Information Systems, Analysis and Design
Class Projects (2022-23, Fall Semester)

Projects by Dimitrios Tsoumakos

1. Comparison between Time Series DataBases

In this project, you will be asked to compare the performance of various aspects of two popular time-series database systems. This entails the following aspects that must be tackled by you:

1) **Installation and setup of two specific Time-Series DBs:** Using local or okeanos-based resources you are asked to successfully install and setup the two systems. This also means that if any (or both) of the DBs got a cluster edition (distributed mode) that this will be also available for testing.
2) **Data generation (or discovery of real data) and loading to the two DBs:** Using either a specific data generator, online data or artificially creating data yourself, you should identify and load a significant amount of tuples into the databases. Ideally, loaded data should be: a) big (or as big as possible), not able to fit in main memory, in the order of several GB or millions of records, b) the same data loaded in both databases. Data loading is a process that should be monitored, namely: the time it takes to

load a small, medium and the final amount of data, as well as the storage space it takes in each of the databases (i.e., how efficient data compression is).

3) **Query generation to measure performance:** A set of queries (common to both DBs) must be compiled in order to test the querying performance of the timeseries DB storage and indexing. Depending on the data, queries should target the time dimension in both point and range queries (and multiple ranges and grouping functions, i.e., average, group-by, window queries).
4) **Measurement of relevant performance metrics for direct comparison:** Client process(es) should pose the queries of the previous step and measure the DB performance (query latency, throughput, CPU load if possible, etc). Teams should be careful and compare meaningful statistics in this important step.

Teams undertaking this project can take advantage of existing benchmarking code either in principle or in whole. I strongly suggest looking at the Time Series Benchmark Suite (TSBS, https://github.com/timescale/tsbs), which for some of the DBs contains code for data generation, loading, queries and measurement.
Besides the aforementioned project aspects, you are free to improvise in order to best demonstrate the relative strengths and weaknesses of each system. By the end of your project, you should have a pretty good idea of what a modern time-series DB is, its data and query processing model and convey their strong/weak points. This project has 4 different DB combinations:

ProjectD1a: InfluxDB, QuestDB

ProjectD1b: InfluxDB, TimescaleDB

ProjectD1c: CrateDB, Apache Druid

ProjectD1d: Prometheus, Aerospike (or Cassandra)

2. Comparison between Key-Value NoSQL stores

In this project, you will be asked to compare the performance of various aspects of two popular key-value NoSQL systems. This entails the following aspects that must be tackled by you:

1) **Installation and setup of two specific stores:** Using local or okeanos-based resources you are asked to successfully install and setup the two systems. This also means that if any (or both) of the DBs got a cluster edition (distributed mode) that this will be also available for testing.
2) **Data generation (or discovery of real data) and loading to the two DBs:** Using either a specific data generator, online data or artificially creating data yourself, you should identify and load a significant amount of tuples into the databases. Ideally, loaded data should be: a) big (or as big as possible), not able to fit in main memory, in the order of several GB or millions/billions of records, b) the same data loaded in both databases. Data loading is a process that should be monitored, namely: the time it takes to load a small, medium and the final amount of data versus the cluster resources allocated to the store.
3) **Query generation to measure performance:** A set of queries (common to both DBs) must be compiled in order to test the performance of the

   key-value store. Queries should target the key “column” or attribute in both point and range queries (and multiple ranges and grouping functions).
4) **Measurement of relevant performance metrics for direct comparison:** Client process(es) should pose the queries of the previous step and measure the DB performance (query latency, throughput, CPU load if possible, etc) over different total cluster resources. This means that the queries of the previous step should be posed over i) a different number of DB nodes/workers, ii) different rates of queries/sec. Teams should be careful and compare meaningful statistics in this important step.

Teams undertaking this project can take advantage of existing benchmarking code either in principle or in whole. I strongly suggest consulting/using the YCSB benchmark (https://github.com/brianfrankcooper/YCSB), which for many of the DBs contains code for data generation, loading, queries and measurement.
Besides the aforementioned project aspects, you are free to improvise in order to best demonstrate the relative strengths and weaknesses of each system (ability to answer queries fast, scalability with memory/cores, etc). By the end of your project, you should have a pretty good idea of what a modern key-value store is, its data and query processing model and convey their strong/weak points. This project has 4 different combinations:

Project D2a: HBase, Cassandra

Project D2b: ScyllaDB, Couchbase

Project D2c: Aerospike, Riak KV

Project D2d MongoDB, ArangoDB

3. Distributed Execution of SQL Queries Using Trino

Trino is a distributed SQL query engine designed to query large data sets distributed over one or more heterogeneous data sources. In this project, you will be asked to benchmark its performance over different types of queries, data locations and underlying storage technologies. In more detail, the following aspects must be tackled by you:

1) **Installation and setup of three specific stores:** Using local or okeanos-based resources you are asked to successfully install and setup three open-source storage systems or Databases and Trino. These will be used as the source of data for distributed execution of queries by Trino.
2) **Data generation and loading:** Using a specific data generator, you should identify and load a significant amount of tuples into the three databases. Loaded data should be: a) big (or as big as possible), not able to fit in main memory, in the order of several GB or millions/billions of records, b) Different tables of the data should be stored in different stores, according to a strategy that you will devise. This strategy will entail different ways of distributing tables so that you will be able to measure how Trino behaves.
3) **Query generation to measure performance:** A set of queries must be selected in order to test the performance of Trino and its optimizer. Queries should be diverse enough to include both simple queries (e.g., simple selects) and complex ones (multiple joins and aggregations) and cover some or all the loaded tables.
4) **Measurement of Trino performance:** You should then pose the queries of the previous step and measure the performance (query latency, optimizer plan) over different Trino cluster resources. This means that each of the queries of the previous step should be posed over i) a different number of Trino workers, ii) a different data distribution plan. Our goal is to identify how the Trino execution engine and optimization works under a varying amount of data, data distributed in different engines and with queries of different difficulty.

Teams undertaking this project should take advantage of existing benchmarking code, namely the TPC-DS benchmark (https://www.tpc.org/tpcds/), which contains code for data generation and relevant queries.

Besides the aforementioned project aspects, you are free to improvise in order to best understand the performance of different queries. By the end of your project, you should have a pretty good idea of how query processing in Trino works and

propose better data distribution strategies. This project has 4 different combinations:

Project D3a: PostgreSQL, Cassandra, Redis

Project D3b: MySQL, MongoDB, Redis

Project D3c: PostgreSQL, MySQL, Accumulo

Project D3d: MongoDB, Cassandra, Redis

4. Prediction of ML Operator Performance

Machine Learning operators executed in big data analytics runtimes (e.g., Apache Spark, Apache Flink) are often complex code that requires a significant amount of time to complete over data of large volume. In this project, you will be asked to take multiple measurements of how the execution of an operator progresses over time and then utilize a learning algorithm in order to create a model that will allow you to predict its performance without even executing it. In more detail, the following aspects must be tackled by you:

1) **Installation and setup of a specific runtime:** Using local or okeanos-based resources you are asked to successfully install and setup an open-source, distributed analytics engine, where different operators can be executed. The option of Apache Spark (or Flink or Apache Heron) is strongly encouraged.
2) **Operators for modeling:** A minimum of three operators must be selected in order to model their performance. Operators should ideally be diverse but also in the same family (i.e., only graph operators, only batch operators or streaming).
3) **Data generation and loading:** Using an artificial data generator, you should create a number of sample input data for the operators. Input data should be: a) of different sizes (small/medium to quite large, not able to fit in main memory), b) of different structure (e.g., graphs of different type, or data points of different dimensions, etc).
4) **Measurement of performance and modeling step:** You should execute multiple combinations of data to operator and monitor, for each combination, the total running time, min/max/avg CPU and main memory

cluster usage (as well as other useful statistics). This data will be then fed by you into a suitable learner of your choice (e.g., a neural network, regression, random forest, etc) in order to create a model of how the operator behaves (i.e., what is its running time, the total memory it requires, etc). Your goal is to create an accurate prediction model with minimal error in unseen data inputs.

Teams undertaking this project should take advantage of existing operator code for known runtimes such as Spark (https://spark.apache.org/docs/latest/ml-guide.html). Besides the aforementioned project aspects, you are free to improvise in order to best model the performance of different operators. By the end of your project, you should have a pretty good idea of how ML processing in Spark works. This project has 4 different combinations:

Project D4a: Spark - MLLib Operators: k-means, Random forest regression, Word2Vec

Project D4b: Spark - GraphX Operators: Pagerank, Connected Components, Triangle Counting

Project D4c: Spark - MLLib Operators: Decision Tree Classifier, FP-Growth, PCA

Project D4d: Spark - MLLib Operators: Bisecting k-means, Linear Support Vector Machine, Linear regression

Projects by Marios Koniaris

Project M1a Δημιουργία συστήματος συστάσεων με τεχνικές συνεργατικού φιλτραρίσματος (Collaborative filtering)

Ένα σύστημα συστάσεων επιδιώκει να προβλέψει τη «βαθμολόγηση» ή την «προτίμηση» που θα έδινε ένας χρήστης σε ένα στοιχείο. Μια κατηγορία τεχνικών που χρησιμοποιούνται για την διαμόρφωση των συστάσεων αποκαλούνται τεχνικές συνεργατικού φιλτραρίσματος (Collaborative filtering). Οι πιο κοινές τεχνικές συνεργατικού φιλτραρίσματος περιλαμβάνουν τη χρήση ενός πίνακα συσχέτισης που βασίζεται στις ενέργειες του χρήστη, συσχετίζοντας κάθε χρήστη με κάθε αντικείμενο. Ένα μεγάλο πρόβλημα για αυτήν την κατηγορία είναι η επεκτασιμότητα καθώς σε μεγάλα σύνολα δεδομένων ο πίνακας συσχέτισης θα έχει 10^7 και άνω κελιά. Για να ξεπεραστεί αυτό το είδος προβλήματος υπάρχουν τεχνικές για τη μείωση των διαστάσεων του πίνακα συσχέτισης πχ SVD.

Σκοπός της άσκησης είναι η ανάπτυξη ενός μοντέλου συστάσεων (συνεργατικού φιλτραρίσματος) για ανέκδοτα χρησιμοποιώντας το σύνολο δεδομένων Jester https://eigentaste.berkeley.edu/dataset/

Βήματα Υλοποίησης

1. Εξερεύνηση και εξοικείωση με το σύνολο δεδομένων (Βασικές Οπτικοποιήσεις πχ Κατανομή Βαθμολογιών, Κατανομή Χρηστών)
2. Προετοιμασία Δεδομένων: Επιλογή δεδομένων και Κανονικοποίηση (πχ ελάχιστος αριθμός χρηστών ανά αντικείμενο και αντίστροφα)
3. Καθορισμός training/test sets πχ k-fold (training: χρήστες από τους οποίους μαθαίνει το μοντέλο, test: χρήστες στους οποίους προτείνουμε αντικείμενα)
4. Εφαρμογή τεχνικών για την μείωση του πίνακα συσχέτισης πχ SVD σε Apache Spark
5. Έλεγχος αποτελεσμάτων

Project M1b Δημιουργία συστήματος συστάσεων με τεχνικές συνεργατικού φιλτραρίσματος (Collaborative filtering)

Ένα σύστημα συστάσεων επιδιώκει να προβλέψει τη «βαθμολόγηση» ή την «προτίμηση» που θα έδινε ένας χρήστης σε ένα στοιχείο. Μια κατηγορία τεχνικών που

χρησιμοποιούνται για την διαμόρφωση των συστάσεων αποκαλούνται τεχνικές συνεργατικού φιλτραρίσματος (Collaborative filtering). Οι πιο κοινές τεχνικές συνεργατικού φιλτραρίσματος περιλαμβάνουν τη χρήση ενός πίνακα συσχέτισης που βασίζεται στις ενέργειες του χρήστη, συσχετίζοντας κάθε χρήστη με κάθε αντικείμενο. Ένα μεγάλο πρόβλημα για αυτήν την κατηγορία είναι η επεκτασιμότητα καθώς σε μεγάλα σύνολα δεδομένων ο πίνακας συσχέτισης θα έχει 10^7 και άνω κελιά. Για να ξεπεραστεί αυτό το είδος προβλήματος υπάρχουν τεχνικές για τη μείωση των διαστάσεων του πίνακα συσχέτισης πχ SVD.

Σκοπός της άσκησης είναι η ανάπτυξη ενός μοντέλου συστάσεων (συνεργατικού φιλτραρίσματος) για ταινίες χρησιμοποιώντας το σύνολο δεδομένων MovieLens https://grouplens.org/datasets/movielens/25m/

Βήματα Υλοποίησης

1. Εξερεύνηση και εξοικείωση με το σύνολο δεδομένων (Βασικές Οπτικοποιήσεις πχ Κατανομή Βαθμολογιών, Κατανομή Χρηστών)
2. Προετοιμασία Δεδομένων: Επιλογή δεδομένων και Κανονικοποίηση (πχ ελάχιστος αριθμός χρηστών ανά αντικείμενο και αντίστροφα)
3. Καθορισμός training/test sets πχ k-fold (training: χρήστες από τους οποίους μαθαίνει το μοντέλο, test: χρήστες στους οποίους προτείνουμε αντικείμενα)
4. Εφαρμογή τεχνικών για την μείωση του πίνακα συσχέτισης πχ SVD σε Apache Spark
5. Έλεγχος αποτελεσμάτων

Project M1c Δημιουργία συστήματος συστάσεων με τεχνικές συνεργατικού φιλτραρίσματος (Collaborative filtering)

Ένα σύστημα συστάσεων επιδιώκει να προβλέψει τη «βαθμολόγηση» ή την «προτίμηση» που θα έδινε ένας χρήστης σε ένα στοιχείο. Μια κατηγορία τεχνικών που χρησιμοποιούνται για την διαμόρφωση των συστάσεων αποκαλούνται τεχνικές συνεργατικού φιλτραρίσματος (Collaborative filtering). Οι πιο κοινές τεχνικές συνεργατικού φιλτραρίσματος περιλαμβάνουν τη χρήση ενός πίνακα συσχέτισης που βασίζεται στις ενέργειες του χρήστη, συσχετίζοντας κάθε χρήστη με κάθε αντικείμενο. Ένα μεγάλο πρόβλημα για αυτήν την κατηγορία είναι η επεκτασιμότητα καθώς σε μεγάλα σύνολα δεδομένων ο πίνακας συσχέτισης θα έχει 10^7 και άνω κελιά. Για να ξεπεραστεί αυτό το είδος προβλήματος υπάρχουν τεχνικές για τη μείωση των διαστάσεων του πίνακα συσχέτισης πχ SVD.

Σκοπός της άσκησης είναι η ανάπτυξη ενός μοντέλου συστάσεων (συνεργατικού φιλτραρίσματος) για τραγούδια χρησιμοποιώντας το σύνολο δεδομένων Yahoo! Music https://webscope.sandbox.yahoo.com/catalog.php?datatype=r

Βήματα Υλοποίησης

1. Εξερεύνηση και εξοικείωση με το σύνολο δεδομένων (Βασικές Οπτικοποιήσεις πχ Κατανομή Βαθμολογιών, Κατανομή Χρηστών)
2. Προετοιμασία Δεδομένων: Επιλογή δεδομένων και Κανονικοποίηση (πχ ελάχιστος αριθμός χρηστών ανά αντικείμενο και αντίστροφα)
3. Καθορισμός training/test sets πχ k-fold (training: χρήστες από τους οποίους μαθαίνει το μοντέλο, test: χρήστες στους οποίους προτείνουμε αντικείμενα)
4. Εφαρμογή τεχνικών για την μείωση του πίνακα συσχέτισης πχ SVD σε Apache Spark
5. Έλεγχος αποτελεσμάτων

Project M1d Δημιουργία συστήματος συστάσεων με τεχνικές συνεργατικού φιλτραρίσματος (Collaborative filtering)

Ένα σύστημα συστάσεων επιδιώκει να προβλέψει τη «βαθμολόγηση» ή την «προτίμηση» που θα έδινε ένας χρήστης σε ένα στοιχείο. Μια κατηγορία τεχνικών που χρησιμοποιούνται για την διαμόρφωση των συστάσεων αποκαλούνται τεχνικές συνεργατικού φιλτραρίσματος (Collaborative filtering). Οι πιο κοινές τεχνικές συνεργατικού φιλτραρίσματος περιλαμβάνουν τη χρήση ενός πίνακα συσχέτισης που βασίζεται στις ενέργειες του χρήστη, συσχετίζοντας κάθε χρήστη με κάθε αντικείμενο. Ένα μεγάλο πρόβλημα για αυτήν την κατηγορία είναι η επεκτασιμότητα καθώς σε μεγάλα σύνολα δεδομένων ο πίνακας συσχέτισης θα έχει 10^7 και άνω κελιά. Για να ξεπεραστεί αυτό το είδος προβλήματος υπάρχουν τεχνικές για τη μείωση των διαστάσεων του πίνακα συσχέτισης πχ SVD.

Σκοπός της άσκησης είναι η ανάπτυξη ενός μοντέλου συστάσεων (συνεργατικού φιλτραρίσματος) για βιβλία χρησιμοποιώντας το σύνολο δεδομένων Book-Crossing Dataset http://www2.informatik.uni-freiburg.de/~cziegler/BX/

Βήματα Υλοποίησης

1. Εξερεύνηση και εξοικείωση με το σύνολο δεδομένων (Βασικές Οπτικοποιήσεις πχ Κατανομή Βαθμολογιών, Κατανομή Χρηστών)
2. Προετοιμασία Δεδομένων: Επιλογή δεδομένων και Κανονικοποίηση (πχ ελάχιστος αριθμός χρηστών ανά αντικείμενο και αντίστροφα)

3. Καθορισμός training/test sets πχ k-fold (training: χρήστες από τους οποίους μαθαίνει το μοντέλο, test: χρήστες στους οποίους προτείνουμε αντικείμενα)
4. Εφαρμογή τεχνικών για την μείωση του πίνακα συσχέτισης πχ SVD σε Apache Spark
5. Έλεγχος αποτελεσμάτων

Project M2a Σύγκριση Απόδοσης Κατανεμημένων Graph Databases

Σκοπός της άσκησης είναι η ανάπτυξη ενός συστήματος που επιτρέπει την σύγκριση απόδοσης κατανεμημένων Graph Databases. Η σύγκριση αυτή θα πρέπει να αφορά τα ίδια σύνολα δεδομένων, ίδιο φόρτο εργασίας ερωτημάτων, ίδιο περιβάλλον (hardware) καθώς και επικύρωση του αποτελέσματος. Το σύστημα θα πρέπει να εκτελεί τα προκαθορισμένα ερωτήματα σε κάθε σύστημα υπό δοκιμή (System under test – SUT), να μετρά το χρόνο που είναι απαραίτητος για την ολοκλήρωση μιας συγκεκριμένης εργασίας, να υπολογίζει για καθένα ένα από αυτά τις καθορισμένες μετρικές απόδοσης και να παράγει γραφικές παραστάσεις για κάθε μετρική / σύνολο SUT.

Οι λειτουργίες που επιθυμούμε να συγκρίνουμε αφορούν την δημιουργία, ανάγνωση, ενημέρωση και διαγραφή (CRUD).

Τα συστήματα προς προς σύγκριση είναι:

- Neo4j [neo4j.com],
- JanusGraph [https://janusgraph.org/]

Οι μετρικές απόδοσης που θα υπολογίζονται είναι:

- Μέσος χρόνος εκτέλεσης ερωτήματος: Μέσος χρόνος για την εκτέλεση ενός μεμονωμένου ερωτήματος τύπου x πολλές φορές έναντι του SUT.
- Ελάχιστος/Μέγιστος χρόνος εκτέλεσης ερωτήματος: Χρόνος εκτέλεσης κάτω και άνω ορίου για ερωτήματα τύπου x.

Τα σύνολο δεδομένων που θα χρησιμοποιηθούν θα είναι της τάξης 10^n, n =1,2,3,4,5,6 κόμβων.

Βήματα Υλοποίησης

1. Εγκατάσταση και εξοικείωση με τα συστήματα SUT
2. Καθορισμός συνόλου δεδομένων και ερωτημάτων
3. Σχεδίαση και υλοποίηση συστήματος.
4. Υλοποίηση και δοκιμές ελέγχου.

Project M2b Σύγκριση Απόδοσης Κατανεμημένων Graph Databases

Σκοπός της άσκησης είναι η ανάπτυξη ενός συστήματος που επιτρέπει την σύγκριση απόδοσης κατανεμημένων Graph Databases. Η σύγκριση αυτή θα πρέπει να αφορά τα ίδια σύνολα δεδομένων, ίδιο φόρτο εργασίας ερωτημάτων, ίδιο περιβάλλον (hardware) καθώς και επικύρωση του αποτελέσματος. Το σύστημα θα πρέπει να εκτελεί τα προκαθορισμένα ερωτήματα σε κάθε σύστημα υπό δοκιμή (System under test – SUT), να μετρά το χρόνο που είναι απαραίτητος για την ολοκλήρωση μιας συγκεκριμένης εργασίας, να υπολογίζει για καθένα ένα από αυτά τις καθορισμένες μετρικές απόδοσης και να παράγει γραφικές παραστάσεις για κάθε μετρική / σύνολο SUT.

Οι λειτουργίες που επιθυμούμε να συγκρίνουμε αφορούν την δημιουργία, ανάγνωση, ενημέρωση και διαγραφή (CRUD).

Τα συστήματα προς προς σύγκριση είναι:

- Infinitegraph [https://github.com/apache/incubator-hugegraph]
- OrientDB [https://github.com/orientechnologies/orientdb]

Οι μετρικές απόδοσης που θα υπολογίζονται είναι:

- Μέσος χρόνος εκτέλεσης ερωτήματος: Μέσος χρόνος για την εκτέλεση ενός μεμονωμένου ερωτήματος τύπου x πολλές φορές έναντι του SUT.
- Ελάχιστος/Μέγιστος χρόνος εκτέλεσης ερωτήματος: Χρόνος εκτέλεσης κάτω και άνω ορίου για ερωτήματα τύπου x.

Τα σύνολο δεδομένων που θα χρησιμοποιηθούν θα είναι της τάξης 10^n, n =1,2,3,4,5,6 κόμβων.

Βήματα Υλοποίησης

1. Εγκατάσταση και εξοικείωση με τα συστήματα SUT
2. Καθορισμός συνόλου δεδομένων και ερωτημάτων
3. Σχεδίαση και υλοποίηση συστήματος.
4. Υλοποίηση και δοκιμές ελέγχου.

Project M2c Σύγκριση Απόδοσης Κατανεμημένων Graph Databases

Σκοπός της άσκησης είναι η ανάπτυξη ενός συστήματος που επιτρέπει την σύγκριση απόδοσης κατανεμημένων Graph Databases. Η σύγκριση αυτή θα πρέπει να αφορά τα ίδια σύνολα δεδομένων, ίδιο φόρτο εργασίας ερωτημάτων, ίδιο περιβάλλον (hardware) καθώς και επικύρωση του αποτελέσματος. Το σύστημα θα πρέπει να εκτελεί τα προκαθορισμένα ερωτήματα σε κάθε σύστημα υπό δοκιμή (System under test – SUT), να μετρά το χρόνο που είναι απαραίτητος για την ολοκλήρωση μιας συγκεκριμένης εργασίας, να υπολογίζει για καθένα ένα από αυτά τις καθορισμένες μετρικές απόδοσης και να παράγει γραφικές παραστάσεις για κάθε μετρική / σύνολο SUT.

Οι λειτουργίες που επιθυμούμε να συγκρίνουμε αφορούν την δημιουργία, ανάγνωση, ενημέρωση και διαγραφή (CRUD).

Τα συστήματα προς προς σύγκριση είναι:

- Neo4j [neo4j.com],
- Infinitegraph [https://github.com/apache/incubator-hugegraph]

Οι μετρικές απόδοσης που θα υπολογίζονται είναι:

- Μέσος χρόνος εκτέλεσης ερωτήματος: Μέσος χρόνος για την εκτέλεση ενός μεμονωμένου ερωτήματος τύπου x πολλές φορές έναντι του SUT.
- Ελάχιστος/Μέγιστος χρόνος εκτέλεσης ερωτήματος: Χρόνος εκτέλεσης κάτω και άνω ορίου για ερωτήματα τύπου x.

Τα σύνολο δεδομένων που θα χρησιμοποιηθούν θα είναι της τάξης 10^n, n =1,2,3,4,5,6 κόμβων.

Βήματα Υλοποίησης

1. Εγκατάσταση και εξοικείωση με τα συστήματα SUT
2. Καθορισμός συνόλου δεδομένων και ερωτημάτων
3. Σχεδίαση και υλοποίηση συστήματος.
4. Υλοποίηση και δοκιμές ελέγχου.

Project M2d Σύγκριση Απόδοσης Κατανεμημένων Graph Databases

Σκοπός της άσκησης είναι η ανάπτυξη ενός συστήματος που επιτρέπει την σύγκριση απόδοσης κατανεμημένων Graph Databases. Η σύγκριση αυτή θα πρέπει να αφορά τα ίδια σύνολα δεδομένων, ίδιο φόρτο εργασίας ερωτημάτων, ίδιο περιβάλλον (hardware)

καθώς και επικύρωση του αποτελέσματος. Το σύστημα θα πρέπει να εκτελεί τα προκαθορισμένα ερωτήματα σε κάθε σύστημα υπό δοκιμή (System under test – SUT), να μετρά το χρόνο που είναι απαραίτητος για την ολοκλήρωση μιας συγκεκριμένης εργασίας, να υπολογίζει για καθένα ένα από αυτά τις καθορισμένες μετρικές απόδοσης και να παράγει γραφικές παραστάσεις για κάθε μετρική / σύνολο SUT.

Οι λειτουργίες που επιθυμούμε να συγκρίνουμε αφορούν την δημιουργία, ανάγνωση, ενημέρωση και διαγραφή (CRUD).

Τα συστήματα προς προς σύγκριση είνα:

- JanusGraph [https://janusgraph.org/]
- OrientDB [https://github.com/orientechnologies/orientdb]

Οι μετρικές απόδοσης που θα υπολογίζονται είναι:

- Μέσος χρόνος εκτέλεσης ερωτήματος: Μέσος χρόνος για την εκτέλεση ενός μεμονωμένου ερωτήματος τύπου x πολλές φορές έναντι του SUT.
- Ελάχιστος/Μέγιστος χρόνος εκτέλεσης ερωτήματος: Χρόνος εκτέλεσης κάτω και άνω ορίου για ερωτήματα τύπου x.

Τα σύνολο δεδομένων που θα χρησιμοποιηθούν θα είναι της τάξης 10^n, n =1,2,3,4,5,6 κόμβων.

Βήματα Υλοποίησης

1. Εγκατάσταση και εξοικείωση με τα συστήματα SUT
2. Καθορισμός συνόλου δεδομένων και ερωτημάτων
3. Σχεδίαση και υλοποίηση συστήματος.
4. Υλοποίηση και δοκιμές ελέγχου.

Project M3a Σύγκριση Απόδοσης συστημάτων επεξεργασίας γραφημάτων

Σκοπός της άσκησης είναι η ανάπτυξη ενός συστήματος που επιτρέπει την σύγκριση απόδοσης συστημάτων επεξεργασίας γραφημάτων. Η σύγκριση αυτή θα πρέπει να αφορά τα ίδια σύνολα δεδομένων, ίδιο φόρτο εργασίας ερωτημάτων, ίδιο περιβάλλον (hardware) καθώς και επικύρωση του αποτελέσματος. Το σύστημα θα πρέπει να εκτελεί τα προκαθορισμένα ερωτήματα σε κάθε σύστημα υπό δοκιμή (System under test – SUT), να μετρά το χρόνο που είναι απαραίτητος για την ολοκλήρωση μιας συγκεκριμένης εργασίας, να υπολογίζει για καθένα ένα από αυτά τις καθορισμένες

μετρικές απόδοσης και να παράγει γραφικές παραστάσεις για κάθε μετρική / σύνολο SUT.

Οι λειτουργίες που επιθυμούμε να συγκρίνουμε αφορούν εύρεση shortest Path, Degree Centrality, Betweenness centrality και Weakly Connected Components.

Τα συστήματα προς σύγκριση είναι:

- Apache Giraph [https://giraph.apache.org/],
- GraphX [https://spark.apache.org/graphx/]

Οι μετρικές απόδοσης που θα υπολογίζονται είναι:

- Μέσος χρόνος εκτέλεσης ερωτήματος: Μέσος χρόνος για την εκτέλεση ενός μεμονωμένου ερωτήματος τύπου x πολλές φορές έναντι του SUT.
- Ελάχιστος/Μέγιστος χρόνος εκτέλεσης ερωτήματος: Χρόνος εκτέλεσης κάτω και άνω ορίου για ερωτήματα τύπου x.

Βήματα Υλοποίησης

1. Εγκατάσταση και εξοικείωση με τα συστήματα SUT
2. Καθορισμός συνόλου δεδομένων και ερωτημάτων
3. Σχεδίαση και υλοποίηση συστήματος.
4. Υλοποίηση και δοκιμές ελέγχου.

Project M3b Σύγκριση Απόδοσης συστημάτων επεξεργασίας γραφημάτων

Σκοπός της άσκησης είναι η ανάπτυξη ενός συστήματος που επιτρέπει την σύγκριση απόδοσης συστημάτων επεξεργασίας γραφημάτων. Η σύγκριση αυτή θα πρέπει να αφορά τα ίδια σύνολα δεδομένων, ίδιο φόρτο εργασίας ερωτημάτων, ίδιο περιβάλλον (hardware) καθώς και επικύρωση του αποτελέσματος. Το σύστημα θα πρέπει να εκτελεί τα προκαθορισμένα ερωτήματα σε κάθε σύστημα υπό δοκιμή (System under test – SUT), να μετρά το χρόνο που είναι απαραίτητος για την ολοκλήρωση μιας συγκεκριμένης εργασίας, να υπολογίζει για καθένα ένα από αυτά τις καθορισμένες μετρικές απόδοσης και να παράγει γραφικές παραστάσεις για κάθε μετρική / σύνολο SUT.

Οι λειτουργίες που επιθυμούμε να συγκρίνουμε αφορούν εύρεση shortest Path, Degree Centrality, Betweenness centrality και Weakly Connected Components.

Τα συστήματα προς σύγκριση είναι:

- Apache Giraph [https://giraph.apache.org/],

- Apache Flink [https://flink.apache.org/]

Οι μετρικές απόδοσης που θα υπολογίζονται είναι:

- Μέσος χρόνος εκτέλεσης ερωτήματος: Μέσος χρόνος για την εκτέλεση ενός μεμονωμένου ερωτήματος τύπου x πολλές φορές έναντι του SUT.
- Ελάχιστος/Μέγιστος χρόνος εκτέλεσης ερωτήματος: Χρόνος εκτέλεσης κάτω και άνω ορίου για ερωτήματα τύπου x.

Βήματα Υλοποίησης

1. Εγκατάσταση και εξοικείωση με τα συστήματα SUT
2. Καθορισμός συνόλου δεδομένων και ερωτημάτων
3. Σχεδίαση και υλοποίηση συστήματος.
4. Υλοποίηση και δοκιμές ελέγχου.

Project M3c Σύγκριση Απόδοσης συστημάτων επεξεργασίας γραφημάτων

Σκοπός της άσκησης είναι η ανάπτυξη ενός συστήματος που επιτρέπει την σύγκριση απόδοσης συστημάτων επεξεργασίας γραφημάτων. Η σύγκριση αυτή θα πρέπει να αφορά τα ίδια σύνολα δεδομένων, ίδιο φόρτο εργασίας ερωτημάτων, ίδιο περιβάλλον (hardware) καθώς και επικύρωση του αποτελέσματος. Το σύστημα θα πρέπει να εκτελεί τα προκαθορισμένα ερωτήματα σε κάθε σύστημα υπό δοκιμή (System under test – SUT), να μετρά το χρόνο που είναι απαραίτητος για την ολοκλήρωση μιας συγκεκριμένης εργασίας, να υπολογίζει για καθένα ένα από αυτά τις καθορισμένες μετρικές απόδοσης και να παράγει γραφικές παραστάσεις για κάθε μετρική / σύνολο SUT.

Οι λειτουργίες που επιθυμούμε να συγκρίνουμε αφορούν εύρεση shortest Path, Degree Centrality, Betweenness centrality και Weakly Connected Components.

Τα συστήματα προς σύγκριση είναι:

- GraphX [https://spark.apache.org/graphx/]
- Gradoop [https://github.com/dbs-leipzig/gradoop]

Οι μετρικές απόδοσης που θα υπολογίζονται είναι:

- Μέσος χρόνος εκτέλεσης ερωτήματος: Μέσος χρόνος για την εκτέλεση ενός μεμονωμένου ερωτήματος τύπου x πολλές φορές έναντι του SUT.
- Ελάχιστος/Μέγιστος χρόνος εκτέλεσης ερωτήματος: Χρόνος εκτέλεσης κάτω και άνω ορίου για ερωτήματα τύπου x.

Βήματα Υλοποίησης

1. Εγκατάσταση και εξοικείωση με τα συστήματα SUT
2. Καθορισμός συνόλου δεδομένων και ερωτημάτων
3. Σχεδίαση και υλοποίηση συστήματος.
4. Υλοποίηση και δοκιμές ελέγχου.

Project M4a Δημιουργία συστήματος διαχείρισης καναλιών μηνυμάτων (message channels) μεταξύ messaging συστημάτων.

Σκοπός της άσκησης είναι η ανάπτυξη ενός συστήματος που δημιουργεί και διαχειρίζεται πολλαπλές ροές δεδομένων σε πραγματικό χρόνο, μεταξύ ετερογενών συστημάτων. Το σύστημα μέσω κατάλληλου REST API [https://en.wikipedia.org/wiki/Representational_state_transfer] θα δίνει την δυνατότητα καθορισμού source adaptor (σύνδεση στο σύστημα που έχει τα μηνύματα) και destination adaptor (σύνδεση στο σύστημα που θα λαμβάνει τα μηνύματα) και θα υλοποιεί ένα κανάλι επικοινωνίας μεταξύ των 2. Το σύστημα δεν θα μεταφράζει/ επεξεργάζεται τα μηνύματα παρά μόνο θα φροντίζει για την μεταφορά τους από το source (s) στο destination (d). Τα μηνύματα που ανταλλάσσονται θεωρούμε ότι είναι συμβολοσειρές (JSON serialization).

Επίσης μέσω του ίδιου API ο χρήστης θα μπορεί όχι μόνο να δημιουργήσει ένα message channel, αλλά και να επιτελέσει εργασίες διαχείρισης: α) να λάβει όλα τα ενεργά κανάλια, β) να λάβει στατιστικά λειτουργίας για κάποιο κανάλι (πχ χρόνος λειτουργίας, αριθμός απεσταλμένων μηνυμάτων στην μονάδα χρόνου, αριθμός μη απεσταλμένων μηνυμάτων στην μονάδα χρόνου) και γ) να διαγράψει ένα κανάλι επικοινωνίας.

Για την υλοποίηση του συστήματος θα χρησιμοποιηθούν για το source Apache Kafka Broker [https://kafka.apache.org/] και για το producer RabbitMQ Broker [https://www.rabbitmq.com/]

Επίσης συνίσταται η χρήση του Spring Boot [https://spring.io/projects/spring-boot]

Βήματα Υλοποίησης

1. Εγκατάσταση και εξοικείωση με τα συστήματα source και destination
2. Στήσιμο καναλιών υποδοχής/αποστολής μηνυμάτων και δημιουργία εικονικών δεδομένων σε πραγματικό χρόνο.
3. Σχεδίαση λειτουργιών API και μοντέλου δεδομένων συστήματος.

4. Υλοποίηση και δοκιμές ελέγχου.

Χρήσιμοι σύνδεσμοι:

- Messaging Patterns Overview
  https://www.enterpriseintegrationpatterns.com/patterns/messaging/

Project M4b Δημιουργία συστήματος διαχείρισης καναλιών μηνυμάτων (message channels) μεταξύ messaging συστημάτων.

Σκοπός της άσκησης είναι η ανάπτυξη ενός συστήματος που δημιουργεί και διαχειρίζεται πολλαπλές ροές δεδομένων σε πραγματικό χρόνο, μεταξύ ετερογενών συστημάτων. Το σύστημα μέσω κατάλληλου REST API [https://en.wikipedia.org/wiki/Representational_state_transfer] θα δίνει την δυνατότητα καθορισμού source adaptor (σύνδεση στο σύστημα που έχει τα μηνύματα) και destination adaptor (σύνδεση στο σύστημα που θα λαμβάνει τα μηνύματα) και θα υλοποιεί ένα κανάλι επικοινωνίας μεταξύ των 2. Το σύστημα δεν θα μεταφράζει/ επεξεργάζεται τα μηνύματα παρά μόνο θα φροντίζει για την μεταφορά τους από το source (s) στο destination (d). Τα μηνύματα που ανταλλάσσονται θεωρούμε ότι είναι συμβολοσειρές (JSON serialization).

Επίσης μέσω του ίδιου API ο χρήστης θα μπορεί όχι μόνο να δημιουργήσει ένα message channel, αλλά και να επιτελέσει εργασίες διαχείρισης: α) να λάβει όλα τα ενεργά κανάλια, β) να λάβει στατιστικά λειτουργίας για κάποιο κανάλι (πχ χρόνος λειτουργίας, αριθμός απεσταλμένων μηνυμάτων στην μονάδα χρόνου, αριθμός μη απεσταλμένων μηνυμάτων στην μονάδα χρόνου) και γ) να διαγράψει ένα κανάλι επικοινωνίας.

Για την υλοποίηση του συστήματος θα χρησιμοποιηθούν για το source Mosquitto Broker [http://mosquitto.org/] και για το producer Apache Kafka Broker [https://kafka.apache.org/]

Επίσης συνίσταται η χρήση του Spring Boot [https://spring.io/projects/spring-boot]

Βήματα Υλοποίησης

1. Εγκατάσταση και εξοικείωση με τα συστήματα source και destination
2. Στήσιμο καναλιών υποδοχής/αποστολής μηνυμάτων και δημιουργία εικονικών δεδομένων σε πραγματικό χρόνο.
3. Σχεδίαση λειτουργιών API και μοντέλου δεδομένων συστήματος.
4. Υλοποίηση και δοκιμές ελέγχου.

Χρήσιμοι σύνδεσμοι:

- Messaging Patterns Overview
  https://www.enterpriseintegrationpatterns.com/patterns/messaging/

Project M4c Δημιουργία συστήματος διαχείρισης καναλιών μηνυμάτων (message channels) μεταξύ messaging συστημάτων.

Σκοπός της άσκησης είναι η ανάπτυξη ενός συστήματος που δημιουργεί και διαχειρίζεται πολλαπλές ροές δεδομένων σε πραγματικό χρόνο, μεταξύ ετερογενών συστημάτων. Το σύστημα μέσω κατάλληλου REST API [https://en.wikipedia.org/wiki/Representational_state_transfer] θα δίνει την δυνατότητα καθορισμού source adaptor (σύνδεση στο σύστημα που έχει τα μηνύματα) και destination adaptor (σύνδεση στο σύστημα που θα λαμβάνει τα μηνύματα) και θα υλοποιεί ένα κανάλι επικοινωνίας μεταξύ των 2. Το σύστημα δεν θα μεταφράζει/ επεξεργάζεται τα μηνύματα παρά μόνο θα φροντίζει για την μεταφορά τους από το source (s) στο destination (d). Τα μηνύματα που ανταλλάσσονται θεωρούμε ότι είναι συμβολοσειρές (JSON serialization).

Επίσης μέσω του ίδιου API ο χρήστης θα μπορεί όχι μόνο να δημιουργήσει ένα message channel, αλλά και να επιτελέσει εργασίες διαχείρισης: α) να λάβει όλα τα ενεργά κανάλια, β) να λάβει στατιστικά λειτουργίας για κάποιο κανάλι (πχ χρόνος λειτουργίας, αριθμός απεσταλμένων μηνυμάτων στην μονάδα χρόνου, αριθμός μη απεσταλμένων μηνυμάτων στην μονάδα χρόνου) και γ) να διαγράψει ένα κανάλι επικοινωνίας.

Για την υλοποίηση του συστήματος θα χρησιμοποιηθούν για το source RabbitMQ Broker [https://www.rabbitmq.com/] και για το producer Mosquitto Broker [http://mosquitto.org/]

Επίσης συνίσταται η χρήση του Spring Boot [https://spring.io/projects/spring-boot]

Βήματα Υλοποίησης

1. Εγκατάσταση και εξοικείωση με τα συστήματα source και destination
2. Στήσιμο καναλιών υποδοχής/αποστολής μηνυμάτων και δημιουργία εικονικών δεδομένων σε πραγματικό χρόνο.
3. Σχεδίαση λειτουργιών API και μοντέλου δεδομένων συστήματος.
4. Υλοποίηση και δοκιμές ελέγχου.

Χρήσιμοι σύνδεσμοι:

- Messaging Patterns Overview
  https://www.enterpriseintegrationpatterns.com/patterns/messaging/

Project M4d Δημιουργία συστήματος διαχείρισης καναλιών μηνυμάτων (message channels) μεταξύ messaging συστημάτων.

Σκοπός της άσκησης είναι η ανάπτυξη ενός συστήματος που δημιουργεί και διαχειρίζεται πολλαπλές ροές δεδομένων σε πραγματικό χρόνο, μεταξύ ετερογενών συστημάτων. Το σύστημα μέσω κατάλληλου REST API [https://en.wikipedia.org/wiki/Representational_state_transfer] θα δίνει την δυνατότητα καθορισμού source adaptor (σύνδεση στο σύστημα που έχει τα μηνύματα) και destination adaptor (σύνδεση στο σύστημα που θα λαμβάνει τα μηνύματα) και θα υλοποιεί ένα κανάλι επικοινωνίας μεταξύ των 2. Το σύστημα δεν θα μεταφράζει/ επεξεργάζεται τα μηνύματα παρά μόνο θα φροντίζει για την μεταφορά τους από το source (s) στο destination (d). Τα μηνύματα που ανταλλάσσονται θεωρούμε ότι είναι συμβολοσειρές (JSON serialization).

Επίσης μέσω του ίδιου API ο χρήστης θα μπορεί όχι μόνο να δημιουργήσει ένα message channel, αλλά και να επιτελέσει εργασίες διαχείρισης: α) να λάβει όλα τα ενεργά κανάλια, β) να λάβει στατιστικά λειτουργίας για κάποιο κανάλι (πχ χρόνος λειτουργίας, και γ) να διαγράψει ένα κανάλι επικοινωνίας.

Για την υλοποίηση του συστήματος θα χρησιμοποιηθούν για το source OPC-UA Broker και για το producer Mosquitto Broker [http://mosquitto.org/].

Επίσης συνίσταται η χρήση του Spring Boot [https://spring.io/projects/spring-boot]

Βήματα Υλοποίησης

1. Εγκατάσταση και εξοικείωση με τα συστήματα source και destination
2. Στήσιμο καναλιών υποδοχής/αποστολής μηνυμάτων και δημιουργία εικονικών δεδομένων σε πραγματικό χρόνο.
3. Σχεδίαση λειτουργιών API και μοντέλου δεδομένων συστήματος.
4. Υλοποίηση και δοκιμές ελέγχου.

Χρήσιμοι σύνδεσμοι:

Με βάση τις προδιαγραφές του industry 4.0 το OPC-UA αποτελεί ένα πρότυπο για την επικοινωνία και την ανταλλαγή δεδομένων μεταξύ βιομηχανικών asset. [https://en.wikipedia.org/wiki/OPC_Unified_Architecture]

Live OPC-UA demo servers (με δεδομένα που μπορείτε να συνδεθείτε):

- opc.tcp://opcuaserver.com:48010 --- urn:opcuaserver.com:UnifiedAutomation:UaServerCpp

- opc.tcp://uademo.prosysopc.com:53530/OPCUA/SimulationServer --- urn:UADEMO:OPCUA:SimulationServer
- Messaging Patterns Overview https://www.enterpriseintegrationpatterns.com/patterns/messaging/

Projects by Verena Kantere

Θέματα Εργασιών

για το μάθημα

Ανάλυση και Σχεδίαση Πληροφοριακών Συστημάτων

Οκτώβριος 2022

υπό την επίβλεψη της

κας Βασιλικής Καντερέ

σε συνεργασία με τον

Υ.Δ. Παρασκευά Κερασιώτη

1. IoT Live Streaming

Στις συγκεκριμένες εργασίες καλείστε να χρησιμοποιήσετε συνδυαστικά open source εργαλεία για την δημιουργία ενός live streaming συστήματος που θα είναι το prototype ενός πραγματικού IoT συστήματος.

Συγκεκριμένα, θα δημιουργηθεί ένα σύστημα στο οποίο:

1. **Device layer:** Θα δημιουργηθούν εικονικά δεδομένα σε πραγματικό χρόνο.
2. **Messaging Broker Layer:** Θα αποσταλούν σε έναν Message Broker.
3. **Live Streaming Layer:** Θα μετατρέπονται σε ημερήσια ή άλλου είδους μετατροπές με την χρήση frameworks επεξεργασίας δεδομένων σε πραγματικό χρόνο.
4. **Data/Storage Layer:** Θα αποθηκεύονται σε μία NoSQL/Timeseries/In Memory Βάση δεδομένων.
5. **Presentation Layer:** Θα παρουσιάζονται σε Dashboards με την χρήση open source εργαλείων.

Βήματα:

1. Αποστολή εικονικών δεδομένων στο Message Layer (Python, Java, Scala κ.α)

    a. Παραγωγή δεδομένων σειριακά στο χρόνο (15 λεπτών) 1.0/10
    b. Random παραγωγή δεδομένων μη σειριακά στο χρόνο (1 στα 30 δεδομένα) **Late Events** 0.2/10
    c. Αποστολή δεδομένων στον broker, παραμετροποίηση broker, scalability 1.0/10

    2.2/10

2. Επεξεργασία δεδομένων

    a. Aggregations 3.0/10
    b. **Late events** (1.b) detection και αποθήκευση 0.4/10

    3.4/10

3. Αποθήκευση σε Βάση δεδομένων 2.5/10
    - Αποθήκευση raw data
    - Αποθήκευση aggregated Data
    - Αποθήκευση Late Event Data

    2.5/10

4. Απεικόνιση σε Dashboard

    a. Charts + Tables 2.0/10
    b. Table with late events 0.2/10
    c. Websockets για live απεικόνιση δεδομένων 0.8/10

    3.0/10

11.1/10

ΒΚ 1.1) Επεξεργασία ροών δεδομένων σε πραγματικό χρόνο με χρήση των open source κατανεμημένων συστημάτων : Apache Kafka + KStreams, InfluxDB και Grafana+websockets.

Messaging Systems-Broker:	Apache Kafka [1]
Live Streaming Processing System:	KStreams (Apache Kafka) [2]
Timeseries Database:	InfluxDB [3]
Presentation Layer:	Grafana [4]

Στην συγκεκριμένη εργασία καλείστε να χρησιμοποιήσετε συνδυαστικά open source εργαλεία για την δημιουργία ενός live streaming συστήματος που θα είναι το prototype ενός πραγματικού IoT συστήματος. Συγκεκριμένα, θα δημιουργηθεί ένα σύστημα στο οποίο θα στέλνουν εικονικά δεδομένα οι αισθητήρες σε πραγματικό χρόνο στον Kafka Message broker και στην συνέχεια θα μετατρέπονται τα δεδομένα σε ημερήσια ή άλλου είδους μετατροπές με την χρήση των KStreams του Apache Kafka και θα αποθηκεύονται σε μία Timeseries Βάση δεδομένων, την InfluxDB. Τέλος, τα δεδομένα θα παρουσιάζονται σε Dashboards με την χρήση του open source εργαλείου Grafana και με την χρήση websockets θα παρουσιάζονται live..

[1] https://kafka.apache.org/
[2] https://kafka.apache.org/ https://kafka.apache.org/documentation/streams/
[3] https://www.influxdata.com/products/influxdb/
[4] https://grafana.com/

ΒΚ 1.2) Επεξεργασία ροών δεδομένων σε πραγματικό χρόνο με χρήση των open source κατανεμημένων συστημάτων Apache Kafka, Apache Structure Streaming , TimescaleDB και Grafaba+websockets.

Messaging Systems-Broker:	Apache Kafka [1]
Live Streaming Processing System:	Apache Structure Streaming [2]
Timeseries Database:	TimescaleDB [3]
Presentation Layer:	Grafana [4]

Στην συγκεκριμένη εργασία καλείστε να χρησιμοποιήσετε συνδυαστικά open source εργαλεία για την δημιουργία ενός live streaming συστήματος που θα είναι το prototype ενός πραγματικού IoT συστήματος. Συγκεκριμένα, θα δημιουργηθεί ένα σύστημα στο οποίο θα στέλνουν εικονικά δεδομένα οι αισθητήρες σε πραγματικό χρόνο στον Kafka Message broker και στην συνέχεια θα μετατρέπονται τα δεδομένα σε ημερήσια ή άλλου είδους μετατροπές με την χρήση του Apache Structure Streaming και θα αποθηκεύονται σε μία Timeseries Βάση δεδομένων, την TimescaleDB (PostgreSQL timeseries). Τέλος, τα δεδομένα θα παρουσιάζονται σε Dashboards με την χρήση του Grafana και με την χρήση websockets θα παρουσιάζονται live..

[1] https://kafka.apache.org/
[2] https://spark.apache.org/docs/latest/index.html
[3] https://docs.timescale.com/
[4] https://grafana.com/

ΒΚ 1.3) Επεξεργασία ροών δεδομένων σε πραγματικό χρόνο με χρήση των open source κατανεμημένων συστημάτων : Apache Kafka, Apache Flink, Redis Timeseries και Grafana +websockets.

Messaging Systems-Broker:	Apache Kafka [1]
Live Streaming Processing System:	Apache Flink [2]
In Memory Timeseries Database:	Redis (Timeseries) [3]
Presentation Layer:	Grafana [4]

Στην συγκεκριμένη εργασία καλείστε να χρησιμοποιήσετε συνδυαστικά open source εργαλεία για την δημιουργία ενός live streaming συστήματος που θα είναι το prototype ενός πραγματικού IoT συστήματος. Συγκεκριμένα, θα δημιουργηθεί ένα σύστημα στο οποίο θα στέλνουν εικονικά δεδομένα οι αισθητήρες σε πραγματικό χρόνο στον Kafka Message broker και στην συνέχεια θα μετατρέπονται τα δεδομένα σε ημερήσια ή άλλου είδους μετατροπές με την χρήση του Apache Flink και θα αποθηκεύονται σε μία In Memory Timeseries Βάση δεδομένων, την Redis (Timeseries Plugin). Τέλος, τα δεδομένα θα παρουσιάζονται σε Dashboards με την χρήση του open source εργαλείου Grafana και με την χρήση websockets θα παρουσιάζονται live.

[1] https://kafka.apache.org/
[2] https://flink.apache.org/
[3] https://redis.io/ https://oss.redis.com/redistimeseries/
[4] https://grafana.com/

ΒΚ 1.4) Επεξεργασία ροών δεδομένων σε πραγματικό χρόνο με χρήση των open source κατανεμημένων συστημάτων Apache Kafka, Apache Flink, IoTDB και Grafana+websockets.

Messaging Systems-Broker:	Apache Kafka [1]
Live Streaming Processing System:	Apache Flink [2]
Timeseries Database:	IoTDB [3]
Presentation Layer:	Grafana [4]

Στην συγκεκριμένη εργασία καλείστε να χρησιμοποιήσετε συνδυαστικά open source εργαλεία για την δημιουργία ενός live streaming συστήματος που θα είναι το prototype ενός πραγματικού IoT συστήματος. Συγκεκριμένα, θα δημιουργηθεί ένα σύστημα στο οποίο θα στέλνουν εικονικά δεδομένα οι αισθητήρες σε πραγματικό χρόνο στον Kafka Message broker και στην συνέχεια θα μετατρέπονται τα δεδομένα σε ημερήσια ή άλλου είδους μετατροπές με την χρήση του Apache Flink και θα αποθηκεύονται σε μία Timeseries Βάση δεδομένων, την IoTDB. Τέλος, τα δεδομένα θα παρουσιάζονται σε Dashboards με την χρήση του open source εργαλείου Grafana και με την χρήση websockets θα παρουσιάζονται live..

[1] https://kafka.apache.org/

[2] https://flink.apache.org/
[3] https://iotdb.apache.org/
[4] https://grafana.com/

ΒΚ 1.5) Επεξεργασία ροών δεδομένων σε πραγματικό χρόνο με χρήση των open source κατανεμημένων συστημάτων Apache Kafka, Apache Storm, ElasticSearch και Kibana+websockets.

Messaging Systems-Broker:	Apache Kafka [1]
Live Streaming Processing System:	Apache Storm [2]
Database:	ElasticSearch [3]
Presentation Layer:	Kibana [4]

Στην συγκεκριμένη εργασία καλείστε να χρησιμοποιήσετε συνδυαστικά open source εργαλεία για την δημιουργία ενός live streaming συστήματος που θα είναι το prototype ενός πραγματικού IoT συστήματος. Συγκεκριμένα, θα δημιουργηθεί ένα σύστημα στο οποίο θα στέλνουν εικονικά δεδομένα οι αισθητήρες σε πραγματικό χρόνο στον Kafka Message broker και στην συνέχεια θα μετατρέπονται τα δεδομένα σε ημερήσια ή άλλου είδους μετατροπές με την χρήση του Apache Storm και θα αποθηκεύονται σε μία NoSQL Βάση δεδομένων, την ElasticSearch. Τέλος, τα δεδομένα θα παρουσιάζονται σε Dashboards με την χρήση του open source εργαλείου Kibana και με την χρήση websockets θα παρουσιάζονται live..

[1] https://kafka.apache.org/
[2] http://storm.apache.org/index.html
[3] https://www.elastic.co/elasticsearch/
[4] https://www.elastic.co/kibana/

ΒΚ 1.6) Επεξεργασία ροών δεδομένων σε πραγματικό χρόνο με χρήση των open source κατανεμημένων συστημάτων multiple Apache Kafka, Apache Flink, HBase και Grafana+websockets.

Messaging Systems-Broker:	Apache Kafka [1]
Live Streaming Processing System:	Apache Flink [2]
Database:	HBase [3]
Presentation Layer:	Grafana [4]

Στην συγκεκριμένη εργασία καλείστε να χρησιμοποιήσετε συνδυαστικά open source εργαλεία για την δημιουργία ενός live streaming συστήματος που θα είναι το prototype ενός πραγματικού IoT συστήματος. Συγκεκριμένα, θα δημιουργηθεί ένα σύστημα στο οποίο θα στέλνουν εικονικά δεδομένα οι αισθητήρες σε πραγματικό χρόνο στον Kafka Message broker και στην συνέχεια θα μετατρέπονται τα δεδομένα σε ημερήσια ή άλλου είδους μετατροπές με την χρήση του Apache Flink και θα αποθηκεύονται στην HBase. Τέλος, τα δεδομένα θα παρουσιάζονται σε Dashboards με την χρήση του open source εργαλείου Grafana και με την χρήση websockets θα παρουσιάζονται live..

[1] https://kafka.apache.org/
[2] https://flink.apache.org/
[3] https://hbase.apache.org/
[4] https://grafana.com/

ΒΚ 1.7) Επεξεργασία ροών δεδομένων σε πραγματικό χρόνο με χρήση των open source κατανεμημένων συστημάτων RabbitMQ ή (multiple MosquitoMQTT), Apache Spark Structure Streaming, MongoDB και Grafana+websockets.

Messaging Systems-Broker: MosquitoMQTT - RabbitMQ [1] [2]

Live Streaming Processing System: Apache Spark Structure Streaming [3]

Timeseries Database: MongoDB (Timeseries Plugin) [4]

Presentation Layer: Grafana [5]

Στην συγκεκριμένη εργασία καλείστε να χρησιμοποιήσετε συνδυαστικά open source εργαλεία από τις παρακάτω ομάδες εργαλείων για την δημιουργία ενός live streaming συστήματος που θα είναι το prototype ενός πραγματικού IoT συστήματος. Συγκεκριμένα, θα δημιουργηθεί ένα σύστημα στο οποίο θα στέλνουν εικονικά δεδομένα οι αισθητήρες σε πραγματικό χρόνο στον MQTT broker (είτε το RabbitMQ ή πολλαπλά mosquitoMQTT) και στην συνέχεια θα μετατρέπονται τα δεδομένα σε ημερήσια ή άλλου είδους μετατροπές με την χρήση του Apache Spark Structure Streaming και θα αποθηκεύονται σε μία NoSQL Βάση δεδομένων, την MongoDB χρησιμοποιώντας το plugin για Timeseries δεδομένα. Τέλος, τα δεδομένα θα παρουσιάζονται σε Dashboards με την χρήση του open source εργαλείου Grafana και με την χρήση websockets θα παρουσιάζονται live..

[1] https://mosquitto.org/

[2] https://www.rabbitmq.com/
[3] https://spark.apache.org/docs/latest/index.html
[4] https://www.mongodb.com/ https://docs.mongodb.com/manual/core/timeseries-collections/
[5] https://grafana.com/

ΒΚ 1.8) Επεξεργασία ροών δεδομένων σε πραγματικό χρόνο με χρήση των open source κατανεμημένων συστημάτων RabbitMQ ή (multiple MosquitoMQTT), Apache Flink, OpenTSDB και Grafana+websockets.

Messaging Systems-Broker: MosquitoMQTT - RabbitMQ [1] [2]

Live Streaming Processing System: Apache Flink [3]

Timeseries Database: OpenTSDB [4]

Presentation Layer: Grafana [5]

Στην συγκεκριμένη εργασία καλείστε να χρησιμοποιήσετε συνδυαστικά open source εργαλεία από τις παρακάτω ομάδες εργαλείων για την δημιουργία ενός live streaming συστήματος που θα είναι το prototype ενός πραγματικού IoT συστήματος. Συγκεκριμένα, θα δημιουργηθεί ένα σύστημα στο οποίο θα στέλνουν εικονικά δεδομένα οι αισθητήρες σε πραγματικό χρόνο στον MQTT broker (είτε το RabbitMQ ή πολλαπλά mosquitoMQTT) και στην συνέχεια θα μετατρέπονται τα δεδομένα σε ημερήσια ή άλλου είδους μετατροπές με την χρήση του Apache Flink και θα αποθηκεύονται σε μία Timeseries Βάση δεδομένων, την OpenTSDB. Τέλος, τα δεδομένα θα παρουσιάζονται σε Dashboards με την χρήση του open source εργαλείου Grafana και με την χρήση websockets θα παρουσιάζονται live.

[1] https://mosquitto.org/

[2] https://www.rabbitmq.com/
[3] https://flink.apache.org/
[4] http://opentsdb.net/
[5] https://grafana.com/

ΒΚ 1.9) Επεξεργασία ροών δεδομένων σε πραγματικό χρόνο με χρήση των open source κατανεμημένων συστημάτων ActiveMQ, Apache Flink, Apache Cassandra και Grafana.

Messaging Systems-Broker: Apache ActiveMQ [1]

Live Streaming Processing System: Apache Flink [2]

Timeseries Database: Cassandra (Timeseries Plugin) [3]

Presentation Layer: Grafana [4]

Στην συγκεκριμένη εργασία καλείστε να χρησιμοποιήσετε συνδυαστικά open source εργαλεία από τις παρακάτω ομάδες εργαλείων για την δημιουργία ενός live streaming συστήματος που θα είναι το prototype ενός πραγματικού IoT συστήματος. Συγκεκριμένα, θα δημιουργηθεί ένα σύστημα στο οποίο θα στέλνουν εικονικά δεδομένα οι αισθητήρες σε πραγματικό χρόνο στον ActiveMQ broker και στην συνέχεια θα μετατρέπονται τα δεδομένα σε ημερήσια ή άλλου είδους μετατροπές με την χρήση του Apache Flink και θα αποθηκεύονται σε μία NoSQL Βάση δεδομένων, την Cassandra. Τέλος, τα δεδομένα θα παρουσιάζονται σε Dashboards με την χρήση του open source εργαλείου Grafana και με την χρήση websockets θα παρουσιάζονται live.

[1] https://activemq.apache.org/
[2] https://flink.apache.org/
[3] https://cassandra.apache.org/_/index.html
[4] https://grafana.com/

ΒΚ 1.10) Επεξεργασία ροών δεδομένων σε πραγματικό χρόνο με χρήση των open source κατανεμημένων συστημάτων RabbitMQ ή (multiple MosquitoMQTT), Apache Flink, QuestDB και Grafana.

Messaging Systems-Broker: Apache ActiveMQ [1]

Live Streaming Processing System: Apache Flink [2]

Timeseries Database: QuestDB [3]

Presentation Layer: Grafana [4]

Στην συγκεκριμένη εργασία καλείστε να χρησιμοποιήσετε συνδυαστικά open source εργαλεία από τις παρακάτω ομάδες εργαλείων για την δημιουργία ενός live streaming συστήματος που θα είναι το prototype ενός πραγματικού IoT συστήματος. Συγκεκριμένα, θα δημιουργηθεί ένα σύστημα στο οποίο θα στέλνουν εικονικά δεδομένα οι αισθητήρες σε πραγματικό χρόνο στον ActiveMQ broker και στην συνέχεια θα μετατρέπονται τα δεδομένα σε ημερήσια ή άλλου είδους μετατροπές με την χρήση του Apache Flink και θα αποθηκεύονται σε μία Timeseries Βάση δεδομένων, την QuestDB . Τέλος, τα δεδομένα θα παρουσιάζονται σε Dashboards με την χρήση του open source εργαλείου Grafana και με την χρήση websockets θα παρουσιάζονται live.

[1] https://activemq.apache.org/
[2] https://flink.apache.org/
[3] https://questdb.io/
[4] https://grafana.com/

2. IoT Flow Automation

ΒΚ 2.1) Μελέτη εργαλείου Node-Red και υλοποίηση συστήματος έξυπνου σπιτιού.

Το Node-Red είναι ένα visual open source εργαλείο προγραμματισμού με σκοπό την εύκολη διασύνδεση συσκευών, API, και διαδικτυακών υπηρεσιών. Για την δημιουργία των εφαρμογών χρησιμοποιείται flow based programming, δηλαδή η αναπαράσταση των λειτουργιών ως ένα μαύρο κουτί, ή αλλιώς “nodes” όπως τα ονομάζει το Node-RED. Κάθε node, παρέχει μια συγκεκριμένη και καλά καθορισμένη λειτουργία, δηλαδή δεδομένου εισερχόμενων δεδομένων σε ένα node, αυτά επεξεργάζονται και προωθούνται στο επόμενο node. Το συγκεκριμένο καθιστά την δημιουργία εφαρμογών ευκολότερη από παραδοσιακές τεχνικές, καθώς ο χρήστης δίνει περισσότερη έμφαση στην διασύνδεση λειτουργιών παρά την παραγωγή κώδικα. Επίσης η γραφική απεικόνιση των λειτουργιών καθιστά την ανάγνωση και κατανόηση μιας εφαρμογής αρκετά πιο εύκολη από την ανάγνωση πολλών σελίδων κώδικα. Το Node-RED έχει υλοποιηθεί πάνω στο Node.js, και επομένως διατίθεται ως πακέτο του δημοφιλή packet manager NPM.

Στην συγκεκριμένη εργασία καλείστε να δημιουργήσετε ένα mini Smart Home Συστημα με χρήση του Node-Red ως βασικού εργαλείου ανταλλαγής μηνυμάτων και τεχνολογιών όπως:

1. MQTT (Mosquitto MQTT) για την παραλαβή μηνυμάτων [1]
2. Data Flows με Node-Red [2]
3. Αποθήκευση σε RDBMS (MYSQL, PostgreSQL) [3]
4. Απεικόνιση σε Dashboard (Grafana) + websockets [4]
5. Αποστολή μηνυμάτων σε Slack ή άλλο εργαλείο [5]

[1] https://mosquitto.org/
[2] https://nodered.org/
[3] https://www.mysql.com/ https://www.postgresql.org/
[4] https://grafana.com/
[5] https://medium.com/segmentx/writing-a-simple-slack-bot-using-node-red-fast-6bb131af4ea1

ΒΚ 2.2) Μελέτη εργαλείου Node-Red και υλοποίηση συστήματος έξυπνου σπιτιού.

Το Node-Red είναι ένα visual open source εργαλείο προγραμματισμού με σκοπό την εύκολη διασύνδεση συσκευών, API, και διαδικτυακών υπηρεσιών. Για την δημιουργία των εφαρμογών χρησιμοποιείται flow based programming, δηλαδή η αναπαράσταση των λειτουργιών ως ένα μαύρο κουτί, ή αλλιώς “nodes” όπως τα ονομάζει το Node-RED. Κάθε node, παρέχει μια συγκεκριμένη και καλά καθορισμένη λειτουργία, δηλαδή δεδομένου εισερχόμενων δεδομένων σε ένα node, αυτά επεξεργάζονται και προωθούνται στο επόμενο node. Το συγκεκριμένο καθιστά την δημιουργία εφαρμογών ευκολότερη από παραδοσιακές τεχνικές, καθώς ο χρήστης δίνει περισσότερη έμφαση στην διασύνδεση λειτουργιών παρά την παραγωγή κώδικα. Επίσης η γραφική απεικόνιση των λειτουργιών καθιστά την ανάγνωση και κατανόηση μιας εφαρμογής αρκετά πιο εύκολη από την ανάγνωση πολλών σελίδων κώδικα. Το Node-RED έχει υλοποιηθεί πάνω στο Node.js, και επομένως διατίθεται ως πακέτο του δημοφιλή packet manager NPM.

Στην συγκεκριμένη εργασία καλείστε να δημιουργήσετε ένα mini Smart Home Συστημα με χρήση του Node-Red ως βασικού εργαλείου ανταλλαγής μηνυμάτων και τεχνολογιών όπως:

1. Apache Kafka για την ανταλλαγή μηνυμάτων [1]
2. Data Flows με Node-Red [2]
3. Αποθήκευση σε RDBMS (MYSQL, PostgreSQL) και Kafka [3]
4. Απεικόνιση σε Dashboard (Grafana) + websockets [4]
5. Δημιουργία ενός Chat Bot με την χρήση του RedBot [5]

[1] https://kafka.apache.org/
[2] https://nodered.org/
[3] https://www.mysql.com/ https://www.postgresql.org/
[4] https://grafana.com/
[5] http://red-bot.io

ΒΚ 2.3) Μελέτη εργαλείου Node-Red και υλοποίηση συστήματος έξυπνου σπιτιού.

Το Node-Red είναι ένα visual open source εργαλείο προγραμματισμού με σκοπό την εύκολη διασύνδεση συσκευών, API, και διαδικτυακών υπηρεσιών. Για την δημιουργία των εφαρμογών χρησιμοποιείται flow based programming, δηλαδή η αναπαράσταση των λειτουργιών ως ένα μαύρο κουτί, ή αλλιώς “nodes” όπως τα ονομάζει το Node-RED. Κάθε node, παρέχει μια συγκεκριμένη και καλά καθορισμένη λειτουργία, δηλαδή δεδομένου εισερχόμενων δεδομένων σε ένα node, αυτά επεξεργάζονται και προωθούνται στο επόμενο node. Το συγκεκριμένο καθιστά την δημιουργία εφαρμογών ευκολότερη από παραδοσιακές τεχνικές, καθώς ο χρήστης δίνει περισσότερη έμφαση στην διασύνδεση λειτουργιών παρά την παραγωγή κώδικα. Επίσης η γραφική απεικόνιση των λειτουργιών καθιστά την ανάγνωση και κατανόηση μιας εφαρμογής αρκετά πιο εύκολη από την ανάγνωση πολλών σελίδων κώδικα. Το Node-RED έχει υλοποιηθεί πάνω στο Node.js, και επομένως διατίθεται ως πακέτο του δημοφιλή packet manager NPM.

Στην συγκεκριμένη εργασία καλείστε να δημιουργήσετε ένα mini Smart Home Συστημα με χρήση του Node-Red ως βασικού εργαλείου ανταλλαγής μηνυμάτων και τεχνολογιών όπως:

1. MQTT (Mosquitto MQTT) για την παραλαβή μηνυμάτων [1]
2. Data Flows με Node-Red [2]
3. Αποθήκευση σε Elastic Search[3]
4. Απεικόνιση σε Dashboard (Grafana) + websockets [4]
5. Live απεικόνιση Events και αμφίδρομη επικοινωνία με τους αισθητήρες μέσω MQTT ή API calls

[1] https://mosquitto.org/
[2] https://nodered.org/
[3] https://elastic.io/elasticsearch/
[4] https://grafana.com/

Οδηγίες για εργασίες

IoT Live Streaming

Συγκεκριμένα, θα δημιουργηθεί ένα σύστημα στο οποίο:

- **Device layer:** Θα δημιουργηθούν εικονικά δεδομένα σε πραγματικό χρόνο.
- **Messaging Broker Layer:** Θα αποσταλούν σε έναν Message Broker.
- **Live Streaming Layer:** Θα μετατρέπονται σε ημερήσια ή άλλου είδους μετατροπές με την χρήση frameworks επεξεργασίας δεδομένων σε πραγματικό χρόνο.
- **Data/Storage Layer:** Θα αποθηκεύονται σε μία NoSQL/Timeseries/In Memory Βάση δεδομένων.
- **Presentation Layer:** Θα παρουσιάζονται σε Dashboards με την χρήση open source εργαλείων.

Device layer: Δημιουργία εικονικών δεδομένων: (10 αισθητήρες)

- Δεδομένα
    - 2 αισθητήρες θερμοκρασίας: TH1, TH2
        - aggregation method: Average
        - interval : 15min
        - data value range : (12-35 ^{o}C), (12-35 ^{o}C)
    - 2 αισθητήρες μέτρησης ενέργειας κλιματιστικών: HVAC1, HVAC2
        - aggregation method: Sum
        - interval : 15min
        - data value range : (0-100 Wh), (0-200 Wh)
    - 2 αισθητήρες μέτρησης ενέργειας λοιπών ηλεκτρικών συσκευών: MiAC1, MiAC2
        - aggregation method: Sum
        - interval : 15min: (0-150 Wh), (0-200 Wh)
    - 1 αισθητήρα μέτρησης αθροιστικής ενέργειας: Etot
        - aggregation method: Max
        - interval : 1 day (end of day)
        - Κάθε μέρα αύξηση ενέργειας κατά (2600x24 ± 1000 Wh)
    - 1 αισθητήρα ανίχνευσης κίνησης: Mov1
        - aggregation method: (Sum ή Count)
        - interval : δεν έχει (1)
    - 1 αισθητήρα μέτρησης κατανάλωσης νερού: W1
        - aggregation method: Sum
        - interval : 15min
        - data value range: (0-1 lt)
    - 1 αισθητήρα μέτρησης αθροιστικής κατανάλωσης νερού: Wtot
        - aggregation method: Max
        - interval : 1 day (end of day)
        - κάθε μέρα αύξηση κατανάλωσης κατά 110 ± 10 lt

Ενδεικτικά θα πρέπει να παραχθούν ενεργειακά δεδομένα στην εξής μορφή:

TH1 :				
2020-01-01 00:15 | 12.4	2020-01-01 00:30 | 12.6	…	2020-01-01 23:45 | 12.1	
TH2 :				
2020-01-01 00:15 | 22.4	2020-01-01 00:30 | 22.6	…	2020-01-01 23:45 | 22.1	
HVAC1 :				
2020-01-01 00:15 | 0.4	2020-01-01 00:30 | 50.6	…	2020-01-01 23:45 | 0.1	
HVAC2 :				
2020-01-01 00:15 | 100.8	2020-01-01 00:30 | 100.6	…	2020-01-01 23:45 | 0.1	
Etot :				
2020-01-01 00:00 | 1500.8	2020-01-02 00:00 | 1500.8+2600x24+500	….		

Αντίστοιχα θα πρέπει να παραχθούν δεδομένα για κατανάλωση νερού. Για την κίνηση θα έχουμε τυχαία δεδομένα μέσα στην ημέρα της μορφής (να παραχθούν τουλάχιστον 4 με 5 άσοι την ημέρα τυχαία):

Mov1 :			
2020-01-01 13:12 |1	2020-01-01 13:15 |1	2020-01-01 21:01 |1	2020-01-01 23:15 |1

- Τρόπος δημιουργίας τους:

  Python, Java, php , nodeJS, C, Scala ή ότι άλλη γλώσσα προγραμματισμού προτιμάτε.

  Παράδειγμα:

  1. Μία function σε python που:
     a. θα παράγει δεδομένα ανά 1 δευτερόλεπτο για κάθε αισθητήρα 15λεπτου
        i. πχ TH1, 2020-01-01 00:15, 12.4
     b. κάθε φορά θα παράγει δεδομένα για τους αισθητήρες των δεκαπενταλέπτων θα φροντίζουμε η τιμές να είναι μέσα στο επιθυμητό range.
     c. Θα παράγει δεδομένα ανά 96 (24x4x1) δευτερόλεπτα για κάθε αισθητήρα ημέρας
     d. κάθε φορά θα παράγει δεδομένα για τους αισθητήρες των ημερών που θα είναι αθροιστικές (να αυξάνονται με βάση το επιθυμητό range).
        i. Etot, 2020-01-01 00:00, 1500.8
           Etot, 2020-01-02 00:00,1500.8+2600x24+500
     e. Θα παράγει 4 με 5 άσσους μέσα στην ημέρα (στα 96 δευτερόλεπτα) τυχαία.

Προσοχή! Θα πρέπει τα timestamps να είναι αυξανόμενα και όχι να πηγαίνουμε πίσω στον χρόνο, σαν να είχατε αισθητήρες σε ένα κτίριο ή στο σπίτι σας.

- Ετεροχρονισμένα δεδομένα

Μόνο για τον αισθητήρα κατανάλωσης νερού W1 θα δημιουργήσετε ένα επιπλέον script-function που θα παράγει δεδομένα ανά 20 δευτερόλεπτα με timestamp που θα αντιστοιχεί σε 2 μέρες πίσω και ανά 120 δευτερόλεπτα timestamp που θα αντιστοιχεί σε 10 μέρες πίσω.

Πχ. Αν το script ξεκίνησε με ημερομηνία 2020-01-04 00:00 έχει τρέξει για 20 δευτερόλεπτα και είναι να δημιουργήσει δεδομένα για το 15λεπτο 2020-01-04 05:00 θα δημιουργήσει επιπλέον ένα

δεδομένο με timestamp 2020-01-02 05:00 και αντίστοιχα όταν περάσουν 120 δευτερόλεπτα 10 μέρες πίσω.

Σκοπός είναι κατά την δημιουργία των aggregations στο **Live Streaming Layer** να δημιουργηθούν τα κατάλληλα φίλτρα για τον προσδιορισμό των δεδομένων που είναι 10 μέρες πίσω ως late rejected events και των δεδομένων που είναι 2 μέρες πίσω ως late processed events.

Πως τα διαχειριζόμαστε:

Τα 2 μέρες πίσω δεδομένα τα λαμβάνουμε υπόψη στα daily aggregations (εδώ μπορείτε να δείτε τις παραμέτρους που έχουν τα window aggregations) και υπολογίζονται όλα σωστά.

Τα 10 μέρες πίσω φιλτράρονται και αποστέλλονται σε διαφορετικό stream (πχ topic στον kafka) και γράφονται και στην βάση σε διαφορετικό σημείο. Επιπλέον τα δεδομένα αυτά απεικονίζονται και στο Grafana στην συνέχεια στο αντίστοιχο table.

Messaging Broker Layer

Ανάλογα την ομάδα θα χρησιμοποιηθούν διαφορετικοί message brokers για την αποστολή των δεδομένων που παρήχθησαν προηγουμένως.

Θα χρησιμοποιηθούν οι παρακάτω:

Apache Kafka, Mosquito MQTT, RabbitMQ, ActiveMQ

Θα πρέπει σε κάθε περίπτωση να δημιουργήσετε το περιβάλλον και να εγκαταστήσετε τον κάθε broker για να μπορείτε να του στείλετε τα δεδομένα. Θα είναι σημαντικό η επιλογή του τρόπου υλοποίησης να είναι τεκμηριωμένη στην τελική σας εξέταση. Πχ partitions στον kafka , topics στο MQTT ,QoS κα. Θα συζητηθούν και στις διαλέξεις που θα ακολουθήσουν

Live Streaming Layer

Ανάλογα την ομάδα θα χρησιμοποιηθούν διαφορετικά εργαλεία για την live επεξεργασία των δεδομένων που θα αντλούνται αποκλειστικά και μόνο από τους message brokers.

Θα χρησιμοποιηθούν τα παρακάτω:

Apache Flink, KStreams(Kafka), Apache Storm, Apache Spark Structured Streaming

Θα σας ζητηθεί ανά περίπτωση να τραβάτε τα δεδομένα από του message brokers, να τα επεξεργάζεστε στο αντίστοιχο framework και να τα στέλνεται στα επόμενα βήματα της εκάστοτε εργασίας.

Συγκεκριμένα θα σας ζητηθεί να κάνετε τις εξής live πράξεις αξιοποιώντας με τον καλύτερο δυνατό τρόπο που πιστεύετε, την παραμετροποίηση που κάνατε στο message broker:

- AggDay[x] - Aggregations σε day για όλους του αισθητήρες 15λέπτου

- Θα πρέπει να λάβετε υπόψη ότι κάποιοι αισθητήρες παράγουν δεδομένα που πρέπει να αθροιστούν στην μέρα ενώ άλλοι δεδομένα που πρέπει να βρεθεί ο μέσος όρος στην μέρα. Το τελικό timestamp θα πρέπει να είναι το 00:00 της επόμενης ημέρας δλδ (2022-10-20 00:15, 2022-10-20 00:30… 2022-10-20 23:45, 2022-10-21 00:00(επομένης)) να γραφτεί στο 2022-10-21 00:00(επομένης)

- ***Για ευκολία όποιος θέλει να ορίσει αρχή της ημέρας το 00:00 και τέλος το 23:45 είναι δεκτόν, απλά θα πρέπει την ημερήσια τιμή να την γράφετε στην αρχή της ημέρας δλδ στο παραπάνω παράδειγμα στις 2022-10-20 00:00.***

- AggDayDiff[y] - Aggregations διαφοράς στην μέρα για τους αισθητήρες day:

  - Θα πρέπει να βρείτε την διαφορά για κάθε μέρα και να την γράψετε όπως πριν. (2022-10-22 00:00 – 2022-10-21 00:00) να γραφτεί στο 2022-10-22 00:00.

- AggDayRest – Aggregation Διαρροής:

  - AggDayDiff[Etot] - AggDay[HVAC1] - AggDay[HVAC2] - AggDay[MiAC1] - AggDay[MiAC2]

  - AggDayDiff[Wto] – AggDay[W1]

Tip! Να χρησιμοποιήσετε window aggregations όπου είναι δυνατόν

Data/Storage Layer : Βάσεις δεδομένων (Timeseries, Document, Graph, InMemory)

Ανάλογα την ομάδα θα χρησιμοποιηθούν διαφορετικά εργαλεία για την αποθήκευση των δεδομένων που θα γίνεται αποκλειστικά και μόνο με την χρήση των streaming processing frameworks είτε με plugins πάνω σε αυτά (πχ kafka sink connectors).

Tips-Cheats:

Πολλά εργαλεία πχ το Flink παρέχουν sink connectors και σας βοηθάνε να γράφετε τα δεδομένα εύκολα και γρήγορα και είναι συνήθως πού αποτελεσματικά. Τα χρησιμοποιείται αν θέλετε.

Πολλές φορές μπορείτε να βρείτε ανάλογα την εργασία sink connectors από kafka σε βάση. Πως μπορεί να χρησιμοποιηθεί? Θα μπορούσατε να γράφετε τα aggregations στον Kafka και από εκεί να χρησιμοποιείται τον sink kafka connector και να γράφετε στην βάση. (θα πιαστεί σωστό, ακόμα και αν στην εργασία σας δεν έχετε kafka και δεν βρίσκεται άλλο τρόπο να γράψετε τα δεδομένα είναι μία τίμια λύση)

Θα αποθηκεύσετε στην βάση δεδομένων τόσο τα raw data όσο και τα aggregated.

Μπορείτε για τα raw να χρησιμοποιήσετε και άλλες μεθόδους όπως αυτή που αναφέρω παραπάνω μέσω kafka sink connectors!!

Μπορείτε να χρησιμοποιήσετε και άλλα εργαλεία παρόμοια, πχ για την influxdb μπορείτε να χρησιμοποιήσετε το telegraf κ.α

Presentation Layer: Απεικόνιση δεδομένων

Ανάλογα την περίπτωση θα χρησιμοποιηθούν δύο πολύ γνωστά open source εργαλεία Grafana kai Kibana για την παρουσίαση των δεδομένων.

Θα πρέπει σε κάθε περίπτωση να απεικονιστεί ένα chart με κάποια μέτρηση 15λεπτου, 1 chart με κάποια μέτρηση ημέρας, 1 table με τιμές για κάποια μέτρηση 15λεπτου, 1 table με τα late rejected events.

Οτιδήποτε επιπλέον θα σας δώσει επιπλέον βαθμό σε αυτό που έχετε.

Tips

Αν δεν βρίσκεται τρόπο να απεικονίσετε τα δεδομένα στο Grafana με κάποιο έτοιμο plugin μην διστάσετε να χρησιμοποιήσετε το JSON plugin όπου μπορείτε εύκολα να φτιάξετε ένα API και να ζητάτε τα δεδομένα. Δεν χρειάζεται να το κάνετε παραμετρικό αν πάτε σε αυτήν την περίπτωση. Αρκεί να φέρει τα δεδομένα που θέλετε και η μόνη παράμετρος να είναι ο χρόνος που τα ζητάει το Grafana.

Για τα live streaming

Το Grafana έχει από μόνο web sockets και αρκεί να στέλνετε σε συγκεκριμένο path τα δεδομένα που θέλετε. Θα γίνει παρουσίαση και στο μάθημα για το πως γίνεται.

IoT Flow Automation

Θα ακολουθήσουν οδηγίες τις επόμενες ημέρες.