ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΜΕΤΑΠΤΥΧΙΑΚΟ ΔΙΠΛΩΜΑ ΕΙΔΙΚΕΥΣΗΣ (MSc)
στα ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Πληροφοριακά Συστήματα Μέτρησης Ενεργειακής Αποδοτικότητας και Αξιοποίησή της στη Λήψη Αποφάσεων

Ευστάθιος Πλήτσος

M3100002

ΑΘΗΝΑ, ΦΕΒΡΟΥΑΡΙΟΣ 2012

ΜΕΤΑΠΤΥΧΙΑΚΟ ΔΙΠΛΩΜΑ ΕΙΔΙΚΕΥΣΗΣ (MSc)
στα ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Πληροφοριακά Συστήματα Μέτρησης Ενεργειακής Αποδοτικότητας και Αξιοποίησή της στη Λήψη Αποφάσεων

Ευστάθιος Πλήτσος

M3100002

Επιβλέπων Καθηγητής: Παναγιώτης Μηλιώτης
Εξωτερικός Κριτής: Ιωάννης Μούρτος

ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΑΘΗΝΑ, ΦΕΒΡΟΥΑΡΙΟΣ 2012

Special thanks for their support to

- Dr. P. Miliotis, Dr. I. Mourtos, Dr. K. Pramatari, from the Athens University of Economics and Business;
- Dr. V. Nikolopoulos, CEO of Intelen;
- the people of the Greek Research and Technology Network (GRNET).

Especially I want to thank the members of my family for the support and patience they showed during the writing of this thesis.

Contents

List of Figures

List of Tables

Abstract

Energy efficiency is considered a crucial factor for the minimization of both monetary and environmental cost, especially within developed economies that are simultaneously large energy consumers and environmentally aware social systems. Governmental restrictions regarding energy consumption and carbon emissions of industries signify energy efficiency as a crucial factor for environmental impact. That is, apart from monetary cost, the environmental impact of processes and products defines also an important efficiency index for modern supply chains.

Considering that more environmentally aware supply chains are the desirable future condition, we developed an information system that is able to capture energy consumption information across the supply chain and map it on processes or products, dealing efficiently with the issue of very large volume of energy data. Additionaly, by a way of an exercise, we developed a mathematical model for production planning that provides optimal decisions with respect to energy consumption.

1 Introduction

This thesis examines the problem of exploiting energy efficiency for decision support in supply chain management through measuring energy consumption and mapping it on processes or products.

Governmental restrictions [8] regarding energy consumption and carbon emissions of industries signify energy efficiency as a crucial factor for environmental impact. That is, apart from monetary cost, the environmental impact of processes and products defines also an important efficiency index for modern supply chains [26].

Considering that more environmentally aware supply chains are the desirable future condition, we approach the problem of the use of energy efficiency in decision support with an information system based on a three-layered architecture, able to deal with the issues of

- energy consumption information acquisition across the supply chain;
- management and storage of very large volume of data, considering the number of possible energy consumption sources and different partners across the supply chain;
- data processing through the use of logic and optimization with respect to energy efficiency.

Each such issue defines a layer in the proposed architecture.

For the data acquisition layer we have made the assumption that we have a metering device, based on standards [19, 3], that can measure energy consumption and can connect to a computer network, so that these measurements can be sent to an energy consumption information listening module.

Taking into consideration the large volume of data, for the second layer, we have selected selected three open-source tools Hadoop [13], HBase [14] and Zookeeper [27], which tackle this problem efficiently, and we have developed developed a fourth one that manages to listen and pre-process incoming energy consumption information.

For the data processing layer, we have developed, by way of an exercise, a mathematical model, based on existing ones [15, 20], that provides optimal deecisions for production planning with respect to energy consumption.

The non-existing modules of the proposed system were developed in the programming language Java.

Thanks to the people of the Greek Research and Technology Network [23] who provided access to Okeanos cloud [6], the system was fully implemented and tested.

Finally, implication for future research is given towards these three layers considering

- security issues;
- expandability of the data acquisition layer towards the ability of multiple metrics measurement, such as carbon emissions, water and waste managements, etc.;
- system sustainability;
- expandability of the data processing layer so that optimization models can take into consideration multiple environmental efficiency indices.

1.1 Motivation

Energy efficiency is considered a crucial factor for the minimization of both monetary and environmental cost, especially within developed economies that are simultaneously large energy consumers and environmentally aware social systems [26]. A rising problem seems to be the supply chain of the future with respect to the existing and growing environmental issues.

According to a global IBM survey based on face-to-face conversations with nearly 400 supply chain executives [12], the vision of the future describes a supply chain that has to be instrumented, which means that information previously created by people will increasingly be machine-generated (coming out of sensors); interconnected, making worldwide networks of supply chains able to plan and make decisions together; intelligent, leading to much smarter decisions made by smarter systems automatically, thus increasing responsiveness and limiting the need for human intervention. As stated in the survey, sustainability in the supply chain, including the management of energy, water, waste and carbon emissions, "is an increasing concern due to more environmental accountability customer demands and governmental compliance mandates" .

Greener supply chains include greener nodes/partners in a particular industry, a fact that seems

to lead current condition of supply chain management, slowly but steadily, into a new era where collaboration is not only cost and trust guided, but environmentally aware as well. This new era provides motivation both for potential partners to be as environmentally aware as they can, and for the development of modules, systems and networks within an industry containing continuous and frequently updated information of the environmental impact per partner. The course to this point is slow due to the new systems and architectures that have to be developed and adopted within an organization and steady not only because of the customer demands (more than half of the questioned executives in [12] have modified product design or packaging to address environmental considerations) but because of governmental restrictions too [8].

Taking into consideration these two issues, i.e. the future supply chain and the environmental sustainability in it, it is clear enough that decisions leading to energy-efficient/greener processes and products have to be based on real and accurate data, which map the current environmental impact, flowing automatically and constantly across the supply chain.

Environment impact is definitely not a "single variable function". Multiple variables participate in the construction of total environmental harm done by a single process in a supply chain, such as energy consumption, carbon emissions, water and waste management etc. Given the large spectrum of research question related to the above, our contribution focuses on how to obtain optimal decisions with respect to electrical energy consumption. Although restrictive, the suggested technical approach is designed to allow future implementations to be able to map upon processes and products multi-criteria environmental impact.

1.2 Our Approach

Considering the importance of energy consumption measurement per product or process across a supply chain and how this measurement can be used to optimize production planning and scheduling, an energy monitoring system, with a very generic and easily expandable architecture, can be used for this purpose. Our approach introduces a layered architecture, which deals with the issues of

1. data acquisition whith the use of energy consumption sensors or metering devices,
2. data repositories through traditional and non-traditional databases,
3. and data processing, through the use of logic and optimization with respect to energy efficiency

modules.

Main goal of this architecture is to make real-time monitoring feasible taking into consideration management of very large data volume, factors that lead to the need for high performance and scalability of this system.

This system is not designed to be only another monitoring system, but groundwork contribution to a future network containing continuous and frequently updated information of environmental impact of partners within an industry. Consequently, high scalability appears as a very important factor, due to the vast volume of data arriving from multiple sensors and partners across a supply chain. This future state of information sharing across a supply chain provides higher motivation to every potential partner to be as environmentally aware as possible.

This approach is clearly based on the three dimensional IBM vision of the supply chain of the future [12], which is

1. instrumented, using data acquisition equipment to measure energy consumption,
2. interconnected, by setting the course for the creation of a network containing continuous and frequently updated information of environmental impact of potential partners with the use of modules that are able enough to deal with very large volume of energy consumption information, and
3. intelligent, with the use of logic components and data processing techniques leading to optimal decisions with respect to energy consumption.

Expandability, concerning additional metrics of environmental impact such as carbon emissions, water and waste management etc, and how they can be measured and contribute to optimal decisions with respect to environmental impact is certainly an open case for future research and development.

The contribution of this thesis can be summarised as follows

- selection of specific distributed modules, such as Hadoop [13] and HBase [14], leads to high and adjustable to higher scalability with the use of commodity personal computers.
- the proposed distributed modules responsible for data aggregation, Hadoop, and storing, HBase, have not been used yet in the domain of energy consumption monitoring.

- every module in the proposed architecture is an open-source project, thus this architecture is guided by the open-source philosophy.
- a mathematical model, by way of an exercise, based on existing models [15], [20], proposed to take into consideration not only cost reduction and demand satisfaction but energy consumption minimization as well.

The remainder of this thesis is organised as follows; in Section 2 are being described general system requirements and the architecture as seen from a higher point of view. In Section 3 is described the basis of the system, data capturing layer, including sensors and data - energy consumption information format description. In Section 4 is described the data repositories layer, including descriptions of modules that allow management of very large volume of data such as distributed databases and aggregation interfaces; a database-schema, able to map and store energy consumption information, meeting the system requirements. In Section 5 is described the implementation and the performance evaluation of the system. In Section 6 are described two tested alternative approaches towards the data repositories layer. In Section 7 is described the data processing layer, including a mathematical model able to consume energy data and provide optimal solutions for production planning, as derived by a commercial solver. In Section 8 are described implications for future research and development. In Appendix A can be found the proposed and used energy consumption information format. In Appendix B can be found commented Java code written for this sytem. In Appendix C can be found Hadoop's runtime output.

2 System Analysis and High Level Architecture

2.1 Requirements Analysis

Requirements analysis is the starting point for every information system.

In this section general requirements are described as stated by users of a similar system, currently under development, under the umbrella of a european research project [21] for the reduction of energy consumption in the textile industry. Requirements, stated by users from this industry, where generalized, meeting requirements from several industries.

Different use-cases are described as scenarios defining sets of processes needed.

Scenario 1: Real-time monitoring / actual state / alarming

Objectives:

1. Transparency, analysis of critical situations, capacity overload of resources
2. Alarming in order to be able to react manually
3. Controlling of critical situations

Description:

Commonly used resources often are limited, or should not exceed certain thresholds. On one production site, the electricity consumptions vary widely. The main cause for energy peaks are several facilities, which have a particularly high demand in specific, rather short phases (like start-up). If several of those energy demand peaks add up, this can cause really critical peaks, which have to be avoided. If the electricity demand does reach a critical level, an alarm is triggered.

Scenario 2: Mapping of actual (measured) energy consumptions to production orders

- Capturing of data on machinery level, mapping to production orders
- Differentiation between setting up/heating up, production and cleaning phase

Objectives:

1. Post calculation analysis

2. Analysis of accumulated data, in order to compare different alternatives (e.g. using different machinery for one process, sequence optimisation of production orders)

Description:

The measuring of the energy consumption is quite useful, to get an overall image of the consumption. A further step is to allocate those consumptions to production orders in order to be able to evaluate the energy costs of this product. Sometimes several production orders can be bundled concerning some production processes. In order to determine the energy consumption of a production order different processes have to be summed up, determining the proportion which has to be assigned to the product, concerning multi-order processes.

Scenario 3: Analysing and determining the actual energy consumption of products / allocation of energy-overhead.

Determining the energy consumption of non-production processes and its proportion to be assigned to individual products

- Product development, sample production
- Sales
- In-house transport
- Buildings

Objectives:

1. Identification and reduction of significant energy-overheads.
2. Determining more complete footprints of products.

Description:

Existing methodologies and tools developed to measure the environmental impact of product and processes such as Life Cycle Assessment (LCA) and its offsprings, Life Cycle Inventory (LCI) and Life Cycle Impact Assessment (LCIA) [24], often only take the production's processes into account. This is quite appropriate regarding large scale series production. For small series and special orders processes like sample production may have a significant impact on the overall LCI.

Scenario 4: Mapping of energy consumption across the supply chain

Objectives:

1. Reviewing energy consumption per product or process across the supply chain including energy consumed by partners
2. Propose greener partners for collaboration

Description:

Mapping of energy consumption upon supply chain nodes seems to be a key factor for visualization of energy consumption on the whole of supply chain. This inculdes processes that are being outsourced, leading to the ability of choice from a potential partner network with environmental criteria by storing and processing of environmetal impact per partner.

2.2 Prerequisites

Detailed description of this architecture requires a set of definitions first.

Sensor: A device that measures a physical quantity and converts it into a signal which can be read by an observer or by an instrument.

Data stream: A flow of on-line data arriving in a continuous fashion per quantum of time. The shortest this quantum is, the closer to real-time the arriving data are.

Data Stream Management System (DSMS): a computer program that controls the maintenance and querying of data in data streams. A key feature of these DSMSs is the ability to execute a continuous query against a data stream.

Extensible Markup Language (XML): A set of rules for encoding documents in a machine readable form.

Relational Database Management System (RDBMS): A software package with computer programs that control the creation, maintenance, and use of a database in which data is stored in tables and the relationships among the data are also stored in tables.

Big data: Data-sets that grow so large that they become awkward to work with using on-hand database management tools. Difficulties include capture, storage, search, sharing, analytics and visualizing.

Distributed system: System consisted of multiple autonomous computers that communicate through a computer network. The main idea of distributed computing is to utilize processing power and storage capacity of these computers. In general, distributed computing is a very efficient solution to the big data management problem.

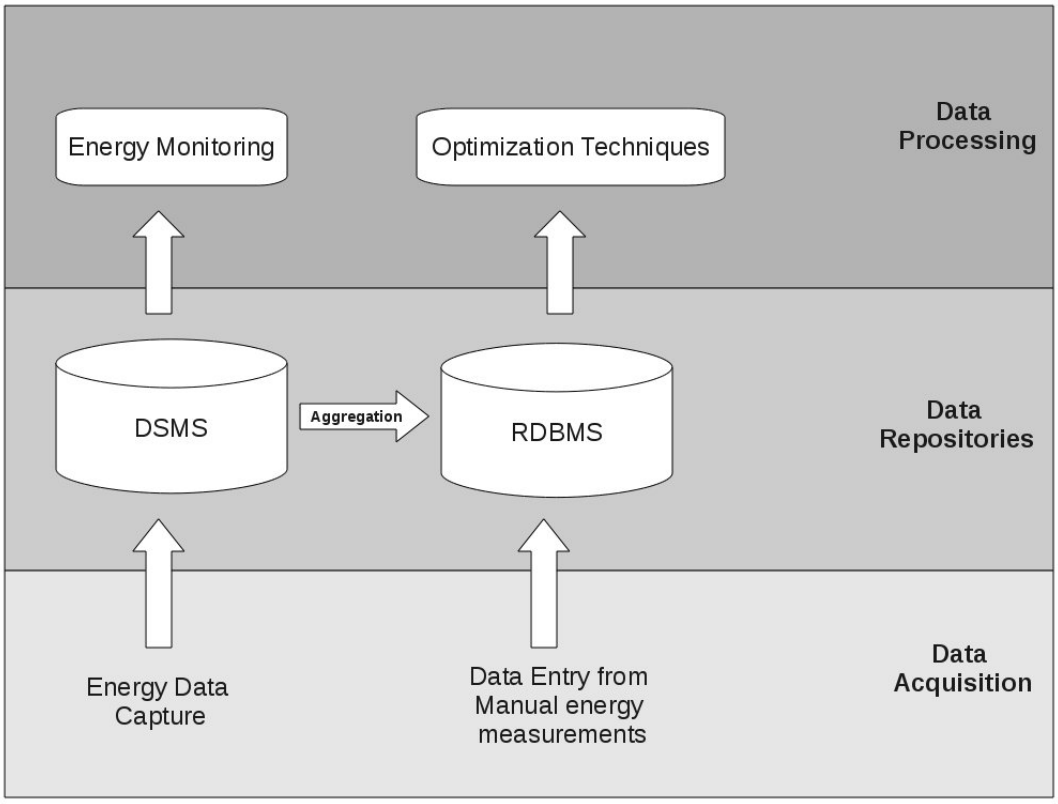
Bigtable: A distributed storage system for managing structured data that is designed and implemented by Google to scale to a very large size, e.g. petabytes of data across thousands of commodity servers [4].

Map-Reduce: A programming model and an associated implementation, designed and implemented by Google, for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model as shown in [11].

2.3 System Architecture - High level approach

In this section is described the system's architecture as viewed from a higher point of view. A three-layered architecture is being introduced enabling interoperability both internally and among supply chain partners. Figure 1 gives a schematic representation of the proposed architecture.

Figure 1: System Architecture - A higher level approach

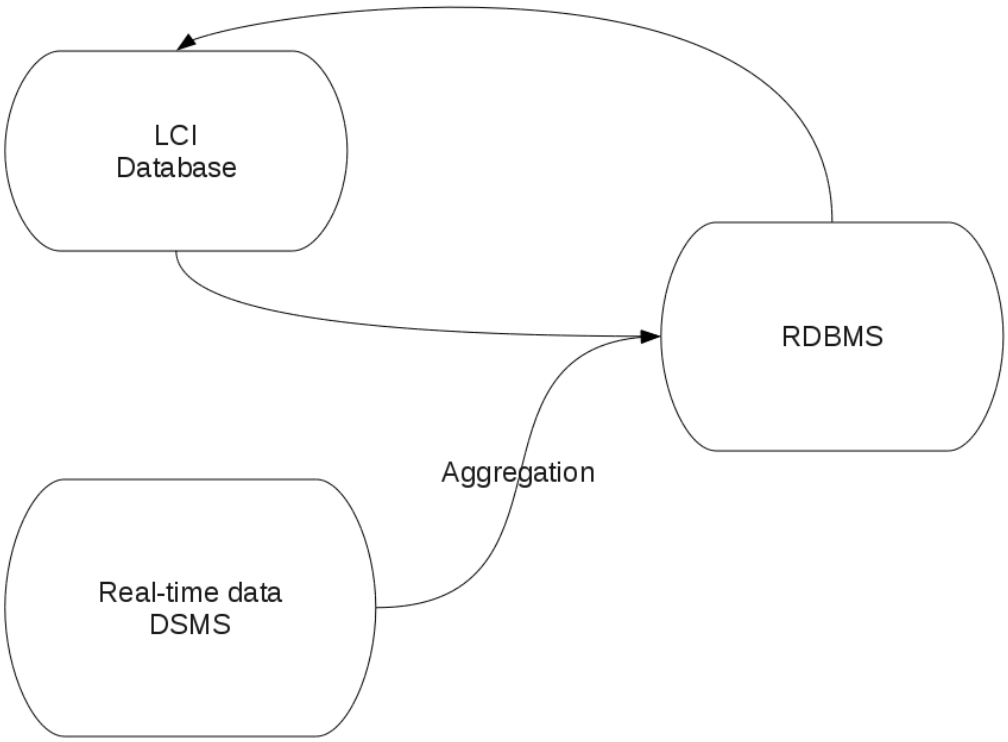


The lower layer, Data Acquisition, is the basis of the system. Its role is to acquire energy consumption data across a supply chain. As shown in figure 1 there are two different ways of data acquisition, automatically (Energy Data Capture) through the use of energy sensors or metering devices and data entries from manual energy consumption measurements. Manual insertion of data works as an alternative way of data acquisition towards three directions: for security reasons when automatic capturing fails, for evaluation of automated measurements and last, in cases where automatical capturing is not applicable, e.g. specific processes in the supply chain are being outsourced and the responsible partner for it does not have installed such equipment or information system. The second and third issue can be solved through the use of existing methodologies and tools devel-

oped to measure the environmental impact of product and processes such as Life Cycle Assessment (LCA) and its offsprings, Life Cycle Inventory (LCI) and Life Cycle Impact Assessment (LCIA) [24]. Data born from these tools can be manually inserted, or used to evaluate data flowing automatically across the supply chain.

The second layer, Data repositories, is the backbone of the system. This layer holds the burden of very large data volume management, aggregation and storing. Considering the volume of data flowing out of multiple energy consumption sensors or metering devices from multiple partners, high scalability and high performance of this layer appears as a vital factor. Incoming data from sensors are being managed as data-streams, stored in a non-traditional distributed database [14], and through an aggregation technique [11] able to manage this volume of data, are finally stored in a RDBMS. Data inserted manually are directly stored in the RDBMS. Figure 2 displays the way LCA and sensor data interact.

Figure 2: System Architecture - Data interrelationship



The third and last layer, Data Processing, is the logic layer of the system. It consists of two components: Energy Monitoring and Optimization Techniques. Energy Monitoring implies real-time monitoring which is feasible via high performance of the second layer. Optimization Techniques refers to a mathematical model which consumes aggregated energy data and provides optimal solutions for production planning, as derived by a commercial solver.

Each layer will be described in detail in a separate section.

3 Data Acquisition

3.1 Data Source

Energy consumption in a single plant occurs in production, storage and administration departments, where electric and electronic devices. anything from a personal computer to a production machine, are being used. Data in this case is energy consumption of this equipment, which can be either collected automatically flowing out of energy sensors, or manually.

Automated capturing of energy consumption information is based on energy sensors or metering devices placed on or near to electric or electronic equipment, providing every period of time measurements of energy consumption. A rising question seems to be the place where these devices should be installed. This factor defines the detail of measurements taken by sensors, and depends on the user requirements. Sensors can be placed either on a single workstation providing the energy consumption of it, or on the switchgear providing the energy consumption of the set of workstations plugged on it.

Manual insertion of energy consumption information is another data source based on existing methodologies and tools developed to measure the environmental impact of products and processes such as Life Cycle Assessment (LCA) and its offsprings, Life Cycle Inventory (LCI) and Life Cycle Impact Assessment (LCIA). According to [24] "Life cycle assessment is a 'cradle-to-grave' approach for assessing industrial systems. 'Cradle-to-grave' begins with the gathering of raw materials from the earth to create the product and ends at the point when all materials are returned to the earth. LCA evaluates all stages of a products life from the perspective that they are interdependent, meaning that one operation leads to the next. LCA enables the estimation of the cumulative environmental impacts resulting from all stages in the product life cycle, often including impacts not considered in more traditional analyses ". Since this kind of data source may preexists, it should be used to evaluate real-time data which flow out of sensors. Additionally, in some cases energy sensoring devices are not applicable, e.g. many product parts are produced by a partner who has not installed such equipment or information system, or, the metrics which set up the desirable key performance indicators (KPIs) are not measurable with the use of such equipment.

3.2 Data Format

The main objective of this section is to describe how interoperability issues, concerning multiple sensors or metering devices, are being tackled, hence several different formats of energy consumption information can be transformed into one.

Data acquisition, depends on energy consumption sensors or metering devices. Specific hardware devices should not be imposed, because the main objective of this approach is a flexible architecture, where interoperability issues can be tackled and existing sensors and metering devices can be embodied in it.

Manual insertion of data is a trivial scenario which should not be further described, beacause once data are inserted, automatical storage in a RDBMS takes place. In the following sections the database schema is described.

The main issue addressed here is, how data flowing out of sensors can be transformed into a single computer readable form. One solution comes with the use of integrated software services which convert several types of data format into one, e.g. Oracle Sensor Edge Server (server). How this software can tackle this issue can be found in the documentation [17]. The main idea is that the developer is responsible to locate every sensor and metering device used, find or develop program drivers compatible with Oracle Sensor Edge Server for these sensors and metering devices, and finally configure the server to communicate with these devices [17]. This approach, is clear enough that tackles interoperability issues in a way that lacks flexibility and user friendliness.

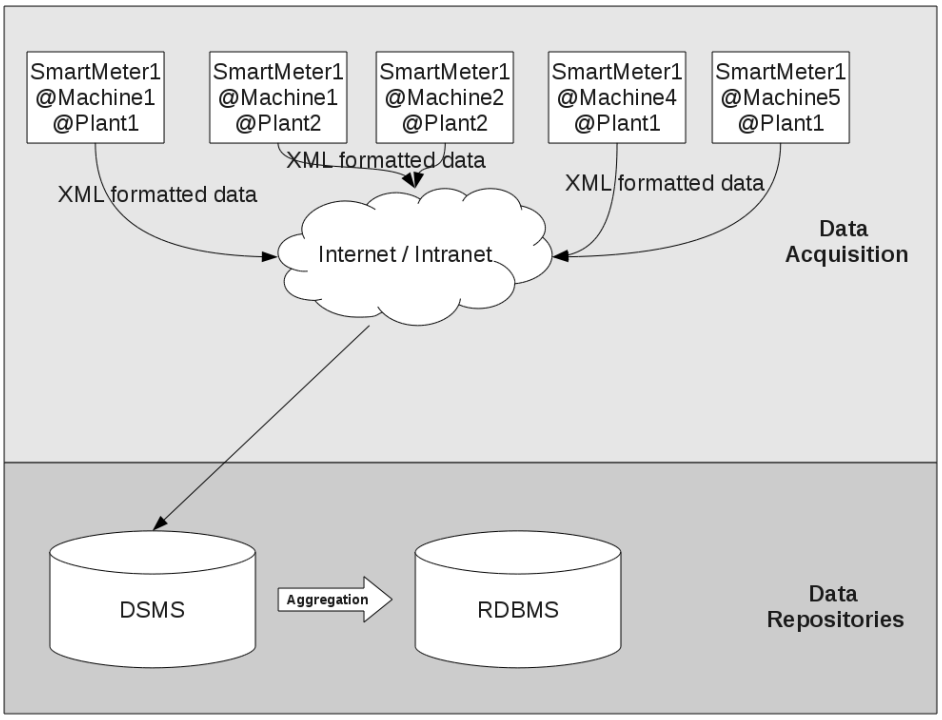
A second solution, comes with the use of middleware devices able to communicate with any kind of sensor or metering device, transform several energy consumption information formats into one, and connect to a computer network. This is the main assumption of this architecture based on existing devices such as the smart-meter designed and manufactured by Intelen [3], [19]. The chosen data format is XML due to its generic and user defined structure and easily and efficiently, through several existing Application Programming Interfaces (APIs), manageable way of processing. This middleware device (smart-meter) provides data from sensors every n seconds, which are sent through a network to the data processing server. Each XML file sent over the network is the stream of information from each smart-meter.

The structure of XML files produced by smart-meters, has to be predefined, so as to be processed.

The least information needed in the file, is the value of energy consumption measurement taken by a sensor, and the sensor's identification or serial number, so that mapping of energy consumption - from which sensor, thus from which machine, this measurement is taken - can be performed. A proposed XML file structure based on [3] can be found in the Appendix A.

The smart-meter approach tackles interoperability issues in a plug-and-play way, which appears as a very flexible and user friendly approach. Figure 3 describes how smart-meters are pluged in a workstation, machine or switch-gear and how data are sent over a network to the second layer.

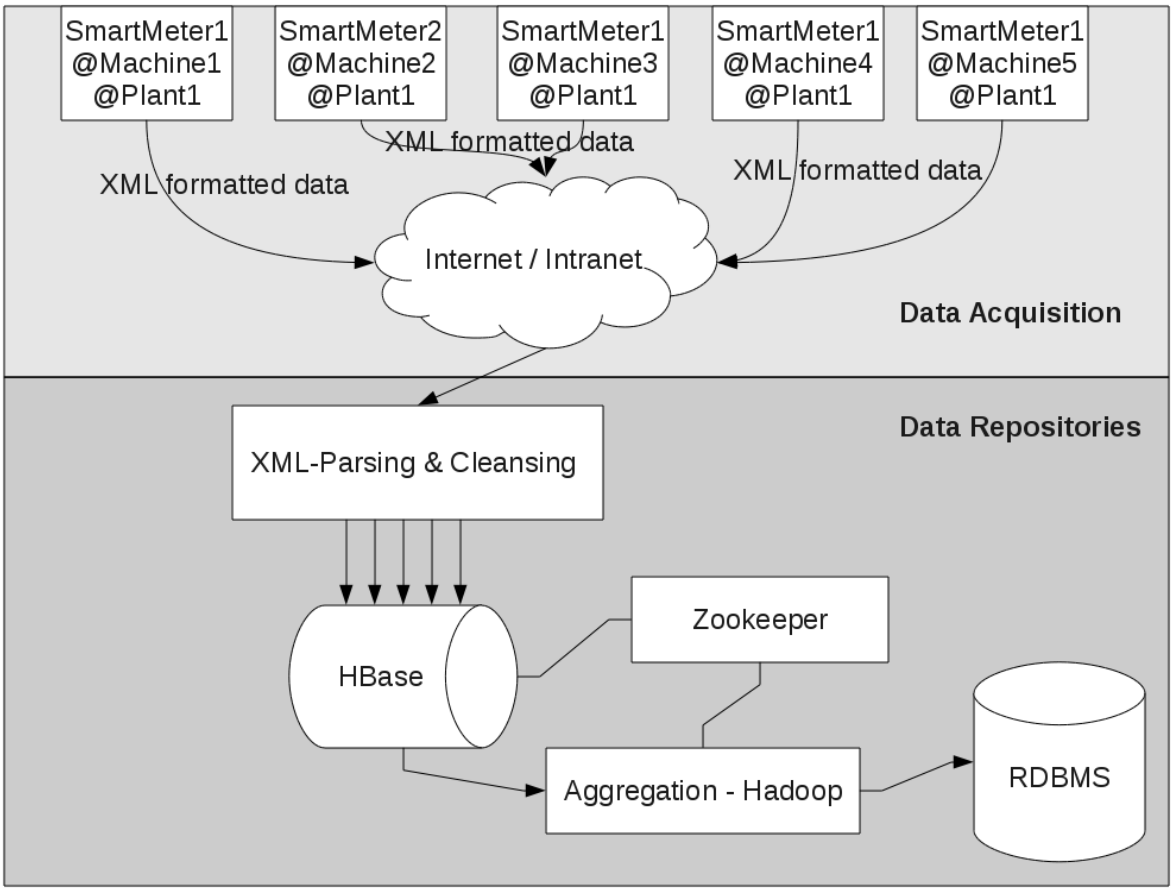
Figure 3: System Architecture - Data Acquisition



4 Data Repositories

In this section the Data Repositories layer is being described in detail. Figure 4 describes how this layer collaborates with the first layer, Data Acquisition.

Figure 4: System Architecture - Data Acquisition, Data Repositories



Data Repositories layer is consisted of five elements: XML-Parsing and Cleansing, a server responsible for listening to the XML data stream from smart-meter clients, secondly HBase [14], a distributed database, responsible to store temporarily real time energy data. The third module is Hadoop [13], which performs distributed aggregation of real-time data stored in HBase. In order to keep HBase and Hadoop working efficiently a fourth module, Zookeeper [16], [27] , is needed responsible for distributed coordination.

Each module of Data repositories Layer will be described in detail in a separate subsection.

4.1 XML Parsing and Cleansing

This module is a server application responsible for listening to the XML data stream from smart-meter clients. A server model, both time and resources efficientt, of responding to multiple requests is the multithreaded, which offers the ability of processing data, in this case XML documents, individually and concurrently. Possible errors within the XML data-stream may occur, such as arrivals of corrupted XML documents or false energy consumption values due to traffic in the network or smart-meter malfunctions. These issues have to be dealt with a cleansing method - another open case for future system releases.

This module is developed in Java programming language and the code can be found in the appendix B.

4.2 Data-Stream Management Systems

There are several stream processing systems; we indicatively discuss COSTES[5] and STREAM [1].

COSTES was designed and developed to efficiently perform complex real-time analysis on top of streams of AutoID events. In [5], is also proposed a set of linguistic requirements for RFID data management (RFDM) queries and describe an SQL-like construct to address these [5]. The AutoID domain requires high scalability, therefore COSTES meets with this requirement.

STREAM stands for STanford stREam datA Manager. It is another data-stream management system developed for applications such as network monitoring, telecommunications data management, clickstream monitoring, manufacturing, sensor networks, and others, data takes the form of continuous data streams rather than finite stored data sets, and clients require long-running continuous queries as opposed to one-time queries [1] .

Nevertheless, a solution based on distributed systems appeals more challenging where scalability issues are dealt efficiently.

4.3 HBase

HBase [14] is an open source, distributed, versioned, column-oriented store modeled after Google's Bigtable [4]. It is a distributed storage system higly scalable, while at the same time by allowing

real-time read write access [14], [4], it offers high performance and availability making real-time monitoring feasible.

HBase is not a traditional database. According to [4] a table in Bigtable, in HBase respectively, is a 'sparse, distributed, persistent multidimensional sorted map. Data is organized into three dimensions, rows, columns and timestamps. A cell is the storage referenced by a particular row key, column key, and timestamp'. In this case row keys are smart-meter ids or serial numbers - values included in the XML file, the column key is single, the name of the column, containing arriving energy consumption measurements and timestamp the moment of data arrival in milliseconds.

Once arriving data are parsed and cleaned, temporarily storage in HBase takes place. Real-time monitoring depends on HBase since new incoming energy consumption information is stored in it and can be read from it. Figure 5 displays an example of real-time data stored in the HBase table.

Figure 5: Data Repositories - HBase Table

SensorID	Timestamp	Value
Sensor1	1234533	0.1
Sensor2	1234567	0.2
Sensor3	1234589	0.1
Sensor1	1234591	0.2
Sensor3	1234599	0.3
Sensor2	1234605	0.1
Sensor2	1234606	0.2
Sensor3	1234608	0.1
Sensor1	1234620	0.2
Sensor2	1234626	0.3
Sensor3	1234628	0.1
Sensor2	1234632	0.2
Sensor1	1234635	0.1
Sensor3	1234646	0.2
Sensor1	1234548	0.3
....	....	....
....	....	....
....	....	....
....	....	....
....	....	....

4.4 Hadoop - Aggregation

The aggregation module is based on Hadoop [13]. The Apache Hadoop software library is a framework that allows distributed processing of large data sets across clusters of computers using a simple programming model, Map-Reduce [11]. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-avaiability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-availabile service on top of a cluster of computers, each of which may be prone to failures. Hadoop is a very efficient and therefore popular software, currently used by various big-data owners such as Facebook, Yahoo, Amazon etc. [13].

Hadoop's Distributed File System (HDFS) is the primary storage system used by Hadoop applications. HDFS creates multiple replicas of data blocks and distributes them on compute nodes throughout a cluster to enable reliable, extremely rapid computations [13], [25]. On top of the

HDFS stands and works HBase [14], where data are kept structured in a very large column oriented table. Aggregation is based on the Map-Reduce programming model and is a scheduled procedure that takes place periodically. After aggregation is finished data are stored in the RDBMS and the HBase table is flushed. Synchronization issues, such as flushing of the table and new arriving data are dealt with the use of two tables used alternately, where one table is used for storing and the other for aggregating.

Figure 6 displays an HBase Table after aggregation of data per sensorID in figure 5.

Figure 6: Data Repositories - HBase Table after aggregation

SensorID	Timestamp	Value
Sensor1	1234533	0.9
Sensor2	1234567	1.0
Sensor3	1234589	0.8
....	....	....
....	....	....
....	....	....
....	....	....
....	....	....

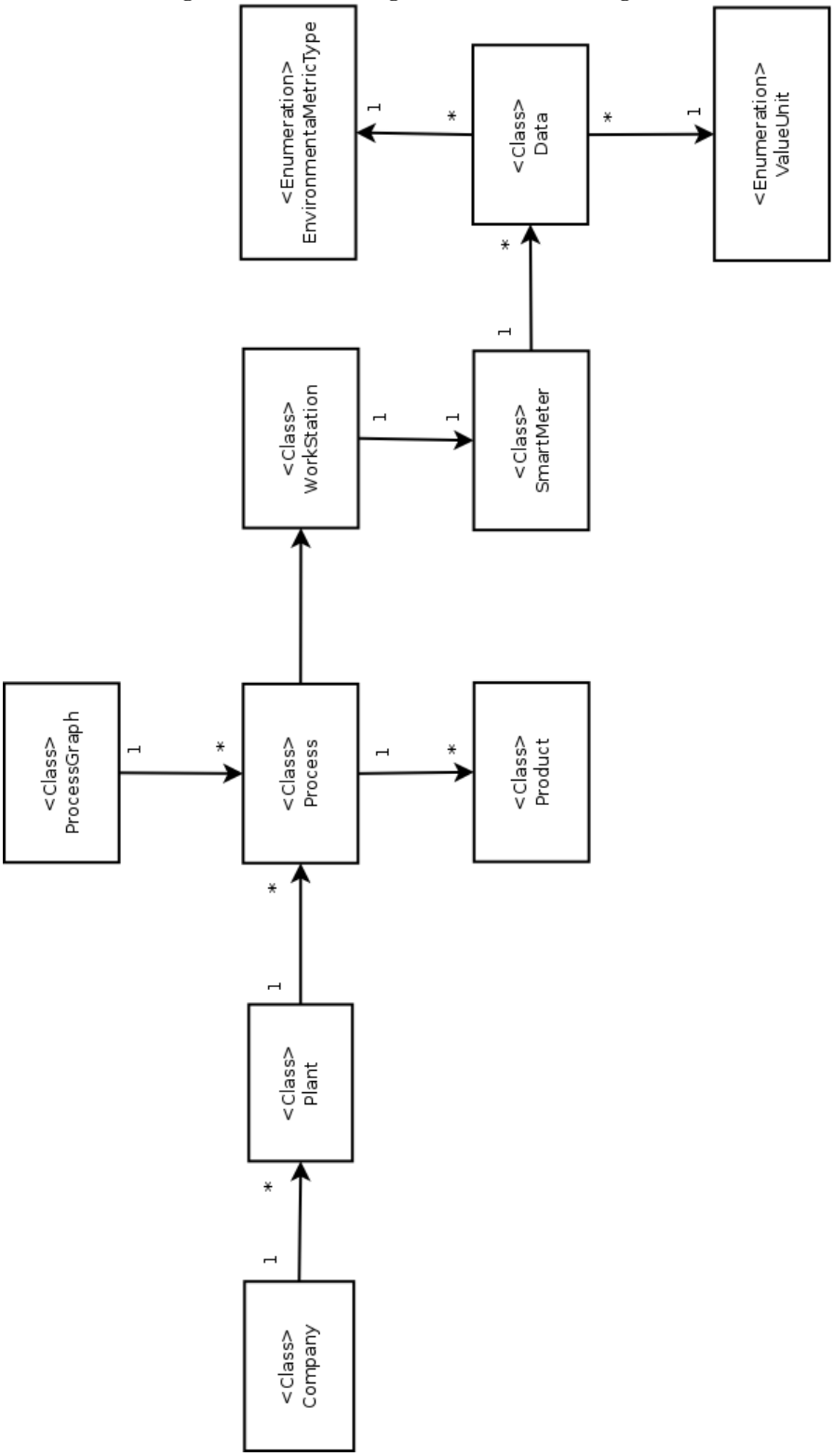
4.5 Zookeeper - Distributed Coordination

ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. All of these kinds of services are used in some form or another by distributed applications [27], [16]. This module is responsible for coordination of both distributed modules in this architecture, HBase and Hadoop.

4.6 Relational Database Management System

The question addressed here is how energy consumption information can be mapped on products, workstations, processes, plants etc. Based on the requirements analysis described in Section 2 figure 7 is a Unified Modeling Language (UML) domain diagram describing entities - classes and relationships between them. The passage from UML domain diagram to Entity - Relationship diagram

Figure 7: UML Diagram - Domain Diagram



Each class is described as follows.

Company: Depicts every company in a specific industry.

Plant: Depicts production or working plants owned by companies.

Process: Any process that takes place within a company's plant. It can be either production or

ProcessGraph: A set of processes. Depicts how processes are related.

Product: Any product that can be produced from a process.

WorkStation: Anything that consumes electrical energy and employees use it during working hours.

SmartMeter: Smart-meters that collect and send over the internet energy consumption measurements

Data: Energy consumption measurements collected by smart-meters

EnvironmentalMetricType: Multiple variables participate in the construction of total environmental harm done by a single process in a supply chain, such as energy consumption, carbon emissions, water and waste management etc. This enumeration allows future releases to take into concideration these metrics.

5 Implementation and Performance Evaluation

5.1 Implementation

In this section is described the implementation of the system including access to a cluster of machines online, operating systems, development/runtime platforms and Hadoop, HBase and Zookeeper versions.

The Greek Research and Technology Network (GRNET) [23] has provided access to Okeanos cloud [6] for the implementation of this system in a cluster of machines. Okeanos is an Infrastructure as a Service - service, that allows to users to create as many virtual machines as they want with any operating system they want. In this case was given access to a 3-machine cluster.

The specifications of each machine in the 3-server cluster are described as follows:

- CPUs: 4
- RAM (MB): 4096
- System Disk (GB): 20
- Operating System: Linux Debian 6.0 x64 Architecture

The Java Runtime Environment installed in each machine of the cluster and the Java Development Kit under which the system has been developed were

- Java(TM) SE Runtime Environment 1.6.0
- Java(TM) Platform, Standard Edition Development Kit 6

The version of Apache Hadoop, HBase and Zookeper were,

- Apache Hadoop 0.20.205
- Apache HBase 0.90.4
- Apache Zookeper 3.4.2

5.2 Performance Evaluation

In this section is described the performance and durability testing scenario with the results as well. The tests carried out for performance and durability issues were 30-minutes in duration. The included hardware devices were the 3-server cluster, described in the previous section, and 5 personal computers simulating production plants with smart-meters installed in them. Every 5 minutes smart-meters within a "plant" are doubled so that the crashing point of performance can be found quite fast.

The smart-meter simulation is a simple Java program that takes as input the number of smart-meters n, creates n threads - one for every smart-meter - in random time slots, varying from 1 to 10 seconds, and every k seconds each thread sends a random value as an XML document in the data-stream listening server. At the time where a value is being sent over the network to the data-stream server, the thread is not shut down, so that, if a failure occurs, the sum of failing "smart-meters" can be identified.

There were three 30-minutes tests conducted, in which the starting number of sensors where 700. A varying factor was the measurement rate from the smart-meters. In the first test a "smart-meter" sends a value every 2 seconds, in the second every 5 seconds, and in the third every 10 seconds.

The network characteristics, where the 5 personal computers - "plants" were installed, as measured by a simple online speed-test are described as follows:

- download speed: 12 Mbps
- upload speed: 10 Mbps

A results summary from these three tests are described bellow, where the total number of alive "smart-meters" in the end of each test, the total number of measurements by "smart-meters", and the aggregation time are shown. Detailed aggregation results and Hadoop's Map-Reduce output can be found in Appendix C.

Table 1: Performance Evaluation Test 1 - Sensors Test Results

Variable	Value
Test duration	30 minutes
Measurement Rate	1 value/2sec
Sensors at start	700
Sensors at end	359
Total records	436997

Table 2: Performance Evaluation Test 1 - Aggregation Results

Variable	Value
Total records	436997
Aggregation time	58 seconds
Configured Capacity	59.06 GB
Distributed File System used	297.2 MB
Non Distributed File System Used	17.64 GB
Distributed File System Used%	0.48%
Distributed File System Remaining%	69.1%
CPU time spent(ms)	16830

Table 3: Performance Evaluation Test 2 - Sensors Test Results

Variable	Value
Test duration	30 minutes
Measurement Rate	1 value/5sec
Sensors at start	700
Sensors at end	1730
Total records	437553

Table 4: Performance Evaluation Test 2 - Aggregation Results

Variable	Value
Total records	437553
Aggregation time	64 seconds
Configured Capacity	59.06 GB
Distributed File System used	297.65 MB
Non Distributed File System Used	17.64 GB
Distributed File System Used%	0.49%
Distributed File System Remaining%	69.1%
CPU time spent(ms)	17100

Table 5: Performance Evaluation Test 3 - Sensors Test Results

Variable	Value
Test duration	30 minutes
Measurement Rate	1 value/10sec
Sensors at start	700
Sensors at end	2794
Total records	323551

Table 6: Performance Evaluation Test 2 - Aggregation Results

Variable	Value
Total records	323551
Total aggregation time	53 seconds
Configured Capacity	59.06 GB
Distributed File System used	102.95 MB
Non Distributed File System Used	17.73 GB
Distributed File System Used%	0.49%
Distributed File System Remaining%	69.81%
CPU time spent(ms)	13940

6 Alternative Approaches

This section describes alternative approaches of the repositories layer tested, before the described approach in the previous section has been selected. The modules that alter in the following approaches are the XML-Parsing and cleansing and aggregation modules. Nevertheless, the RDBMS module remains the same since information of energy consumption and how it is mapped upon products or processes is irrelevant with how it is aggregated.

Common ground for these three approaches is that in each of them, XML document structure has to be predefined so that an XML parser can be developed to process incoming XML formated energy consumption measurements. XML document format compatibility issue is tackled via the proposal of the smart-meter able to format energy consumption information in a specific XML format which can be found in the appendix A.

Two similar approaches will be described in detail in the following subsections.

6.1 Listen - Immediately Aggregate - Store

This section describes the first approach tested. The basic server function remains the same, listening to the stream of energy information and sending it directly for aggregation (online aggregation). As usual energy consumption is aggregated per smart-meter. After this step, aggregated data are stored in the RDBMS.

Traditional Apache Hadoop is incapable of performing online aggregation. It is able to aggregate big chunks of data, not stream flows [13], [2]. [10] suggests a Hadoop Online Prototype (HOP), which enables Map-Reduce programs to be written for applications such as event monitoring and stream processing. This seems to be quite the solution to a problem such as monitoring huge data streams. The installation of HOP-Hadoop is identical with traditional Hadoop. The key point that allows online aggregation is that, results can be taken from intermediate spill files before aggregation process is finished [10]. Consequently, monitoring is done by reading periodically these spill files.

The problem with this technique is that HOP, according to [10], is a prototype, therefore not recommended for production use yet. Possibly, next versions of HOP might be functional enough. Source code of this project and documentation can be found at [9].

6.2 Listen - Aggregate periodically - Store

The second technique uses traditional Hadoop, therefore data aggregation cannot take place immediately. The main idea is, receive data (XML files), store them into the HDFS file system, and periodically aggregate them. XML files are not parsed as they arrive, but during the map function execution so that this work can be dispensed to many machines as well.

Hadoop provides a tool that allows file parsing, named Hadoop Streaming. It is a tool that can be configured to treat file lines - the input for map function differently. Nevertheless, a different and more efficient technique has been used. According to [22], a class from a different, yet related with Hadoop, project named Mahout [18], can be injected into the Hadoop core jar file, and used to parse XML files. The output of map-reduce processing is a big file with comma separated aggregated values of energy data. Aggregation takes place periodically and is followed by data insertion into the relational database management system (RDBMS).

Data insertion into a RDBMS can be performed easily with an open Source tool provided by Cloudera [7] named SQOOP (SQl-HadOOP). This tool consumes comma separated values and inserts them with an SQL query to the appropriate table. The reverse work is feasible, but in this case is not needed. Unfortunately, SQOOP is available only with Cloudera's Hadoop distribution (CHD). CHD is free only for a 50-node cluster application. The number of nodes in the cluster depends on the size of data. In this case, a 4-8 node cluster (4 nodes is the minimum number of machines for Hadoop utilization [13], [7]) is sufficient enough to perform data aggregation.

An issue that rises in this implementation is that the shortest job in CHD takes 24 seconds to complete [7]. Assuming that XML parsing is not a serious complexity job, which is a very naive assumption, energy data can be viewed at least 24 seconds after their arrival. This means that approximately more than half a minute is needed to perform aggregation and view current energy consumption values. This is not a real-time approach but near-real-time.

Another issue is that this approach has no queue management of arriving data. This means that every next CHD execution across time, will have more data to aggregate and take more time for completion than the previous one. This leads to total chaos quite fast in an environment with a large number of smart-meters.

Due to these issues this implementation was abandoned. Nevertheless, with a queue management

sub-system it is suitable enough for environments where near-real-time approach is appropriate.

7 Data Processing Layer

Data processing layer includes real-time energy monitoring and optimization techniques towards cost reduction, demand satisfaction and energy consumption minimization.

Real-time energy monitoring considering the large volume of data is feasible via the high scalability and performance of the data repositories layer. In the following subsections will be be described the optimization problem and the mathematical modelization of it.

7.1 Problem Description

A manufacturer produces a wide variety of products. We are interested in the

- cost reduction
- demand satisfaction
- energy consumption minimization

of production.

There exists a single plant with one storage room, multiple machines - workstations and weekly time periods in which energy cost is variant. The machines - workstations have specific capacity, a constant production rate, a energy set-up cost per product, and a fixed set-up energy cost for all products. Each product has a different energy cost, and different time demand for completion on each machine. There also exists demand of products from customers which varies for each product and time period.

7.2 Mathematical Model

Each table bellow describes separately the Objects and Indices, Data, Variables, Constraints and the Objective Function needed to construct the model.

Table 7: Mathematical Model - Objects and Indices

Objects and Indices	Mathematical Notations
One Plant	————————
Product Families	————————
Individual Finished Products	Object: products
	Index: i=1,...,NI
Weekly time Periods	Object: periods
	Index: t=1,...,NT
Workstations	Object: workstation
	Index: k=1,...,NK
Storage resources	Single Storage Room Env. Friendly

Table 8: Mathematical Model - Data

Data	Mathematical Notations
Machine Capacity	for machine k: L_k
Constant production rate	for product i, machine k: a_{ik}
Energy cost per period, per product	for product i, period t: e_{it}
Inventory energy cost per period, per product	for product i, period t: inv_en_{it}
Set-up energy cost per product, per machine	for product i, machine k: fix_en_{ik}
Fixed set-up energy cost per machine	for machine k: $mfix\_en_k$
Demand	for product i, period t: d_{it}
Time demand per product, per machine	for product i, period t, machine k: z_{itk} per period

Table 9: Mathematical Model - Variables

Variables	Mathematical Notations
Batch size	for product i, period t: $x_{it} \geq 0$
Production set-up	for product i, period t, machine k: $y_{itk} \in \{0, 1\}$
Machine set-up	for period t, machine k: $w_{tk} \in \{0, 1\}$
Storage	for product i, period t: $s_{it} \geq 0$

Table 10: Mathematical Model - Constraints

Constraints	Mathematical Notations
Capacity restriction	for product i, machine k: cap_res_{ik}
Demand satisfaction	for product i, period t: dem_sat_{it}
Real binary relation 1	for product i, period t, machine k: $rel1_{itk}$
Real binary relation 2	for product i, period t, machine k: $rel2_{itk}$
Batch size satisfaction	for product i, period t, machine k:$bsize\_sat_{itk}$

Table 11: Mathematical Model - Objective Function

Objective Function	Mathematical Notations
Minimize energy cost	$energy$

The constraints and the objective function are defined bellow:

$dem\_sat_{itk} := x_{it} + s_{it-1} = d_{it} + s_{it}$

$bsize\_sat_{itk} := \sum_{k}^{NK} z_{itk} \cdot a_{ik} = x_{it}$

$cap\_res_{ik} := \sum_{i}^{NI} z_{itk} \leq L_k$

$rel1_{itk} := z_{itk} \leq y_{itk} \cdot M$

$rel2_{itk} := z_{itk} \leq w_{tk} \cdot M$

$min\{energy := \sum_{it}^{NI,NT} x_{it} \cdot e_{it} + \sum_{itk}^{NI,NT,NK} y_{itk} \cdot fix\_en_{ik} + \sum_{tk}^{NT,NK} w_{tk} \cdot mfix\_en_k + \sum_{it}^{NI,NT} s_{it} \cdot inv\_en_{it}\}$

Assuming that energy cost is minimized a boundary for energy consumption is not needed.

8 Implications for future Research

8.1 Security issues

A very important implication for future research and development of the proposed system seems to be issues considering privacy and sharing of energy consumption scattered upon computer networks. A part of the proposed system allows to users-partners within an industry, to have a look on energy consumption per process from other partners in the same industry. This kind of information can reveal a part of production cost, information that is very important to competitors.

By no means this kind of information should be reachable from malicious users or used for competition purposes. Therefore a security schema for information sharing and privacy is needed. Considering that too much security is bad security, this schema should treat these issues in a flexible way. Besides, the goal is to contribute to more environmental efficient supply chains through information sharing not to another highly secure monitoring system. For example confidentiality of energy consumption information could be defined by the user, and shared in a way of a trading protocol, e.g. a partner provides information of energy efficiency of a process only when asked and receives information for a set of processes being run by the asking partner.

8.2 Data Capturing Layer Expandability

Considering that multiple variables participate in the construction of total environmental harm done by a single process in a supply chain, such as energy consumption, carbon emissions, water and waste management etc. the data capturing layer should take into consideration these factors too. In this case the main metric chosen from this very large spectrum of efficiency indicies is energy consumption, thus hardware devices, including sensors and smart-meters, selected for the data capturing layer implementation measure energy consumption.

Consequently, if environmental impact is not examined only from the energy consumption point of view, the infrastructure of the data processing layer should be expanded so that several metrics mentioned above can participate in the environmental impact measurement. This includes not only sensors that can measure different metrics, but smart-meters that can communicate with variable types of sensors as well.

Data capturing layer expandability is a very important factor for data processing layer expansion towards optimal decisions with respect to several efficiency indices, as described in Section 8.4.

8.3 System Sustainability

Another implication for future development is the system sustainability towards the data acquisition and repositories layer. This issue is divided to three main factors:

1. XML cleansing module
2. failing sensors and smart-meters detection and reporting
3. server failure detection, reporting and recovery.

Both XML cleansing, and failing sensors and smart-meters detection and reporting, are parts of the data acquisition layer and secure the accuracy and correctness of energy consumption data. XML cleansing is the module that corrects XML documents that may arrive corrupted or contain false energy consumption values due to traffic in the network or smart-meters and sensors malfunctions.

A smart-meter monitoring module is needed due to the importance of detection and reporting of failing sensors and smart-meters in real-time. System users can view the failing sensors or smart-meters, thus replace them with new ones.

Additionally, server failure detection, reporting and recovery seems to be highly important, because the ensurance of normal XML-Parsing and cleansing resver function, allows arriving data to be stored, thus processed. Consequently, if XML-Parsing and Cleansing server fails for any possible reason, it should recover as fast as possible, so that the system can continue its operation. Hence a server monitoring module is needed, that can detect and report malfunctions, and it also can recover the server's function from the exact point of failure.

8.4 Data Processing Layer Expandability

As mentioned in Sections 1.1 and 8.2, environmental impact is definitely not a "single variable function". Multiple variables participate in the construction of total environmental harm done by

a single process in a supply chain, such as energy consumption, carbon emissions, water and waste management etc.

Consequently, a production planning model is needed, which takes into consideration multiple environmental efficiency indices, such as the above, and provides optimal decisions towards environmental efficiency and cost reduction. This is certainly a great implication for future research due to the large spectrum of environmental efficiency indices that apply on several industries.

This idea certainly includes first the definition of environmental efficiency indices upon industries and secondly a production planning model upon each industry that provides optimal decisions based on these indices.

A Energy XML Document Structure

```
<?xml version="1.0" encoding="UTF-8"?>
  <data>
    <timestamp>
      2012-02-07 15:25:23.004
    </timestamp>
    <identifier>
      6622224741725267708DEADB
    </identifier>
    <value>
      0.30308743802077
    </value>
  </data>
```

B Java Code

B.1 Package: core

```
/**
* Represents energy Consumption measurements taken by smart-meters.
* @author Stathis Plitsos, stathis.plitsos[at]sgmail.com
*/
public class Data implements Serializable{

/** Constructor setting all private fields to custom values*/
public Data(EnvironmentalMetricType environmentalMetricType,
Date timestamp, SmartMeter smartmeter, float value,
ValueUnit valueUnit)

/** Constructor setting private fields to default values*/
public Data()

/**
 * Getter for the private field environmental metric type
 *
 * @return Returns the environmental metric type of measurement*/
public EnvironmentalMetricType getEnvironmentalMetricType()

/**
 * Setter for private field environmental metric type
*/
public void setEnvironmentalMetricType(
EnvironmentalMetricType environmentalMetricType)

/** Getter for the private field timestamp
 *
 * @return Returns the timestamp of the measurement*/
public Date getTimestamp()

      /**
* Setter for the private field environmental metric type
*/
public void setTimestamp(Date currentTimestamp)

/** Getter for the private field smart-meter
 *
 * @return Returns the smart-meter that took this measurement */
public SmartMeter getSmartmeter()

/**
 * Setter for the private field smart-meter
 */
public void setSmartmeter(SmartMeter smartmeter)


/** Getter for private field value
 *
 * @return Returns the measurement value*/


/**
 * Setter for the private field value
```

```
*/
public void setValue(float value)


/**Getter for private field value unit
 *
 * @return Returns the measurement unit of this object*/
public ValueUnit getValueUnit()


/**
 * Setter for the private field valueUnit
*/
public void setValueUnit(ValueUnit valueUnit)

/**Getter for private field id
 *
 * @return Returns the auto assigned id number*/
public int getId()


/**
 * Setter for the private field id
*/
public void setId(int id)
}

////////////////////////////////////////////////////////////////////////////////

/**
 * Enumeration of possible environmental metrics
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 * */
public enum EnvironmentalMetricType {}

/**
 * Class representing a plant of a company across the supply chain.
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 */

////////////////////////////////////////////////////////////////////////////////

public class Plant implements Serializable{

/** Constructor setting all private fields to custom values.*/
public Plant(List<Process> process, String name, Integer id)


/** Constructor setting private fields to default values.*/
public Plant()

/** Getter for the private field processes.
 *
 * @return Returns a list of processes conducted in the specific plant.*/
public List<Process> getProcess()

/**
 * Setter for private field processes - a list of processes conducted in the specific plant.
*/
public void setProcess(List<Process> process)
```

```
/** Getter for the private field name.
 *
 * @return Returns the plant's identification name.*/
public String getName()

/**
 * Setter for private field name - the plant's identification name.
 */
public void setName(String name)

/** Getter for the private id.
 *
 * @return Returns the plant's identification number.*/
public Integer getId()

/**
 * Setter for private field id - the plant's identification number.
*/
public void setId(Integer id)

/**
 * Adds a process to the list of processes conducted in the specific plant.
 * @param p Object of process
 */
public void addProcess(Process p)
}

////////////////////////////////////////////////////////////////////////////////

/**
 * Class representing a process which takes place in a plant.
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 */
public class Process implements Serializable{

/**
 * Constructor setting all private fields to custom values.
 */
public Process(List<Product> product, List<WorkStation> workStation,
String name, String description, Integer id,
ProcessGraph processgraph)

/**
 * Constructor setting all private fields to default values.
 */
public Process()

/**
 * Constructor setting all private fields to default values.
 */
public Process()

/**
 * Setter of the private field products - a list of products related with this process.
 * @param products A list of products.
 */
public void setProduct(List<Product> product)

/**
```

```
 * Getter of the private field workStations - a list of work-stations related with this process.
 * @return A list of work-stations.
 */
public List<WorkStation> getWorkStation()

/**
 * Setter of the private field workStations - a list of work-stations related with this process.
 * @param workStations A list of workStations.
 */
public void setWorkStation(List<WorkStation> workStation)

/**
 * Getter of the private field name - the name of the process.
 * @return The name of the process.
 */
public String getName()

/**
 * Setter of the private field name - the name of the process.
 * @param name The name of the process.
 */
public void setName(String name)

/**
 * Getter of the private field description - a short description of what this process does.
 * @return The process description.
 */
public String getDescription()

/**
 * Setter of the private field description - a short description of what this process does.
 * @param description The process description.
 */
public void setDescription(String description)

/**
 * Getter of the private field id -The identification number of the process.
 * @return The identification number of the process.
 */
public Integer getId()

/**
 * Setter of the private field id - The identification number of the process.
 * @param id The identification number of the process.
 */
public void setId(Integer id)

/**
 * Getter of the private field processGraph - The graph of processes in which
 * this process exists.
 * @return The graph of processes in which this process exists.
 */
public ProcessGraph getProcessgraph()

/**
 * Setter of the private field processGraph - The graph of processes in which
 * this process exists.
 * @param processgraph The graph of processes in which this process exists.
 */
public void setProcessgraph(ProcessGraph processgraph)
```

```
/**
 * Getter of the private field plant - where this process takes place.
 * @return Plant of this process.
 */
public Plant getPlant()

/**
 * Setter of the private field plant - where this process takes place.
 * @param plant Plant of this process.
 */
public void setPlant(Plant plant)

/**
 * Adds a workstation to the workstations list related with this process
 * @param workstation the new workstation to be added.
 */
public void addWorkStation(WorkStation w)

/**
 * Adds a product to the products list related with this process
 * @param product the new product to be added.
 */
public void addProduct(Product p)
}

////////////////////////////////////////////////////////////////////////////
/**
 * Class representing a graph of related processes.
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 *
 */
public class ProcessGraph implements Serializable{

/**
 * Constructor setting private fields to default values.
 */
public ProcessGraph()

/**
 * Constructor setting private fields to default values.
 */
public ProcessGraph(List<Process> process, Integer id)

/**
 * Getter of the private field processes.
 * @return The list with processes from which this graph is consisted of.
 */
public List<Process> getProcess()

/**
 * Setter of the private field processes.
 * @param process The list with processes from which this graph is consisted of.
 */
public void setProcess(List<Process> process)

/**
 * Getter of the private field id - the process graph identification number.
 * @return The process graph identification number.
 */
```

```
public Integer getId()

/**
 * Setter of the private field id - the process graph identification number.
 * @param id The process graph identification number.
 */
public void setId(Integer id)

/**
 * Adds a process to the graph of related processes.
 * @param p The new process to be added.
 */
public void addProcess(Process p)
}

////////////////////////////////////////////////////////////////////////////////

 * Class representing a product.
 * @author{Stathis Plitsos, stathis.plitsos[at]gmail.com}
public class Product implements Serializable{
/**
 * Constructor setting private fields to default values.
 */
public Product()

/**
 * Constructor setting private fields to custom values.
 */
public Product(String name, Integer id, String description)

/**
 * Getter of the private field name - the products name.
 * @return The name of the product.
 */
public String getName()

/**
 * Setter of the private field name - the products name.
 * @param name The name of the product.
 */
public void setName(String name)

/**
 * Getter of the private field id - the products identification number.
 * @return The products identification number.
 */
public Integer getId()

/**
 * Setter of the private field id - the products identification number.
 * @param id The product's identification number.
 */
public void setId(Integer id)

/**
 * Getter of the private field description - a short description of the product.
 * @return The description of the product.
 */
public String getDescription()
```

```
/**
 * Setter of the private field description - a short description of the product.
 * @param description The description of the product.
 */
public void setDescription(String description)

/**
 * Getter of the private field process - the process related with this product.
 * @return The process related with this product.
 */
public Process getProcess()

/**
 * Setter of the private field process - the process related with this product.
 * @param process the process related with this product.
 */
public void setProcess(Process process)
}

////////////////////////////////////////////////////////////////////////////////

/**
 * Class representing a smart-meter which collects energy information data from workstations
 *  in a plant.
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 */
public class SmartMeter implements Serializable {

/**
 * Constructor setting private fields to default values.
 */
public SmartMeter()

/**
 * Constructor setting private fields to custom values.
 */
public SmartMeter(List<Data> environmentalMetricData, Integer id,
WorkStation workstation, Integer timeWindow, TimeUnit timeUnit)

/**
 * Getter of private field environmentalMetricData - list of measurements taken
 * by this smart-meter.
 * @return A list of Data - measurements taken by this smart-meter.
 */
public List<Data> getEnvironmentalMetricData()

/**
 * Setter of private field environmentalMetricData - list of measurements taken
 * by this smart-meter.
 * @param environmentalMetricData A list of Data - measurements taken by this smart-meter.
 */
public void setEnvironmentalMetricData(List<Data> environmentalMetricData)

/**
 * Getter of the private field id - the identification number of the smart-meter.
 * @return The identification number of the smart-meter.
 */
public Integer getId()

/**
```

```
 * Setter of the private field id - the identification number of the smart-meter.
 * @param id The identification number of the smart-meter.
 */
public void setId(Integer id)

/**
 * Getter of the private field workstation - the workstation from which measurements are taken.
 * @return The workstation from which measurements are taken.
 */
public WorkStation getWorkstation()

/**
 * Setter of the private field workstation - the workstation from which measurements are taken.
 * @param workstation The workstation from which measurements are taken.
 */
public void setWorkstation(WorkStation workstation)

/**
 * Getter of the private field timeWindow - the frequency of measurements.
 * @return The frequency of measurements.
 */
public Integer getTimeWindow()

/**
 * Setter of the private field timeWindow - the frequency of measurements.
 * @param timeWindow The frequency of measurements.
 */
public void setTimeWindow(Integer timeWindow)

public TimeUnit getTimeUnit()

public void setTimeUnit(TimeUnit timeUnit)

/**
 * Adds a measurement to the measurements List.
 * @param d New data - measurement taken by smart-meter.
 */
public void addData(Data d)

/**
 * Getter of the private field serialNumber - the serial number of the smart-meter.
 * @return The serial number of the smart-meter.
 */
public String getSerialNumber()

/**
 * Setter of the private field serialNumber - the serial number of the smart-meter.
 * @param serialNumber The serial number of the smart-meter.
 */
public void setSerialNumber(String serialNumber)
}

////////////////////////////////////////////////////////////////////////////////

/**
 * Enumeration of possible measurements units of environmental metrics
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 * */
public enum ValueUnit {}
```

```
////////////////////////////////////////////////////////////////////////////////

/**
 * Class representing a workstation on which products are produced
 *  in a plant.
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 */
public class WorkStation implements Serializable{
/**
 * Constructor setting private fields to default values.
 */
public WorkStation() {
super();
}

/**
 * Constructor setting private fields to custom values.
 */
public WorkStation(List<SmartMeter> sensor, String name, String description,
Integer id, Process process)

/**
 * Getter of private field sensors - smart-meters that measure a specific environmental metric.
 * @return List of smart-meters attached to the workstation.
 */
public List<SmartMeter> getSensor()

/**
 * Setter of private field sensors - smart-meters that measure a specific environmental metric.
 * @param sensors List of smart-meters attached to the workstation.
 */
public void setSensor(List<SmartMeter> sensor)

/**
 * Getter of the private field name - the name/machineType of the workstation.
 * @return The name/machineType of the workstation.
 */
public String getName()

/**
 * Setter of the private field name - the name/machineType of the workstation.
 * @param name The name/machineType of the workstation.
 */
public void setName(String name)

/**
 * Setter of the private field description - a short description of the workstation.
 * @return A short description of the workstation.
 */
public String getDescription()

/**
 * Getter of the private field description - a short description of the workstation.
 * @param description A short description of the workstation.
 */
public void setDescription(String description)

/**
 * Getter of the private field id - the identification number of the workstation.
 * @return The identification number of the workstation.
```

```
 */
public Integer getId()

/**
 * Setter of the private field id - the identification number of the workstation.
 * @param id The identification number of the workstation.
 */
public void setId(Integer id)

/**
 * Getter of the private field process - the process related with this workstation.
 * @return The process related with this workstation.
 */
public Process getProcess()

/**
 * Setter of the private field process - the process related with this workstation.
 * @param process The process related with this workstation.
 */
public void setProcess(Process process)

/**
 * Adds a smart-meter to the smart-meter list related with this workstation
 * @param smartMeter the new smart-meter to be added.
 */
public void addSmartMeter(SmartMeter s)
}
```

B.2 Package: dao

```
/**
 * Abstract Data Acces Object(DAO) Factory.
 * The implemented class is defined by system's properties.
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 *
 */
public abstract class DAOFactory {
    /**
     * Returns the factory for the DAO Objects creation.
     * @return The factory for the DAO Objects creation.
     */
    public static DAOFactory getFactory() ;

    static void setStub(DAOFactory afactory);

    static void reset();

    /**
     * Returns the object for the interface DataDAO {@link DataDAO}.
     * @return The DAO object.
     */
    public abstract DataDAO getDataDAO();

    /**
     * Returns the object for the interface PlantDAO {@link PlantDAO}.
     * @return The DAO object.
```

```
     */
    public abstract PlantDAO getPlantDAO();

    /**
     * Returns the object for the interface ProcessDAO {@link ProcessDAO}.
     * @return The DAO object.
     */
    public abstract ProcessDAO getProcessDAO();

    /**
     * Returns the object for the interface ProcessGraphDAO {@link ProcessGraphDAO}.
     * @return The DAO object.
     */
    public abstract ProcessGraphDAO getProcessGraphDAO();

    /**
     * Returns the object for the interface ProductDAO {@link ProductDAO}.
     * @return The DAO object.
     */
    public abstract ProductDAO getProductDAO();

    /**
     * Returns the object for the interface SmartMeterDAO {@link SmartMeterDAO}.
     * @return The DAO object.
     */
    public abstract SmartMeterDAO getSmartMeterDAO();

    /**
     * Returns the object for the interface WorkStationDAO {@link WorkStationDAO}.
     * @return The DAO object.
     */
    public abstract WorkStationDAO getWorkStationDAO();

}


////////////////////////////////////////////////////////////////////////////////

/**
* Interface for the DataDAO Object.
* @author Stathis Plitsos, stathis.plitsos[at]gmail.com
*/
public interface DataDAO extends GenericDAO<Data> {

}

////////////////////////////////////////////////////////////////////////////////


/**
 * General purpose interafce for basic data access actions of DAO Objeccts.
 * Used for acces of DAOs from external data sources (e.g. databases) .
 * It is a parametric interface where the parameter is the class for which the
 * data access methods are applied.
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 * @param <T> The class for which the
 * data access methods are applied.
 */

public interface GenericDAO<T> {
```

```
    /**
     * Returns all objects from the external data source.
     * @return A list with all objects.
     */
    List<T> findAll();

    /**
     * Stores an object in the external data source.
     * The new object may be a new object that does not exist in the data source
 * or one that already exists, thus its state is updated.
     * @param entity The object which state is stored in the external data source.
     */
    void save(T entity);

    /**
     * Deletes the object from the external data source.
     * @param entity The object to be deleted..
     */
    void delete(T entity);
}

////////////////////////////////////////////////////////////////////////////////

/**
* Interface for the PlantDAO Object.
* @author Stathis Plitsos, stathis.plitsos[at]gmail.com
*/
public interface PlantDAO extends GenericDAO<Plant>{
Plant findByName(String name);

}

////////////////////////////////////////////////////////////////////////////////

/**
* Interface for the ProcessDAO Object.
* @author Stathis Plitsos, stathis.plitsos[at]gmail.com
*/
public interface ProcessDAO extends GenericDAO<Process>{

}

////////////////////////////////////////////////////////////////////////////////


/**
* Interface for the ProcessGraphDAO Object.
* @author Stathis Plitsos, stathis.plitsos[at]gmail.com
*/
public interface ProcessGraphDAO extends GenericDAO<ProcessGraph>{

}

////////////////////////////////////////////////////////////////////////////////

/**
* Interface for the ProductDAO Object.
* @author Stathis Plitsos, stathis.plitsos[at]gmail.com
*/
```

```
public interface ProductDAO extends GenericDAO<Product>{

}

////////////////////////////////////////////////////////////////////////////////


/**
* Interface for the SmartMeterDAO Object.
* @author Stathis Plitsos, stathis.plitsos[at]gmail.com
*/
public interface SmartMeterDAO extends GenericDAO<SmartMeter>{

}

////////////////////////////////////////////////////////////////////////////////

/**
* Interface for the WorkStationDAO Object.
* @author Stathis Plitsos, stathis.plitsos[at]gmail.com
*/
public interface WorkStationDAO extends GenericDAO<WorkStation>{

}
```

B.3 Package: fileManagement

```
/**
 * Class responsible for creating XML files with energy consumption measurements taken by
 * smart-meters.
 * This class is part of the smart-meter simulation.
 * @author Stathis Plitsos, stathis.plitsos@gmail.com
 */
public class XMLCreator {
 * Constructor setting private fields to custom values.
 * @param filename The name of the XML file to be created.
 * @param rvg Random value generator for smart-meter serial number and energy consumption value.
 * @throws IOException
 */
public XMLCreator(String filename, RandomValuesGenerator rvg) throws IOException

/**
 * Creates XML file with energy consumption values and smart-meter serial number.
 * @param flag If flag is set to true, then serial-number is set to "StathisTheGreat",
 * just a test value for monitoring of a specific serial number.
 * @throws IOException
 */
public void createXML(boolean flag) throws IOException

/**
 * Creates XML file with energy consumption values and smart-meter serial number.
 * @throws IOException
 */
public void createXML() throws IOException

/**
 * Tester main of XMLCreator functionality.
```

```
 * @param args
 */
public static void main(String args[])
}

////////////////////////////////////////////////////////////////////////////////

/**
 * Class responsible for parsing XML documents with energy consumption measurements taken by
 * smart-meters.
 * In case of a different XML document format this class MUST be changed, so that XML documents
 * can be correctly parsed.
 * @author Stathis Plitsos, stathis.plitsos@gmail.com
 */
public class XMLParser extends DefaultHandler{

/**
 * Constructor of the object setting private fields to default values.
 */
public XMLParser()

/**
 * Constructor creating a file with custom filename
 * @param aFileName - The name of the file to be created.
 */
public XMLParser(String aFileName)

/**
 * Parses xml documents
 * @param fileName The XML document to be parsed.
 * @throws SAXException
 * @throws IOException
 */
public void parseXMLFile(String fileName) throws SAXException, IOException

/**
 * While Parsing the XML file, if extra characters like space or enter Character
 * are encountered then this method is called. If you don't want to do anything
 * special with these characters, then you can normally leave this method blank.
 */
public void characters(char[] buffer, int start, int length)

/**
 * Called when the starting of the Element is reached. For Example if we have Tag
 * called <Title> ... </Title>, then this method is called when <Title> tag is
 * encountered while parsing the Current XML File. The AttributeList Parameter has
 * the list of all Attributes declared for the Current Element in the XML File.
 */
public void startElement (String uri, String name,String qName, Attributes atts)

/**
 * Called when the Ending of the current Element is reached. For example in the
 * above explanation, this method is called when </Title> tag is reached.
 */
public void endElement (String uri, String name, String qName)

/**
 * Getter of the private field data - the energy consumption measurement.
 * @return Energy consumption measurement
 */
```

```
public String getData()
/**
 * Getter of the private field sArray - a array with every valuable info within the
 * XML document
 * @return Array with every valuable info within the
 * XML document
 */
public String[] getsArray()

 /**
 * Setter of the private field sArray - a array with every valuable info within the
 * XML document
 * @param sArray Array with every valuable info within the
 * XML document
 */
public void setsArray(String[] sArray)

}
```

B.4 Package: gui

```
/**
 * Graphical User Interface of real-time monitoring of energy consumption.
 * @author Sathis Plitsos, stathis.plitsos[at]gmail.com
 */
public class RealTimeEnergyConsumption extends javax.swing.JFrame {

    /** Creates new form RealTimeEnergyConsumption */
    public RealTimeEnergyConsumption()

    /** This method is called from within the constructor to
     * initialize the form.
     */
    private void initComponents()

    /**
     * Updates image of chart with new values.
     * @param chart The new chart to be presented.
     */
    public void updateChart(JFreeChart chart)
}
```

B.5 Package: jpaDao

```
/**
 * Implemented Data Access Object DataDAO Class
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 *
 */
public class DataJpaDAO extends GenericJpaDAO<Data> implements DataDAO{
public List<Data> findAll()
}
```

```
////////////////////////////////////////////////////////////////////////////////

/**
 * Implementation of the basic data access of DAO Interfaces.
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 * @param <T> The class for which the
 * data access methods are applied.
 */
abstract class GenericJpaDAO<T> implements GenericDAO<T> {

    public void delete(T entity)

    public void save(T entity)
}

////////////////////////////////////////////////////////////////////////////////

/**
 * Implemented Data Access Object factory Class
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 *
 */
public class JpaDAOFactory extends DAOFactory {

    @Override
    public DataDAO getDataDAO()

    @Override
    public ProcessGraphDAO getProcessGraphDAO()

    @Override
    public PlantDAO getPlantDAO()

    @Override
    public ProcessDAO getProcessDAO()

    @Override
    public ProductDAO getProductDAO()

    @Override
    public SmartMeterDAO getSmartMeterDAO()

    @Override
    public WorkStationDAO getWorkStationDAO()

}

////////////////////////////////////////////////////////////////////////////////

public class JPAUtil {


    public static EntityManagerFactory getEntityManagerFactory()

    public static EntityManager getCurrentEntityManager()

    public static void setPersistenceUnit(String unit)
}
```

```
////////////////////////////////////////////////////////////////////////////////

/**
 * Implemented Data Access Object Plant Class
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 */
public class PlantJpaDAO extends GenericJpaDAO<Plant> implements PlantDAO{

public Plant findByName(String name)
}

////////////////////////////////////////////////////////////////////////////////

/**
 * Implemented Data Access Object ProcessGraph Class
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 */
public class ProcessGraphJpaDAO extends GenericJpaDAO<ProcessGraph> implements ProcessGraphDAO{
    public List<ProcessGraph> findAll()
}

////////////////////////////////////////////////////////////////////////////////

/**
 * Implemented Data Access Object Process Class
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 */
public class ProcessJpaDAO extends GenericJpaDAO<Process> implements ProcessDAO{
public List<Process> findAll()
}

////////////////////////////////////////////////////////////////////////////////

/**
 * Implemented Data Access Object Product Class
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 */
public class ProductJpaDAO extends GenericJpaDAO<Product> implements ProductDAO{
public List<Product> findAll()
}

////////////////////////////////////////////////////////////////////////////////

/**
 * Implemented Data Access Object SmartMeter Class
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 */
public class SmartMeterJpaDAO extends GenericJpaDAO<SmartMeter> implements SmartMeterDAO{
public List<SmartMeter> findAll()
}

////////////////////////////////////////////////////////////////////////////////

/**
 * Implemented Data Access Object WorkStation Class
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 */
public class WorkStationJpaDAO extends GenericJpaDAO<WorkStation> implements WorkStationDAO{
public List<WorkStation> findAll()
}
```

B.6 Package: main

```
 /**
 * Tester main of smart-meter threads - creates N threads/smart-meters which send
 * XML documents to the server hosted locally.
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 *
 */
public class ClientMain {
public static void main(String args[])
}

////////////////////////////////////////////////////////////////////////////////

/**
 * Tester main of real-time monitoring Graphical User Interface.
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 *
 */
public class DataPresenterMain {
public static void main(String args[])

}

////////////////////////////////////////////////////////////////////////////////

/**
 * Tester main of entity Objects creation and store using Java Persistence API
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 *
 */
public class Main {
public static void main(String [ ] args)
}

////////////////////////////////////////////////////////////////////////////////

/**
 * Tester main of smart-meter threads - creates N threads/smart-meters which send
 * XML documents to the server hosted remotely.
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 *
 */
public class RemoteClientMain {
public static void main(String args[])
}
```

B.7 Package: mapReduce

```
/**
 * Map-Reduce Class. Aggregation is performed over smart-meter serial numbers. For
 * each smart-meter serial number energy consumption measurements are summed up.
```

```
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 */
public class MapReduce {

/**
 * Map Class.
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 */
    public static class MapperClass extends TableMapper<ImmutableBytesWritable, IntWritable> {

        /**
         * Mapping function.
         * For every key/smart-meter serial number the values are written
         * as a pair <key,value> to the mapper-reducer context.
         * @param row The row key. Keys are not stored as
         * smart-meter serial-numbers only but as a composite key - a concatenation of
         * serialNumber+Timestamp. Thus extraction of the key from the composite key
         * is first performed.
         * @param value Energy consumption measurements stored.
         * @param context The mapper-reducer context.
         */
        public void map(ImmutableBytesWritable row, Result values, Context context) throws IOException
    }

    /**
 * Reduce Class.
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 *
 */
    public static class ReducerClass extends TableReducer<ImmutableBytesWritable, IntWritable, ImmutableBytesWritable> {

     /**
      * Reduce function. For every key/smart-meter serial number values are summed up.
      * @param key smart meter serial number
      * @param values a iterable of energy consumption values from the specific key.
      * @param context The mapper-reducer context.
      */
        public void reduce(ImmutableBytesWritable key, Iterable<IntWritable> values, Context context)
    }

    /**
     * Called internally by the Distributed File System Management thread.
     * @param conf HBase configuration object.
     * @throws IOException
     * @throws InterruptedException
     * @throws ClassNotFoundException
     */
    public void runJob(Configuration conf) throws IOException, InterruptedException, ClassNotFoundException

    /**
     * Main function for Map-Reduce job. Necessary for map-reduce runnable jar file which must be
     * distributed to each and every node of the system.
     * @param args
     * @throws Exception
     */
    public static void main(String[] args) throws Exception
}
```

B.8 Package: service

```
/**
 * The Smart-meter Class.
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 *
 */
public class Client implements Runnable{

  /**
   * Constructor setting private fields to default values.
   */
  public Client()

/**
 * Constructor setting private fields to custom values.
 * @param port The server's listening port.
 * @param server The server's name.
 * @param flag A flag passed to the random value generator, which assigns to the smart-meter
 * the serial-number stathisTheGreat.
 */
  public Client(int port,String server,boolean flag)
  /**
   * Constructor setting 2 private fields to custom values.
   * @param port The server's listening port.
   * @param server The server's name.
   */
  public Client(int port,String server)

  /**
   * "Turns on" the smart-meter.
   */
public void run()

/**
 * Smart-meter tester main sending values to local hosted server.
 * @param args
 */
public static void main(String args[])
}

////////////////////////////////////////////////////////////////////////////////

/**
 * Distributed File System management thread.
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 *
 */
public class DFSManager implements Runnable{

/**
 * Constructor setting private field conf to custom value.
 * @param conf HBase configuration.
 */
public DFSManager(Configuration conf)

/**
 * Constructor setting all private fields to custom values.
 */
public DFSManager(FileSystem dfs, Configuration conf, String sourcePath)
```

```
/**
 * Constructor setting all private fields to default values.
 */
public DFSManager()

/**
 * Starts the distributed file system management thread. Sleeps for N minutes
 * and calls the map-reduce function recursively.
 */
public void run()

}

////////////////////////////////////////////////////////////////////////////////

/**
 * The Server Class.
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 */
public class MultiThreadedServer implements Runnable{
    /**
     * Constructor setting private field port to custom value.
     * @param port The server's listening port.
     * @throws IOException
     */
    public MultiThreadedServer(int port) throws IOException

    /**
     * Starts the server thread, responsible for listening to the XML data stream and setting
     * a separate thread for xml parsing per incoming XML document.
     */
    public void run()

}

////////////////////////////////////////////////////////////////////////////////

/**
 * Tester main of the multithreaded server.
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 */
public class MultiThreadedServerServiceTest {
public static void main(String args[])

}

////////////////////////////////////////////////////////////////////////////////

/**
 * Rando value generator Class.
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 */
public class RandomValuesGenerator {

/**
 * Generates a random value from 0 to 1.
 * @return random energy consumption value
 */
public double generateValue()
```

```
/**
 * Generates random serial number.
 * @return random serial number as a String.
 */
public String generateSerialNumber()

/**
 * Getter for the private field serialnumber
 * @return the serial number.
 */
public String getSerialNumer()

/**
 * Setter for the private field serialnumber
 * @param serialNumber The serial number.
 */
public void setSerialNumer(String serialNumer)

/**
 * Getter for the private field timeStamp
 * @return the timeStamp.
 */
public String getTimestamp()

/**
 * Setter for the private field serialnumber
 * @param timestamp the timestamp.
 */
public void setTimestamp(String timestamp)

/**
 * Getter for the private field value - energy consumption measurement.
 * @return energy consumption measurement.
 */
public long getValue()

/**
 * Setter for the private field value - energy consumption measurement.
 * @param value energy consumption measurement.
 */
public void setValue(long value)
}

////////////////////////////////////////////////////////////////////////////////

/**
 * Service Class for data request from the HBase table and representation of real-time energy
 * consumption measurements in a graph.
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 *
 */
public class RealTimeDataPresenter implements Runnable{

/**
 * Constructor setting private field conf to a sutom HBase configuration object.
 * @param conf HBase configuration object.
 * @throws IOException
 */
public RealTimeDataPresenter(Configuration conf) throws IOException
```

```
/**
 * Starts the real-time measurements presenter.
 */
public void run()
}

////////////////////////////////////////////////////////////////////////////////

/**
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 */
public class ServerService {

public ServerService(){

}

/**
 * Thread that manages the connection from a socket and initiates the parsing threads.
 * @param connection Listening socket.
 * @throws IOException
 */
private void serviceThread(Socket connection) throws IOException

/**
 * Starts the listening thread.
 */
public void run()

public static void main(String args[])
}

////////////////////////////////////////////////////////////////////////////////

public class TimestampComparator implements Comparator{
public int compare(Object o1, Object o2) {

  public int compare(String s1, String s2)

}

////////////////////////////////////////////////////////////////////////////////


/**
 * Worker Class. The Thread that takes in charge after a XML document has reached the server's
 * listening port.
 * @author Stathis Plitsos, stathis.plitsos[at]gmail.com
 */
public class WorkerRunnable implements Runnable{

    /**
     * Constructor setting private fields to custom values.
     */
    public WorkerRunnable(Socket clientSocket, String serverText, FileSystem dfs, Configuration conf){//, JobClient jc)

    /**
     * Constructor setting private fields to custom values.
     */
```

```
    public WorkerRunnable(Socket clientSocket, String serverText, FileSystem dfs, Configuration conf,HTable htable){//, JobClient jc)

    /**
     * Starts the worker thread..
     */
    public void run()
}
```

C Hadoop's Map-Reduce Output

C.1 Test 1

```
12/02/07 16:57:30 INFO mapred.JobClient: Running job: job_201202071437_0002
12/02/07 16:57:31 INFO mapred.JobClient:  map 0\% reduce 0\%
12/02/07 16:58:05 INFO mapred.JobClient:  map 100\% reduce 0\%
12/02/07 16:58:14 INFO mapred.JobClient:  map 100\% reduce 33\%
12/02/07 16:58:17 INFO mapred.JobClient:  map 100\% reduce 75\%
12/02/07 16:58:20 INFO mapred.JobClient:  map 100\% reduce 93\%
12/02/07 16:58:23 INFO mapred.JobClient:  map 100\% reduce 100\%
12/02/07 16:58:28 INFO mapred.JobClient: Job complete: job_201202071437_0002
12/02/07 16:58:28 INFO mapred.JobClient: Counters: 28
12/02/07 16:58:28 INFO mapred.JobClient:   Job Counters
12/02/07 16:58:28 INFO mapred.JobClient:     Launched reduce tasks=1
12/02/07 16:58:28 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=22608
12/02/07 16:58:28 INFO mapred.JobClient:     Total time spent by all reduces waiting after reserving slots (ms)=0
12/02/07 16:58:28 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving slots (ms)=0
12/02/07 16:58:28 INFO mapred.JobClient:     Launched map tasks=1
12/02/07 16:58:28 INFO mapred.JobClient:     Data-local map tasks=1
12/02/07 16:58:28 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=16297
12/02/07 16:58:28 INFO mapred.JobClient:   File Output Format Counters
12/02/07 16:58:28 INFO mapred.JobClient:     Bytes Written=0
12/02/07 16:58:28 INFO mapred.JobClient:   FileSystemCounters
12/02/07 16:58:28 INFO mapred.JobClient:     FILE_BYTES_READ=15731910
12/02/07 16:58:28 INFO mapred.JobClient:     HDFS_BYTES_READ=68
12/02/07 16:58:28 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=23656487
12/02/07 16:58:28 INFO mapred.JobClient:   File Input Format Counters
12/02/07 16:58:28 INFO mapred.JobClient:     Bytes Read=0
12/02/07 16:58:28 INFO mapred.JobClient:   Map-Reduce Framework
12/02/07 16:58:28 INFO mapred.JobClient:     Map output materialized bytes=7865952
12/02/07 16:58:28 INFO mapred.JobClient:     Map input records=436997
12/02/07 16:58:28 INFO mapred.JobClient:     Reduce shuffle bytes=7865952
12/02/07 16:58:28 INFO mapred.JobClient:     Spilled Records=1310991
12/02/07 16:58:28 INFO mapred.JobClient:     Map output bytes=6991952
12/02/07 16:58:28 INFO mapred.JobClient:     CPU time spent (ms)=16830
12/02/07 16:58:28 INFO mapred.JobClient:     Total committed heap usage (bytes)=278659072
12/02/07 16:58:28 INFO mapred.JobClient:     Combine input records=0
12/02/07 16:58:28 INFO mapred.JobClient:     SPLIT_RAW_BYTES=68
12/02/07 16:58:28 INFO mapred.JobClient:     Reduce input records=436997
12/02/07 16:58:28 INFO mapred.JobClient:     Reduce input groups=426742
12/02/07 16:58:28 INFO mapred.JobClient:     Combine output records=0
12/02/07 16:58:28 INFO mapred.JobClient:     Physical memory (bytes) snapshot=367239168
12/02/07 16:58:28 INFO mapred.JobClient:     Reduce output records=426742
12/02/07 16:58:28 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=1091702784
12/02/07 16:58:28 INFO mapred.JobClient:     Map output records=436997
```

C.2 Test 2

```
18:20:12 INFO mapred.JobClient:  map 100% reduce 0%
12/02/07 18:20:24 INFO mapred.JobClient:  map 100% reduce 71%
12/02/07 18:20:27 INFO mapred.JobClient:  map 100% reduce 87%
12/02/07 18:20:30 INFO mapred.JobClient:  map 100% reduce 100%
12/02/07 18:20:35 INFO mapred.JobClient: Job complete: job_201202071818_0001
12/02/07 18:20:35 INFO mapred.JobClient: Counters: 29
12/02/07 18:20:35 INFO mapred.JobClient:   Job Counters
12/02/07 18:20:35 INFO mapred.JobClient:     Launched reduce tasks=1
12/02/07 18:20:35 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=66378
12/02/07 18:20:35 INFO mapred.JobClient:     Total time spent by all reduces waiting after reserving slots (ms)=0
12/02/07 18:20:35 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving slots (ms)=0
12/02/07 18:20:35 INFO mapred.JobClient:     Rack-local map tasks=1
12/02/07 18:20:35 INFO mapred.JobClient:     Launched map tasks=2
12/02/07 18:20:35 INFO mapred.JobClient:     Data-local map tasks=1
12/02/07 18:20:35 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=16280
12/02/07 18:20:35 INFO mapred.JobClient:   File Output Format Counters
12/02/07 18:20:35 INFO mapred.JobClient:     Bytes Written=0
12/02/07 18:20:35 INFO mapred.JobClient:   FileSystemCounters
12/02/07 18:20:35 INFO mapred.JobClient:     FILE_BYTES_READ=15751926
12/02/07 18:20:35 INFO mapred.JobClient:     HDFS_BYTES_READ=68
12/02/07 18:20:35 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=23686509
12/02/07 18:20:35 INFO mapred.JobClient:   File Input Format Counters
12/02/07 18:20:35 INFO mapred.JobClient:     Bytes Read=0
12/02/07 18:20:35 INFO mapred.JobClient:   Map-Reduce Framework
12/02/07 18:20:35 INFO mapred.JobClient:     Map output materialized bytes=7875960
12/02/07 18:20:35 INFO mapred.JobClient:     Map input records=437553
12/02/07 18:20:35 INFO mapred.JobClient:     Reduce shuffle bytes=0
12/02/07 18:20:35 INFO mapred.JobClient:     Spilled Records=1312659
12/02/07 18:20:35 INFO mapred.JobClient:     Map output bytes=7000848
12/02/07 18:20:35 INFO mapred.JobClient:     CPU time spent (ms)=17100
12/02/07 18:20:35 INFO mapred.JobClient:     Total committed heap usage (bytes)=286785536
12/02/07 18:20:35 INFO mapred.JobClient:     Combine input records=0
12/02/07 18:20:35 INFO mapred.JobClient:     SPLIT_RAW_BYTES=68
12/02/07 18:20:35 INFO mapred.JobClient:     Reduce input records=437553
12/02/07 18:20:35 INFO mapred.JobClient:     Reduce input groups=430543
12/02/07 18:20:35 INFO mapred.JobClient:     Combine output records=0
12/02/07 18:20:35 INFO mapred.JobClient:     Physical memory (bytes) snapshot=362729472
12/02/07 18:20:35 INFO mapred.JobClient:     Reduce output records=430543
12/02/07 18:20:35 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=1039171584
12/02/07 18:20:35 INFO mapred.JobClient:     Map output records=437553
```

C.3 Test 3

```
12/02/10 15:07:58 INFO mapred.JobClient: Running job: job_201202101430_0001
12/02/10 15:07:59 INFO mapred.JobClient:  map 0% reduce 0%
12/02/10 15:08:32 INFO mapred.JobClient:  map 100% reduce 0%
12/02/10 15:08:44 INFO mapred.JobClient:  map 100% reduce 78%
12/02/10 15:08:47 INFO mapred.JobClient:  map 100% reduce 100%
12/02/10 15:08:52 INFO mapred.JobClient: Job complete: job_201202101430_0001
12/02/10 15:08:52 INFO mapred.JobClient: Counters: 28
12/02/10 15:08:52 INFO mapred.JobClient:   Job Counters
12/02/10 15:08:52 INFO mapred.JobClient:     Launched reduce tasks=1
12/02/10 15:08:52 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=18666
12/02/10 15:08:52 INFO mapred.JobClient:     Total time spent by all reduces waiting after reserving slots (ms)=0
12/02/10 15:08:52 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving slots (ms)=0
12/02/10 15:08:52 INFO mapred.JobClient:     Launched map tasks=1
12/02/10 15:08:52 INFO mapred.JobClient:     Data-local map tasks=1
12/02/10 15:08:52 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=13342
12/02/10 15:08:52 INFO mapred.JobClient:   File Output Format Counters
12/02/10 15:08:52 INFO mapred.JobClient:     Bytes Written=0
12/02/10 15:08:52 INFO mapred.JobClient:   FileSystemCounters
12/02/10 15:08:52 INFO mapred.JobClient:     FILE_BYTES_READ=11647854
12/02/10 15:08:52 INFO mapred.JobClient:     HDFS_BYTES_READ=68
12/02/10 15:08:52 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=17530399
12/02/10 15:08:52 INFO mapred.JobClient:   File Input Format Counters
12/02/10 15:08:52 INFO mapred.JobClient:     Bytes Read=0
12/02/10 15:08:52 INFO mapred.JobClient:   Map-Reduce Framework
12/02/10 15:08:52 INFO mapred.JobClient:     Map output materialized bytes=5823924
12/02/10 15:08:52 INFO mapred.JobClient:     Map input records=323551
12/02/10 15:08:52 INFO mapred.JobClient:     Reduce shuffle bytes=0
12/02/10 15:08:52 INFO mapred.JobClient:     Spilled Records=970653
12/02/10 15:08:52 INFO mapred.JobClient:     Map output bytes=5176816
12/02/10 15:08:52 INFO mapred.JobClient:     CPU time spent (ms)=13940
12/02/10 15:08:52 INFO mapred.JobClient:     Total committed heap usage (bytes)=273547264
12/02/10 15:08:52 INFO mapred.JobClient:     Combine input records=0
12/02/10 15:08:52 INFO mapred.JobClient:     SPLIT_RAW_BYTES=68
12/02/10 15:08:52 INFO mapred.JobClient:     Reduce input records=323551
12/02/10 15:08:52 INFO mapred.JobClient:     Reduce input groups=320369
12/02/10 15:08:52 INFO mapred.JobClient:     Combine output records=0
12/02/10 15:08:52 INFO mapred.JobClient:     Physical memory (bytes) snapshot=371212288
12/02/10 15:08:52 INFO mapred.JobClient:     Reduce output records=320369
12/02/10 15:08:52 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=1167138816
12/02/10 15:08:52 INFO mapred.JobClient:     Map output records=323551
```

References

[1] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R Motwani, U. Srivastava, and J. Widom, *Stream: The stanford data stream management system*, Technical Report 2004-20, Stanford InfoLab, 2004.

[2] D. Carstoiu, A. Cernian, and A. Olteanu, *Hadoop hbase-0.20.2 performance evaluation*, New Trends in Information Science and Service Science (NISS), 2010 4th International Conference on, may 2010, pp. 84 –87.

[3] V. Nikolopoulos Intelen CEO, *Intelen's infrastructure and services*, Nov 2011, http://www.intelen.com/corporate/home, personal communication.

[4] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber, *Bigtable: A distributed storage system for structured data*, ACM Trans. Comput. Syst. **26** (2008), 4:1–4:26.

[5] Damianos Chatziantoniou, Katerina Pramatari, and Yannis Sotiropoulos, *Supporting real-time supply chain decisions based on rfid data streams*, Journal of Systems and Software **84** (2011), no. 4, 700 – 710.

[6] Okeanos cloud, https://okeanos.grnet.gr/.

[7] Cloudera, http://www.cloudera.com/.

[8] European Commission, *Saving 20% by 2020*, Action plan, European Union, 2007.

[9] Tyson Condie, Neil Conway, Peter Alvaro, Joseph M. Hellerstein, Khaled Elmeleegy, and Russell Sears, *Hop configuration - implementation wiki*, EECS Department, University of California, Berkeley.

[10] ———, *Mapreduce online*, Proceedings of the 7th USENIX conference on Networked systems design and implementation (Berkeley, CA, USA), NSDI'10, USENIX Association, 2010, pp. 21–21.

[11] Jeffrey Dean and Sanjay Ghemawat, *Mapreduce: simplified data processing on large clusters*, Commun. ACM **51** (2008), 107–113.

[12] IBM Institute for Business Value, *The smarter supply chain of the future - insights from the global chief supply chain officer study.*, Global survey, IBM, 2010.

[13] Apache Hadoop, http://hadoop.apache.org/.

[14] Apache HBase, http://hbase.apache.org/.

[15] J. Hooker, *Integrated methods for optimization*, International series in operations research & management science, Springer, 2007.

[16] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira, and Benjamin Reed, *Zookeeper: wait-free coordination for internet-scale systems*, Proceedings of the 2010 USENIX conference on USENIX annual technical conference (Berkeley, CA, USA), USENIXATC'10, USENIX Association, 2010, pp. 11–11.

[17] Oracle Application Server Documentation Library, *Oracle sensor edge server administrator's guide 10g release 2 (10.1.2)*, Oracle.

[18] Apache Mahout, http://mahout.apache.org/.

[19] V. Nikolopoulos, G. Mpardis, I. Giannoukos, I. Lykourentzou, and V. Loumos, *Web-based decision-support system methodology for smart provision of adaptive digital energy services over cloud technologies*, IET Software **5** (2011), no. 5, 454–465.

[20] Y. Pochet and L.A. Wolsey, *Production planning by mixed integer programming*, Springer series in operations research and financial engineering, Springer, 2006.

[21] Artisan Project, http://spring.bologna.enea.it/artisan/default.asp?lingua=en.

[22] Shuja Rehman, *Xml processing in hadoop*, http://xmlandhadoop.blogspot.com/, personal communication, Nov 2011.

[23] Greek Research and Technology Network, http://www.grnet.gr/.

[24] Scientific Applications International Corporation (SAIC) and U.S. Environmental Protection Agency, *Life cycle assessment: Principles and practice*, Introductory overview, U.S. E.P.A, 2006.

[25] K. Shvachko, Hairong Kuang, S. Radia, and R. Chansler, *The hadoop distributed file system*, Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on, may 2010, pp. 1 –10.

[26] T.H. Tietenberg, *Environmental economics and policy*, Addison-Wesley series in economics, Pearson Addison Wesley, 2004.

[27] Apache Zookeeper, http://zookeeper.apache.org/.