Modifiche codice

**Asynchronous publish:** we modified the way in which we publish the data on the cluster from a fire-and-forget way (default in Kafka) to an Asynchronous way. It improves the throughput and if the message doesn’t arrive an exception will be thrown.

**Asynchronous storage:** we imported the asynchronous storage driver for mongo DB in order to not wait the DB when we call the method to store (insertOne()).

**Commit management:** we modified the Kafka commit management from an automatically management to a manual asynchronous management. Doing this we are sure that if something happens and the data will not successfully store on Mongo DB, the whole data will be available on the Kafka cluster. In fact we commit if and only if we successfully store the data.

Committing asynchronous we improve the throughput of the system.

**Parallelization:** after some tests, we discover that for the actual throughput we don’t need to parallelize the Kafka Client-Storing. Will be always possible to run more instances of the client.

|  |  |
| --- | --- |
| **Assumption** | **Description** |
| Network availability | We assume that we have network coverage across the monitored areas. |
| Microcontrollers | We assume that we can use a microcontrollers to manage each sensor and each actuator. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Risk** | **Date the risk is identified** | **Date the risk is resolved** | **Explanation on how the risk has been managed** |
| Critical messages delivery times  (in case of disaster) | 13/11/2017 | 20/11/2017 | We plan on deploying redundant servers (located in a different geographic zone, which host the application) that will go online in case of active server’s failures. |
| Big data storage | 13/11/2017 | 17/11/2017 | We plan on using a dedicated NoSQL database to handle that amount of data. |
| Sensors failures | 13/11/2017 | 24/11/2017 | We plan on using redundancy of sensors, which will start working in case of failure (of the active sensors). |
| Learning of the Kafka framework | 13/11/2017 | 15/12/2017 | We plan on using Kafka framework to develop the system: we discovered that it is quite simple to learn and use, even if to exploit its advanced features we had to go deeper into the documentation. |
| Multi-database integration | 20/11/2017 | 28//11/2017 | We plan on using 2 databases at the same time in order to store data: the relational one will store structural information and the NoSQL will store raw data (as the two are not communicating and they contain different kind of data we do not have synchronization problems). |
| Hardware Management | 4/01/2018 | - | (se l’hardware si adatta, se c’è corrente...etc) |

Class DIAGRAM description:

The component is composed by the following sub-packages such as:

* Dao: that is equal to the Kafka Client Storing class diagram (miss only the Mongo Db part that isn’t needed).
* Model: that is equal to the Kafka Client Storing class diagram (miss only the Mongo Db part that isn’t needed).

The class **ConsumerManager** contains the methods that allow to consume the messages published on the area topics that are retrieved (using the DAO infrastructure) from the relational database.

The **Utils** class contains general purpose static methods.

The **Main** class contains the initialization of the **properties** objectthat allows to parse properties files that can be used to set configuration variables such as the address of the kafka cluster, the credential to access the databases etc.

It also contains the instantiation of the **ConsumerManager** object.

The **producer Manager** class contains the methods that allow to produce messages in order to trigger the correspondent Actuator. The **ProducerCallback** is a **sub-class** of the **ProducerManager** and allow us to produce messages in asynchronous way.