Real time change point detection by incremental PCA in large scale sensor data

Dmitry Mishin, Kieran Brantner-Magee, Ferenc Czako and Alexander S. Szalay
Department of Physics and Astronomy, Johns Hopkins University
Baltimore, Maryland 21218
Email: dmitry@pha.jhu.edu, kieran@pha.jhu.edu, fczako1@jhu.edu, szalay@jhu.edu

***Abstract*—The article describes our work with the deployment of a 600-piece temperature sensor network, data harvesting framework, and real time analysis system in a Data Center (hereinafter DC) at the Johns Hopkins University. Sensor data streams were processed by robust incremental PCA and K-means clustering algorithms to identify outlier and changepoint events. The output of the signal processing system allows us to better understand the temperature patterns of the DataCenter's inner space and make possible the online detection of unusual transient and changepoint events, thus preventing hardware breakdown, optimizing the temperature control efficiency, and monitoring hardware workloads.**

I. Introduction

A common issue when designing or managing a datacenter involves controlling the layout for maximizing the amount of hardware that can be used safely without creating a thermal overload for the hardware or the DC AC systems. As important as this topic is, or perhaps due to its importance, we found few detailed, real life temperature datasets for large datacenters, fewer that had not been elided to preserve trade secrets.

Therefore we set out to build a large scale thermal sensor data harvesting architecture in our data center in order to study the temperature distribution patterns during different conditions. We also undertook developing hardware and software environment to process our data, and explored the possibilities of automatic detection of changepoints or atypical time series events in very high dimensional sensor data streams.

The layout of nearly 600 temperature sensors in a 3'x3' grid located on the DC walls and the server racks polled at approximately twice a minute provides a relatively fine granularity of sensor data across most of the physical volume of the datacenter. In order to accomodate certain "edge case events" that might provide the most useful data, we designed the sensor data harvesting architecture to be robust against typical failure modes.

The resulting time series data has been stored in MSSQL and further processed by our custom developed program in IBM InfoSphere Streams framework.

We have looked for and applied an algorithm, that should learn the typical system behavior and report the outlier time points as they occur. In order to be computationally feasible we implemented a robust incremental PCA algorithm for reducing the dimensionality of the input data vector to the minimum possible number of dimensions, which still allows us to find the changes in a subset of monitored sensors. The K-Means algorithm was used to cluster the typical system conditions and detect the unexpected events by monitoring the appearance of new condition clusters. By projection of the data input points trajectory into the subspace spanned by the eigenvectors transient events can easily be identified visually. By summing up the distances of a given input point from the clusters centers as a function of time, a one dimensional quantity was generated to further facilitate changepoint detection.

II. Architecture

A. Sensor network layout

We set out initially to instrument the most core surfaces of the datacenter, the server racks themselves. Utilizing approximately 220 One Wire sensors connected by cat5e cable in 9 sensor long strings, we instrumented every other rack by placing a sensor at the bottom, middle, and top of both the hot and cold side of the rack, as well as underneath the floor on both sides, and a final sensor on the top. This resulted in a 3'x3' grid of sensors across the surfaces of the racks. The network was expanded after the prototype initial deployment to the datacenter walls to contain nearly 600 sensors total, to be polled approximately every 20 seconds. Since these sensors do not poll themselves a set of "harvesting nodes" and power circuitry needed to be constructed (Figure 1).

We decided that three FitPC's would be able to drive the rack deployment. This put between 8-10 sensor strings, connected by USB One Wire ethernet to USB busses to USB hubs, to each harvesting node. We observed that too many strings, as well as too many sensors on each string, increased not only the obvious time needed to poll the network, but also increased the occurance rate of various hardware failure modes. The nodes, and anything below them, are entirely unlinked from each other and can act independently. These machines are on custom power circuitry to allow them to be robust against full DC emergency power outages for up to 10 hours.

B. Data harvesting

The methods for polling the data properly faced a somewhat different set of constraints. Primarily, it had to function as completely as possible with as little hardware "up" as possible. Broken sensors, buggy sensor busses, unreliable FitPC boot device hardware, and potential power/network outages were all considered regular occurances. At its core, the software, known as OWtool, attempts to perform two tasks. First, it provides a framework for polling multiple sensor streams in parallel,

978-1-4799-6233-4/14/$31.00 ©2014 IEEE

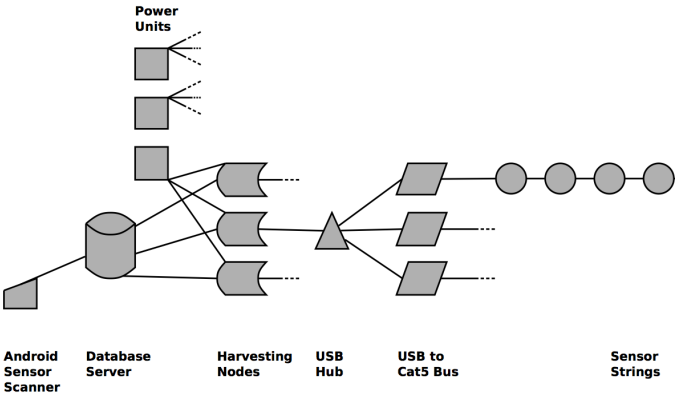

Fig. 1. The data harvesting architecture

attempting to catch and handle or recover from failures at the sensor or string level as they occur without impacting data from other sensors on that string, or other strings. Secondly, it attempts to buffer data locally if whatever sinks it is sending data to are offline. Written in Python, this codebase eventually expanded to include a wrapper around pyserial simply to interact more effectively with the sensor strings, attempting to encapsulate certain failures more cleanly, provide a more clean API, and work around or catch failure modes which occasionally violate the OneWire USB to ethernet bus spec.

At a high level, the software spins off a thread for each sensor string, polling its string according to various parameters/timings specified at launch, attempting to gather data from each sensor, and accumulating it in buffer files. When it can communicate with its data sinks, it attempts to regularly drain its buffer files into the sinks, with very liberal "fail and preserve the data to try again later" conditions to try and avoid lost data. If a sensor string fails, it will attempt to apply one of arbitrary "restart mechanics" to that thread to bring it back to life, and keep trying as long as it is failed. If a single sensor fails, it attempts to keep polling remaining sensors on that string; sufficient failures can, however, cue a string restart.

C. DataBase environment

The sink for the data streams is a MSSQL DB, located on a high availability server backed virtual machine. This attempts to provide a degree of resiliency in what would otherwise be a single point of failure. In the schema there are three tables. The primary table contains the sensor geometry (x,y,z in feet accurate to within approximately 2 inches, sensor barcode, location id, deployment and removal timestamps). The second table holds sensor data (value in degrees c to 4 decimal points with questionable accuracy beyond one, sensor barcode, sensor bus, and poll timestamp). There is additionally a positionEnum table to allow a more descriptive arbitrary human readable location to be associated with the location id in the sensor geometry table.

The sensor data table is populated exclusively from the harvesting nodes, each sensor reading generates a single row, at a rate of between 20-40s/row. The DB currently contains over 280 million rows at 16,380 MB of raw data and growing at a 80 MB/day rate.

D. Streaming environment

The implementation of the streaming processing is based on the IBM Infosphere Streams [1] framework, so that we have a scalable application that can be run on a large distributed cluster. Running the implemented algorithm as a service in the cloud is a good way to get the flexible scalable system that would satisfy the needs of large data stream realtime processing when the source data grows. IBM InfoSphere Streams is a high-performance computing platform that allows user-developed applications to rapidly ingest, analyze, and correlate structured data streams as it arrives from real-time sources. The application component logic defines the data flow graph and processing routines applied to the data blocks. The architecture of the application leverages the infrastructure of InfoSphere streams throughout to achieve scalability on large clusters. The solution is ready to be deployed in a Cloud. A dynamic scalable Cloud cluster would be able to meet the demand of large data streams realtime processing by adding additional nodes to the processing cluster when needed.

III. Robust Incremental PCA

We used a novel way of running the parallel streaming PCA algorithm, which efficiently estimates the eigensystem by processing the data stream in parallel and periodically synchronizing the estimated eigensystems. As a detour we discuss the iterative procedure and the statistical extensions that enable our method to be capable of coping with outliers as well as noisy and incomplete data. The streams synchronization algorithm which will run the processing in parallel is discussed in detail in [2].

We focus on a fast data stream, whose elements are high-dimensional $\boldsymbol{x}_n$ vectors. Similarly to the recursion formulas commonly used for evaluating sums or averages, e.g., the mean $\boldsymbol{\mu}$ of the elements, we can also write the covariance matrix $\mathbf{C}$ as a linear combination of the previous estimate $\mathbf{C}_{\mathrm{prev}}$ and a simple expression of the incoming vector. If we solve for the top p significant eigenvectors that account for most of the sample variance, the $\mathbf{E}_p\mathbf{\Lambda}_p\mathbf{E}_p^T$ product is a good approximation of full covariance matrix, where $\{\mathbf{\Lambda}_p, \mathbf{E}_p\}$ is the truncated eigensystem. The iterative equation for the covariance becomes

$$\mathbf{C} \approx \gamma \mathbf{E}_p \mathbf{\Lambda}_p \mathbf{E}_p^T + (1-\gamma)\boldsymbol{y}\boldsymbol{y}^T = \mathbf{A}\mathbf{A}^T \qquad (1)$$

where $\boldsymbol{y} = \boldsymbol{x} - \boldsymbol{\mu}$, and we can also write this as the product of a matrix $\mathbf{A}$ and its transpose, where $\mathbf{A}$ has only $(p+1)$ columns, thus it is potentially much lower rank than the covariance matrix itself [3]. The columns of $\mathbf{A}$ are constructed from the previous eigenvalues λ_k, eigenvectors $\boldsymbol{e}_k$, and the next data vector $\boldsymbol{y}$ as

$$\boldsymbol{a}_k = \boldsymbol{e}_k\sqrt{\gamma\lambda_k}, \qquad \text{for} \quad k = 1 \ldots p \qquad (2)$$

$$\boldsymbol{a}_{p+1} = \boldsymbol{y}\sqrt{1-\gamma} \qquad (3)$$

Then the singular-value decomposition of $\mathbf{A} = \mathbf{U}\mathbf{W}\mathbf{V}^T$ provides the updated eigensystem, because the eigenvectors are $\mathbf{E} = \mathbf{U}$ and the eigenvalues $\mathbf{\Lambda} = \mathbf{W}^2$.

A. Robustness against Outliers

The classic PCA procedure essentially fits a hyperplane to the data, where the eigenvectors define the projection matrix onto this plane. If the truncated eigensystem consists of p basis vectors in $\mathbf{E}_p$, the projector is $\mathbf{E}_p\mathbf{E}_p^T$, and the residual of the fit for the nth data vector is

$$\boldsymbol{r}_n = (\mathbf{I} - \mathbf{E}_p\mathbf{E}_p^T)\boldsymbol{y}_n \tag{4}$$

Using this notation, PCA solves the minimization of the average $\sigma^2 = \langle r_n^2 \rangle$, The sensitivity of PCA to outliers comes from the fact that the sum will be dominated by the extreme values in the data set.

The current state-of-the-art technique was introduced by [4] to overcome these limitations, which is based on a robust M-estimate [5] of the scale, called M-scale. Here we minimize a different σ that is the M-scale of the residuals, and satisfies

$$\frac{1}{N}\sum_{n=1}^{N}\rho\left(\frac{r_n^2}{\sigma^2}\right) = \delta \tag{5}$$

where the robust ρ-function is bound and assumed to be scaled to values between $\rho(0)=0$ and $\rho(\infty)=1$. The parameter δ controls the breakdown point where the estimate explodes due to too much contamination of outliers. By implicit differentiation the robust solution yields a very intuitive result, where the location $\boldsymbol{\mu}$ is a weighted average of the observation vectors, and the hyperplane is derived from the eigensystem of a weighted covariance matrix,

$$\boldsymbol{\mu} = \left(\sum w_n \boldsymbol{x}_n\right) \Big/ \left(\sum w_n\right) \tag{6}$$

$$\mathbf{C} = \sigma^2 \left(\sum w_n (\boldsymbol{x}_n - \boldsymbol{\mu})(\boldsymbol{x}_n - \boldsymbol{\mu})^T\right) \Big/ \left(\sum w_n r_n^2\right) \tag{7}$$

where $w_n = W(r_n^2/\sigma^2)$ and $W(t) = \rho'(t)$. The weight for each observation vector depends on σ^2, which suggests the appropriateness of an iterative solution, where in every step we solve for the eigenvectors and use them to calculate a new σ^2 scale; see [4] for details. One way to obtain the solution of eq.(5) is to re-write it in the intuitive form of

$$\sigma^2 = \frac{1}{N\delta}\sum_{n=1}^{N} w_n^\star r_n^2 \tag{8}$$

where $w_n^\star = W^\star(r_n^2/\sigma^2)$ with $W^\star(t) = \rho(t)/t$. Although, this is not the solution as the right hand side contains σ^2 itself, it can be shown that its iterative re-evaluation converges to the solution. By incrementally updating the eigensystem, we can allow for the solution for σ^2 simultaneously.

B. Algorithm Extensions

When dealing with the online arrival of data, there are several options to maintain the eigensystem over varying temporal extents, including a damping factor or time-based windows. These issues are orthogonal to the parallel PCA computation and can both be applied provided a single source of data that provides a well-ordered stream. Both approaches can be implemented, exploiting sharing strategies for sliding window scenarios, as well as data-driven control and coupling of the duration of the window with the eigensystem size. Our algorithm is also capable of handling missing data and gaps in the data stream.

The recursion equation for the mean is formally almost identical to the classic case, and we introduce new equations to propagate the weighted covariance matrix and the scale,

$$\boldsymbol{\mu} = \gamma_1 \boldsymbol{\mu}_{\text{prev}} + (1-\gamma_1)\boldsymbol{x} \tag{9}$$

$$\mathbf{C} = \gamma_2 \mathbf{C}_{\text{prev}} + (1-\gamma_2)\ \sigma^2 \boldsymbol{y}\boldsymbol{y}^T/r^2 \tag{10}$$

$$\sigma^2 = \gamma_3 \sigma^2_{\text{prev}} + (1-\gamma_3)\ w^\star r^2/\delta \tag{11}$$

where the γ coefficients depend on the running sums of 1, w and wr^2 denoted below by u, v and q, respectively [6].

$$\gamma_1 = \alpha v_{\text{prev}}/v \quad \text{with} \quad v = \alpha v_{\text{prev}} + w \tag{12}$$

$$\gamma_2 = \alpha q_{\text{prev}}/q \quad \text{with} \quad q = \alpha q_{\text{prev}} + wr^2 \tag{13}$$

$$\gamma_3 = \alpha u_{\text{prev}}/u \quad \text{with} \quad u = \alpha u_{\text{prev}} + 1 \tag{14}$$

The parameter α introduced here, which takes values between 0 and 1, adjusts the rate at which the evolving solution of the eigenproblem *forgets* about past observations. It sets the characteristic width of the sliding window over the stream of data; in other words, the effective sample size.[1] The value $\alpha = 1$ corresponds to the classic case of infinite memory. Since our iteration starts from a non-robust set of eigenspectra, a procedure with $\alpha < 1$ is able to eliminate the effect of the initial transients. Due to the finite memory of the recursion, it is clearly disadvantageous to put the spectra on the stream in a systematic order; instead they should be randomized for best results.

It is worth noting that robust "eigenvalues" can be computed for any basis vectors in a consistent way, which enables a meaningful comparison of the performance of various bases. To derive a robust measure of the scatter of the data along a given eigenspectrum $\boldsymbol{e}$, one can project the data on it, and formally solve the same equation as in eq.(5) but with the residuals replaced with the projected values, i.e., for the kth eigenspectrum $r_n = \boldsymbol{e}_k \boldsymbol{y}_n$. The resulting σ^2 value is a robust estimate of λ_k.

IV. Changepoint detection

As a result of data processing we aimed to get a data set that is easy to visualize and which clearly marks unexpected events for automatic detection systems. Our approach allows the reduction of the dimensionality of incoming data vectors to just above ten by using robust incremental PCA. Each vector will contain a subset of incoming data variability and changes in this vector will clearly mark the changes in source data. The source data vectors are projected into the calculated eigensystem basis to reduce complexity while still preserving significant variations. The result can then be visualized as a 2D map of scatter plots for each pair of projection dimensions. Typical system cycles/conditions will be marked out by increased scatter point density and transitional states will also become visible. Applying cluster algorithms to the data will allow assigning clusters to typical system conditions for automatic detection.

Conceptual steps in the streaming workflow:

1) Run streaming PCA on source data to determine the eigensystem to represent typical behavior
2) Project the source data into eigensystem basis

[1] For example, the sequence u rapidly converges to $1/(1-\alpha)$.

3) Calculate (define) clusters in projected source data
4) Input real time data item block
5) Visualize/evaluate - detect outliers / lookup the time-stamp of input data item(s)
6) Go to 1

First we calculate the eigensystem to learn the typical behavior. The eigensystem will contain information about all typical conditions of the environment. We found that 15 eigenvectors are sufficient to catch over 95% of the variability of our dataset by plotting first 100 eigenvalues (Figure 2). We can reduce the data to these 15 eigenvectors without losing much useful information. The StreamingPCA algorithm and implementation used is described in [2].

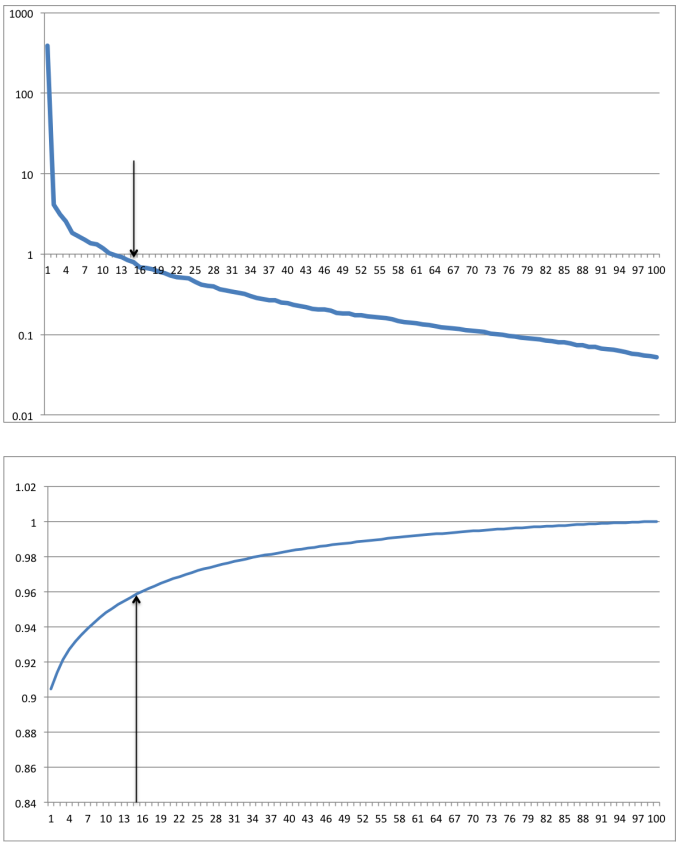

Fig. 2. Eigenvalue distribution and corresponding cumulative value function.

We then project the input data vector stream into the subspace spanned by the first 15 eigenvectors. The input data stream can be filtered by a specififc time range or streamed in realtime for constant monitoring. There are multiple ways to further analyze the projected data. We have decided to apply k-means clustering [7] to determine the typical behavior clusters for multidimensional data. The goal was to identify the system's "normal" state clusters so it becomes easy to detect when a data input point transitions that is, leaves one of these clusters.

Given a set of observations (x_1, x_2, , x_n), where each observation is a d-dimensional real vector, k-means clustering aims to partition the n observations into k sets ($k \leq n$) S = S_1, S_2, , S_k so as to minimize the within-cluster sum of squares (WCSS):

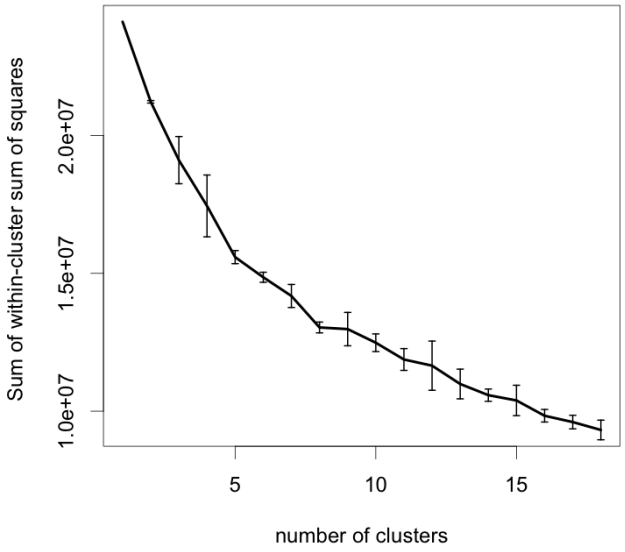

Fig. 3. Sum of all cluster points distance from the centers of the clusters (WCSS) as a function of the (k) number of clusters (20 simulation runs).

$$\arg\min_{s} \sum_{i=1}^{k} \sum_{x_j \in S_i} \|x_j - \mu_i\|^2$$

where μ_i is the mean of points in S_i.

The number of clusters, which is a required parameter of k-means clustering algorithm, was estimated by computing the sum of WCSS of all clusters for different numbers of clusters (Figure 3). The data shown are the avarage WCSS and its standard deviation for 20 k-means clustering run with randomized initial starting points. From the plot it was decided to look for 12 clusters in all computations for our sample dataset.

A. *Visualization Of Projected Data*

The 2-dimensional slices of 15-dimensional space were generated as scatter plots. The points are colored according to the identified cluster number. On these plots we can clearly spot the typical data conditions as condensed clusters. The image displays 2-dimensional slices of dimensions 1, 2, 3 and 4 in 2000-point window (Figure 4).

B. *Changepoint Event Detection*

To model a situation of identifying a transition event in real time, sample data should be aggregated in a sliding window long enough to cover the changepoint event and some typical, steady state data before and after the event. We have used data collected during an intentionally executed AC switch off, which we considered to be a transient temperature event. Three different ACs in the DC were switched off and back on in 15 -minute steps. On the plots we can clearly see the typical data

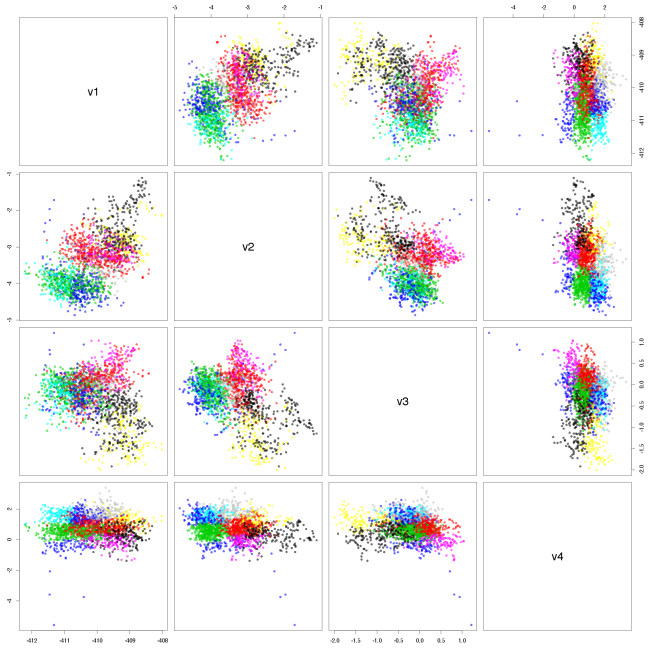

Fig. 4. Typical behavior. Points represent time steps, colors represent k-means clusters. 2000 points

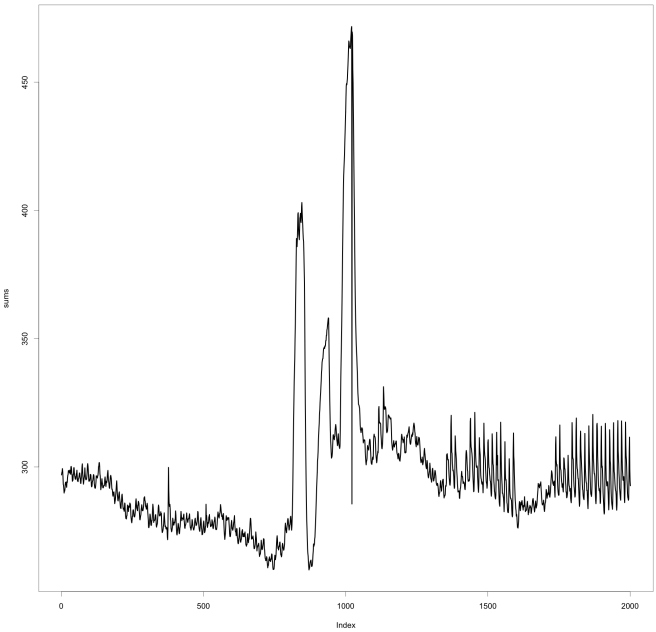

Fig. 6. The sum of the distances from the cluster centers as a function of time - the changepoint events can easily be indentified

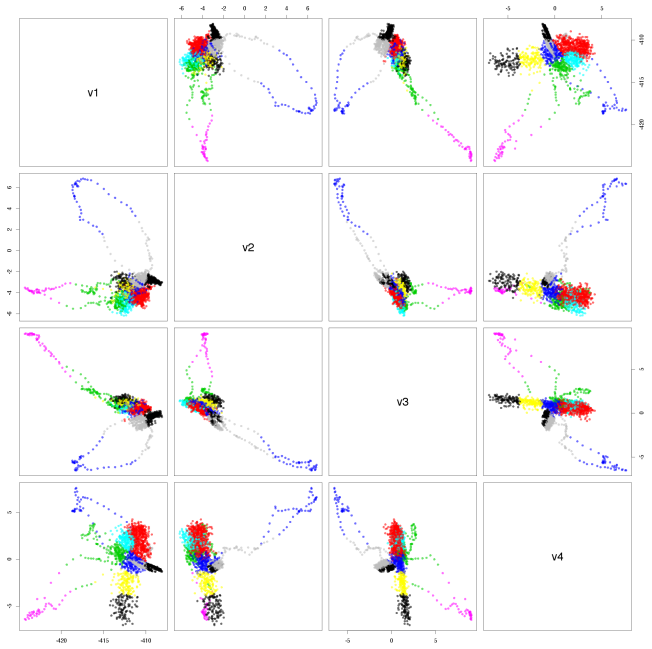

Fig. 5. The AC switch off event. The outlier trajectory is clearly marked. After the event system returns to the original state. 2000 points time window

conditions (condensed clusters of data) and also detect outlier trajectories deviating far from their original clusters (Figure 5). Monitoring the current position relative to the cluster shows the abnormal behaviour of the environment for automatic events detection.

In order to further facilitate changepoint event detection, we set out to translate the outlier movement into a single scalar value, that can be easily monitored and gives a sharp signal at nonstandard events. Therefore we calculated the sum of the distances of the actual data points from the cluster centers as a function of the timesteps. (Figure 6) As on the scatter plots one can also notice the normal cyclicity of the system, but large peaks appear when the AC events occur.

V. Conclusion

We developed a robust sensor network environment to provide high granularity real life temperature sensor data streams to study DC thermal behavior patterns.

We experimented with a streaming approach to enable the analysis of our large scale sensor data real time. By applying robust incremental PCA and K-means clustering we successfully concluded a method for easy visual or conventional signal processing detection of outliers and changepoint events. This method can also be applied to complex systems with multiple types of sensor data streams exhibiting multiple normal or steady states, different trajectories and transients, for finding anomalies and unexpected system behavior.

Current results open the door to process very high numbers of even different types of sensor input streams in a real time/streaming way. The output can also be customised to serve as input to further change point detection and identification processes e.g. automatic (streaming) visual detection, sonification forming one component of a streaming processing pipeline.

References

[1] IBM Infosphere Streams description. [Online]. Available: http://www.ibm.com/software/products/en/infosphere-streams

[2] D. Mishin, T. Budavari, A. S. Szalay, and Y. Ahmad, "Incremental and parallel analytics on astrophysical data streams." in *SC Companion*. IEEE Computer Society, 2012, pp. 1078–1086. [Online]. Available: http://dblp.uni-trier.de/db/conf/sc/sc2012c.html

[3] Y. Li, L. qun Xu, J. Morphett, and R. Jacobs, "An integrated algorithm of incremental and robust pca," in *Proceedings of IEEE International Conference on Image Processing*, 2003, pp. 245–248.

[4] R. Maronna, "Principal components and orthogonal regression based on robust scales," *Technometrics*, vol. 47, no. 3, pp. 264–273, Aug. 2005. [Online]. Available: http://dx.doi.org/10.1198/004017005000000166

[5] P. J. Huber, *Robust statistics*. Wiley, New York, 1981.

[6] T. Budavari, V. Wild, A. S. Szalay, L. Dobos, and C.-W. Yip, "Reliable eigenspectra for new generation surveys," 2008.

[7] K-means clustering libray. R documentation. [Online]. Available: https://stat.ethz.ch/R-manual/R-devel/library/stats/html/kmeans.html