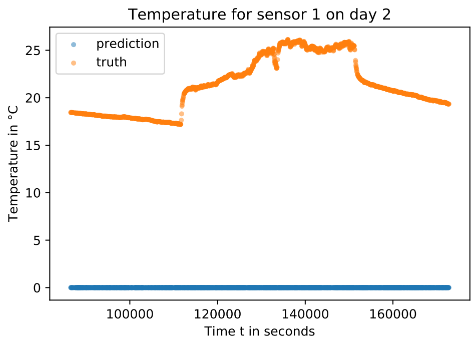
The Persistence model is a very simple model which consists of assigning the prediction temperature $\hat{T}(t)$ to its previous value $T(t-1)$. The prediction equation can thus be summarized as follows:

$$\hat{T}(t) = T(t-1) \tag{1}$$

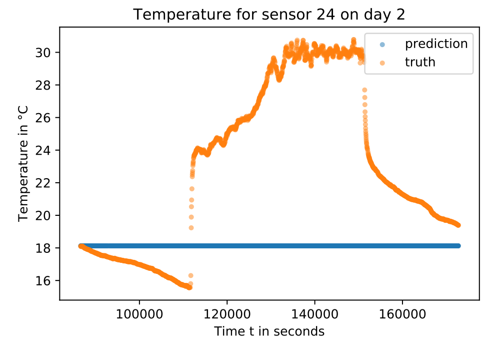
This naive approach has several weaknesses. First, it gives poor results when the sensor data are missing for a long time since the temperature will remain constant until new data are coming. As the model should be able to fill an entire day of missing values, it is obvious that the Persistence model is not well fitted for our application. A second disadvantage of this model is the fact that it takes only the last temperature into account, which implies that this model is very sensitive to noise as it will simply repeat potential noisy data.

4.1.2 Results

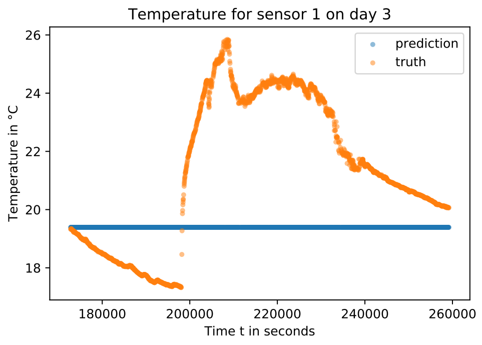
In Figures 3, 4 and Table 10, one can see that the predictions of this model are very poor. Indeed, it achieves very bad performances in terms of MSE. This is due to the fact that the prediction considers only the last value of the previous day, which is one of the coldest temperature within a day. Plus, this makes the Persistence model very sensitive to outliers.



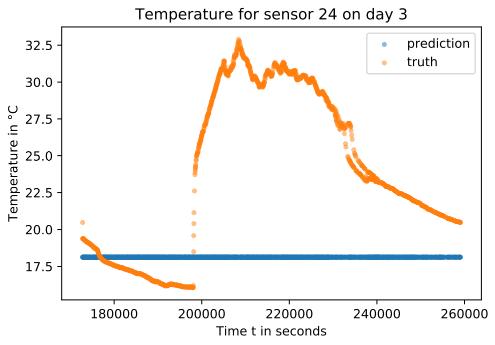
((a)) $MSE(sensor1, day2) = 469.2078$



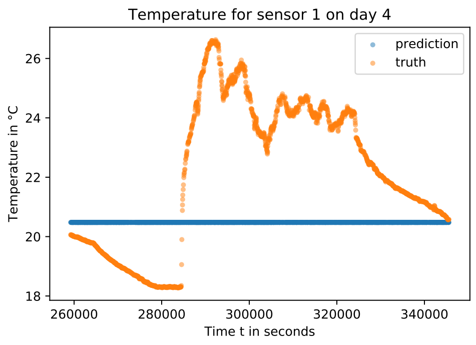
((b)) $MSE(sensor24, day2) = 50.0524$



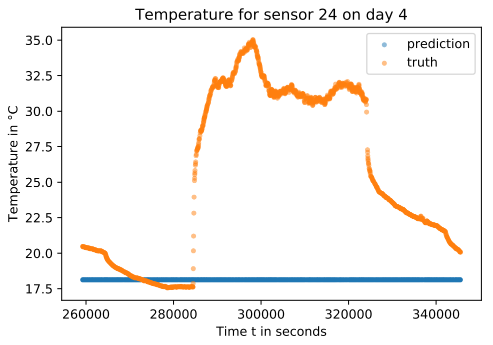
((c)) $MSE(sensor1, day3) = 10.0465$



((d)) $MSE(sensor24, day3) = 64.2188$

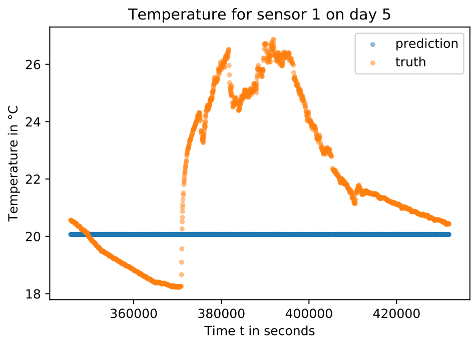


((e)) $MSE(sensor1, day4) = 8.7593$

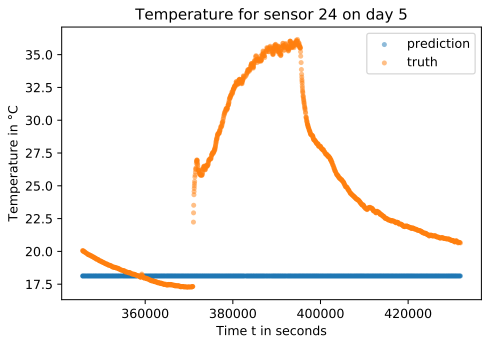


((f)) $MSE(sensor24, day4) = 89.2870$

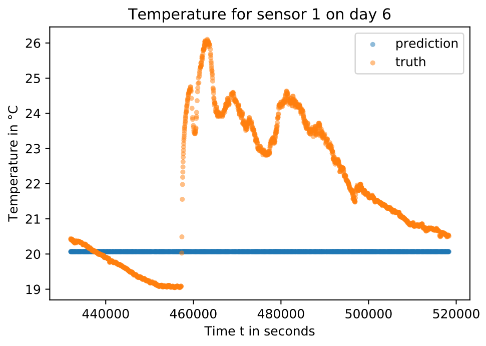
Figure 3: Temperature prediction of days 2 to 4 for Sensor 1 (left) and Sensor 24 (right) using the persistence model.



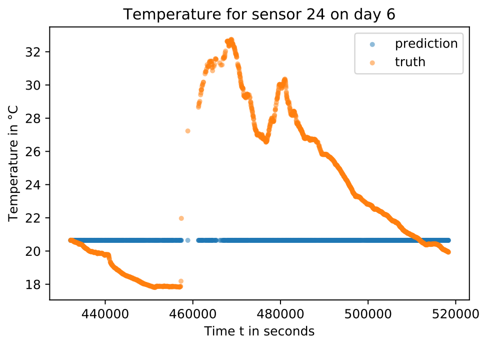
((a)) $MSE(sensor1, day5) = 10.6558$



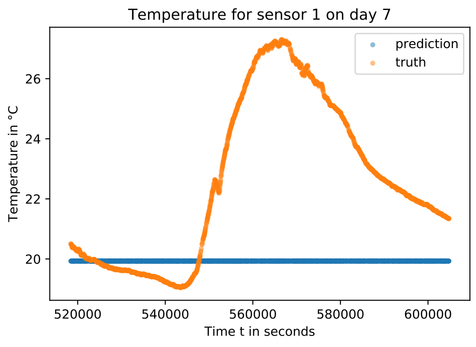
((b)) $MSE(sensor24, day5) = 75.6931$



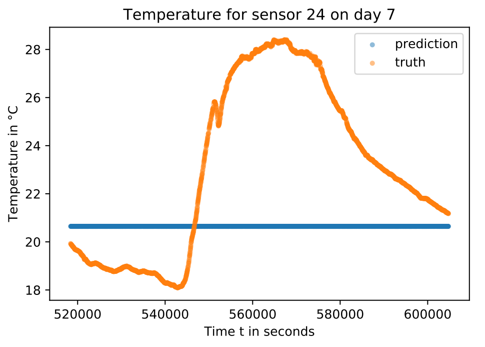
((c)) $MSE(sensor1, day6) = 6.9079$



((d)) $MSE(sensor24, day6) = 25.0309$

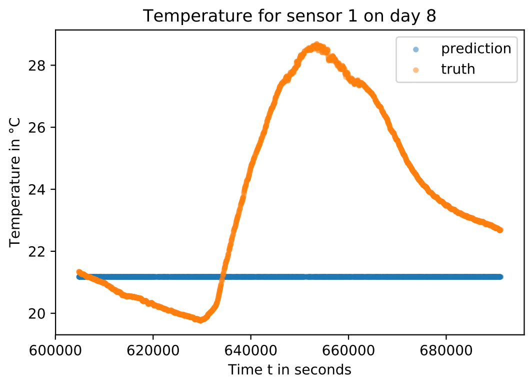


((e)) $MSE(sensor1, day7) = 14.2251$



((f)) $MSE(sensor24, day7) = 19.1663$

Figure 4: Temperature prediction of days 5 to 7 for Sensor 1 (left) and Sensor 24 (right) using the persistence model.



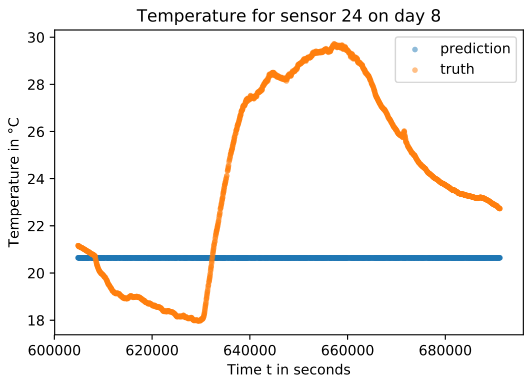


Table 10: $MSE(sensor1, day8) = 14.6868$; $MSE(sensor24, day8) = 27.6864$

4.2 LTI System optimized by RLS

4.2.1 Model

In order to circumvent the two major issues of the Persistence model, we have to design a model that depends on a larger set of past data. Moreover, the model should be aware of the noisy nature of the data. If we assume that the temperature evolution follows a linear model, the expected temperature data should look like the following:

$$T(t) = \sum_{k=1}^{m} \theta_k T(t-k) + e(t) \tag{2}$$

In equation 2, θ_k are the weights assigned to each previous temperature. Therefore the model assumes that the predicted temperature at time t is a linear function of a set of temperatures seen in the past. The function $e(t)$ models the error between the model and the measures. There are two main components that constitute the error: (i) The stochastic errors and (ii) the modeling errors. In the case of sensors, the stochastic errors are caused by the noise mainly generated by electronic devices. It is well known that the thermal noise present in temperature sensors can be considered as an Additive White Gaussian Noise (AWGN) process. Thus we can consider that the noise follows a Normal distribution $n \sim N(0, \sigma^2)$ where the noise power is modeled by the variance of the distribution σ^2. Following this model, we have to find the parameters θ_k that minimize the MSE between the predicted temperature $\hat{T}(t)$ and the true temperature $T(t)$. This corresponds to solving the following problem:

$$\underline{\hat{\theta}} = \underset{\underline{\theta}}{\text{argmin}} \sum_{i=1}^{t} \lambda^{t-i} [\hat{T}(i) - T(i)]^2 \tag{3}$$

Equation 3 can also be expressed by replacing $\hat{T}(t)$ by its vector notation:

- $\hat{T}(t) = \underline{X}^T(t)\underline{\hat{\theta}}$
- $\underline{\hat{\theta}} = \begin{bmatrix} \theta_1 & \theta_2 & ... & \theta_m \end{bmatrix}^T$

- $\underline{X}(t) = \begin{bmatrix} T(t-1) & T(t-2) & ... & T(t-m) \end{bmatrix}^T$

If we assume that the model is accurate enough with respect to the reality, then solving the optimization problem 3 maximizes the likelihood of the prediction since the error is mainly stochastic and thus Normally distributed around 0. However, in practice the model does not fit perfectly the reality, which implies that additional errors due to the modeling imprecision have to be taken into account. This implies that the RLS algorithm would be optimal in the likelihood sense only if the error was purely stochastic. Note that in equation 3, λ is a forgetting factor whose role is to give more importance to recent temperature as these are more correlated with the current temperature. This hyper-parameter should belong to the interval $[0, 1]$ and is usually set to 0.99. The optimal solution of the weighted least square optimization problem 3 is well known:

$$\underline{\hat{\theta}} = (\underline{\underline{X}}^T \Lambda \underline{\underline{X}})^{-1} \underline{\underline{X}}^T \Lambda \underline{T} \tag{4}$$

Where $\underline{T}$ is the vector composed of each $T(i), \forall i = 1, .., t$ and $\underline{\underline{X}}$ is the matrix $t \times m$ of the m previous temperature, at each time. However, we are interested in finding a recursive solution that adapts the parameters $\underline{\theta}$ in an online fashion. Hence, we have to express $\underline{\hat{\theta}}(t)$ as a function of the solution of the previous iteration $\underline{\hat{\theta}}(t-1)$. The recursive solution will therefore modify the previous parameters $\underline{\hat{\theta}}(t-1)$ to a new one $\underline{\hat{\theta}}(t)$ as new data arrive. This avoids recomputing the whole solution of the entire dataset at each iteration, which saves a lot of computation time. The updating rule of $\underline{\hat{\theta}}$ is written in equations 5 to 8.

$$\underline{\hat{\theta}}(t) = \underline{\hat{\theta}}(t-1) + \alpha(t)e(t) \tag{5}$$

$$e(t) = T(t) - \underline{X}^T(t)\underline{\hat{\theta}}(t-1) \tag{6}$$

$$\alpha(t) = V(t)\underline{X}^T(t) \tag{7}$$

$$V(t) = \frac{1}{\lambda}[V(t-1) - \frac{V(t-1)\underline{X}^T(t)\underline{X}(t)V(t-1)}{1 + \underline{X}(t)V(t-1)\underline{X}^T(t)}] \tag{8}$$

The estimated parameters $\underline{\hat{\theta}}$ evolve with time because the data are assumed to be fed in a streaming fashion. In order to apply this update rule, the values of $\underline{\hat{\theta}}$ and V have to be initialized. Usually, $\underline{\hat{\theta}}(0)$ is set to zero while $V(0) = aI$, with $a > 0$.

So far we designed an optimization algorithm that minimizes the MSE between the linear model 2 and the measured temperature for only one sensor. There are 2 ways of extending the current algorithm to a larger number N of sensors (or groups of sensors):

1. We simply learn the parameters of each sensor independently from one to each other. That means that we simply have to solve equation 3 N times and that each sensor will have its own set of parameters. The estimator of the temperature will thus be expressed as in equation 9, where each row is solved independently using RLS algorithm. The advantage of this method is that we take into account

the eventual **diversity** of each sensor by attributing them their own set of parameters. However this is at the cost of a larger computation time, or at the cost of larger computation resources if the algorithm is run in a distributed environment.

$$\hat{T}(t) = \begin{bmatrix} \underline{X_1}^T(t)\, \underline{\hat{\theta}_1} \\ \underline{X_2}^T(t)\, \underline{\hat{\theta}_2} \\ \dots \\ \underline{X_N}^T(t)\, \underline{\hat{\theta}_N} \end{bmatrix} \tag{9}$$

2. Another alternative is to jointly train one set of parameters $\underline{\hat{\theta}}$ for all the sensors. That means that the parameters $\underline{\hat{\theta}}$ will try to fit the whole set of sensors at once. This method consists of solving equation 3 with $\hat{T}(t)$ being expressed as in equation 10. This has the advantage of being much faster to compute compared to the first approach since only one optimization problem is solved. However we lose the **diversity** of the sensors in return since this approach will work only if the sensors behave similarly. Under this assumption, jointly training all the sensors will turn into an advantage because the parameters $\underline{\hat{\theta}}$ will become more **robust** to eventual aberrant measures. Indeed, if one sensor measure is abnormal, this will not influence too much the model assuming that the majority of the sensors behave normally. This **robustness** to malfunctioning sensors is not possible in equation 9.

$$\hat{T}(t) = \begin{bmatrix} \underline{X_1}^T(t) \\ \underline{X_2}^T(t) \\ \dots \\ \underline{X_N}^T(t) \end{bmatrix} \underline{\hat{\theta}} = \underline{\underline{X}}^T(t)\, \underline{\hat{\theta}} \tag{10}$$

Hence, we can conclude that there exists a **diversity-robustness** trade-off between these two approaches. However, a balance can be found between models 9 and 10. Indeed, let us consider K clusters in which sensors are spatially correlated one with respect to each other. We can then extend equation 10 to a set of K independent equations such as expressed in equation 9. Therefore, we obtain a system of K equations that can be solved independently using RLS algorithm. Therefore, each equation can be solved independently in a smaller distributed environment than in equation 9.

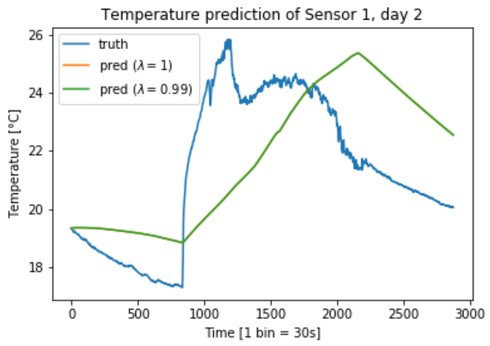
$$\hat{T}(t) = \begin{bmatrix} \underline{\underline{X_1}}^T(t)\, \underline{\hat{\theta}_1} \\ \underline{\underline{X_2}}^T(t)\, \underline{\hat{\theta}_2} \\ \dots \\ \underline{\underline{X_K}}^T(t)\, \underline{\hat{\theta}_K} \end{bmatrix} \tag{11}$$

To conclude, in equation 11 the diversity of each sensor is taken into account since a set of parameters $\underline{\hat{\theta}_i}$ is attributed to each cluster of sensors. The robustness to outliers is not lost since the parameters $\underline{\hat{\theta}_i}$ are updated by taking into account the measurements of all the sensors of cluster i. In terms of computation, the RLS algorithm has to be applied $K < N$ times.

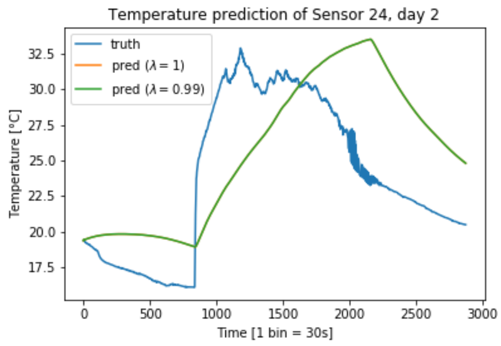
4.2.2 Results

In order to implement the LTI system, some pre-processing on the data are required. Indeed, the data do not come at a fixed interval of time. This implies that, the difference of time between $t-1$ and $t-2$ is not necessary the same as the difference between $t-2$ and $t-3$. This is an undesired effect since the algorithm exploits time correlation between temperatures. To circumvent this problem, the temperatures of each sensor have been rolled up into time bins of $30s$ as the sensors are supposed to provide 1 temperature data every $30s$. The temperature value of each time-bin has been set to the average temperature within it. The empty bins have been filled by interpolating the temperature values of the neighboring time-bins.
The predicted temperature using this model are illustrated in Figures 5 and 6. The prediction are made from day 3 until day 8 because the model needs at least 2 days of training. The reason of this will be explained later. One can see that the model is good for predicting the near future temperatures, however it is less suited to predict an entire day, as required in this project. Indeed, let us assume that the temperature of the last day is known and let us train this model using last day temperature. The first design choice that comes to our mind concerns the number of parameters to use. At first, we chose 10 parameters $\underline{\theta^{ST}} = \begin{bmatrix}\theta_1^{ST} & \theta_2^{ST} & ... & \theta_{10}^{ST}\end{bmatrix}^T$. Knowing that each bin of data comes each $30s$, the first parameter multiplies the aggregated temperature of last $30s$, the second one multiplies the aggregated temperature of last $60s$ and so on. Hence the model represents the prediction of the new temperature based on the past $5min$ data. When it comes to predict the next day, we can hopefully predict accurately the first $5min$ as the model is trained for this purpose. But after $10min$, we do not have access to previous known data anymore. Indeed, all we have is the last $5min$ of **predicted** temperature. Therefore the next predictions become poor in precision as time goes since the latter are based on uncertain data, which makes any error propagate and grow with time. The problem of this model is clear: it is a short-term memory model whose parameters only represent recent temperatures. A better solution would be to add long-term memory parameters that will use for example the data of the same clock time of previous day. Therefore, during the prediction, the model can learn its prediction based on both long-term and short-term temperatures. Such a model has been implemented to circumvent the short-memory issues explained above. Indeed, 10 additional parameters have been added to the model $\underline{\theta^{LT}} = \begin{bmatrix}\theta_1^{LT} & \theta_2^{LT} & ... & \theta_{10}^{LT}\end{bmatrix}^T$, which represents the temperature of the same $5min$ time window of previous day. The introduction of the long-term parameters $\underline{\theta^{LT}}$ explains why the temperature prediction can only start from day 3. The prediction model that is observed in Figures 5 and 6 uses both short-term and long-term parameters $\underline{\theta^{ST}}$ and $\underline{\theta^{LT}}$.

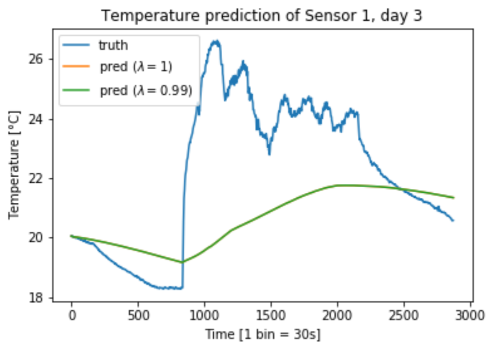
In Figures 5 and 6, we observe a low-complexity prediction on days 3 and 4. However, on days 5 and 6, the model begins to become more complex and overfits the entire prediction day using exclusively the previous day temperatures. On day 7, the overfitting effect is reduced, but this arises a completely wrong prediction for sensor 24. Finally, the parameters stabilizes and produce better results for day 8. This is again probably due to the fact that temperature curves of days 7 and 8 are very similar. However, unlike days 5 and 6, it did not completely overfit the previous day, which is good news. The conclusion of this model is that predicting temperature using only time correlation is very hard to achieve and produces good results in terms of MSE only if the temperature evolution of one day is similar to the previous ones.



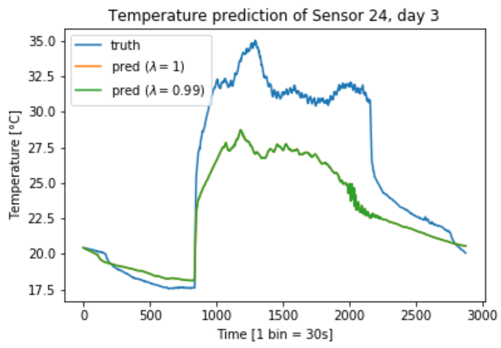
((a)) $MSE(sensor1, day3) = 6.562491$



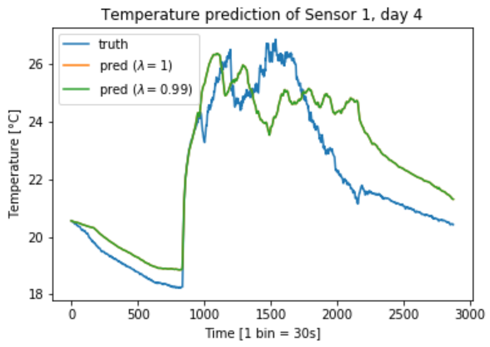
((b)) $MSE(sensor24, day3) = 27.236141$



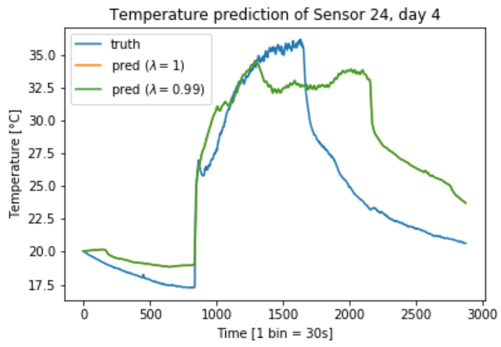
((c)) $MSE(sensor1, day4) = 7.385705$



((d)) $MSE(sensor24, day4) = 14.430116$

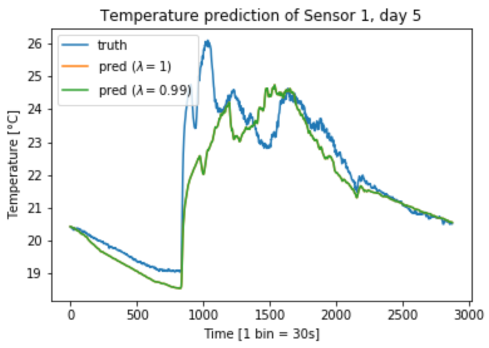


((e)) $MSE(sensor1, day5) = 1.706927$

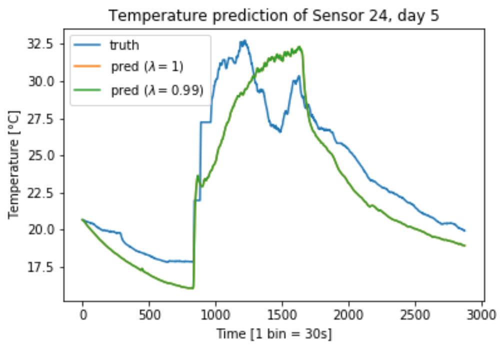


((f)) $MSE(sensor24, day5) = 15.896992$

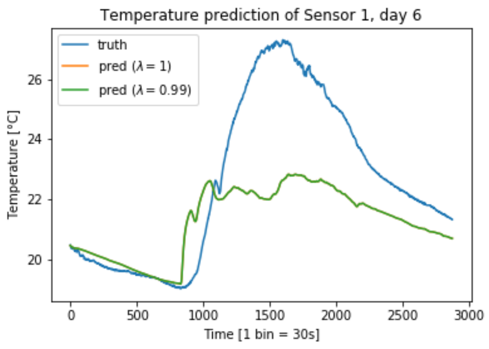
Figure 5: Temperature prediction of days 3 to 5 for Sensor 1 (left) and Sensor 24 (right) using LTI system optimized with RLS. The time scale is $1bin = 30s$



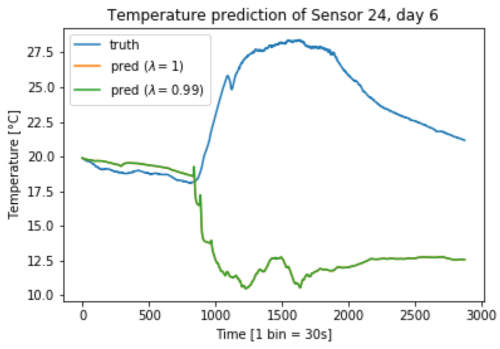
((a)) $MSE(sensor1, day6) = 0.829093$



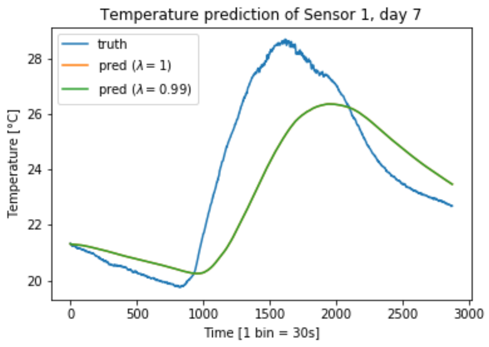
((b)) $MSE(sensor24, day6) = 5.007601$



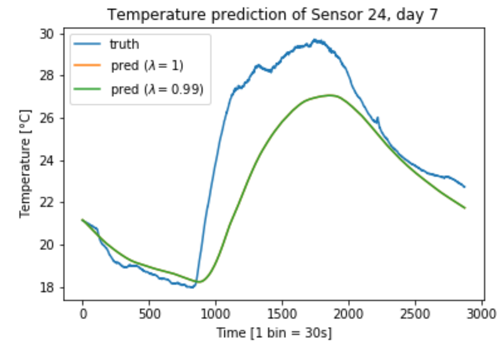
((c)) $MSE(sensor1, day7) = 5.062765$



((d)) $MSE(sensor24, day7) = 120.32363$



((e)) $MSE(sensor1, day8) = 3.756039$



((f)) $MSE(sensor24, day8) = 5.496232$

Figure 6: Temperature prediction of days 6 to 8 for Sensor 1 (left) and Sensor 24 (right) using LTI system optimized with RLS. The time scale is $1bin = 30s$. The final MSE are 0.944 for Sensor 1 and 0.467 for Sensor 24.

4.3 EMA

4.3.1 Model

In Section 4.2, we exploited temporal correlation between the measures to build a predictive model of the temperature. Another interesting prediction criterion to investigate is to exploit the spatial correlation between the temperature measured by each sensor to predict missing measures. A simple way to do it is to estimate the temperature of a sensor i as an exponential moving average temperature of its N neighboring sensors. Formally, this can be expressed as follows:

$$\hat{T}_i(t) = \alpha \overline{T_i}(t) + (1-\alpha)\hat{T}_i(t-1) \tag{12}$$

$$\overline{T_i}(t) = \frac{1}{N} \sum_{\substack{n=1 \\ n \neq i}}^{N} T_n(t) \tag{13}$$

This model is much more robust than model 4.2 since it uses the temperature of neighboring sensors measured at approximately the same time we want to predict. Indeed, the prediction of the sensor i is mainly determined by the measured temperature in its neighborhood at the same time t. The parameter $\alpha \in [0, 1]$ is introduced to balance the effect of the new prediction $\overline{T_i}(t)$ with respect to the last one $\hat{T}_i(t-1)$. Hence, the α parameter controls the model sensitivity to new predictions. A small value of α ensures robustness to outliers at the expense of introducing a lag in the temperature prediction when temperature moves fast. In this model, the value of α is fixed to 0.2. Another alternative would be to optimize this parameter in the least-square sense using RLS algorithm for example. An even smarter prediction model would filter out outliers before averaging. This ensures that corrupted data are not taken into account in the prediction model. A temperature measure can be detected as an outlier if it does not lie in a confidence interval. If we consider that the stochastic error generated by the sensors follows a Gaussian distribution, a good confidence interval is $[\mu - 2\sigma, \mu + 2\sigma]$. This model is quite simple to implement and has the advantage that it is does not require any training. The only question that remains is how to define the neighborhood of a sensor. The simplest way to do it is to cluster the sensors according to their spatial coordinates. This can be easily be done using a clustering algorithm such as K-means. Note that this method assumes that the temperature measures are spatially correlated, which is not true for all the applications. But, fortunately this assumption can be made for our case since it is about temperature sensors located in a city or buildings.

4.3.2 Results

The EMA algorithm has been implemented using the parameter $\alpha = 0.2$. The sensors have been clustered into 5 categories using K-means. The clusters are depicted in Figure 7. Although the clustering do not take into account the effect of the walls and other side-effects, it gives a good estimation of which sensors are spatially correlated. Hence, the prediction of Sensor 1 will use the temperature measures of sensors of the bottom-left cluster while the prediction of Sensor 24 will be based on the measures of the sensors belonging to the bottom-right cluster. The results of the EMA model are

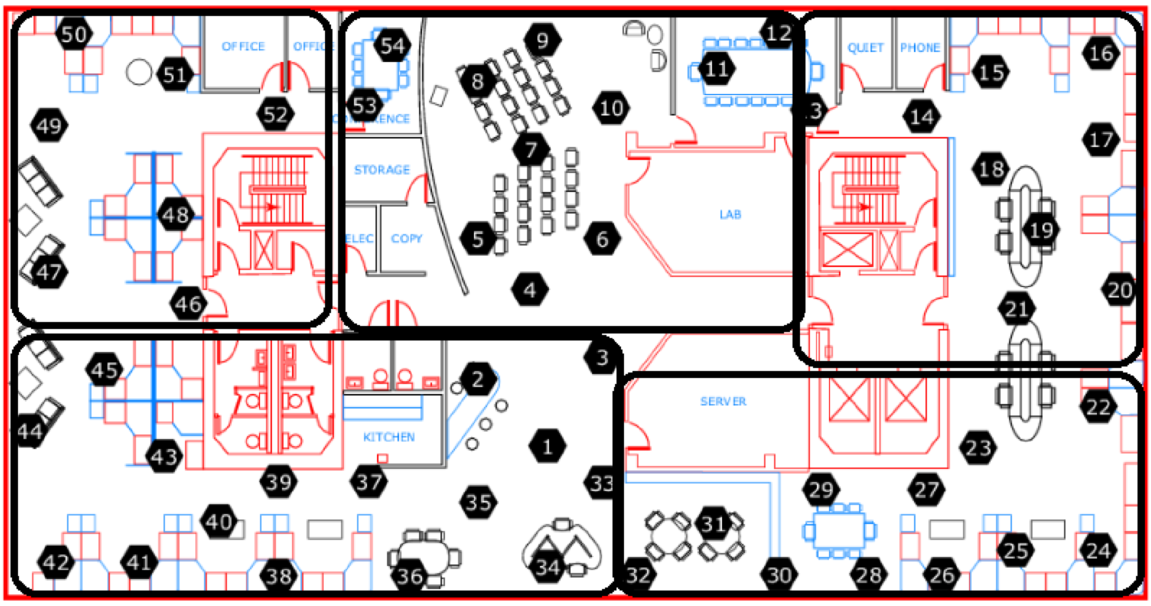
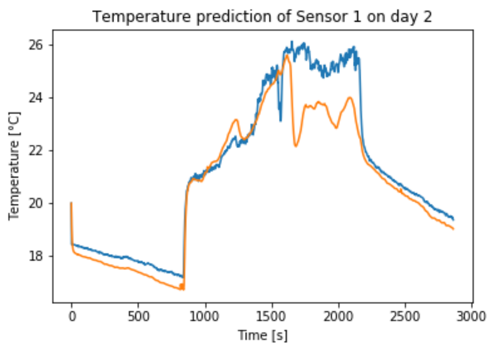
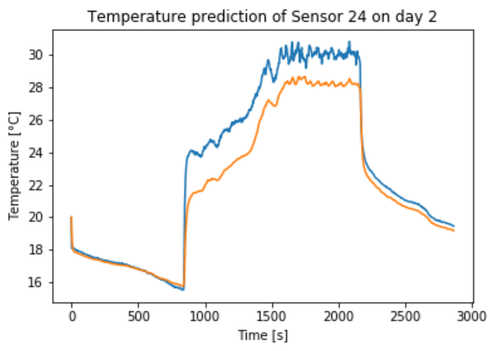


Figure 7: Spatial clustering of the sensors

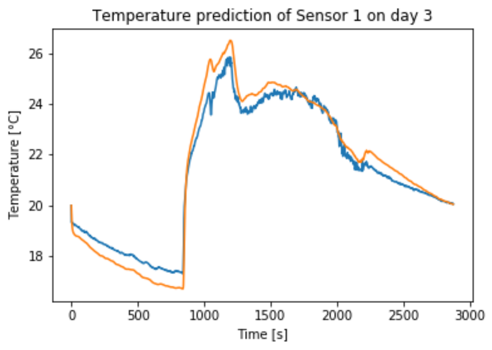
shown in Figures 8 and 9. One can see that the EMA algorithm achieves better results than LTI system (one order of magnitude better). This is expected since spatial correlation is a much more robust criterion than time correlation to evaluate the temperature. Indeed the MSE of day 8 are 0.944 and 0.467 for sensors 1 and 24 respectively. One important point to rise is the fact that the MSE does not improve while time goes because the EMA is not a learning algorithm. One can also observe the lagging effect of the parameter α which makes the prediction more difficult after a temperature overshooting, however this parameters reduces the error that would be introduced by corrupted measures.



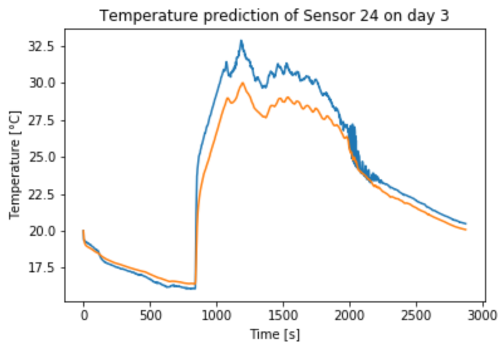
((a)) $MSE(sensor1, day2) = 0.965886$



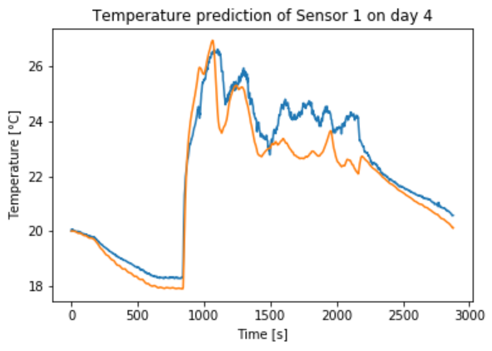
((b)) $MSE(sensor24, day2) = 2.119075$



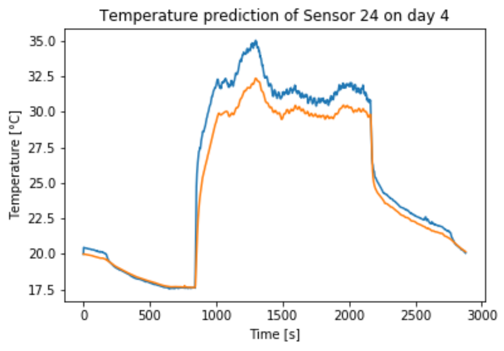
((c)) $MSE(sensor1, day3) = 0.189871$



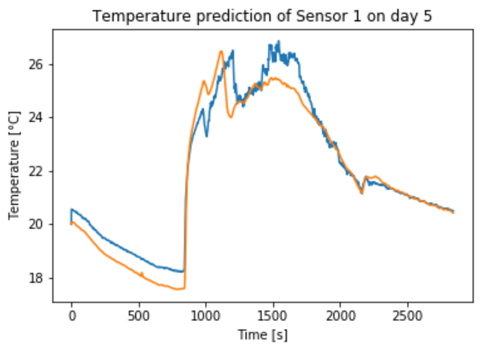
((d)) $MSE(sensor24, day3) = 1.438498$



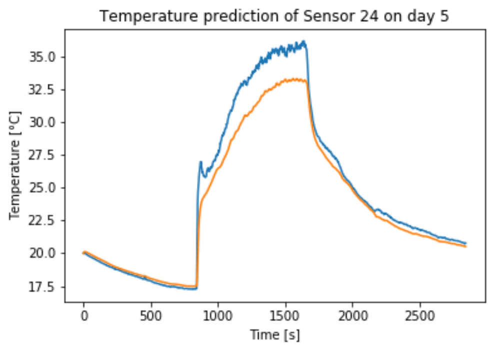
((e)) $MSE(sensor1, day4) = 0.612515$



((f)) $MS(sensor24, day4)E = 1.491121$

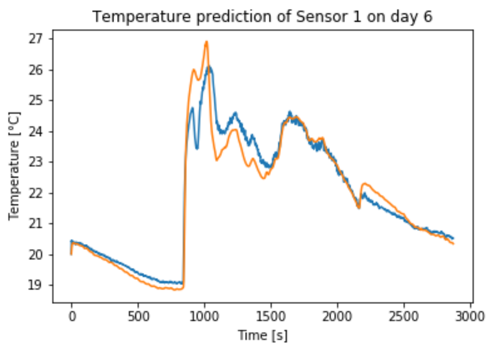


((g)) $MSE(sensor1, day5) = 0.388339$

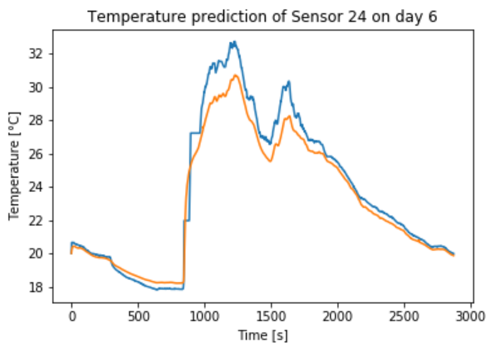


((h)) $MSE(sensor24, day5) = 1.789515$

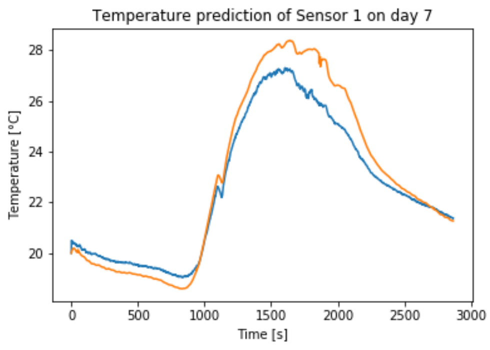
Figure 8: Temperature prediction of days 2 to 5 for Sensor 1 (left) and Sensor 24 (right) using EMA. The true temperature is in blue and the prediction is in orange. The time scale is $1bin = 30s$



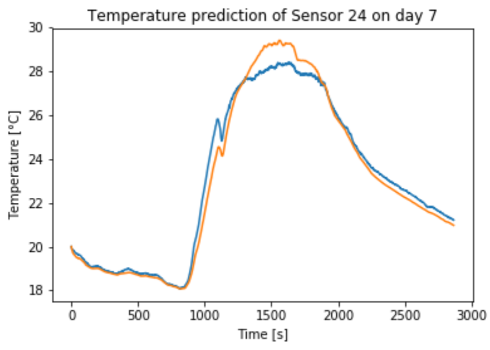
((a)) $MSE(sensor1, day6) = 0.216574$



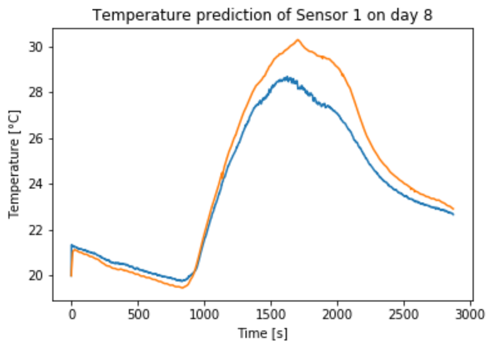
((b)) $MSE(sensor24, day6) = 0.808092$



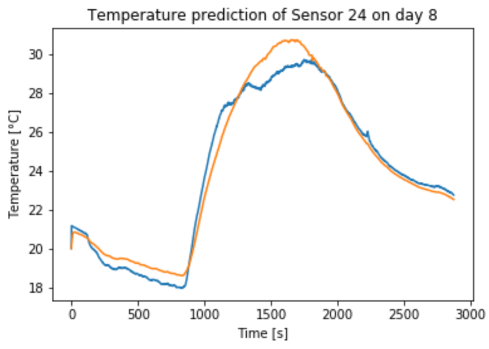
((c)) $MSE(sensor1, day7) = 0.578553$



((d)) $MSE(sensor24, day7) = 0.242776$



((e)) $MSE(sensor1, day8) = 0.944241$



((f)) $MSE(sensor24, day8) = 0.466780$

Figure 9: Temperature prediction of days 6 to 8 for Sensor 1 (left) and Sensor 24 (right) using EMA. The true temperature is in blue and the prediction is in orange. The time scale is $1bin = 30s$. The final MSE are 0.944 for Sensor 1 and 0.467 for Sensor 24.

5 Scalability Analysis

In order to assess the scalability of the RLS algorithm, multiple scalability metrics can be used. There are three different metrics when analysing the scalability of a system: speed-up, size-up and scale-up. Speed-up of course is not something we concern ourselves too much as it relates to the speed-up of execution time of an algorithm by adding more computational power to the system. Since we are mostly interested in handling large amounts of data we look at size-up and scale-up and consider speed-up only a sub-problem. Size-up represents the system's increase in execution time when adding more data and keeping the computational power constant. Scale-up is the combination of speed-up and size-up: we increase the amount of data to process but we also increase the computational power of the system in the same proportion as the data.

$$Size-up(m) = \frac{T(mN, p)}{T(N, p)} \tag{14}$$

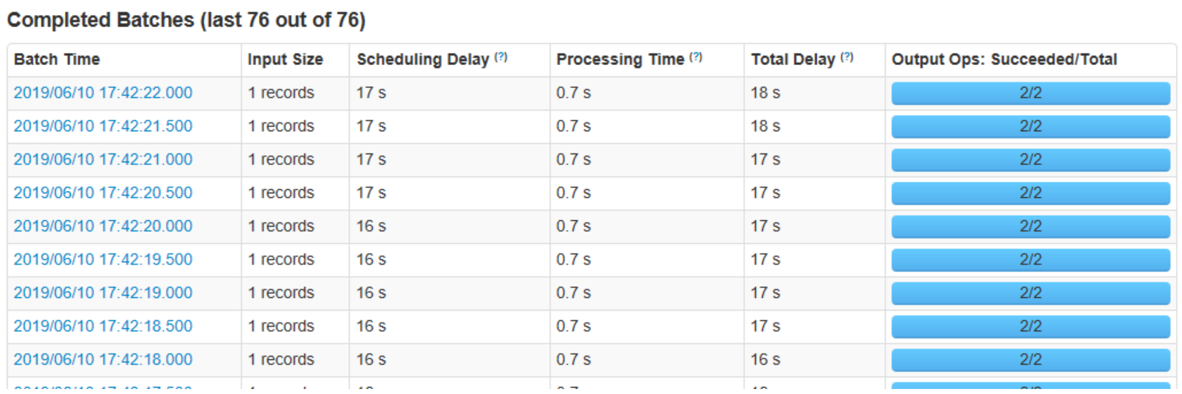
Where N is the size of data of the original system, m is the factor of increase of the data size in the larger system and p is the processing power (number of workers) and T() the execution time of a system.

$$Scale-up(m) = \frac{T(N, p)}{T(mN, mp)} \tag{15}$$

5.1 LTI system optimized with RLS

At first the scale-up of the RLS is analyzed. On the one hand, Figures 10 and 11, the process statistics are represented for a 2-node cluster receiving data from only one sensor. On the other hand, Figures 12 and 13 represent the same statistics for 2 sensors sending data to a 4-node cluster. No scaling-up appears in this comparison since the average processing time of the second experiment is approximately twice the one of the former. This can be explained by the fact that the number of records sent to the cluster is too low to observe the effect of the scale-up. Indeed, if the number of records is too low, the relative time spent by each process in communication is too high compared to their computation time. Hence the scale-up analysis does not show performance improvements.

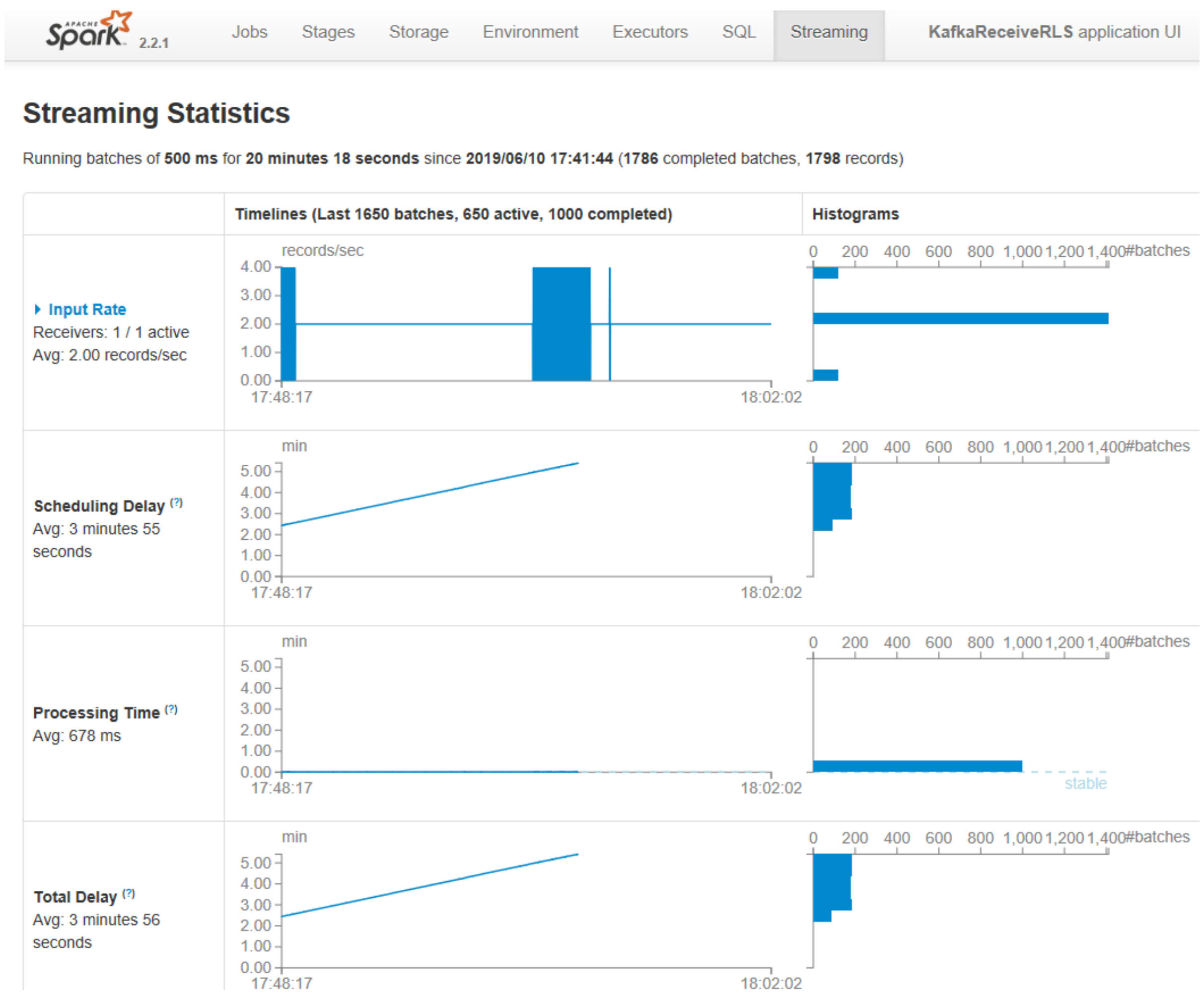
A size-up analysis of the RLS algorithm is now proposed by analyzing the statistics of a 2-node cluster running for 2 sensors, illustrated in Figures 14 and 15. When comparing with the results obtained in Figures 10 and 11 (2-node cluster and 1 sensor), it appears that the processing time is approximately the same even if the number of data to process has doubled. Again, these good performances are essentially due to the small number of records that are exchanged during the training of RLS. This small number of records per sensor is due to the fact that the training data are communicated as a vector and not one by one.



Completed Batches (last 76 out of 76)

Batch Time	Input Size	Scheduling Delay (?)	Processing Time (?)	Total Delay (?)	Output Ops: Succeeded/Total
2019/06/10 17:42:22.000	1 records	17 s	0.7 s	18 s	2/2
2019/06/10 17:42:21.500	1 records	17 s	0.7 s	18 s	2/2
2019/06/10 17:42:21.000	1 records	17 s	0.7 s	17 s	2/2
2019/06/10 17:42:20.500	1 records	17 s	0.7 s	17 s	2/2
2019/06/10 17:42:20.000	1 records	16 s	0.7 s	17 s	2/2
2019/06/10 17:42:19.500	1 records	16 s	0.7 s	17 s	2/2
2019/06/10 17:42:19.000	1 records	16 s	0.7 s	17 s	2/2
2019/06/10 17:42:18.500	1 records	16 s	0.7 s	17 s	2/2
2019/06/10 17:42:18.000	1 records	16 s	0.7 s	16 s	2/2

Figure 10: RLS algorithm tasks running time , in a 2-node Spark cluster with 1 sensor's data.

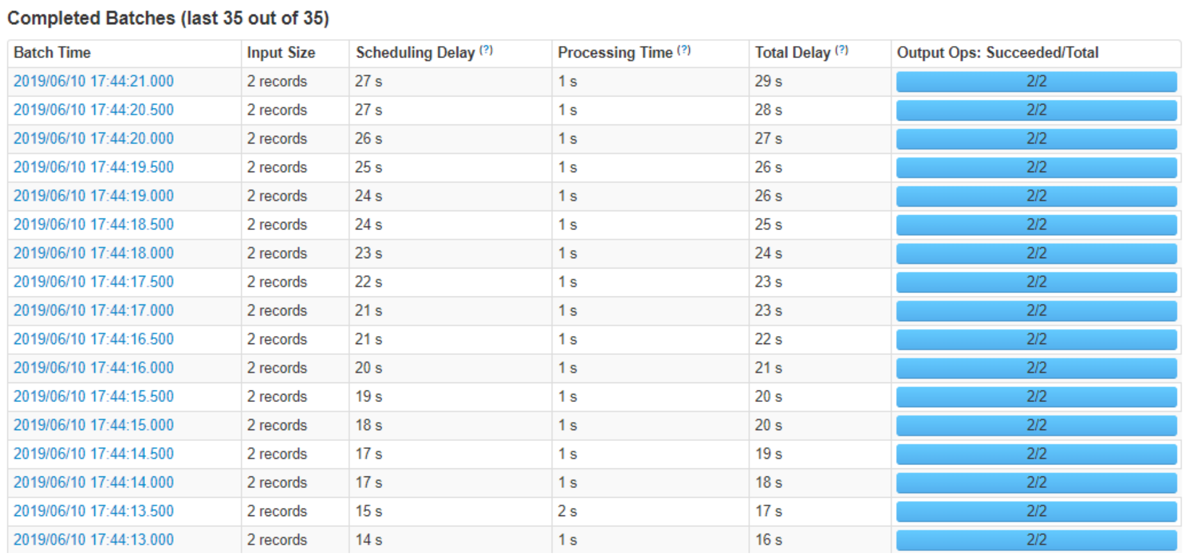


Streaming Statistics

Running batches of **500 ms** for **20 minutes 18 seconds** since **2019/06/10 17:41:44** (**1786** completed batches, **1798** records)

	Timelines (Last 1650 batches, 650 active, 1000 completed)	Histograms
Input Rate Receivers: 1 / 1 active Avg: 2.00 records/sec		
Scheduling Delay (?) Avg: 3 minutes 55 seconds		
Processing Time (?) Avg: 678 ms		
Total Delay (?) Avg: 3 minutes 56 seconds		

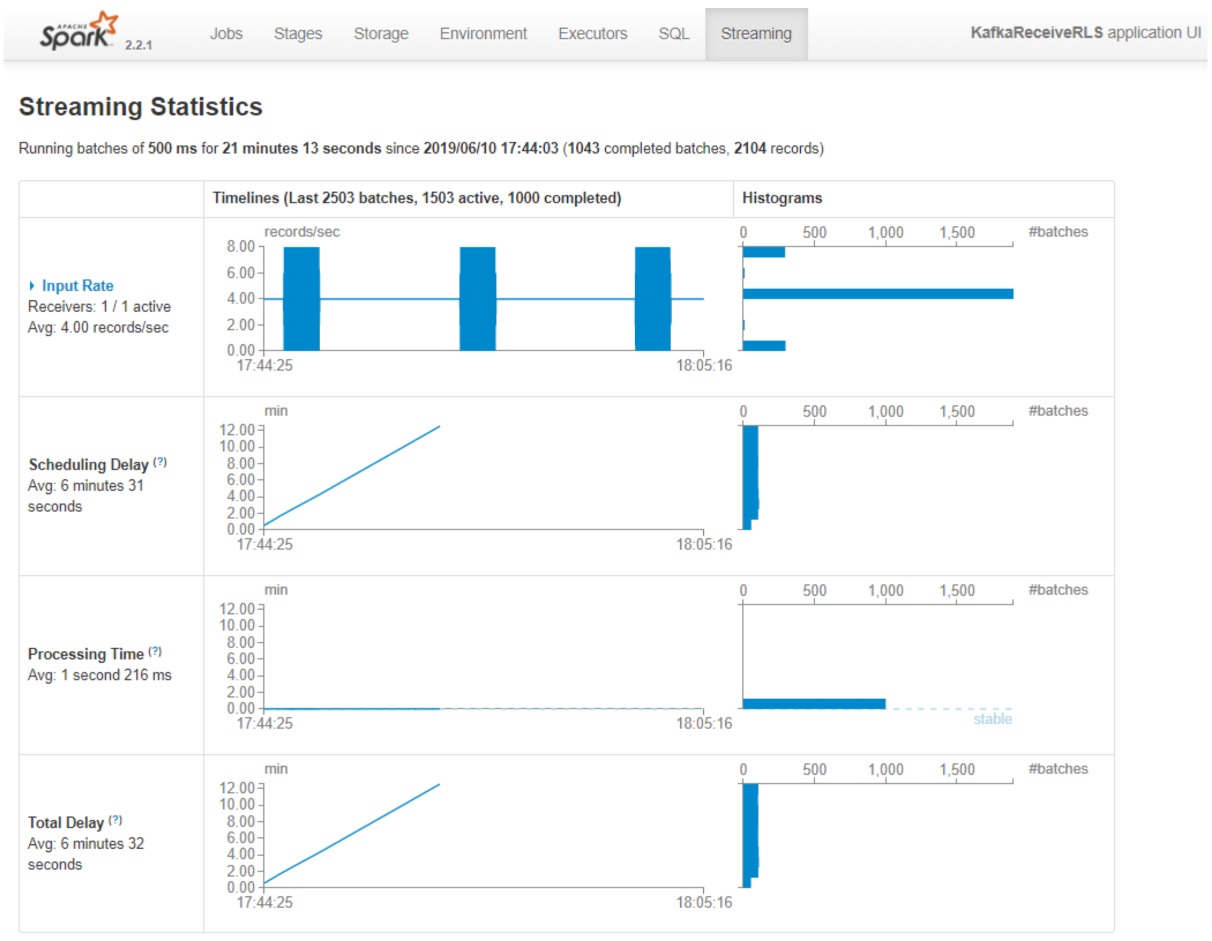
Figure 11: RLS algorithm tasks summary, in a 2-node Spark cluster with 1 sensor's data.



Completed Batches (last 35 out of 35)

Batch Time	Input Size	Scheduling Delay (?)	Processing Time (?)	Total Delay (?)	Output Ops: Succeeded/Total
2019/06/10 17:44:21.000	2 records	27 s	1 s	29 s	2/2
2019/06/10 17:44:20.500	2 records	27 s	1 s	28 s	2/2
2019/06/10 17:44:20.000	2 records	26 s	1 s	27 s	2/2
2019/06/10 17:44:19.500	2 records	25 s	1 s	26 s	2/2
2019/06/10 17:44:19.000	2 records	24 s	1 s	26 s	2/2
2019/06/10 17:44:18.500	2 records	24 s	1 s	25 s	2/2
2019/06/10 17:44:18.000	2 records	23 s	1 s	24 s	2/2
2019/06/10 17:44:17.500	2 records	22 s	1 s	23 s	2/2
2019/06/10 17:44:17.000	2 records	21 s	1 s	23 s	2/2
2019/06/10 17:44:16.500	2 records	21 s	1 s	22 s	2/2
2019/06/10 17:44:16.000	2 records	20 s	1 s	21 s	2/2
2019/06/10 17:44:15.500	2 records	19 s	1 s	20 s	2/2
2019/06/10 17:44:15.000	2 records	18 s	1 s	20 s	2/2
2019/06/10 17:44:14.500	2 records	17 s	1 s	19 s	2/2
2019/06/10 17:44:14.000	2 records	17 s	1 s	18 s	2/2
2019/06/10 17:44:13.500	2 records	15 s	2 s	17 s	2/2
2019/06/10 17:44:13.000	2 records	14 s	1 s	16 s	2/2

Figure 12: RLS algorithm tasks running time, in a 4-node Spark cluster with 2 sensor's data.

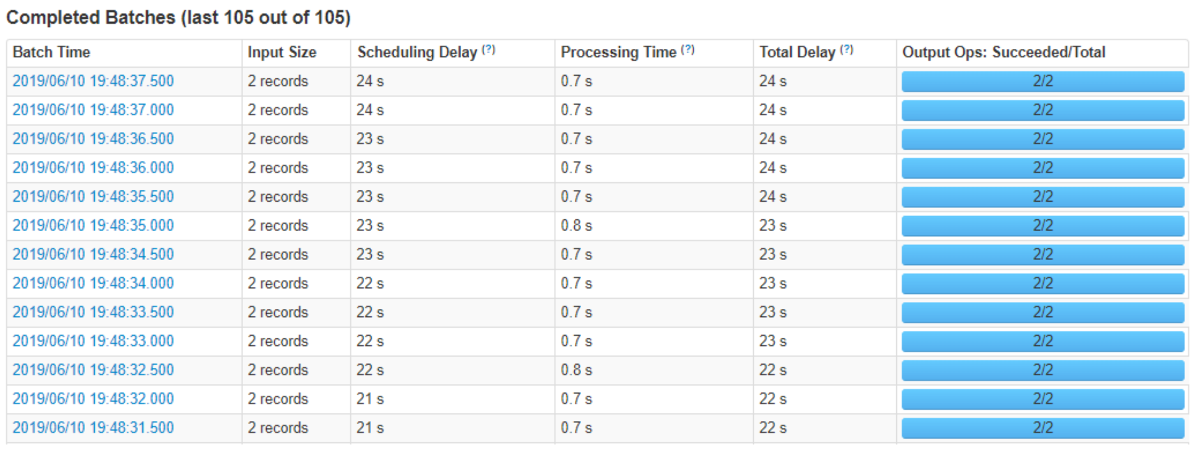


Streaming Statistics

Running batches of **500 ms** for **21 minutes 13 seconds** since **2019/06/10 17:44:03** (**1043** completed batches, **2104** records)

	Timelines (Last 2503 batches, 1503 active, 1000 completed)	Histograms
Input Rate Receivers: 1 / 1 active Avg: 4.00 records/sec		
Scheduling Delay (?) Avg: 6 minutes 31 seconds		
Processing Time (?) Avg: 1 second 216 ms		
Total Delay (?) Avg: 6 minutes 32 seconds		

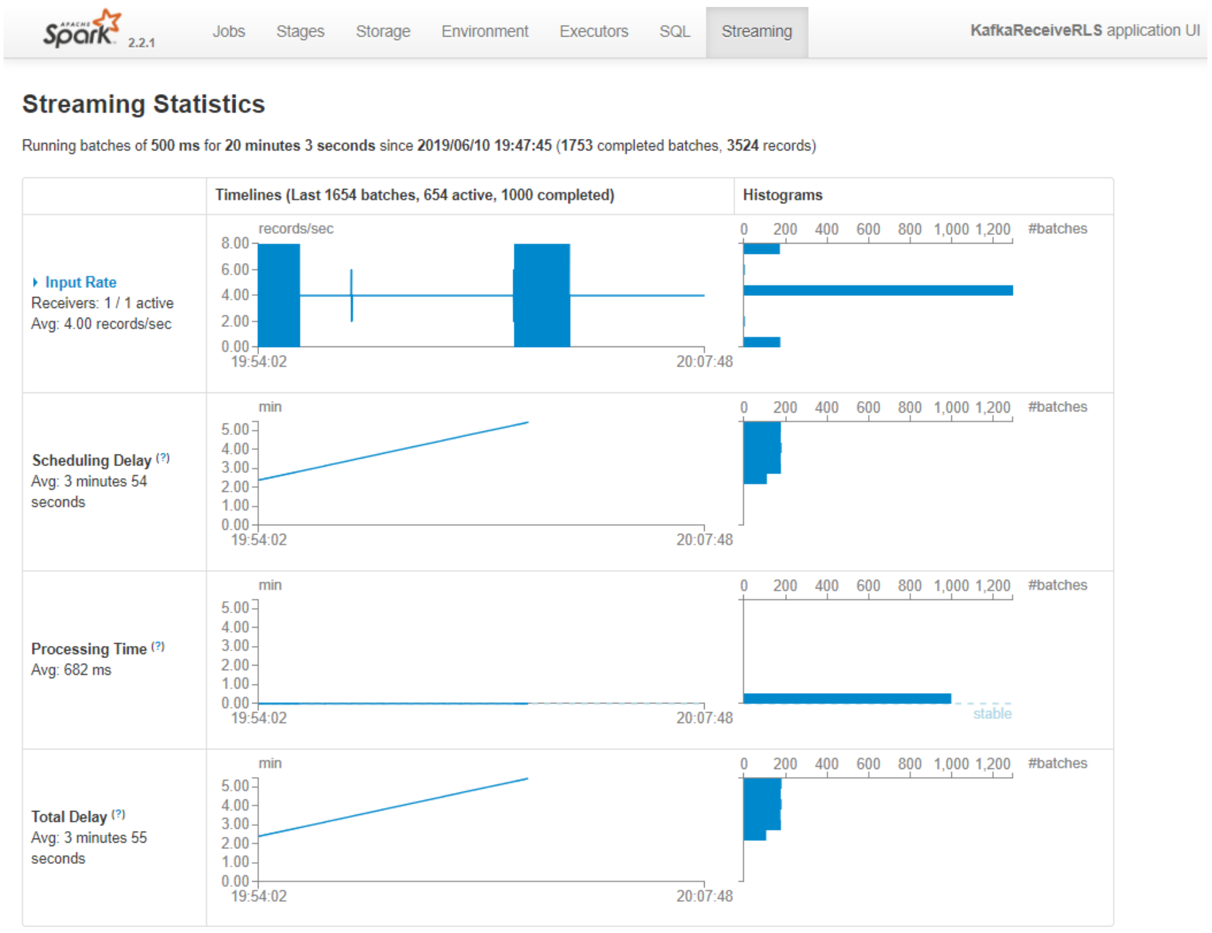
Figure 13: RLS algorithm tasks summary, in a 4-node Spark cluster with 2 sensors' data.



Completed Batches (last 105 out of 105)

Batch Time	Input Size	Scheduling Delay (?)	Processing Time (?)	Total Delay (?)	Output Ops: Succeeded/Total
2019/06/10 19:48:37.500	2 records	24 s	0.7 s	24 s	2/2
2019/06/10 19:48:37.000	2 records	24 s	0.7 s	24 s	2/2
2019/06/10 19:48:36.500	2 records	23 s	0.7 s	24 s	2/2
2019/06/10 19:48:36.000	2 records	23 s	0.7 s	24 s	2/2
2019/06/10 19:48:35.500	2 records	23 s	0.7 s	24 s	2/2
2019/06/10 19:48:35.000	2 records	23 s	0.8 s	23 s	2/2
2019/06/10 19:48:34.500	2 records	23 s	0.7 s	23 s	2/2
2019/06/10 19:48:34.000	2 records	22 s	0.7 s	23 s	2/2
2019/06/10 19:48:33.500	2 records	22 s	0.7 s	23 s	2/2
2019/06/10 19:48:33.000	2 records	22 s	0.7 s	23 s	2/2
2019/06/10 19:48:32.500	2 records	22 s	0.8 s	22 s	2/2
2019/06/10 19:48:32.000	2 records	21 s	0.7 s	22 s	2/2
2019/06/10 19:48:31.500	2 records	21 s	0.7 s	22 s	2/2

Figure 14: RLS algorithm, size-up for tasks running time, in a 2-node Spark cluster with 2 sensor's data.



Streaming Statistics

Running batches of **500 ms** for **20 minutes 3 seconds** since **2019/06/10 19:47:45** (**1753** completed batches, **3524** records)

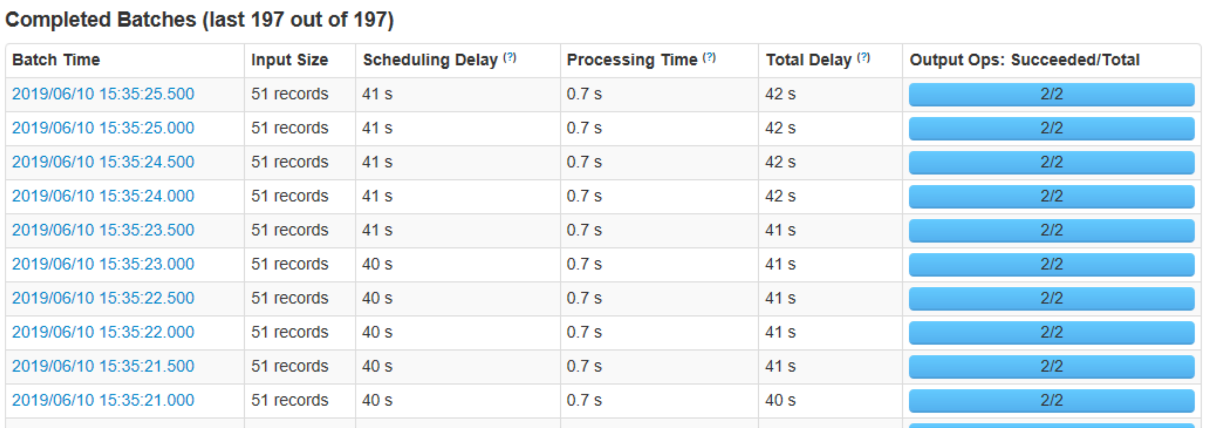
	Timelines (Last 1654 batches, 654 active, 1000 completed)	Histograms
Input Rate Receivers: 1 / 1 active Avg: 4.00 records/sec		
Scheduling Delay (?) Avg: 3 minutes 54 seconds		
Processing Time (?) Avg: 682 ms		
Total Delay (?) Avg: 3 minutes 55 seconds		

Figure 15: RLS algorithm, size-up tasks summary, in a 2-node Spark cluster with 2 sensor's data

5.2 EMA

We also perform a Scale-up and Size-up analysis for EMA. The Scale up analysis can be observed by looking at the Spark summary statistics in figure 17 in which we use a 2-node Spark cluster and 1 sensor data to perform the predictions and figure 19 in which we use 2 sensor data and 4 workers. We observe that EMA with 4 workers is a little less worse than twice with 2 workers. If we only observe the average we don't see EMA scaling up by adding more resources and data. However, when we observe figure 16 we see the time per task and the number of records it processes and we can notice that data in figure 18 is not twice as bigger but three times bigger. That means that the second sensor has added more data than the first one. We should then add a correction factor of 3/2 to the numerator in equation 15 and then we can say that EMA has scaled up successfully by adding data and resources.

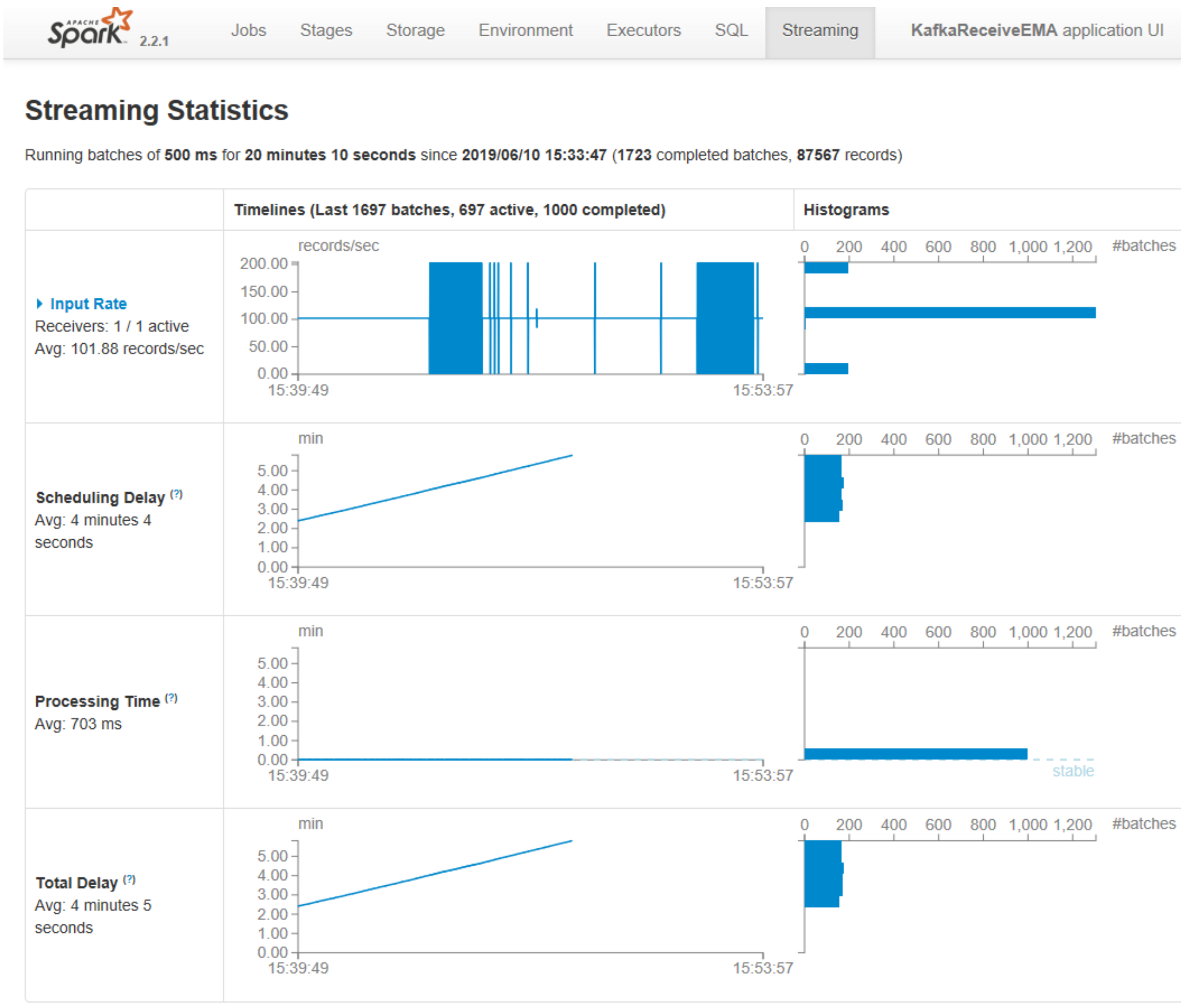
On the other hand, we have size-up EMA by just adding the extra sensor and no more resources. We can observe the statistical summary of the stream performance in figure 21 and we observe that the average time remains very similar to figure 17 which means that EMA has been also able of sizing up. When we observe figure 20 we can see that compared to 16 it's a little higher but still very close.



Completed Batches (last 197 out of 197)

Batch Time	Input Size	Scheduling Delay (?)	Processing Time (?)	Total Delay (?)	Output Ops: Succeeded/Total
2019/06/10 15:35:25.500	51 records	41 s	0.7 s	42 s	2/2
2019/06/10 15:35:25.000	51 records	41 s	0.7 s	42 s	2/2
2019/06/10 15:35:24.500	51 records	41 s	0.7 s	42 s	2/2
2019/06/10 15:35:24.000	51 records	41 s	0.7 s	42 s	2/2
2019/06/10 15:35:23.500	51 records	41 s	0.7 s	41 s	2/2
2019/06/10 15:35:23.000	51 records	40 s	0.7 s	41 s	2/2
2019/06/10 15:35:22.500	51 records	40 s	0.7 s	41 s	2/2
2019/06/10 15:35:22.000	51 records	40 s	0.7 s	41 s	2/2
2019/06/10 15:35:21.500	51 records	40 s	0.7 s	41 s	2/2
2019/06/10 15:35:21.000	51 records	40 s	0.7 s	40 s	2/2

Figure 16: EMA algorithm tasks running time, in a 2-node Spark cluster with 1 sensor's data.



Streaming Statistics

Running batches of **500 ms** for **20 minutes 10 seconds** since **2019/06/10 15:33:47** (**1723** completed batches, **87567** records)

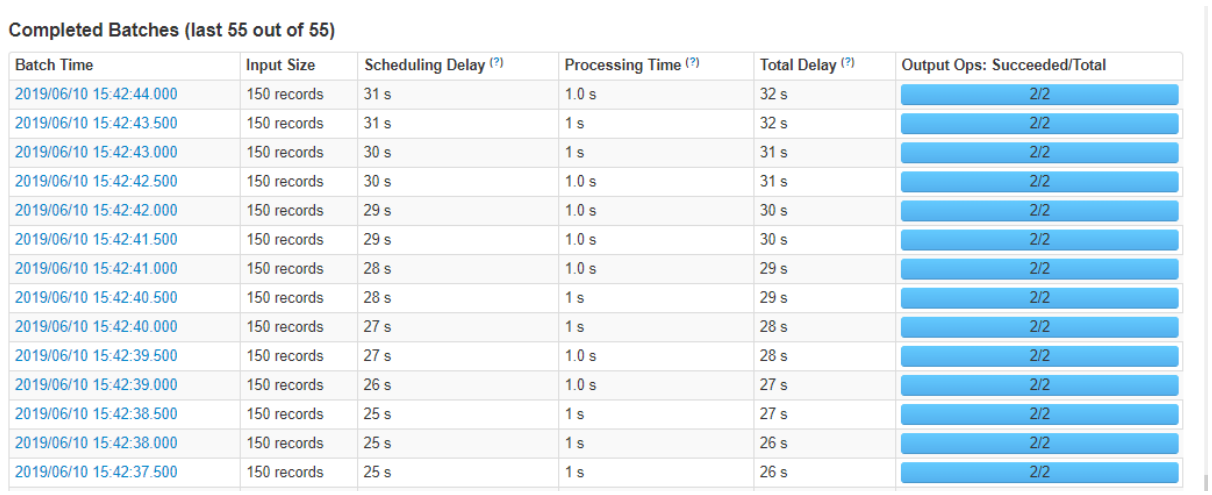
Input Rate
Receivers: 1 / 1 active
Avg: 101.88 records/sec

Scheduling Delay (?)
Avg: 4 minutes 4 seconds

Processing Time (?)
Avg: 703 ms

Total Delay (?)
Avg: 4 minutes 5 seconds

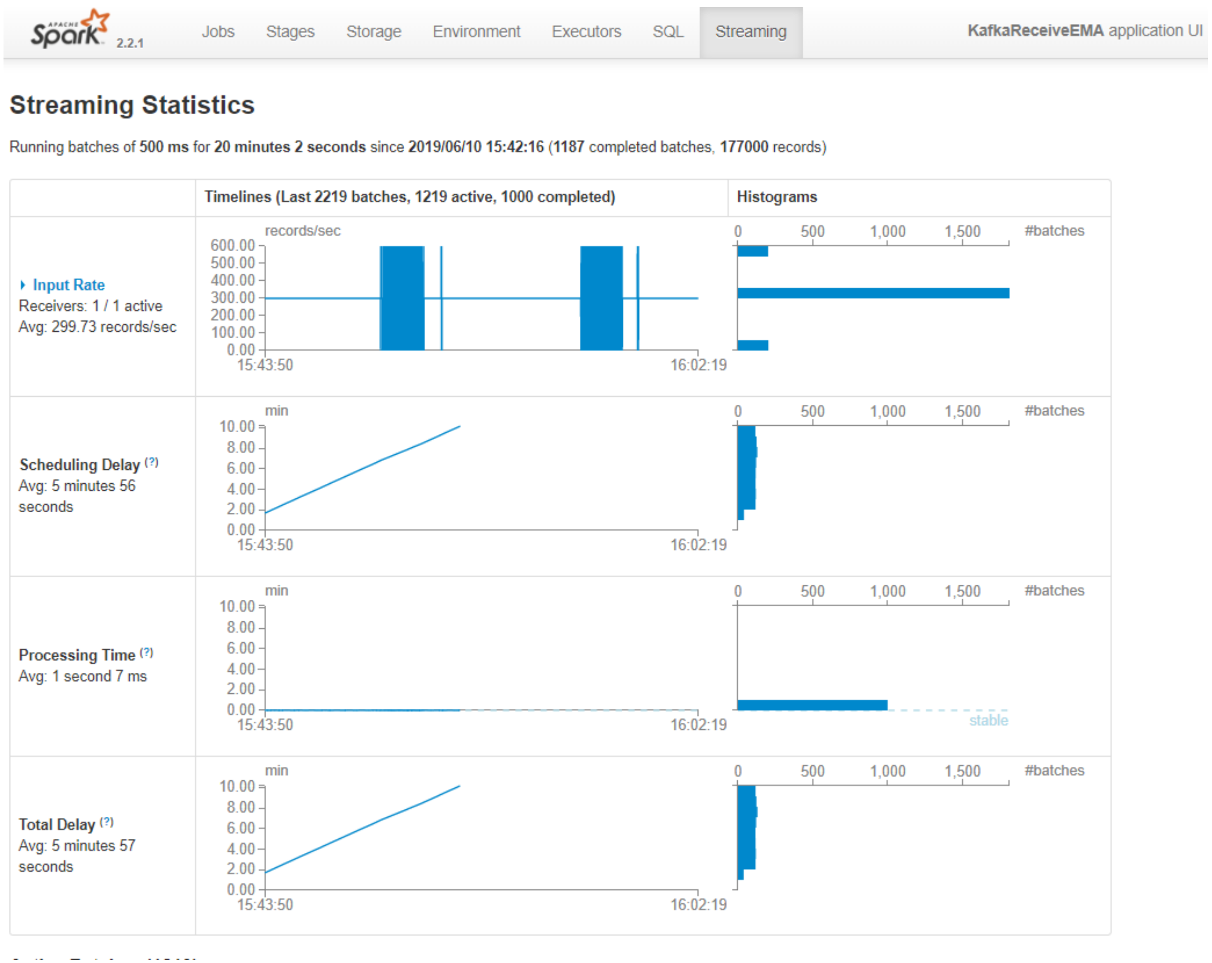
Figure 17: EMA algorithm tasks summary, in a 2-node Spark cluster with 1 sensor's data.



Completed Batches (last 55 out of 55)

Batch Time	Input Size	Scheduling Delay (?)	Processing Time (?)	Total Delay (?)	Output Ops: Succeeded/Total
2019/06/10 15:42:44.000	150 records	31 s	1.0 s	32 s	2/2
2019/06/10 15:42:43.500	150 records	31 s	1 s	32 s	2/2
2019/06/10 15:42:43.000	150 records	30 s	1 s	31 s	2/2
2019/06/10 15:42:42.500	150 records	30 s	1.0 s	31 s	2/2
2019/06/10 15:42:42.000	150 records	29 s	1.0 s	30 s	2/2
2019/06/10 15:42:41.500	150 records	29 s	1.0 s	30 s	2/2
2019/06/10 15:42:41.000	150 records	28 s	1.0 s	29 s	2/2
2019/06/10 15:42:40.500	150 records	28 s	1 s	29 s	2/2
2019/06/10 15:42:40.000	150 records	27 s	1 s	28 s	2/2
2019/06/10 15:42:39.500	150 records	27 s	1.0 s	28 s	2/2
2019/06/10 15:42:39.000	150 records	26 s	1.0 s	27 s	2/2
2019/06/10 15:42:38.500	150 records	25 s	1 s	27 s	2/2
2019/06/10 15:42:38.000	150 records	25 s	1 s	26 s	2/2
2019/06/10 15:42:37.500	150 records	25 s	1 s	26 s	2/2

Figure 18: EMA algorithm tasks running time, in a 4-node Spark cluster with 2 sensors' data.

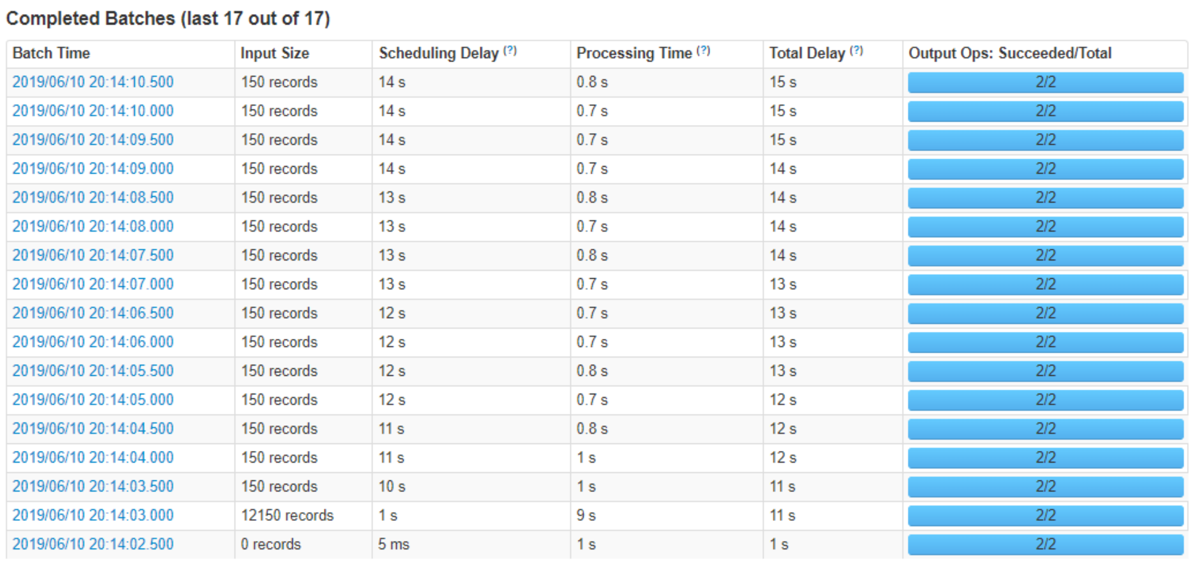


Streaming Statistics

Running batches of **500 ms** for **20 minutes 2 seconds** since **2019/06/10 15:42:16** (**1187** completed batches, **177000** records)

	Timelines (Last 2219 batches, 1219 active, 1000 completed)	Histograms
Input Rate Receivers: 1 / 1 active Avg: 299.73 records/sec		
Scheduling Delay (?) Avg: 5 minutes 56 seconds		
Processing Time (?) Avg: 1 second 7 ms		
Total Delay (?) Avg: 5 minutes 57 seconds		

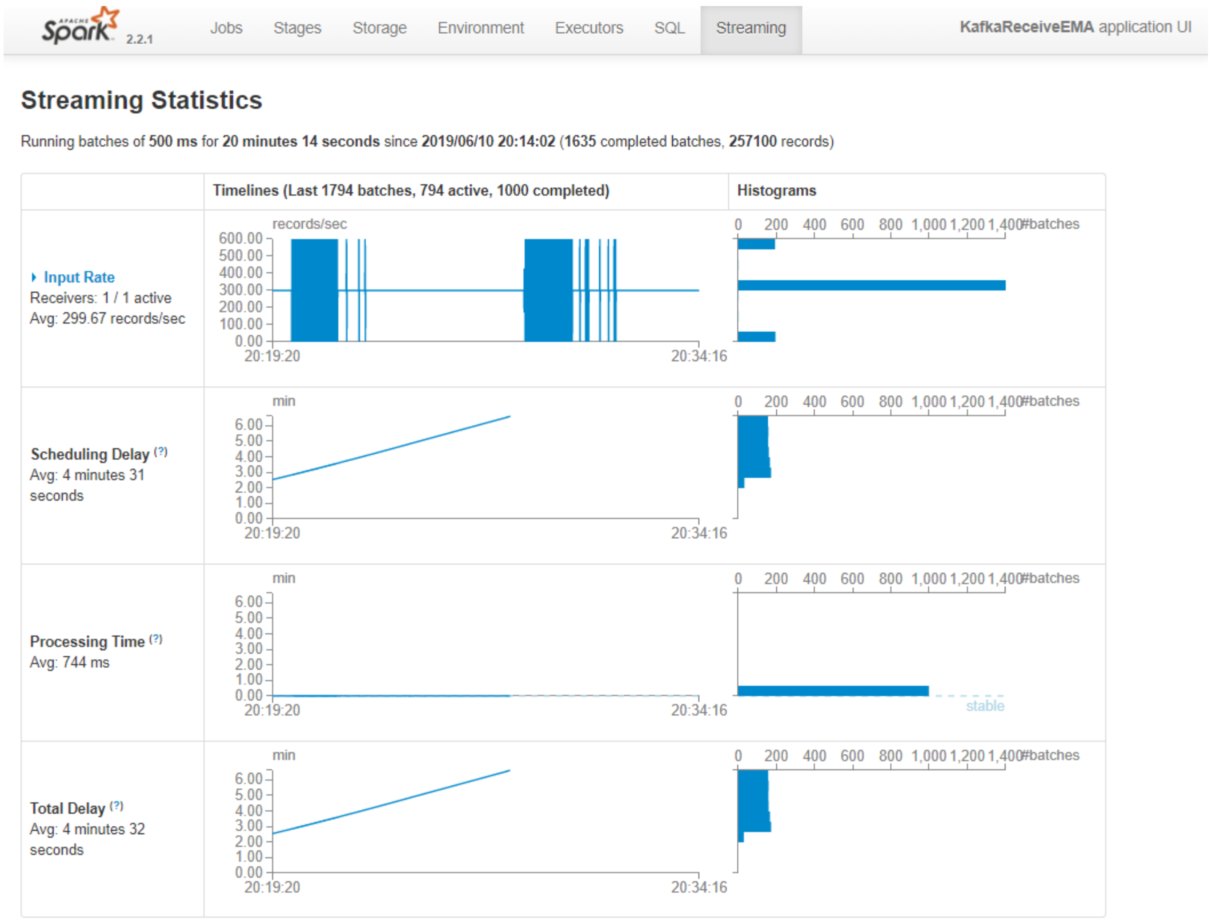
Figure 19: EMA algorithm tasks summary, in a 4-node Spark cluster with 2 sensors' data



Completed Batches (last 17 out of 17)

Batch Time	Input Size	Scheduling Delay (?)	Processing Time (?)	Total Delay (?)	Output Ops: Succeeded/Total
2019/06/10 20:14:10.500	150 records	14 s	0.8 s	15 s	2/2
2019/06/10 20:14:10.000	150 records	14 s	0.7 s	15 s	2/2
2019/06/10 20:14:09.500	150 records	14 s	0.7 s	15 s	2/2
2019/06/10 20:14:09.000	150 records	14 s	0.7 s	14 s	2/2
2019/06/10 20:14:08.500	150 records	13 s	0.8 s	14 s	2/2
2019/06/10 20:14:08.000	150 records	13 s	0.7 s	14 s	2/2
2019/06/10 20:14:07.500	150 records	13 s	0.8 s	14 s	2/2
2019/06/10 20:14:07.000	150 records	13 s	0.7 s	13 s	2/2
2019/06/10 20:14:06.500	150 records	12 s	0.7 s	13 s	2/2
2019/06/10 20:14:06.000	150 records	12 s	0.7 s	13 s	2/2
2019/06/10 20:14:05.500	150 records	12 s	0.8 s	13 s	2/2
2019/06/10 20:14:05.000	150 records	12 s	0.7 s	12 s	2/2
2019/06/10 20:14:04.500	150 records	11 s	0.8 s	12 s	2/2
2019/06/10 20:14:04.000	150 records	11 s	1 s	12 s	2/2
2019/06/10 20:14:03.500	150 records	10 s	1 s	11 s	2/2
2019/06/10 20:14:03.000	12150 records	1 s	9 s	11 s	2/2
2019/06/10 20:14:02.500	0 records	5 ms	1 s	1 s	2/2

Figure 20: EMA algorithm, size-up for tasks running time, in a 2-node Spark cluster with 2 sensor's data.



Streaming Statistics

Running batches of **500 ms** for **20 minutes 14 seconds** since **2019/06/10 20:14:02** (**1635** completed batches, **257100** records)

	Timelines (Last 1794 batches, 794 active, 1000 completed)	Histograms
Input Rate Receivers: 1 / 1 active Avg: 299.67 records/sec		
Scheduling Delay (?) Avg: 4 minutes 31 seconds		
Processing Time (?) Avg: 744 ms		
Total Delay (?) Avg: 4 minutes 32 seconds		

Figure 21: EMA algorithm, size-up tasks summary, in a 2-node Spark cluster with 2 sensor's data

6 Future work

For future work we suggest improving the models used but also investigate a combination of temporal and spatial correlation. As of now we did not combine said approaches but consider it a way of making a more robust model as one decreases the amount of points of failures. At any time even neighbouring sensors can fall out and one is left with fewer of them to predict current temperatures, but using temporal knowledge one might be able to assure a certain level of certainty for predictions made. Of course one should make sure as to not make the model too complex as this causes an increase in training time and also one generally prefers simple models to complex ones.

Another possible aspect to look into is the use of models like regressors to make predictions for the problem at hand. Having experimented a bit with a Multi-layer Perceptron (MLP) regressor using the first day as training data we achieved good results for predictions, however it was not done in online-fashion since new data was not incorporated in improving the model. Online approaches for improving 'classical' models on the fly could solve a range of problems. Currently some models allow to perform a partial fit using new data, but this partial fit only performs one epoch of training [1], meaning that there is no guarantee of it actually finding a minimum of the cost function. There also is no guarantee that it is scalable, but given the good performance of such a predictor in general should be enough reason to check the viability of feed-forward networks. Currently, the spark machine learning api does not provide an implementation of a MLP regressor, only one for a classifier [2].

7 Conclusion

We present two models, a linear time-invariant (LTI) system optimized by RLS and an exponential moving average (EMA) model, apart from the persistence model. While the persistence model obviously fails to predict any sensible information, it serves as a base line for models to improve on.

The LTI model takes into consideration temperature readings from the past but struggles to effectively predict longer periods of time as it is basically a short-term memory model using only recent readings to make predictions. To resolve this short-term memory problem we introduced an additional set of parameters to link to the same time window of the previous day. Results show that it is very prone to overfitting as it only makes decent predictions when there is not a major change in readings between consecutive days. We thus conclude that using a purely temporal model does not do a good enough job to be considered in the future.

We then moved on to a model using spatial correlation between sensors to predict the temperature of the sensors we are interested in. The idea behind this approach is that since sensors share the same space, their readings should be similar and if one sensor does not broadcast any reading, using neighbouring sensors we should be able to infer the missing value. Results show that it achieves better results than the LTI

[1]https://scikit-learn.org/stable/modules/generated/sklearn.neural$_n$*etwork.MLPRegressor.htmlsklearn.neural*$_n$*etwo*

[2]https://spark.apache.org/docs/latest/ml-classification-regression.html

system and the *MSE* is consistently low, especially for sensor 1. We thus conclude that spatial correlation is a more robust approach to predict missing values.

We also analysed the scalability of our models and come to the conclusion that while EMA shows to be scalable, for the LTI model we lack enough proof to confidently say that one can scale it up.

References

[1] N. Marz and J. Warren, *Big data: principles and best practices of scalable real-time data systems*. Shelter Island, NY: Manning, 2015, oCLC: ocn909039685.

[2] "Apache Kafka." [Online]. Available: https://kafka.apache.org/quickstart

[3] "Welcome to Apache Flume — Apache Flume." [Online]. Available: https://flume.apache.org/

[4] "HDFS Architecture Guide." [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

[5] "Spark Streaming | Apache Spark." [Online]. Available: https://spark.apache.org/streaming/

[6] "OpenTSDB - A Distributed, Scalable Monitoring System." [Online]. Available: http://opentsdb.net/

[7] "Grafana - The open platform for analytics and monitoring." [Online]. Available: https://grafana.com/

[8] "Apache flume - The open platform ingesting data." [Online]. Available: https://github.com/apache/flume/

[9] "Apache flume - Fetching data." [Online]. Available: https://www.tutorialspoint.com/

[10] "How to build a big data pipeline." [Online]. Available: https://dzone.com/