ID DA-O-025

Real-Time & Scalable Anomaly Detector based on Correlation Analysis - Toshihiro Takahashi

e30137@jp.ibm.com
IBM – Tokyo Research Laboratory
1623-14, Shimotsuruma, Yamato-shi Kanagawa 242-8502 Japan
Phone: +81 -462154769 Fax: +81-462733413

Introduction

Semiconductor's manufacturing processes consist of many various processes. Under real circumstances, various anomalies in each process are often unavoidable. And some anomalies affect seriously to quality and yield of whole semiconductor process. Real-time anomaly detection and appropriate quick handling are most important technical challenges in semiconductor area. But major approach is workers' observation of instruments and sensors. In such human-based approaches, it is difficult to make quick response for anomalies. In some state of the art researches of industry area, many approaches are investigated to detect anomalies quickly and automatically for helping workers' observations.

Some threshold based methods using sensors' knowledge are successfully applied in some previous researches. But such threshold based method often makes miss-alert and false-alert because it focuses on only small and large of each sensors. We developed correlation-based method which computes anomaly score with using correlation among sensors and we could get some good performance comparing with existing approaches. But our algorithm is designed as batch procedure to pick anomalies up from static past data. So we can not apply our method as it is. We developed online version and parallelized for multi-node computers and achieved real-time correlation-based anomaly detection.

Correlation-based Anomaly Detector

This section says the algorithm outline of our correlation-based anomaly detector. It computes "normal correlation graph" by using normal sensor data beforehand. (figure 1) During processes run, the system cuts sensor data off as window of certain period of time (1sec – hours), computes "correlation graph" of each window (figure 2) and compares with "normal correlation graph". If the structures of these two graphs are largely different, the system makes alert.

The correlation graph means linear correlation matrix between sensors. But linear correlation is often unstable because sensor values are often noisy. If we can set larger window size, we may compute stable linear correlation. But large size window conflict with our purpose of quick anomaly detection. Our algorithm uses GLASSO technique to compute stable linear correlation of small size window. We can get the stable correlation graph $\hat{\Theta}$ to minimize the following formula with respect to Θ.

$$\hat{\Theta} = \arg\min_{\Theta}(\log\det(\Theta) - tr(S\Theta) - \lambda\|\Theta\|_1$$

Here, N is the number of sensors, S is N*N linear correlation matrix among N normalized sensors which average is 0 and variance is 1, and $\hat{\Theta}$ is N*N matrix which means correlation graph. To compute $\hat{\Theta}$ for each window, our algorithm update Θ from initial state Θ_0 with iterative algorithm.

Speeding up of Computing Correlation Graph

Because sensors' values are time series data, so neighbor windows are similar to each other. We could eliminate the number of steps to convergence by using the previous window's result $\hat{\Theta}_T$ as next window's initial state Θ_0 in iterative procedure. We achieved about 4 times faster result than offline version by using this invention. (figure 3 and 5)

The iterative procedure for each window is individual. We parallelized our algorithm for multi-node computers. Our system dispatches the windows to waiting node and achieves load balancing. (figure 4 and 6)

By these inventions, we achieved real-time and scalable correlation-based anomaly detector.

Figure1. Computing “normal correlation graph”

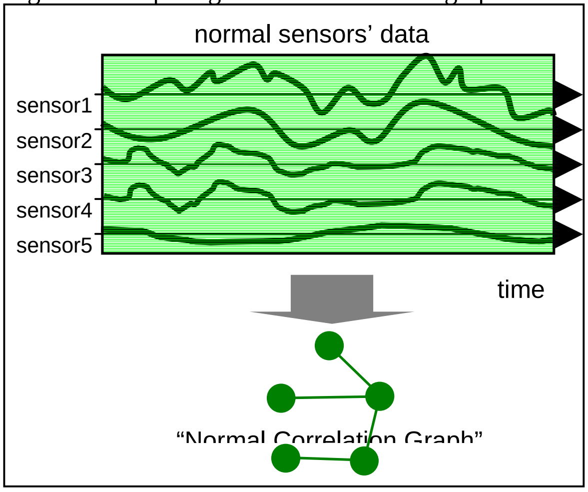


Figure2. Computing real-time correlation graph

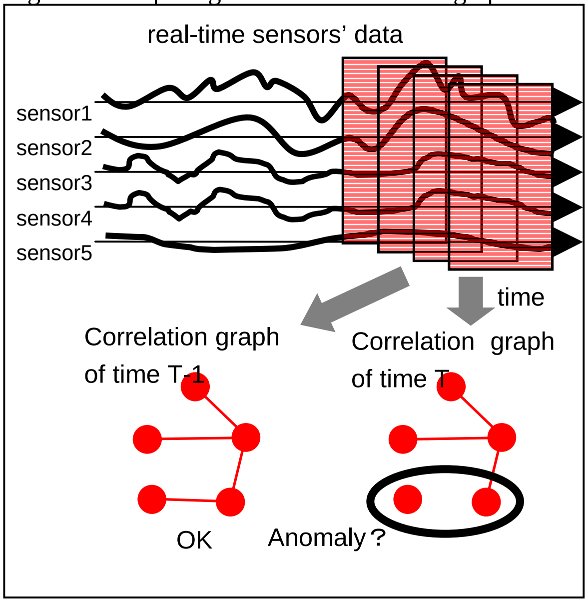
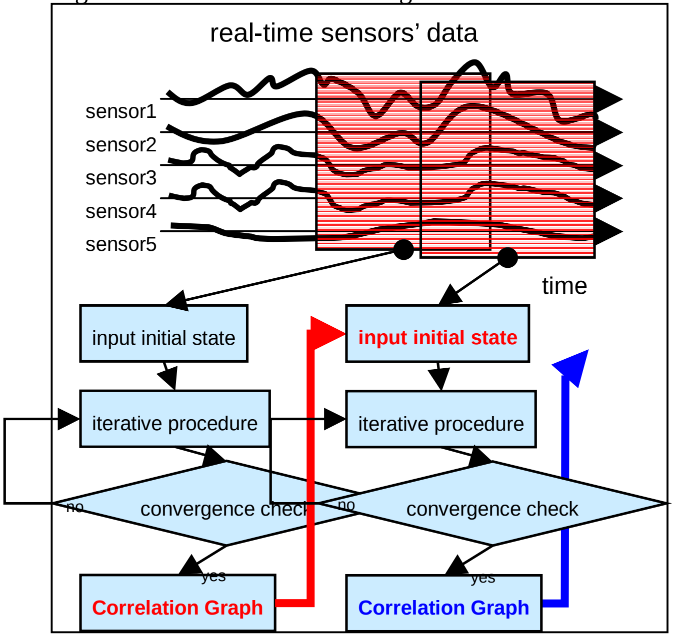


Figure3. The outline of online algorithm



Figur4. The outline of parallel algorithm

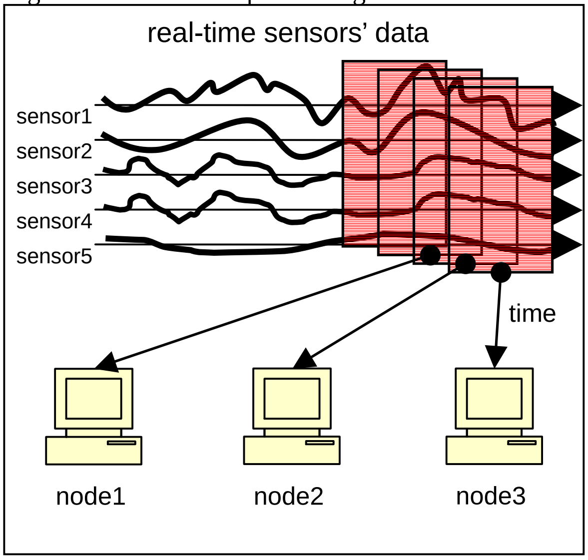


Figure5. Performance comparison of online and offline

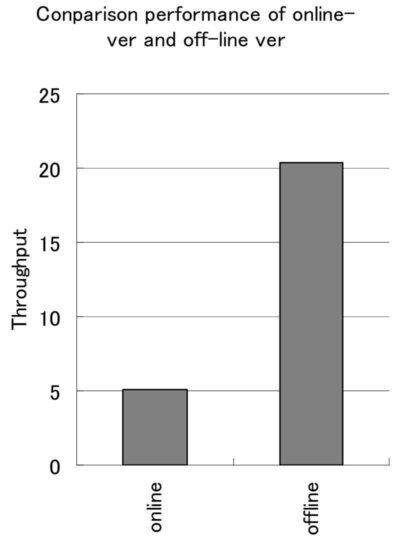


Figure6. # of nodes VS throughput