Anomaly Detection Report

Andrea Buratti, Matteo Carrese

March 2024

1 Introduzione

Il sistema che abbiamo deciso di implementare è un sistema di Anomaly Detection, implementato in C++ e comprendente un database gestito con postgreSQL. Il nostro sistema,composto da vari processi si interfaccia quindi con dei sensori che generano vari flussi di dati.

2 Requisiti Di Sistema

I requisiti di cui necessita il nostro sistema sono quindi un "Server" che risponderà alle richieste dei nostri processi e si interfaccerà direttamente con il database, e i vari "Client",rappresentati dai processi che si occuperanno di provare a rilevare le varie anomalie,calcolando medie e covarianze e confrontando i dati salvati fino a quel momento.

3 Implementazione

3.1 Le componenti

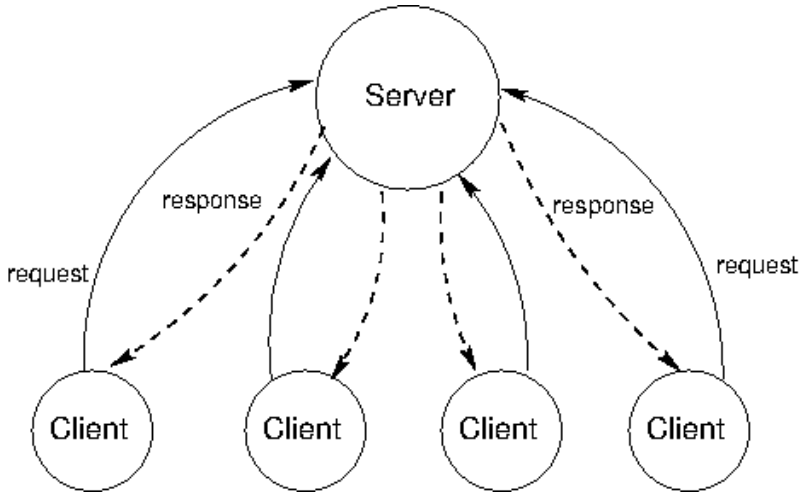

Figure 1: Server-Client implementation

Il software presenta tre componenti principali, che sono il "Server",per rispondere alle richieste dei processi "client" e dei "monitor", che gestiscono nell'effettivo tutte le funzionalità del sistema.

3.2 Server

Il lato "Server" del nostro applicativo è composto da un database gestito tramite postgreSQL, e diverse classi che ci permettono di gestire in modo semplice e conciso i dati. L'intera inizializzazione del database viene effettuata dal processo "Database-set". Il main del processo infatti connette a un database PostgreSQL utilizzando la libreria pqxx, quindi crea un nuovo database se non esiste già e crea delle tabelle all'interno di un altro database chiamato "anomaly_detection". Dopo aver eseguito queste operazioni, chiude le connessioni ai database.

3.3 Monitors

I monitor da noi definiti sono rappresentati dalla cartella include. Essa comprende due header,messi a disposizione per il sistema,che consentono sia di interfacciarsi direttamente con il database, sia di gestire il flusso di dati tramite le classi appropriate.

- database-classes.hpp : l'header mette a disposizione svariate classi che consentono di salvare e caricare i dati in maniera concorde rispetto al

database. Ogni classe presenta una serie di attributi,resi private cosi che l'accesso sia garantito esclusivamente dai metodi specifici di ogni classe.

- database-methods.hpp : l'header è il fulcro dell'implementazione dei monitor. Presenta ogni metodo necessario ad interfacciarsi direttamente con il database,infatti ognuno prende come input almeno il puntatore alla connessione col database. Inoltre si occupa di fornire output riguardo la responsività del sistema.

3.4 Processi

I processi sono il fulcro del nostro applicativo. Essi sono responsabili di alleggerire il payload generale del sistema. L'idea è infatti che ogni processo soddisfa uno ed un solo requisito. Ogni processo ha quindi accesso ai metodi e alle strutture messe a disposizione dai monitor,in questo modo si genera una distinzione netta tra i livelli "client" e "server" del sistema.I processi principali sono tre : "Get-Average" , "Get-Covariance" e "Get-Detection".

3.4.1 Get-Average

Il processo preleva dalle stream redis le finestre contenenti le rilevazioni. Quindi per ogni finestra calcola la media dei valori rilevati e procede a salvare la finestra all'interno del database. Get-Average è composto da 6 file ".cpp":

- main.cpp : Inizializzazione delle variabili: Vengono dichiarate variabili per memorizzare i nomi degli stream (streams), i nomi dei sensori (sensors), i tempi (times) e le finestre temporali (windowSlot). Viene anche inizializzato un puntatore a una struttura di contesto Redis (c). Connessione a Redis: Viene tentata la connessione a Redis utilizzando redisConnect(). Se la connessione fallisce, il programma stampa un messaggio di errore e termina. Inizializzazione del database: Viene chiamata la funzione truncateAverages() per inizializzare il database all'avvio del test. Ciclo principale: L'applicazione entra in un ciclo infinito. Recupero delle medie da Redis: All'interno del ciclo, l'applicazione attende finché l'indicatore "isReady_" su Redis non diventa "yes". Una volta che diventa "yes", l'applicazione chiama la funzione isReady() per impostare l'indicatore su Redis e procedere. Quindi, vengono recuperati i dati di flusso, i sensori e i tempi da Redis utilizzando le funzioni getStreamFromRedis(), getSensor() e getTimes(). Calcolo delle medie: Per ogni stream recuperato, viene chiamata la funzione getAvarage() per calcolare la media dei dati dello stream nell'intervallo di tempo specificato. Le medie calcolate vengono memorizzate nel vettore averages, insieme alle relative finestre temporali nel vettore windowSlot. Connessione al database PostgreSQL: Viene stabilita la connessione al database PostgreSQL utilizzando pqxx::connection. Se la connessione fallisce, il programma stampa un messaggio di errore e termina. Inserimento delle medie nel database: Per ogni media calcolata,

viene creata un'istanza dell'oggetto Average e inserita nel database utilizzando la funzione insertAverage(). Dopo l'inserimento delle medie nel database, viene impostato l'indicatore "isReady2" su Redis per segnalare a "Get-Covariance" che può iniziare il calcolo delle covarianze. Disconnessione dal database: Una volta completate le operazioni sul database, la connessione al database viene chiusa utilizzando disconnect(). Gestione delle eccezioni: Se si verifica un'eccezione durante l'esecuzione delle operazioni, viene stampato un messaggio di errore e il programma termina.

- get-from-redis-1.cpp : contiene le funzioni "getInfoFromRedis()" ,per recuperare informazioni da una struttura di dati Redis di tipo stream denominata "stream_window" utilizzando il comando XRANGE. Restituisce un intero che rappresenta il valore della finestra. Utilizza la libreria hiredis per comunicare con il server Redis tramite il puntatore redisContext c. Esegue il comando XRANGE stream_window - + per ottenere tutti gli elementi dello stream "stream_window". Controlla se la risposta ricevuta è valida. Se reply è NULL, viene stampato un messaggio di errore e il programma esce. Estrae il valore intero dal primo elemento della risposta usando atoi. Libera la memoria allocata per la risposta utilizzando freeReplyObject(reply). "getTimes()" , Utilizza la libreria hiredis per comunicare con il server Redis tramite il puntatore redisContext c. Esegue il comando XRANGE time_window - + per ottenere tutti gli elementi dello stream "time_window". Controlla se la risposta ricevuta è valida. Se reply è NULL, viene stampato un messaggio di errore e il programma esce. Estrae i timestamp dal primo elemento della risposta e li inserisce nel vettore times utilizzando stoi. Libera la memoria allocata per la risposta utilizzando freeReplyObject(reply). Restituisce il vettore di timestamp.

- get-sensor-from-redis.cpp : Utilizza la libreria hiredis per comunicare con il server Redis tramite il puntatore redisContext c. Esegue il comando XRANGE sensor_names - + per ottenere tutti gli elementi dello stream "sensor_names". Controlla se la risposta ricevuta è valida. Se reply è NULL, viene stampato un messaggio di errore e il programma esce. Itera attraverso gli elementi della risposta e estrae i nomi dei sensori, li converte in interi e li inserisce nel vettore sensors. Libera la memoria allocata per la risposta utilizzando freeReplyObject(reply). Restituisce il vettore di nomi dei sensori.

- get-stream-from-redis-2.cpp : Utilizza la libreria hiredis per comunicare con il server Redis tramite il puntatore redisContext c. Esegue il comando XRANGE stream_names - + per ottenere tutti gli elementi dello stream "stream_names". Controlla se la risposta ricevuta è valida. Se reply è NULL, viene stampato un messaggio di errore e il programma esce. Itera attraverso gli elementi della risposta e estrae i nomi degli stream, li inserisce nel vettore streams. Libera la memoria allocata per la risposta

utilizzando freeReplyObject(reply). Restituisce il vettore di nomi degli stream.

- truncate-averages.cpp : Connessione al database: Viene creata una connessione al database PostgreSQL chiamato "anomaly_detection" utilizzando la libreria pqxx. Le informazioni di connessione sono specificate tramite i parametri dbname, user, password, hostaddr e port. Se la connessione è stabilita con successo, viene stampato un messaggio che indica il nome del database a cui si è connessi. Esecuzione di truncateAverages() Disconnessione dal database: Una volta completate le operazioni, la connessione al database viene chiusa utilizzando il metodo disconnect(). Viene quindi stampato un messaggio che indica che la connessione è stata chiusa. Gestione delle eccezioni: Tutte le eccezioni che vengono lanciate durante l'esecuzione delle operazioni nel blocco try vengono catturate e stampate tramite std::cerr.

3.4.2 Get-Covariance

Il processo preleva dal database le finestre precedentemente salvate e procede a calcolare la covarianza delle coppie di stream. Get-Covariance è anch'esso composto da 6 file ".cpp":

- get-average-from-db.cpp : Viene dichiarato un vettore averanges per contenere i valori medi recuperati dal database. Viene tentata la connessione al database PostgreSQL utilizzando la libreria pqxx. Le informazioni di connessione sono specificate tramite i parametri dbname, user, password, hostaddr e port. Se la connessione è stabilita con successo, viene stampato un messaggio che indica il nome del database a cui si è connessi. Viene chiamata una funzione chiamata selectAverages() (presumibilmente definita altrove nel codice), passando come argomenti i timestamp times[0] e times[1] per specificare l'intervallo di tempo e la connessione al database. Questa funzione si presume che recuperi i valori medi dal database e li inserisca nel vettore averanges. Una volta completate le operazioni, la connessione al database viene chiusa utilizzando il metodo disconnect(). Viene restituito il vettore averanges contenente i valori medi ottenuti dal database. Se viene generata un'eccezione durante l'esecuzione delle operazioni nel blocco try, l'eccezione viene catturata e stampata tramite std::cerr, e viene restituito un vettore vuoto averanges.

- get-covariance.cpp : engono eseguiti due comandi Redis utilizzando redisCommand() per ottenere tutti gli elementi degli stream streamName1 e streamName2. I risultati vengono memorizzati rispettivamente in reply1 e reply2. Viene inizializzata la variabile covariance per contenere la somma delle covarianze tra i valori dei due stream. Viene iterato attraverso gli elementi degli stream utilizzando un loop for. Ogni elemento contiene otto campi, e l'iterazione avviene ogni otto elementi (indice i

inizia da 3). Per ogni coppia di elementi (uno da reply1 e uno da reply2), viene calcolata la covarianza utilizzando la formula: covariance = $\sum_i (\text{value1}_i - \text{average1}) \cdot (\text{value2}_i - \text{average2})$ Dove value1 e value2 sono i valori dei due stream, e average1 e average2 sono i valori medi corrispondenti. Viene controllato se i valori dei due stream sono "NULL" (indicando la mancanza di dati). Se entrambi i valori sono validi, vengono calcolate e sommate le covarianze. Se uno dei valori è "NULL", viene considerata solo la parte della covarianza relativa al valore valido. Se entrambi i valori sono "NULL", la covarianza viene lasciata invariata. Una volta completato il calcolo della covarianza, vengono liberate le risorse utilizzate per le risposte Redis utilizzando freeReplyObject(). Viene controllato se il valore di j (il numero di coppie di valori considerati) è zero. In tal caso, viene restituita una tupla contenente 0 come valore della covarianza e 1 come indicatore di errore. Altrimenti, viene restituita una tupla contenente la covarianza calcolata divisa per j come valore della covarianza e 0 come indicatore di errore. Infine, la funzione restituisce la tupla contenente la covarianza calcolata e l'indicatore di errore.

- get-stream-from-redis-2.cpp :getStreamFromRedis(redisContext *c): Questa funzione ottiene i nomi degli stream presenti nel database Redis. Utilizza il comando XRANGE per ottenere tutti gli elementi dello stream "stream_names" e li memorizza in un vettore di stringhe. Se il comando non ha successo, la funzione stampa un messaggio di errore e termina il programma. getTimes(redisContext *c): Questa funzione ottiene i tempi dall'intervallo di tempo "time_window" nel database Redis. Utilizza il comando XRANGE per ottenere il primo e l'ultimo elemento dell'intervallo e li converte in interi, che vengono memorizzati in un vettore di interi. Se il comando non ha successo, la funzione stampa un messaggio di errore e termina il programma. isReady2(redisContext *c): Questa funzione controlla lo stato di un'indicatore chiamato "isReady2" nel database Redis. Utilizza il comando GET per ottenere il valore dell'indicatore e lo restituisce come stringa. Se il comando non ha successo, la funzione stampa un messaggio di errore e termina il programma. setIsReady2(redisContext *c): Questa funzione imposta lo stato dell'indicatore "isReady2" nel database Redis su "no". Utilizza il comando SET per impostare il valore dell'indicatore. Se il comando non ha successo, la funzione stampa un messaggio di errore e termina il programma. getSensor(redisContext *c): Questa funzione ottiene i nomi dei sensori presenti nel database Redis. Utilizza il comando XRANGE per ottenere tutti gli elementi dello stream "sensor_names" e li converte in interi, che vengono memorizzati in un vettore di interi. Se il comando non ha successo, la funzione stampa un messaggio di errore e termina il programma.

- get-temp-from-db.cpp: Viene dichiarato un vettore temps di vettori di interi per contenere i dati di temperatura e umidità recuperati dal database. Viene tentata la connessione al database PostgreSQL utilizzando la libreria pqxx. Le informazioni di connessione sono specificate tramite i

parametri dbname, user, password, hostaddr e port. Se la connessione è stabilita con successo, viene stampato un messaggio che indica il nome del database a cui si è connessi. Viene chiamata una funzione chiamata getTempsAverages() (presumibilmente definita altrove nel codice), passando come argomento la connessione al database e il vettore temps. Questa funzione si presume che recuperi i dati di temperatura e umidità dal database e li inserisca nel vettore temps. Una volta completate le operazioni, la connessione al database viene chiusa utilizzando il metodo disconnect(). Viene restituito il vettore temps contenente i dati di temperatura e umidità ottenuti dal database. Se viene generata un'eccezione durante l'esecuzione delle operazioni nel blocco try, l'eccezione viene catturata e stampata tramite std::cerr, e viene restituito il vettore temps vuoto.

- main.cpp: Viene stabilita una connessione a un'istanza di Redis su 127.0.0.1 sulla porta 6379. Se la connessione a Redis fallisce, viene stampato un messaggio di errore e il programma termina. Viene chiamata la funzione truncateCovariance(), che tronca la tabella delle covarianze nel database PostgreSQL. Il programma entra in un loop infinito. All'interno del loop, il programma attende fino a quando l'indicatore "isReady2" in Redis diventa "yes". Una volta che l'indicatore diventa "yes", viene chiamata la funzione setIsReady2() per impostarlo su "no", in modo che altre istanze del programma sappiano che i calcoli stanno per iniziare. Viene stampato un messaggio per indicare l'inizio del calcolo delle nuove covarianze. Vengono ottenuti i nomi degli stream Redis, le medie dei dati dal database PostgreSQL, i tempi e i sensori da Redis. Viene iterato su coppie di stream ottenuti. Per ogni coppia, viene calcolata la covarianza utilizzando i valori medi dei due stream e viene inserita nel database PostgreSQL come oggetto Covariance. Se l'indicatore di errore restituito dalla funzione getCovariance() è zero, la covarianza viene inserita nel database con il valore calcolato. Altrimenti, viene inserito un valore di covarianza nullo. Viene stampato un messaggio di successo dopo l'inserimento delle covarianze nel database PostgreSQL. Viene chiusa la connessione al database PostgreSQL. Il programma ritorna al passaggio 5 e continua il loop. Dopo la chiusura del loop, viene rilasciata la connessione a Redis e il programma termina.

- truncate-averages.cpp: Viene tentata la connessione al database PostgreSQL utilizzando la libreria pqxx. Le informazioni di connessione sono specificate tramite i parametri dbname, user, password, hostaddr e port. Se la connessione è stabilita con successo, viene stampato un messaggio che indica il nome del database a cui si è connessi. Viene chiamata una funzione chiamata truncateCovariances() Viene stampato un messaggio che indica che la tabella delle covarianze è stata troncata con successo. Viene chiusa la connessione al database utilizzando il metodo disconnect(). Se viene generata un'eccezione durante l'esecuzione delle operazioni nel blocco try, l'eccezione viene catturata e stampata tramite std::cerr.

3.4.3 Get-Detection

Get-Detection è infine il processo più corposo,composto da 11 file ".cpp", implementati in modo tale da consentire al processo di effettuare tutte le sue operazioni durante una singola sessione di connessione. Svolge infatti gran parte del lavoro dovendo prelevare medie e covarianze dal database, ed analizzando gli stream di dati in cerca di anomalie.

- main.cpp
- find-anomalies-averages.cpp :La funzione prende in input un vettore di valori (values), una matrice di intervalli (ranges), un vettore di sensori (sensors), un fattore k e una connessione al database PostgreSQL (C). Calcolo della media globale e della deviazione standard: Viene calcolata la media globale dei valori del flusso utilizzando la funzione getGlobal() e la deviazione standard utilizzando la funzione calculateStandardDeviation(). Ricerca di anomalie: Per ogni valore nel vettore values, viene confrontato con la media globale più o meno un certo numero di deviazioni standard (k). Se un valore è al di fuori di questa soglia, viene identificata un'anomalia. Viene stampato un messaggio che indica l'anomalia trovata, che include l'intervallo in cui è stato rilevato l'anomalia e il sensore associato. Inserimento dell'anomalia nel database: Viene creata un'istanza dell'oggetto Average_anomaly per rappresentare l'anomalia trovata e viene inserita nel database PostgreSQL utilizzando la funzione insertAnomalyAverage().
- find-anomalies-covariances.cpp : La funzione prende in input un vettore di valori (values), una matrice di intervalli (ranges), una matrice di coppie di sensori (sensors), un fattore k e una connessione al database PostgreSQL (C). Calcolo della media globale e della deviazione standard: Viene calcolata la media globale dei valori di covarianza utilizzando la funzione getGlobal1() e la deviazione standard utilizzando la funzione calculateStandardDeviation1(). Ricerca di anomalie: Per ogni valore nel vettore values, viene confrontato con la media globale più o meno un certo numero di deviazioni standard (k). Se un valore è al di fuori di questa soglia, viene identificata un'anomalia. Viene stampato un messaggio che indica l'anomalia trovata, che include l'intervallo in cui è stato rilevato l'anomalia e i sensori associati. Inserimento dell'anomalia nel database: Viene creata un'istanza dell'oggetto Covariance_anomaly per rappresentare l'anomalia trovata e viene inserita nel database PostgreSQL utilizzando la funzione insertAnomalyCovariance().
- get-average.cpp : Questa funzione accetta un riferimento a un oggetto di connessione al database PostgreSQL come parametro. Inizialmente, viene dichiarato un vettore di tipo double chiamato "averanges" che conterrà le medie valide recuperate dal database. Successivamente, viene chiamata la funzione selectValidAverages, passando come argomenti la connessione al database e il vettore "averanges". Questa funzione si occupa di recuperare

le medie valide dal database e memorizzarle nel vettore. Infine, il vettore "averanges" viene restituito come risultato della funzione.

- get-covariance.cpp : Questa funzione accetta un riferimento a un oggetto di connessione al database PostgreSQL come parametro. Inizialmente, viene dichiarato un vettore di tipo double chiamato "covariances" che conterrà le covarianze valide recuperate dal database. Successivamente, viene chiamata la funzione selectValidCovariances, passando come argomenti la connessione al database e il vettore "covariances". Questa funzione si occupa di recuperare le covarianze valide dal database e memorizzarle nel vettore. Infine, il vettore "covariances" viene restituito come risultato della funzione.

- get-global.cpp : La funzione getGlobal accetta un vettore di valori come input e calcola la somma di tutti i valori nel vettore. Successivamente, divide la somma per la lunghezza del vettore per calcolare la media globale, che restituisce come risultato. La funzione calculateStandardDeviation accetta un vettore di valori e la media globale calcolata precedentemente come input. Calcola la somma del quadrato delle differenze tra ogni valore nel vettore e la media globale. Successivamente, divide questa somma per la lunghezza del vettore e calcola la radice quadrata del risultato per ottenere la deviazione standard, che restituisce come risultato.

- get-global2.cpp : La funzione getGlobal1 calcola la somma di tutti i valori nel vettore values e quindi divide questa somma per la lunghezza del vettore per ottenere la media globale. Questa media globale viene quindi restituita come risultato. La funzione calculateStandardDeviation1 calcola la somma dei quadrati delle differenze tra ciascun valore nel vettore values e la media globale avarange fornita come parametro. Questa somma viene quindi divisa per la lunghezza del vettore e la radice quadrata del risultato viene calcolata per ottenere la deviazione standard. Questa deviazione standard viene restituita come risultato della funzione.

- Get-sensor.cpp : La funzione getSensorFromAverages accetta un riferimento a un oggetto di connessione al database PostgreSQL come parametro. Successivamente, chiama una funzione selectSensorFromAverages, passando la connessione al database e un vettore di interi sensors. Questa funzione si occupa di recuperare le informazioni sui sensori dai dati delle medie nel database e le memorizza nel vettore sensors. Infine, il vettore sensors viene restituito come risultato della funzione. La funzione getSensorsFromCovariances è simile alla prima, ma si occupa di recuperare le informazioni sui sensori dalle covarianze nel database. Accetta anche un riferimento a un oggetto di connessione al database PostgreSQL come parametro e restituisce un vettore di vettori di interi sensors, che contiene le informazioni sui sensori recuperate dalle covarianze nel database.

- get-stream-from-db-cov.cpp : La funzione getStreamCovariance accetta un riferimento a un oggetto di connessione al database PostgreSQL come

parametro. All'interno della funzione, viene creato un oggetto pqxx::work utilizzando la connessione al database C. Successivamente, viene eseguita una query SQL per selezionare le colonne temp_start e temp_end dalla tabella covariances. Il risultato della query viene memorizzato nell'oggetto R. La funzione itera su tutte le righe del risultato della query e per ciascuna riga crea un vettore di interi temp che contiene i valori delle colonne temp_start e temp_end. Questo vettore temp viene quindi aggiunto al vettore bidimensionale temps, che contiene tutte le informazioni sulle finestre di tempo associate alle covarianze. Infine, viene eseguito il commit della transazione utilizzando l'oggetto W e il vettore temps viene restituito come risultato della funzione.

- get-stream-from-db.cpp : La funzione inizia creando una connessione al database PostgreSQL utilizzando le credenziali specificate. Successivamente, chiama la funzione selectSensorFromAverages per recuperare le informazioni sui sensori associati ai dati di medie dal database. Queste informazioni vengono memorizzate nel vettore temps. Infine, la connessione al database viene chiusa e il vettore temps, contenente le informazioni sui flussi di dati, viene restituito come risultato della funzione.

- get-temp-from-db.cpp : Accetta un riferimento a un oggetto di connessione al database PostgreSQL come parametro. Successivamente, chiama la funzione getTempsAverages2 passando la connessione al database e un vettore di vettori di interi temps. Questa funzione si occupa di recuperare le informazioni sulle temperature dal database e le memorizza nel vettore temps. Infine, il vettore temps contenente le informazioni sulle temperature viene restituito come risultato della funzione.

4 Testing

Per la fase di Testing delle performance abbiamo deciso di analizzare 6 SET di dati,prendendo in considerazione 4,8 e 10 sensori, prima con una finestra da 10 e poi con una finestra da 100 valori.

4.0.1 1°SET – 4x10

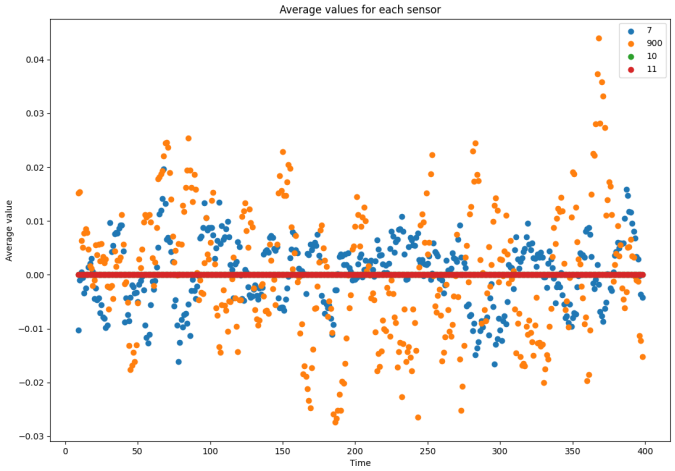

Figure 2: Average: 4 sensori, finestra di dimensione 10

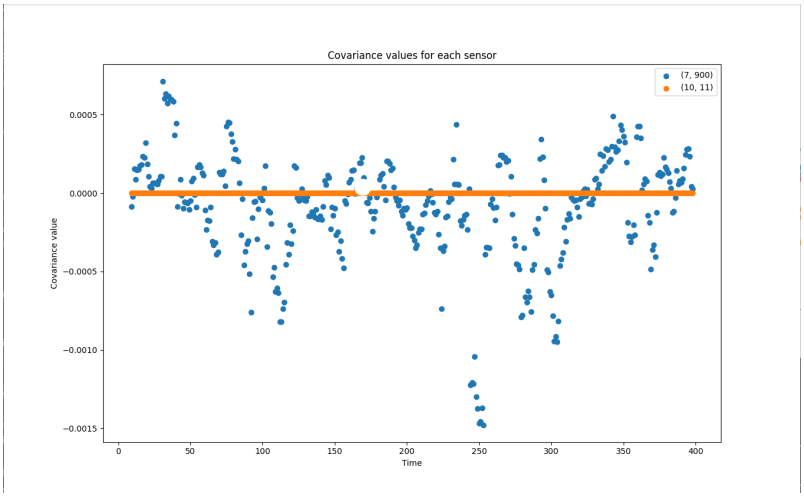

Figure 3: Covariance: 4 sensori, finestra di dimensione 10

4.0.2 2°SET – 4x100

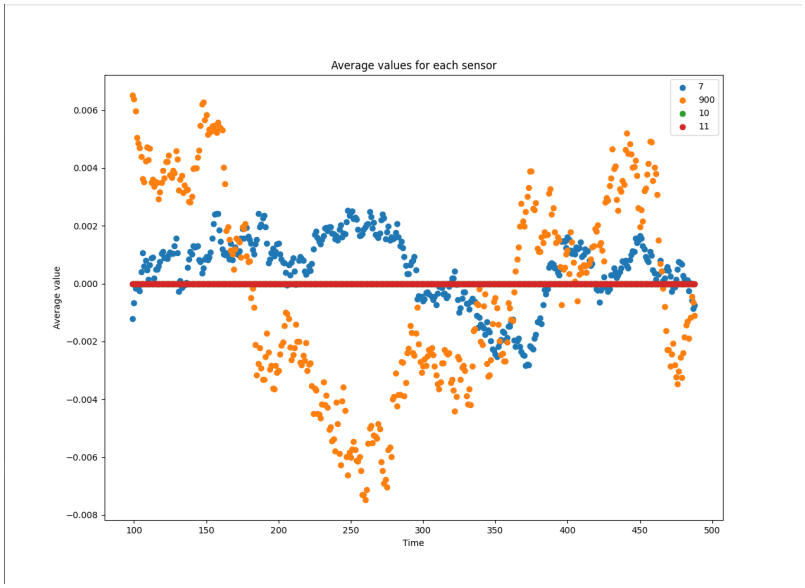

Figure 4: Average: 4 sensori, finestra di dimensione 100

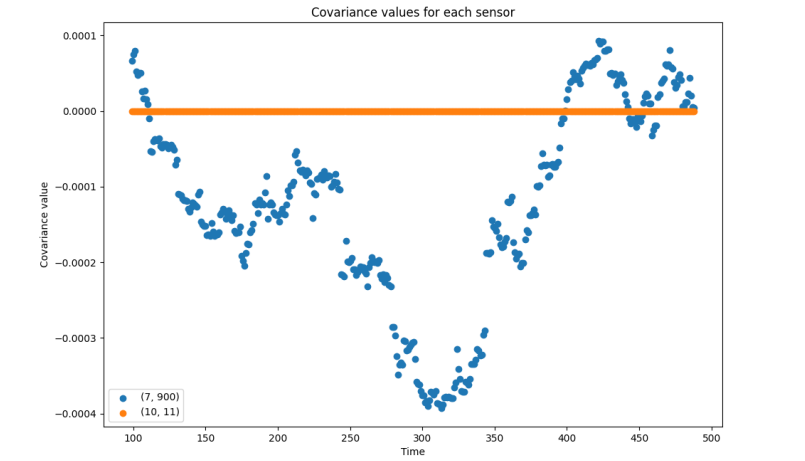

Figure 5: Covariance: 4 sensori, finestra di dimensione 100

4.0.3 3°SET – 8x10

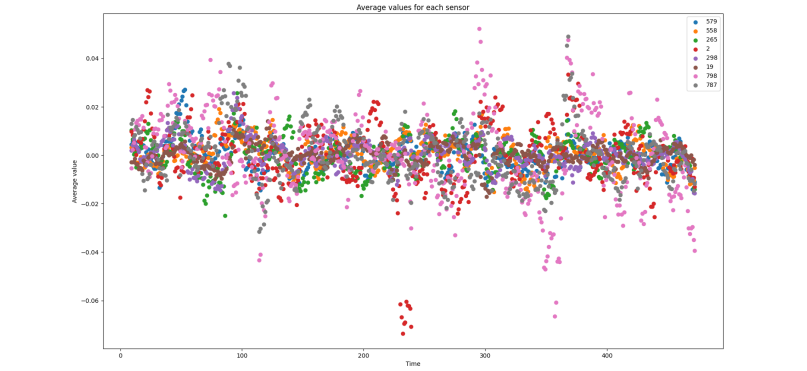

Figure 6: Average : 8 sensori, finestra di dimensione 10

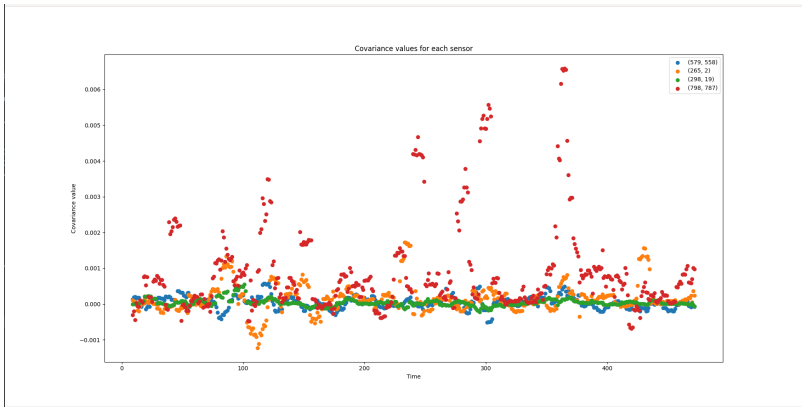

Figure 7: Covariance : 8 sensori, finestra di dimensione 10

4.0.4 4°SET – 8x100

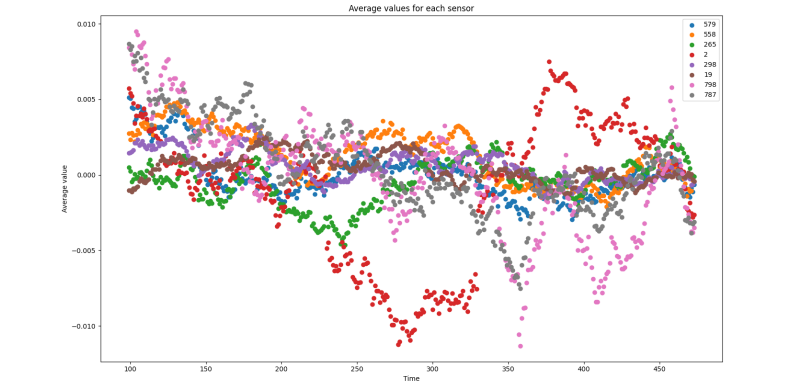

Figure 8: Average : 8 sensori, finestra di dimensione 100

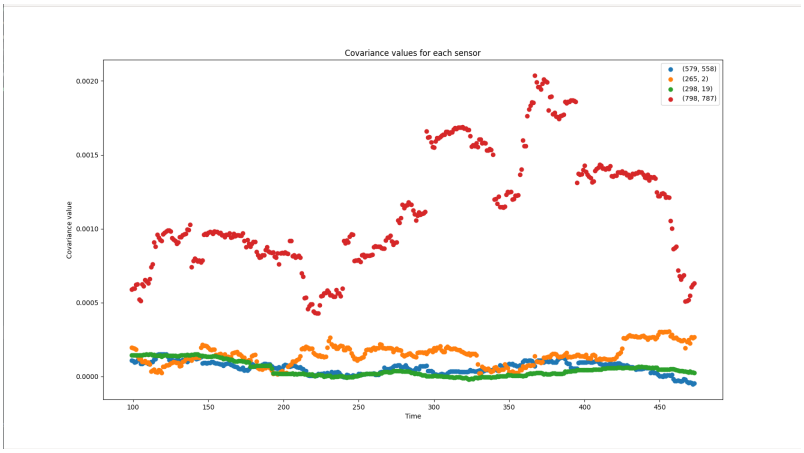

Figure 9: Covariance: 8 sensori, finestra di dimensione 100

4.0.5 5°SET – 10x10

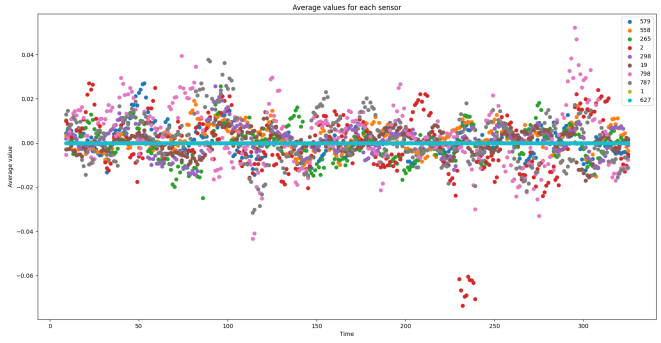

Figure 10: Average: 10 sensori, finestra di dimensione 10

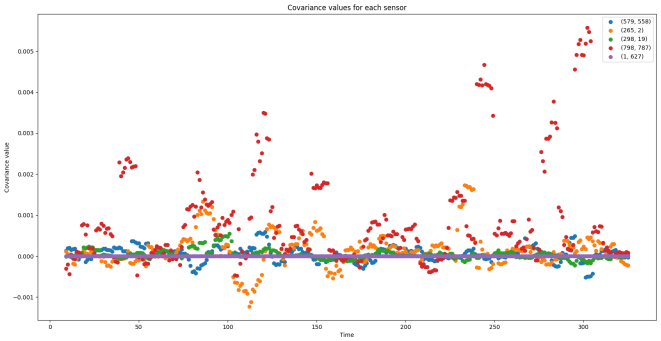

Figure 11: Covariance: 10 sensori, finestra di dimensione 10

4.0.6 6°SET – 10x100

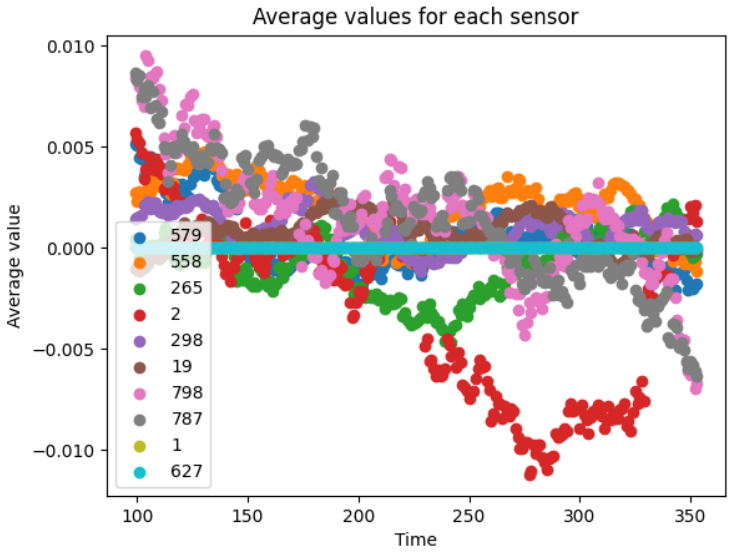

Figure 12: Average: 10 sensori, finestra di dimensione 100

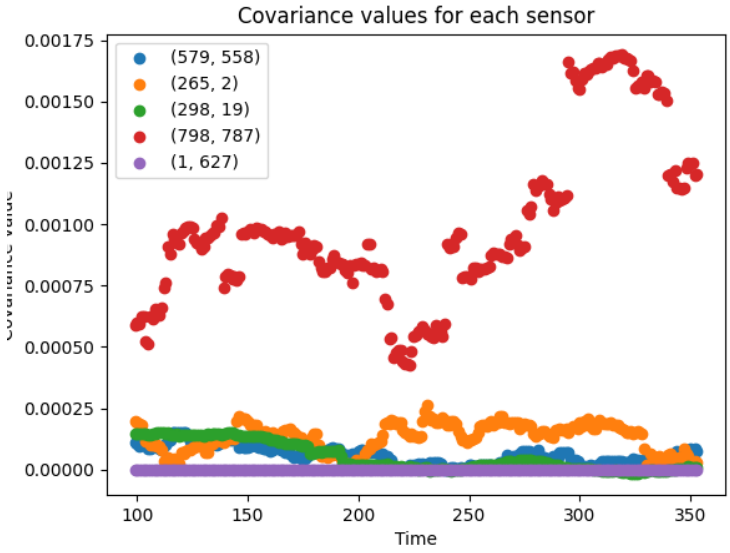

Figure 13: Covariance: 10 sensori, finestra di dimensione 100