FABIOLA: Defining the Components for Constraint Optimization Problems in Big Data Environment

Luisa Parody *lparody@us.es*
Dto. Lenguajes y Sistemas Informáticos / Universidad de Sevilla
Sevilla, Spain

Ángel Jesús Varela Vaca *ajvarela@us.es*
Dto. Lenguajes y Sistemas Informáticos / Universidad de Sevilla
Sevilla, Spain

María Teresa Gómez-López *maytegomez@us.es*
Dto. Lenguajes y Sistemas Informáticos / Universidad de Sevilla
Sevilla, Spain

Rafael M. Gasca *gasca@us.es*
Dto. Lenguajes y Sistemas Informáticos / Universidad de Sevilla
Sevilla, Spain

Abstract

The optimization problems can be found in several examples within companies, such as the minimization of the production costs, the faults produced, or the maximization of customer loyalty. The resolution of them is a challenge that entails an extra effort. In addition, many of today's enterprises are encountering the Big Data problems added to these optimization problems. Unfortunately, to tackle this challenge by medium and small companies is extremely difficult or even impossible. In this paper, we propose a framework that isolates companies from how the optimization problems are solved. More specifically, we solve optimization problems where the data is heterogeneous, distributed and of a huge volume. FABIOLA (FAst BIg cOstraint LAb) framework enables to describe the distributed and structured data used in optimization problems that can be parallelized (the variables are not shared between the various optimization problems), and obtains a solution using Constraint Programming Techniques.

Keywords: Big Data, Optimization Problem, Constraint Programming, Data Structure.

1. Introduction

Nowadays, huge volumes of data are generated by running services for organization's information systems. The concept of Big Data has been defined as data that exceeds the capability of commonly used hardware environments and software tools to capture, manage, and process it within a tolerable elapsed time for its user population [3]. This concept is being increasingly defined by the four Vs, which are: 1) Volume, which represents the size of the data; 2) Velocity, that represents the speed at which data is created, stored, analyzed, processed, and visualized in real-time; 3) Variety, which distinguishes the forms of data by considering two aspects: syntax and semantics; and 4) Value, that is especially linked to the commercial value that any new sources and forms of data can add to the business.

Software technologies have been evolving to facilitate the management of the Big Data. Hadoop [2, 21] is a popular open-source map-reduce implementation which is being used as an alternative to store and process extremely large data sets on commodity hardware. However, the map-reduce programming model is very low level and requires developers to write custom programs which are hard to maintain and reuse. For this reason, more abstract solutions have been developed to elevate the abstract level, such as Spark. Spark [20] is nowadays the most active Big Data project in the open source community, and it is already used by more than a thousand organizations. Spark is a cluster computing engine that is optimized for an in-memory processing, and it unifies support for a variety of workloads, including batch, interactive querying, streaming, and iterative computations.

One of the challenge of the Big Data solution is the isolation of the users from the heterogeneous use and location of data. For this reason, several components have been developed in the ecosystem of Hadoop. Unfortunately, the optimization of problems using the Constraint Programming techniques is still a problem to be solved.

In this paper, we present FABIOLA (FAst BIg cOstraint LAb) framework, an open-source data problem optimization solution built on top of Hadoop. FABIOLA supports the distribution of the Constraint Optimization Problems in order to obtain the optimal solutions for independent subsets of distributed data. Constraint Optimization Problems are compiled into map-reduce jobs executed on Hadoop that can be combined with Hive [19] in order to infer more information after founding the optimized values.

The rest of the paper is organized as follows. Section 2 introduces previous and related works. Section 3 describes FABIOLA data model, the Constraint Optimization problem, and the parameters that can be included to obtain the evaluation of the optimization. Section 4 describes the architecture and an overview of the query life-cycle. Section 5 provides various real examples where and how FABIOLA is used. Finally, conclusions are drawn and future work is proposed in Section 6.

2. Related Work

Big Data faces up new challenges [12, 14] in how to carry out optimization problems with heterogeneous, incomplete, and uncertainty data in addition to immediately responses for some types of questions.

The Apache Hadoop project [2] actively supports multiple projects with the aim of extending Hadoop’s capabilities and make it easier to use. There are several top-level projects to helping in the creation of development tools as well as for managing Hadoop data flow and processing. Many commercial third-party solutions build on their developed technologies within the Apache Hadoop ecosystem.

Spark [20], Pig [15], and Hive [19] are three of the best-known Apache Hadoop projects. All of them are used to create applications to process Hadoop data. While there are a lot of articles and discussions about which is the best one, in practice, many organizations use various of them since each one is optimized for a specific functionality. Although FABIOLA is not a new Big Data solution, it aims to be part of the Hadoop ecosystem. FABIOLA provides the necessary components to drive the solution of constraint optimization problems with distributed data on a Hadoop-based architecture.

Constraint Programming (CP) presents a challenge in the scalability by solving some type of hard problems. However, CP has been successfully applied in different domains for solving optimization problems, such as scheduling and planning. Although there exist several CP tools, such as IBM-ILOG CPLEX Optimization [7] and Choco Solver [4], none of them provides a Big Data solution. Big Data provides to CP a new perspective with regard to the size and volume of data, and it is a great opportunity to exploit its possibilities to gain efficiency and optimization in operational processes [16]. Additionally, Big Data tackles new challenges [6] dealing with automation of decision-making that involves several (millions) decision variables in optimization of resource consumption, sustainability services, and finance. Nevertheless, the optimization problems in CP need more flexibility and adaptability since the exploration of heterogeneous, enormous and dynamic generation of data requires a quick adaptation of optimization problem in order to provide more holistic solutions.

Although there is an initiative to create a new language to adapt CP languages for Big Data applications [18], it is currently a very immature approach with no continued development.

3. Formalization of the problem

The elements that conform FABIOLA framework are: The Constraint Optimization Problem, Data Model (Input data, Output data, Other data), and a set of parameters to delimit the search. The search consists on to find the optimal solution described in the Constraint Optimization Problem for each set of input data and in different nodes. Then, several optimizations are executed in parallel.

3.1. Constraint Optimization Problem

FABIOLA enables to find the optimal solution for several data input. In other words, FABIOLA solves the same type of problem but with different input data, thereby founding its corresponding and different optimal solutions. A Constraint Optimization Problem (COP) is created in order to find these solutions. To introduce COPs, it is firstly necessary to explain what is a Constraint Satisfaction Problem (CSP).

A CSP [17] represents a reasoning framework consisting of variables, domains and constraints $\prec V, D, C \succ$, where V is a set of n variables $v_1, v_2 \ldots v_n$ whose values are taken from finite domains $D_{v_1}, D_{v_2} \ldots D_{v_n}$ respectively, and C is a set of constraints on their values. The constraint $c_k\left(x_{k_1}, x_{k_2}, \ldots, x_{k_n}\right)$ is a predicate that is defined on the Cartesian product $D_{k_1} \times \ldots \times D_{k_j}$. This predicate is true iff the value assignment of these variables satisfies the constraint c_k. If only the solution that optimize (minimize or maximize) a function f wants to be obtained, it is called a Constraint Optimization Problem (COP).

Some of the variables V can be matched with the input and output variables defined in the Data Model (defined in next subsection). As a consequence, some input variables fix their values to a subdomain and modify the possible optimal solutions found for each tuple. It seems a meta-COP which is partially instantiated for each tuple. In order to understand it better, the following subsection sells out an example.

3.2. Data Model: Input, Output and Other Data

FABIOLA works with tables, analogous to tables in relational databases. It does not mean that the information is stored in a relational database, but there exists a view where the data is structured in tuples and with the same set of attributes. This data might be stored in HDFS, NFS or local directories in different nodes. Each table can have one or more partitions which determine the distribution of data in the various nodes.

Being $\{A_1, A_2 \ldots A_n\}$ attributes for the domains $\{D_1, D_2 \ldots D_n\}$, where the set $\{A_1: D_1, A_2: D_2 \ldots A_n: D_n\}$ is a relational-schema. Each tuple is $\{A_1: d_1, A_2: d_2 \ldots A_n: d_n\}$ where $\{d_1 \in D_1, d_2 \in D_2, \ldots, d_n \in D_n\}$. FABIOLA supports primitive column types (Integers, Floating point numbers, Strings, Dates and Booleans).

This set of attributes is divided into three disjoined groups: Input (IN), Output (OUT) and Others (OT), where $IN \cap OUT = \emptyset$, $IN \cap OT = \emptyset$ and $OT \cap OUT = \emptyset$. The descriptions are:

- IN describes the input variables used in the optimization problem.
- OUT describes the variables of the optimization problem obtained after the search.
- OT describes other variables of the table that can be used to make further queries combining the outputs and these variables. They are not related to the optimization problem since they do not influence in its resolution.

Example of a COP evaluated with multiple tuples

A clear example to understand how the input affects the obtained outputs is a model-based diagnosis problem [5]. The example represents a component composed of *5* elements (two summations and three multipliers). The component obtains two outputs (*f* and *g*) according to the inputs (*a, b, c,* and *d*). Each element is associated to a Boolean variable (C_1, C_2, C_3, C_4, C_5)

that describes if its behavior is correct or incorrect (a *true* or *false* value). To know if the component *i* is working correctly, it is necessary to know if every C_i can take the value *true*, thereby the value of the variable *sum* is *5*. The Constraint Optimization Problem is shown in Table 1.

Table 1: Constraint Satisfaction Problem example.

//Variables and Domains
$a, b, c, d, f, g, x, y, x$: FLOAT;
C_1, C_2, C_3, C_4, C_5: Boolean;
sum: Integer;

// Constraints
$C_1 = (a * c = x) \wedge C_2 = (b * d = y) \wedge C_3 = (c * e = z) \wedge C_4 = (x + y = f) \wedge$
$C_5 = (y + z = g)$

// Optimization Function
$sum = C_1 + C_2 + C_3 + C_4 + C_5$

Table 2 shows some possible scenarios, where the output data is obtained according to each input data (per tuple). Every input data can be instantiated (*tuples #1* and *#2*), and the output is obtained describing that every element is working correctly (*sum* is equal to *5*), or not (*sum* is equal to *4*). Also, it is possible that some input values are unknown, and the COP tries to find values for them to optimize the variable sum (*tuples #3*, *#4* and *#5*).

Table 2: Example of tuples in Model-Based Diagnosis.

	Inputs							**Outputs**	**Others**	
#TupleID	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**sum**	**Test ID**	**Manufacturer**
#1	2	3	3	2	2	10	12	4	Test1	Telco
#2	1	3	5	7	2	26	31	5	Test2	IBM
#3	2	3	3	2	2	12	12	5	Test3	Telco
#4	null	3	null	null	2	10	12	5	Test4	IBM
#5	5	null	2	2	1	9	18	3	Test5	Sony

Other attributes, such as *Manufacturer*, are not part of and do not influence over the resolution of the optimization problem. However, they can be of help to answer queries, such as: Which is the manufacturer with more failed components?

3.3. Configuration of Outputs

Since the data is evaluated in different nodes, and the values of this data are also different according to each tuple, the optimization time and the obtained outputs can be extremely different. For this reason, FABIOLA enables to indicate some parameters to adequate the search to each case:

- Maximum Time (*t*): In order to avoid the delay of evaluating a set of tuples, it is possible to delimit the maximum time to solve each COP. If *t* is reached, it can obtain a solution since a local optimal solution can be found during the search. But if the search has not finished, we cannot ensure that the solution found is the global optimal.
- Optimal found? (*optimal*): If the solution obtained cannot be ensured as the optimal, it would be possible to determine whether the best solution found in *t* is included in the outputs or not. If the parameter takes the value *true*, a new column is automatically added to the output variables (called *optimal*) and it describes the nature of the optimal: global or partial. The column *optimal* can take *true* or *false* value.

- Output Type (*any* or *range*). The output that is optimized can be achieved with fixed values in other variables, or with a set of different values (*range*). To determine if *any* value might be obtained or the possible *range* wants to be known, the output data can be attributed with the parameter *any* or *range*, being *any* by default.

4. FABIOLA's Architecture and Methodology

4.1. Architecture

Figure 1 shows the main components of FABIOLA architecture and its integration with a Big Data infrastructure based on Hadoop.

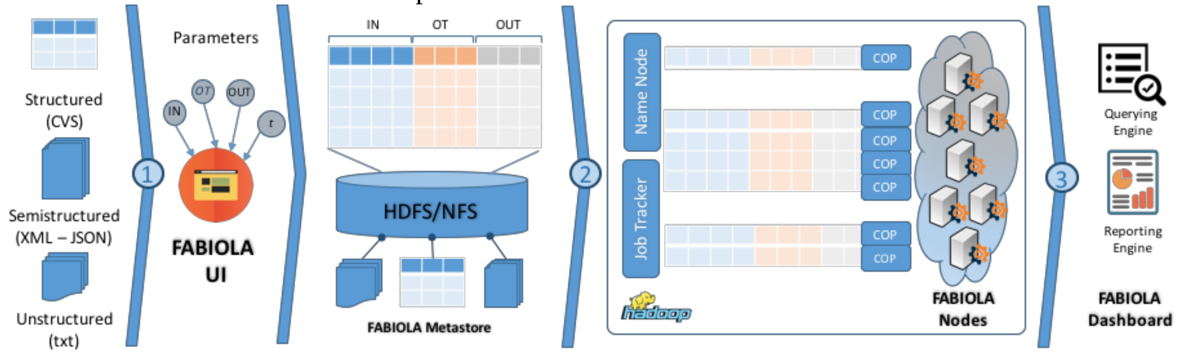


Fig. 1: Architecture of FABIOLA.

FABIOLA is composed of four components:

1. **FABIOLA UI** is a client side (web application) which enables to upload and load data from heterogeneous external resources. Currently, the supported data sources are: (1) structured data from databases; (2) semistructured data, such as XML or JSON files; and (3) unstructured data from text files or similar. FABIOLA UI also enables users to point out attributes from imported data as input data of the COP (cf. *IN* in Figure 1), the establishment of output data (cf. *OUT*), other types of attributes necessary for the problem (cf. *OT*), and finally the specific values for the configuration variables, such as t (cf. *t* at Figure 1).
2. **FABIOLA Metastore** is provided as a system catalog where data is organized in the form of tables and schemes, such as in Hive [19]. These tables and schemes are only a virtual representation or view of the data since it is internally organized in the original format of a distributed file system, for example, HDFS, NFS or AFS.
3. **FABIOLA Nodes** are solver nodes on top of Hadoop that enables to compute COPs. Thus, each row (tuple) instantiates a COP which is uniformly allotted among the available nodes in order to be solved. The possible solutions of those COPs feed the *OUT* column of each tuple in the Metastore.
4. **FABIOLA Dashboard** is a reporting and querying component that enables users an easy-querying and visualization of data and results.

With the aim of a better understanding of the application of the architecture to any problem, a systematic list of steps is given as a methodology in the next section.

4.2. Methodology

The necessary steps to fully execute and take all the advantages of FABIOLA framework are:

1. **Data Load and pre-processing.** The user is in charge of identifying the data to be processed by FABIOLA. More specifically, he must:
   a. Create table/schemes in FABIOLA Metastore.
   b. Load data from external resources through FABIOLA UI.

   c. Establish which attributes from loaded data are *IN* and point out them in FABIOLA Metastore.
   d. Establish which attributes are *OUT* and point out them in FABIOLA Metastore[1].
   e. Specify the values of the configuration parameters if they are necessary. Such as, the values of maximum time (*t*), *optimal* and output type (*any*|*range*).
2. Data Processing. FABIOLA components are the responsible of automatically processing the data:
   a. FABIOLA takes a meta-COP (explained in Subsection 3.1) and instantiates a COP model for each row of data in FABIOLA Metastore. To do that, *IN* and *OUT* values of each row are aligned with the variables of the COP.
   b. The instantiated COPs are sent in parallel to FABIOLA Nodes in order to be computed.
   c. Each FABIOLA Node is executed to solve each COP.
   d. Afterwards, the process finishes and *OUT* attributes are fulfilled (if a solution has been found in *t* and according to the configuration parameters explained in Subsection 3.3).
3. Results are visualized in a configurable dashboard, where the user can even submit queries by combining every attribute of the FABIOLA Metastore.

Figure 2 shows two snapshots of FABIOLA tool: (a) the form to load and pre-processing the data, and (b) the presentation of results in FABIOLA Dashboard. As it is shown, the interface is user-friendly, and supports user throughout the application of the methodology, either explaining each step or suggesting specific configurations. Furthermore, although the description of the set of constraints can be a hard task, it is done once and applied to every set of data. FABIOLA also provides an easy language to define the constraints based on the query language of Constraint Databases [9, 10, 11].

5. Example of Application Scenarios

FABIOLA can be applied to different types of contexts and we have used it in several real scenarios, as explained in the following subsections.

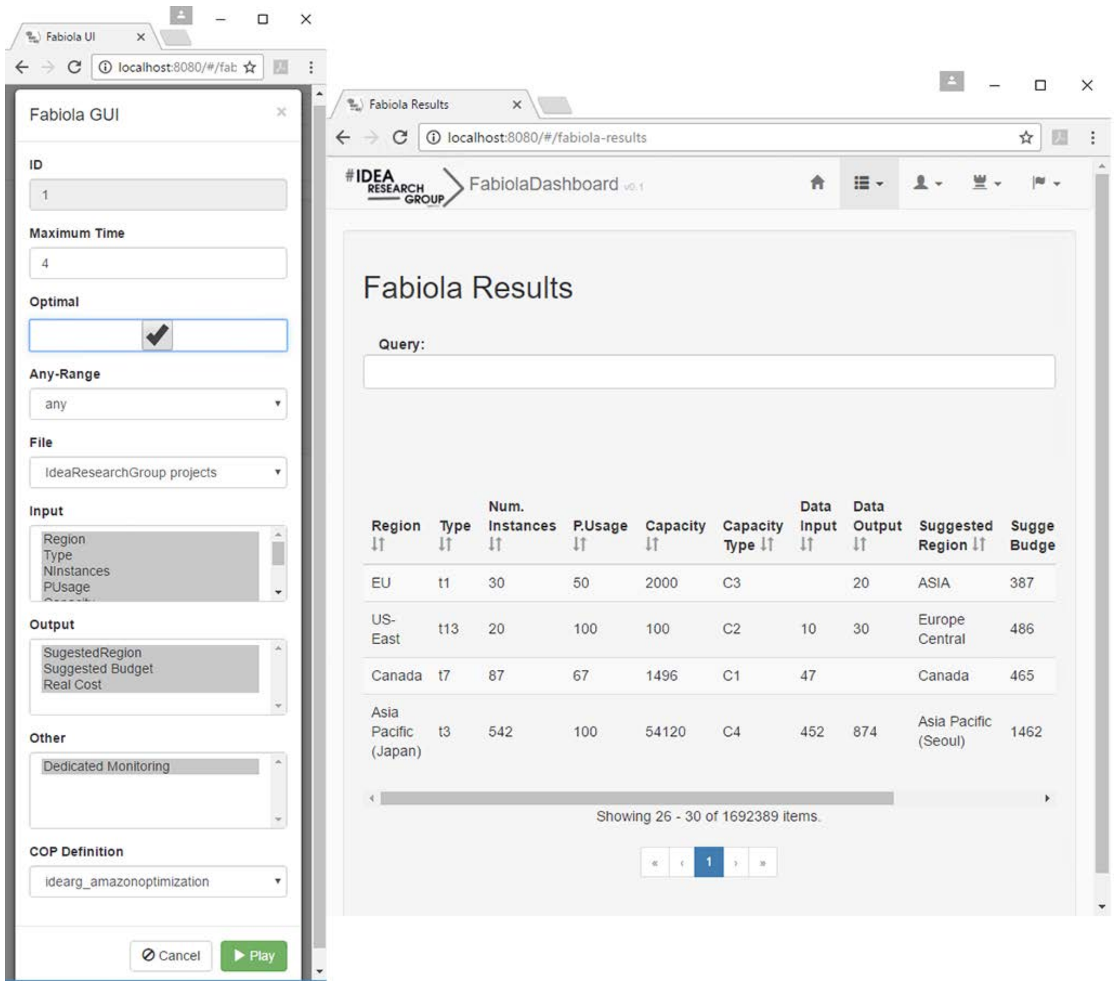
5.1. Contract and Use of Supply Services

In order to success in their operations, customers and companies must contract third companies' services for some basic supplies, such as communications, light and water, services on the cloud, etc. Most of the time, customers hire more resources than they need in order to avoid shortcoming problems. However, it turns into paying more than it is required. On the contrary, contracting services below requirements might imply non-availability of the services or even an extra cost for overused resources. The study of resources consumption for each individual requirements and the imposed market constraints might result in an elastic and customized plan with considerable cost savings for customers and high benefits for companies.

A clear example is Amazon Web Services (AWS) [1], one of the most leading product for hosting services on the cloud. AWS offers multiple pricing plans depending on several characteristics, such as the number of instances, region of location, operative system, etc. A wrong foresight on resource or data consumption could range in an unexpected high cost. AWS's cost is determined by four main categories:

- *Region*, fourteen available regions where computational instances and storage are located.

[1] All the attributes that are not categorized as *IN* or *OUT* are grouped in *OT* in FABIOLA Metastore



Fabiola GUI

ID: 1

Maximum Time: 4

Optimal: ✓

Any-Range: any

File: IdeaResearchGroup projects

Input: Region, Type, NInstances, PUsage

Output: SugestedRegion, Suggested Budget, Real Cost

Other: Dedicated Monitoring

COP Definition: idearg_amazonoptimization

Cancel | Play

Fabiola Results

Query:

Region	Type	Num. Instances	P.Usage	Capacity	Capacity Type	Data Input	Data Output	Suggested Region	Sugge Budge
EU	t1	30	50	2000	C3		20	ASIA	387
US-East	t13	20	100	100	C2	10	30	Europe Central	486
Canada	t7	87	67	1496	C1	47		Canada	465
Asia Pacific (Japan)	t3	542	100	54120	C4	452	874	Asia Pacific (Seoul)	1462

Showing 26 - 30 of 1692389 items.

Fig. 2: (a) FABIOLA User Interface, and (b) FABIOLA Dashboard.

- *Compute*: defined by: the number of instances of a specific type *i* (*NInstances*); the expected percentage of usage of instances of type *i* (*%Usage*); the cost per hour ($Cost - Region(i, r)$) of performing instances of type *i* in region *r*.
- *Storage*: defined by various parameters: capacity of storage (*StorageCapacity*), type of storage (*StorageType*), and cost of the capacity *c* for the type of storage *st* ($Cost - Storage(c, st)$)).
- *Data Transfer*: gigabyte of input data per month (*DataInput*), gigabyte of output data per month (*DataOutput*), and the cost of transfer data into/outside the services in a region *r* (*CostDataTransfer(r)*).

There are several optional parameters that can be used to customize the services but they do not increase the final price of the service, such as monitoring options. The objective is to determine the minimum cost in any region when the percentage of usage in computational instances is greater than *0%*. For example, which is the minimum cost for an expected hardware whose requirements are *n* instances of type *t1*, a pool of storage of at least 1000 GB, 20GB/Month of input/output data, and a maximum workload of *50%* for each *t1*.

- **Data Model:**
  - **IN**: *Region, InstanceType, NumberOfIntances, %Usage, StorageCapacity, StorageT ype, DataInput, DataOutput.*
  - **OUT**: *Region, EstimatedCost*
  - **OT**: *Description, DedicatedMonitoring, ELB, OperativeSystem.*
- **Constraint Optimization Problem**:
  - *{Region, InstanceType, …, DataOutput} Integer;*

- $\forall \mathrm{i} \in \{0 \dots \mathrm{NumberIfInstances}\} \%Usage_i \geq 1 \& \%Usage_i \leq 50 \rightarrow CostCompute = \%Usage_i * CostRegion(InstanceType, Region);$
- $CostStorage = CostStorage(StorageType, Region) * StorageCapacity;$
- $CostData = CostDataTransfer(Region) * DataOutput;$
- $Minimize\ (CostCompute + CostStorage + CostData);$

- **Example of Queries:** Which is the region where the estimated cost is minimum, or less than 100 per month, with a specific configuration? Which are the instances with more than one hundred of extra elastic IPs?

5.2. Diagnosis Problem for Heat Exchangers

The diagnosis problem in a set of reading sensors is an important challenge that manages a huge number of systems that can be diagnosed in parallel. Model-based diagnosis can be described as an optimization problem, where the minimal explanation of a malfunction can be found [13]. A set of constraints describes the relation that each part of the system must follow. The objective is to describe the relations between them and to detect and diagnose the possible errors that occur in the system.

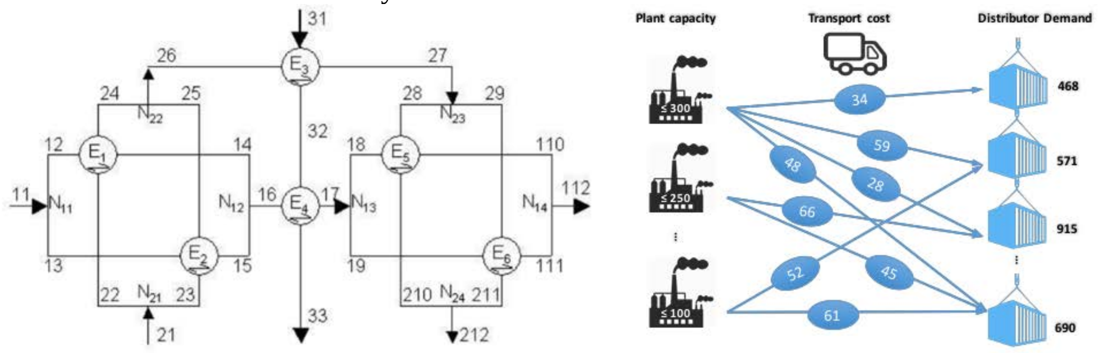


Fig. 3: (a) Heat Exchanger (b) Automobile Supply Chain Model.

Each system, shown in Figure 3.(a), is composed by: six heat exchangers, called $E_1, E_2, E_3, E_4, E_5, E_6$, and eight nodes, called $N_{11}, N_{12}, N_{13}, N_{14}, N_{21}, N_{22}, N_{23}, N_{24}$. Each connection (tube) between an exchanger and a node is defined by two parameters: a flow (f_i) and a temperature (t_i) (where i is the enumerated name of the connection). For example, the three input arrows of the system in Figure 3.(a), enumerated as *11*, *21*, and *31*, defines three input flows, called f_{11}, f_{21}, and f_{31}, and three temperatures, called t_{11}, t_{21}, and t_{31}. This nomenclature is applied to the rest of connections between exchangers and nodes. The correctness of a system is defined by a set of constraints that relates the flow and temperature of each connection that the exchangers and nodes manage [9]. More specifically, there are a set of polynomial constraints that defines three different kinds of balances that the exchangers and nodes must satisfy:

- $\sum_i f_i = 0$: mass balance at each node.
- $\sum_i f_i \dots t_i = 0$: thermal balance at each node.
- $\sum_{IN} f_i \dots t_i - \sum_{OUT} f_i \dots t_i = 0$: enthalpy balance for each heat exchanger.

A component works correctly if it satisfies its corresponding balances constraints. Thus, in order to diagnoses the minimum malfunction components, the objective is to maximize the number of correct components. Following the defined parts of FABIOLA, the components are:

- **Data Model:**
  - **IN**: $t_{11}, t_{12}, t_{13}, t_{16}, t_{17}, t_{18}, t_{19}, t_{112}, t_{21}, t_{26}, t_{27}, t_{212}, t_{31}, t_{33}, f_{11}, f_{12}, f_{13}, f_{16}, f_{17}, f_{18}, f_{19}, f_{112}, f_{21}, f_{26}, f_{27}, f_{212}, f_{31}, f_{33}$
  - **OUT**: $N_{11}, N_{12}, N_{13}, N_{14}, N_{21}, N_{22}, N_{23}, N_{24}, E_1, E_2, E_3, E_4, E_5, E_6$
  - **OT**: *Name*, *Location*

- **Constraint Optimization Problem**:
    - $\{t_{11}, t_{12}, \dots, t_{31}, t_{33}, f_{11}, f_{12}, \dots, f_{33}\}$ *Integer;*
    - $N_{12}^a, N_{12}^b, N_{21}^a, N_{22}^b, E_1^a, E_1^b, E_1^c, E_2^a, E_2^b, E_2^c$ *Boolean;*
    - $N_{12}^a = f_{14} + f_{15} - f_{16} = 0$
    - $N_{12}^b = f_{14} * t_{14} + f_{15} * t_{15} - f_{16} * t_{16} = 0$
    - $N_{21}^a = f_{21} - f_{22} - f_{23} = 0$
    - $N_{21}^b = f_{21} * t_{21} - f_{22} * t_{22} - f_{23} * t_{23} = 0$
    - $N_{22}^a = f_{24} - f_{25} - f_{26} = 0$
    - $N_{22}^b = f_{24} * t_{24} - f_{25} * t_{25} - f_{26} * t_{26} = 0$
    - $E_1^a = f_{12} - f_{14} = 0$
    - $E_1^b = f_{22} - f_{24} = 0$
    - $E_1^c = f_{12} * t_{12} - f_{14} * t_{14} + f_{22} * t_{22} - f_{24} * t_{24} = 0$
    - $E_2^a = f_{13} - f_{15} = 0$
    - $E_2^b = f_{23} - f_{25} = 0$
    - $E_2^c = f_{13} * t_{13} - f_{15} * t_{15} + f_{23} * t_{23} - f_{25} * t_{25} = 0$
    - $Maximize(N_{12}^a + N_{12}^b + N_{21}^a + N_{22}^b + E_1^a + E_1^b + E_1^c + E_2^a + E_2^b + E_2^c)$
- **Example of Queries:** Which are the locations where N_{12}^a is failing (N_{12}^a or N_{12}^b is *false*)? Which is the *Name* of the systems that are working correctly?

5.3. Automotive Supply Chain Problem for a Demand-Driven Distribution

In the area of automotive industry, and any other kind of industry where production and delivery are necessary, it is necessary to transport the products from production centres (*n* Plants) to distribution centres (*m* Dist), where $n \ll m$ (see Figure 3.(b)). Different demands on these last centres determine the different solutions, that are represented in a bidimensional array, called *Assign[n,m]*. The objective is to obtain the minimal transport cost in order to maintain a competitive advantage. The constraints of this problem are defined according to the characteristics of the different logistic enterprises of the market. Following the defined parts of FABIOLA, where each tuple represents the distribution per day, the components are:

- **Data Model:**
    - **IN**: *Capacity: Array[NFact] of Integer, Demand: Array[MDist] of Integer, Cost: Array[NFact,MDist] of Integer.*
    - **OUT**: *TotCost: Integer, Assign: Array[NFact,MDist] of Integer*
    - **OT**: *Name of the Logistic Enterprise.*
- **Constraint Optimization Problem**:
    - *Assign[NFact, MDist]: Integer; TotalCost: Integer;*
    - $\forall \text{p} \in \{0, \dots, \text{n}\} \sum_{c=1}^{m} Assign[p, c] \leq capacity[p]$;
    - $\forall \text{c} \in \{0, \dots, \text{m}\} \sum_{p=1}^{m} Assign[p, c] \leq demand[c]$;
    - $\text{TotalCost} = \sum_{p=1}^{n} \sum_{c=1}^{m} Assign[p, c] * cost[p, c]$
    - $Minimize\ (TotalCost)$;
- **Example of Queries:** Which are the Assign and $TotalCost$ for the DHL logistic enterprise? Which are the logistic enterprises whose $TotalCost$ is less than 50.000?

6. Conclusions and Future work

Optimization problems are found in several real examples. It becomes a higher problem when the data involved is in a Big Data environment, which implies huge quantity of information, distributed and heterogeneous. FABIOLA framework has been formalized to support the definition and resolution of distributed Constraint Optimization Problems, isolating from where the data is, and how the optimal outputs are found. Three different examples have been introduced to show the flexibility of the proposal. An interface has also been developed to approach the solution to final users.

As future work, it would be interesting to extend the type of elements included in the input and output variables, such as nestable collection types-array and map. These types imply the management of semi-structure and non-structured information. The possibility of giving users the opportunity to define their own types programmatically could be also helpful.

Acknowledgements

This work has been partially funded by the Ministry of Science and Technology of Spain (TIN2015-63502-C3-2-R) and the European Regional Development Fund (ERDF/FEDER).

References

1. Amazon-Web Services, http://aws.amazon.com/whitepapers/. Accessed April 15, 2017.
2. Apache Hadoop, http://wiki.apache.org/hadoop. Accessed April 15, 2017.
3. Chattopadhyay, B., Lin, L., Liu, W., Mittal, S., Aragonda, P., Lychagina, V., Kwon, Y., and Wong, M. Tenzing a sql implementation on the mapreduce framework. In Proceedings of VLDB, pp. 1318--1327, (2011)
4. Choco Solver, http://www.choco-solver.org/. Accessed April 15, 2017.
5. Ceballos, R., Gómez-López, M. T., M. Gasca, R., Del Valle Sevillano, C. A compiled model for faults diagnosis based on different techniques. AI Commun. 20(1), 7-16 (2007)
6. Freuder, E.F., and O'Sullivan, B. Grand challenges for constraint programming. Constraints, 19(2), 150-162 (2014)
7. IBM-ILOG CPLEX Studio, http://www-03.ibm.com/software/products/es/ibmilogcpleoptistud. Accessed April 15, 2017.
8. Gómez-López, M. T., Ceballos, R., M. Gasca, R., Del Valle, C. Applying Constraint Databases in the Determination of Potential Minimal Conflicts to Polynomial Model-based Diagnosis. CDB 2004: pp- 75-89 (2004)
9. Gómez-López, M.T., Ceballos, R., M. Gasca, R., del Valle Sevillano, C. Developing a labelled object-relational constraint database architecture for the projection operator. Data Knowl Eng 68(1), 146–172 (2009)
10. Gómez-López MT, M. Gasca, R. Using constraint programming in selection operators for constraint databases. Expert System Applications 41(15), 6773–6785 (2014)
11. Gómez-López MT, M. Gasca, R. Object Relational Constraint Databases for GIS. Encyclopedia of GIS 2017, 1449-1457 (2017)
12. Labrinidis, A. and Jagadish, H.V. Challenges and opportunities with big data. Proc. VLDB Endow., 5(12), 2032-2033 (2012)
13. M. Gasca, R. , Del Valle Sevillano, C., Gómez-López, M. T., Ceballos, R. NMUS: Structural Analysis for Improving the Derivation of All MUSes in Overconstrained Numeric CSPs. CAEPIA 2007, pp.160-169 (2007)
14. Nasser T and Tariq RS. Big data challenges. Computer Engineering \& Information Technology, 4(3), 1-10 (2015)
15. Olston, C., Reed, B., Silberstein, A., and Srivastava, U. Automatic optimization of parallel dataflow programs. In USENIX 2008 Annual Technical Conference, ATC'08, pp. 267--273, Berkeley, CA, USA (2008)
16. O'Sullivan, B. Opportunities and challenges for constraint programming. In Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, AAAI'12, 2148--2152. AAAI Press (2012)
17. Rossi, F., van Beek, P., and Walsh, T. Handbook of Constraint Programming. Elsevier (2006)
18. Rossi, F. and Saraswat, V. Constraint programming languages for big data applications.

19. Thusoo, A., Sarma, J.S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., and Murthy, R. Hive: A warehousing solution over a map-reduce framework. Proc. VLDB Endow., 2(2), 1626-1629 (2009)
20. Zaharia, M., Xin, RS., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M.J., Ghodsi, A., Gonzalez, J., Shenker, S., and Stoica, I. Apache spark: a unified engine for big data processing. Commun. ACM, 59(11), 56-65 (2016)
21. White, T. Hadoop: The Definitive Guide. O'Reilly Media, Inc., 1st edition (2009)