A data-driven adaptive model-identification based large-scale sensor management system: application to Self Powered Neutron Detectors

Nahit Pawar M.N. Belur M. Bhushan A.P. Tiwari M.G. Kelkar M. Pramanik and Virendra Singh

***Abstract*—In this paper we propose an adaptive approach to manage large number of correlated sensors. Our approach is able to extract information (models) from these sensors that is relevant for performing fault diagnosis of these sensors. Such a situation involving large number of correlated sensors is encountered in large core nuclear reactors, for example. Since fault diagnosis methods are computationally intensive, it is helpful to organize the sensors into groups such that strongly correlated sensors belong to the same group. However, the groups/clusters need to be reorganized depending on operating conditions. We propose an adaptive method that is scalable to a large number of sensors and can adapt to changing operating conditions. Also, within each cluster, it is often required to adaptively rebuild new models/relations for sensors inside that cluster. We use the k-means algorithm for obtaining clusters and Principal Component Analysis (PCA) for finding relations between the sensors within a cluster. We demonstrate that significant speedup is achieved by parallelizing the various aspects of the above computation.**

A key requirement in managing a large number of sensors is the data and processing management. We demonstrate and compare a serial and parallel implementation of this method using SQLite for database management, Python for numerical computations, the Pycluster module for clustering and the Python multiprocessing module for code parallelization. The method is demonstrated for the above nuclear reactor application: with 140 sensors and 14,000 measurements for each sensor. The method turns out to scale very easily to such a large number. The implementation codes of our approach have been made available online. The utilized packages all being open source (FOSS) helps in the use of these codes in various safety critical applications which typically require complete verification/ratification. The cost saved due to the FOSS aspect of our implementation is another advantage.

Keywords: **Clustering, Principal Component Analysis, Adaptive re-clustering, Fault diagnosis, Parallel Implementation, FOSS**

I. Introduction

When dealing with a large number of sensors in real-time estimation and control problems, it is often computationally infeasible to build linear relations adaptively for all of them together. Sensor Management is crucial when dealing with such large network of sensors, as has been rightly emphasized in the literature, for example in [8] and [3]. We elaborate on the various issues in Sensor Management using the example of Self Powered Neutron detectors (SPNDs) which play an important role in on-line neutron flux estimation, safety and control of large core nuclear reactor. A typical nuclear reactor contains several of these sensors for measuring flux at different locations inside the reactor. Measurement of neutron flux provides a direct measure of reactor power and thus measurements from these sensors are used for safety, regulatory and control purposes. Thus healthy operation of SPNDs is important for safe reactor operation.

In case of a fault in the SPND due to various reasons the usual practice is to manually rectify the fault as soon as possible and this involves unscheduled reactor shutdown which will be time consuming and have cost and personnel safety implications. Hence in the typical situation of evolving reactor operating conditions, there is a strong need to develop self-monitoring and evolving strategies that can effectively monitor the health of these sensors using the available on-line measurements. In the context of computational aspects, which is very relevant for online and real-time management of sensor measurements, [3] discusses the computational feasibility of scaling present methods to large-scale problem involving sensing system with fast changing operating conditions.

A. Objective

As mentioned above, real-time data-driven self-monitoring and adaptive techniques are required when dealing with large number of sensors. Such situations also require that the sensors be clustered into smaller groups based on their time-series response. Clustering techniques in the context of sensor management have been proposed in [1], where the authors discuss and compare different clustering algorithms for sensor management. The focus of this paper is about adaptive methods to recluster and to rebuild models within a cluster. For a smaller group of sensors obtained by clustering, it is often required to adaptively build linear models (using Principal Component Analysis (PCA), for example) on the sensors in each group. The models are then utilized for data reconciliation, fault detection and diagnosis of SPNDs. It is also essential to evolve triggering mechanisms for re-clustering and re-development of models due to the changes in reactor operating regime. A key requirement is the constant validation of current data against various models available in the database to detect and diagnose faults. In order to effectively implement the above proposed methods for on-line monitoring, it is essential to use modern database management systems for creation and maintenance of a database of the large number of sensor values and models. For safety critical applications, it is essential to first simulate various scenarios by setting up a client computer for on-line data generation which emulates the existing systems in the control room of nuclear power plant. This further requires the setting up of an interface for data-acquisition on the computer

N. Pawar and V. Singh are in the Department of Electrical Engineering, and M. Bhushan is in the Department of Chemical Engineering at the Indian Institute of Technology Bombay. A.P. Tiwari is in the Reactor Control Division, Bhabha Atomic Research Center, Mumbai. M.G. Kelkar and M. Pramanik are in the Nuclear Power Corporation of India Ltd. (NPCIL), Mumbai, India. M.N. Belur is in the Department of Electrical Engineering and in the Centre for Technology Alternatives for Rural Areas (CTARA), Indian Institute of Technology Bombay. The research was supported by the Board for Research in Nuclear Sciences (BRNS), India. Corresponding author email: belur@iitb.ac.in.

hosting the standalone software so that it can receive data from the client. The standalone aspect is essential for complete ratification of the developed codes: this is again relevant in safety critical operations. This paper addresses the above objectives of a typical large scale adaptive sensor management system. The paper's contribution and organization of the paper is as follows.

B. Organization of the paper and summary of contribution

This paper is organised as follows. Section II presents SPNDs data management with particular focus on use of SQlite database engine for handling data. Section III presents clustering method and the use of Pycluster module [15] and Python for grouping SPNDs. In Section IV we first review PCA techniques followed by model identification and residual calculation using numpy.linalg module [12] and Python on clusters developed in Section III. Section V summarizes the data reconciliation and gross error detection technique using the relationship developed in Sections III and IV. Section VI explains the proposed adaptive model updating and re-clustering technique, server-client interaction, followed by overall software flowchart. Section VII contains the profiling results. Finally, Section VIII contains future work and concluding remarks. The FOSS aspect of the packages used is central for a complete ratification of the codes when using such adaptive sensor network management system in safety critical applications: the codes have been available online at [10], [11].

When compared with other methods in the literature about sensor management (for example, [1], [3], [8]), the approach in this paper is different in the sense that for applications involving large-scale sensors network which demand efficient sensor management and data fusion, we propose a scalable method (of adaptively reclustering and model identification techniques) and demonstrate it by a parallel implementation in packages that are more preferred for safety critical applications (due to their FOSS aspect).

II. SPNDs DATA MANAGEMENT

In this section we elaborate on the typical difficulties faced in the context of management of a large number of sensors. We use the case of SPNDs in a large core nuclear reactor as an example to highlight the challenges faced.

A. Characteristics of SPND data

SPNDs are characterized [19] as prompt SPNDs or delayed SPNDs depending on whether the major component of the signal is prompt or delayed. For example, SPNDs having Cobalt as emitter material are prompt in nature whereas SPNDs having Vanadium as emitter material are delayed in nature thus have delayed response to external neutron flux. The SPND data used in our work corresponds to an Indian Pressurized Heavy Water Reactor (PHWR). The reactor consists of a cylindrical core with 42 Cobalt and 102 Vanadium SPNDs. The core of PHWR for which data is available to us, is considered to be divided into fourteen control zones with each zone containing three Cobalt SPNDs. The 144 SPNDs are labeled using serial numbers, since data from SPNDs provided for this study are in the form of text file and according to the format of text file Vanadium SPNDs are labeled from 29 to 130 and Cobalt SPNDs from 1 to 42. For example, V35 and Co26 represents Vanadium SPND number 35 and Cobalt SPND number 26 respectively, the same naming scheme is used to locate sensor in database. Data from SPNDs sampled at 1 minute intervals is available for a period of 10 days.

B. Characteristics of missing observation

Out of 14,400 data points some observations are missing which are labeled as 'BAD' or 'NOREPLY' in text files. (This is possibly due to communication delays at that time instant, for example.) These observation are replaced by -1 while creating database of SPNDs. Since nature of missing observation in Vanadium and Cobalt datasets falls under the category Missing Completely At Random (MCAR) [18], therefore these data points are discarded from our work.

C. Use of SQlite and Python for managing SPND data

We have used SQlite [22] database engine which provides lightweight disk-based database and allows accessing the database using nonstandard variant of SQL query language. We used sqlite3 module [23] of Python which provides interface for using SQlite database engine in Python. In our work this interface is used to maintain three different database one for storing SPND data and other two for storing cluster and model information as descried in Sections III and IV respectively. Each row of the SPND database contains single observation from 144 SPNDs along with time-stamp. Initial part of the database is created off-line for storing training data, for this purpose we have created graphical user interface (GUI) in Figure 1 using Python's Tkinter module which provides operator with user friendly interface for storing and viewing SPND data off-line. The source code for this interface is made available at [10].

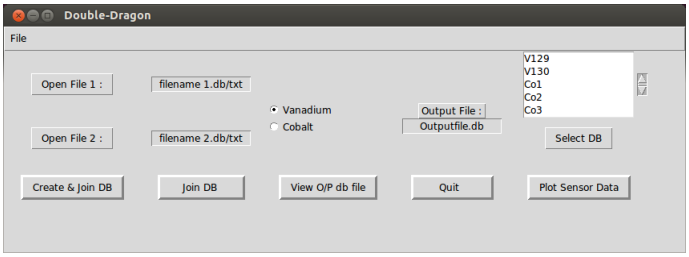

Fig. 1: Snapshot of user interface for creating database

D. Training and test data

Operator can select different sets of training data for clustering (Section III) and model identification (Section IV). In this paper, we have used first 7000 data points, which is about half of the data points available to us for clustering and model identification, and the rest half as test data.

III. CLUSTERING OF SPNDs

SPNDs which are in the same zone or close to each other are expected to be subjected to similar variation in the neutron flux. However, it is also possible that SPNDs in different zones

gives correlation measurements depending on the operating regime of the reactor. So using the data available to us, we group the SPNDs into smaller groups such that those SPNDs with strongly correlated measurements are grouped together. This allows us to work on individual clusters of SPNDs rather than entire set of SPNDs. For more details, the interested readers are referred to [17].

The clustering is performed using Pycluster module [15] of Python which uses k-means algorithm [4], [20]. The aim of this algorithm is to partition set of points into specified number of clusters such that the sum of distances of each point from its cluster centroid in minimized. For our case, the time series (training data) of single SPND is taken as a point. Since we require those SPNDs having high correlation between each other to be grouped together i.e. two points are close to each other if the correlation between them is high therefore we used *absolute Pearson correlation distance* for defining distance between two SPNDs as [17]:

$$d_{A,B} = 1 - |r_{A,B}| \tag{1}$$

where $d_{A,B}$ denotes distance between SPND A and SPND B; $r_{A,B}$ denotes the Pearson correlation coefficient between SPND A and SPND B and is defined as:

$$r_{A,B} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{a_i - \bar{a}}{\sigma_a}\right)\left(\frac{b_i - \bar{b}}{\sigma_b}\right) \tag{2}$$

where $\bar{a}, \bar{b}$ denotes sample mean of SPND A and SPND B respectively; σ_a, σ_b denotes sample standard deviation of SPND A and SPND B respectively; n denotes number of observations.

The input to the k-means algorithm consist of training data on which the clustering is to be performed and the number of cluster k, where k is chosen suitably [17]. Since clustering solution depends on the initial assignment of SPNDs to clusters, kcluster routine in Pycluster module repeats k-means algorithm many times, each time starting with different initial random clustering. The sum of distances of SPNDs to their clusters center is saved for each run and the solution with smallest value of this sum is returned as the overall clustering solution. This is done automatically by kcluster routine.
A routine build_cluster written in Python which uses kcluster routine for the purpose of partitioning SPNDs into clusters. The first step in this routine is to read training data from database followed by generating data matrix - $data[i][j]$. Each row of data matrix represents observations from single SPND in which missing observations is stored as -1. Information pertaining to missing observation in the data matrix is used to create mask matrix - $mask[i][j]$, such that $mask[i][j] = 0$ if $data[i][j]$ is missing otherwise $mask[i][j] = 1$. Given the distance function (equation 1) and how the center of the cluster is found in our case (arithmetic mean), kcluster routine is called with data and mask matrix as input. The output of kcluster routine is used to create clusterId to SPND map, i.e. given the cluster number it will show which SPNDs belongs to this cluster. Finally the clusterId to SPND map is stored in database with time-stamp. The overall flow of build_cluster routine is shown in Figure 2.

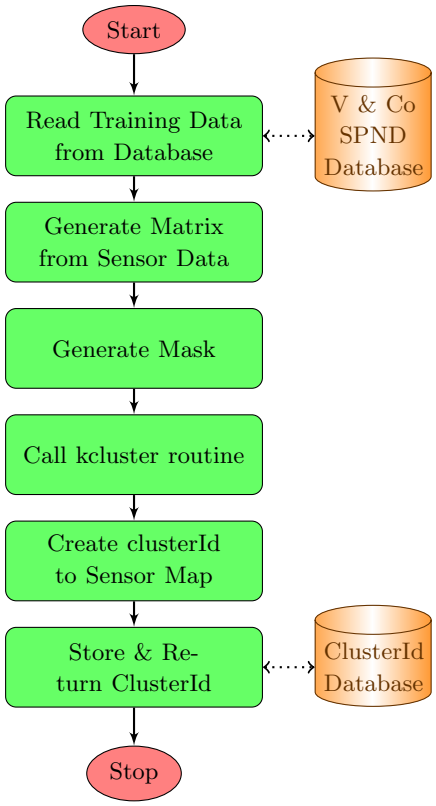

Fig. 2: build_cluster - flow chart

IV. Identifying Models on Clusters

Once the clusters are obtained, we use the PCA technique to identify linear models on a single cluster of SPNDs. We review this in the following subsection. Sections IV-B and IV-C discuss procedures to identify the model and residual calculation respectively.

A. Review: Principal Component Analysis

We give only a brief description of this well-known technique. A detailed description can be found [2], [5], for example. PCA is a data driven modeling technique that allows us to transform a set of correlated variables into a new set of uncorrelated variables. These new variables are such that the first principal component direction has the largest variance and each succeeding component in turn has highest variance possible under the constrain that it be orthogonal to preceding component. These principal component directions are the eigenvectors of the data covariance matrix and the variance along each direction is given by the corresponding eigenvalue.

Let X be an $N \times n$ data matrix where N is the number of observations and n is the number of variables. Let $P = [T|Q]$ where T is an $n \times d$ matrix containing the d significant eigenvectors and Q is an $n \times (n-d)$ matrix with the $(n-d)$ least significant eigenvectors as its columns. Then X can be decomposed as:

$$X = \hat{X} + E,$$

where the matrices $\hat{X}$ and E represent modeled and unmodeled variations of X, respectively:

$$\hat{X} = X(TT^T),$$

$$E = X(QQ^T).$$

The matrix TT^T is orthogonal projection matrix and $\hat{X}$ is the projection of X on the d dimensional subspace known as principal component space or estimation space. If we assume true variation in measurement data are along these d significant principal axes, then we have for the true data

$$XQ = 0 \quad \text{or} \quad Q^T X^T = 0. \tag{3}$$

Thus if $x(t)$ is the true measurement vector $(n \times 1)$ at some instant t, we have a relation

$$Ax(t) = 0. \tag{4}$$

Here $A = Q^T$ contains the least significant $n-d$ eigenvectors along the rows. For any given measurement vector $y(t) = x(t) + e(t)$, where $e(t)$ is the measurement noise. The residual is given by

$$r(t) = Ay(t) \tag{5}$$

A is called the constraint matrix or model matrix. The residuals $r(t)$ are used for fault detection purpose as explained in Section V. The eigenvectors of the covariance matrix can be obtained by computing Singular Value Decomposition (SVD) of the data matrix X. If SVD of X is given by $X = USV^T$, then the columns of U are eigenvectors of the data covariance matrix.

B. Model Identification

The first 7000 points are used as training data for building models on each single cluster. For this, the linear algebra module (numpy.linalg) [12] in Python is used for computing SVD on the clusters. The relations are the eigenvectors corresponding to the least significant eigenvalues. Those eigenvalues that are less than the average are typically considered as less significant: see [2, page 441] and [9]. A routine build_model has been written in Python that uses numpy.linalg [12] and sqlite3 [23] module for identifying models and storing it in database. The first step in build_model is to decide on whether to build cluster from scratch using training data or to use already built clusters from database both of which result in clusterId to SPND map. For each cluster in the map it prepares data matrix from database of SPND measurements it then uses this data matrix to compute model matrix using SVD function (numpy.linalg.svd) in numpy.linalg module. Finally it returns and stores all the computed models in the database of model matrix. The overall flow of build_model routine is shown in Figure 3.

Since the model identification on individual cluster is inherently independent of each other, we used Python's multiprocessing capability to compute each model in parallel. The overall flow of our *parallel* code is same as Figure 3 except for: preparation of data matrix, model computation (by SVD computation) and storage of model matrix; this is done in parallel for each cluster.

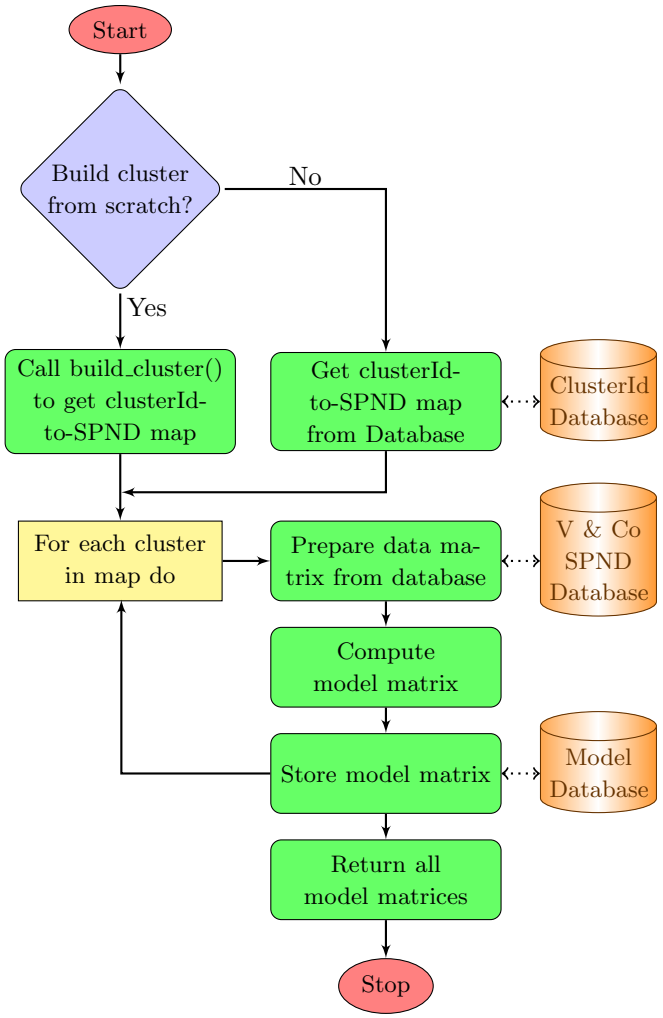

Fig. 3: build_model - flow chart

C. Residual Calculation

As explained in Section IV, given the constraint matrix A and measurement vector $y(t)$ the residual is given by equation (5). Let R be the matrix of residuals given by:

$$R = AY \tag{6}$$

where Y is matrix of measurement vectors or the training data. The residual R is used for calculating the noise covariance matrix Σ_ϵ as explained in Section V. A routine calc_residual written in Python uses output of build_model and numpy.linalg module [12] to calculate R on each model as shown in Figure 4.

V. Data Reconciliation and Gross Error Detection

The total error in the sensor measurements can be represented as sum of the contribution from two types of errors: random error and gross error. Random error are inherently unpredictable and are always present in the measurement. Gross error are caused by non-random events and occur as a result of sensor malfunctioning and at any given time error has certain magnitude and sign. This section summarizes the data

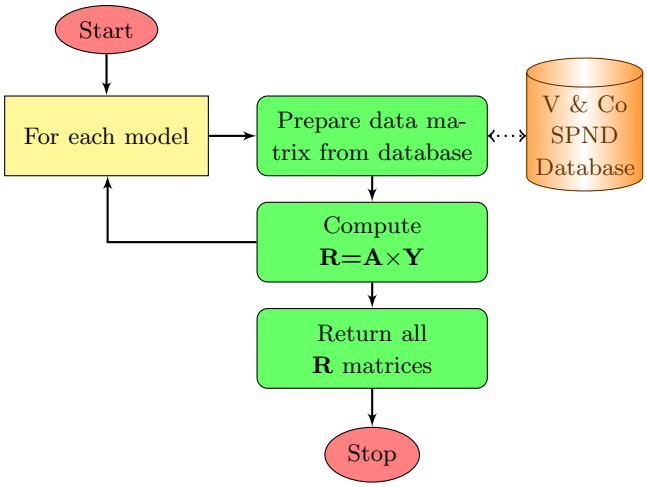

Fig. 4: calc_residual - flow chart

reconciliation and gross error detection techniques that exploits the relationship developed in Sections III and IV between SPND to reduce the effect of both types of errors. We used data reconciliation techniques (Section V-A) that explicitly make use of model constraints (equation (4)) and estimate the output of SPNDs by adjusting the measurement so that the estimates satisfy the constraints. Data reconciliation technique works when there is no gross error [6], therefore we used gross error detection techniques (equation (V-B)) to identify and remove gross error in the measurement.

A. Review: Data reconciliation

Given the model constraint matrix A and noisy measurements, the estimate of true value of measurement is given by following weighted least square optimization problem:

$$\min_{x(t)} \quad (y(t) - x(t))^T \mathbf{W} (y(t) - x(t)), \tag{7}$$

$$\text{with the constraint } Ax(t) = 0,$$

where $\mathbf{W}$ is an $n \times n$ weighting matrix usually taken as the inverse of the noise covariance matrix Σ_ε as defined after equation (6). The solution to the above optimization problem [6] is given by:

$$\hat{x} = y(t) - \mathbf{W}^{-1} A^T (A W^{-1} A^T)^{-1} A y(t) \tag{8}$$

where $\hat{x}$ are the reconciled estimate of the true measurement at time t. For more details, the interested readers are referred to [6].

B. Review: Gross error analysis

In our work, we considered gross error to be constant biases and zero output from the sensors. Consider a gross error scenario, where we assumed the gross error to be constant biases in p_k number of sensors, with the indices of these p_k sensor stored in set P_k. Hence, the measurement vector becomes:

$$y(t) = x(t) + \varepsilon(t) + \sum_{j \epsilon P_k} b_j e_j \tag{9}$$

where b_j is the gross error in the j^{th} variable, e_j is the unit vector with 1 at the j^{th} location and 0's elsewhere and $\varepsilon(t)$ is the random error component of the measurements as before. The method for gross error analysis (i.e. detection, identification and estimation) based on individual observation reviewed below is elaborated in [6].

Given the model constraint matrix A and the measurement $y(t)$, the residual is given by equation (5). For gross error detection the following test statistic has been considered:

$$\gamma(t) = r(t)^T V^{-1} r(t) \tag{10}$$

where V is the noise covariance matrix of residuals (see equation (6)). In the absence of any gross error equation (10) follows a χ^2-distribution with m degrees of freedom. Choosing a confidence level α, we then obtain a threshold value as $\chi^2_{1-\alpha,m}$. Among all clusters, a gross error is detected within a cluster if $\gamma(t) \geq \chi^2_{1-\alpha,m}$: we use statistical module (scipy.stats) [13] of Python to implement this. We then use Generalized Likelihood Ratio (GLR) method [7] for identifying those sensors inside the cluster with gross error along with estimate of the gross error. For more details, we refer to [6] and [7].

VI. Online Fault Analysis

This section explains the proposed on-line adaptive techniques related to model updating and re-clustering in dealing with widely varying operating conditions, server-client interaction and the overall software flowchart.

A. Adaptive model updating

The linear model (equation (4)) identified on each cluster in Section IV-B is typically valid only in a limited range of operating conditions. As the operating conditions change, new models are required on existing cluster/clusters or re-using those existing models in the database which best describes the current operating condition. The following rule is used to trigger new model identification on-line: Due to dynamic operating conditions there can be a possibility of large residuals (equation (5)), we then check for residuals from all models

in the database for a given cluster. If any model results in small residual, then no fault is declared and this matched model becomes our current model. On the other hand if none of the models in the database explains the magnitude of residuals, then we declare the fault, identify the faulty sensor and estimates its true value (Section V). The information related to faulty sensor is then displayed to operating personnel. An input from this person will then be required to validate the result in terms of whether a fault has actually occurred or not. If the operator specifies that no fault has occurred, which means the data coming from that sensor is normal and there is no model in the database which can describe the current operating conditions. In this situation new model is identified and is then added to the database of models.

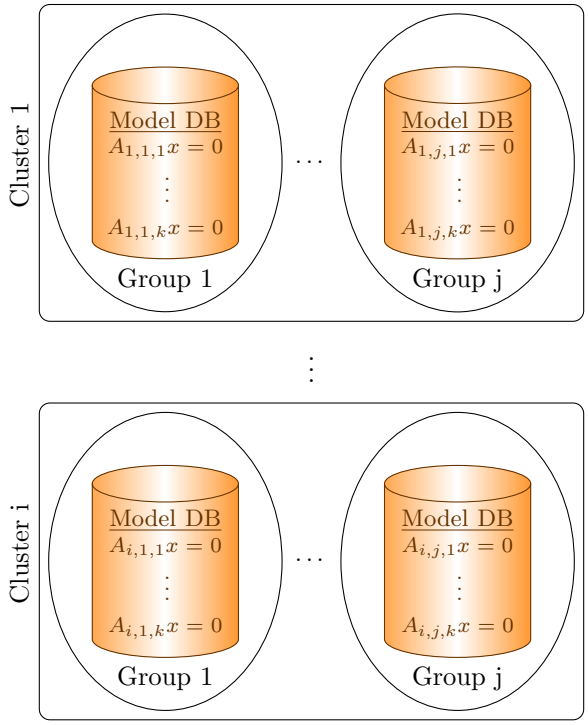

Fig. 5: Adaptive model updating and re-clustering

B. Adaptive re-clustering

As explained in Section III the clusters configuration is based on the sensors' time series data. As the plant behavior changes which results in different normal mode of plant operating condition, the optimal clusters configuration may also change. The decision on when to change current clusters configuration is at the higher level than updating model. The following rule is used for re-clustering sensors on-line: We propose to trigger re-clustering if faults are detected in multiple clusters simultaneously as this most likely indicative of models in those clusters not reflecting the current process condition. Before re-clustering, once again feedback will be taken from operating personnel. For every set of clusters, a database of models is maintained as shown in Figure 5 and as discussed in Section VI-A.

C. Emulating server-client interaction and summary of the software flow

We have set-up a client computer which is responsible for sending test data samples at one minute interval, this emulates the behavior of the system in the control room of nuclear power plant. Server side is equipped with our standalone software which handles all the request from client and performs fault analysis for each incoming test data. The server and client interact with each other through Local Area Network (LAN), the interface for the same is designed using socket module [21] in Python.

The overall abstract view of our software is as follows. The complete source code is made available at [11]. After building initial set of clusters and models using training data, our software then responds to incoming stream of on-line sensor data from client system by performing on-line fault detection and analysis (Sections V and VI) on each cluster using the residuals calculated on current models. Each step in the software is dealt using the database of sensor data, model and cluster information.

VII. Results

A. Profiling

The module cProfile [16] of Python is used to profile different sections of our standalone software. Table I shows system specifications on which profiling is done. For cluster building, model building and residual calculation 7000 data points is used as training data. Table II shows the clustering parameters used to build clusters. Table III shows profiling results of build_cluster routine for both Cobalt and Vanadium SPNDs. Table IV shows profiling results of serial and parallel version of build_model routine for both Cobalt and Vanadium SPNDs. Table V shows profiling results of calc_residual routine for both Cobalt and Vanadium SPNDs.

TABLE I: System Specifications

Operating System	CPU	RAM	HDD
Linux (Ubuntu 12.04)	Intel core i7 2.3 GHz	8 GB	1TB

TABLE II: Clustering Parameters

SPND	Num. of clusters	npass	nfound
Cobalt	7	100	13
Vanadium	20	100	1

TABLE III: Timing Analysis - Cluster building

SPND	Time (sec) - build_cluster	Time (sec) - kcluster
Cobalt	4.192	3.292
Vanadium	25.278	23.277

VIII. Conclusion and Future Work

In this paper, we proposed adaptive techniques to manage a large number of correlated sensors. To efficiently extract meaningful information from these sensors, it is important to

TABLE IV: Timing Analysis - Model building

SPND	Time (sec) - Serial Code	Time (sec) - Parallel Code
Cobalt	0.428	0.015
Vanadium	1.314	0.058

TABLE V: Timing Analysis - Residual Calculation

SPND	Time (sec) - calc_residual
Cobalt	0.322
Vanadium	0.657

cluster these sensors into smaller groups of strongly correlated sensors. This in-turn also enables parallel implementation of model building and mode verification in real-time. To account for changing process operations, the sensors often need to be re-clustered thereby leading to the adaptive framework proposed in this work. Within a cluster too, it is essential to adaptively identify linear models in case the existing models are not able to satisfactorily obtain the given observations. In our work, this model identification is triggered by changing statistical properties of the residuals. Currently, the decisions are required to be ratified by the operating personnel since the thresholds in the residuals might require fine-tuning based on extensive testing based on the application: thus our method is currently semi-supervised. Once the thresholds are tuned, we expect this method to be a fully adaptive system for re-clustering and re-building of models.

In this paper, we also described the newly developed database management (involving SQLite) and computational tool (involving Pycluster/NumPy) for handling large datasets due to a typical large scale sensor network: these codes have been made available online [10], [11]. The technique entails significant computation in real-time when the residual is matched to the best possible model from the database of models. A key challenge would be to implement the above proposed techniques real-time but on processors with limited memory and limited processing power. The optimum number of models and clusters might have to be tailored depending on the limitations. In a typical application, the actual interaction with the client system and remotely situated server requires knowledge of the network protocol and thus establishment of real-time communication between the two systems. Finally we compared parallel implementation of the proposed method and showed the speedup in Table IV versus serial implementation on a system having specification listed in Table I.

REFERENCES

[1] A.A. Abbasi and M. Younis, A survey on clustering algorithms for wireless sensor networks, *Computer Communications Journal*, vol. 30, pp. 2826-2841, 2007.

[2] H. Abdi and L.J. Williams. *Principal component analysis*. Wiley Computational Statistics, vol. 2, pp. 433-459, 2010.

[3] A.O. Hero III and D. Cochran, Sensor management: past, present, and future, *IEEE Sensors Journal*, vol. 11, pp. 3064-3075, 2011.

[4] A.K. Jain and R.C. Dubes, *Algorithms for Clustering Data*, Prentice Hall Advanced Reference Series, Prentice Hall, 1988.

[5] I.T. Jolliffe, *Principal Component Analysis*, Springer Series in Statistics, 2002.

[6] S. Narasimhan and C. Jordache, *Data Reconciliation and Gross Error Detection: an Intelligent Use of Process Data*, Gulf Pub. Co., 2000.

[7] S. Narasimhan and R.S.H. Mah, Generalized Likelihood Ratio Method for Gross Error Identification, *AIChE Journal*, vol. 33, pp. 1514-1521, 1987.

[8] G.W Ng and K.H Ng, Sensor management: what, why and how, *Information Fusion Journal*, vol. 1, pp. 67-75, 2000.

[9] P.R. Peres-Neto, D.A. Jackson and K.M. Somers, How many principal components? Stopping rule for determining the number of non-trivial axes revisited, *Computation Statistics & Data Analysis*, vol. 49, pp. 974-997, 2005.

[10] N. Pawar, SQLite for Data Management, [Online], available: https://github.com/leonahi/SensorDataManagement.git.

[11] N. Pawar, Cluster and Model Building, Residual Calculation and Error Detection [Online],
available: https://github.com/leonahi/SensorProject.git.

[12] Python linear algebra module: numpy.linalg
http://docs.scipy.org/doc/numpy/reference/routines.linalg.html.

[13] Python statistical module: scipy.stats,
http://docs.scipy.org/doc/scipy/reference/stats.html.

[14] Python multiprocessing module: Multiprocessing, [Online],
http://docs.python.org/3.2/library/multiprocessing.html.

[15] Pycluster module: The C clustering library,
https://pypi.python.org/pypi/Pycluster.

[16] Python profiling module: cProfile,
http://docs.python.org/3.2/library/profile.html.

[17] R.A. Razak, M. Bhushan, M.N. Belur, A.P. Tiwari, M.G. Kelkar and M. Pramanik, Data reconciliation and gross error analysis of self powered neutron detectors: comparison of PCA and IPCA based models, *International Journal of Advances in Engineering Sciences and Applied Mathematics*, vol. 4, no. 1-2, pp. 91-115, 2012.

[18] D. B. Rubin, Inference and missing data, *Biometrika*, vol. 63, pp. 581-592, 1976.

[19] K. Srinivasarengan, L. Mutyam, M.N. Belur, M. Bhushan, A.P. Tiwari, M.G. Kelkar and M. Pramanik, Flux estimation from Vanadium and Cobalt Self Powered Neutron Detectors (SPNDs): nonlinear exact inversion and Kalman filter approaches, *Proceedings of the IEEE American Control Conference (ACC)*, Montreal, Canada, pp. 318-323, 2012.

[20] H. Spath, Cluster Analysis Algorithms for Data Reduction and Classification of Objects, *Computers and their Applications Series*, E. Horwood, 1980.

[21] Socket module: *http://docs.python.org/3.2/library/socket.html*.

[22] SQlite: SQL database engine (*http://www.sqlite.org/*).

[23] Sqlite3 module: Python DB-API 2.0 interface for SQLite databases
http://docs.python.org/3.2/library/sqlite3.html.