A Real-time Data Processor for Service Quality Management

Ángelo Francisco de Matos Pingo

Thesis to obtain the Master of Science Degree in

Telecommunications and Informatics Engineering

Supervisors: Prof. Rui António dos Santos Cruz
Eng. Rui Alves da Cunha

Examination Committee

Chairperson: Prof. Paulo Jorge Pires Ferreira
Supervisor: Prof. Rui António dos Santos Cruz
Member of the Committee: Prof. Diogo Manuel Ribeiro Ferreira

June 2017
Acknowledgments

I would like to thank my academic dissertation supervisors Prof. Rui Cruz and also to my Nokia supervisors Eng. Rui Cunha, for all the assistance and guidance in this project and for the technical knowledge they have provided me with to make this thesis possible.

At a technical level, I would like to thank João Martins, Hugo Mascarenhas and Paulo Carvalho, who provided me with very important help in building the Real-Time Performance Management (RTPM).

I would also like to thank my girlfriend Márcia Lopes for all the support she gave me, which was an essential contribution.

Last but not least I would like to thank my mother Isabel Matos for the very important contribution she provided with the non-technical review in order to evaluate the main idea and the structure of this thesis document.

To each and every one of you – Thank you.
Abstract

The increasing number of devices connected to mobile networks of different technologies such as Global System for Mobile Communications (GSM), Universal Mobile Telecommunications System (UMTS) and Long Term Evolution (LTE), is driving the growth of the network infrastructure to support them. In parallel the support systems attached to these networks need to process increasing volumes of data, in real-time, to ensure that the infrastructure is fully operational. A key task of those support systems is calculating Key Performance Indicators (KPIs) and Key Quality Indicators (KQIs), from the counters provided by Network Elements (NEs). These KPIs and KQIs are essential to understand how the network is performing. To enable Communication Services Providers (CSPs) to monitor and react to changes in performance of their networks, Nokia offers the Service Quality Manager (SQM) product, that calculates KPIs and KQIs in “near” real-time. To optimize SQM, for better performance and timely delivery of results, the current calculation module needs to be re-engineered in order to handle higher data volumes with a lower latency, and prepared for future evolution. This new module, named Real-Time Performance Management (RTPM), here described and implemented, is based on new and more efficient tools for real-time computation, such as “Apache Spark Streaming” for data processing and “Apache Kafka” message broker.

Keywords

Mobile services; Key Quality Indicator (KQI); Key Performance Indicator (KPI); Real-time Computation; Service assurance; Service Quality Manager (SQM).
Resumo

O crescente número de dispositivos ligados às redes móveis e as diferentes tecnologias tais como o Global System for Mobile Communications (GSM), Universal Mobile Telecommunications System (UMTS) e Long Term Evolution (LTE), está a levar a um crescimento da infraestrutura da rede para suportá-los. Em paralelo, os sistemas de suporte a estas redes precisam de processar volumes cada vez maiores de dados e em tempo real, para assegurar que a infraestrutura está completamente operacional. Uma tarefa chave destes sistemas de suporte é calcular os Key Performance Indicator (KPI) e Key Quality Indicator (KQI), provenientes dos contadores fornecidos pelos Network Element (NE). Estes KPIs e KQIs são essenciais para entender o funcionamento da rede. Para permitir que os Communication Services Provider (CSP) consigam monitorizar e reagir às mudanças de desempenho das suas redes a Nokia oferece o produto Service Quality Manager (SQM), que calcula KPIs and KQIs em “quase” tempo real. Para otimizar o SQM, obtendo um melhor desempenho e entrega mais rápida de resultados, o módulo de cálculo atual precisa de ser alterado para conseguir lidar com volumes de dados maiores, e menor latência preparando-se para uma futura evolução. Este novo módulo chamado de Real-Time Performance Management (RTPM), é aqui descrito e implementado, baseia-se em novas ferramentas mais eficientes para processamento em tempo real, tais como o Apache Spark Streaming para processamento de dados e o Apache Kafka como intermediário de mensagens.

Palavras Chave

Serviços de Redes Móveis; Key Quality Indicator (KQI); Key Performance Indicator (KPI); Computação em Tempo Real; Garantia de serviço; Service Quality Manager (SQM).
# Contents

1 Introduction ............................ 1
   1.1 Objectives .................................. 4
   1.2 Document Organization ....................... 4

2 Background .............................. 7
   2.1 Communication Services Infrastructures and Operations .......... 9
   2.2 Network Architecture and Topology ......................... 11
   2.3 Mediation Processes and Dataflows ....................... 13
   2.4 Service Quality Manager (SQM) ......................... 14

3 System Design and Implementation ............... 19
   3.1 Architecture .................................. 21
   3.2 Implementation Options ......................... 22
   3.3 Message Broker Tools .......................... 22
   3.4 Real-time Computation System Tools ................. 23
   3.5 System Implementation .......................... 24
      3.5.1 Extract, Transform and Load (ETL) ................. 26
      3.5.2 Calculator ................................ 35

4 Evaluation ............................... 41
   4.1 Evaluation Environment Specifications .................. 43
      4.1.1 Spark setup ................................ 44
      4.1.2 Cassandra setup ............................ 44
      4.1.3 Kafka and Zookeeper setup .................... 44
   4.2 ETL and Calculator evaluation ...................... 45
      4.2.1 Synchronous versus asynchronous ............... 45
      4.2.2 Evaluation of RAM utilization ............... 50
      4.2.3 Different input files in ETL evaluation ......... 52
      4.2.4 Kafka partitions evaluation .................. 54
      4.2.5 Scalability evaluation in ETL and Calculator .... 56
5 Conclusion

5.1 Conclusions ................................................................. 63
5.2 System Limitations and Future Work ............................... 63
List of Figures

2.1 eTOM level 0 ................................................................. 10
2.2 eTOM level 2 - Operations ............................................. 11
2.3 Typical Mobile Broadband Architecture Topology. ................. 12
2.4 Dataflows in Current System Design with SQM ....................... 14
2.5 SQM Components .......................................................... 15
2.6 Example of OMeS file ..................................................... 16

3.1 System Design with new RTPM Module ................................. 21
3.3 System Implementation – ETL ............................................ 27
3.4 System Implementation – ETL Parser – Identify Sources component. ..... 28
3.5 System Implementation – ETL Parser – Parse Files component. ....... 28
3.6 System Implementation – ETL Merger – Split Counters by KPI component. 30
3.7 System Implementation – ETL Merger – Wait for All Counters component. 30
3.8 System Implementation – Calculator component .......................... 35
3.9 System Implementation – Calculator – Calculate KPI module. ........... 36
3.10 Formulas. ...................................................................... 39
3.11 System Implementation – Calculator – Store Result module. .......... 39

4.1 ETL: Burst of 5000 files, 1 VM, 16 Partitions. .......................... 46
4.2 Calculator: Burst of 682500 CounterData, 1 VM, 16 Partitions. ....... 47
4.3 Calculator: CPU utilization in asynchronous mode. .................... 47
4.4 ETL: CPU utilization in asynchronous mode. ............................ 48
4.5 Calculator: CPU utilization in synchronous mode. ...................... 48
4.6 ETL: CPU utilization in synchronous mode. ............................. 49
4.7 ET: RAM utilization in synchronous mode. ............................. 49
4.8 ETL: RAM utilization in asynchronous mode. ........................... 50
4.9 Calculator: RAM utilization in asynchronous mode. ................... 50
4.10 Calculator: RAM utilization in synchronous mode.

4.11 Calculator: CPU vs. Network utilization in synchronous mode.


4.13 ETL: Burst of 5000 files, 4 VMs, 16 Partitions.

4.14 Calculator: Burst of 682500 CounterData's, 4 VMs, 16 Partitions.

4.15 RAM: Asynchronous, 4 VMs, 16 Cores.

4.16 ETL: Burst of 5000 input files, Asynchronous, 4 VMs, 16 Cores, 10GB RAM.

4.17 Kafka Partitions in Calculator: Burst of 682500 CounterData's, Asynchronous, 4 VMs, 16 Cores.

4.18 Kafka Partitions in Calculator: Burst of 682500 CounterData's, Synchronous, 4 VMs, 16 Cores.

4.19 ETL: Burst of 5000 files.

4.20 Calculator: Burst of 682500 CounterData's.
List of Tables

3.1 ETL Properties File .................................................. 27
3.2 Calculator Properties File ............................................ 36
3.3 CounterData Subset ................................................... 37
4.1 Network characteristics for evaluation of the project ............ 56
## Listings

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Part of OMEaS File tested</td>
<td>29</td>
</tr>
<tr>
<td>3.2</td>
<td>Object CounterData after Avro Serialization</td>
<td>31</td>
</tr>
<tr>
<td>3.3</td>
<td>CounterData Schema</td>
<td>33</td>
</tr>
<tr>
<td>3.4</td>
<td>KpiResult Schema</td>
<td>33</td>
</tr>
<tr>
<td>3.5</td>
<td>Object KpiResult after Avro Serialization</td>
<td>38</td>
</tr>
<tr>
<td>3.6</td>
<td>Network Topology construction</td>
<td>40</td>
</tr>
</tbody>
</table>
Acronyms

2G  second generation mobile
3GPP  3rd Generation Partnership Project
3G  third generation mobile
4G  fourth generation mobile
5G  fifth generation mobile
API  Application Programming Interface
BSC  Base Station Controller
BS  Base Station
BTS  Base Transceiver Station
CM  Configuration Manager
CPU  Central Processing Unit
CSP  Communication Services Provider
CSV  Comma Separated Value
CS  Circuit Switched
DOM  Document Object Model
E-UTRAN  Evolved Universal Terrestrial Access Network
EDGE  Enhanced Data Rates for GSM Evolution
EEM  Event Engine Manager
EMS  Element Management System
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETL</td>
<td>Extract, Transform and Load</td>
</tr>
<tr>
<td>eTOM</td>
<td>Business Process Framework</td>
</tr>
<tr>
<td>FM</td>
<td>Fault Management</td>
</tr>
<tr>
<td>GB</td>
<td>Gigabyte</td>
</tr>
<tr>
<td>GERAN</td>
<td>GSM/EDGE Radio Access Network</td>
</tr>
<tr>
<td>GSM</td>
<td>Global System for Mobile Communications</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HNI</td>
<td>Home Network Identity</td>
</tr>
<tr>
<td>HTTPS</td>
<td>Hyper Text Transfer Protocol Secure</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hyper Text Transfer Protocol</td>
</tr>
<tr>
<td>IMSI</td>
<td>International Mobile Subscriber Identity</td>
</tr>
<tr>
<td>IPTV</td>
<td>IP Television</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>JAR</td>
<td>Java ARchive</td>
</tr>
<tr>
<td>JAXP</td>
<td>Java API for XML Processing</td>
</tr>
<tr>
<td>JMS</td>
<td>Java Message Service</td>
</tr>
<tr>
<td>JPL</td>
<td>Jet Propulsion Laboratory</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>KPI</td>
<td>Key Performance Indicator</td>
</tr>
<tr>
<td>KQI</td>
<td>Key Quality Indicator</td>
</tr>
<tr>
<td>LNBTS</td>
<td>Flexi Multiradio BTS</td>
</tr>
<tr>
<td>LNCEL</td>
<td>Flexi Multiradio BTS Cell</td>
</tr>
<tr>
<td>LTE</td>
<td>Long Term Evolution</td>
</tr>
<tr>
<td>MCC</td>
<td>Mobile Country Code</td>
</tr>
<tr>
<td>MEAS</td>
<td>Measurement</td>
</tr>
<tr>
<td>Acronym</td>
<td>Full Form</td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
</tr>
<tr>
<td>MME</td>
<td>Mobility Management Entity</td>
</tr>
<tr>
<td>MNC</td>
<td>Mobile Network Code</td>
</tr>
<tr>
<td>MOM</td>
<td>Message-Oriented Middleware</td>
</tr>
<tr>
<td>MRBTS</td>
<td>Multiradio Base Transceiver Station</td>
</tr>
<tr>
<td>MSIN</td>
<td>Mobile Subscriber Identification Number</td>
</tr>
<tr>
<td>NE</td>
<td>Network Element</td>
</tr>
<tr>
<td>NFS</td>
<td>Network File System</td>
</tr>
<tr>
<td>NSN</td>
<td>Nokia Solutions and Networks</td>
</tr>
<tr>
<td>OMeS</td>
<td>Open Measurement Standard</td>
</tr>
<tr>
<td>OSS</td>
<td>Operations Support Services</td>
</tr>
<tr>
<td>P2P</td>
<td>Peer-to-Peer</td>
</tr>
<tr>
<td>PLMN</td>
<td>Public Land Mobile Network</td>
</tr>
<tr>
<td>PM</td>
<td>Performance Manager</td>
</tr>
<tr>
<td>PoC</td>
<td>Proof of Concept</td>
</tr>
<tr>
<td>PSK</td>
<td>Phase-Shift Keying</td>
</tr>
<tr>
<td>PS</td>
<td>Packet Switch</td>
</tr>
<tr>
<td>QAM</td>
<td>Quadrature Amplitude Modulation</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>RAN</td>
<td>Radio Access Network</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>RNC</td>
<td>Radio Network Controller</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote Procedure Call</td>
</tr>
<tr>
<td>RTPM</td>
<td>Real-Time Performance Management</td>
</tr>
<tr>
<td>S-GW</td>
<td>Serving Gateway</td>
</tr>
<tr>
<td>Acronym</td>
<td>Full Form</td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
</tr>
<tr>
<td>SAX</td>
<td>Simple API for XML</td>
</tr>
<tr>
<td>SID</td>
<td>Information Framework</td>
</tr>
<tr>
<td>SLA</td>
<td>Service-Level Agreement</td>
</tr>
<tr>
<td>SMS</td>
<td>Short Message Service</td>
</tr>
<tr>
<td>SNR</td>
<td>Signal-to-Noise Ratio</td>
</tr>
<tr>
<td>SOA</td>
<td>Service-Oriented Architecture</td>
</tr>
<tr>
<td>SP</td>
<td>Service Provider</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>SQM</td>
<td>Service Quality Manager</td>
</tr>
<tr>
<td>SSL</td>
<td>Secure Sockets Layer</td>
</tr>
<tr>
<td>StAX</td>
<td>Streaming API for XML</td>
</tr>
<tr>
<td>TAM</td>
<td>Application Framework</td>
</tr>
<tr>
<td>TCP</td>
<td>Transport Control Protocol</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>UMTS</td>
<td>Universal Mobile Telecommunications System</td>
</tr>
<tr>
<td>UTRAN</td>
<td>Universal Terrestrial Radio Access Network</td>
</tr>
<tr>
<td>vCPU</td>
<td>Virtual CPU</td>
</tr>
<tr>
<td>VM</td>
<td>virtual machine</td>
</tr>
<tr>
<td>VoLTE</td>
<td>Voice over LTE</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
Introduction

Contents

1.1 Objectives ........................................................................................................ 4
1.2 Document Organization .................................................................................... 4
Nowadays, operators have numerous access technologies, mainly due to the fast development of mobile networks. The most common is still the third generation mobile (3G) technology. While the fourth generation mobile (4G) is having a significant presence in the provision of mobile operators [1–3], the fifth generation mobile (5G) is already in the standardization phase [4]. The adoption of new access technologies brings some consequences, being one, the increase in use of higher spectral bands (higher frequencies), which reduces the radius of coverage of each radio Base Station (BS), and another, more variations in the signal modulation—from 8 Phase-Shift Keying (PSK) in Enhanced Data Rates for GSM Evolution (EDGE) [5] to 256 Quadrature Amplitude Modulation (QAM) in Long Term Evolution (LTE) [6]—increasing enormously the available bandwidth and transmission speeds (higher spectral efficiency), therefore leading to an increase in the number of BSs for the same coverage area, and to the upgrade of the network infrastructure for larger volumes of data traffic. Another consequence comes from the evolution of the end-user terminals, allowing higher bandwidths and transmission speeds, and the increasing use of multimedia services, such as real-time Peer-to-Peer (P2P) communications, video and audio streaming, as well as specialized mobile “apps” using “Cloud” technologies, that will lead to a very large increase in the consumption of data. At the end of 2016 the global mobile traffic data was of 7.2 exabytes per month, and by 2020 it is estimated that this figure will rise to 34.6 exabytes [7] and 49 exabytes in 2021. In this manner, operators have more and more information on network equipment and also on all the usage by customers to process, and with increasingly tighter time constraints. All this information is very important for the operator business, not just in order to have a more “real” view of the state of the network, but, more importantly, to be able to provide a better experience, therefore increasing customer satisfaction and retention (reducing churn). To offer this kind of service quality it is not just enough to collect all information on customers usage, it is also necessary to timely process that very information to be able to take actions in real-time.

Current systems have the capacity to gather all the information and to process it, but they are still not able to do it in a timely manner, failing to provide a good “management experience” to the end-user. Most operator’s networks in the world have typically tens of millions customers, with some reaching more than a hundred million [8]. The volume of information these operators have to deal with is really massive, with the addition of having time constraints on the usefulness of that information for the provision of the best customer experience.

The mentioned information data is typically collected with an Element Management System (EMS), which, when referring to Nokia, is a product called “NetAct”. After collecting and processing the raw data from Network Elements (NEs), “NetAct” will transform it into Extensible Markup Language (XML) files which are then sent to the Performance Manager (PM) application. These XML files contain the network hierarchy information and the respective counters. The “NetAct” is agnostic to the technology, i.e., is multi-technology [9]. Those counters are very important because Key Performance Indicators (KPIs)
can be calculated with them. KPIs are indicators of the current state of the network and Key Quality Indicators (KQIs), which are indicators of the network service, can be calculated by the counters and KPIs. The KPIs are calculated in various levels so that the network can be monitored at different scales, from the last level, for example a cell of a Base Transceiver Station (BTS), to the network state of various Base Station Controllers (BSCs) that represent a geographic area, such as a city or a region.

The increasing volume of data to be processed in Nokia’s current Service Quality Manager (SQM) architecture, is raising the computation time and delaying the system work-flows. The speed of the entire calculation process of the Key Performance Indicator (KPI) and Key Quality Indicator (KQI), since the arrival of the data containing the counters of the NEs in Service Quality Manager (SQM) up to the presentation of useful information to the operator, is therefore the key bottleneck in the system, and an improvement that Nokia seeks to develop in order to introduce in future products.

1.1 Objectives

This project has two objectives, being the first the development of a Proof of Concept (PoC) low-latency network and transactional “Real-Time Data Processor”, to process data arriving from different sources/ formats, and compute KPI, and the second to evaluate the performance of the developed tool, namely for the defined targets (described in Chapter 4), to estimate the number of machines needed (and with which specifications) so that the intended performance can be achieved.

The “Real-Time Data Processor” here proposed and described, will eventually replace the processing logic components of the current PM tool, with a new Real-Time Performance Management (RTPM) module that uses more efficient tools for real-time computation and for the exchange of data between the Extract, Transform and Load (ETL), Calculator and Collector components.

The RTPM should be able to consume counters from the NEs that come in XML files supplied by the “NetAct”, calculate the respective KPIs in the smallest time period possible and deliver them to the “Message Broker” so that they can be consumed by the SQM and PM.

1.2 Document Organization

This document is organized as follows.

Chapter 1 provides the context, defines the problem and presents the objectives of the project.
Chapter 2 gives some background information and describes the related technologies.
Chapter 3 provides a generic architecture for the project and briefly describes the tools used for the implementation, and detailing the architectural aspects of the module. It further specifies the entire implementation of RTPM.
Chapter 4 presents all the tests performed to the RTPM in order to verify the performance of this tool and Chapter 5 draws conclusions on the work performed, and discusses future enhancements and improvements.
# Background

## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Communication Services Infrastructures and Operations</td>
<td>9</td>
</tr>
<tr>
<td>2.2</td>
<td>Network Architecture and Topology</td>
<td>11</td>
</tr>
<tr>
<td>2.3</td>
<td>Mediation Processes and Dataflows</td>
<td>13</td>
</tr>
<tr>
<td>2.4</td>
<td>Service Quality Manager (SQM)</td>
<td>14</td>
</tr>
</tbody>
</table>
Nokia is a multinational company that, over the years, has found and nurtured success in several sectors, including cable, mobile devices, paper products, rubber boots and tires, and telecommunications infrastructure equipment.

The company is currently focused in five business areas: Mobile Networks, Fixed Networks, Applications & Analytics, IP/Optical Networks, and Nokia Technologies [10].

The objective of this project falls on the Applications & Analytics area.

In order to provide the best architecture to its customers, Nokia Solutions and Networks (NSN) follows industry standards and frameworks for its technologies, as well as for the Applications & Analytics area, namely those from TM Forum, explained in the next sections.

2.1 Communication Services Infrastructures and Operations

A Communication Services Provider (CSP), in addition to all the technology and integration with other manufacturers and partners, also needs to interact with customers by providing products, services, campaigns and support. For that purpose, CSPs follow international standards, recommendations, industry best practices, as well as architectural models for their infrastructures and operations. These guidelines, that are horizontal to all CSPs are very important to Nokia because they allow for the creation of tools that can deliver services that go beyond equipments exclusive to Nokia. This is very important because CSPs usually have heterogeneous networks in terms of technologies and equipments.

To facilitate business processes and technology integration to “greenfield” operators or changes to existing operators, the TM Forum, a global industry association (of operators, manufacturers, standardization bodies) developed a standard blueprint for Business and Operations Processes, named “Frameworx” [11]. This standard is based on a Service-Oriented Architecture (SOA) paradigm, promoting the best practices in the deployment of business and operations processes. With its adoption the CSPs can reduce operating costs and increase their business agility. The TM Forum “Frameworx” has the following components:

- Business Process Framework (eTOM) - initially conceived as “Telecommunications Operations Map”, this framework consists of the standardization and description of business and operations processes, further defining key elements and their interactions, ranging from the design and evaluation of products to the operation of infrastructures and interaction with suppliers and other stakeholders.

- Information Framework (SID) - corresponds to the standardization of terms, definitions and relationships to be used between stakeholders such as manufacturers and partners. With SID, conditions are created for ensuring that interoperability exists between different manufacturers, reducing the risks and integration costs.
- Application Framework (TAM) - consists of the description of systems and their functions as well as their interactions, providing a reference guide for Service Providers (SPs) and their partners.

**Figure 2.1: eTOM level 0.**

More relevance will be given to the eTOM framework, as this project has more bearing upon it. As illustrated in Figure 2.1, eTOM is divided in four parts: Customers; Operations; Strategy, Infrastructure, and Product and Enterprise Management.

Within eTOM the components that will be worked in this project will be situated at the Operations area, which is a fundamental part for a SP, as it corresponds to all the subsystems that will be managing the infrastructure. Some Operations Support Services (OSS) functions are “Network Inventory” and “Configuration”, service “Provisioning” and “Activation”, and “Fault Management” [12].

As illustrated in Figure 2.2, the eTOM operations level 2 processes consists of four groups. This project focuses more specifically on the “Assurance” group, which is a vertical end-to-end process that is responsible for the proactive and reactive enforcement of maintenance activities to ensure that the service provided to the client is continuously available and also that the contracted Service-Level Agreement (SLA) and Quality of Service (QoS) are met. This area has undergone an alteration in the paradigm, not being any more just a centralized view of network management but essentially a user-
centric view of the provided services. With this paradigm change the concept of KQI was born, which corresponds to indicator of the performance of services, rather than just the KPIs (metrics of the network), and the SLAs (the contractual minimum level of service quality).

### 2.2 Network Architecture and Topology

In order to get a notion of the NEs that normally constitute a Mobile Broadband network, of which the RTPM will receive the counters, a typical reference architecture topology is presented, based on the specifications of the 3rd Generation Partnership Project (3GPP), which is illustrated in Figure 2.3. Each operator has one or more Public Land Mobile Networks (PLMNs) which is identified by its Home Network Identity (HNI), allocated by a National Regulator. The HNI consists of the Mobile Country Code (MCC), which corresponds to the “country code” where the operator provides the service and the Mobile Network Code (MNC) which is a code assigned by the Regulator to the operator. An example of HNI for a mobile operator in Portugal (Vodafone Portugal), is 26801 [13], where the MCC corresponds to the value 268 and the MNC has the value 01. Another identifier, the International Mobile Subscriber Identity (IMSI), uniquely identifies a subscriber of an operator and is composed by the HNI and the Mobile Subscriber Identification Number (MSIN).
Within a PLMN there are two major components, the **Radio Access Network (RAN)** and the **Core**.

**Radio Access Network (RAN)**: The RAN comprehends all the radio equipment of the Access Network, i.e., antennas, BSs and the respective controllers. Depending of the technologies and Radio Systems used, a CSP RAN may have a mix of GSM/EDGE Radio Access Network (GERAN), Universal Terrestrial Radio Access Network (UTRAN) and Evolved Universal Terrestrial Access Network (E-UTRAN), respectively allowing second generation mobile (2G), 3G and 4G connectivity. Since there is no separate radio controller in LTE Systems, as are the cases of 2G and 3G technologies, because each “eNodeB” itself has that function, Nokia created the so called Multiradio Base Transceiver Station (MRBTS) to obtain a clear separation between the BS and the “controller” entity.
**Core:** The Core corresponds to the networks and systems where user management is executed and where the bridging is made between the radio links and other users, whether from the same operator or from other operators, as well as the Internet. The Core is still divided in a Circuit Switched (CS) domain that handles traditional voice calls and Short Message Services (SMSs), and a Packet Switch (PS) domain that handles all Internet Protocol (IP) data flows (voice and data) of users from 2G or 3G technologies. In 4G everything is based in IP, therefore the Core does not need to be split for voice and for data, and so, all packets for those services pass through the Serving Gateway (S-GW).

### 2.3 Mediation Processes and Dataflows

One of the key components in the Core, related with Operations, corresponds to the Mediation subsystems. Mediation has the responsibility to collect, receive and process information (events, measurements, accounting records, etc.) from all NEs, regardless of technology, as well as from probes located at different points of the infrastructure. An example of such a probe is one that is inserted between the Mobility Management Entity (MME) and the S-GW, to inspect all the traffic passing between those NEs with the objective of analysing and producing statistics, among others. For that purpose, Mediation systems use Adaptation agents, i.e., mechanisms that understand the protocols, formats, and connection modes of each NE or probes. The NEs send event records and measurements according to the format defined by 3GPP [14] usually called Measurement (MEAS). These are XML files with a well defined structure and all manufacturers use this standard. After receiving this information from the NEs and probes, the Mediation transforms this information into Open Measurement Standard (OMeS) XML format files that have an agnostic source, allowing the remaining components of the System to know how to extract information regardless of the NE, as can be observed in Figure 2.4, which depicts the current System design.

The OMeS are then fed to an “ET Loader” component that parses and loads them into a database.

The “Aggregation” aggregates at level of the objects, which is done before loading into the database, as no previous data is needed. It makes time Aggregations of a certain period, where it will need the data stored in the database in order to carry out these Aggregations.

The database contains the raw data from the “ET Loader” and also the temporal data (e.g., daily, weekly and monthly) resulting from aggregations. Depending on the type, that data has a different period of life.

With all the data stored in the database the PM is finally able to calculate all the necessary KPIs. After having been calculated, the PMs will export them to Comma Separated Value (CSV) files. In this way, CSV collector will load the KPIs in the SQM.

Currently, the collection and processing of NEs data is done with “NetAct” which in turn provides the
Figure 2.4: Dataflows in Current System Design with SQM.

data to the PM which in turn supplies them to the SQM.

2.4 Service Quality Manager

Service Quality Manager (SQM) is a software developed by Nokia with the objective of monitoring the quality of services, such as Voice over LTE (VoLTE) or IP Television (IPTV).

With this tool it is possible to identify future failures, for example, when a service is being monitored and a constant increase in degradation is observed (e.g., the number of responses in that service), that is a good indicator that the service may fail in the near future. That indicator reduces the time for troubleshooting, allowing preventive management measures to be applied instead of reactive management measures, which are triggered only when a failure occurs. This decrease in response time may normally be less than one hour instead of a day or more [15], turning therefore service operations simplified by optimizing, integrating and adapting the content for multi-vendor and multi-technology environments. Without a real-time or “near” real-time analysis, the success of preventive management would not be obtained. Hence the importance of this project and the need to reduce the processing time of KPIs for faster performance.

Figure 2.5 represents a general diagram of the SQM system with its main components. SQM collects data from the “NetAct” subsystem (Mediation) using the different types of collector components. The Configuration Manager (CM) collector receives settings, the Fault Management (FM) receives alarms,
and the Performance Manager (PM) and CSV collectors receive counters. The counters are received in PM and CSV due to existing legacy systems. Currently the KPIs are calculated within the “Collector”, in the PM component.

When Data enters in the system (from the respective collector) it is forwarded to a specific “Collector Queue” in messages using Java Message Service (JMS). These messages are called “events” and can contain groups of KPIs, FM data or CM data. The events are then “enriched” in the Event Engine Manager (EEM), meaning that the EEM checks if there are SQM Core instances interested in each of the events. If there are, then they are placed on the “Source Queue” and wait to be consumed by the interested SQM Core instances through the EEM. The processing order of events for each service is guaranteed because all events of a certain service enter through a specific queue, and only one consumer reads the events from each queue.

A practical example can be an instance for a 3G “NodeB” interested in events of missed calls and their success rate. When events of that type enter into the EEM, they are delivered to the respective “NodeB” instance that would then calculate KQIs from the data. In the end, the result of the analysis of the KPIs and the KQIs is performed by the SQM Core and is presented to the user through the Graphical
An example of an OMeS file, with the network hierarchy and the respective counters, is illustrated in Figure 2.6. All XML files have a unique timestamp and information (counters and topology) of NEs. A NE may be a Radio Network Controller (RNC) or a MRBTS. In this latter case, the source of information is a MRBTS with the identification number 8571, i.e., a LTE radio controller, as illustrated in line 224 of Figure 2.6. It is easy to follow the topology of this measure in that figure, where a MRBTS with the identifier 8571, is composed of a Flexi Multiradio BTS (LNBS) with the identifier 8571 and a Flexi Multiradio BTS Cell (LNCEL) with the identifier 20. The counters have the last level, in this case LNCEL. In line 228 of Figure 2.6 there is a “Distinguished Name” corresponding to the identifier of the PLMN, i.e., its MCC and MNC. The type of measure in this case is a LTE Power and Quality UpLink that contains the next counters on lines 230 to 257. This measure aims to provide the power level of the antenna and the LTE uplink connection quality. To calculate the KPI of this measure it is necessary to use the Equation (2.3) which in turn utilizes values from Equations (2.1) and (2.2). For the illustrated case Equations (2.1) and (2.2) produce the values of 86, 707 and 3,984 respectively. The final result
obtained from Equation (2.3), is then of 21.7638. This value corresponds to a Signal-to-Noise Ratio (SNR) measured in dB, with typical values for this KPI of around 15 dB or more, meaning that the value encountered is pretty good.

\[
X_1 = -10 \times M_{8005C96} - 9 \times M_{8005C97} - 7 \times M_{8005C98} \\
\quad - 5 \times M_{8005C99} - 3 \times M_{8005C100} - 1 \times M_{8005C101} \\
\quad + 1 \times M_{8005C102} + 3 \times M_{8005C103} + 5 \times M_{8005C104} \\
\quad + 7 \times M_{8005C105} + 9 \times M_{8005C106} + 11 \times M_{8005C107} \\
\quad + 13 \times M_{8005C108} + 15 \times M_{8005C109} + 17 \times M_{8005C110} \\
\quad + 19 \times M_{8005C111} + 21 \times M_{8005C112} + 23 \times M_{8005C113} \\
\quad + 25 \times M_{8005C114} + 27 \times M_{8005C115} + 29 \times M_{8005C116} \\
\quad + 30 \times M_{8005C117} \tag{2.1}
\]

\[
X_2 = M_{8005C96} + M_{8005C97} + M_{8005C98} + M_{8005C99} \\
\quad + M_{8005C100} + M_{8005C101} + M_{8005C102} + M_{8005C103} \\
\quad + M_{8005C104} + M_{8005C105} + M_{8005C106} + M_{8005C107} \\
\quad + M_{8005C108} + M_{8005C109} + M_{8005C110} + M_{8005C111} \\
\quad + M_{8005C112} + M_{8005C113} + M_{8005C114} + M_{8005C115} \\
\quad + M_{8005C116} + M_{8005C117} \tag{2.2}
\]

\[
LTE_{Pwr\ and\ Qual\ UL} = \frac{X_1}{X_2} \tag{2.3}
\]

With this type of equations, SQM can verify the state of equipment and services to provide the best user experience to its users. Nevertheless, to obtain a preventive management of equipment and services it is necessary to calculate KPIs and KQIs as quickly as possible to achieve “near” real-time monitoring. However, with the exponential growth of equipment and the volume of data that transits in the CSP networks, Nokia begins to reach the limit of scalability. A new module to solve this problem is presented in the next chapter.
System Design and Implementation

Contents

3.1 Architecture .............................................................. 21
3.2 Implementation Options ............................................... 22
3.3 Message Broker Tools ................................................. 22
3.4 Real-time Computation System Tools ............................. 23
3.5 System Implementation .................................................. 24
This chapter describes the general architecture proposed for this project as well as the tools used for the development of the Proof of Concept (PoC) solution finalizing with the description of its implementation.

3.1 Architecture

The general architecture proposed for this project is illustrated in Figure 3.1, which, when compared with Figure 2.4, shows a new RTPM module that will receive the OMeS files (and other types of files in the future) from Mediation, which are then loaded by an ETL component that parses the XML files to aggregate counters by KPIs.

As we are dealing with real-time information that comes in streams, it will be necessary to have a message broker to buffer the information because it will not get all the complete structure at once, among other reasons that will be discussed later.

Once obtained the information in such a way that it can be read and the counters aggregated by KPI, the Calculator component will perform the KPIs computation. Bringing again the previous example of OMeS illustrated in Figure 2.6 for the LTE Power and Quality UpLink measurement, the ETL of the new RTPM module, will parse the file and group counters from M8005C96 up to M8005C117, after which the corresponding KPI will be computed in the Calculator. As compound KPIs will exist, it is also necessary to re-feed the Calculator with KPIs already computed in order to perform aggregations at a higher level.

Between the ETL, the Calculator and the collectors (for SQM and PM) there is a Message Broker that will receive and send messages, which, in this case, are counters or KPIs. It would be possible to use a simpler solution, such as sockets, but given the system requirements, i.e., it must be scalable.
and able to parallelize the maximum possible tasks, it was obvious the decision for a Message Broker, providing low latency, high throughput, scalability, and reliability.

Additionally, using a well known Message Broker tool (already used in other projects at Nokia) the complexity of the project can be reduced, turning the solution modular, as the ETL and Calculator components do not need to know who the target is, when it is available and if it has received the messages; a method to send the data just needs to be called and the remaining work is performed by the Message Broker. Therefore, the Message Broker will have the responsibility of establishing a communication channel with the different components (Interoperability), ensuring that the data is received (reliability), and also ensuring that the preceding operations are carried out efficiently (performance), regardless of the number of components and volume of data (scalability) [16].

For the ETL and the Calculator components, a well known real-time computation tool will also be used (for the same reasons previously considered for the Message Broker), in order to provide scalable, efficient and fault-tolerant calculations.

3.2 Implementation Options

From the System Design, a decision needed to be taken in terms of the tools to be used for the implementation of the Message Broker, the ETL and the Calculator components, considering that the solution will be a distributed system, scalable, resilient, reliable, fault-tolerant and efficient.

The sections that follow describe the mentioned tools.

3.3 Message Broker Tools

For the Message Brokers the tools that were analysed were: “Apache ActiveMQ” [17], “Apache Kafka” [18], “RabbitMQ” [19] and “ZeroMQ” [20].

**Apache ActiveMQ:** “ActiveMQ” is an open-source Message-Oriented Middleware (MOM) and can be configured with message patterns of point-to-point and publish/subscribe [17]. It was designed for high availability, performance, scalability, reliability, and security. “ActiveMQ” was developed and is maintained by the Apache Software Foundation, it is written in Java with JMS, Representational State Transfer (REST) and WebSocket interfaces. In terms of transport it supports protocols such as Hyper Text Transfer Protocol (HTTP), Hyper Text Transfer Protocol Secure (HTTPS), IP multicast, Secure Sockets Layer (SSL), Transport Control Protocol (TCP), User Datagram Protocol (UDP) among others. Given its architecture it is easy to develop custom modules, e.g., core engine plugins, transport connectors, message dispatch policies, persistence adapters and network services.
Apache Kafka: “Kafka” is an open-source messaging system with basic publish-subscribe message patterns, designed to be fast, scalable, durable and distributed [18]. It was initially developed by LinkedIn Corporation, but now is an Apache project. It is written in Java, requiring a Java Virtual Machine (JVM) in order to run.

The tool has good documentation, is very simple to describe at a high level, is easy to scale out, has a very high throughput and permits persist messages on disk [21].

RabbitMQ: This tool is an open-source messaging system that can be configured for message patterns of Work Queues, Publish/Subscribe, Routing and Remote Procedure Call (RPC) [19]. “RabbitMQ” was designed to be reliable, easy to use and robust. It was developed and is maintained by Pivotal, and is written in Earlang, a functional language originally developed by Ericsson and made open-source in 1998.

ZeroMQ: This tool is also an open-source distributed messaging system with message patterns of request-response, publish-subscribe and pipeline [20]. It was designed to have patterns, QoS, performance and to be user-friendly. “ZeroMQ” was developed and is maintained by iMatix, it is written in C++, and can be used for various types of transport such as in-process, inter-process, TCP, and multicast. Besides supporting advanced messaging scenarios, the implementation of scenarios is done by combining various pieces of the “ZeroMQ” framework. As opposed to other message brokers, “ZeroMQ” does not have brokers implemented; therefore it is necessary to implement all necessary components.

3.4 Real-time Computation System Tools

For parsing of XML files, aggregating counters by KPI and calculation of KPIs in a large scale, the implementation requires a cluster of “machines” for parallel processing in a coordinated manner, therefore the most promising tools that were analysed for that purpose were: “Apache Flink” [22], “Apache Samza” [23], “Apache Spark Streaming” [24] and “Apache Storm” [25].

Apache Flink: “Flink” is an open-source system streaming dataflow engine [22]. It was designed to have high throughput, low latency, flexible windowing and fault-tolerance. “Flink” was developed by Stratosphere as a collaboration with Technical University Berlin and is currently an Apache Software Foundation project. Is written in Java and Scala. One or more sources originate streaming dataflows in the “Flink” program; a streaming dataflow is composed of operators interlinked by streams; an operator can be a source, transformation or sink; streams are intermediate results into which transformations are applied and from which result streams will originate; streams can be of two types: one-to-one, where they will always go from one determined subtask (portion of an operator) to another specific subtask, or redistribution, where each operator subtask sends data to different target
subtasks, depending on the selected transformation; finally, the dataflows end in one or more sinks. **Apache Samza:** “Samza” is an open-source distributed stream processing framework that was designed to have a simple Application Programming Interface (API), managed state, fault-tolerance, durability, scalability, pluggability, security and processor isolation [23]. It was developed by LinkedIn and is currently an Apache Software Foundation project. It is written in Java and Scala. “Samza” receives streams that are immutable messages of a similar type or category; messages can be appended to a stream or read from a stream, can have any number of consumers and the messages can optionally have an associated key which is used for partitioning; Jobs are applied to streams, i.e., a job is a logical transformation on a set of input streams through a piece of code, from which an output message is obtained that is added to an output stream.

**Apache Spark Streaming:** “Spark” is an open-source system for processing live data streams [24]. It was designed to be scalable, fault-tolerant and to have high throughput. “Spark” started as a research project at UC Berkeley in the AMPLab and is currently a project of the Apache Software Foundation. It is written in Scala, Java, Python and R languages. “Spark Streaming” is an extension of the “Spark Core”, meaning that the Core Engine is required to use “Spark Streaming” [26]. “Spark Core” contains the basic functionality of “Spark”, including components for task scheduling, memory management, fault recovery, interacting with storage systems, etc.

**Apache Storm:** “Storm” is an open-source distributed real-time computation system [25]. It was designed to be distributed, scalable, fault-tolerant, fast and reliable. It was developed by BackType and acquired by Twitter, being currently an Apache Software Foundation project. It is written in Java and Clojure. “Storm” receives streams in a component named Spout, which can be reliable or unreliable. The Spouts will send the data to Bolts where all the processing is done, e.g., filtering, functions, aggregations, joins, talking to databases, etc. After executing the transformation that they were designed to do, the Bolts will save the data in some type of storage or will send the data to another Bolt.

### 3.5 System Implementation

As previously explained, the new RTPM module, illustrated in Figure 3.2, receives OMeS that are XML files which are ingested by the ETL.

After obtaining the data from XML files for Java objects (Parser), it is necessary to split counters by KPI and wait for all counters, this is accomplished by the Merger component. Thus, the Merger is divided into two functions: one that “Split Counters by KPI”, and another that “Wait for All Counters” to check if it already has all the counters needed for a particular KPI, based on “Adaptation”. The “Split Counters by KPI” component, go through all the counters present in the XML and check if there is a KPI
that needs that counter. If there are one or more KPIs that use the read counters, a collection of these KPIs is created with the respective counters as value.

After having all the counters for the KPIs, the “Calculator” component receives the counter groups and asks for “Adaptation”, the formulas to calculate the KPIs, calculates them and sends them to the next component, the “Store Result”.

It is also necessary to have the information on the network topology, which is provided by the “Network Topology” component for the calculation of compound KPIs. For example, to know how many cells a certain eNodeB has so as to calculate a KPI aggregate of the various cells.

The “Store Result” component uses “kafka” to send the KPIs per technology to the SQM Collector. The “Apache Spark” tool was the choice for the ETL and calculator modules. The decision to use “Apache Spark” was based on the following:

• Increased maturity of the “Apache Spark” project with a larger number of commits and releases;

• To be an alternative that supports a lambda architecture where, in addition to real-time processing, batch processing can also be added, making possible in the future to expand this project to create reports, where off-line processing is necessary as an alternative to the use of “Hadoop”. Besides “Spark”, “Flink” also supports lambda architecture;

• In terms of guarantees, “Apache Spark” provides exactly-once, as does “Flink”. The other tools only offer at-least-once guarantees;

• “Spark Streaming” is stateful in terms of state management, just like “Samza” and “Flink”. “Storm”
is stateless, requiring the development of this function or the usage of a “Trident” tool;

• “Apache Spark” provides high throughput and low overhead of fault-tolerance mechanism;

• “Flink” would be an alternative with plenty of potential, and is more efficient than “Spark” [27].
   However, “Flink” is still in its initial stage and for this project it was very important to choose a tool
   that is mature and that has proven its success and with a high probability of its long-term support;

• Examples of companies using “Spark”: Yahoo!, NASA Jet Propulsion Laboratory (JPL), eBay Inc.
   and Amazon [28].

Another implementation decision, for parsing within the ETL, was to use Java API for XML Processing
(JAXP) [29]. It has three basic parsing interfaces, Document Object Model (DOM), Simple API for
XML (SAX) and Streaming API for XML (StAX).

StAX is used more specifically, namely for Streaming, which goes towards what this project involves.
Besides that, it has a light memory footprint, such as SAX, the navigation by XML performs better than
in SAX because it can jump elements that are not needed and achieves a higher throughput than with
the DOM [30].

To exchange the counters and KPIs between the modules, “Kafka” was chosen as depicted in Fig-
ure 3.2. The decision to use “Kafka” for the Message Brokers was based on the following:

• “Kafka” has better performance when compared with “ActiveMQ” and “RabbitMQ” [31]. However,
  the “ZeroMQ” also has high performance [32,33];

• Replication is possible among different nodes. Only “ZeroMQ” does not have this capability, be-
  cause there is no broker function, therefore a quite complex implementation would be required to
  obtain this function;

• “Kafka” is not the most mature tool, but it is used in many projects, such as “LinkedIn”, “Yahoo”,
  “Twitter”, “Netflix” and “Nokia” [34]. So it already has a very strong track record.

3.5.1 Extract, Transform and Load (ETL)

As illustrated in Figure 3.3, the ETL consists of two parts: the driver program that runs on “Spark”’s
master node and the other part where all the processing is done in the workers. When the ETL starts,
the driver program cleans the database so there are no old counters through the “Clean BD” and the
“Check old counter DB” launches a thread that clears all counters that are older than one determined
time in the database and in the buffer that is used by the “Wait for all Counters” component. This
thread runs periodically with a programmable time interval. Then the driver program starts monitoring
the new files with counters in the specified shared directory with the “Spark” built-in component called
“TextFileStream”. The files are then distributed among the workers. These are processed and result in a list of objects called “CounterData” that contain all the counters needed to calculate a certain KPI for a given network element and moment in time. Each Worker consists of two parts, the first called “Parser” where the useful information of the files is identified (Identify Sources) and extracted (Parse Files), and the second part called “Merger” that groups the counters by KPI (Split Counters by KPI) and, when it obtains all the counters to calculate the KPI, sends them to “Kafka” (Wait for all Counters). For the ETL to be able to run, it must receive a parameter with the path to a file with properties. Each line of this file must have the following format: `property = value`. The Table 3.1 contains the possible properties that can be specified in this file.

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
<th>TYPE</th>
<th>EXAMPLE</th>
<th>MANDATORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>app.name</td>
<td>Name of the application that is passed on to the Spark.</td>
<td>String</td>
<td>ETL</td>
<td>yes</td>
</tr>
<tr>
<td>kpi.registry</td>
<td>IP address and port of component that provides the definition of KPIs.</td>
<td>IP_ADDRESS:PORT</td>
<td>10.10.10.10:22222</td>
<td>yes</td>
</tr>
<tr>
<td>filesystem.data.provider</td>
<td>Directory where the files that are consumed by ETL will be placed.</td>
<td>PATH</td>
<td>/shared/files/</td>
<td>yes</td>
</tr>
<tr>
<td>counter.data.queue</td>
<td>IP address and port of the Kafka server.</td>
<td>IP_ADDRESS:PORT</td>
<td>10.10.10.10:9092</td>
<td>yes</td>
</tr>
<tr>
<td>cassandra.ip</td>
<td>IP address of the Cassandra server.</td>
<td>IP_ADDRESS</td>
<td>10.10.10.10</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Identify Sources**: This component receives Strings, which are the lines of the new files. Each file is run by a transformation of the “Spark” called `mapPartition()`. This type of transformation is chosen because it allows reading the files completely, since each file is a partition and is only processed by
one worker. Another type of transformation can be used as a \texttt{map()}, in this case “Spark” would split the files and distribute them amongst the workers, losing the order and where each row belongs. The files have to be processed in full, otherwise the information could be lost, such as the timestamp or to which NE a certain counter belongs. The “Identify Sources” component is illustrated in Figure 3.4, and it reads the lines of the files until it is able to identify their type in the “Identifier” sub-component. After successfully identifying, it returns an object called “Source” that has the capacity of creating a FileReader per file. In the case of this PoC solution, the file types analyzed were only of OMeS type. To add new types, a new class should be created that implements \texttt{SourceTypeFactory<Type>} and puts it in the package rtpm.spark.source.types.

The FileReader object consists of the file type and its contents and the start and end code for commenting on this type, thus ignoring lines that are comments that are not useful for this tool.

**Parse Files:** This component is illustrated in Figure 3.5, it receives FileReader objects that were created by the “Identify Sources” component, depending on the type of FileReader objects (e.g., OMeS), creates a “Parser” that has the ability to process this type, because it knows the structure of the file. This “Parser” extracts all information from each file, such as timestamp, interval, network element, measure type, counters and their values. In order to extract the information of the OMeS type files, the “Parser” of this type uses StAX, where it reads all the lines of the XML files and, depending on the tags found the CounterListerner, stores the useful information in a buffer of objects called MeasurementData. That is, the “Parser”, when it identifies a counter, sends it to the CounterListerner where it checks if there exist a MeasurementData of that measurement and NE and, if it exists, it adds
the counter to the existing ones, otherwise it creates a new MeasurementData and saves that counter with the technology information, timestamp, interval, NE and type of measure. For example in the Listing 3.1, “Parse Files” creates a new MeasurementData for the type of measure SCSCF that is in line 8. To create this object it reads line 3 to save the timestamp and the interval and in line 6 it saves the NE with its topological hierarchy. Then it adds all the counters that are from line 10 to line 28 and all the counters that are omitted in this example as well. Finally, through the CounterListener buffer, the “Parser” returns a list of MeasurementData. Each MeasurementData contains counters of a measurement and a NE. Each of these contains the technology, timestamp, interval, network element, type of measurement and the counters and their respective values.

Listing 3.1: Part of OMeS File tested

```xml
<OMeS version="2.3">
  <PMSetup startTime="2017-04-12T15:39:41.402+03:00" interval="1">
    <MO dimension="network_element">
      <DN>CSCF-1/SCSCF-1</DN>
      <!-- Call Session Control Function 1 - Serving CSCF 1 -->
    </MO>
    <PMTarget measurementType="SCSCF">
      <!-- DATA OMITTED -->
      <Mif400RespSentRegCtr>432530</Mif400RespSentRegCtr>
      <Mif401RespReclnInvCtr>55149</Mif401RespReclnInvCtr>
      <Mif401RespSentInvCtr>204911</Mif401RespSentInvCtr>
      <Mif401RespSentRegCtr>620688</Mif401RespSentRegCtr>
      <Mif403RespReclnInvCtr>205690</Mif403RespReclnInvCtr>
      <Mif403RespSentInvCtr>592026</Mif403RespSentInvCtr>
      <Mif403RespSentRegCtr>416660</Mif403RespSentRegCtr>
      <Mif404RespReclnInvCtr>709548</Mif404RespReclnInvCtr>
      <Mif404RespSentInvCtr>60796</Mif404RespSentInvCtr>
      <Mif404RespSentRegCtr>76588</Mif404RespSentRegCtr>
      <Mif408RespReclnInvCtr>878883</Mif408RespReclnInvCtr>
      <Mif408RespSentInvCtr>45481</Mif408RespSentInvCtr>
      <Mif408RespSentRegCtr>318111</Mif408RespSentRegCtr>
      <Mif480RespReclnInvCtr>147307</Mif480RespReclnInvCtr>
      <Mif480RespSentInvCtr>861643</Mif480RespSentInvCtr>
      <Mif480RespSentRegCtr>648526</Mif480RespSentRegCtr>
      <Mif482RespReclnInvCtr>162326</Mif482RespReclnInvCtr>
      <Mif482RespSentInvCtr>559899</Mif482RespSentInvCtr>
      <Mif482RespSentRegCtr>377855</Mif482RespSentRegCtr>
      <!-- DATA OMITTED -->
    </PMTarget>
  </PMMOResult>
</PMSetup>
</OMeS>
```

Split Counters by KPI: As shown in Figure 3.6, this component receives MeasurementData's and each of these contains all the counters of a certain measurement and of a certain NE and, in the end, it returns a collection of CounterData's. CounterData are objects that have the data for the calculation of a KPI when complete.

Firstly, the CounterGrouper for each counter present in each MeasurementData asks KpiRegistry which KPIs use this counter, since a counter can be used by zero or more KPIs. There is an object called CachedKpiRegistry that is a buffer and stores all data requested to KpiRegistry to improve performance, decreasing network traffic and waiting time until it receives KpiRegistry data. For each
returned KPI a CounterData and a key are created. The Key is created based upon the returned KPI and the timestamp and NE found in the MeasurementData. A Key Value Pair collection is created in this way, where for each key its CounterData is stored as value, this list is stored in an object called CounterDataMap that is managed by the CounterGrouper. Thus, each KPI uses one or more counters of a NE that were generated at the same moment in time. CounterData’s objects consist of a key (composed of KPI, timestamp and NE), technology, a KPI, a timestamp, a NE with hierarchy, the parent of this NE and by the counters and their respective values. An example of a CounterData can be viewed in Listing 3.2, where a CounterData from the MeasurementData of Listing 3.1 for the KPI cscf_37a is shown. Finally, only the values of the CounterDataMap collection, that is, CounterData’s, are returned.

**Figure 3.6:** System Implementation – ETL Merger – Split Counters by KPI component.

**Figure 3.7:** System Implementation – ETL Merger – Wait for All Counters component.

**Wait for all Counters:** As shown in Figure 3.7, this component receives CounterData’s and, for each one, the “All Counters Ready” extract the KPI and ask “CachedKpiRegistry” for the complete list of counters for that KPI. With the list of all the counters, the “All Counters Ready” verifies if each
CounterData has all its counters. When a CounterData has all the counters needed to calculate the KPI, it is sent to the “Producer” that sends to “Kafka” with the topic rtpm_counter_data and a key that is for the first two levels of the network hierarchy. For example, for network hierarchy PLMN-PLMN/CSCF-1/EATF-1 the key would be PLMN-PLMN/CSCF-1.

If all the counters are not yet available, there is an attempt to complete by going to “Cassandra” to check if there are any counters that have come in a previous file processed by another “Spark” Worker. If it has been processed by the same worker, it is found in an object called CounterDataManager that buffers each worker and manages the connection to the database. If it is in “Cassandra” or in the buffer and if with these it is complete, that is, with all the counters for the calculation of this KPI, it sends it to the “Producer”.

If it is still incomplete, it saves the new counters in the buffer and in “Cassandra”. If there is no such CounterData in the buffer and in “Cassandra”, it just adds it to the buffer and to “Cassandra” for later use. The CounterData stored in these two places after a specified time is erased (by the “Check old counter DB” launched in the driver program), having no further utility because they are old.

### Listing 3.2: Object CounterData after Avro Serialization

```json
{
  "technology": "nokia",
  "kpi": "cscf_37a",
  "starttime": 1492000781402,
  "networksubsetCurrent": "PLMN-PLMN/CSCF-1/SCSCF-1",
  "networksubsetParent": "PLMN-PLMN/CSCF-1",
  "countervalue": {
    "mif482respsentregctr": 377835,
    "mif404respsentregctr": 76588,
    "mif480respsentregctr": 648526,
    "mif403respsentregctr": 416660,
    "mif408respsentregctr": 318111,
    "mif400respsentregctr": 432530,
    "mif401respsentregctr": 620688,
  },
  "key": 1507581990
}
```

### Adaptation / KPI Registry: The KPI Registry is an external application to this project, provided by Nokia, using sockets to communicate with other applications, and allowing to obtain the following information:

- List of counters to calculate a KPI;
- List of KPIs used by a counter;
- Formula to calculate a certain KPI;
- How to aggregate two equal counters (average, sum, maximum or minimum);
- Whether the calculation of a KPI in a given network level is applicable.

In order to calculate the KPIs in upper levels (CounterData upper level) that are explained in Section 3.5.2, it is necessary to know how to do the aggregation of equal counters. In this PoC this...
aggregation can be a mean, sum, maximum or minimum. Thus, to calculate a upper-level KPI it is necessary to aggregate several CounterData’s and the way to accomplish it is to extract the same counter from two CounterData and to aggregate them and apply this procedure to all counters. This gives rise to a single CounterData where the formula can be applied to calculate the respective KPI.

There are certain network levels where KPI calculation is not applicable. These were created to parallelize the calculation of KPIs at the higher level. These levels are called intermediate levels and always end in “*” instead of a specific identifier. A concrete example of a NE with the network hierarchy PLMN-PLMN/MRBTS-15/LNBTS-162/LNCEL-3 up to the PLMN level can be checked in Table 3.3, where the intermediate levels are the underlined lines where the parent always ends with “*”. In this example of a 2G cell it is intended to calculate the KPIs at four different levels: in the BTS cell with the identifier 3 (LNCEL-3), in the BTS with the identifier 162 (LN BTS-162) where it has several cells, at the level of BSC with identifier 15 (MRBTS-15) where it will has various BTSs and finally in PLMN.

When checking the Table 3.3 there are three intermediate levels for the calculation of the BSC 15, this means, first the counters of the BTSs aggregate where the identifiers begin with 16; next, the counters of the BTSs that start with 1 aggregate and, at the same time, the counters of the BTSs starting with 2 are aggregated; at the end, the BTS counters beginning with 1 are aggregated with the ones remaining, for example those starting with 2, in other words, the aggregation of all the BTSs of BSC 15.

To improve performance, an object was created for this project called CachedKpiRegistry which stores all the information requested of the KPI Registry. In this way it decreases the time of communication with the KPI Registry. In the event that the information in the KPI Registry changes, such as a formula or adding/removing KPIs or counters, there is a version that changes every time there is a modification in the KPI Registry. The CachedKpiRegistry periodically checks to see if it has the latest version and, if there is a new one, it deletes all the buffer contents.

In addition to improving performance, it also has the function of creating an interface that makes the request of this information transparent. Whenever it is necessary to change the external application that provides this information, simply create a new class that implements KpiRegistryFactory and override the method create() where it creates an object that knows how to communicate with a new application and implements KpiRegistry, where it does an override to the methods which return the information described previously.
Cassandra: “Spark” provides two ways of sharing information between nodes; they are “Broadcast” variables and “Accumulators”. “Accumulators” are only meant to add objects in an associative and commutative way, so they do not fit the need to share incomplete CounterData that are added, changed (counter addition) and deleted (when complete) in this project. “Broadcast” variables are read-only objects that are replicated by the workers, and in this way they are not suitable because it is not possible to change or remove CounterData. Since “Spark” does not provide a way to share CounterData as required in this project, “Apache Cassandra” was chosen because it is a distributed database, offers linear scalability, high availability, replicating across multiple datacenters, lower latency and also because it works very well With “Spark” [35].

Kafka: The serialization of the data when sending it to “Kafka” is being performed with a tool called “Apache Avro” [36], so any application that wants to acquire the data must use this tool and know the scheme used in the objects in question.

The schemes used in this project are shown in Listings 3.3 and 3.4.

Listing 3.3: CounterData Schema
```
1 { "type":"record",
2   "name":"class rtpm.counter.CounterData",
3   "fields": [
4     { "name":"technology", "type":"string" },
5     { "name":"kpi", "type":"string" },
6     { "name":"starttime", "type":"long" },
7     { "name":"networksubsetCurrent", "type":"string" },
8     { "name":"networksubsetParent", "type":"string" },
9     { "name":"countervalue", "type":
10       { "type": "map", "values": "int" } },
11     { "name":"key", "type": "int" }]
```

Listing 3.4: KpiResult Schema
```
1 { "type":"record",
2   "name":"class rtpm.calculator.kpi.KpiResult",
3   "fields": [
4     { "name":"technology", "type":"string" },
5     { "name":"kpi", "type":"string" },
6     { "name":"starttime", "type":"long" },
7     { "name":"networksubsetCurrent", "type":"string" },
8     { "name":"networksubsetParent", "type":"string" },
9     { "name":"value", "type": "int" }]
```

To create parallelization in “Kafka” when sending the ETL counters to the “Calculator”, keys were used in sending the messages. These keys are the first two levels in the hierarchy of the NEs in the case of counters (CounterData). Each Key belongs to a single partition on the “Kafka” and so it is guaranteed that there is only order for messages that have the same key, since there is no need for order in the different second levels of the network hierarchy, so one can parallelize the processing.
This imposes a limit of parallelism depending on the topology of the network, more specifically the amount of second level of the network hierarchy existing in the topology.

**Tuning and optimization:** “Spark” does not really work in stream, unlike for example “Flink”. What “Spark” does is simulate a stream, this is accomplished by creating periodic batches called DStream. In this PoC periods of 10 seconds were used for the ETL to check new files and for the “Calculator” to poll new messages in “Kafka”. Within each DStream the mapPartitions method is called several times where the processing is performed, so the ETL is called whenever there is a new file and the “Calculator” whenever new messages arrive to a partition. The problem with mapPartitions is that whenever it is called all objects that are in the code of this method are created.

For example, in the WaitForAllCounters in ETL for each file a new “Producer” is created to send the CounterData and, soon after that, the same connection is closed. For a scenario where 1000 files are read where “Spark” only has one machine with one Central Processing Unit (CPU) core, 1000 instances of “Producer” are created, which implies 1000 connections to “Kafka”. In addition to overloading the “Kafka” with multiple connections immense time and processing is spent to create these instances where only one instance would be required. In addition to the connections to “Kafka”, there are also connections to “Cassandra”, KpiRegistry and the problem of objects that are shared within the same core as CounterDataBuffer, CounterGrouper and NetworkDataTracker. This implies that a large amount of objects and their connections are created unnecessarily.

To solve this problem the singleton pattern has been applied to these classes, but with a variant that takes into account that this project works on a cluster with several different machines and each of them with several different CPU cores. In this way there are two problems: (i) how to identify different machines so that each one has its own instances, and (ii) within each machine how to create only one instance for each CPU core.

To solve the first problem a very simple way was used that does not imply having to identify the different machines of the cluster, when the Master of “Spark” sends the objects to the Workers these objects were defined as transient. When defining the objects in this way Java is being informed that these objects should not be serialized, so it is almost as if we were running the code in only one machine, because these objects are instantiated on all the machines independently.

Now that on all machines the behaviour is as if they are independent, it is necessary to create instances for each CPU core. In order to solve this problem a static HashMap has been created in each class where the keys are identifiers of the CPU cores and in the value the instances for each CPU core are stored. To create the instances the method called getInstance was created which
is static and synchronized, so it cannot be executed simultaneously in several different CPU cores inside the same machine, since they modify the HashMap. Whenever getInstance is called upon, it is checked if there is in the HashMap the identification of the thread that is executing it. If it does not exist (initial situation) it will create a new instance of the object in question and saves it in the HashMap with the identification of the thread. The identifier is easy to obtain, since Java provides this information by calling the static method Thread.currentThread().getId(). If a key of the thread that is running exists in the HashMap, it only returns the saved instance and in this way it is only created once for each CPU core and this connection is only closed when ETL is finished.

In a scenario of four virtual machines and each one with four CPU cores, in a total of 16 cores, there are only four instances on each machine, which totals 16 instances of each of the previous objects in the entire cluster.

3.5.2 Calculator

As shown in Figure 3.8, the “Calculator” consists of two components that are executed in the Workers, the “Calculate KPI” that receives counters and calculates KPIs and the “Store Result” that sends the calculated KPIs to the “Kafka”.

![Figure 3.8: System Implementation – Calculator component.](image)

The driver program in “Calculator” only manages the consumption counters connections in “Kafka” and in this way distribute the CounterData’s to the Workers according to the keys. The connection to “Kafka” is performed with the spark-streaming-kafka component and the Direct Stream approach is used. This provides simple parallelism, 1:1 correspondence between “Kafka” partitions and “Spark”
partitions. The “Calculator” must receive two parameters in order to work:

- The first one with the path to a file with properties. Each line in this file must have the following format: property=value. Table 3.2 contains the properties that can be specified in this file;
- The second one with the path for the files with the network topology with the extension .template.

Table 3.2: Calculator Properties File

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
<th>TYPE</th>
<th>EXAMPLE</th>
<th>MANDATORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>counter.data.queue</td>
<td>IP address and port of the Kafka server that have the counter topic.</td>
<td>IP_ADDRESS:PORT</td>
<td>10.10.10.10:9092</td>
<td>yes</td>
</tr>
<tr>
<td>kpi.result.store</td>
<td>IP address and port of the Kafka server that have the KPI topic.</td>
<td>IP_ADDRESS:PORT</td>
<td>10.10.10.10:9092</td>
<td>yes</td>
</tr>
<tr>
<td>kpi.registry</td>
<td>IP address and port of the component that supplies the KPIs definitions.</td>
<td>IP_ADDRESS:PORT</td>
<td>10.10.10.10:22222</td>
<td>yes</td>
</tr>
</tbody>
</table>

Figure 3.9: System Implementation – Calculator – Calculate KPI module.

Calculate KPI: This component, illustrated in Figure 3.9, receives “Kafka” messages produced by the ETL with the topic rtpm_counter_data that come in tuples with a key and a CounterData. As mentioned in the ETL, the key are the first two levels of the hierarchy of the NE. First, NetworkDataTracker verifies that each CounterData has already been processed through its key, confirming that it is contained in a list of CounterData Keys that has been processed. If this is the case it does not take any action, because the respective KPI has already been calculated. In case it has not yet been processed, the key is added to the list of CounterData’ processed and initiates the procedures for calculating the KPI. There are two types of CounterData, the raw and the upper level. The raw ones are the CounterData that are generated by ETLs that come from the counters that have been received. The upper levels are CounterData generated from another CounterData where the NE and the parent are in a higher level in the network hierarchy to calculate higher level KPIs. All the hierarchical levels of a raw CounterData up to the top level, the PLMN, can be viewed
in the Table 3.3. In this case, four KPIs are calculated for the following NEs: LNCEL-3, LNBTS-162, MRBTS-15 and PLMN.

Table 3.3: CounterData Subset

<table>
<thead>
<tr>
<th>Type</th>
<th>Network Element:</th>
<th>Parent:</th>
</tr>
</thead>
<tbody>
<tr>
<td>upper</td>
<td>PLMN-PLMN/MRBTS-1</td>
<td>PLMN-PLMN</td>
</tr>
<tr>
<td>upper</td>
<td>PLMN-PLMN/MRBTS-15</td>
<td>PLMN-PLMN/MRBTS-1</td>
</tr>
<tr>
<td>upper</td>
<td>PLMN-PLMN/MRBTS-15/LNBTS-1</td>
<td>PLMN-PLMN/MRBTS-15</td>
</tr>
<tr>
<td>upper</td>
<td>PLMN-PLMN/MRBTS-15/LNBTS-16</td>
<td>PLMN-PLMN/MRBTS-15/LNBTS-16</td>
</tr>
<tr>
<td>upper</td>
<td>PLMN-PLMN/MRBTS-15/LNBTS-162/LNCEL-3</td>
<td>PLMN-PLMN/MRBTS-15/LNBTS-162/LNCEL-3</td>
</tr>
</tbody>
</table>

If it is a CounterData upper level, it checks if it is complete, that is, NetworkDataTracker verifies that there are all CounterData of the lower levels of the parent of the NE. If it is incomplete, the hierarchy of this CounterData is added in a buffer where it is stored as key a key of CounterData (because it is a hash based on the parent that is the same in all the NE that have the same parent) and in the value a list of the Hierarchies of the NEs (the hierarchy with NE is stored to verify in an easy way the existence of all the NEs of a parent). If it is a new NE in the buffer, it is checked if it has all the NEs to calculate the KPI of the parent. This is done by asking the “Network Topology” all the NE of a certain parent and with these it is verified if the buffer contains all these NEs and, if so, it marks this CounterData as complete. In CounterDataBuffer there is another map that buffers CounterData. This checks if there is a key for each CounterData on the map, if it does not exist, add it. If it exists, it performs an aggregation of CounterData counters stored in the buffer with what has just arrived and updates this CounterData in the buffer, this aggregation is only done in CounterData upper level. To perform the aggregation it is necessary to ask the CachedKpiRegistry the aggregation formula of these counters that can be an average, sum, maximum or minimum. Then the CalculateKpiValue checks through the NetworkDataTracker if each CounterData is complete. This is always true for the raw CounterData. For the upper level CounterData it is necessary to have all CounterData buffered in the lower hierarchy of the CounterData parent. If the CounterData is complete, it is removed from the CounterDataBuffer and the CalculateKpiValue calculates its KPI. For the CalculateKpiValue to calculate the KPI, it first needs to check whether the KPI calculation is applicable in the network level of each CounterData, that is, if the parent’s network level is not an intermediate level. The intermediate level’s objective is to perform the parallel calculation of higher level KPIs. This process is explained in Section 3.5.1. If it is not applicable, the KPI is not calculated. If applicable, the CachedKpiRegistry is asked for the formula for the KPI in question. After receiving the formula, it is checked whether the formula is valid, and, if so, the formula is applied to the counters. The result is stored in a new object called KpiResult and is sent to the “Store Result”.

37
The formulas that are requested of the “KPI Registry” come in the format of the Structured Query Language (SQL) query, where there are values and in the middle of these there are operations and parentheses are used to define the order of calculation. The values can be a numeric number or a pair String, Number called Nvl, where the String is the name of the counter and the Number is the default value. The Nvl has as an objective to use the counters extracted by the ETL. An example of a formula received by the KPI Registry is illustrated in the Figure 3.10.

Formulas received by the KPI Registry are converted to a new formula to be used by the “Calculator”. An example is illustrated in the Figure 3.10 in step 1. The calculation of the KPIs are recursively performed as a binary tree. Each node is an operation (for example sum, subtraction, division ...), and each node has two branches that can be another node (operation) or a leaf. The leaves can be of two types, a numeric value called Literal or an Nvl. The implementation of this system is done with an interface called Function where it has a method called Apply that receives a Map where the keys are strings (counter name) and the values are a number (the values of the counters) and it returns a number (KPI result). Each operation must have a class that extends from the PrimitiveFunction class and where the Function interface is implemented. This new class PrimitiveFunction always has two Function variables that are the branches. The classes of the operations override the Apply method where the operations are done, for example in the class Add the sum of the two Function variables is done. With the formula converted by the “Calculator” this Function object is created with the respective tree with the operations, Nvl’s and Literals. The JSqlParser framework is used to parse the formula and construct the Function object using the Visitor Pattern. Finally, the Apply method is called recursively always passing the counters with the respective values that are used by the Nvl. The end result is a single value.

KpiResult contains the technology, KPI, timestamp, network element, network level and the value of the KPI. Finally, a new CounterData upper level of the CounterData received is created by the CalculateKpiValue, if there is a parent. This new CounterData is sent to “Kafka” through the CounterData’s Object Producer (topic "rtpm_counter_data") for the KPI calculation at the next upper level. This way the “Calculator” is fed with new objects which turns it into an amplifier.

Store Result: This component, illustrated in Figure 3.11, receives a list of KpiResult and send them to “Kafka” through the KpiResult Object Producer (topic kpi.result.store).

An example of a KpiResult used in this PoC is illustrated in Listing 3.5, which follows the example of Listings 3.1 and 3.2 (OMeS and CounterData).
Network Topology: For this PoC the network topology is obtained internally by the “Calculator”, that is, the “Calculator” is given a set of files that are analysed, from where the network topology is extracted. These files are XML files where all the NEs must be contained within their hierarchy defined in `<DN>` tags. An example of these files can be observed in Listing 3.6.

However, the code is designed to allow the reception of the network topology to be changed by another external application, giving the NetworkTopology a new class that implements the
Listing 3.6: Network Topology construction

```xml
<?xml version="1.0" encoding="UTF-8"?>

#pm_setup

<DN>PLMN-PLMN/MRBTS-15</DN>
<DN>PLMN-PLMN/MRBTS-15/LNBTS-162</DN>
<DN>PLMN-PLMN/MRBTS-15/LNBTS-162/LNCEL-1</DN>
<DN>PLMN-PLMN/MRBTS-15/LNBTS-162/LNCEL-2</DN>
<DN>PLMN-PLMN/MRBTS-15/LNBTS-162/LNCEL-3</DN>
<DN>PLMN-PLMN/MRBTS-15/LNBTS-176</DN>
<DN>PLMN-PLMN/MRBTS-15/LNBTS-176/LNCEL-1</DN>
<DN>PLMN-PLMN/MRBTS-15/LNBTS-176/LNCEL-2</DN>
<DN>PLMN-PLMN/MRBTS-15/LNBTS-176/LNCEL-3</DN>
<DN>PLMN-PLMN/MRBTS-11</DN>
<DN>PLMN-PLMN/MRBTS-11/LNBTS-92</DN>
<DN>PLMN-PLMN/MRBTS-11/LNBTS-92/LNCEL-1</DN>
<DN>PLMN-PLMN/MRBTS-11/LNBTS-92/LNCEL-2</DN>
<DN>PLMN-PLMN/MRBTS-11/LNBTS-92/LNCEL-3</DN>
<DN>PLMN-PLMN/MRBTS-11/LNBTS-92/LNCEL-4</DN>
```

The Provider used for this PoC reads all the files with the extension .template, where it parses them and extracts all lines that contain the `<DN>` tag to extract the NE and its hierarchy. In this way all the NEs and their hierarchies are obtained, making it easy to identify the NEs that are of levels inferior to a certain NE.
4

Evaluation

Contents

4.1 Evaluation Environment Specifications .................................. 43
4.2 ETL and Calculator evaluation ............................................. 45
For the evaluation of the PoC solution, several scenarios were set up in order to determine if the performance requirements from Nokia were met, namely in terms of synchronous vs. asynchronous processing, RAM utilization, different types of input files, partitioning of topics and scalability.

4.1 Evaluation Environment Specifications

The Evaluation Environment consisted on seventeen virtual machines, each with several Virtual CPUs (vCPUs) in order to test the PoC solution in conditions similar to a production system.

**Virtual machine characteristics:**
- 4 vCPU cores (2.8 GHz per core);
- 16 GB of RAM;
- Operating system: Red Hat 7.2.

**System Configuration:**
- VM01: Spark Master, Kafka, Zookeeper, Cassandra and Network File System (NFS) server;
- VM02 to VM17: Spark Worker and NFS client.

**Software Configuration:**
- Java version: 8u112;
- Spark Core and Streaming version: 2.1.0;
- Kafka version: 0.10.1.1;
- Zookeeper version: 3.4.9;
- Cassandra version: 3.9;
- Avro version: 1.8.1;
- Shared folder: NFS.

In each ETL performance test, 5000 files (OMeS) were processed which resulted in 695000 CounterData, but only 682500 were complete (they have all the counters needed to calculate the KPI). Regarding the performance tests of the “Calculator”, 682500 CounterData were received that produced 1045000 KPIs.

The OMeS files were generated based on two template OMeS files provided by Nokia with the following characteristics:
• Template 1: This file has 975 lines and 851 counters. It contains 275 CounterData, of which 270 are complete. Thus approximately 98% of the CounterData are complete and have approximately 87.3% of counters per line and 0.277 CounterData per line;

• Template 2: This file has 359 lines and 81 counters. It contains 3 CounterData and they are all complete. It has approximately 22.6% counters per line and 0.008 CounterData per line.

4.1.1 Spark setup

First, it is necessary to launch the Spark Master and the Workers to build a cluster. In the case of this PoC the Spark standalone cluster mode was used, where a cluster is built executing the script start-master.sh in the virtual machine (VM) where the master is meant to run and the script start-slave.sh <master-spark-URL> on the VMs that are intended to be workers.

With the cluster running, the script spark-submit was used to perform tasks in Spark. It receives arguments such as the name of the class to be executed, the master’s address, the Random Access Memory (RAM) memory reserved for the Spark for each machine, the total number of cores which are meant to be used and the Java ARchive (JAR) file with the compiled Java code as well. In this case, several clusters were built with different amounts of VMs to test the performance. To simulate VMs with different amounts of cores, the argument core total was supplied with various different quantities from 1 to 64. In addition to the cores, different values were also given in the RAM argument reserved for Spark for each machine.

4.1.2 Cassandra setup

Cassandra, in addition to being a distributed database, only used one node to evaluate the worst-case scenario. The cluster name was defined as RTPM. The IP of the master was defined in the listen_address, rpc_address and seeds parameters as well, and, since the Cassandra cluster is just a machine, the seed parameter only has the IP of the master.

4.1.3 Kafka and Zookeeper setup

Similar to Cassandra, only one machine with a Kafka server was used to assess the worst-case scenario. To configure Kafka, the broker.id parameters were defined with the value “1”, the listeners and the zookeeper.connect with the master IP, since both Kafka and Zookeeper were only executed in the master. To configure Zookepeer, the server.1 parameter with the IP of the master was set.
4.2 ETL and Calculator evaluation

The scenarios were performed with a VM with Spark Master, Kafka, Zookeeper, Cassandra and NFS server and several VMs for the workers, which in the latter case, were always in exponentials of two. In this way we were able to evaluate if the ETL and the Calculator have scalability close to linear.

The first scenario was to check the performance difference between having Kafka configured in synchronous and asynchronous modes in two different environments. This test also aimed to evaluate the scalability of the system in a cluster with several VMs.

The second scenario was to evaluate the impact of the amount of RAM reserved for Spark workers in ETL and Calculator.

The third scenario was to evaluate the impact of files with different characteristics when processed by ETL. The consistency of the ETL was also evaluated by having performed the same task with the same input several times.

The fourth scenario evaluated the performance of Calculator with different amounts of partitions configured in Kafka in two distinct environments, these being the synchronous and the asynchronous modes. The consistency of the Calculator was also evaluated by having the same task performed several times with the same input.

In the fifth and final scenario, the performance of the ETL and the Calculator with several quantities of VMs was evaluated in order to ascertain the scalability of these tools and to try to define the number of VMs necessary for the three types of networks illustrated in the Table 4.1.

4.2.1 Synchronous versus asynchronous

In the first environment two VMs are used, the first (VM01) with Spark Master, Kafka server, Zookeeper server, Cassandra server and NFS server and the second (VM02) with Spark Worker and NFS client. In this way all the processing of the files and counters were executed in only one machine, having used 1, 2 and 4 cores.

In the second environment five VMs were used, where one (VM01) runs the Spark Master, Kafka server, Zookeeper server, Cassandra server and NFS server and the remaining four were the workers. The generation of OMeS files in all tests was done with a small program that uses two templates; it changes the timestamp to the current time and adds identifiers to the NEs, finally creating as many OMeS files as requested.

To test the Calculator, another application was made that takes the ETL output of 682500 CounterData stored a priori and sends them to the Kafka in the respective CounterData topic. The ETL is therefore simulated, but with the CounterData already processed and in this way they are sent in a burst to Kafka in the shortest possible time. Sixteen partitions were always used in this scenario for
the topics in Kafka, the ETL input was 5000 files and the Calculator’s was 682500 CounterData, and in each VM 10 GB of RAM was reserved for the Spark worker.

Figure 4.1: ETL: Burst of 5000 files, 1 VM, 16 Partitions.

First, the ETL and Calculator were evaluated in the scenario where only two VMs were used. In this scenario three tests were done for the synchronous mode and for the asynchronous mode. What differentiated these tests was the amount of cores used: 1, 2 and 4 cores. The objective of this evaluation was to discover the number of counters and KPIs produced per second in only one VM. The results can be observed in Figures 4.1 and 4.2.

As can be verified, in the case of using only one core the result is that fewer counters and KPIs per second were produced than with two and four cores, that is, the more cores used the more counters and KPIs per second are produced, for both synchronous and asynchronous modes. This result was to be expected since there is parallel processing, i.e., in the case of ETL each core processes different files and thus process more files and consequently more counters. In the case of the Calculator each core processes a partition of a topic and thus parallelizing the processing of CounterData.

As can be observed in Figures 4.1 and 4.2, when the ETL and Calculator are operating in synchronous mode, the results are inferior to when they are in asynchronous mode. This difference originates from the waiting period caused by the “Wait for all Counters” component in the case of ETL and “Calculate KPI” and “Store Result” in the case of Calculator when CounterData and KpiResult are sent to Kafka. When the synchronous mode is used in Kafka to send objects, the ETL and Calculator are blocked until Kafka returns a response. This response is intended to ensure that the message is delivered and that it is not lost, corrupted or otherwise sent in duplicate. This implies that messages are sent
In Figures 4.3 and 4.4 where a VM was used in asynchronous mode, it can be verified that the processor, when all four cores have been used, almost depletes the processing power of the VM. The ETL, given that it uses the Cassandra for the incomplete CounterData, suffers from a performance penalty caused by the response time of queries to the database. When using only two cores, 50% of the CPU is used in average and, in the case of only one core being used, approximately 25% of the CPU is used in average. These values were to be expected since two cores are half the cores available in the CPU and, in the case of only one core, it is $1/4$ of the cores available.

In the case of the utilization of the synchronous mode, it is possible to observe in Figures 4.5 and 4.6
that the utilization of the processor is not as high since ETL and Calculator have the waiting time for Kafka responses when sending each message.

The case of the usage of RAM memory for the previous case, can be compared in Figures 4.7 and 4.8 for the ETL and for the synchronous and asynchronous modes respectively. When analyzing both graphs, it turns out that the ETL uses no more than 20% of the total memory of the VM.

In the case of Calculator, more RAM is used, as can be observed in Figures 4.9 and 4.10. The memory used by Calculator is approximately 30% of the total VM memory. This uses more memory than the ETL because it saves a considerable amount of CounterData to perform the aggregations.

As previously mentioned, when the asynchronous mode is used a much higher performance is achieved for it is much faster. This happens because instead of each message waiting for the delivery confirmation, a new thread is created to perform the deliveries that return immediately without any confirmation of delivery. For this reason it is called asynchronous. This new thread is created only once in each core and has as a task to send all the messages of that core. To be as efficient as it is possible,
the messages are placed in a queue and can be sent in a batch instead of being sent individually. That is, the performance gain is due to the fact that ETL and Calculator do not wait for each message to be sent individually with a delivery confirmation, it only sends to this new thread that works separately and returns immediately and so it can proceed to a new task. Besides not waiting for a response it is also more efficient because it sends several messages at once. This behavior can be verified in the graph of Figure 4.11 where it can be seen that in synchronous mode the transmission speed is almost constant because Calculator is always sending messages individually. Here it is still possible to verify that when the transmission speed is higher the utilization of the processor is smaller since there is a waiting time for the response of Kafka. In the case of the asynchronous mode, the graphic of Figure 4.12 allows to verify that there are times when there is no transmission and others when there is a peak of transmission, this means that several messages were sent at once in a few moments. At the beginning of the graphs it can be seen that there is a very sharp peak in data reception, this is the jar with the code sent by Spark Master to all workers / VMs, which in this case is just one.

As in the case of using only one VM as Spark worker, when using several, the results illustrated in
Figures 4.13 and 4.14, the synchronous mode has a much lower performance than the asynchronous. In this scenario five VMs are used where four are Spark workers. The distribution of the usage of cores in Spark is always the highest possible. In this case, when ETL or the Calculator is launched with a core, one of the cores of one of the VMs is chosen, with two cores a core is used in two distinct VMs, with four cores each VM uses a core since there are four VMs, with eight cores each VM uses two cores and in the case of sixteen cores all the VMs use all the cores.

In this last scenario, it can be seen that in synchronous mode the scalability of Calculator is much lower than that of ETL. This happens because Calculator sends more messages to Kafka (CounterData and KpiResult) than ETL, and this results in more execution time, that is, being blocked waiting for Kafka’s answers. In asynchronous mode, the growth between ETL and Calculator is very similar.

### 4.2.2 Evaluation of RAM utilization

In this scenario, the environment for this performance test is as follows:
• One VM for the Spark Master and four VMs for the Spark Worker with four cores each were used, totaling 16 cores;

• Ten Gigabyte (GB) of RAM was reserved for the Spark worker on each VM;

• Sixteen partitions were used for the topics in Kafka;

• Kafka was configured in asynchronous mode;

• Five thousand files were sent to ETL and 682500 CounterData to Calculator.

As shown in the graph of Figure 4.15 there is a significant impact in the usage of less than 3 GB of RAM in ETL and it can also be seen that up to 6 GB of RAM the performance increases. However, when using more than 6 GB there is no increase in performance because RAM memory is reserved
for Spark that is not used and the operating system remains with less and less memory to manage the hardware and other applications such as the NFS client. In Calculator, the impact is more significant with the usage of less than 4 GB RAM and it increases performance up to 6 GB and, as with ETL, no performance improvements are recorded when increasing to more than 6 GB of RAM. Thus, in the following tests 6 GB of RAM was used in ETL and Calculator.

### 4.2.3 Different input files in ETL evaluation

In this scenario, the environment for this performance test is as follows:

- One VM for the Spark Master and four VMs for the Spark Worker with four cores each were used, totaling 16 cores;
- Six GB of RAM was reserved for the Spark worker on each VM;
- Sixteen partitions were used for the topics in Kafka;
- Kafka was configured in asynchronous mode;
- ETL was fed with two different files, template 1 and template 2, as well as with a mixture of the two in a proportion of 50% each.

Different types of files can influence the performance of the system and to evaluate this aspect a test was performed where ETL was fed with different files (OMeS) with different characteristics. The result can be visualized in the graph of Figure 4.16. The two files used were template 1 and template 2 described above.
When analyzing the graph it can be verified that template 1 can obtain results that are much higher than template 2. This result originates from the fact that there is a higher density of counters per file. In the case of template 1, 851 lines are counters in a total of 975 lines, therefore this shows that this file has approximately 87.3% of counters per line. In the case of template 2, there are only 81 lines with counters in a total of 359 lines, in other words, it has approximately 22.6% of counters per line. This means that the parser extracts more useful information (counters) from template 1 than from template 2.

Another important fact is the amount of information contained in the files, i.e., to open a file and then to close that connection adds execution time, to read a file it is necessary to make a request to the disk, it then has to read the blocks on the hard drive that contains the information of the file and place it into volatile memory. Since template 1 has much more information than template 2, less time is spent for the same amount of information read in both files.

For a closer assessment to reality, a proportion of 50% of each of the templates was used in all tests. In this way, it can be verified that the dashed line with triangles that symbolizes the mixture of the two templates has an intermediate performance, since template 2 lowered the final performance compared to when using template 1 only. In all the performance tests, except this one, the two templates were used in a proportion of 50% each. With this graph it is also possible to verify that ETL has an almost constant performance and the results do not vary much in the several repetitions with the same characteristics and inputs.
4.2.4 Kafka partitions evaluation

The environment for this performance test is as follows:

- One VM for the Spark Master and four VMs for the Spark Worker with four cores each were used, totaling 16 cores;
- Six GB of RAM was reserved for the Spark worker on each VM;
- Between sixteen and 1024 partitions were used for the topics in Kafka;
- Kafka was configured in synchronous and asynchronous modes.

The Tests were performed only on Calculator to assess the impact of the number of partitions on the performance of RTPM. This decision was made due to the fact that only Calculator consumes Kafka messages (CounterData). ETL does not suffer a significant impact with the change in the amount of partitions in the topics because it only produces messages, i.e., the producer only sends the messages to Kafka regardless of how many partitions the topics may have. Calculator as a consumer behaves differently, because the number of partitions are distributed by the number of cores that are being executed by Spark.

If 16 partitions are defined for the topic `rtpm_counter_data` which is the topic where `CounterData` is produced and consumed, and in case Spark is working with 16 cores, in each Spark cycle each core performs a single task with all new messages. If in the same case as above the number of partitions is changed to 1024, this means that each of the 16 cores and in each Spark cycle \( \frac{1024}{16} = 64 \) average tasks per core are executed.
Sixteen partitions per topic were tested, which is the minimum number of partitions since the system used 16 cores, up to the amount of 1024 partitions per topic. The result is illustrated in Figures 4.17 and 4.18, where it can be seen that with the increase of partitions per topic the performance increases significantly up to 128 partitions, both having Kafka configured in synchronous or asynchronous mode. This means that with 128 partitions each core performed per cycle 8 tasks on average. The increase in performance is due to the fact that the tasks are divided so that when a faster core finishes a task or because it has more resources or because the task was simpler/faster, it prompts another new task.

If only 16 partitions are used what can be observed is that there are cores that finish their processing faster than others and wait for those which are still finishing their tasks, creating a waiting time to start a new cycle. On the other hand, if split too much, creating a very large amount of partitions, the system starts failing to get better performance at a certain point or may even worsen its performance.
degradation has to do with the excessive increase in the division of tasks, which creates more and more tasks and with it metadata associated with them and an increase in their management. Too much processing in the management of tasks begins to exist, more than in the useful processing (calculate KPIs).

With this test, it was also verified that Calculator has an almost constant functioning since the results with the same inputs always have very close values. Upon the analysis of this test it was decided to use a quantity of partitions eight times the number of cores used for the next tests, since it is the most balanced quantity in order to optimize the results.

### 4.2.5 Scalability evaluation in ETL and Calculator

![Figure 4.17: Kafka Partitions in Calculator: Burst of 682500 CounterData's, Asynchronous, 4 VMs, 16 Cores.](image)

<table>
<thead>
<tr>
<th>Execution Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th># of Partitions</th>
<th>Counters</th>
<th>KPS</th>
<th>SQM</th>
<th>KPIs</th>
<th>Counters</th>
<th>KPS</th>
<th>SQM</th>
<th>KPIs</th>
<th>Counters</th>
<th>KPS</th>
<th>SQM</th>
<th>KPIs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>108.073</td>
<td>103.932</td>
<td>106.662</td>
<td>99.817</td>
<td>102.845</td>
<td>108.540</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>114.176</td>
<td>109.125</td>
<td>108.828</td>
<td>112.454</td>
<td>114.872</td>
<td>114.368</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>107.018</td>
<td>107.731</td>
<td>108.662</td>
<td>108.281</td>
<td>106.048</td>
<td>112.742</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>102.373</td>
<td>101.943</td>
<td>102.096</td>
<td>105.456</td>
<td>100.748</td>
<td>105.506</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>79.701</td>
<td>78.906</td>
<td>84.977</td>
<td>84.512</td>
<td>84.387</td>
<td>82.344</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>75.393</td>
<td>69.690</td>
<td>69.977</td>
<td>69.740</td>
<td>74.161</td>
<td>74.871</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>59.328</td>
<td>60.573</td>
<td>60.735</td>
<td>58.218</td>
<td>58.644</td>
<td>58.684</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This last evaluation test aims to answer the second point of the objectives, which is to verify which size of the cluster and its characteristics required to reach the values defined by Nokia, written in Table 4.1.
In order to answer the previous question, ETL and Calculator were tested on clusters with various amounts of VMs always using all available cores, with 6 GB RAM per VM and with a number of partitions eight times the number of cores. The number of VMs used varied, having started with one and doubling the number for the next clusters until reaching a cluster with 16 VMs and 64 cores. The results were obtained with extrapolation from this cluster, given the lack of resources to test ETL and Calculator with a larger number of VMs. To find out the best function to model each extrapolation a regression analysis was used, and the analyzed functions were exponential, linear, logarithmic, polynomial and power.

The results from the previous scenario for ETL are illustrated in the graph of Figure 4.19 where the actual results for multiple clusters from one VM to sixteen VMs can be verified. First, only one VM was used, where 26177 counters/s and 14970 counters/s were obtained for asynchronous and synchronous mode respectively. Then the same test was done using two VMs where the values 44517 counters/s and 26174 counters/s were obtained for the asynchronous and synchronous modes respectively. In this way resources were doubled and as a result there was a performance increase of 170% and 175% for asynchronous and synchronous modes respectively.

Whenever the cluster was increased there was always an increase in performance compared to the previous cluster but with a slight degradation. Using polynomials of the second order, with the polynomial method least squares for extrapolation, two equations were obtained to model the results
with more than sixteen VMs for synchronous mode and another for asynchronous mode. The equation obtained to model the results of the ETL for the asynchronous mode was the equation \( y = 5315.5x^2 + 2533.9x + 18189 \). A polynomial function was chosen because, of all the functions, it had the Multiple R (correlation coefficient) and R squared (Coefficient of Determination) closer to 1, that is, with greater correlation and in this way we were able to model the results with the smallest possible error.

These results show that for a small network with a performance of 65387 counters/s, approximately 4 VMs are required with the characteristics used in the tests, with at least 6 GB of RAM reserved for Spark for each VM and the usage of 128 partitions for Kafka. For a performance of 268000 counters/s for a medium network, approximately 64 VMs are required with a total of 256 cores and 2048 partitions for Kafka, according to the extrapolation used. For a large network where 670000 counters/s have to be processed, it is needed approximately 1024 VMs with 4096 cores and 32768 partitions, also according to extrapolation. For synchronous mode the equation that best models the results is \( y = 233.51x^2 + 11951x + 2339.1 \). Given this equation in synchronous mode, we can only obtain performance for a small network with 16 VMs within a reasonable amount of resources.

In relation to Calculator, the results are shown in the graph of Figure 4.20 where it has the same format as the ETL results referred above. Analyzing the graph, it can be verified that to achieve the performance of 45771 KPIs/s for a small network it is necessary to have approximately 2 VMs, and for a performance of 187600KPI/s for a medium network it is enough to have approximately 16 VMs. The
Figure 4.20: Calculator: Burst of 682500 CounterData’s.

The equation $y = 5660.6x^2 + 8994x + 13538$ was used to model the asynchronous mode. In the case of a large network where a performance of 469,000 KPIs/s is required, approximately 256 VMs is required according to the extrapolation carried out.

In the synchronous mode, for Calculator to have a performance of 45771 KPIs/s for a small network it is necessary to have approximately 8 VMs. To perform the extrapolation of results in synchronous mode in Calculator, instead of using polynomials a power function was used. This because the method with polynomials did not model the expected results well, creating a decreasing curve. Thus, of the methods analyzed the one that modeled the results with less error was an extrapolation with the power regression model and from it the equation $y = 7094.4x^{0.7609}$ was obtained. For a medium and large network the resources required are an exceedingly large number of VMs.
5

Conclusion

Contents

5.1 Conclusions .............................................. 63
5.2 System Limitations and Future Work ..................... 63
5.1 Conclusions

With this project the conclusion can be reached that it is possible to increase the performance of current systems to ensure that operators can monitor their growing networks and offer the best user experience to their customers. To get the results intended by Nokia is necessary to use new tools, such as Kafka. However, it is necessary to know how to use and configure them so as to achieve the intended goal. In order to achieve the desired performance Kafka must be configured to use the asynchronous mode, because too many resources are required for the synchronous mode. Spark's scalability is not exactly linear, but it's close. However, it suffers from a degradation whenever the number of VMs is increased which indicates that at a certain point Spark stops obtaining better performance with the increase of the resources. The lack of a way of sharing synchronized information between different cores in Spark workers has resulted in the use of Cassandra, where another performance penalty is added when queries are made that need to be answered since they have to be done in synchronous mode. Another component that affects the performance of RTPM is the KPI Registry, because even though it has a buffer of the data per core, it adds a relevant execution time. The RTPM can be used for other similar tools, as long as they use the OMeS specification or if a new type of source is implemented in the RTPM. This means that this tool can be used for other scopes as long as the goal is to ingest data such as NE counters and the intended output is something like KPIs, that is, take the ingested data and perform logic with them, such as arithmetic operations so that a final value is obtained.

5.2 System Limitations and Future Work

Files with counters (OMeS) must be moved or copied in full to the folder that the ETL is monitoring (filesystem.data.provider). This means that after they are in this folder, they can not be altered. To improve performance, some improvements can be made to boost performance, one of them being a learning machine that can be implemented in the Network Topology module in RTPM, so that it could learn the network topology on its own without having to depend on an external system and thus be more dynamic and tolerable of network changes. Sometimes it is not necessary to have a KPI with all the aggregations of every single one of the equipments of a certain topological level for real-time monitoring. This means that if 80% of the equipment of the lower level of an NE is calculated, for example, it may be sufficient to have a good indicator of any anomaly.
Bibliography


