CoherentPaaS

Coherent and Rich PaaS with a Common Programming Model

ICT FP7-611068

Use Cases Design

D9.2

September 2014

Document Information

Scheduled delivery	30.09.2014
Actual delivery	24.10.2014
Version	1.0
Responsible Partner	Neurocom

Dissemination Level:

PU Public

Revision History

Date	Editor	Status	Version	Changes
15.06.2014	Vassilis Spitadakis	Draft	0.1	Draft Table of Contents
5.9.2014	Raquel Pau, Luis Cortesao	Draft	0.2	Input by Sparsity, INRIA and PTIN
9.9.2014	Vassilis Spitadakis	Draft	0.3	Input by Neurocom
12.9.2014	Vassilis Spitadakis	Draft	0.4	Revision of all inputs. Content harmonisation.
22.9.2014	Vassilis Spitadakis	Draft	0.5	Update for internal review
10.10.2014	Raquel Pau, Luis Cortesao, Vassilis Spitadakis	Draft	0.6	Include execution plan diagrams
16.10.2014	François SAVARY Vassilis Spitadakis Raquel Pau Luis Cortesao	Draft	0.7	Revised according to review comments of Quartet FS
22.10.2014	Vassilis Spitadakis, Raquel Pau	Final	1.0	Revised according to comments made by FORTH and MonetDB

Contributors

NEUROCOM, SPARSITY, PTIN, INRIA
Luis Cortesão, Vassilis Spitadakis, George Kotsis, Raquel Pau, Patrick Valduriez

Internal Reviewers

FORTH, QUARTETFS, MonetDB
Angelos Bilas, Antoine Chambille, Martin Kersten

Acknowledgements

Research partially funded by EC 7th Framework Programme FP7/2007-2013 under grant agreement n° 611068.

More information

Additional information and public deliverables of CoherentPaaS can be found at: http://coherentpaas.eu

Glossary of Acronyms

Acronym	Definition
ACID	Atomicity, Consistency, Isolation and Durability
ARPU	Average Revenue Per Unit
BSS	Business Support System
CCF	Communication Control Field
CDI	Contexts and Dependency Injection
CDR	Call Detail Record
CEP	Complex Event Processing
CPaaS	CoherentPaaS
CRUD	Create Read Update and Delete
E2E	End-to-End
HDFS	Hadoop Distributed File System
IDF	Inverse Document Frequency
KPI	Key Performance Indicator
KQI	Key Quality Indicator
M2M	Machine-to-Machine
MVC	Model View Controller
POIs	Points Of Interest
RDBMS	Relational Database Management System
TF	Term Frequency
VTP	Vehicle Telematics Provider

Table of Contents

LIST OF FIGURES

LIST OF TABLES

1 EXECUTIVE SUMMARY

The use cases of CPaaS are experiencing multi-disciplinary data access patterns in terms of load, speed and complexity, under the same application. Therefore, though existing solutions have addressed their needs adequately in selected usage environments, a broader view for their roadmap will face unresolved problems unless their architecture and technologies engaged are revisited. The options of foreseen measures are many; however, final choices should be made emphasizing on the actual needs in terms of key performance indicators and the use of appropriate technology in terms of technical ability, scientific excellence and market maturity.

1.1 Overview of use-cases

A quick overview of the use cases and their overview design are listed below:

- **CDR analytics:** The Telecom Operator Call-Detail-Record (CDR) analytics case produces invoices through the application of all possible combinations of pricing plans to past, current and hypothetical network usage of existing (or hypothetical) customer base. The main reason behind the process is to identify and compare pricing scenarios and estimate their effect into revenue. Moreover, it offers the means to analyze competition and justify further pricing decisions. A wide list of aggregate statistics on those simulated invoices, together with the interactive production of new invoices and their comparison with the simulated charges are among the basic query functions. The number of customers, the usage volumes (function also of a given use duration) and the number of pricing scenarios are the growth factors for the involved data. This imposes limits to scalability, as well as it does not permit the current solution to support interactive scenarios. The major challenge is to extract useful answers to customers, as several functions are required to be performed interactively, many times in front of the customer. That requires, that data subsets should be quickly loaded and comparison analysis should be efficiently performed.

  Columnar stores, together with relational data and in-memory data sets for the interactive data, key-value stores for raw usage data, are the main data store components considered, to enable quicker data loading and high-performance analytics. Several times, the customer selection (which is an integral part of functional parameters) is defined as a sub-graph where links correspond to relations among customers based on relevant usage analysis; henceforth, graph database is enabling efficient customer selections when such criteria are focused.

  CoherentPaas will enable the use of different technologies over the same programming and transaction layer. Otherwise, the employment of so many data stores could be extremely complex. Moreover, consistency across data stores while sizes and complexity grow is challenged; the holistic transaction management component of CoherentPaas aims to achieve coherence and orchestration of actions below the query interface.

- **M2M:** In the Machine-to-Machine case, a vehicle monitoring application has been chosen; many vehicles with sensor devices are involved, continuously increasing in terms of their population and the complexity of information processing. They send information about speed, fuel consumption, current position, engine status and many other vehicle-related metrics. End user applications are responsible to store such

information, to analyze it on regular or extra-ordinary basis, as well as to produce real-time views for the current status and alerting messages upon specific events.

Current implementation will not be able to respond to the growing needs, and the main concern is to follow new solutions that can scale efficiently and respond to burstiness of event traffic. An appropriate design should compensate with both the need of analytical queries in the huge raw data volume, as well as with the real-time information required. The design preferred will produce table views on both the real-time and past analytical data; when necessary, it should be able to combine the query results from both the real-time view (stored in-memory) with analytical data views (from columnar stores). High throughput and quick processing of event data streams will be supported by complex event processing. Responses should also carry data from relational databases. Combining all these frameworks (CEP, in-memory, columnar and relational) need the unifying layer of CPaaS, especially useful when a query function is actually facing the need to provide the union of results across two different stores; one storing analytical data and the second storing the real-time data. Finally, in this use case, CPaaS is not only utilized in the user query process, but also in the system's internal query process that is necessary for building system views for both the real-time and the history (analytical) context.

- **Media Planning:** The use case of Media Planning deals with the analysis of huge amounts of data produced by the social platforms. Such analysis results are very interesting for marketing purposes, but cannot be analyzed with traditional technologies, especially considering the high growth rate of social media data. The analysis usually involves the characterization of social network entities, mainly persons with their influencing power and their followers' network. It also involves the addition of new messages, where the target audience must be identified and the impact of this message needs to be assessed in advance. Current approaches use a single technology store which does not fulfill all the needs of media planning functions.

  The new design considers the use of graph data stores for filtering purposes, document stores for data with flexible sizes, key-value stores to persist complex items related with a given key and relational databases for common information that is necessary to complete the queries' responses. This mixture is expected to efficiently respond to needs; however, the ability to work with all these stores needs to be facilitated by CPaaS, through both its common query language interface and its holistic transaction management.

- **Telco Network Performance Analysis:** The objective of the Real-Time Network Performance Analysis in a Telco Environment use case is to detect network problems before any degradation or unavailability of services occur, by actively supervising it. However, monitoring the whole network implies analyzing big amounts of data in real-time and the current solution does not provide the required degree of scalability that can be found in cloud environments. The existing end-to-end (E2E) system actively detects deterioration in a network using almost real-time KPIs and key quality indicators (KQIs), finds the cause of performance problems and the produced data is always ready to be analyzed through reports and dashboards to check the network's performance.

  The plan is to use CPaaS to provide a rich Platform-as-a-Service that supports several data stores accessible via a uniform programming model and language. Based on an analysis of data access patterns from the use case data store layers - data stores with non-normalized tables (like document database) will be used to store the same type

of data, i.e. raw data; key-value stores for information drill-down functions; other data bases (mainly columnar) will store only aggregated data, i.e. calculated KPIs and KQIs, using a star schema -, different data store alternatives will be accessed and the most adapted will be applied. The CPaaS based platform will have to comply with demanding delay, throughput and data volume requirements. The use of complex event processing will be the key to process and correlate events before the production of the appropriate alarm. Additionally, CPaaS will keep the needed traceability of performance bottlenecks and debugging of errors in applications. It will also enable easier addition of a new data store component or change of an existing component with other solution.

- **Bibliographic Search:** The use case of the Bibliographic Search employs a collection of queries which aim to extract information from bibliographic databases. These data bases collect the researchers' knowledge and are important resources to find experts and institutions working in specific research areas. Mainly, they are used by researchers to know the state of the art of a given topic. However, they can be used for many other purposes such as looking for reviewers, collaboration, similar papers or works and so on. The existing system performs such type of queries using a graph database. However, the current design is not scalable and has no transactions. The system is redesigned, having different datastores with more appropriate structures for each scenario.

  The mixture of relational database (for basic data), documental stores (for textual data), graph data (to store relationships) and key-value datastores for complex items which need to be analyzed and are associated to a common key, provides a promising solution. The design aims to increase queries throughput. The use of CPaaS will simplify development through the use of a common syntax for the integration of inputs and outputs of different datastores. Holistic transaction management will be responsible to combine consistency and coherence across data stores with scalability.

1.2 Challenges for CPaaS

The challenging issues for CPaaS raised by the designed use cases are:

- the transparent usage of better adapted data stores for the big data scenarios that all use cases will have to face; being able to use the best suited data store for the data being stored, while maintaining the top application untouched; This is very important, as existing systems and application are the ones who need to migrate to more powerful environments, and this must be achieved without imposing complete application re-engineering and development times
- while following a compound query and data store approach, using NoSQL, SQL and CEP cloud data technologies, holistic transaction management should guarantee full ACID features even at scale,
- simplifying development over different data technologies; no need to write different code for each data store. This is key for the reduction of complexity and incurred costs which is considered as a major barrier for real-time practices in big data
- enhancing scalability and performance; moving business intelligence and analytics at the front end, within interactive usage contexts is a major achievement expected by the CoherentPaaS enabling layer and the appropriate usage of data stores

- co-existence of real-time analytics with analytical queries on historical data; combination of the two different contexts. Through the usage of the CoherentPaaS CEP system, we expect to reduce framework processing and anticipate real-time alarms and event processing, while benefiting from the ease of usage provided by its SQL-like language and associated database operators.
- Flexibility to change a specific data store with another choice and ability to enable easy future integration of additional cloud data stores in the future

1.3 Common CPaaS Components across use-cases

Having reviewed the designs of all use cases, we have identified a set of components that can be shared across different use cases with a similar architecture. They are described in Section 3.5, 4.5, 5.5, 6.5 and 7.5 and we summarize them below:

1. **Trident-CloudMdsQL**[1]: Common trident plugins to make persistent queries and aggregates inside a trident topology for a specific data store.
2. **Storm components:** Storm has a set of different components types: bolts, spouts and aggregators. Many storm components which are hard to code could be shared and reused between use cases.
3. **Xwork-cloudMdsQL**: XWork (as explained in Section 5.2) has a pluggable architecture and then interceptors, actions and results can be shared across different use cases. A shared component Xwork-cloudMdsQL plugin can inject the query engine into the actions and manage the transaction workflow before executing an action.
4. **Union operation of cloudMdsQL**: An interesting common component is the union operation among the results of two queries, as applied into two parallel data stores (in many cases, the two queries bring similar data from different contexts i.e. a real-time context and a persistence context).
5. **Projection by graph node members**: It is the frequent case, when a table must be filtered based on the result of a graph relation that fulfills a certain criteria.

1.4 Expected Benefits

In terms of anticipated benefits remain the same with the performance goals as prescribed within D.9.1 “Use Case Requirements Analysis”. In brief:

CDR Analytics Use Case: The most important benefits expected by our design are:

- Ability to utilize multiple stores efficiently; until now, tariff simulation development cannot attempt to involve multiple types of datastores, though it is sure that for certain cases, different stores are applicable and preferred in order to achieve the best performance. Involving different technologies will increase complexity and will require a unified framework to address all stores transparently; the need for CloudMdsQL becomes apparent

[1] CloudMdsQL is the name of the common query language interface across heterogeneous data stores in CoherentPaaS

- Ability to apply interactive functions; a feature which was not possible, but with the appropriate datastore employment (in-memory and columnar) it can be accomplished. Efficient load of selected usage data is one need, addressed already by first testing scenarios where the in-memory solution has been used.
- Ability to scale in cloud environments; increasing number of scenarios, number of billing cycles and number of customers will not be an upper bound for the system.

M2M Use Case: The existing architecture imposes serious shortcomings, especially in respect to the ability to accommodate increasing needs, in terms of message throughput, analytical query execution and real-time alarm identification and handling. Therefore, the proposed approach

1. Moves all processes and store facilities into cloud cluster settings,
2. Separates real-time processing from analytical oriented processes
3. Enables the union of the real-time context with the past data results, when that is required
4. Selected the best appropriate data store framework for each data set according to the data needs

Separation of the real-time context and utilization of different data technologies, from Complex Event processing (Storm) and fast OLTP (Derby in-memory) to analytical databases (MonetDB), will be facilitated through CPaaS. In that context the ability to handle larger message throughputs as number of devices increases will not be affected by the load of user queries and dashboard views. The system should handle each event in not more than 4 ms.

Apart from the above, more queries will be defined that will satisfy important business needs. Raw data will be processed efficiently so that the notion of trips will be stored and trip statistics will become also available. In that context, the real-time processing does not only facilitate the creation of real-time views and alarm handling, but they can also supplement raw data with additional information when that is necessary (in case of trip identification, assignment of trip identification to events is a useful data enrichment, filling the gap of sensor data).

Media Planning Use Case:

Nowadays, we have an existing system for Acceso and Media Planning Group with similar queries that is completely build just using a graph database and a documental database. However, the current design is not scalable and has no transactions. We aim to achieve the loading of 1 million documents in less than 12 hours, and the solving of all analytical queries in less than 1 minute. Also, we expect having different datastores (graph, document, key-value and RDBMS) with more appropriate structures for each scenario increase queries throughput.

Telco Network Performance analysis Use Case: Most of the expected benefits will emerge from the transparent usage of better adapted data stores for the big data scenario that this use case will have to face. Previous experiments using Impala over Hadoop showed some improvements regarding the use case requirements. Hence, we expect that using CoherentPaaS we will be able to use the best suited data store for the data being stored, while maintaining the Altaia system relatively untouched. Data stores and associated expected benefits:

- As our initial approach regarding the DBN0 data store used a single flat table in an Oracle database instance, we decided to test the CoherentPaaS platform with the Derby (DQE) and the MongoDB data stores.
- Columnar data stores, such as MonetDB or Derby (DQE) - HBase based -, are a possibility for the DBN1 data store layer, due to the usage of star schemas, which can lead to better compression, and the low delay imposed when selecting information but not while inserting. In-Memory columnar data stores such as ActivePivot will be evaluated but should be take into account the huge amount of data that was required to store in memory, in order to achieve a high query hit ratio;
- Key-Value data stores, such as Eutropia, are well designed for the drill-to-detail queries to the DBN0 data store layer due to its specific data access pattern, i.e. getting single values using a key. We won't use them, however, because that would require having the data replicated, meaning terabytes of additional space.
- Through the usage of the CoherentPaaS CEP system, we might expect to reduce framework processing (alarming) and anticipate real-time alarms, while benefiting from the ease of usage provided by its SQL-like language and associated database operators.

Bibliographic Search Use Case: Nowadays, UPC has an existing system called Sciencea with similar queries that is completely build just using a graph database. However, the current design is not scalable and has no transactions. Also, we expect having different datastores (document, graph) with more appropriate structures for each scenario increase the performance of the number of queries per second. Actually we expect that the system will be able to solve all analytical queries in less than 1 minute. Moreover, having CloudMdsQL simplifies the development because this component integrates inputs and outputs of different datastores using a common syntax. Otherwise, we would need to write a lot of code to parse and sent data from different datastores in each functionality.

1.5 Overview of dependencies

The table below illustrates all dependencies on data components and stores across the use cases (in bold are the most important ones for the use case):

Table 1: Use cases & components dependencies overview

Component / Use case	CEP	Key-Value	Columnar	RDBMS	Document	Graph	In-memory
1. CDR analytics		Hbase	**MonetDB**	Derby		SparkSee	**ActivePivot**
2. M2M	**Storm**		**MonetDB** Hbase	Derby			Derby in-memory
3. Media Planning	Trident/ Storm	Hbase		**Derby**		**SparkSee**	
4. Telco Network Performance	Storm		**MonetDB**	**Derby**	MongoDB		ActivePivot
5. Bibliographic Search	Trident/ Storm	Hbase		Derby	**MongoDB**	**SparkSee**	

In subsequent sections we present in more detail the architecture and design of each use case. Our analysis shows that each use case requires multiple data stores to address

efficiently its needs. The role of CPaaS is key to support the co-existence of many data stores within each use case:

- CloudMdsQL (delivered by CPaaS) will simplify the code of integrating result sets of different datastores. Otherwise, a lot of code will be required to parse and send data from different datastores in each functionality.
- CPaaS based platforms will have to comply with demanding delay, throughput and data volume requirements.
- CPaaS will keep the needed traceability of performance bottlenecks and debugging of errors in applications.
- CPaaS simplifies the logic and use of the query interface and provides a single transaction shell for complex functions across different stores.

2 NOTATION: QUERY PLAN SYMBOLS

Figure 1 and Figure 2 below present the symbols that will be used for the operators in the several queries decomposition diagrams as well as the color selected for the illustration of each data store.

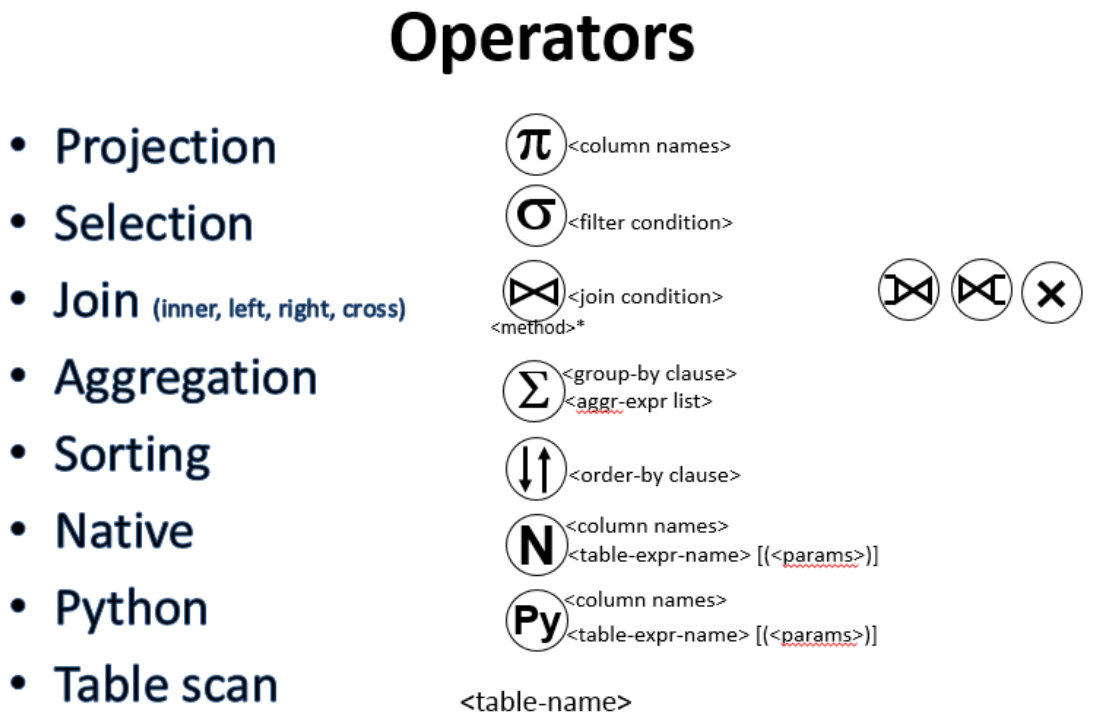

Figure 1: Operators Symbols

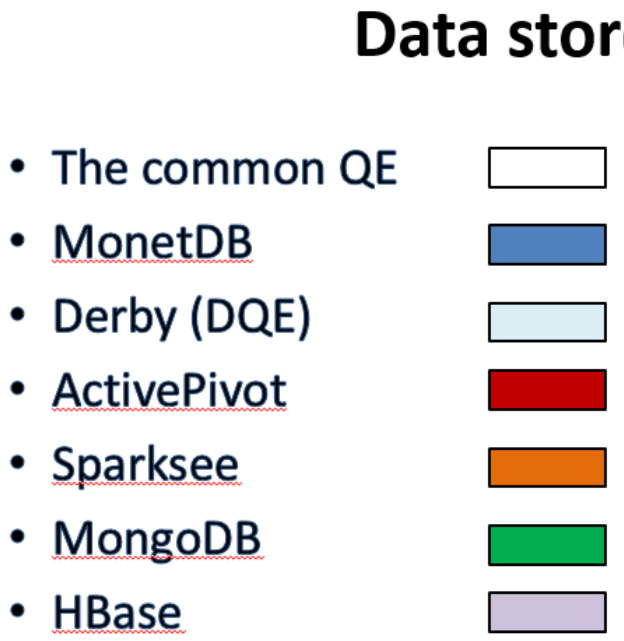

Figure 2: Data store colors, as used in query diagrams henceforth

3 CLOUD TELECOM/M2M USE CASE – CDR ANALYTICS

3.1 CDR Analytics: Use Case Overview

Telecom operators are eager to use software solutions to obtain insight to the trends that are formed (or could be formed) in their customer base. Such solutions engage among others, the comparison and what-if analysis of invoicing output at customer level or customer segment level, based on the knowledge of current agreements and billing operations, past and current network usage, and the assumption of possible alternative charging schemes. *Analytics on high volume data are mainly needed for telecoms product development and management.*

The outputs of such processing are assisting decision management in respect to the creation and impact assessment of new offers and the identification of target customer segments affected by new invoicing policies; they are providing the means to analyze competition and relevant threats, reduce customer churn, control ARPU, control costs, and increase competitiveness. Limitations and concerns in such use cases are due to the increasing volume of usage data and the number of alternative scenarios that need to be studied; *number of scenarios, usage data and number of customers are the main multipliers of stakeholder's data size.*

The case becomes more demanding when application results need to be provided interactively to the customer. *Interactivity is necessary for "what-if" scenarios per customer.* The need for better performing data solutions becomes evident, and therefore the use case becomes challenging for data store technologies to achieve cost-effective scalability and elasticity of applications.

Customers and subscriber numbers are not unrelated to each other; they are nodes of usage-based graph representations (a type of social network) that indicate key customers (leaders, opinion influencers or weak, under churn risk). *Analytics are also useful and valuable to consider social network criteria, as created by call usage analysis, and customer groups defined by such 'social' relations (as represented within graph representations).*

The main, indicative, functionalities provided by the use case are:

Interactive Tariff Simulation - Per customer / group of / subnetwork of contracts

- Find the best scenario for a customer or for a specific number
- Find the best offers (scenarios) that achieve a specific target
- Compare charges produced by a new scenario with current charges
- Find the effect of changing a specific scenario or changing the scenarios of a specified set/network of contracts

Aggregate – Customer Segmentation

- Find customers with the biggest delta (distance) from their best scenario;
- Find distance of each customer from his best scenario or from a specific scenario
- Find average distance of the invoices from the corresponding best scenario per scenario;

Social Network - based

- Find customers which are key (leaders or very weak) according to their social network characteristics (as derived from their calls and the related usage); find the most appropriate scenario to propose to them
- Find through the social network interpretation, numbers which are non-subscribers; calculate impact on revenue in case the calls to these numbers become on-net (i.e. they switch their network to the operator's network)

3.2 CDR Analytics: Architecture Level

The overview architecture (Figure 3) of the CDR analytics use case, contains

- **A powerful rating / billing engine** that reads usage data, users data and pricing rules (contracts and scenarios) and produces invoice lines, which are the main elements of end customer (subscriber) invoices. When the customer data are "real" and rating is run by the telecom billing operations unit, the produced invoices are those to send to the customer; but when the data and mainly the pricing schemes are hypothetical, invoices produced are simulated and show what could have been charged if pricing rules were different. Pricing rules are called scenarios and are part of the system's input along with the customer data, the contracts, the agreements and the usage data (CDRs).
- The **analysis application engine** that processes the simulated invoices and makes the necessary comparisons, performs analytical queries, re-invokes rating process to re-produce invoices, takes variable user input (scenario parameters, different pricing targets, customer selection parameters, etc.) and visualizes results.

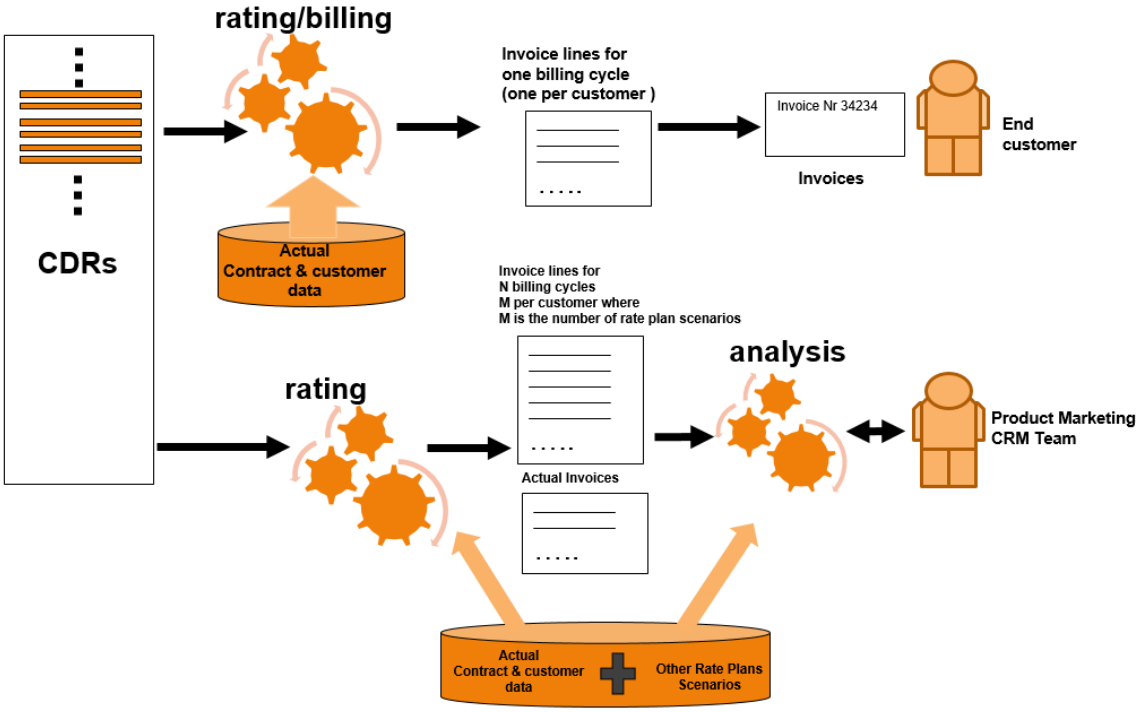

Figure 3: Tariff Simulation Overview

Involved data include:

- **CDRs:** the call detail records (network usage data)
- **Contract and Customer data:** the customer details, their contracts, their numbers and the associated scenarios agreed for charging them,
- **Actual Invoice Lines:** the current charges of contracts for given billing cycles

- **Other Rate Plan/Scenarios:** these are practically all considered scenarios (including hypothetical ones) combining base rateplans, addons, discounts and promos; definition of new scenarios might be interactively made 'on-the-fly' through successively applying modifications
- **Invoice lines:** these are the simulated invoice lines, those produced by the rating of CDR data according to all Other Rate Plans and Scenarios.

The same picture (Figure 4 below), from a different perspective, shows all datasets which are necessary within different contexts of the CDR analytics use case.

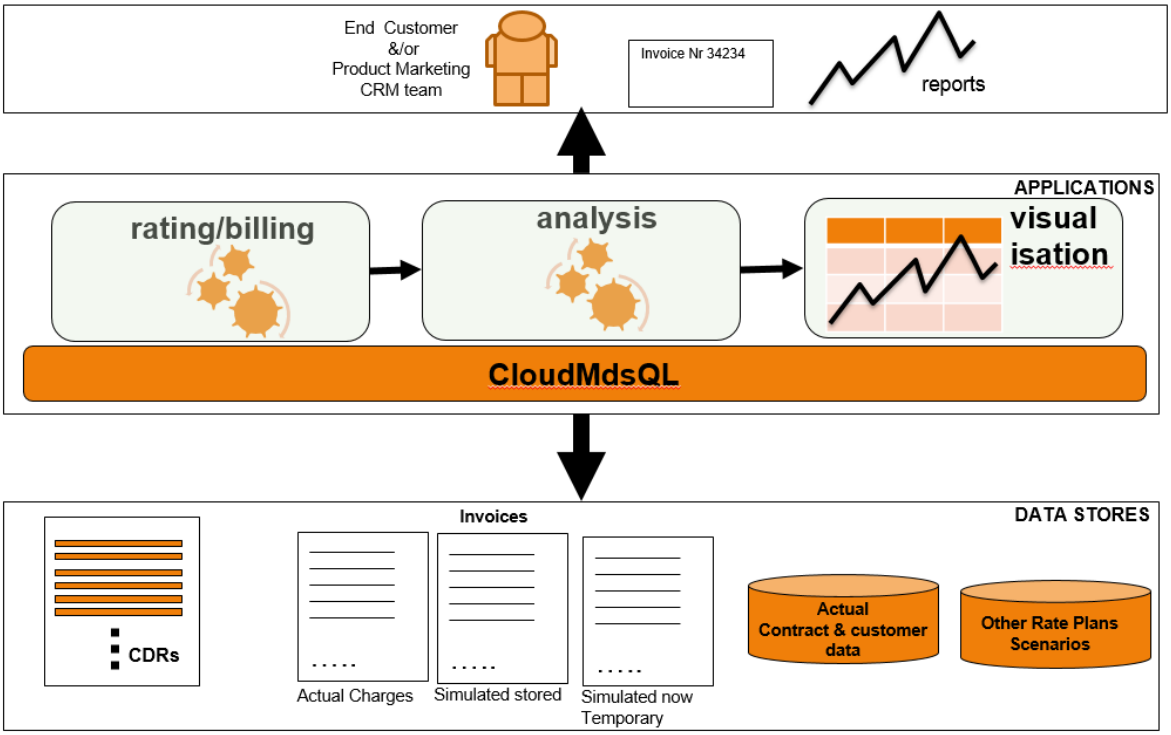

Figure 4: CDR Analytics - Architecture

These sets will be hosted on different types of data stores so that maximum performance can be obtained by considering data access pattern particularities and abilities of different store. They will be accessed by the rating and analysis applications through a single query layer and transaction interface, the CloudMdsQL layer, to simplify the code of integrating result sets of different datastores.

3.3 CDR Analytics: Design Level

3.3.1 Requirements Context in numbers

As described in D.9.1 "Use Case Requirements", the currently developed price simulation and margin analysis applications by Neurocom use relational data stores (Oracle and PostgreSQL) as main data storages. The numbers below, show data use and currently faced times of execution within batch executions, as not any interactive use is attempted due to unacceptable performance of responses:

Current data use and measured performance	
CDRs per month	~410 M
Rating execution time	30 minutes average
Execution time for a tariff simulation cycle	2 hours/billing cycle
Time to load data into margin analysis application	~12 hours
Time for tariff simulation on 3 months data	~24 hours
Execution time of margin analysis in total	~36 hours
Number of scenarios	300
Number of records	5.5 billion invoice lines
Database size for margin analysis	500 GB

Under realistic conditions, the margin analysis functions are not scalable. For instance, the following sets of figures are the elements of a use case requested:

- Number of contract numbers: 500,000;
- Period of time: 3 months;
- Avg. number of invoice lines per invoice: 10;
- Number of combinations: 5000;
- **→ 25 billion invoice records need to be processed.**

For this scenario, it is necessary to make decisions on a **25-billion line table** that exceeds size of 1.2 Terabytes. Current solution is not scalable and cannot be scalable, with relational data stores, especially if results should be available almost real-time through a web interface; for instance the extraction of one customer CDR data (out of a 500 Gigabyte CDR store), which is the typical case for an interactive scenario, cannot be executed into affordable times. The interactive use scenarios demand the following performance:

- The load of one contract's CDRs within 5 to 15 seconds
- The load of 1000 contracts' CDRs within 60 to 120 seconds
- The retrieval of one contract's best scenario within 30 seconds
- The retrieval of 1000 contracts' best scenario within 2.5 to 5 minutes

Consequently, it is promising to migrate from the current architecture, to one that combines more data stores; actually, to in-memory load the usage data and to use data stores with performance excellence on aggregation queries. Hence, CoherentPaaS became an interesting proposal to assess.

Moreover, scalability is desired; sizes of telecom operators and their customer segments vary from case to case. Any solution should and be able to retain performance across different load demands. Finally, in certain cases, the operators want to add the ability to involve more complex queries, where contracts are selected based upon criteria on their relations in a social network produced from the usage data. The need for graph database becomes apparent.

Mixing different data stores to provide holistic transactions across all stores, while at the same time being able to scale, is interestingly challenging for the proposed CoherentPaas infrastructure.

3.3.2 Data Model

In the CDR analytics use case, the main concepts involved are the **invoice lines** (e.g. charges), which actually carry the calculated charges for **contracts** within a **billing cycle**, when **scenarios** (rateplans, addons, discounts and promos) are applied to the **network usage records** (CDRs) that belong to these billing cycles. A contract is owned by a **customer**, owns several **numbers** (MSISDNs) and has its billing cycle defined by the date of each month that a new billing cycles starts over. A number is linked to another number, when the CDRs indicate call usage among these numbers. A scenario is applied on the network usage records of the numbers that belong to the segment and produce alternative invoice lines/invoices. Each invoice line belongs to one scenario and each scenario has multiple invoice lines for the same number and billing cycle.

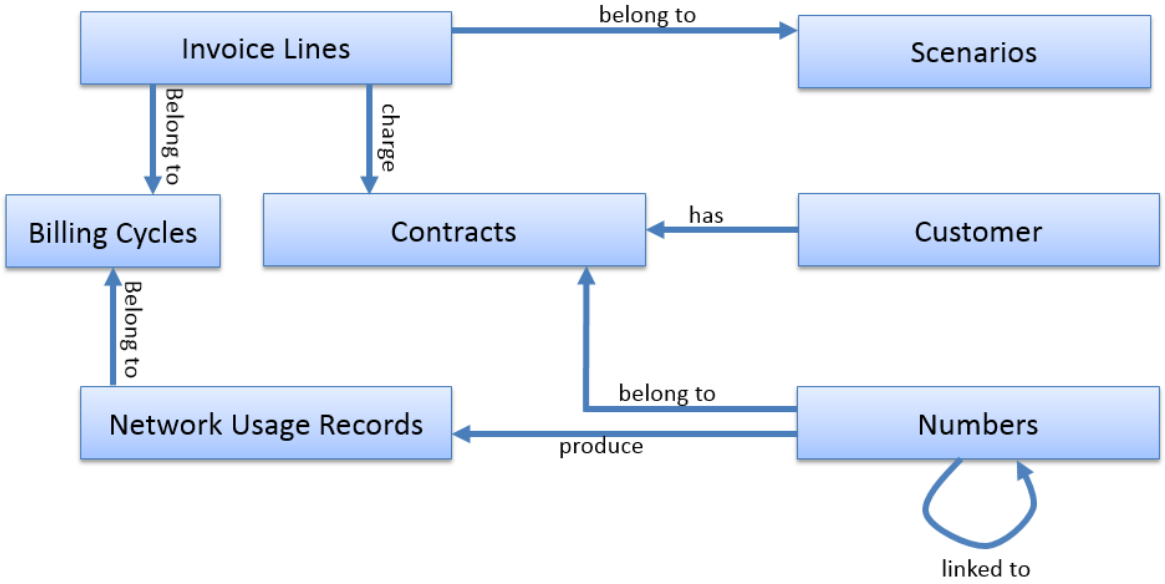

Figure 5: Main Concepts in CDR analytics use case

To persist the data related to these concepts we suggest the use of different datastores to reduce the number of I/O operations and maximize performance. They are listed below, ranked according to their importance in the use case:

- **Columnar**: To persist invoice lines where aggregated data and analytics need to be efficiently produced. MonetBD will be used.
  - Invoice Lines : (id, ContractId, ScenarioId, billing cycle, charge, charge type)
- **In-Memory**: To persist usage data and invoice lines (those interactively produced) into AcvtivePivot when involved into interactive comparison scenarios related to one customer:
  - Network Usage Records (id, date, channel)
  - Invoice Lines : (id, ContractId, ScenarioId, billing cycle, charge, charge type)
- **Graph data store**: To represent the social network of numbers (MSISDNS) based on the statistical processing of usage :
  - A calls B (times, duration, billing cycles, frequency, type)
- **Key-value** (or HDFS) : To persist raw data
  - Network Usage Records

- **RDBMS**: To persist basic data and configuration metadata required to return as part of the projection of a given query.
  - Customer (CustomerId, name, channel)
  - Contract (ContractId, CustomerId, UserId, …)
  - Numbers (MSISDN).

3.3.3 Functional Components

The main functionalities provided by this use case are:

- What is the best scenario for one number/one customer/one group of numbers?
- Which scenario brings charges of one number/one customer/one group of numbers closest to a specific target?
- How revenues will be affected if a scenario is replaced with another scenario? What if such replacement takes place for a specific set of contracts whatever scenario they are having now?
- How a customer invoice is affected when I change his scenario or which scenario is achieving a specific target for his/her invoice charges?
- What is the distance of each customer invoice from its best scenario?
- What is the distance of each customer invoice from a specific scenario? (or what is the distance of a specific scenario from each current invoice?)
- What is the average distance of the invoices produced by a given scenario from the best scenario case?
- Which customers should be approached and which offer is most appropriate to propose them in order to effectively promote my business goals?
- How revenues will be affected if a set of numbers which are non-subscribers switch to our network?

All functionalities use different datastores simultaneously. Having CloudMdsQL will simplify the code of integrating result sets of different datastores. The following subparagraphs, group functionalities according to their main characteristics in terms of needs, describe these functionalities, the representative queries as required and the involved datastores.

3.3.3.1 Basic Tariff Simulation Functionality

Basic functions of tariff simulation involve the following functions, as in the Table below:

Table 2: Basic Tariff Simulation Functions

	For One Number	**For One Customer (list of Numbers)**	**For a Network of numbers** 
Load Usage Data	For a specific number of billing cycles		
Load Invoice Lines	For a specific number of billing cycles		
Find Best Scenario			

	For One Number	For One Customer (list of Numbers)	For a Network of numbers 
• Distance from current charge • Distance from each scenario charge	-Iterate for each number	-Iterate for each customer	
Find Target Scenario Target defined as: • function of current charge • function of best charge	-Iterate for each number -Use variable input target	-Iterate for each customer -Use variable input target	-Use variable input target

Functions are applied on a per number basis, or on a group of number identified

- as a list of numbers,
- as numbers belonging to the same customer
- as linked to a single number according to specific social network criteria

Indicative queries which will be used to realize such functions are described below.

3.3.3.1.1 Best Scenario for one number

Definition: The scenario that brings the best (smallest) charge to a specific subscriber number (contract) for a specific usage, as defined by the network usage within certain billing cycles. A more detailed definition is given below:

GIVEN:

- **Contract Number:** CNTRid
- **Number of Billing Cycles:** N or cycles in period [DDMMYY – DDMMYY] or given billing cycles B_1, B_2,….B_N

FIND: best scenario

- *Retrieve InvoiceLines where Contract is CNTRid and Billing Cycle in [B1, … BN]*
- *Calculate Aggregate Prices by Scenario*
- *Get the Scenario of the Minimum Aggregate Price*

A CNTRid is associated to one number (MSISDN), though one MSISDN might have been engaged into many contracts. Therefore, we make our calculations based on Contracts and not numbers.

3.3.3.1.2 Best Scenarios for a Customer / a Group of numbers / a network of numbers

Definition: The scenario that brings the best (smallest) charge to a specific customer for a specific usage as defined by the network usage within certain billing cycles. A more detailed definition is given below:

GIVEN:

- **Customer:** Cid

- **Number of Billing Cycles:** N or cycles in period [DDMMYY – DDMMYY] or given billing cycles B_1, B_2,….B_N

FIND: best scenario per customer contract (CNTRid of Cid) achieving the best customer bill

- *Retrieve InvoiceLines where CNTRid belongs to Cid and Billing Cycle in [B_1,… B_N]*
- *Calculate Aggregate Prices by (Scenario, CNTRid)*
- *Get the Scenarios of the Minimum Aggregate Prices by CNTRid*

Similar to the case of a customer, is the case of a set of numbers; such set can be defined either as a list of numbers or can be extracted from the social network of number based on certain criteria. The query definition is then modified as it follows:

Definition: The scenario that brings the best (smallest) charge to a specific set of numbers (contracts) for a specific usage, as defined by the network usage within certain billing cycles. In this case, the following query statements are the ones who filter the Invoice lines that are necessary to be processed:

To get the invoice lines for a given list of contracts

- *Retrieve InvoiceLines where CNTRid in [$CNTR_1$, … $CNTR_m$] and Billing Cycle in [B_1, … B_N]*

To get the invoice lines of a specific social network centered round a given number

- *Retrieve InvoiceLines where CNTRid in*  *)and Billing Cycle in [B_1, … B_N]*

The Figure 6 below shows the query execution diagrams (the symbols for operators and colors for Datastore types are explained in Section 2) for the best scenario finding for a list of contracts as selected and for a social network subgraph centered around a given number. A multi-criteria filtering of invoice lines is accomplished based on the contracts selected.

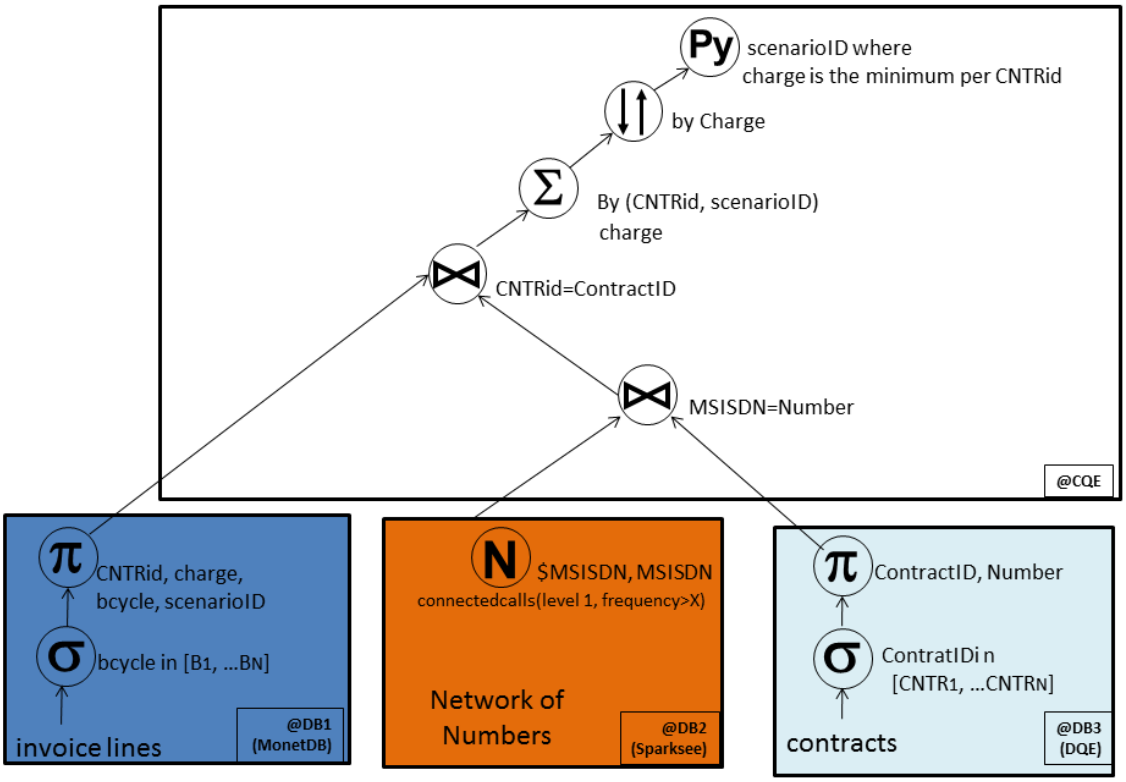

Figure 6: Multi-criteria selection of contracts

3.3.3.1.3 Target scenario for a number

Definition: The scenario that brings the charge of a specific contract closest to a given target (normally expressed as a percentage of reduction of the current charges), for a specific usage, as defined by the network usage within certain billing cycles. A more detailed definition is given below:

GIVEN:

- **Contract Number**: CNTRid
- **Number of Billing Cycles:** N or cycles in period [DDMMYY – DDMMYY] or given billing cycles B_1, B_2,….B_N
- **Current Bill:** CurrentInvoiceCharge[2]

FIND: the scenario that achieve charges that are lower than and closest to (x%) of current charge - and not lower than (y%) of current charge

- *Retrieve InvoiceLines where Contract is CNTRid and Billing Cycle in [B_1, … B_N]*
- *Calculate Aggregate Prices by Scenario*
- *Get the Scenarios (SC_i.,…SC_n) where the Aggregate Price is less than (x%) of CurrentInvoiceCharge sorted based on the Aggregate Price P(CNTRid, SC_i)*

[2] it could be calculated based on given CNTRid and the current Scenario

- *Get the One Scenario from (SC_i.,…SC_n) with the Maximum Aggregate Price P(CNTRid, SC_i) only if this price is Larger than (y%) of CurrentInvoiceCharge*

3.3.3.1.4 Target scenario for a customer / a Group of numbers / a network of numbers

Definition: The scenario that brings the charge of each contract of a specific customer closest to a given target (normally expressed as a percentage of reduction of the current charges of the contract) for a specific usage, as defined by the network usage within certain billing cycles. A more detailed definition is given below:

GIVEN:

- **Customer:** Cid
- **Number of Billing Cycles:** N or cycles in period [DDMMYY – DDMMYY] or given billing cycles B_1, B_2,….B_N
- **Current Bill:** CurrentInvoiceCharge (or it could be calculated based on given CNTRid and the current Scenario

FIND: the scenarios that achieve charges for each customer contract to be lower than and closest to (x%) of current charges - and not lower than (y%) of current charge

FOR EACH CNTRid *that belongs to Customer Cid:*

- *Retrieve InvoiceLines where Contract is CNTRid and Billing Cycle in [B_1, … B_N]*
- *Calculate Aggregate Prices by Scenario*
- *Get the Scenarios (SCi.,…SCn) where the Aggregate Price is less than (x%) of CurrentInvoiceCharge sorted based on the Aggregate Price P(CNTRid, SCi)*
- *Return the One Scenario from (SCi.,…SCn) with the Maximum Aggregate Price P(CNTRid, SCi) only if this price is Larger than (y%) of CurrentInvoiceCharge*

Similar to the case for one customer, is the case of a set of numbers; such set can be defined either as a list of numbers or can be extracted from the social network of a number defined with certain criteria. The query definition is then modified appropriately, as it was in the case of the best scenario finding that was previously described, to find the target scenarios:

FOR EACH CNTRid *in [$CNTR_1$, … $CNTR_m$]* and

FOR EACH CNTRid *in* 

3.3.3.2 Interactive Customer analysis

Several functions require the direct extraction of results because of the fact that they are caused by interaction with the customer, needing answers as quickly as possible, as well as, justified and well supported from the operator's side.

The following Figure 7 illustrates such cases which are mainly the following:

- Searching the target scenario achieving a variable target charge; the user is continuously modifying the target amount, viewing the scenarios which bring the target for a specific customer selection. The actual charges and the simulated ones are utilized.

- Searching the best scenario and calculation of the brought revenue, among different input scenarios. Usually, the user is having a set of constant options within its scenario (pivot parameters) and continuously applying changes to other parameters, while observing how revenues are affected and which is finally the most appropriate option
- Providing a new scenario as an input and getting interactively the charges caused,
- Providing a new scenario as an input and getting a new view on the produced revenues, as well as in comparison with simulated revenues

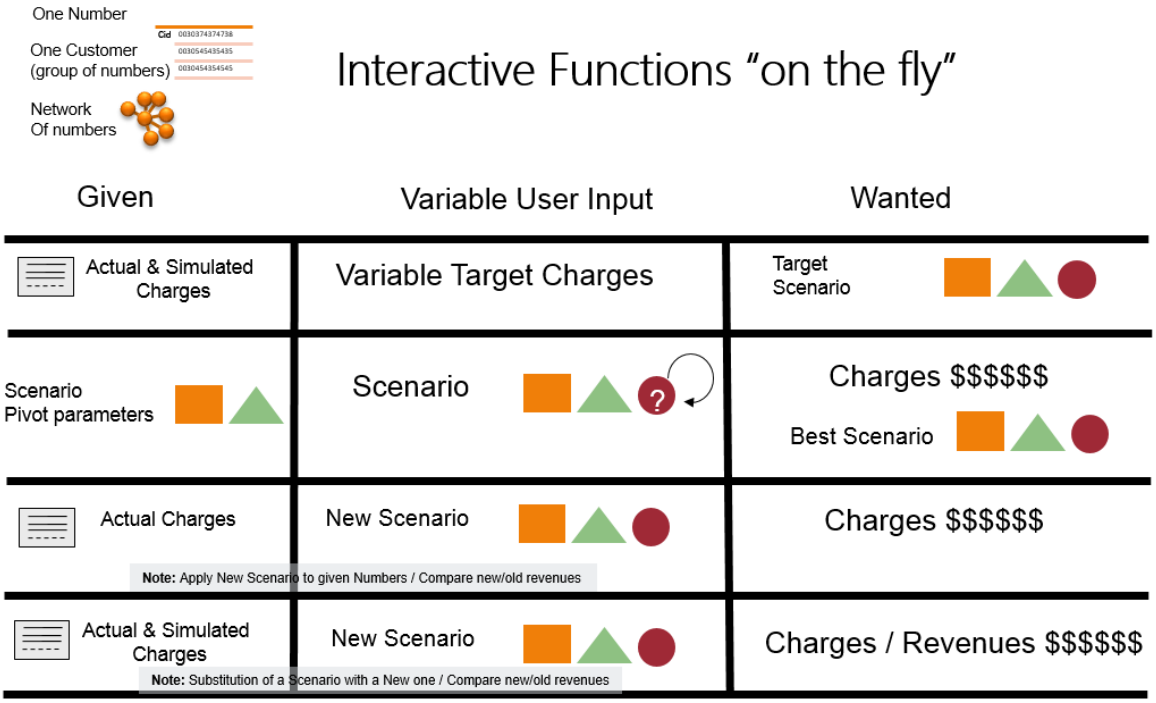

Figure 7: On-the-fly interactive functions

As shown in Figure 8, the critical functions in the process are mostly related to the efficient and quick

- Loading of usage data (for one number or one customer)
- Rating of selected CDR in order to produce new simulated invoices.

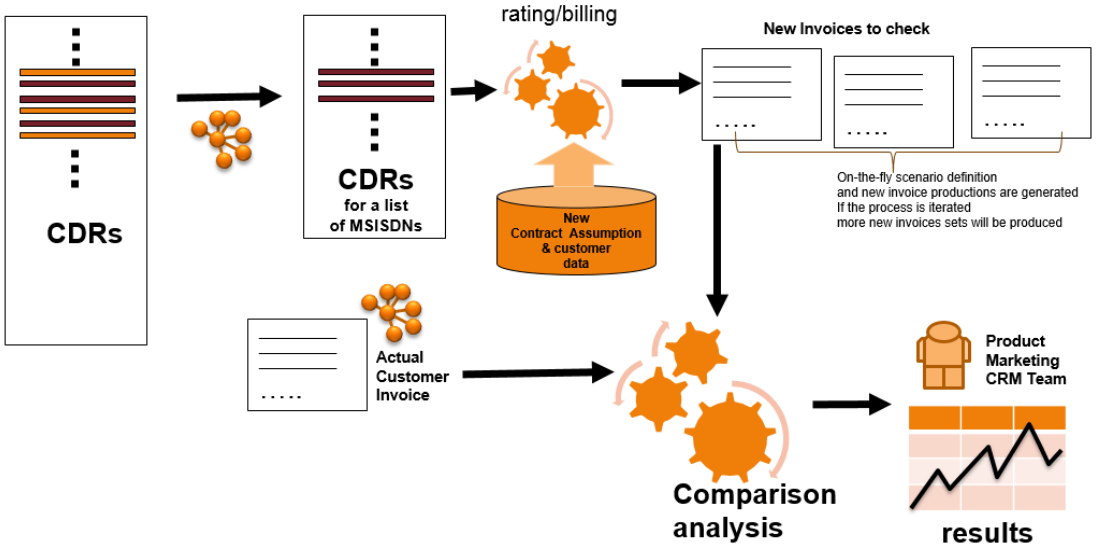

Figure 8: On-the-fly customer (what-if) analysis

The next computation steps in the interactive function, will involve quick and easy comparisons of temporal results with actual and simulated charges. Results for one number or for a list of numbers could be:

- Display all "Deltas", distances (sorted) of invoice charges per scenario from the new invoice
  - Display all scenarios which are affected by the new scenario.
  - Highlight "adjacent" scenarios (scenarios that produce invoice charges adjacent to the new charges)
- Calculate distance from the current actual invoice

Figure 9 below, shows an interactive query that select and loads CDRs of a certain user and his/her network according to certain criteria in his social network graph; it filters only those whose bills are higher (for a certain amount) than their simulated bills for focused (one or more) scenarios. This can be necessary when the application users checks for a new scenario (or a set of new scenarios), how it stands in comparison to current charges. Actually, the query will finally load the CDRs, in order to rerate them with new scenarios which will again be compared.

All this process might be necessary to be performed on the fly. Indicative times have been prescribed in D.9.1 "Use Case Requirements": extraction of a customer usage data (in order to re-rate them again) from a bunch of 5 billion CDRs should not take more than a few seconds (less than 10 seconds), while similar extractions out of simulated invoice is considered affordable only when it remains within a range of 1-2 minutes.

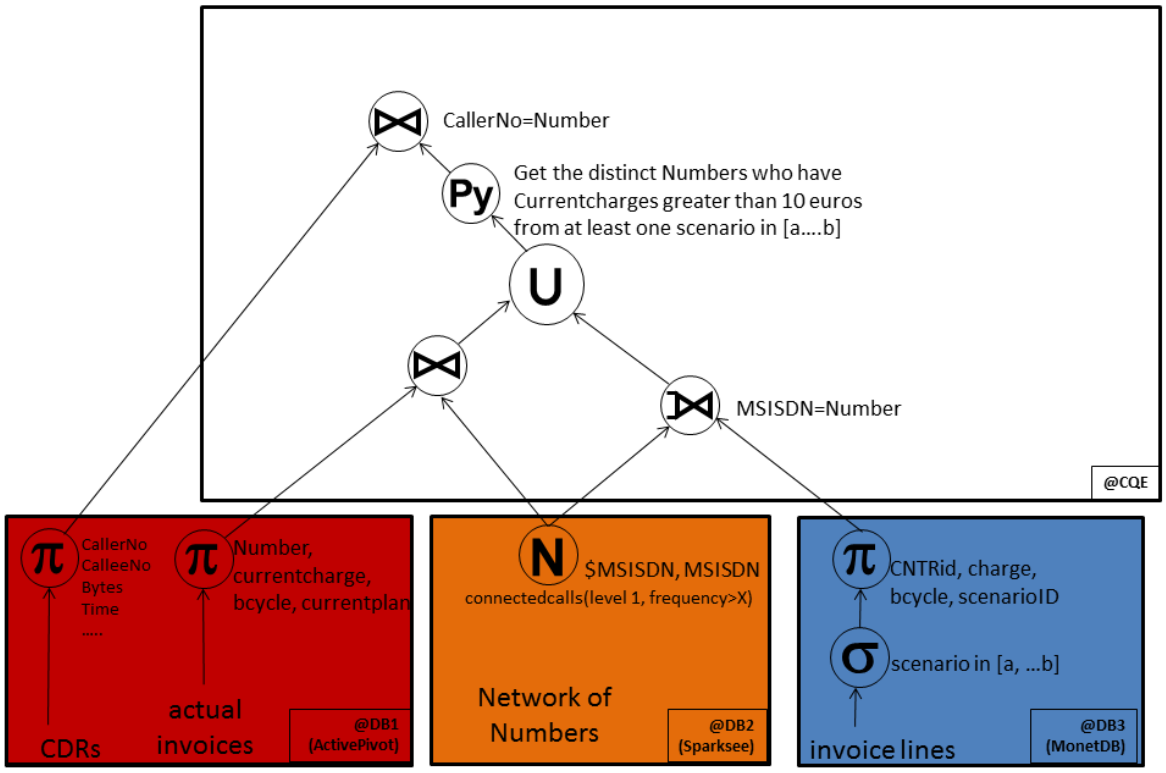

Figure 9: Load CDRs of subscribers and their strongly connected numbers which are currently charged higher than simulated charges of given scenarios

3.3.3.3 What-if functions on revenue impact

Another set of functions would seek to assess the impact on revenue of changes in the applied scenarios for selected contracts (i.e. what could happen when I decide to modify a given scenario) or how a given scenario (i.e. a competitor's one) affects the revenue whenif applied to a certain customer segment. Results of such what-if analysis function could be:

- Calculation of the change in revenue
- Identification of the customer segment which is in risk (where the new charges are larger than the actual charges)
- Calculation of the change in revenue when I change scenario only for the customer segment which is in risk

Certain examples of what-if cases and relevant queries are provided below.

3.3.3.3.1 Modify a Scenario – How it affects my revenue?

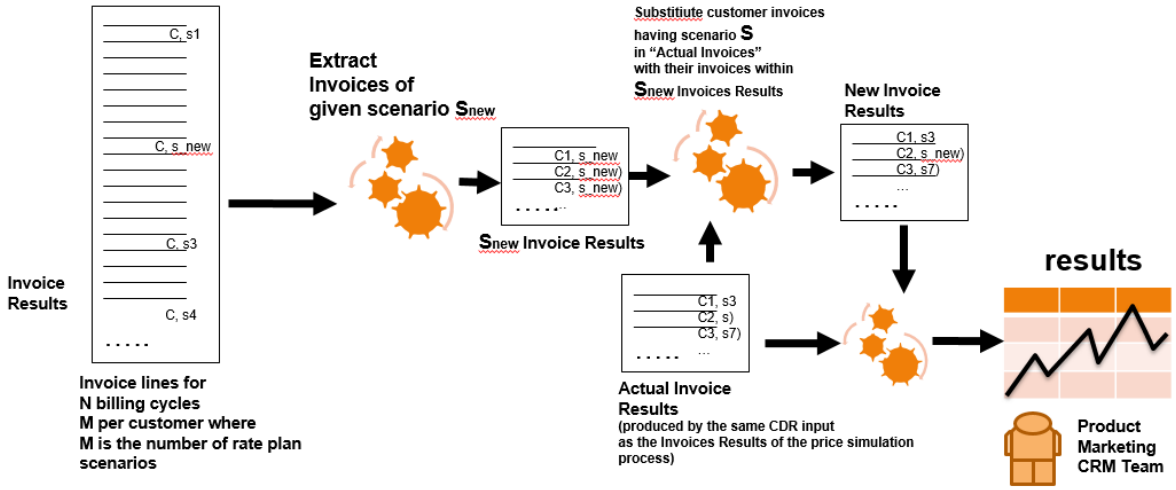

Figure 10: Substitute Scenario S with Scenario S_new

Figure 10 above shows the way that a functional process is analyzing what could happen to actual revenue when a given scenario (S) is changed, and is substituted by another, new scenario (S_new). Actually the "new invoices" contain the old (actual) ones for the contracts where the applicable scenario is not the given one (S) and the invoices (found in the simulated invoices) with the new scenario (S_new) for the contracts which until now are applied to the given scenario (S).

A representative query is described below.

Definition: The change in revenue when a scenario is changed and is replaced by another, new, scenario. The query will attempt to calculate the new revenue; for this, it will sum the actual invoices where the applied scenario is not the new scenario, with the sum of the simulated invoices with the new scenario, only for those contracts which have the old scenario as actual. A more detailed description of the queries is given below:

GIVEN:

- **Scenarios:** S and S_{new}
- **Current Invoices**

FIND: the effect in revenue

- *SUM(charges) in CurrentInvoices where scenario is not S*
- *Select Contracts in CurrentInvoices where scenario is S as S_Contracts*
- *SUM(Charges) in InvoiceLines where Scenario is S_NEW and WHERE Contract is in S_Contracts*
- *Add the two sums*

3.3.3.3.2 Scenario Substitution in a Group of Customers– How it affects my revenue?

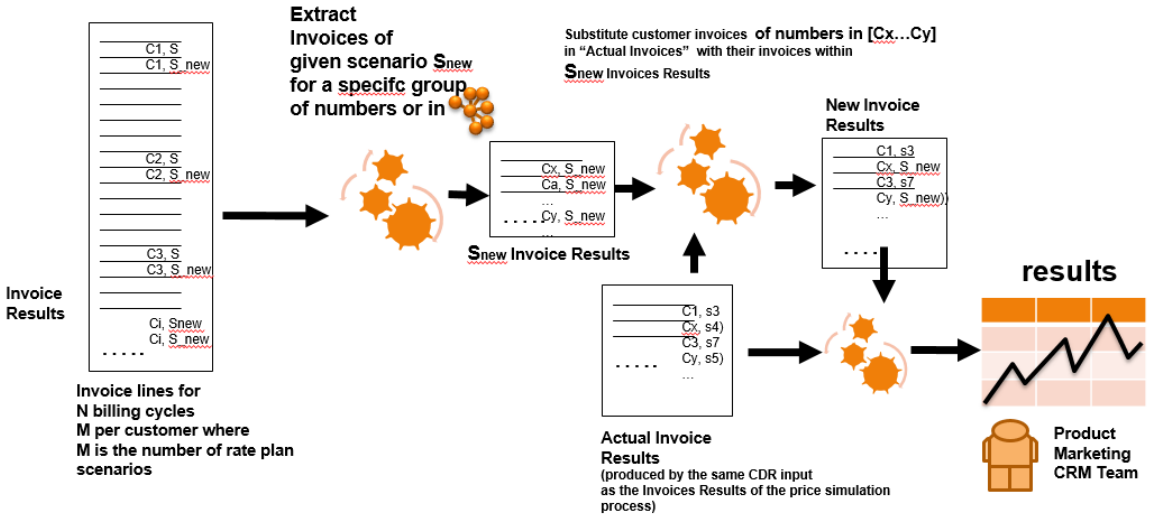

Figure 11: Substitute Scenarios of Numbers with Scenario Snew

Figure 11 above shows the way that a functional process is analyzing what could happen to actual revenue when a given scenario (S_new) is applied to a selection of contracts (defined in various ways, even as a network of contracts with numbers linked to each other according to certain social network criteria). Actually the "new invoices" contain the old (actual) ones for the contracts which are not selected and the invoices (found in the simulated invoices) with the new scenario (S_new) for the selected contracts.

A representative query is described below.

Definition: The change in revenue when a new scenario is applied to a selected group of contracts. The query will attempt to calculate the new revenue; for this, it will sum the actual invoices of contracts which are not selected, with the sum of the simulated invoices with the new scenario, only for those contracts which match the selection criteria. A more detailed description of the query is given below:

GIVEN:

- **Scenarios:** S_{new}
- **Current Invoices**
- **Group of Contracts:** $[CNTR_1 \ldots CNTR_m]$ (m Numbers) (or in)

FIND: the effect in revenue

- *SUM(charges) in CurrentInvoices where Contracts not in* $[CNTR_1, \ldots CNTR_m]$ *(or not in)*
- *Select Contracts in CurrentInvoices where scenario is S as S_Contracts*

- *SUM(Charges) in InvoiceLines where Contracts in [CNTR$_1$,….CNTR$_m$] (or in)and Scenario is S_New*
- *Add the two sums*

3.3.3.4 Aggregate Functions

These are the functions that perform analytical queries on the simulated invoices, mainly in order to provide aggregate statistics or even in order to perform segmentation of the customer base upon certain criteria (as illustrated in Figure 12)

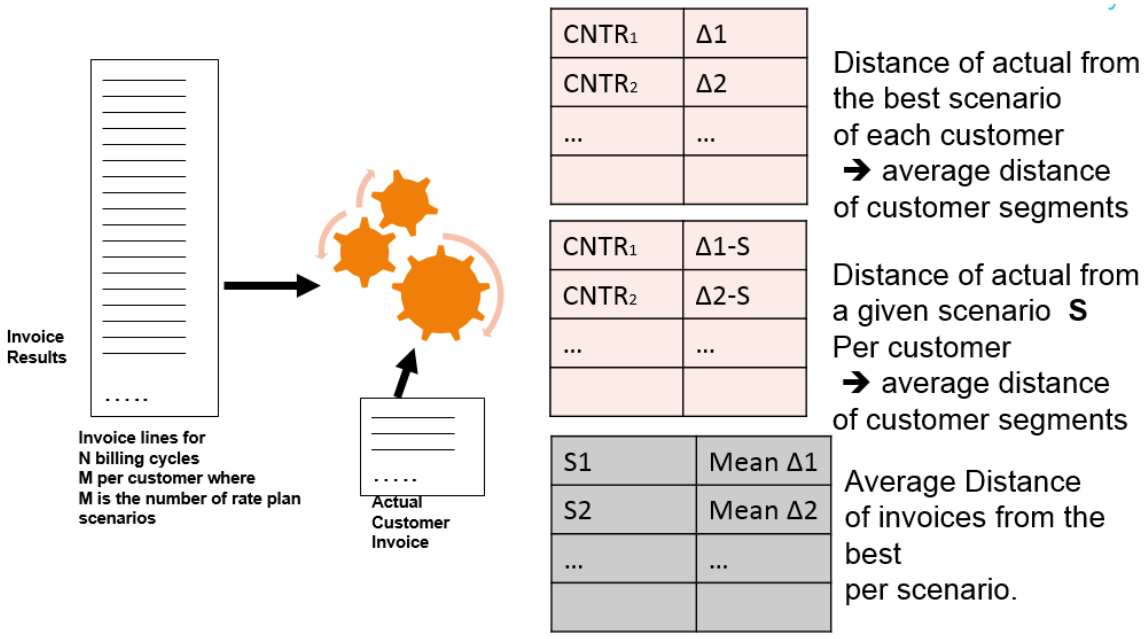

Figure 12: Aggregated Results

Functions may sometimes apply comparison with actual invoice results or certain simulated results. Further post processing can be applied in order to justify customer segmentation scenarios:

- Based on comparing delta-values with current charges
- Based on absolute delta-value range
- Based on absolute charges

For instance, aggregated results could provide categorization of customers into churn probabilities, putting them in lists:

- Where the distance of their actual charge from the best charge is less than 5% of the actual charge (low risk to loose)
- Where the distance of their actual charge from the best charge is more than 5% of the actual charge and less than 20% (medium risk to loose)
- Where the distance of their actual charge from the best charge is more than 20% of the actual charge (high risk to loose)

The indicative queries of the Figure 12 are also described below.

3.3.3.4.1 Customer distance from best scenario

Definition: The distances of all contracts (and all customers) from the invoice produced by the best scenario. The current charges are compared to the produced charges from the

best scenario for each number, on the same usage. The same definition can be expressed as, the distances of the best scenario of each customer from the current charges, as a means to identify customers who are likely to be in a risk because of alternative offerings in the market. A more detailed definition is given below:

GIVEN:

- **Invoice Lines:**
- **Number of Billing Cycles:** N or cycles in period [DDMMYY – DDMMYY] or given billing cycles B_1, B_2,….B_N
- **Current Invoices**

FIND: the distance of each Contract from its Best Scenario

FOR EACH CNTRid:

- *Retrieve InvoiceLines where Contract is CNTRid and Billing Cycle in [B_1, … B_N]*
- *Calculate Aggregate Prices by Scenario*
- *Get the Scenario Sid-best of the Minimum Aggregate Price*
- *BestCharge → P(CNTRid, Sid-best)*
- *Δ → CurrentCharge – BestCharge*
- *Return(CNTRid, Sid-best, Sid-current, BestCharge, CurrentCharge, Δ)*

3.3.3.4.2 Customer distance from a given scenario

Definition: The distances of all contracts (and all customers) from the invoice produced by a specific scenario. The current charges are compared to the produced charges from a specific scenario for each number, on the same usage. The same definition can be expressed as, the distances of a new scenario from the current charges, as a means to identify customers who are likely to be in a risk because of this new offer. A more detailed definition is given below:

GIVEN:

- **Invoice Lines:**
- **Number of Billing Cycles:** N or cycles in period [DDMMYY – DDMMYY] or given billing cycles B_1, B_2,….B_N
- **Current Invoices**
- **Given Scenario** S

FIND: the distance of each Contract from Scenario S

FOR EACH CNTRid:

- *Retrieve InvoiceLines where Contract is CNTRid and Billing Cycle in [B_1, … B_N]*
- *S_Charge ← P(CNTRid, S) i.e. Calculate Aggregate Price where Scenario is S*
- *Δ ← CurrentCharge - S_Charge*
- *Return(CNTRid, Sid-best, Sid-current, S_Charge, CurrentCharge, Δ)*

3.3.3.4.3 Invoices distance from best per scenario

Definition: The average distance of each scenario from the best scenario, defined as the average distance of the invoices produced by the scenario for all contracts, from the best

invoice (as produced by the best scenario) of each contract. This is to identify the equivalency of scenarios application upon specific customer segments as well as quantify the effectiveness of scenarios. A more detailed definition is given below:

GIVEN:

- **Invoice Lines:**
- **Number of Billing Cycles:** N or cycles in period [DDMMYY – DDMMYY] or given billing cycles B_1, B_2,….B_N

FIND: the average distance of invoices from their best per scenario

TABLE BEST_CHARGES *each row is (CNTRid,* S_{id_best}, $Price_{id\_best}$*) where* S_{id_best} *is the best scenario per contract and* $Price_{id\_best}$ *is the best price P (CNTRid,* S_{id_best}*)*

FOR EACH Scenario S:

- *Retrieve InvoiceLines where Scenario is Sid and Billing Cycle in [B1, … BN]*
- *Calculate Aggregate Prices by CNTRid: P(CNTRid, S)*
- *Calculate Distance of Sid from Best by CNTRid: Delta(CNTRid, S)* ← *P(CNTRid, Sid_best) - P(CNTRid, S)*
- *Mean_Delta = SUM(Delta(CNTRid, S)) / (Number of Contracts)*
- *Return(S, Mean_Delta)*

3.3.3.5 Social-network related functions

Functions, which require the identification and analysis of relations of contracts with other contracts, are mentioned as social-network related functions. In previous paragraphs, we have presented cases where the given contracts are not directly referenced (as a list of contracts or as a reference to a customer), but they should be extracted by querying the social network i.e. its representation graph.

For instance, the simplified input:

Given: CONTRACT in the Network of Contracts ()

is equivalent to a graph query:

Given: One Contract with CNTRid

Result: Get me the CONTRACTS [CNTR1 ….CNTRn] WHERE CNTRid is directly linked to CNTR1 (CNTRid calls CNTR1) according to certain criteria (i.e. Number of Calls > 4, every day including weekends, total duration per day > 5 minutes)

Moreover, other social-network related functions are described below.

3.3.3.5.1 Key Customer Finding and Proposal Adjustment

The operator needs to identify certain contracts which can be characterized as key:

- Either weak because they are mostly calling non-subscribers according to certain statistical criterion
- Strong, because they have heavy usage, calling other subscribers; or even they are characterized as influencers according to social media analysis results

When a competitor's offer appears, the operator needs to "protect" the weak part of his customer base, as well as needs to find the best offer to make to them, and an appropriate offer to make to the strong part of its customer base, in order to achieve the most effective penetration of its offer.

A relevant query is described below:

Definition: The appropriate offer to make towards key customers upon the appearance of a given competitor's scenario. The idea here is to identify those key customer according to social-network relations criteria (a graph query will be necessary here) and for those customers, find the best scenario that achieves slightly lower charges than the simulated charges for the new (the competitor's) scenario. A more detailed definition is given below:

GIVEN:

- **Number of Billing Cycles:** N or cycles in period [DDMMYY – DDMMYY] or given billing cycles B_1, B_2,….B_N

FIND: the "best" scenario offer to make to "key" contract owners against a competitive scenario S

- *FIND the KEY CONTRACTS and their Network* 
- *FIND what all CONTRACTs in*  *are paying under Scenario S→ InvoiceCharge*

FOR EACH KEY CONTRACT *(or for each CONTRACT in*  *)*

- *Find the scenarios that achieve charges for each contract to be lower than and closest to (x%)of InvoiceCharges - and not lower than (y%)of InvoiceCharges*

Figure 13 below shows a similar case of a query, where actual and simulated charges are ranked for a selected group of node numbers (those that are considered strong). This is an intermediate step to identify and sort distances of actual scenario from a set of given scenarios, i.e. a competitors' ones.

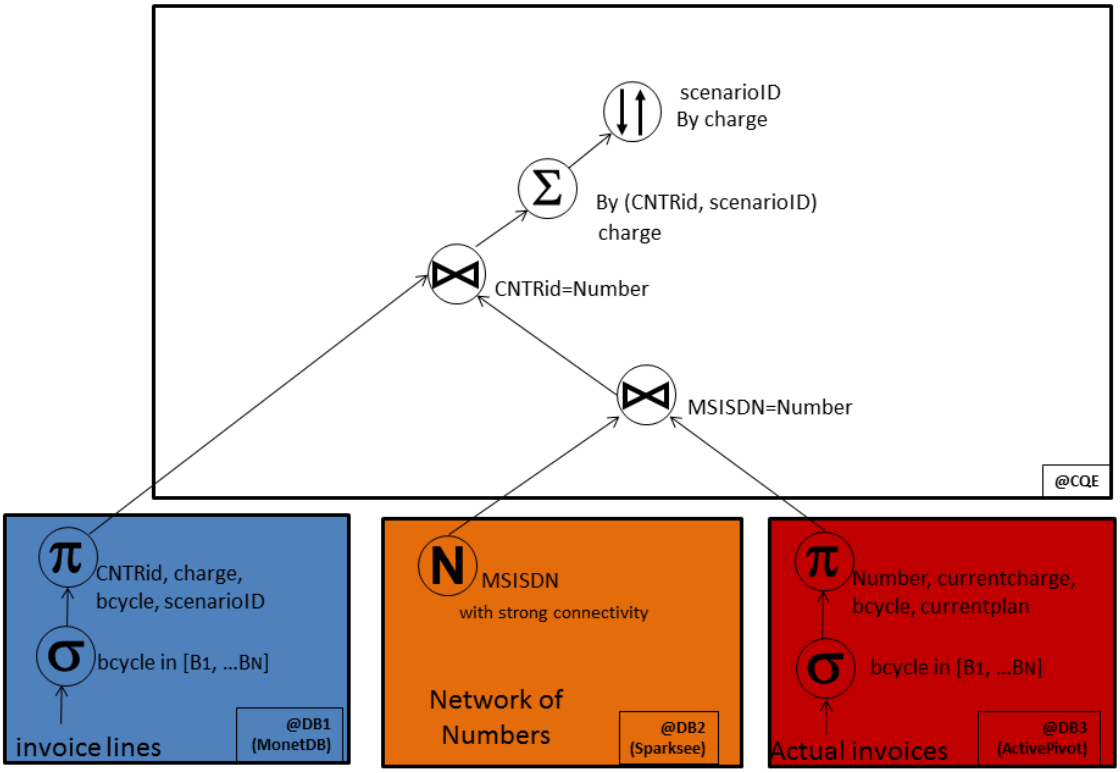

Figure 13: Sort scenarios (among given list of scenarios) including the current plan based on the produced charges for a set of "strong" numbers, as identified in the social network graph

3.3.3.5.2 Moving calls on-net

When an entity becomes a new customer, especially when that customer is a big one, it might be the case that revenue will be affected significantly because this customer (possibly with a large number of MSISDNS) was heavily called by the operator's subscribers. Such a case could be quantified and pre-calculated, before an offer is to be made to this big customer.

Figure 14 illustrated such a case: Contracts A, B, C and D are charged **X** euros. How they will be charged (in total and separately) if **non-customers** E and F switch to the same network with A, B, C, D?

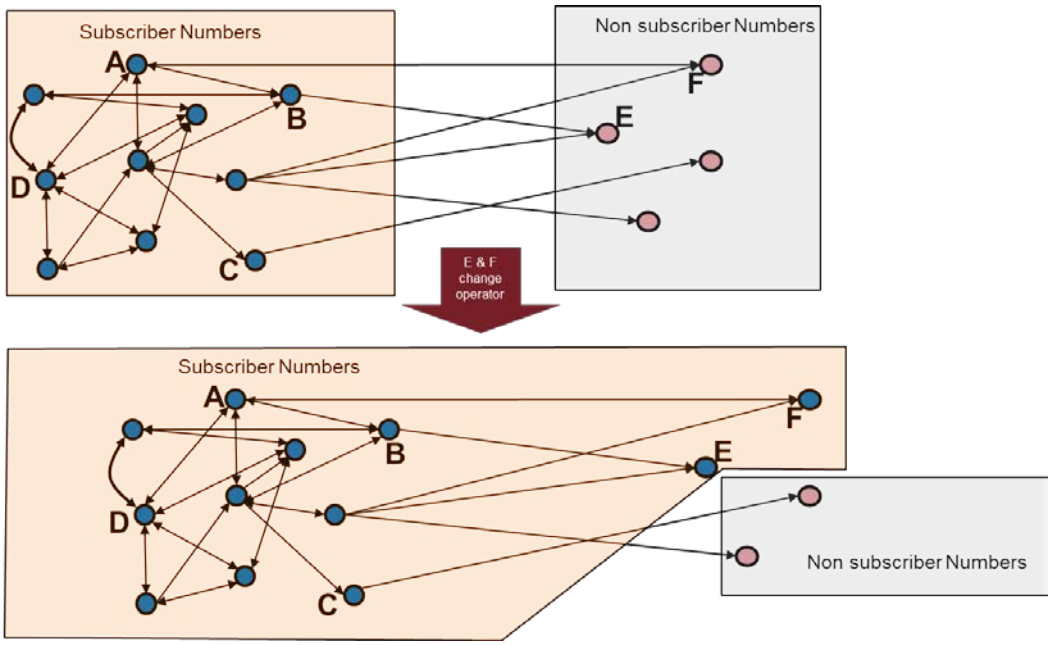

Figure 14: Moving calls on-net

The function will need to extract with graph queries the contracts who are calling the contracts E and F, and then re-rate the usage when these two, will become on-net. An indicative query is described below.

Definition: The target scenarios to offer to a new big customer. First we must identify the level of effect on revenue, when this big customer is getting on the network; the callers of him should be identified, and their usage should be recalculated, charges should be simulated and compared to old ones, so that finally a target can be set that will be used to find also the target scenario for the new customer. A more detailed definition is given below:

GIVEN:

- **Set of non-customer numbers** [a…b]
- **Number of Billing Cycles:** N or cycles in period [DDMMYY – DDMMYY] or given billing cycles B_1, B_2,….B_N

FIND: the effect when Numbers [a,..b] becoming subscribers

FIND THE CONTRACTS:

- *highly calling non customers*
- *according to the Network of Numbers ( ) and their CDRs of Billing Cycles in [B1,…BN]*

FOR EACH KEY CONTRACT *or for each CONTRACT in the Network of Numbers ( ):*

- *Re-rate under the scenario that calls to [a,…b] are on-net calls*
- *Compare with ActualCharges and change in Revenues*
- *Find target scenarios*

3.4 CDR Analytics: Expected Benefits

The most important benefits expected by our design are:

- Ability to utilize multiple stores efficiently; until now, tariff simulation development cannot attempt to involve multiple types of datastores, though it is sure that for certain cases, different stores are applicable and preferred in order to achieve the best performance. Involving different technologies will increase complexity and will require a unified framework to address all stores transparently; the need for CloudMdsQL becomes apparent
- Ability to apply interactive functions; a feature which was not possible, but with the appropriate datastore employment it can be accomplished. Efficient load of selected usage data is one need, addressed already by first testing scenarios where the in-memory solution has been used
- Ability to scale in cloud environments; increasing number of scenarios, number of billing cycles and number of customers will not be an upper bound for the system.

3.5 CDR Analytics: Common CPaaS Components

A common component that could be utilized by other use cases is the filtering (or grouping) operation based on a subgraph definition that is represented by a query (a native one) in the graph database.

The component can be expressed as follows:

```
Filter  from Datastore A columns  <A…B> where column <C> belongs to
subgraph N (C, Cpeer,relation_condition) from DataStore B
```

The selection criteria are not the value of certain columns, but the participation of values (or ids) in certain relations with certain criteria. As defined above, the filter includes those columns that are having links in the subgraph definition; Cpeer can be a constant or can be a variable node definition, depending also on the relation condition.

It is a very common need that can appear in many different contexts. For this, CPaaS could simplify as much as possible the identification of such grouping which are members of the same graph.

4 CLOUD TELECOM/M2M USE CASE – MACHINE-TO-MACHINE

4.1 M2M: Use Case Overview

Vehicle/driver monitoring and sensing is one family of applications which falls in the M2M area. Drivers, fleet owners, transport operations, insurance companies are stakeholders which need to have analytical reporting on the mobility patterns of their vehicles, as well as real-time views in order to support quick and efficient decisions towards eco-friendly moves, cost-effective maintenance of vehicles, improved navigation, safety and adaptive risk management.

Vehicle sensors do continuously provide data, while on-the-move, which are stored and processed in order to provide valuable information to stakeholders. Applications identify speed violations, abnormal driver behaviors, and/or other extraordinary vehicle machine conditions, produce statistics per driver/vehicle/fleet/trip, correlate event with map positions and route, assist navigation, monitor consumptions, and perform many other reporting and alerting functions.

The main functionalities provided by the use case are:

- *Analytics with variant types of aggregation logic*: Average and absolute values in time (speed, fuel, kms, …)
- *Trip-level analysis:* Identification of trips, the relevant trip data and production of statistics on a trip basis
- *Real-time monitoring and alerting:* Production of current vehicle position and route on the map. Production of real-time event notification upon the identification of certain value conditions or upon geo-referenced data (i.e. vehicle is near a certain Point-of Interest)

Data growth is experienced because of increase of sensed vehicles and devices. Processing becomes more complex as the number of devices and measurements per vehicle becomes larger and correlation among measurements is necessary in order to produce informative results. The use case involves both the need for real-time processing and instant response, as well as the need for extensive analysis on past data; *queries should be able to combine both real-time and past data.*

The roles involved in this use case are the drivers, vehicle and fleet owners (companies, car rentals, transport operators and logistics) and the service providers, mainly the application provider and the PaaS provider. The latter can be enabler of several different M2M applications, including the VTP use case.

4.2 M2M: Architecture Level

The architecture that is selected to implement our use – cases follows the principles of **lambda-architecture.** It works with a data-store that is immutable, loaded with event records received by the vehicles. It works in real time and calculates query results on top of an incoming stream of data. Results, once computed, are stored in such a manner that they can be queried by applications, at the analytical layer. Just as with the analytical layer, the speed layer also stores results as they are computed into a view. The following

picture summarizes the main processes involved in a Lambda Architecture. The data stream (e.g., vehicle events), which is the input of the **loading process** of the involved datastores; the precomputed analytical view which is the minimum and optimized output generated by the loading process for the queries invoked from the application; and the precomputed real-time view, which include just those simple data that is ready to be queried.

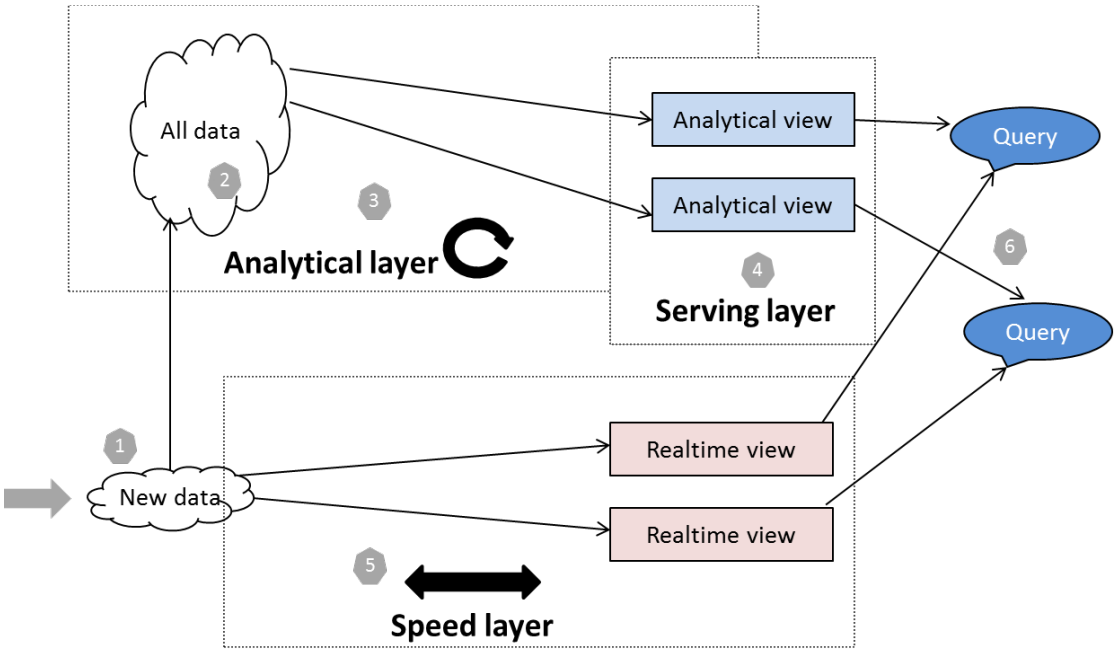

Figure 15: The Use Case Reference Architecture (Lambda Architecture)

The complete architecture is represented in Figure 15. Each enumerated component of the architecture in the picture is described below:

1. All new data is sent to both the analytical layer and the speed layer. In the analytical layer, new data is appended to the master dataset. In the speed layer, the new data is consumed to do incremental updates of the realtime views.

2. The master dataset is an immutable, append-only set of data. The master dataset only contains the rawest information that is not derived from any other information you have.

3. The analytical layer precomputes query functions from scratch. The results of the analytical layer are called "analytical views." The analytical layer runs in a while (true) loop and continuously recomputes the analytical views from scratch. The strength of the analytical layer is its ability to compute arbitrary functions on arbitrary data. This gives it the power to support any application.

4. The serving layer indexes the analytical views produced by the analytical layer and makes it possible to get particular values out of an analytical view very quickly. The serving layer is a scalable database that swaps in new analytical views as they're made available. Because of the latency of the analytical layer, the results available from the serving layer are always out of date by a few hours.

5. The speed layer compensates for the high latency of updates to the serving layer. It uses fast incremental algorithms and read/write databases to produce realtime views that are always up to date. The speed layer only deals with recent data, because any data older than that has been absorbed into the analytical layer and accounted for in the serving layer. The speed layer is significantly more complex than the analytical and

serving layers, but that complexity is compensated by the fact that the realtime views can be continuously discarded as data makes its way through the analytical and serving layers. So the potential negative impact of that complexity is greatly limited.

6. Queries are resolved by getting results from both the analytical and realtime views and merging them together.

Simplifying the overall design the main architectural components involved in our implementation are depicted in Figure 16.

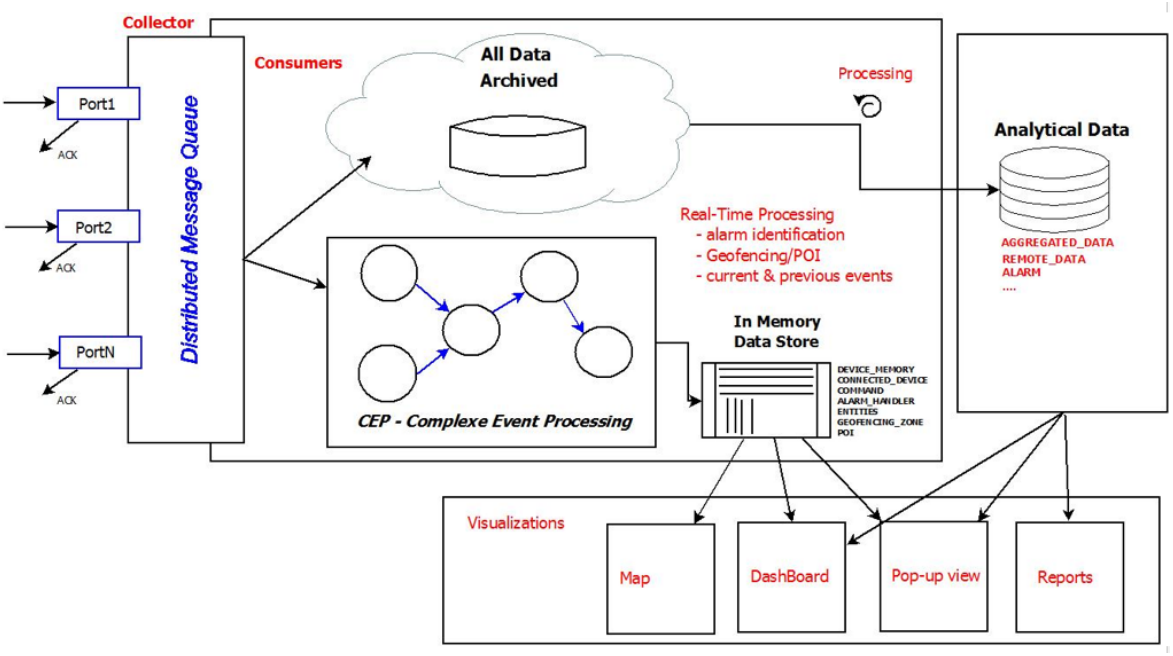

Figure 16: Architectural Components of M2M use case

As already stated we employ a layered system inspired by the Lambda architecture concepts. Streaming data are flowing through the endpoint of our architecture funneling into a distributed queuing system forming a collection mechanism which ensures that data will be forwarded to the available processing components that implement the speed and analytical layer respectively. Speed layer involves the usage of Storm a distributed computation framework used to process streaming data in real time coupled with an in memory key valued database used to store real-time views. On the other hand the analytical layer ingests data from the master data set and creates the resulting analytical views which will be then stored on the serving layer database. Accordingly the analytical views and real-time views are then consolidated to produce the different visualizations customizations needed for the various delivery systems.

4.2.1 Data Collection Architecture

Figure 17 illustrates the approach adopted to leverage the challenges of gathering data from external systems. The selected architectural approach resorts to a modular construction of a system that highlights loosely coupled components separated into several conceptual tiers. Data are collected through TCP/IP connections that gather events occurring at the edge of the system. The collection components are pre-existing

Netty[3] servers which consume incoming streams of data via network, transform and decode them, and then forward them to the next tier in our pipeline. At the next stage we employ an integration framework such as Camel[4] to seamlessly plug our data into the queuing system component designed to handle the distribution of data. Data flow and queuing semantics are handled by a Kafka[5] system that employs a Kafka producer implementation to distribute and organize data into several brokers, which are basically physical servers that constitute the Kafka cluster. Note also that we use Zookeeper[6] as coordination server to manage the data flow of the entire system.

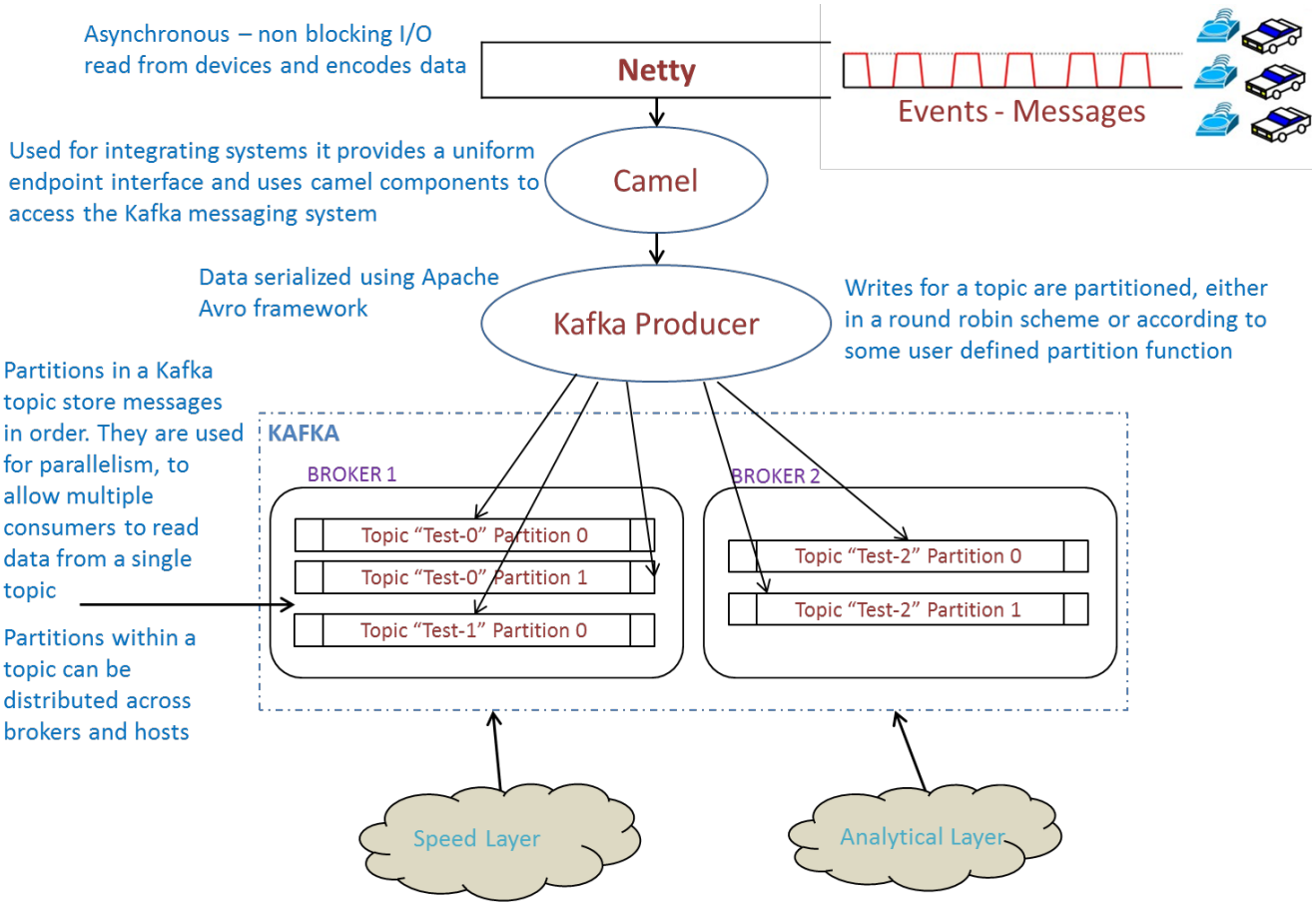

Figure 17: Gathering data from external systems

4.2.2 Storm Topology

The queuing system feeds the speed layer, where "kafka" spouts are receiving events and emitting them into the Storm topology, which is illustrated in Figure 18, and represents the full streaming computation. Spout forms the source of stream pulling the requests

[3] Netty is an asynchronous event-driven network application framework: http://netty.io/

[4] Camel empowers you to define routing and mediation rules in a variety of domain-specific languages: http://camel.apache.org/

[5] Apache Kafka a high-throughput distributed messaging system: http://kafka.apache.org/

[6] ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services: http://zookeeper.apache.org/

from the queue and making tuples. Emitted tuples are grouped by the identificator of devices (deviceId) and drivers (driverId) (Fields Grouping).

The first Bolt

- Persist events,
- Creates trips and
- Identifies Alerts.

A trip is defined by a pair of transmitted state messages that signal engine is on and engine is off.

Every event is stored in the in memory data-store both as a standalone event and as a trip event. Standalone events may be removed when periodical data have been computed and stored in the analytical layer view. In case of Trips, every trip key in the Database handles a sorted set of events that are related with it.

Alarms are published to User Interface as soon as they are identified. They are also stored in the sorted set of their trip, available for future retrieval.

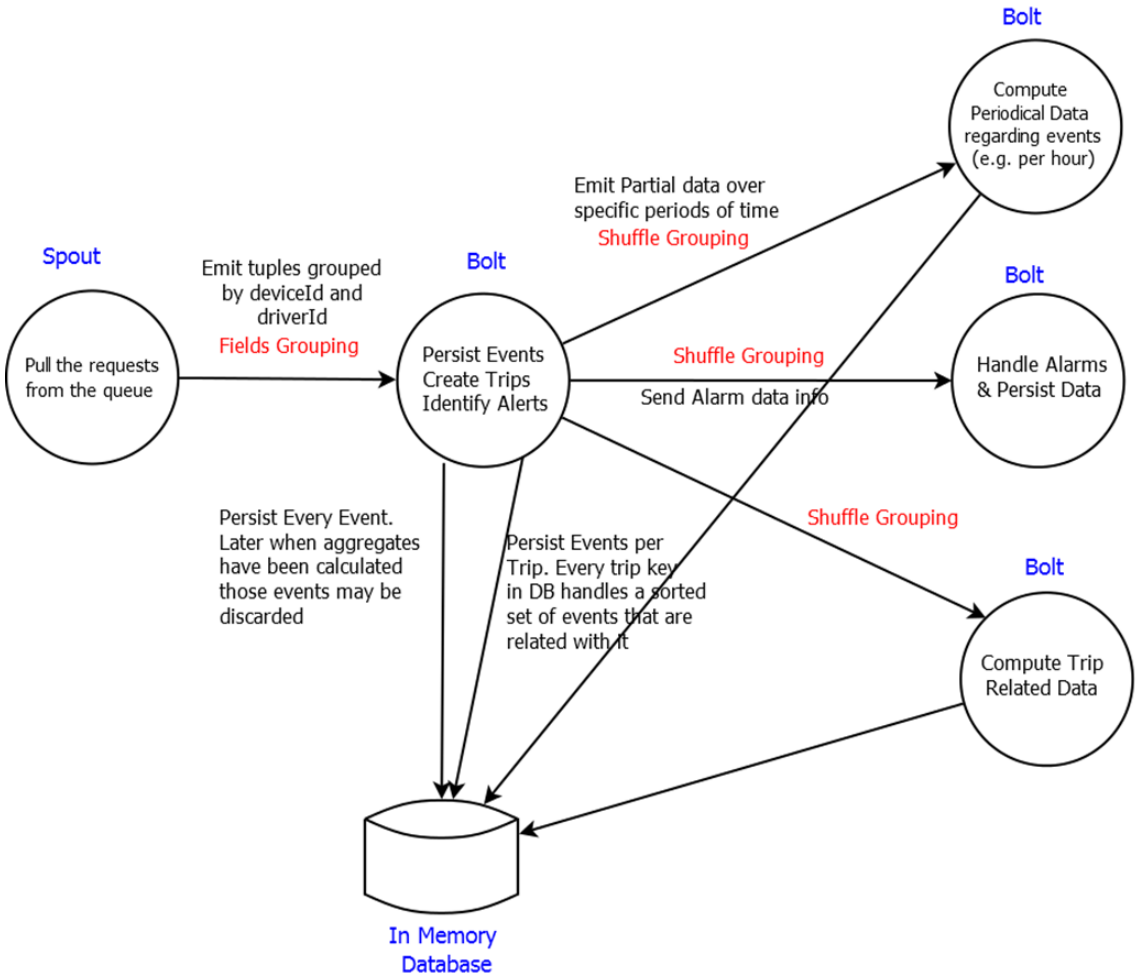

Figure 18: Storm topology

The high demand for real-time reads and writes asks for an in-memory OLTP solution.

4.2.3 Data Expiration

The architecture assumes two parallel mechanisms – the analytical and the speed layer – creating the same views. In order for the query mechanism (of the serving layer) that will rely on CloudMdsQL, to retrieve answers from both views and construct a complete response, appropriate data from the speed layer should be removed. Therefore, a continuous mechanism will remain live, that will discard data from the speed layer, under the condition that the analytical layer has already computed the views including them. To achieve this, the approach followed calculates a time interval for which it is safe to keep data in speed-layer. This introduces redundancy as we need to keep two real-time views and alternate clearing them after each analytical layer run. The flow and the views in time of the expiration mechanism are illustrated in Figure 19:

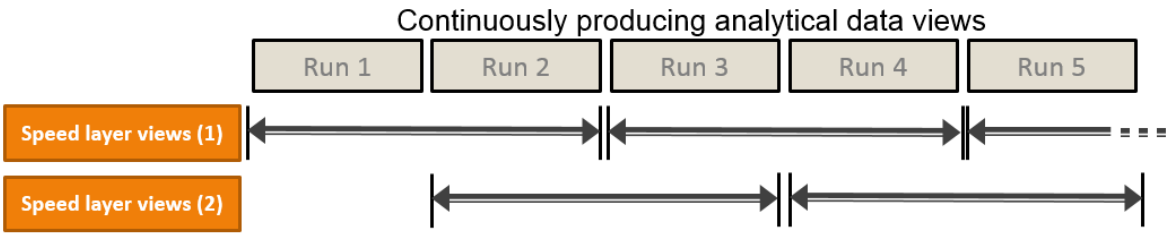

Figure 19: Expiration Mechanism views flow.

4.3 M2M: Design Level

4.3.1 Requirements Context in numbers

As in D.9.1 “Use Case Requirements”, the basic quantitative indicators of performance and success that need to be achieved through re-engineering of the use case are:

- Ability to handle each event at a time lower than 4 ms
- Multiply current throughput as much as possible; reach throughput equal to k X 120.000 events per hour, where k >=3
- Be linearly scalable (able to support 10.000 vehicles increase each year) which means to maintain while load sizes become larger
- Growth of master data size : ~ 25 Gbytes per month (current size: 1 Tb)

To achieve as above, we need to abandon the current architecture (as described in D.9.1) which features bottlenecks in the data layer, especially because statistical queries for dashboard creation become more and more, data size has already increased to 1 Terrabyte (while growth will be accelerated) and reduce database performance into real-time query needs. Latency for event handling is now: 10ms to route the message through queuing system and 300ms for the overall handling, which becomes far from real-time as sizes and query loads grow. It must be noted that statistics calculations locally at the car device are very little, because data need to be sent as quickly as possible to the control center (the fleet manager) where real-time views are required and to be correlated with other event data.

The need to separate real-time context from the persistent data of events is apparent. CEP will take over the real-time processing while analytical stores can efficiently respond to the growing master dataset of events. But you need a way to join the two worlds into one through a holistic transaction shell. CoherentPaaS promises to respond to this need.

4.3.2 Data Model

4.3.2.1 Generic Data Model

The reference data model consists of the main entities, which are the **vehicles** having also **drivers** who are making **trips**. The vehicles are equipped with **devices**, which are the producers of the centric data components, the **events** originated by the vehicle devices. Events can trigger cases of **alarms**, as well as link to **positions** on the map, which can be related to **points-of-interest** (POIs) or can signal the entrance into or leave out-of a **geofenced** area. The network of data entities and the relevant associations among them are visualized in the following Figure 20, including:

- Vehicles
- Devices
- Drivers
- Trips
- Positions
- Geofencing zones
- Points of Interest
- Alarm handlers
- Messages-Events

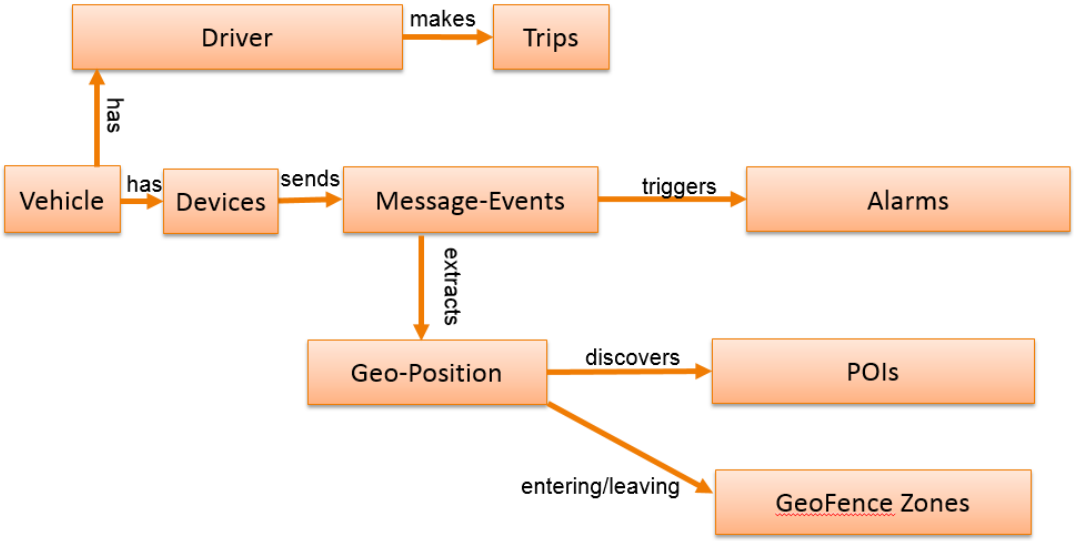

Figure 20: The participating data entities and their relations

Messages-events are appended into the master dataset within the architecture approach. Other entities are reference data which are used to find parts of query parameters or query responses and are primarily stored in relational database.

4.3.2.2 Master Dataset

The master data set is an immutable append-only data set where all event **messages-events** are recorded as tuples. Indicative fields that are included in the event tuples of the master data set are the following:

- Reception: reception datetime in local time
- GPS Date: GPS datetime of the event
- Type: message type. In this case, 0 means a standard tracking message.

- ID: unique device ID
- #: message numerator
- GPS: gps signal available (ex hibernation=not available)
- Flags: technical flags
  Trigger: transmission reason (32=ip up, 44 tracking, 69=driving start, 53=driving stop
- Km: mileage counter
- DriverID : driver identification
- Fix: Last GPS fix timestamp
  - GPS1: gps quality information
  - GPS2: gps quality information
  - Sat: number of satellites used
  - Long.: Longitude
  - Lat.: Latitude
  - Alt.: Altitude
- Speed: ground speed (km/h)
- Dir.: direction of movement

The indicative fields are the ones that will be mostly processed throughout the use case. The event tuples includes many more fields of lees frequent use.

4.3.2.3 Reference Data

These are the fleet and driver's data, the geoinformation and many other items of information:

- POI, geofencing zones
- User entities
- Alarm handler/types
- …

They are mostly wanted to get data and supplement results with the necessary details.

4.3.2.4 Data Views – Speed Layer and Analytical Layer Views

The use case primary data are the events produced and recorded in the master data set. The lambda-architecture will be continuously producing datasets, to construct the variant data views, both in the speed layer and the analytical layer context, which need to be prepared and be available for the queries to be performed.

The data entities that will represent the necessary views (the speed layer views and the analytical layer views, the latter created by the continuous pre-computation process) are presented below and summarized in the Figure 21.

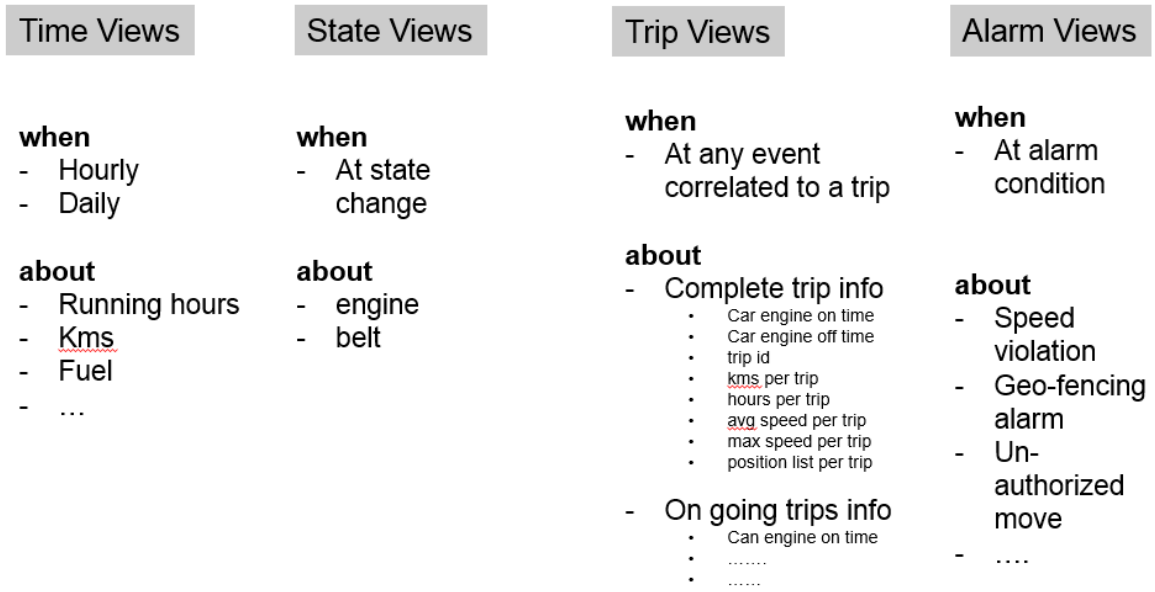

Figure 21: Data Views Summary

All views are categorized as:

- **Time views:** updated periodically with metrics sensed
- **State views:** updated when an state change is recorded for a notable status indicator
- **Trip views:** updated when an event is correlated within an ongoing trip
- **Alarm views:** updated when an alarm is produced

They are analytically described in subsequent paragraphs.

4.3.2.4.1 Views over time

Timely views will be created in the speed layer and precomputed in the analytical layer, for specific metrics; they can be at more than one granularity levels (i.e. per hour, per day, per month…). In our case:

- the hourly granularity will be mostly popular
- a daily granularity view will be also produced

Rolling up hourly values could provide the way to produce the daily values.

There might be queries which might need to access more than one view, i.e. when requesting the kms covered from 20.Apr.2014 17:00 until 22.Apr.2014 23:00 will result in the addition of two/three day values plus/minus hour values.

Timely analytical views will represent:

- the absolute value of metrics at specific periodical time points
- the calculated change of values (deltas) on each period
- the average/maximum/minimum metric value within each period

The following analytical-views are created, where cumulative values might in certain cases also accompanied with Delta-values:

Running hours

- by hour [vehicle, daytime] #hours
- by day [vehicle, day] #hours

Kilometers

- by hour [vehicle, daytime] #kms, delta (#kms)
- by day [vehicle, day] #kms, delta (#kms)

Fuel consumption

- by hour [vehicle, daytime] #lt, delta (#lt)
- by day [vehicle, day] #lt, delta (#lt)

The table below provides an example part of hourly view of Kilometers, both absolute and delta values. The part shows the view for a specific vehicle.[7]

Table 3: Hourly view of Kilometers for a specific vehicle

Vehicle	Hour	#Kms
HKH7561	30/4/2014, 16:00	165.678
HKH7561	30/4/2014, 17:00	165.686
HKH7561	30/4/2014, 18:00	165.686

Vehicle	Hour	Δ #Kms
HKH7561	30/4/2014, 16:00	…
HKH7561	30/4/2014, 17:00	8
HKH7561	30/4/2014, 18:00	0
HKH7561	30/4/2014, 19:00	18

Two views per metric will be necessary for these types of values and their timely views.

The following analytical-views are also created, where average/comparative metric results are presented in time:

Average speed

- by hour [vehicle, daytime] speed
- by day [vehicle, day] speed

Maximum speed

- by hour [vehicle, daytime] speed
- by day [vehicle, day] speed

Two views per metric will be necessary for these types of values and their timely views. There is no meaning for such values to record delta value.

[7] The tables are showing the data views and not the way they are to be stored, in either a k-v store or a columnar store which might be the actual case for most metrics.

4.3.2.4.2 State Views

State views will be recorded in real-time as well as precomputed in the analytical layer; the application needs to record only the changes of status along with the associated time they happened. The most recent state must be kept and updated in order to decide if a state-row must be inserted to indicate status change

The most critical change of status is related to the engine ignition on/off, which also signals the start and end of a trip. Two views will be necessary for the engine status. However, as “on” and “off“ changes of status are paired to form a trip, the trip views also keep the start and end time of a trip (which is the on-daytime and the off-daytime)

Engine State

- On-view [vehicle] daytime
- Off-view [vehicle] daytime

Other state views are related to the seat belt:

Seat-belt State

- Set-view [vehicle] daytime
- No Set-view [vehicle] daytime

Other state views are showing the status of the generator, the temperature sensor, etc. For each state metric, a set of views will be computed at the analytical layer, depending on the number of the state values.

4.3.2.4.3 Trip Views

The system must also create and “precompute” the trip views and provide values also at “trip-granularity” level. This means that trips must be identified and values which are measures over time should be also recorded at trip level. Positions view will be also integrated in trip views.

Pairing on-off state changes of can engine ignition signals a trip. The process should record vehicle values at trip start and trip end, compute delta-values, compute average and maximum values and record list of positions within the trip.

Trip view is a table of trip entries, each one containing the following:

Trip

- Car engine on time
- Car engine off time
- trip id
- mileage per trip
- Hours per trip
- avg speed per trip
- max speed per trip
- position list per trip

Another view is keeping only the current trips. This is a view maintained only in the speed layer.

4.3.2.4.4 Alarm Views

As alarms are produced, they are stored into alarm-type specific view. The system must also precompute relevant analytical-views

Alarm

One view will be created per alarm type i.e. speed violation, geo-fencing alarm, un-authorized move, un-authorized driver, etc.

- Alarm [vehicle] alarm_value(s)

4.3.2.5 Data Stores to use

To persist the data related to these concepts and views we suggest to use different datastores to reduce the number of I/O operations and maximize performance; the selected solutions are listed below, ranked according to their importance in the use case:

- **Complex Event Processing**: To achieve reception and processing of vehicle events at high throughput; Storm framework will be utilized for scalable and fault – tolerant data reception
- **In-memory OLTP**: Derby will be utilized, to persist real time data views, as described in previous sections. The most recent data will be cached there, for quick views on present vehicle status and ongoing trips. Other data which need to be always fetched rapidly might also be stored in an in-memory solution (i.e. alarm handler data, as they are necessary in order to in real-time identify possible alarms)
- **Columnar**: To persist the master data set and analytical layer views where aggregated data and analytics need to be produced, as described in previous sections. Different types of columnar stores might be appropriate, due to the high volume demand of the master dataset which demands for a distributed file system. MonetDB and Hbase will be challenged through the use case.
- **RDBMS**: To persist basic reference data and configuration metadata required to return as part of the projection of a given query, i.e.
    - Drivers (DriverID, Name, Surname, …)

4.3.3 Overall Architecture – CoherentPaaS role

Having analyzed the various datasets, it is evident that apart from the basic data (events and reference data) the use case relies upon the continuous production and update of several data views, residing in both analytical and in-memory solutions (for the real-time data requirements). Therefore we foresee CloudMdsQL layer receiving query requests

- To access data views from both the speed and the analytical area
- To create and update the data views, based on the data output of the stream processing by the CEP frame and the raw data.
- To combine results, with required data which are available within relational database or in cache.

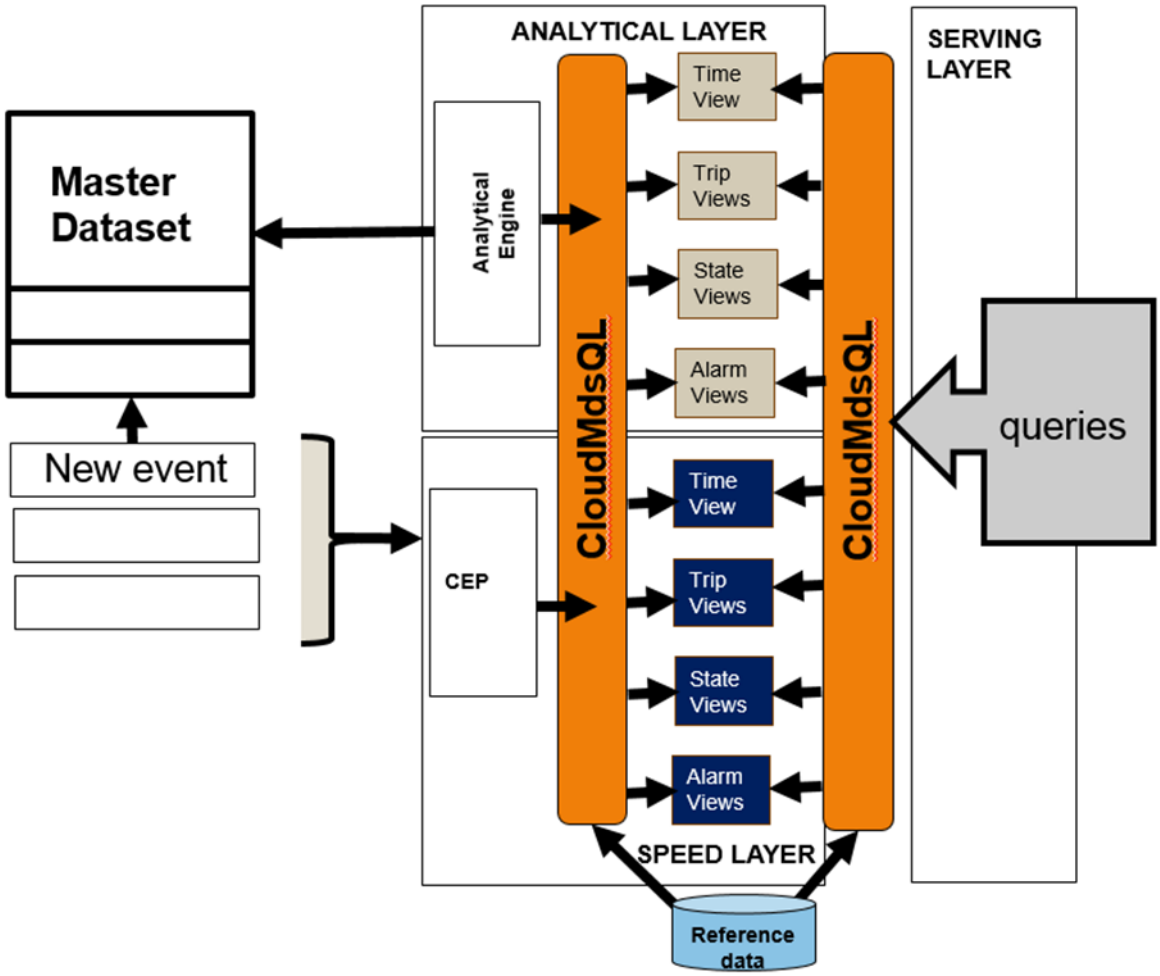

Figure 22: Vehicle Telematics with CPaaS Architecture

The Figure 22 above shows this dual role of CloudMdsQL, as facilitator of application queries and facilitator of core data maintenance functions. Simplification of code, speed acceleration and reliability of real-time data are the mostly wanted features expected from this setup.

4.3.4 Functional Components

The functional categories are the following:

- Dashboard – viewing consolidated information in real time for several metrics for a specific entity (vehicle, driver or group of vehicles). Alarm information should also be given.
- Trip Queries – viewing information for trips (current and past trips, statistical views of trips)
- Positions Map – aiming to show the positions of a vehicle on the map, and continuously update it. Relevant POIs and geo-fencing alarms should be noted.
- Activity Diagram – viewing the activity of a vehicle in time.

Queries are involved in each application family; the most representative will be described below. Queries concern a vehicle, a driver, company or other groups of similar entities. Several queries will need to use different datastores simultaneously. Having CloudMdsQL will simplify the code of integrating result sets of different datastores. The following subparagraphs describe these functionalities and the representative queries as required,

as well as the involved datastores. For simplicity, we do not replicate query description and definitions when we change the applied entity.

4.3.4.1 Dashboard

Information must be consolidated to produce dashboard like:

- Cumulated distance
- Mean speed, max speed
- Driving time, stop time, idling time, …
- Fuel consumed
- Safety scoring

Figure 23: The Vehicle Telematics M2M Case – Dashboard

Dashboards can be built:

- for all vehicles, a group of vehicles, one vehicle
- for a fixed period (this day, this week, this month, this quarter, …) or for a dynamic period, with a breakdown by hour, by day, week, month, …including also real-time views

4.3.4.1.1 View progress of metrics in past interval or now

- Hourly or daily views of kms, consumption, driving time for a given interval
  For instance the query below provides the hourly kilometres on an hourly basis for a given vehicle within a time period (from T1 to T2)

GIVEN:

- **Period** [T1, T2]
- **Vehicle** Vid

FIND: distance per hour

➢ *SELECT Kms-metric, Hour*

*FROM (**kms_hourly view_Analytical***
WHERE Vehicle is Vid And Hour is in [t1, T2])

SELECT Kms-metric, Hour

*FROM (**kms_hourly view_Speed***
WHERE Vehicle is Vid And Hour is in [t1, T2])

- *Calculate delta-change in each hour*

The same query is planned as in Figure 24:

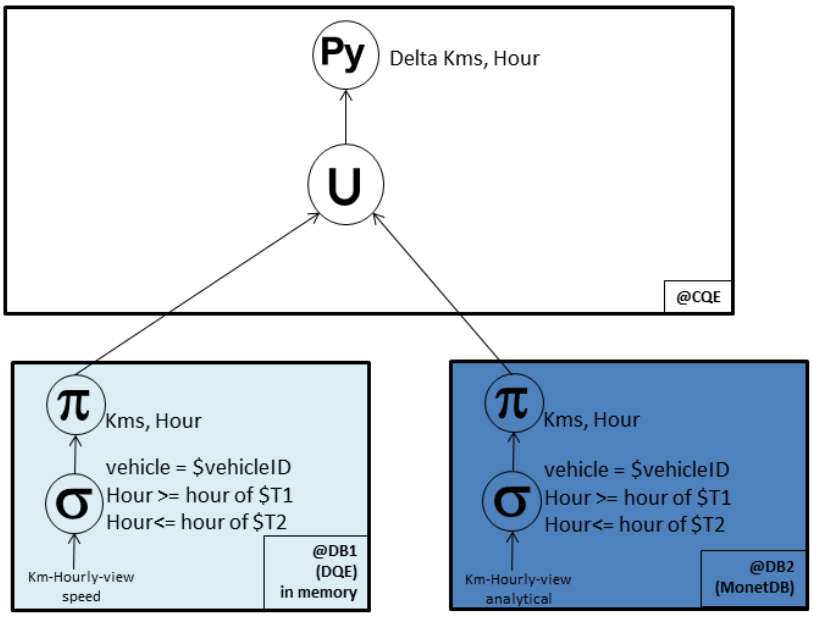

Figure 24: Union of query results from speed and analytical datastores: Dashboard query

This is a frequent query plan for this use case, as it provides the union of results extracted from the two layers. It will be seen in many subsequent query examples.

- Hourly or daily views of avg speed, max speed for a given interval
- Daily cumulated distance

4.3.4.1.2 View statistics

- Fuel consumption per kilometer for each vehicle/driver (a sorted list)
- Total kms per driver or per vehicle
- Driver details sorted by their fuel/km consumption, as in query example described below:

GIVEN:

- **Period** [T1, T2]
- **A list of vehicles** [V1, Vn]

FIND: provide driver details sorted by their fuel/km consumption

- *SELECT DriverID, sum(Mileage), sum(FuelConsumption)*

  FROM (FIND Trips for Period [T1, T2] in TripView_Analytic

  WHERE VehicleId IN [V1, Vn]

  GROUP BY DRIVER ID


```
SELECT DriverID, sum(Mileage), sum(FuelConsumption)
FROM (FIND Trips for Period [T1, T2] in TripView_Speed
WHERE VehicleId IN [V1, Vn]
GROUP BY DRIVER ID
```

- *ORDER By sum(Fuel)/sum(mileage)*
- *GET THEIR Phone Numbers*

4.3.4.1.3 Alarms given – past and real-time

- List of alarms in given interval
- Find a specific alarm type
- Find vehicles that produce a specific alarm within a given interval
- Alarm view in-real time
- Find number of alerts raised per day

4.3.4.2 Trip Queries

The trip is the most substantial concept within vehicle telematics applications. Henceforth, queries at trip level should find information for specific trips, the current trips, past trips, as well as provide statistics for trips.

The queries described below will correspond to a specific vehicle, a specific driver or even a group of them (a fleet, a company).

A complete trip information (trip profile) should consist of a set of values as listed below:

- Time trip started
- Time trip ended
- Distance covered
- Time duration
- Average Speed
- Maximum Speed
- Fuel Consumption

List of positions could be also included in case the application demands to re-create the trip on a map.

4.3.4.2.1 Current Trip Information

- Find Current On-going trip(s)
- Provide in-real time the current trips profile
- Find the telephone numbers of trip drivers now in the GeoFenced Area, as in query example described below:

GIVEN:

- **Geofenced Area** [(x,y) ….. (x,y)]

FIND: the telephone numbers of trip drivers now in the GeoFencedArea

- *SELECT DISTINCT DriverID from **TripView_Speed***
  WHERE Last_Position falls in Geofenced_Area

FOR EACH DriverID

- *Retrieve the Driver Phone Number from DRIVERS*

4.3.4.2.2 Trip Statistics

- Find trips and trip information for all trips
- Find the average maximum speed of all trips of driver X
- Find kms
- Find the longest trips. Sort trips by distance covered
- Find the average distance of trips
- Find the trip with the highest consumption
- Find the fastest driver over trips within a given interval
- Find the maximum speed per trip for all trips of a given vehicle that started after T1, as in query example described below and illustrated in Figure 25 :

GIVEN:

- **Vehicle:** Vid or Driver: DRid
- **Time: T1**

FIND: max speed per trip for trips that started after T1

- *SELECT TripID, MaxSpeed from **TripView_Speed***
  WHERE Vehicle is Vid and Time Trip Starts >=T1

SELECT TripID, MaxSpeed from TripView_Analytic
WHERE Vehicle is Vid and Time Trip Starts >=T1

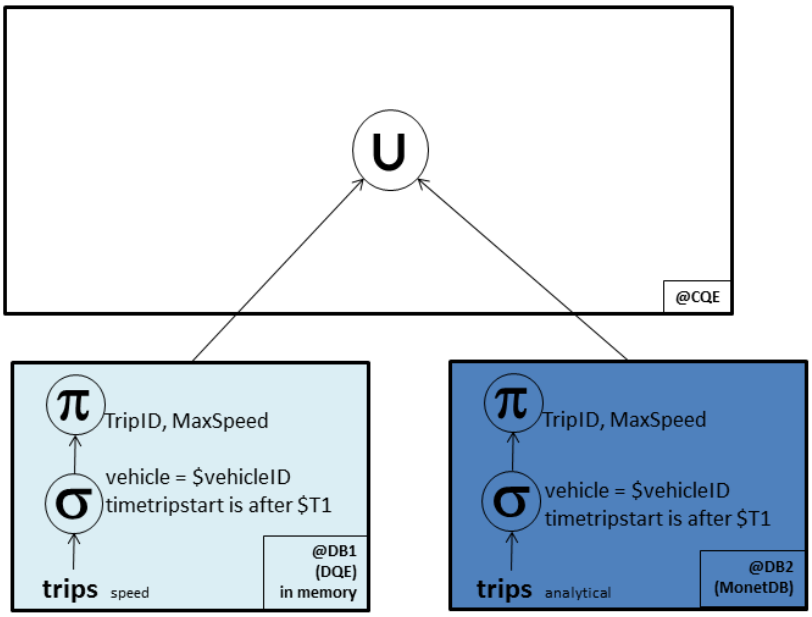

Figure 25: Union of query results from speed and analytical datastores: Trip query

- Find kms per driver
- Find the trips per driver within a given interval
- Find the telephone numbers of trip drivers travelled since T1, as in query example described below:

GIVEN:

- **Time:** T1

FIND: the telephone numbers of trip drivers travelled since T1

- *SELECT TripID, DriverID from TripView_Speed*

  WHERE Time Trip Starts >=T1

  DistinctDriverIDs

  merge

  SELECT TripID, DriverID from TripView_ Analytic

  WHERE Time Trip Starts >=T1

FOR EACH **DriverID**

- *Retrieve the Driver Phone Number from DRIVERS*

The execution plan of such query is also illustrated below, in Figure 26:

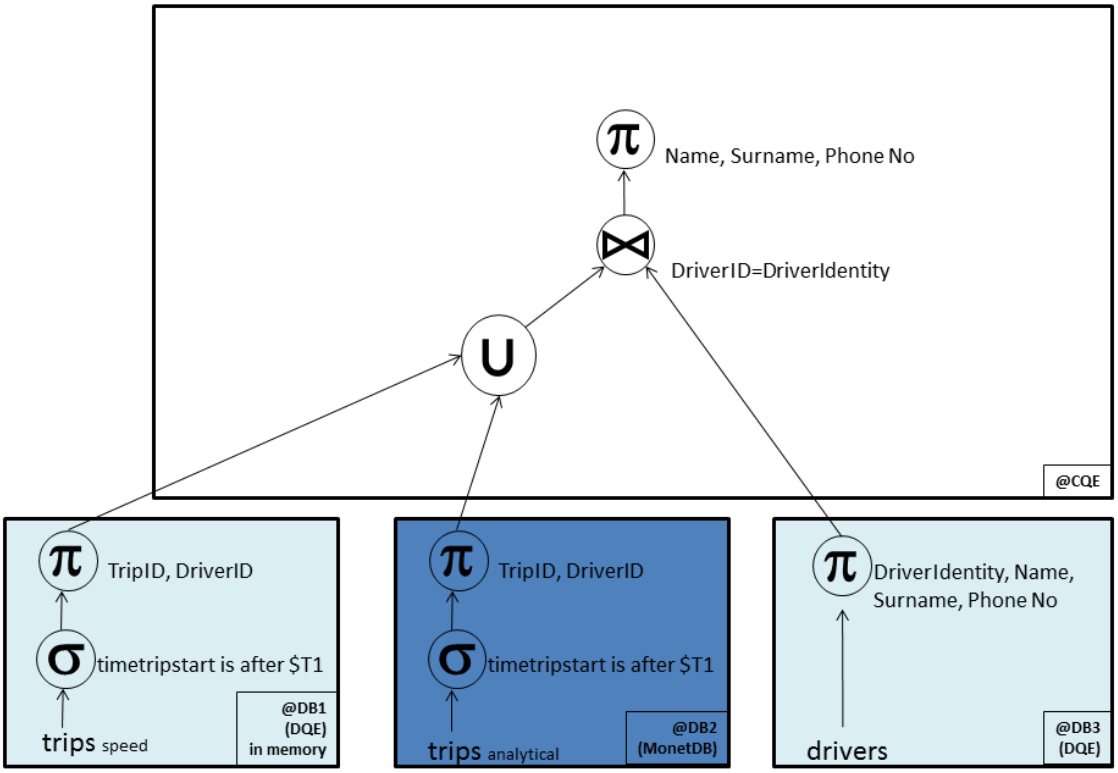

Figure 26: Getting the driver details for a given union result of trips

4.3.4.3 Positions Map

Query results will be delivered to the Web user interface to produce real time map with the trip of the sensor. The positions are updated in a map as soon as they are received.

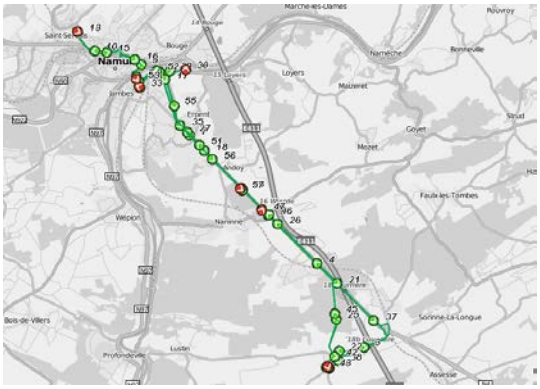

Figure 27: The Vehicle Telematics M2M Case - Mapping New Positions

Positions should be examined, to check relevance to one or more POI, as well as whether a geo-fenced area is entered or exited.

4.3.4.3.1 List of Positions – Past and real-time

- Find list of positions within a given time interval
- Real-time positions of a vehicle/driver
- Find give positions of vehicle from T1 till NOW

4.3.4.3.2 POIs

- Find which vehicles are near a POI / got at a POI within a time interval
- Produce an alarm when position is at POI

4.3.4.3.3 Geofencing event

- Produce an alarm when position is entering a geo-fenced area
- Produce an alarm when position is exiting a geo-fenced area
- Find vehicles / drivers / trips who entered an area within a given interval
- Find the telephone numbers of trip drivers now in the GeoFenced Area
- Find time stay within a geo-fenced area

4.3.4.4 Activity Diagram

The activity (stop, driving, idling …) of a sensor must be updated on real time and query results could present the vehicle activity in time, its current status, as well as, states for other vehicle components.

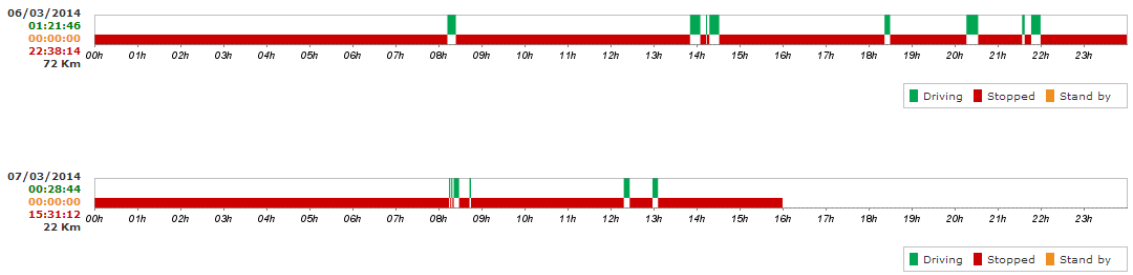

Figure 28: The Vehicle Telematics M2M Case - Monitoring Sensors.

4.3.4.4.1 Engine status time diagram – past and real-time

- Get the engine on-off state changes within a given time interval
- Get the real-time view of engine on-off state

4.3.4.4.2 Find drivers with seat-belt off – past and real-time

- Find the drivers who have made trip at intervals where seat-belt has been found at off-state
- Produce real-time alarms upon drivers putting belt at off-state while on-trip.

4.4 M2M: Expected Benefits

The existing architecture imposes serious shortcomings, especially in respect to the ability to accommodate increasing needs, in terms of message throughput, analytical query execution and real-time alarm identification and handling. Therefore, the proposed approach:

1. Moves all processes and store facilities into cloud cluster settings,
2. Separates real-time processing from analytical oriented processes
3. Enables the union of the real-time context with the past data results, when that is required

4. Selected the best appropriate data store framework for each data set according to the data needs

Separation of the real-time context and utilization of different data technologies, from Complex Event processing and fast OLTP to analytical databases, will be facilitated through CPaaS. In that context the ability to handle larger message throughputs as number of devices increases will not be affected by the load of user queries and dashboard views.

Apart from the above, more queries will be defined that will satisfy important business needs. Raw data will be processed efficiently so that the notion of trips will be stored and trip statistics will become also available. In that context, the real-time processing does not only facilitate the creation of real-time views and alarm handling, but they can also supplement raw data with additional information when that is necessary (in case of trip identification, assignment of trip identification to events is a useful data enrichment, filling the gap of sensor data).

4.5 M2M: Common CPaaS Components

An interesting common component that the use case needs is the union operation among the results of two queries, as applied into two parallel data stores (in our case, the one with the real-time data and the one with the analytical data). The union could be facilitated as one operation.

The union result could define the new column name and the names of columns from the two participating query results which are associated to the new column name. Renaming of columns makes sense when the names are different across different data stores; otherwise they keep the same name also in the union result. For instance, if I want to get the names and addresses from two different stores (DS1 and DS2) of different type that satisfy a certain criterion A in DS1 and criterion B in DS2, and get the result onto the same data table for further processing. I need the union of two query results, the named queries Query1 and Query2. In DS1 the names and addresses are under columns named: Name and Address, while in DS2 they are under the columns named: NameId and LocationAddress.

I.e. The hypothetical component for union of queries (named UnionQ in this example) is defined by its column names and the participating queries, each one accompanied with any necessary renaming:

```
UnionQ (Name, Address; Query1, Query2 (NameID to Name, Address to
LocationAddress))
```

It is explained in the Figure 29 below:

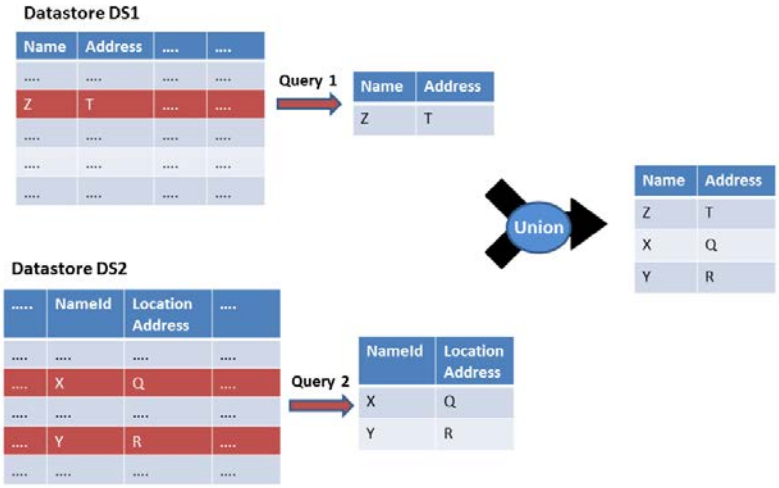

Figure 29: CPaaS Union Operation

5 MEDIA PLANNING USE CASE

5.1 Media Planning: Use Case Overview

Millions of people communicate each other using social network (e.g. Twitter, Facebook, WhatsApp, blogs, forums, e-mail, and so on). Daily, these social platforms produce huge amount of data that is very interesting for **marketing purposes**, but cannot be analyzed with traditional technologies.

In the most popular social networks, the analysis of registered messages allows to know the public brand perception and then perform corrective actions through the social networks to change it, such as designing effective marketing campaigns, analyzing your target audience classifying it by roles or predicting the impact of a given communication with the most influencers in a given topic/community.

The main operations included in this use case are as follows: retrieve the most influencers, retrieve the most important communities and add new documents (messages) in real time. The main stakeholders involved are: community managers, marketing vendors, cloud systems providers, technology providers and data providers.

5.2 Media Planning: Architecture Level

The Media Planning Architecture is a **Lambda Architecture**, because it works with a data-store that is immutable. An immutable data store essentially eliminates the update and deletes aspects of CRUD, allowing only the creation and reading of data records. It works in real time and calculates query results on top of an incoming stream of data. Results, once computed, should be stored in such a manner that they can be queried by applications. Just as with the analytical layer, the real-time layer also stores results as they are computed into a view.

The main processes involved in a Lambda Architecture are: **the data stream** (e.g., tweets emitted in real time), which is the input of the **loading process** of the involved datastores; **the precomputed analytical view** which is the minimum and optimized output generated by the loading process for the queries invoked from the application; and the **precomputed real-time view**, which include just those simple data that is ready to be queried. The main architectural components involved in the loading process are **Trident** and **Storm** because these offer a framework to have a scalable loading architecture with real time queries over the processed data.

The **application** is a REST API containing the queries about communities and influencers. This REST API is exposed through a HTTP server and the queries are designed using a MVC framework called **XWork** because it is completely decoupled from the HTTP layer and then, the whole workflow can be executed without having on an HTTP infrastructure. XWork contains a Dependency Injection component to avoid creating factories and manage different implementations of the same interface. XWork also provides Interceptors to offer a way to configure pre and post processes before applying a request and avoiding aspect oriented programming.

The application and the loading process are connected by the data stores because these are shared resources.

The following diagram shows the main components of the described architecture.

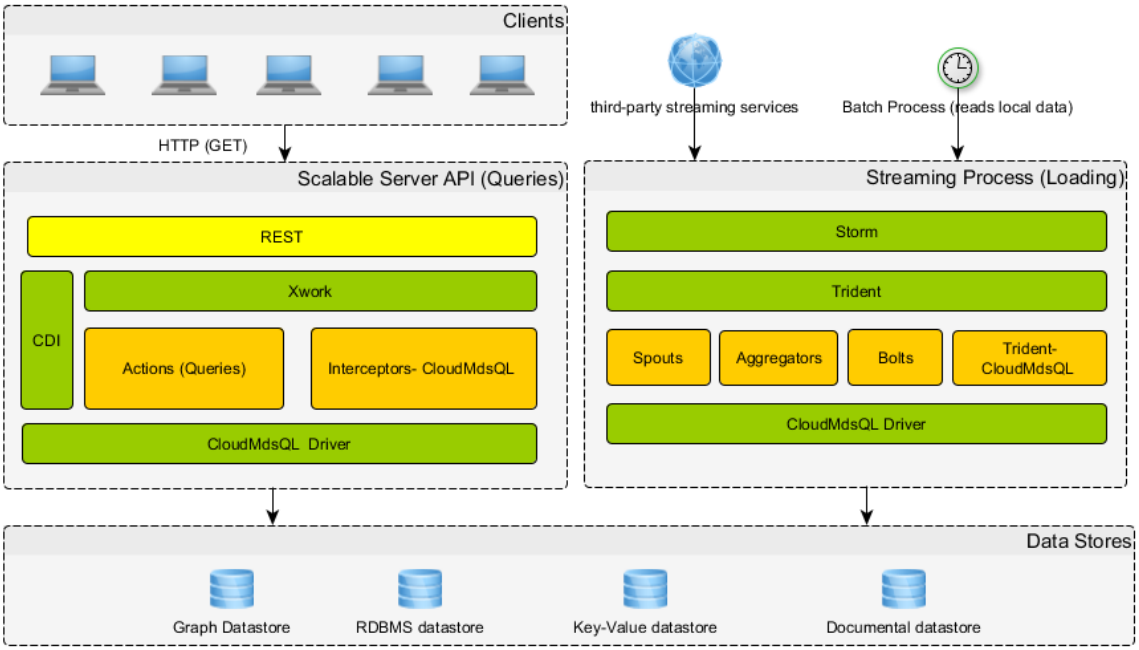

Figure 30: Architecture Main Components

Specifically, the XWork framework works as follows. The public API is configured by means of an XML configuration file called xwork.xml. This configuration file allows defining actions, a mapping between a simple name and a java class. Actions represent requests. Through this configuration file, users can parameterize the CDI parameters and define the set of interceptors applied for each action. So, when a REST endpoint is invoked, the application creates a specific action invocation. Once an action invocation is resolved the configuration part related to this action invocation is instantiated. The workflow continues as follows: interceptors are executed to apply all preprocesses, then the action and the result (a component to process the results offered by the action, e.g. printing them into the http response as a JSON object). Finally, all interceptors finish to apply post processes.

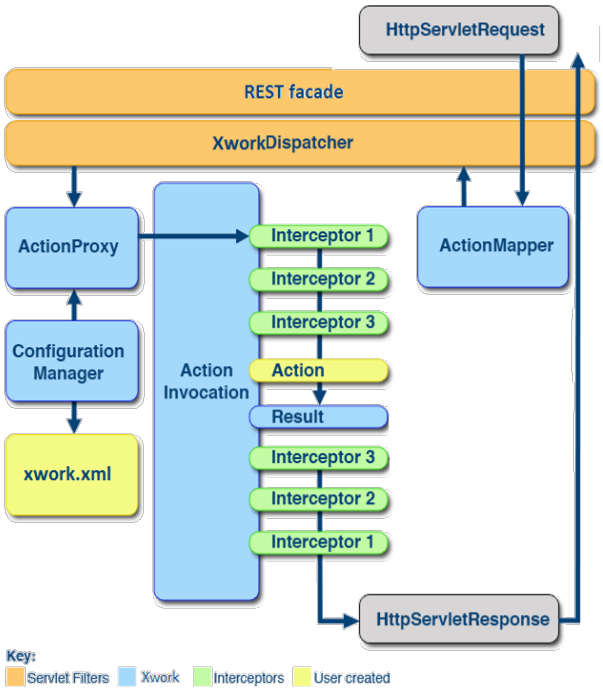

Figure 31: XWork work flow

From the physical point of view, the application needs three main clusters: one for the loading process (Storm), another one for the REST application and finally another one for the CloudMdsQL, the query engine cluster. On the other side, users need to have a partial view of the whole information and therefore, probably it will have implications in the final physical datastores that run on over the CloudMdsQL.

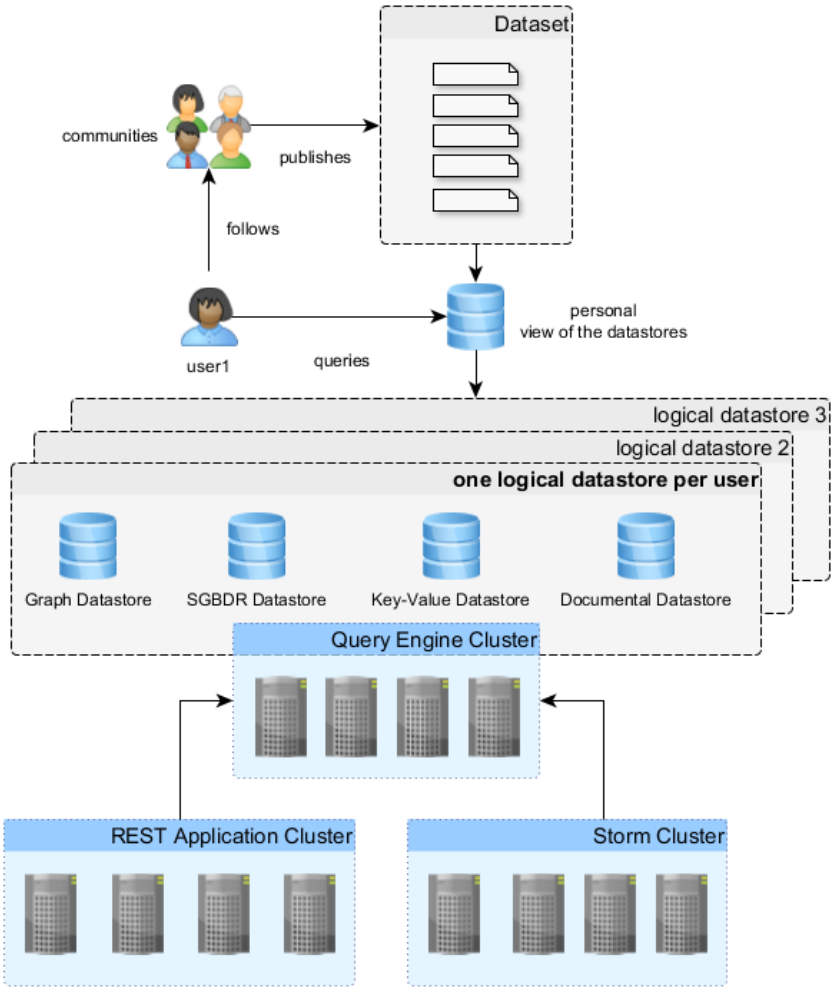

Figure 32: Media Planning Architecture

5.3 Media Planning: Design Level

5.3.1 Data Model

In the Social Media use case, the main concepts involved are **documents** (e.g. microblogging messages), their **references** and **copies** (e.g. retweets), the authors of these documents of the referenced people called **entities**, the **location** of this people and the **topics** employed as **tags** in the existing documents.

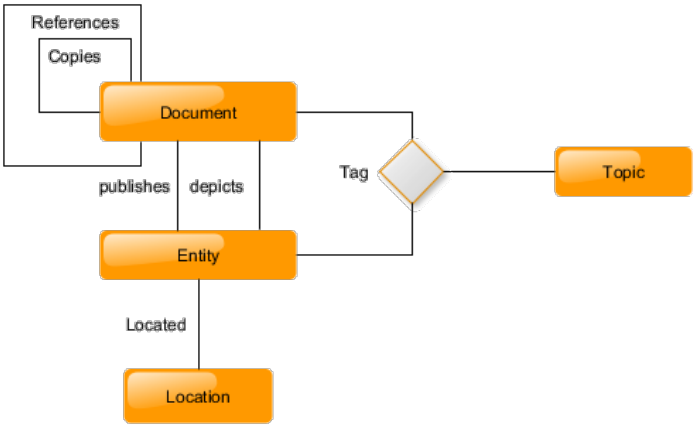

Figure 33: Main Components in the Social Media Use Case

To persist the data related to these concepts we suggest using different datastores to reduce the number of I/O operations. They are listed below, ranked by their importance in the use case:

- **Graph**: To persist all relationships used for filtering purposes (e.g. to improve future updates efficiently avoiding scans and multiple queries).
  - Copies, References.
- **Documental**: To persist all data which could have a flexible size (e.g. text).
  - Document (id, text, follows, topics)
- **Key Value**: To persist list of complex items (more than one basic field) related with a given key that need to be analyzed in the same time during the query.
  - Tags: topic as a key and tag (entity, document) as a value.
- **RDBMS**: To persist all basic data required to return as part of the projection of a given query.
  - Document (id, date, channel)
  - Publishes (idDoc, idEntity)
  - Entity (id, name, screenName, registrationDate).

During the loading process there are more types of information inferred and used in a real time by the designed queries. These types are: influences between entities (*INFLUENCES_TO*), influencers of a given keyword (*INFLUENCES_ABOUT*), the weight of a keyword for a given document (*INFERRED_KEY*), and which document is the result of the propagation of another one (*PROPAGATES_TO*).

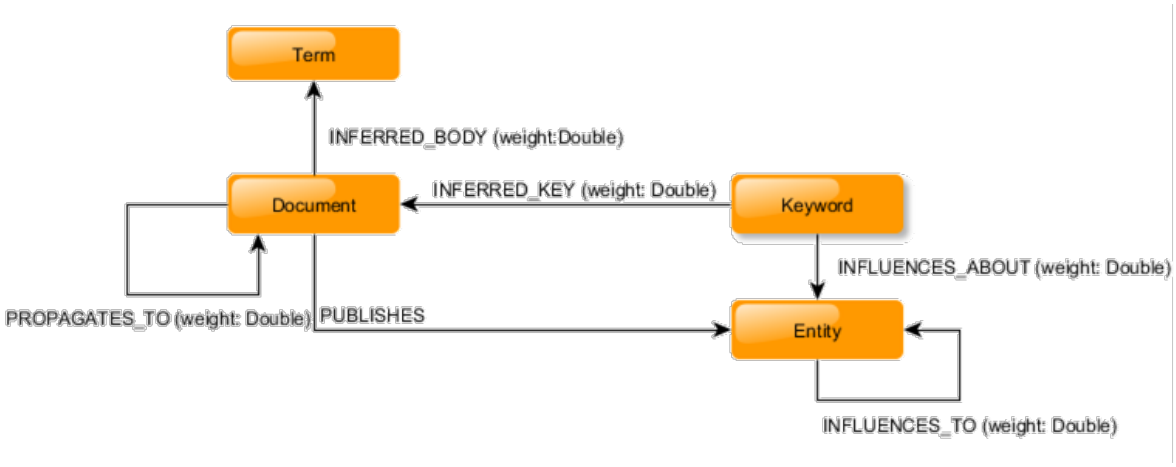

Figure 34: Loading Process

To persist the data related to these concepts we suggest using different datastores to reduce the number of I/O operations. They are listed below, ranked by their importance in the use case:

- **Graph**: To persist all relationships used for filtering purposes (e.g. intersections and unions). Specifically to infer communities and propagation trees.
  - INFLUENCES_TO, PROPAGATES_TO.
- **Key Value**: To persist list of complex items (more than one basic field) related with a given key that need to be analyzed in the same time during the query.
  - INFLUENCES_ABOUT: keyword as a key and entity with a weight as a value.
  - INFERRED_BODY: keyword as a key and documents with a weight as a value.

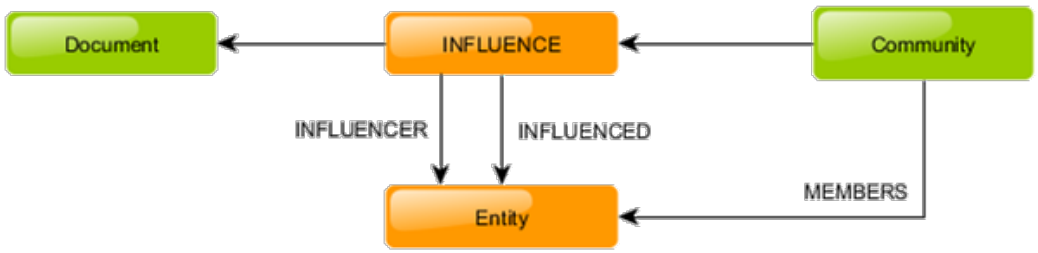

Figure 35: Relationships used for filtering purposes

To persist the data related to these concepts we suggest using different datastores to reduce the number of I/O operations.

- **Graph**: To persist all relationships used for filtering purposes (e.g. intersections and unions.
  - INFLUENCE, INFLUENCER, INFLUENCED, MEMBERS.
- **Key Value**: To persist list of complex items (more than one basic field) related with a given key that need to be analyzed in the same time during the query.
  - CLASSIFIES: keyword as a key and communities with a weight as a value.

5.3.2 Functional Components

The main functionalities provided by this use case are:

- Who are the most influencers?
- Which are the communities around a set of keywords?
- Adding new documents

All functionalities use different datastores simultaneously. So, having CloudMdsQL simplifies the code of integrating result sets of different datastores. The following subsections describe the formal definition of each functionality and the involved part of the schema of each datastore.

5.3.2.1 Who are the most influencers? (QUERY)

Definition: The 10 most influencers about a list of topics sorted by the sum of their probabilities in each topic. For each entity, the system shows:

- The number of (potential) influenced entities
- The maximum propagation graph
- All their basic fields: id, name, date.
- The last published document: id, text, date, total of propagations, channel, number of copies and number of references.

Queried types: *INFLUENCES_ABOUT* (key value), *Document, PUBLISHES, Entity* (RDBMS and documental), *COPIES* (graph), *REFERENCES* (graph) and *PROPAGATES* (graph).

The final execution plan of this query should return the same registers as this one:

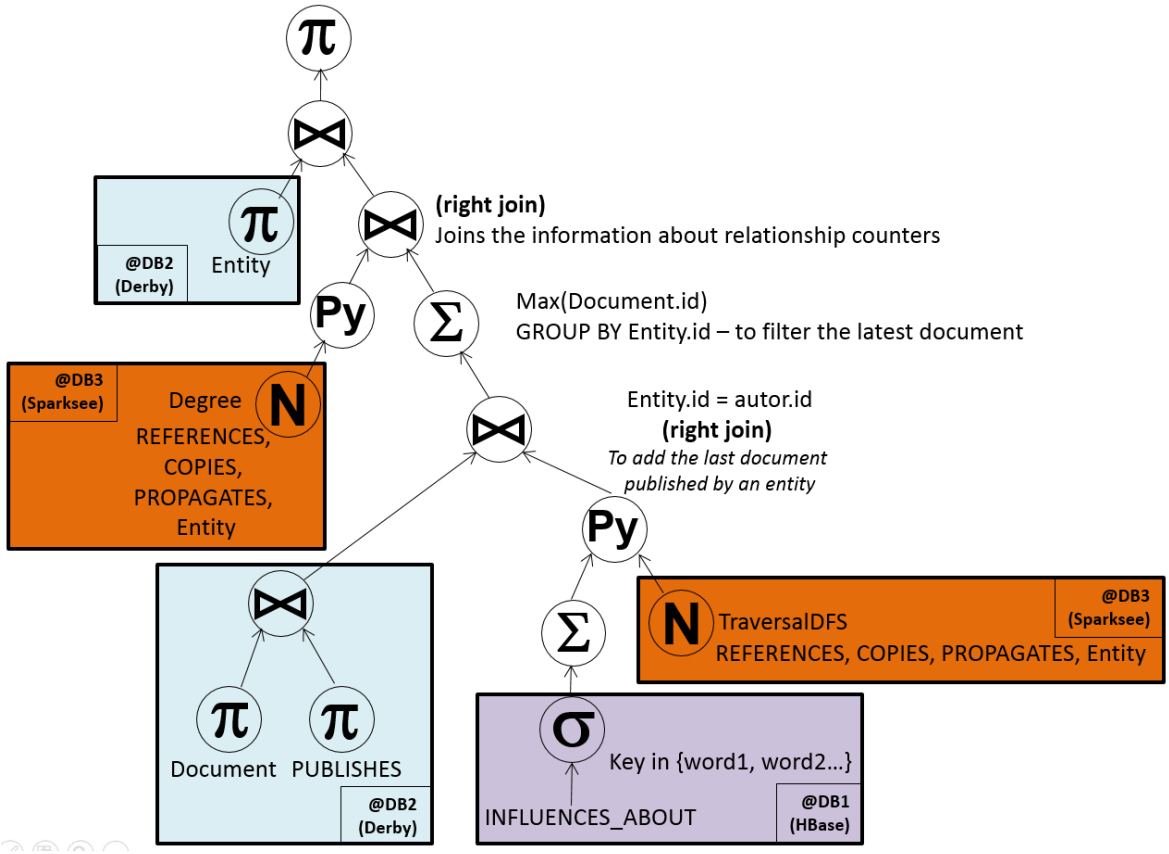

Figure 36: The most influencers query decomposition

5.3.2.2 Which are the communities around a set of keywords? (QUERY)

Definition: The 10 biggest communities for a given set of keywords. For each community, the system must return the 20 most influencers inside the community by microblogging documents. For each member, the system shows:

- The number of (potential) influenced entities inside the community.
- The maximum propagation depth.
- All their basic fields: id, name and date.
- The last microblogging published document: id, text, date, number of propagations, channel, number of copies and references.

Queried types: *INFLUENCER, INFLUENCED, MEMBERS* (Graph), *MEMBERS* (Graph), *CLASSIFIES* (key value), *Document and PUBLISHES* (RDBMS and Documental).

The final execution plan of this query should return the same registers as the following one.

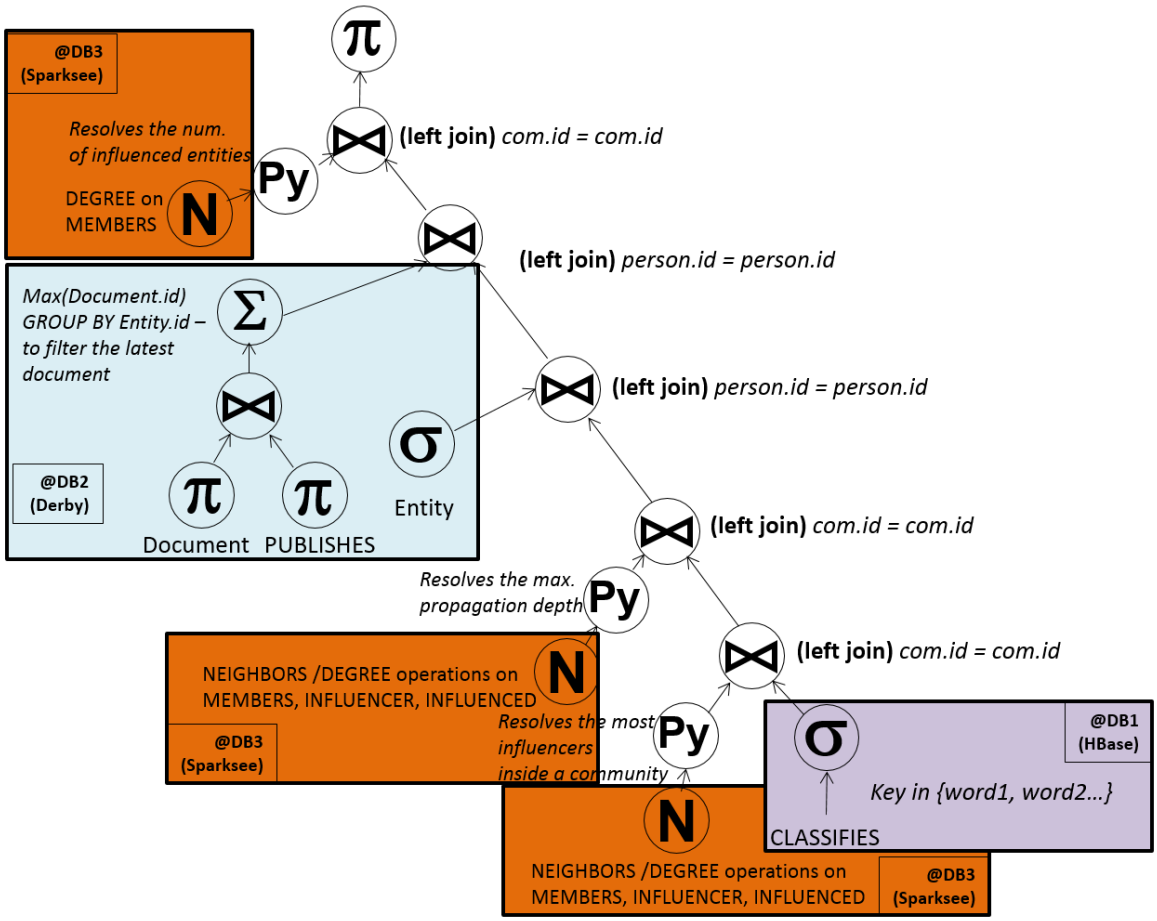

Figure 37: Communities for a given set of keywords query decomposition

5.3.2.3 Adding new documents (UPDATE)

The system needs to add all related basic and inferred information of a given document. In this process, influence relationships and communities are periodically executed. All this information needs to be precomputed to perform fast queries in real time. So, every datastore must be fed with the necessary data to simplify published queries.

Specifically, the system has to do:

- A deduplication process because documents may come from different communication channels.
- Text normalization to improve the quality of the further analysis.
- The document indexation
- Detect propagations among the new documents and the existing ones.
- Detect influence among people by their documents and calculate the communities produced by these relationships.

We describe how we will compute some of these steps.

Deduplication process

The deduplication process consists of process the author name using different criteria (exact value, initials, inverse and random letters) and create different blocks of data with a fixed length and compare between them to solve which registers may represent the same entity. We will use different algorithms, like the Levenshetein distance, to resolve how similar all pairs of registers are inside each block. Those registers that are more similar than a given threshold will be unified into the same entity.

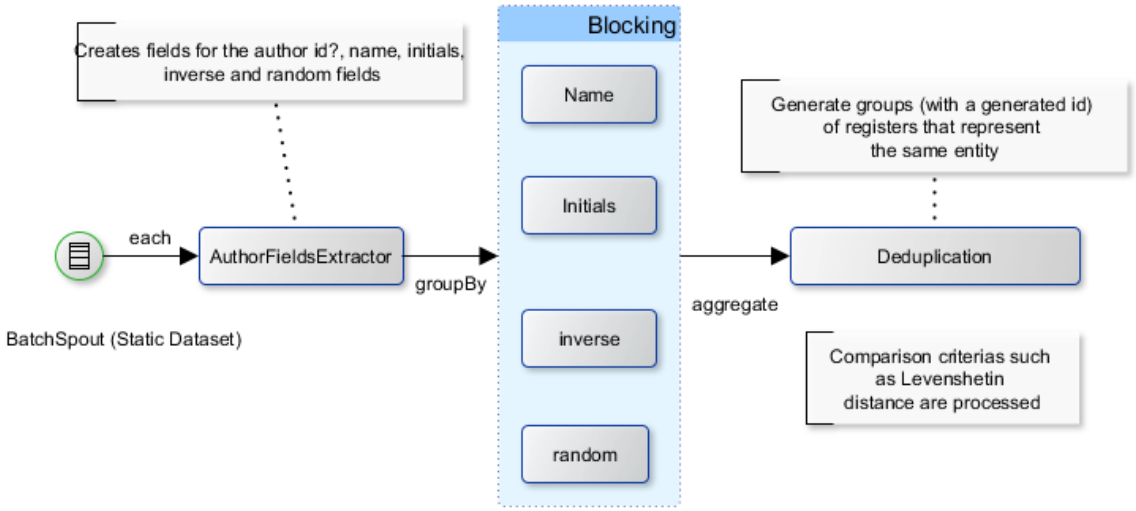

Figure 38: Deduplication process

Document indexation

The document indexation consists of splitting by spaces the documents' text into tokens and calculate the relative weight of each word(token) for the document and the global weight to calculate the **TF-IDF** (Term frequency – Inverse document frequency). This calculus is a well-known approach to **resolve which documents are similar** to a given document and then. The documents comparison is a subpart of the algorithm that **resolves the implicit propagations** produced by documents that appear in different communication channels.

For efficiency quality and efficiency issues, we also will perform the same calculus for **trigrams** (words of 3 tokens).

The following workflow describes how this calculus can be performed with Storm/Trident.

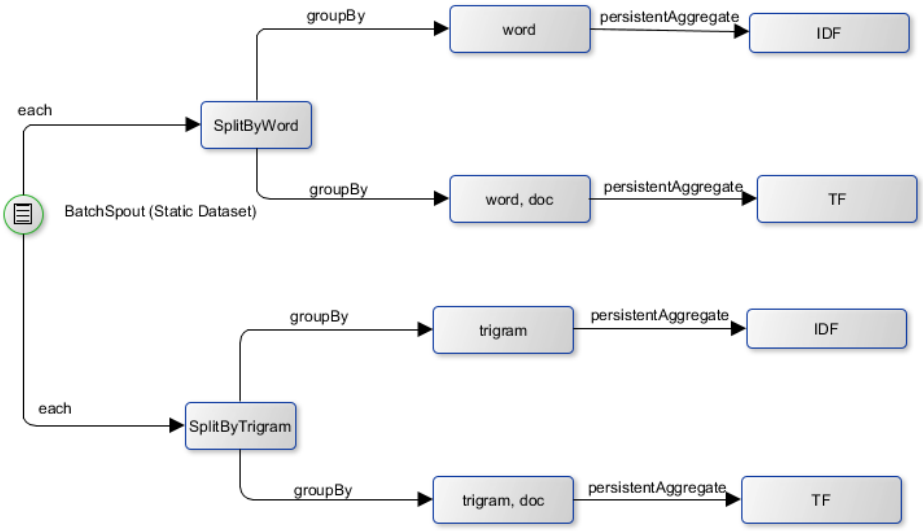

Figure 39: Document indexation

Detect influence relationships

Once propagations are calculated and all explicit relationships in a social network are stored (e.g. copies and references among documents), we create influence relationships between the authors of each pair of documents linked by any of type these relationships (propagates, copies and references) and use them to compute the communities that they produce.

Communities are detected through the influence relationships produced under just those documents that share some keyword. Finally, the computed communities are indexed by the keyword.

The following workflow describes how we can implement this process using Storm/Trident.

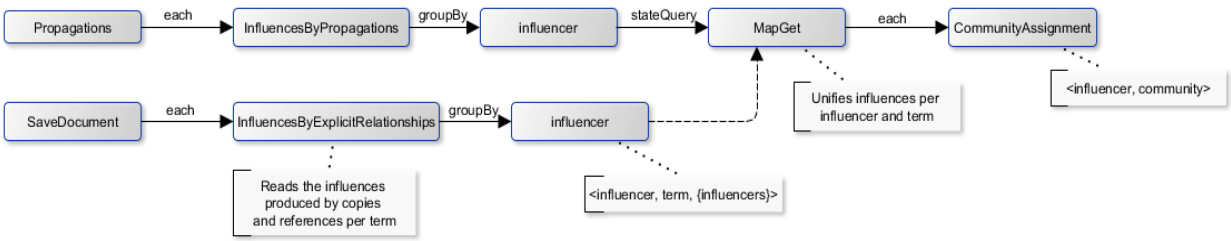

Figure 40: Detect influence relationships

5.4 Media Planning: Expected Benefits

Nowadays, we have an existing system for Acceso and Media Planning Group with similar queries that is completely build just using a graph database and a documental database.

However, the current design is not scalable and has no transactions. Also, we expect having different datastores with more appropriate structures for each scenario increase queries throughput. Also, a key component is the Holistic Transaction Management

which allows to manage the consistency among different datastores. However, according to the document 9.1, our expected performance benefits are as follows:

Main Requirements	Name	Brief Description	Acceptable metric	Target Metric
Req.01	MAXIMUM_DOCS_PER_DAY	The system must be able to load 1 million of documents in less than 12 hours.	12hs	< 12hrs
Req.02	MAXIMUM_TIME_PER_QUERY	The system must be able to solve all analytical queries in less than 1 minute.	1 minute	2 sec.

5.5 Media Planning: Common CPaaS Components

We have identified a set of components that can be shared across different use cases that have a similar architecture:

- **Trident-CloudMdsQL**: Trident provides a way to design transactional analytical processes with Storm to compute aggregates (scores). It is very common to create trident plugins to make persistent queries and aggregates inside a trident topology for a specific data store. So, we suggest creating and sharing a trident plugin to work with the CloudMdsQL engine.
- **Storm components:** Storm has a set of different types of components: bolts, spouts and aggregators. So, those components related with the document indexation and the social network streaming APIs can be shared.
- **Xwork-cloudMdsQL**: XWork has a pluggable architecture and then interceptors, actions and results can be shared across different use cases. We suggest creating and share a Xwork-cloudMdsQL plugin to inject the query engine into the actions MdSQL and to manage the transaction workflow before executing an action.

6 REAL-TIME NETWORK PERFORMANCE ANALYSIS IN A TELCO ENVIRONMENT USE CASE

6.1 Telco Network Performance Analysis: Use Case Overview

The objective of the Real-Time Network Performance Analysis in a Telco Environment use case is to detect network problems before any degradation or unavailability of services occur, by actively supervising it. However, monitoring the whole network implies analyzing big amounts of data in real-time and the current solution does not provide the required degree of scalability that can be found in cloud environments. The existing end-to-end (E2E) system, called Altaia, actively detects deterioration in a network using almost real-time KPIs and key quality indicators (KQIs), finds the cause of performance problems and the produced data is always ready to be analyzed through reports and dashboards to check the network's performance. These reports can be tailored to further analyze the network's performance by creating ad-hoc queries and accessing the data that originated the calculated indicators.

The plan is to use CPaaS to provide a rich Platform-as-a-Service that supports several data stores accessible via a uniform programming model and language. Based on an analysis of data access patterns from the use case data store layers - DBN0 is composed of non-normalized tables that are used to store the same type of data, i.e. raw data, and DBN1 is used to store only aggregated data, i.e. calculated KPIs and KQIs, using a star schema -, different data store alternatives will be accessed and the most adapted will be applied. The CPaaS based platform will have to comply with demanding delay, throughput and data volume requirements. Additionally, CPaaS will keep the needed traceability of performance bottlenecks and debugging of errors in applications.

6.2 Telco Network Performance Analysis: Architecture Level

Altaia is an end-to-end (E2E) system which actively detects deterioration in a network using almost real-time KPIs (Key Performance Indicators) and KQIs (key quality indicators), finds the cause of performance problems and the produced data is always ready to be analyzed through reports and dashboards to check the network's performance. These reports can be tailored to further analyze the network's performance by creating ad-hoc queries and accessing the data that originated the calculated indicators. It is divided in four main modules - Mediation, Correlation, Framework and Portal - and two database layers - DBN0 and DBN1 - as illustrated in Figure 41.

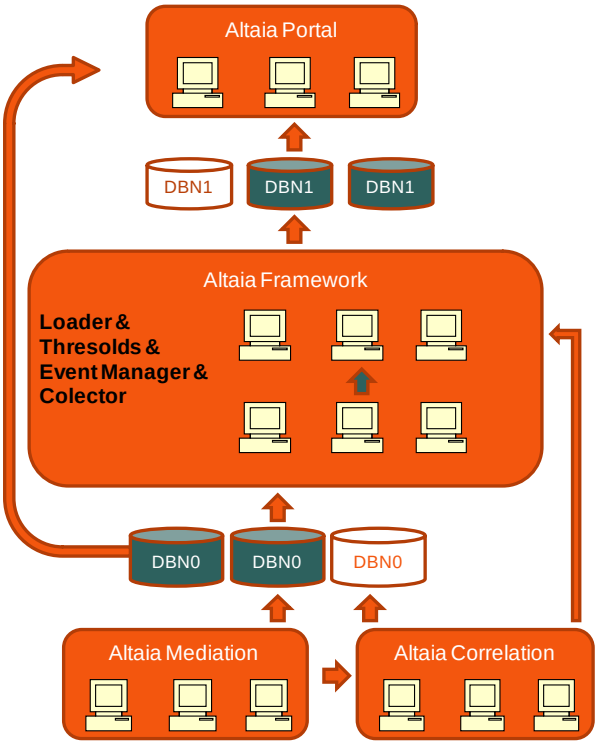

Figure 41: CoherentPaaS adapted Altaia Architecture.

- The Mediation module collects data from various sources using several plugins and can enrich the data
- Correlation module, which will be integrated in the system, will use the CoherentPaaS' CEP engine to detect call drops, collision and to calculate real-time performance indicators. The generated complex events will be fed directly to the Altaia Framework to generate alarms
- The DBN0 mainly stores raw data collected from the network and DBN1 the calculated key performance and quality indicators (KPIs and KQI) from the DBN0 data
- The Framework module is the point where all the KPIs and KQI are produced and transformed for several types of analyses. It reads the data (stored in the DBN0s and other metrics stored in the DBN1) in batch in well-defined periods of time and the produced KPIs and KQIs are then stored in batch into the DBN1, where they are accessible to the Altaia Portal module. During KPIs and KQIs calculation, this module is also responsible for evaluating pattern deviations through fixed and dynamic thresholds
- The Portal module is responsible for analyzing the key indicators from the DBN1 data stores and presents them in reports or in real-time dashboards. It also enables drill-to-detail querying of DBN0 data stores when the analyst needs to consult the data that generated the key indicators

6.3 Telco Network Performance Analysis: Design Level

6.3.1 Requirements Contextualization

We will start this section by discussing the delay, throughput and data volumes that the data stores are required to support, in order to be deployed and used in this use case.

OFrom these, we will be able to discern the adequate data stores for each database layer (DBN0 and DBN1) of the use case.

In terms of **delay**, the goal response from queries:

- Must not exceed 2 minutes to insert data in the DBN0 and DBN1 and to read data from the DBN0 to the Altaia Framework;
- Not greater than 1 second when the read operations are executed from the Altaia Portal module (or equivalent).
- Regarding the CEP engine, it must be capable of consuming and producing events every second.

Regarding satisfactory response times, they can be roughly twice the goal of the above stated delay values.

On the subject of **data volumes**, we start by defining the record sizes and the average number of records handled per hour and per day. These values can be found in Table 4 for DBN0 and Table 5 for DBN1 and are used to calculate the average daily data volume and throughput. We provide the record sizes with and without the current index overheads. As our current design of the DBN1 database uses a star schema, we provide the number of dimensions for each table as well.

Table 4: DBN0 table metadata

DBN0	# table columns	KBytes per record (TOTAL)	KBytes per record (table data)	KBytes per record (index)	records / hour (AVG)	records / day (AVG)
VOZ_2G	169	1,40	1,20	0,20	1650000	39600000
VOZ_3G	203	1,70	1,50	0,20	1380000	33120000

Table 5: DBN1 table metadata

DBN1	# table columns	# Table dimensions	# Metrics in the table	KBytes per record (TOTAL)	KBytes per record (table data)	KBytes per record (index)	records / day (AVG)
F_TRF_ACCOUNT_AG1	27	4	23	1,98	1,79	0,18	4600
F_TRF_ACCOUNT_AG2	31	8	23	0,59	0,24	0,36	1780000
F_TRF_ACCOUNT_AG3	31	8	23	0,72	0,31	0,41	3040000
F_TRF_SACCOUNT_AG1	28	5	23	0,38	0,19	0,20	123000
F_TRF_SACCOUNT_AG2	32	9	23	0,66	0,26	0,39	1900000
F_TRF_SACCOUNT_AG3	32	9	23	0,79	0,34	0,45	3100000
F_TRF_USER	41	18	23	1,28	0,56	0,72	53000000
F_TRF_USER_AG1	29	6	23	0,47	0,17	0,29	39170000
F_TRF_USER_AG4	30	7	23	0,60	0,21	0,39	39170000

From the tables above, we calculated the data volume that the data stores must be able to handle to support the use case. Note that we used the record sizes without the index overhead and we applied two different **multipliers** to the data:

- A 3 times multiplier to update the values for the customer quality management scenario;
- A 10 times multiplier to update the values for a tier 1 network operator.

Table 6 indicates the daily volume stored in the DBN0 in total and without the indexes overhead and Table 7 shows the same values for DBN1.

Table 6: DBN0 data volume with scaling multipliers

DBN0 (w/ multipliers)	Daily volume (GBytes TOTAL)	Daily volume (GBytes TABLE)
VOZ_2G	1586,15	1359,56
VOZ_3G	1610,87	1421,36
TOTAL	3197,02	2780,91

Table 7: DBN1 data volume with scaling multipliers

DBN1 (w/ multipliers)	Daily volume (GBytes TOTAL)	Daily volume (GBytes TABLE)
F_TRF_ACCOUNT_AG1	0,261	0,236
F_TRF_ACCOUNT_AG2	30,046	12,222
F_TRF_ACCOUNT_AG3	62,622	26,962
F_TRF_SACCOUNT_AG1	1,337	0,669
F_TRF_SACCOUNT_AG2	35,877	14,133
F_TRF_SACCOUNT_AG3	70,066	30,155
F_TRF_USER	1940,918	849,152
F_TRF_USER_AG1	526,711	190,513
F_TRF_USER_AG4	672,398	235,339
TOTAL	3340,237	1359,381

Considering a scenario of in-memory data store, we determined that:

- The amount of data to be stored should be the most frequently accessed;
- Aim for a high hit ratio for the executed queries from the Altaia Portal or an equivalent component.

Table 8 shows the data volumes frequently accessed in the different aggregation modes, which are available to solve performance issues of the current relational solution.

Table 8: Frequently accessed data volume

Query time range	Aggregation	Available time range	Most accessed data volume
4 hours	5 minutes	7 days	945 GB
2 days	1 hour	7 days	620 GB
2 months	1 day	1 month	1.35 TB

To calculate the **throughputs** (Table 9) that the datastores must be able to achieve, in order to handle these amounts of data volumes, we determined the daily volumes calculated for each data store layer, without the indexes overhead. We provide the insertion and the selection throughput in KB/s for each data store layer. Note that we added a 2 times multiplier for the insertion in the DBN1 data store, to account for recalculations for an already calculated time period, that might occur due to late data being inserted into the DBN0.

Table 9: Required Data throughputs

Throughput	DBN0	DBN1
Insertion (MB/s)	32	32
Selection (MB/s)	64	(300 records/sec)

Table 10 identifies data throughput requirements of the correlation module features, to be built upon the CEP CoherentPaaS engine.

Table 10: Required Data throughputs

Correlation	Delay (sec/Event)	Throughput (Events/sec)
Consumption	1	30000
Production	1	30000

6.3.2 Data Model

The data model used in the Telco network performance analysis use case is different for the DBN0 and the DBN1 database layers. The DBN0 database layer will store raw data - mediated call data records. The DBN1 database layer will store the KQIs and KPIs, calculated from the CDR data stored in the DBN0 layer.

6.3.2.1 DBN0 Data Model

The DBN0 data store layer is a single flat table that will store call record events from a telco network. These records are comprised of almost 200 attributes, spanning several data types, as illustrated in Figure 42. The data types used in DBN0, defined by the oracle data store, are: *timestamp*, *smallint*, *number*, *varchar* and *integer*.

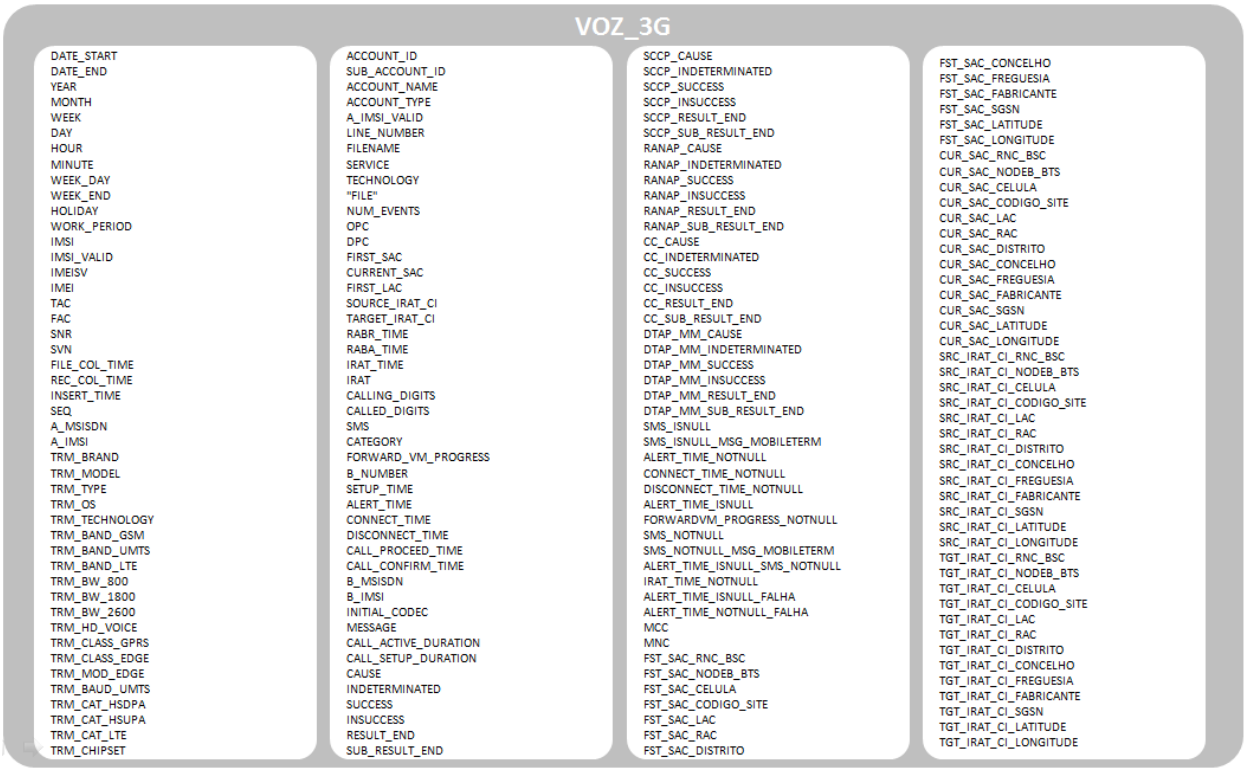

Figure 42: DNB0 table representation, which stores the raw data

As our initial approach regarding the DBN0 data store used a single flat table in an Oracle database instance, we decided to test the CoherentPaaS platform with the Derby (DQE) and the MongoDB data stores

- Distributed File Systems with added layers of functionalities, like Apache Hadoop, is an option as an underlying technology to handle the DBN0 data store layer. It allows having data stored in a cluster instead of a single machine leading to increased robustness. Data stores such as Derby (DQE) can provide these benefits;
- The MongoDB, being a document based data store, was considered as a possible scenario since the single flat table can be viewed as a document.

6.3.2.2 DBN1 data model

The DBN1 data store layer was first designed to store KPIs and KQIs, calculated from the raw data available in the DBN0 data stores, using a set of star schemas. This set of star schemas consists of a base star and spatial and temporal aggregation over the raw data. Figure 43 shows a representation of the base star.

The star schemas are used in the original deployment of the Altaia system to provide pre-calculated values for queries from end users, in order to decrease the average wait time response. Furthermore, it is the Altaia Framework that generates the DBN1 database tables, from the configuration information, so the star schemas are created automatically.

The star schemas are not mandatory. The DBN1 table creation process can be modified, depending on the datastore's requirements, but the performance requirements of the use case must always be satisfied.

In the DBN1 databases, we can usually find some additional information about the metrics stored and the required metrics, therefore the data types used are essentially *number* and *varchar*.

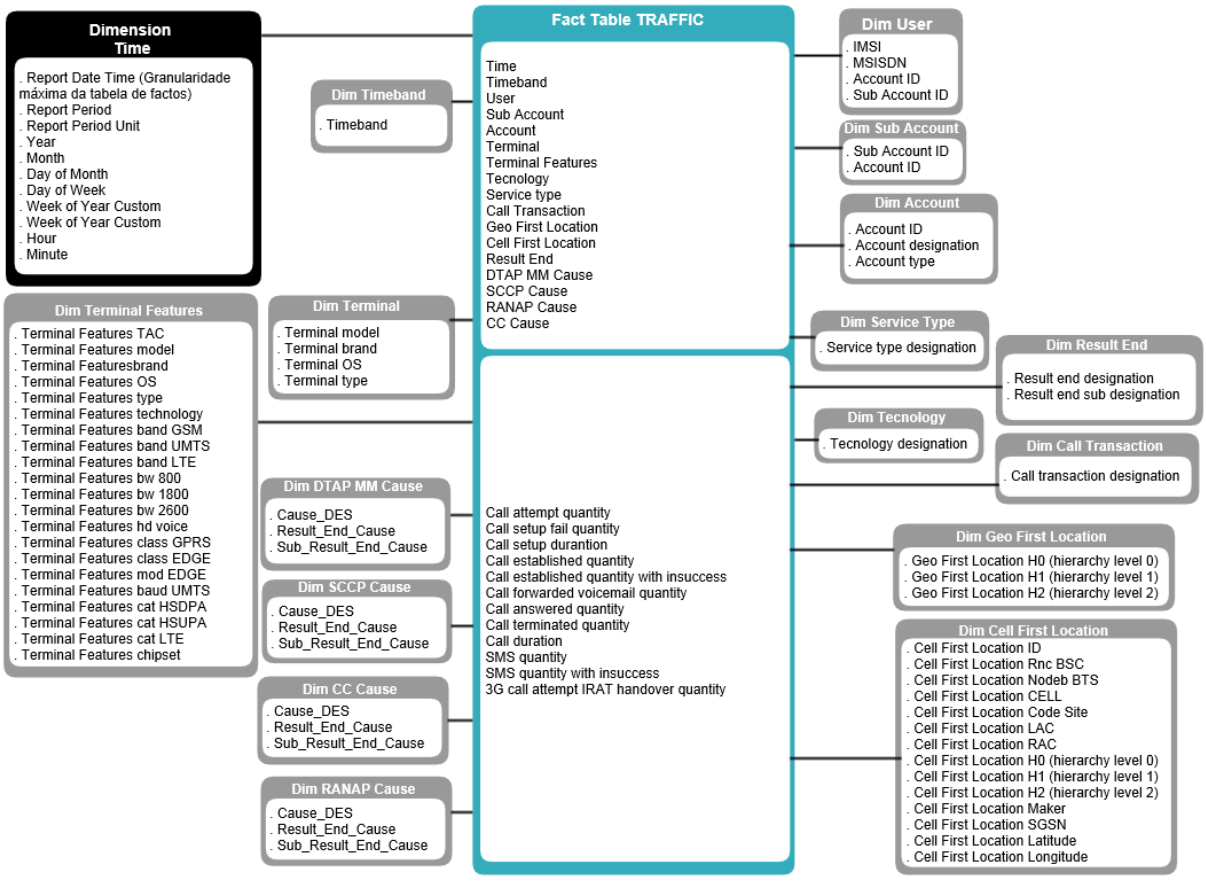

Figure 43: Logical ER diagram for the DBN1 data store layer

Columnar data stores, such as MonetDB or Derby (HBase), are a possibility for the DBN1 data store layer, due to the usage of star schemas, which can lead to better compression, and the low delay imposed when selecting information but not while inserting. In-Memory columnar data stores such as ActivePivot will be evaluated, but should take into account the huge amount of data that is required to store in memory, in order to achieve a high query hit ratio.

6.3.3 Functional Components

6.3.3.1 Framework

The Framework module is the point where all the KPIs and KQI are produced and transformed, enabling the user several types of analyses. It reads the data (stored in the DBN0s and other metrics stored in the DBN1) in batch in well-defined periods of time and the produced KPIs and KQIs are then stored in batch into the DBN1, where they are accessible to the Altaia Portal module. During KPIs and KQIs calculation, this module is also responsible for evaluating pattern deviations through fixed and dynamic thresholds.

Figure 44 shows a simplified architecture of the Framework module, where the focus is on the components that interact with either DBN0 or DBN1. This includes the DBN0.XML, the DBN1.XML, the DBN1Loader, the Collector and the Configuration component. The Filter Manager and the Threshold and Inventory systems do not interact directly with either database layers.

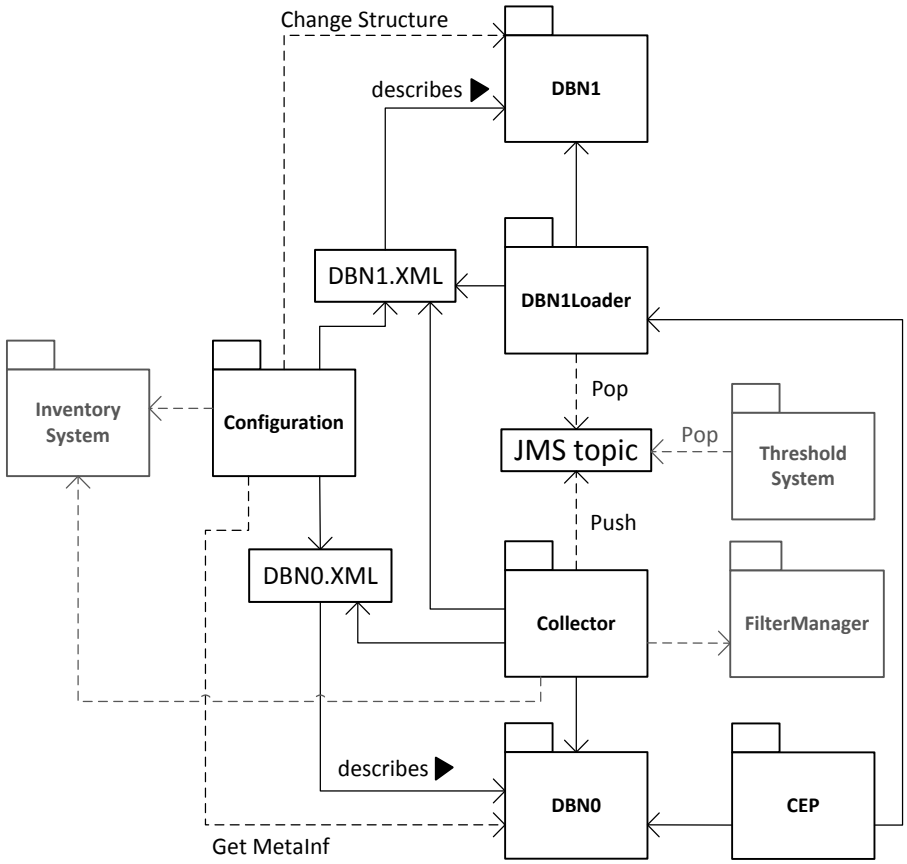

Figure 44: Altaia Framework summarized architecture

6.3.3.2 Configuration

The Configuration component allows changing the structure of the DBN1 and keeps the configuration mechanisms used by other components. It maintains the DBN0.XML file, which maps the types of entities and tables with the ones defined in the inventory, as well as the DBN1.XML file which is used by the other components to find the star where the calculated metrics must be stored.

6.3.3.3 Inventory System

This module is responsible for maintaining the system record, i.e. the definition of the equipment hierarchies and the metric/function's definitions. This module is instantiated on each applicational node to grant fault tolerance to the system.

6.3.3.4 Collector

The Collector component is responsible for collecting data from the DBN0 datastores periodically and, when a new metric is defined in the Inventory, a new collection task is created. It also calculates the key indicators and pushes them into a JMS topic (a messaging service).

It is distributed with two distinct techpacks:

- coltechpack-db – the techpack responsible for synchronizing the creation of metrics in the Inventory module and for collecting data from a DBN0 data store.

- coltechpack-db-recovery - the techpack responsible for recovering old data that may appear out of order in a DBN0 data store. It works in strict collaboration with the coltechpack-db.

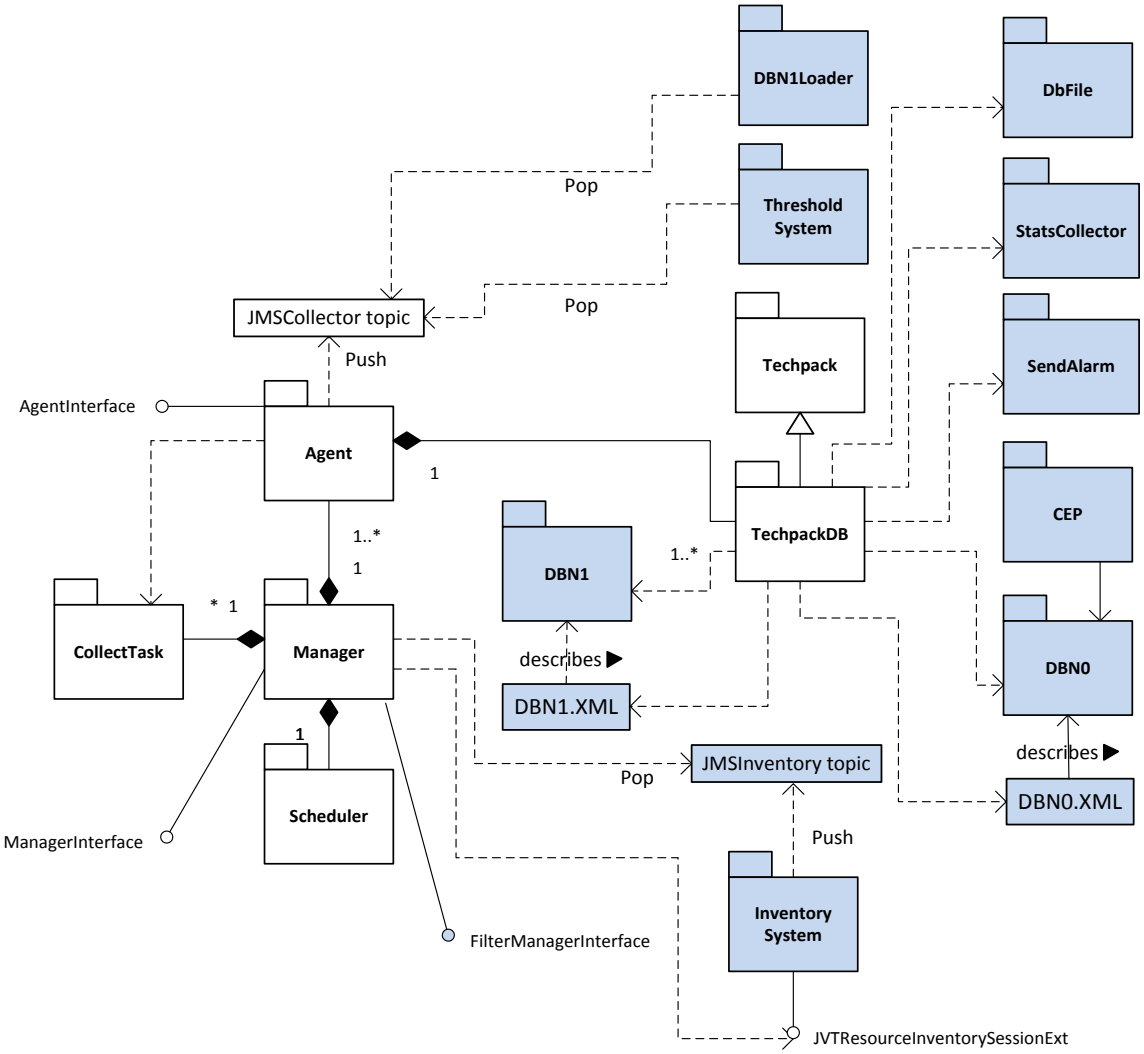

Figure 45: Collector architecture

- We can see in Figure 45 the components that exist in the Collector (in white) and the external components that they interact with (in blue).
- The Collector has an Agent and several CollectTasks which are used by the Agent to calculate and push the metrics into the JMS Topic for further processing. The data required to calculate the metrics are obtained through the TechpackDB component.

The following list of functions and the sample query (executed by the Collector in DBN0) are meant to convey the functions and features that are more frequently used by the Altaia system.

If some of these functions and features are not available in CloudMdSQL or in the data stores to be deployed, then we will have to implement them at a higher level.

Aggregation analytic functions (Oracle based):

- FIRST_VALUE -> DENSE_RANK () KEEP FIRST
  - OVER (PARTITION BY … ORDER BY …)
- RATIO_TO_REPORT
- MEDIAN, PERCENTILE_CONT, PERCENTILE_DISC

- WITHIN GROUP (ORDER BY …)

Select all events

```
SELECT
          DATE_START as DATE_START,
          DATE_END as DATE_END,
          YEAR as YEAR,
          MONTH as MONTH,
          WEEK as WEEK,
          DAY as DAY,
          HOUR as HOUR,
          MINUTE as MINUTE,
          ...
          TGT_IRAT_CI_SGSN as TGT_IRAT_CI_SGSN,
          TGT_IRAT_CI_LATITUDE as TGT_IRAT_CI_LATITUDE,
          TGT_IRAT_CI_LONGITUDE as TGT_IRAT_CI_LONGITUDE
FROM v_voz_3g dbn03g
WHERE dbn03g.DATE_END BETWEEN TO_DATE
(${P_DATE_INTERVAL_BEGIN},'YYYYMMDDHH24MI')
AND TO_DATE (${P_DATE_INTERVAL_END},'YYYYMMDDHH24MI')
AND ACCOUNT_NAME = ${P_ACCOUNT_EXPRESSION}
AND SUB_ACCOUNT_ID = ${P_SACCOUNT_EXPRESSION}
UNION ALL
  SELECT
          DATE_START as DATE_START,
          DATE_END as DATE_END,
          YEAR as YEAR,
          MONTH as MONTH,
          WEEK as WEEK,
          DAY as DAY,
          HOUR as HOUR,
          MINUTE as MINUTE,
          ...
          CUR_CI_SGSN as CUR_CI_SGSN,
          CUR_CI_LATITUDE as CUR_CI_LATITUDE,
          CUR_CI_LONGITUDE as CUR_CI_LONGITUDE
FROM v_voz_2g dbn02g
WHERE dbn02g.DATE_END BETWEEN TO_DATE
(${P_DATE_INTERVAL_BEGIN},'YYYYMMDDHH24MI')
AND TO_DATE (${P_DATE_INTERVAL_END},'YYYYMMDDHH24MI')
AND ACCOUNT_NAME = ${P_ACCOUNT_EXPRESSION}
AND SUB_ACCOUNT_ID = ${P_SACCOUNT_EXPRESSION}
```

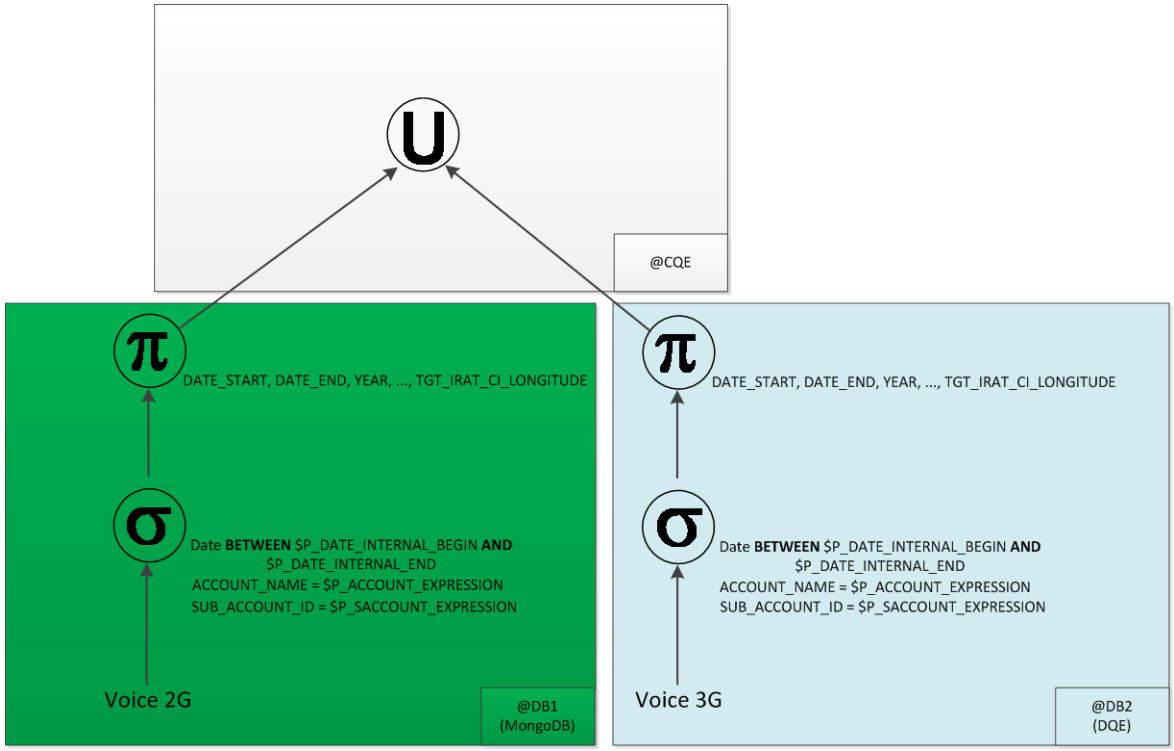

Figure 46: Select all events query decomposition

The above query selects all events that occurred involving 2G or 3G. This query can be adapted to the CloudMdSQL query language and request data from several data sources. In our example, we would be querying two data stores - like MongoDB and Derby (DQE)-, one with the 3G data and the other with the 2G data. Depending on the cost/benefit tradeoff and the possible cost of implementing non-supported native functions, we might opt for only one of those data stores.

Other queries are used to get all SMS metadata or all the calls that failed for some reason. Both these cases only require different selected columns and some changes in the where clause - since both access the 2G and 3G tables - so the same request to different data sources can still be performed.

6.3.3.5 DBN1Loader

The DBN1Loader component retrieves the calculated indicators from the JMS topic and, using the information in the DBN1.XML, stores them into the DBN1.

This component will be modified so that the CoherentPaaS common query engine (CloudMdSQL) can be supported.

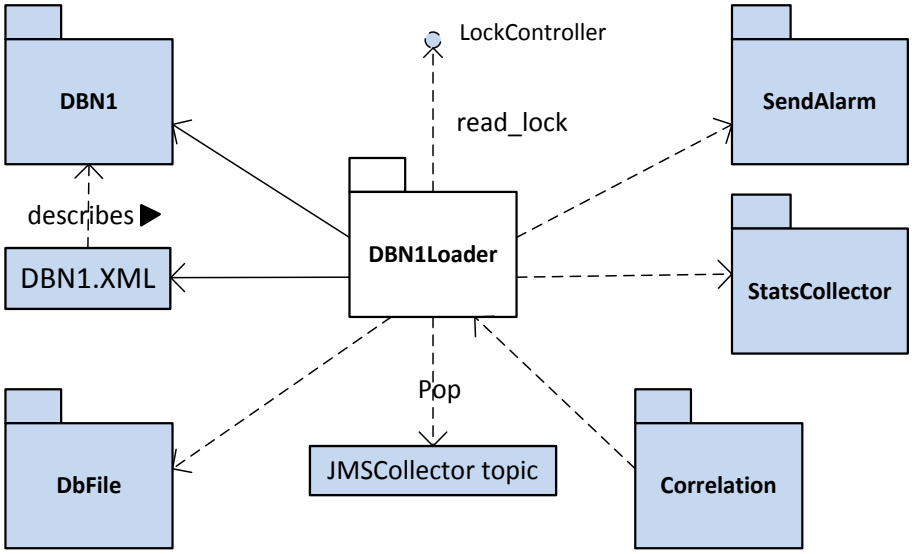

Figure 47: DBN1Loader's architecture.

Figure 47 describes the components that exist in the DBN1Loader (in white) and the external components that they interact with (in blue).

This is a single component that collects the calculated metrics from the JMS Topic and performs several tasks, such as checking for possible alarms to be generated. Then, using the DBN1.XML that describes the datastore, it stores the indicators in the DBN1.

The following sample query to the DBN1 database performs the composition of the account information for a whole month with granularity of a day. These queries are written for an Oracle database instance. Once more, this sample query is meant to convey the functions and features that are more frequently used by the Altaia system. If some of these functions and features are not available in CloudMdSQL or in the data stores to be deployed, then we will have to implement them at a higher level.

Select account (one month aggregated by day)

```
SELECT *
FROM   (SELECT Max(TU.user_account_id),
               SUM(TU.nusers),
               …
    FROM (SELECT Count(1), user_account_idCount, …                    } A
          FROM   da_trf_cuser
          WHERE EXISTS (SELECT …)
          GROUP  BY user_account_id, user_saccount_id),
         (SELECT Round(Decode(SUM(call_established_qty), …))          }
          FROM   vagg_dd_f_trf_account_ag1 f                          } B
          WHERE  report_date_time BETWEEN                             }
           To_date ('201402010000', 'YYYYMMDDHH24MI') AND             }
           To_date ('201402282359', 'YYYYMMDDHH24MI') AND EXISTS (…)))
```

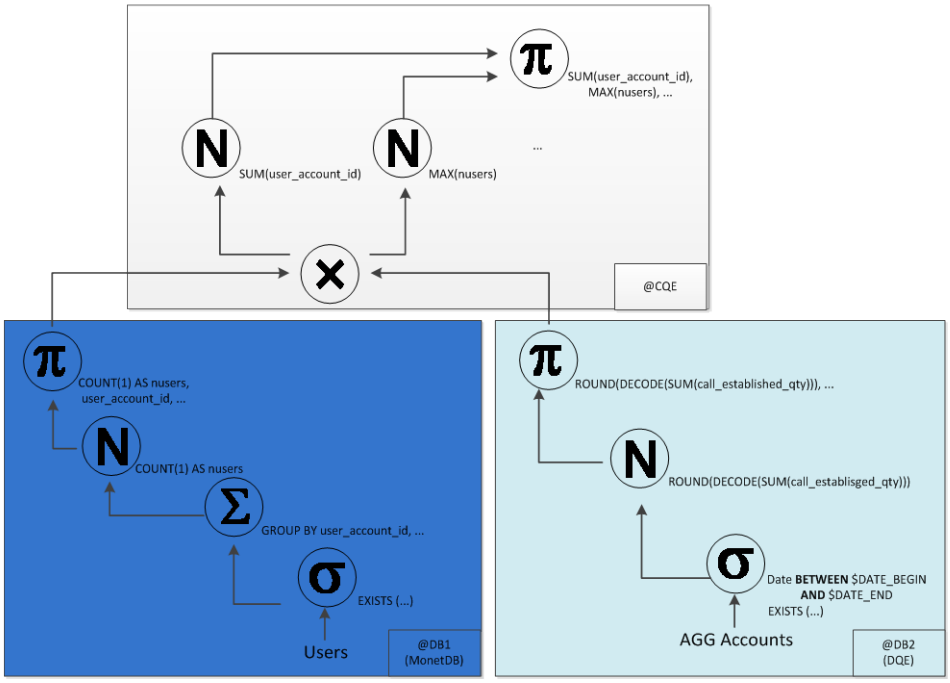

Figure 48: Select account query decomposition

The previous query (simplified for analysis purposes), selects some data related to client accounts, aggregated by day. When adapting this query to use the CloudMdSQL query language, we will use sub queries A and B to different data stores. In this case, the data store queried by the subquery A would store the user's data and the data store queried by the subquery B would store some pre aggregated account data. Once again, and depending on the cost/benefit trade-off and the possible cost of implementing non-supported native functions, we might opt for only one of those data stores.

Other queries can be built for:

- Different data, like the MSISDN instead of the account;
- Different aggregation parameters, like only 4 days of data aggregated by day or 4 hours of data aggregated by 5 minutes.

These queries would also be easily adapted to request data from multiple data sources, as they follow the same semantic and syntactical construction.

6.3.3.6 Correlation

The Correlation module uses data from two different data sources, as illustrated in Figure 41: the Mediation module and the data stores in the DBN0 layer.

The Mediation module receives the data from various heterogeneous sources, such as routes and servers and this data is collected using many different protocols, such as SNMP or FTP. It can also be enriched in the Mediation module before processing.

The data stores in the DBN0 layer will be used as a data source for the Correlation. This allows the reuse of correlated data with new data from Mediation, in order to perform operations such as call assembly.

The implemented Correlation module will provide additional features – namely call drop detection, call collision detection and real-time KPI threshold violation monitoring -, replacing parts of the alarm logic of Framework, and thus enabling the alarms to be fired earlier. The alarm events are then delivered to the DBN1Loader, which will process them.

The Correlation module will be built upon CoherentPaaS CEP system.

6.3.3.7 Portal

Regarding the Portal, which enables data querying on DBN1 and DBN0 data stores, it will be composed of a set of typical queries that will be executed to obtain the data.

The executed queries are generated by the Altaia Portal, depending on the options set by the user. To circumvent modifications at the Portal level we will simulate the Portal by directly injecting the most frequent types of queries into the data stores.

The queries executed to the DBN1 data stores will have the following characteristics:

- The data retrieved will be sparse when compared to the total amount of data stored.
- There will be aggregation operators, such as:
  - Sums/Count [distinct];
  - Averages/Median;
  - Max/Min.
  - First/Last
  - Percentile(cont/disc)
- Ordering of data and filtering by time range.

When executing queries to the DBN0 (in a drill-to-detail context) the queries will have similar characteristics. However, they can also contain sub-queries and joins.

The queries for both layers include operators such as +, -, /, *, if, > , >=, ,<,<=, =, <>, and, or, not, is null, like and between. The aggregations might use functions such as:

- FIRST_VALUE -> DENSE_RANK () KEEP FIRST
  - OVER (PARTITION BY … ORDER BY …)
- RATIO_TO_REPORT
- MEDIAN, PERCENTILE_CONT, PERCENTILE_DISC
  - WITHIN GROUP (ORDER BY …)

Once again, if some of these functions and features are not available in CloudMdSQL or in the data stores to be deployed, then we will have to implement them at a higher level.

6.4 Telco Network Performance Analysis: Expected Benefits

Most of the expected benefits will emerge from the transparent usage of better adapted data stores for the big data scenario that this use case will have to face. Previous experiments using Impala over Hadoop showed some improvements regarding the use case requirements. Hence, we expect that using CoherentPaaS we will be able to use the best suited data store for the data being stored, while maintaining the Altaia system relatively untouched.

Data stores and associated expected benefits are ranked below in respect to their importance in the use case:

- **Distributed File Systems and Document Stores:** As our initial approach regarding the DBN0 data store used a single flat table in an Oracle database instance, we

decided to test the CoherentPaaS platform with the Derby (DQE) and the MongoDB data stores
    - Distributed File Systems with added layers of functionalities, like Apache Hadoop, is an option as an underlying technology to handle the DBN0 data store layer. It allows having data stored in a cluster instead of a single machine leading to increased robustness. Data stores such as Derby (DQE) can provide these benefits
    - The MongoDB, being a document based data store, was considered as a possible scenario since the single flat table can be viewed as a document. The flexibility through CPaaS to change a data store with another data store is also among the expected benefits.
- **Columnar data stores**, such as MonetDB or Derby (DQE) - HBase based-, are a possibility for the DBN1 data store layer, due to the usage of star schemas, which can lead to better compression, and the low delay imposed when selecting information but not while inserting. In-Memory columnar data stores such as ActivePivot will be evaluated but should be take into account the huge amount of data that was required to store in memory, in order to achieve a high query hit ration;
- Key-Value data stores, such as Eutropia, are well designed for the drill-to-detail queries to the DBN0 data store layer due to its specific data access pattern, i.e. getting single values using a key. We won't use them, however, because that would require having the data replicated, meaning terabytes of additional space.
- Through the usage of the CoherentPaaS CEP system, we might expect to reduce framework processing (alarming) and anticipate real-time alarms, while benefiting from the ease of usage provided by its SQL-like language and associated database operators.

6.5 Telco Network Performance Analysis: Common CPaaS Components

No specific components have been identified to meet the proposed criteria for Common CPaaS Components. However, the union operation which is described in Section 4.5 is also used in this use case, as illustrated in Figure 46.

7 BIBLIOGRAPHIC SEARCH USE CASE

7.1 Bibliographic Search: Use Case Overview

Bibliographic databases collect the researchers' knowledge and are important resources to find experts and institutions working in specific research areas. One of these bibliographic databases is DBLP, which contains all papers about computer science; CORDIS which contains all European projects; or SCOPUS, which is one of the biggest bibliographic databases.

Mainly, bibliographic databases are used by researchers to know the state of the art of a given topic. However, they can be used for many other purposes such as looking for reviewers, collaboration, similar papers or works and so on.

Sparsity technologies and INRIA want to go one step forward with Sciencea, a bibliographic web application, trying to merge the contents of CORDIS and DBLP to offer new analytical queries such as: which are the projects probably related with a given article or who is a good reviewer for a given European project.

These are the main functionalities provided by the use case:

- Calculate the best reviewers for a given European project.
- Calculate the most related European projects for a given article.
- The system must allow adding new reviewers (without documents) with a set of keywords to define their specialization.
- Load projects and papers periodically

7.2 Bibliographic Search: Architecture Level

7.2.1 Architecture overview

The architecture is the same as the one for the Media Planning use case, because despite having a different domain, the application workflow is the same and therefore, we will use a **Lambda Architecture**.

The application runs with a data-store that is immutable because it does not have delete or update operations, allowing only create and read data. It works in real time and calculates query results on top of an incoming stream of data. Results, once computed, should be stored in such a manner that they can be queried by applications.

The main software components involved in the loading process are **Trident** and **Storm** because these offer a framework to have a scalable loading architecture with real time queries over the processed data.

Storm is a streaming library that handles transport, messaging and process supervision. Storm has two main components for a given dataflow (topology): spouts, which are input channels of data; and bolts, which processes information. Trident is a component that works on top of Storm with a similar API to Pig / Cascading to apply real time queries that merges the current input data with the existent information in the datastores (DBs or *Memcached*). Trident is especially useful to design stateful analytical processes.

The following snippet shows an example of the word count algorithm using Trident. Trident contains a set of useful methods, such as *each* or *groupBy,* to design easily a topology. In the following example, for each document, the system groups them by words and then computes the total of them using incremental techniques. So, any application is able to report in real time the number of documents without recounting the previous ones every time.

```
TridentTopology topology = new TridentTopology();
TridentState wordCounts =
     topology.newStream("spout1", spout)
       .each(new Fields("sentence"), new Split(), new Fields("word"))
       .groupBy(new Fields("word"))
       .persistentAggregate(new MemoryMapState.Factory(), new Count(), new Fields("count"))
       .parallelismHint(6);
```

Figure 49: An example of the word count algorithm using Trident

7.2.2 Deduplications

The loading process of the bibliographic use case is very complex because there are a lot of entities that needs to be deduplicated. These entities are projects, institutions, cities, countries and regions. Furthermore, other entities need to be indexed and scored by terms/keywords that appear in the projects description.

The deduplication process of any entity type implies a common set of processes:

- **Normalization**: To improve the matching probability.
- **Block providers**: To produce disjoint blocks by different approaches and compare just those registers that appear in the same block. Notice that this algorithm avoids a quadratic evaluation of the registers.
- **Comparison method**: Usually, approximate comparisons are performed. E.g. Levenshtein distance.
- **Emit the unique registers**: The system must implement a valid merge approach of those registers that represent the same entity. The result of all unique registers is produced by this component.

The following subsection shows an example of deduplication process designed with *Storm* and *Trident*.

7.2.2.1 Example: Institutions deduplication

The process for deduplicating CORDIS institutions starts when a new set of projects appears. Then, for every project, the system extracts the following fields to filter and group the input data: project identifier (docId), institution name, initials and, acronym. The rest of the related information for a given institution can be consumed as a complex object later. This task will be performed by a *Storm* bolt called *InstitutionFieldExtractor*.

The second step of the process consists of the creation of the block providers by a *Trident group-by* operators. The system groups registers by *initials* and *acronyms* and it proceeds to append the existing data of the database in the previously created groups by initials and acronyms. This storm process is easily configured with a trident *stateQuery* whose output includes a new field called *blockId* to identify the block where a given register belongs to.

Once all groups (blocks of registers) are created, the system unifies all of them in a single set because one register belongs to a single block. This process is performed by means of the *SharedBlocksQuery* component. The next process identifies those registers that really represent the same entity with the comparison method. This part is processed by a component called *ResolveInstitutionsId* which additionally will insert the new institutions in the database and sends to the next process the identifiers of all processed institutions (referenced as *institutionId*), its name and the project identifier where they appear.

The produced output is used to resolve the relationships among projects and institutions before adding a new European project to the database.

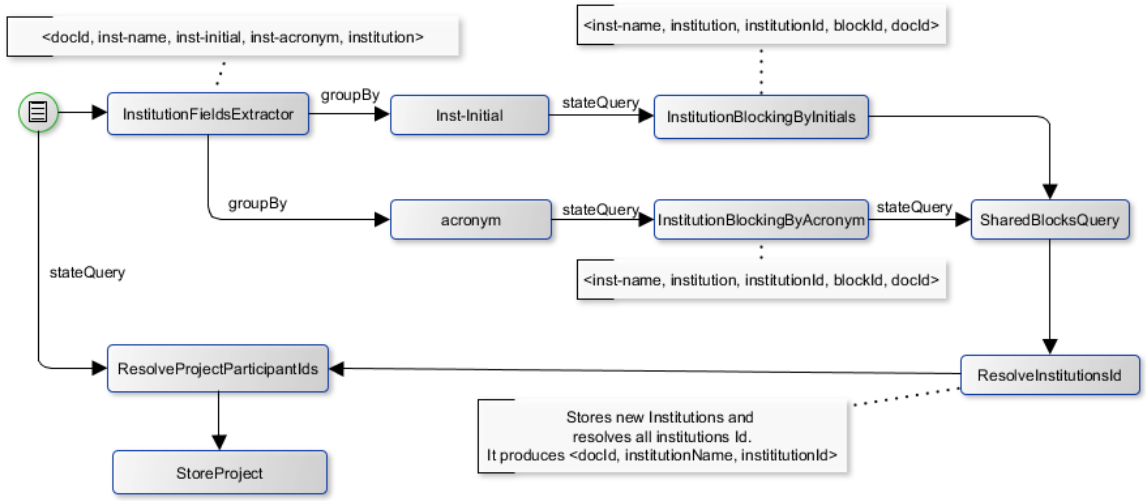

Figure 50: Relationships among Projects and Institutions

7.2.3 Documents indexation

Documents (European projects and papers) need to be quickly retrieved by any of the relevant words included in its description/abstract. Also, other entities (such as programmes, calls and categories) need to be indexed by all words that appear in the projects (e.g. to know the word relevance of all projects of a given call).

Documents indexation is performed with *Trident* and *Storm* and this section shows the detail of the involved components in this process.

Firstly, for each document, the description (abstract) is splitted by words by the component *SplitByWord*. For each document, the output is a set of pairs - word and document. Then, this output is grouped by just words to compute the global relevance of a word in the database (*IDF – Inverse Document Frequency*). Also, the output produced by the *SplitByWord* component is grouped by the pairs of word and document to compute the local relevance of a given word for one document (*TF – Term frequency*).

The IDF and TF calculus are aggregates that can be stored in the database updating the existing values. It is easily performed using the *Trident persistent aggregates*.

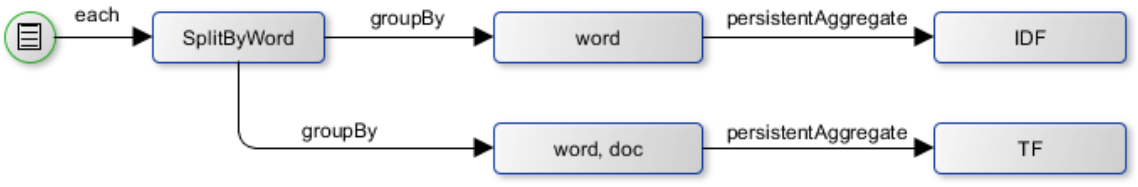

Figure 51: Documents indexation

The rest of indexed entities are not used explicitly by the provided functionality of this use case. However, it could be implemented changing the TF part, grouping them by its values instead of by document.

7.2.4 Scoring

Authors and institutions need to be scored by the words that appear in the document. So, new documents and keywords are grouped by authors, and taking into account the existing documents of the same author, the system computes the score function which evaluates the relevance of an author for a given word.

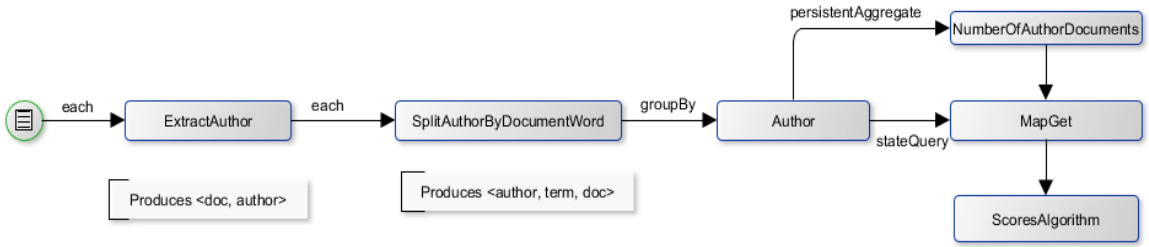

Figure 52: Scoring

7.3 Bibliographic Search: Design Level

This section describes the conceptual schema of this use case. The conceptual schema is the required information to build an information system. It has two parts: the structural schema (i.e. *data model*), which describes the information that need to be stored; and the behavioral schema (i.e. *functional components*), which describes the events/use cases that the system is able to solve.

7.3.1 Data Model

The data model of the bibliographic use case has two conceptual units: the CORDIS part, which stores all the related information about European projects; and the academic part, which stores information about published papers (e.g. from DBLP).

7.3.1.1 CORDIS data model

CORDIS stores information about European projects, which can be classified by a given category or by a call program. A European project has different participants, and one of them is the coordinator and usually, participants are companies or research institutions (e.g. universities).

For this use case, the system needs to know who is expert about a set of keywords. This expertise does not appear explicitly in the raw data and it is inferred and stored for performance issues. Therefore, the project description is tokenized and the relevance of a word for a given project is stored in a relationship called *APPEARS*. The relevance of an institution for a given keyword is inferred using the *APPEARS* relationship and stored in *SCORES* relationships.

The current version of the system needs evaluates the expertise of the most expert coordinators separately from the rest of participants and thus, we have a specific *SCORE* relationship for each role.

The following diagram represents the described concepts:

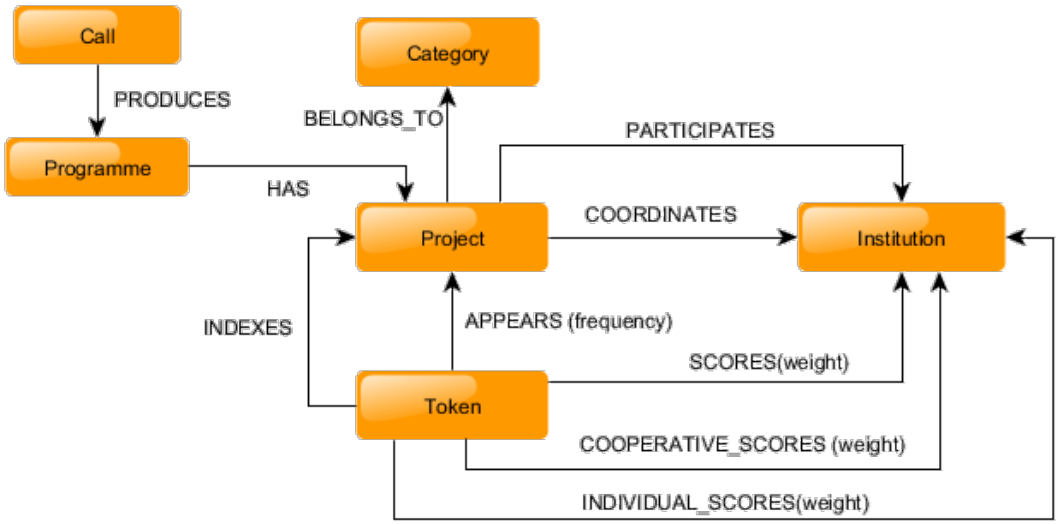

Figure 53: CORDIS Data Model

To persist the data related to these concepts we suggest using different datastores to reduce the number of I/O operations, as listed below and ranked according to their importance in the use case.

- **Documental**: To persist all data which could have a flexible size (e.g. text).
  - Project (id, description)
- **Graph**: To persist all relationships used for filtering purposes (e.g. to improve future updates efficiently avoiding scans and multiple queries).
  - PARTICIPATES, APPEARS.
- **Key Value**: To persist list of complex items (more than one basic field) related with a given key that need to be analyzed in the same time during the query.
  - SCORES: token as a key and Institution with is weight as a value.
  - INDEXES: project id as a key and value the 3 most relevant word of a project.
- **RDBMS**: To persist all basic data required to return as part of the projection of a given query.
  - Project (id, acronym, title, startdate, enddate, cost, funding, programmeAcronym, url, coordinatorCountry, year, cooperative, lastUpdate)
  - Institution (code, name, contactInformation, country, region, city, url, address)
  - COORDINATES (projectId, institutionId).
  - PARTICIPATES (projectId, participantId).

7.3.1.2 Academic data model

In the current version of Sciencea, the CORDIS information is integrated with an academic database by means of the common keywords. However, they manage different conceptual parts. Specifically, the academic database manages documents, its authors and these authors belong to a set of institutions. Moreover, documents can make references to another document to justify their contribution.

For this conceptual part, the system also needs to store who is expert on a given topic and therefore, the *SCORE* relationship also appears between term and person.

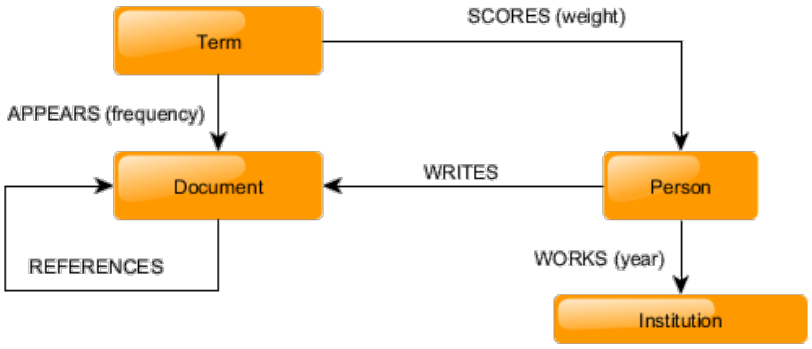

Figure 54: Academic data model

To persist the data related to these concepts we suggest using different datastores to reduce the number of I/O operations, as listed below and ranked according to their importance in the use case.

- **Documental**: To persist all data which could have a flexible size (e.g. text).
  - Document (id, description)
- **Graph**: To persist all relationships used for filtering purposes (e.g. to improve future updates efficiently avoiding scans and multiple queries).
  - WORKS, WRITES.
- **Key Value**: To persist list of complex items (more than one basic field) related with a given key that need to be analyzed in the same time during the query.
  - SCORES: term as a key and Person with is weight as a value.
  - APPEARS (prev. referenced in the documentation as RELEVANT_WORDS): document.id as a key and the set its words sorted by weight as a value.
- **RDBMS**: To persist all basic data required to return as part of the projection of a given query.
  - Document (id, title, state, date, url, pages, firstPage, lastPage, isbn, number, volume, booktitle, publisher, ee, updateDate, school, editor, address, cite, chapter, docType)
  - Person (code, name, contactInformation, country, language)
  - LAST_PAPER (docId, personCode). //INFERRED
  - LAST_AFFILIATION (personCode, InstitutionName). //INFERRED

The REFERENCES relationship is not used from the selected queries of the bibliographic use case, but it is currently used by other Sciencea queries. Therefore, it could be omitted from a practical point of view until the use case will be extended to cover all Sciencea queries.

7.3.2 Functional Components

The main functionalities provided by this use case are:

- Calculate the best reviewers for a given European project.
- Calculate the most related European projects with a given article.
- The system must allow adding new reviewers (without documents) with a set of keywords to define their specialization.
- Load projects and papers periodically.

All functionalities use different datastores simultaneously. So, having CloudMdsQL simplifies the code of integrating result sets of different datastores. The following subsections describe the formal definition of each functionality and the involved part of the schema of each datastore.

7.3.2.1 Calculate the best reviewers for a given European project (QUERY).

Definition: The goal of this query is to return the 10 most scored people in the project expertise area with have not been working in any of the participant institutions. The result must be sorted by the score. For each person, the system shows:

- His / Her current affiliation.
- His / Her last paper.
- His / Her basic fields: code, name, contact information and number of publications.

Queries types:

- SCORES (key value),
- WRITES, WORKS, PARTICIPATES (graph)
- Person, LAST_PAPER, LAST_AFFILIATION and Document (RDBMS and documental).

The execution plan of this query should compute the same result as the following one.

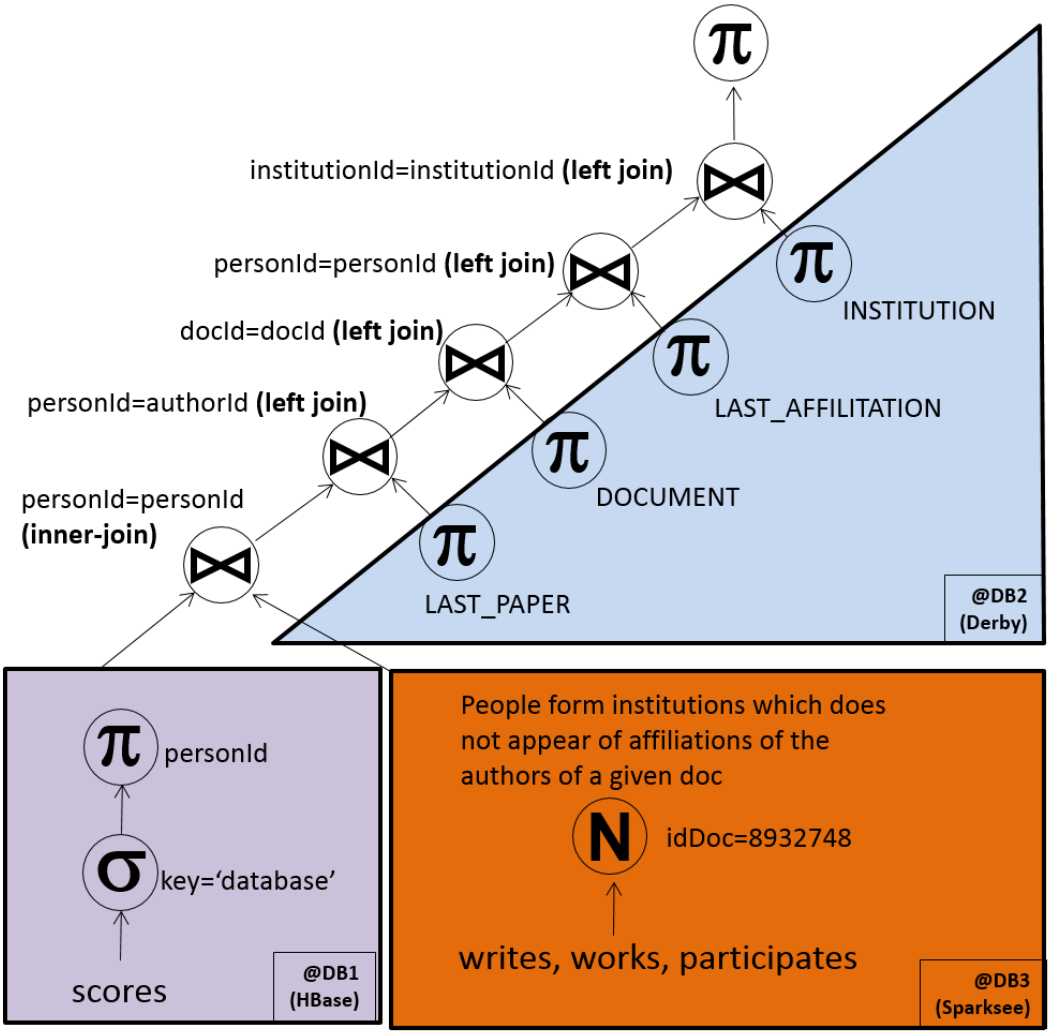

Figure 55: The best reviewers for a given project query decomposition

7.3.2.2 Calculate the most related European projects with a given article (QUERY).

Definition: This query returns the active European projects in a given paper publication date, which share some participants with the affiliations of a given paper and share some of the 3 more relevant words of the paper. For each European project, the system returns:

- The project basic information.
- The most relevant keywords of the project
- The project coordinator and the other participants.

Queries types:

INDEXES (key value), **Definition**: This query returns the active European projects in a given paper publication date, which share some participants with the affiliations of a given paper and share some of the 3 more relevant words of the paper. For each European project, the system returns:

- The project basic information.
- The most relevant keywords of the project
- The project coordinator and the other participants.

Queries types:

- RELEVANT_WORDS, INDEXES (key value),
- WRITES, WORKS relationships (graph).
- Project (RDBMS and documental).

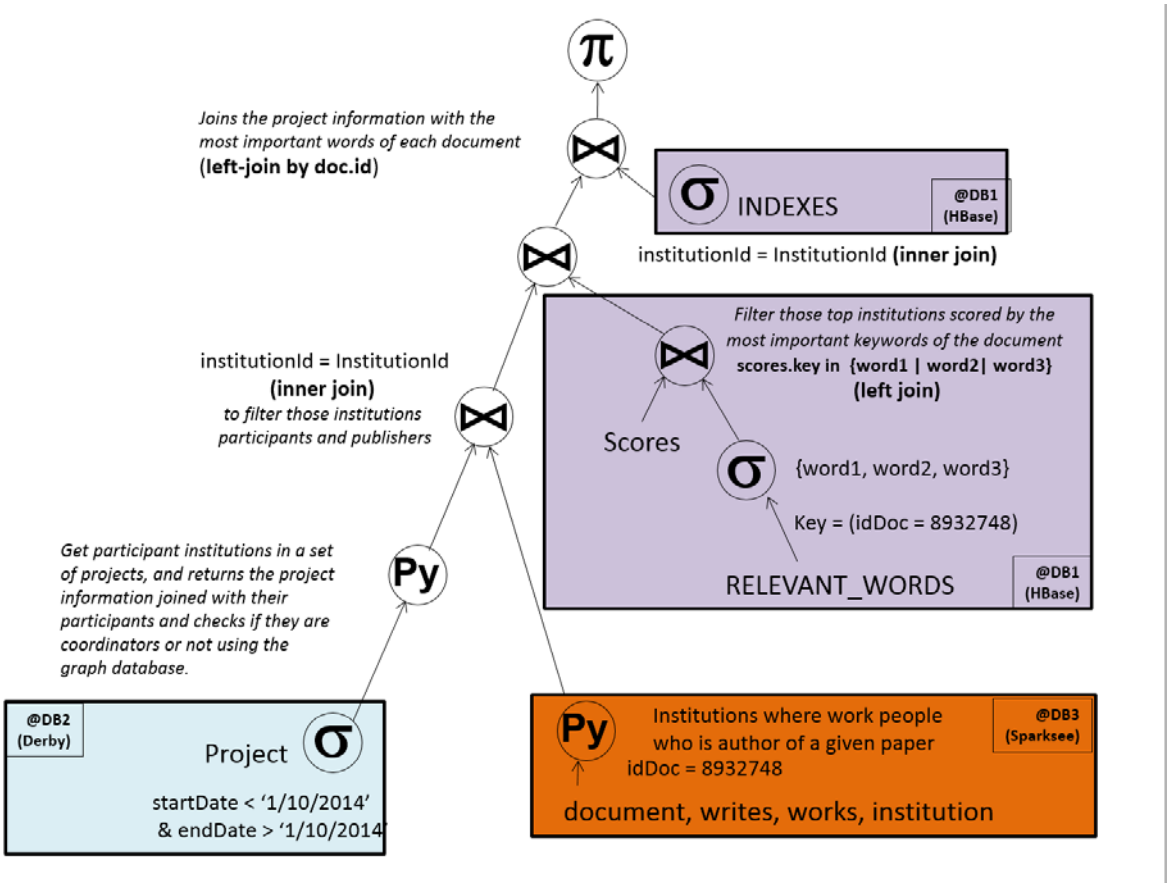

Figure 56: The most related European projects with a given article query decomposition

7.3.2.3 Add new reviewers (UPDATE)

Definition: This operation adds a new person with all the institutions where he/she previously has worked and a set of expertise areas (keywords).

Queries types:

- Person, Institution (RDBMS),
- SCORES and WORKS (graph).

7.3.2.4 Loading process (UPDATE)

The loading process consists of performing the insertions of new papers or EU projects in all datastores. Specifically, it implies lots of deduplications and indexation procedures to assure fast queries over the data.

Specifically, for the European projects loading process, the system needs to deduplicate projects, cities, countries and regions; load the raw data and index projects, institutions and other entity types around the project by keyword. All datastores are affected by this process through the CloudMdsQL.

The loading process of the pure bibliographic part is exactly the same excluding the deduplication process.

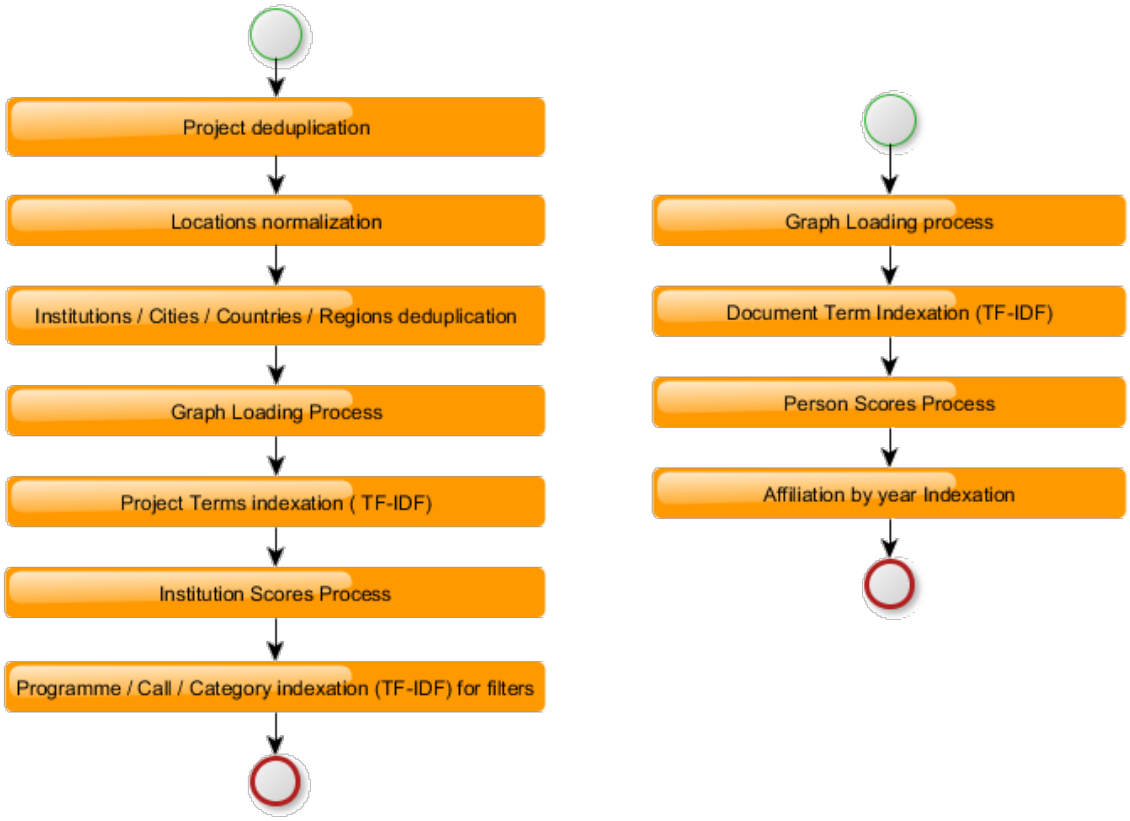

Figure 57: Bibliographic part Loading Process

7.4 Bibliographic Search: Expected Benefits

Nowadays, UPC has an existing system called Sciencea with similar queries that is completely build just using a graph database.

However, the current design is not scalable and has no transactions. Also, we expect having different datastores with more appropriate structures for each scenario increase the performance of the number of queries per second. Also, a key component is the Holistic Transaction Management which allows to manage the consistency among different datastores and also must ensure the quantitative requirements explained in the document D9.1, which are the following ones:

Requirements	Name	Brief Description	Acceptable metric	Target metric
Req.01	MAXIMUM_TIME	The system must be able to solve all analytical queries in less than 1 minute	1 min	2sec

Moreover, having CloudMdsQL simplifies the development because this component integrates inputs and outputs of different datastores using a common syntax. Otherwise, we would need to write a lot of code to parse and sent data from different datastores in each functionality.

7.5 Bibliographic Search: Common CPaaS Components

We have identified a set of components that can be shared across different use cases with a similar architecture:

- **Trident-CloudMdsQL**: Trident provides a way to design transactional analytical processes with Storm to compute aggregates (scores). It is a very common practice to create trident plugins to make persistent queries and aggregates inside a trident topology for a specific data store. Probably this component could be really shared between most of use cases.
- **Storm components:** Storm has a set of different components types: bolts, spouts and aggregators. The document indexation process needs many storm components that are hard to code and could be shared and reused at least between this use case and the social media use case. Probably, we'll realize that other Storm components from other use cases can be shared.
- **Xwork-cloudMdsQL**: XWork has a pluggable architecture and then interceptors, actions and results can be shared across different use cases. We suggest creating and sharing an Xwork-cloudMdsQL plugin to inject the query engine into the actions and to manage the transaction workflow before executing an action. The social media use case also will use the Xwork framework and thus, it can be reused.

8 FUTURE WORK

Ongoing work in the use cases is entering into pre-implementation and implementation tasks. The aim is to achieve the following during the next 12-month period:

- **Data store tests:** results from tests, made within each use case, upon different participating components (data stores). This is necessary to validate adequacy of native functionalities (not implemented on CloudMdsQL), validate performance options on varying loads through experiments, as well as to measure performance in native scenarios
- **Implementation Approach:** definition of techniques that will be chosen and applied to different problems with the use of the CoherentPaaS programming model
- **Implementation of queries:** first CoherentPaaS query implementations