Model Driven Paediatric European Digital Repository

Call identifier: FP7-ICT-2011-9 - **Grant agreement no**: 600932

Thematic Priority: ICT - ICT-2011.5.2: Virtual Physiological Human

Deliverable 14.2

Alfa version Infrastructure Deployment Report

Due date of delivery: 28-02-2015

Actual submission date: 06-03-2015

Start of the project: 01-03-2013

Ending Date: 28-02-2017

Partner responsible for this deliverable: maat-G

Version: 1.0

Dissemination Level: Public

Document Classification

Title	Alfa version Infrastructure Deployment Report
Deliverable	14.2
Reporting Period	2
Authors	Sebastien Gaspard
Work Package	14
Security	PU
Nature	R
Keyword(s)	Alfa version Infrastructure Deployment Report

Document History

Name	Remark	Version	Date
Deliverable 14.2		0.1	18/02/2015
Deliverable 14.2		0.2	03/03/2015
Deliverable 14.2		0.3	04/03/2015
Deliverable 14.2		1.0	05/03/2015

List of Contributors

Name	Affiliation
Sebastien Gaspard	maat-G
Lucian Itu	UTVB
Jérome Revillard	maat-G
David Manset	maat-G

List of reviewers

Name	Affiliation
Eleni Zacharia	ATHENA
Harry Dimitropoulos	ATHENA
Bruno Dallapiccola	OPBG

Abbreviations

CRO	Contract Research Organization
DCV	Data Curation and Validation
DPS	Data Publication Suite
eCRF	Electronic Case Report Form
ICD	International Classification of Diseases
PCDR	Paediatric Cardiac Digital Repository
SNOMED	Systematized Nomenclature of Medicine
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SOKU	Service Oriented Knowledge Utility
VPH	Virtual Physiological Human

TABLE OF CONTENTS

1 Project summary

MD-Paedigree is a clinically-led VPH project that addresses both the first and the second actions of part B of Objective ICT-2011.5.2:

1. it enhances existing disease models stemming from former EC-funded research (Health-e-Child and Sim-e-Child) and from industry and academia, by developing robust and reusable multi-scale models for more predictive, individualised, effective and safer healthcare in several disease areas;

2. it builds on the eHealth platform already developed for Health-e-Child and Sim-e-Child to establish a worldwide advanced paediatric digital repository.

Integrating the point of care through state-of-the-art and fast response interfaces, MD-Paedigree services a broad range of off-the-shelf models and simulations to support physicians and clinical researchers in their daily work. MD-Paedigree vertically integrates data, information and knowledge of incoming patients, in participating hospitals from across Europe and the USA, and provides innovative tools to define new workflows of models towards personalised predictive medicine. Conceived of as a part of the "VPH Infostructure" described in the ARGOS, MD-Paedigree encompasses a set of services for storage, sharing, similarity search, outcome analysis, risk stratification, and personalised decision support in paediatrics within its innovative model-driven data and workflow-based digital repository. As a specific implementation of the VPH-Share project, MD-Paedigree fully interoperates with it. It has the ambition to be the dominant tool within its purview. MD-Paedigree integrates methodological approaches from the targeted specialties and consequently analyses biomedical data derived from a multiplicity of heterogeneous sources (from clinical, genetic and metagenomic analysis, to MRI and US image analytics, to haemodynamic, to real-time processing of musculoskeletal parameters and fibres biomechanical data, and others), as well as specialised biomechanical and imaging VPH simulation models.

2 Executive summary

As an update of D14.1, this document will reuse elements from this former report. In order to be self-consistent elements from D14.1 that have not changed will remain as is.

This document will present the current progress in installation of the MD-Paedigree Infostructure Solution. After placing the project in context, it will present the general view of the current installation. Following this overview, two sections will explain in detail the installation and needs in term of software and hardware, adding information about what is currently installed and recommendations about the target architecture. Then different global topologies for the logical solution will be presented together with the advantages and drawbacks of each choice. Finally, this document will deal with the degree by which the user requirements are satisfied and quickly explain how the Infostructure solution responds to these needs.

3 Introduction

MD-Paedigree validates and brings to maturity patient-specific computer-based predictive models of various paediatric diseases, thus increasing their potential acceptance in the clinical and biomedical research environment by making them readily available not only in the form of sustainable models and simulations, but also as newly-defined workflows for personalised predictive medicine at the point of care. These tools can be accessed and used through an innovative model-driven infostructure powered by an established digital repository solution able to integrate multimodal health data, entirely focused on paediatrics and conceived of as a specific implementation of the VPH-Share[2] project, planned to be fully interoperable with it and cooperating, through it, also with p-Medicine.

In MD-Paedigree, the VPH Infostructure is designed to accommodate the chosen paediatric clinical areas, starting from the considerable experience capitalized in the Health-e-Child and Sim-e-Child projects. The latter developed grid and cloud-based eHealth repositories, models and simulations for specific diseases, and, particularly building on top of current developments within OPBG (Ospedale Pediatrico Bambino Gesù), further eHealth tools for data management and distributed high-performance computing which aim at gradually transferring into clinical practice the most advanced modelling in paediatric cardiology to support more precise outcomes analysis of pathologies and develop optimal therapies.

MD-Paedigree aims at achieving high-level semantic interoperability, thus requiring standards enabling the clinical contents to be interpreted consistently across the different EHR regimes, while complete clinical interoperability between systems will require widespread and dependable access to maintained collections of coherent and quality-assured semantic resources, including models that provide clinical context, mapped to interoperability standards for EHR and PHR and biomedical data, linked to well specified terminology value sets, derived from high quality ontologies.

In order to achieve semantic support at this level, MD-Paedigree takes advantage of recent work achieved in other EC semantic health and Ontology Based Data Access (OBDA) related projects such as SemanticHealthNet and DebugIT. MD-Paedigree also intends to comply with terminological and data interchange standards currently being developed within epSOS, in particular for the Patient Summary. As for biological data, MD-Paedigree will rely on OBO Foundry resources and BioDBcore recommendations. In addition, it aims to relate research, publications, experiments, and data joining forces with other EC – open access related – projects like OpenAIRE, OpenAIREplus and OpenAIRE2020.

- integrate and share highly heterogeneous biomedical information, data and knowledge, using best practices from the biomedical semantic Web,
- develop holistic search strategies to seamlessly navigate through and manage the integrative model-driven infostructure and digital repository,
- jointly develop reusable, adaptable and composable multi-scale VPH workflow models,
- support evidence-based translational medicine at the point of care, and
- ultimately facilitate collaborations within the VPH community.

MD-Paedigree elaborates on a decade of developments initially pioneered in the European FP5 MammoGrid and FP6 Health-e-Child projects, which were then brought further in FP7 Sim-e-Child. More particularly, it leverages on the grid Gateway concept, allowing scientists to abstract from the complexity of underlying grids, clouds and other computing resources they need to use. Nowadays, Science Gateways represent an important emerging paradigm for providing integrated infrastructures. According to Wilkins, a Science Gateway is “a community-developed set of tools, applications, and data that are integrated via a portal or a suite of applications, usually in a graphical user interface, that is further customised to meet the needs of a specific community. Gateways enable entire communities of users associated with a common discipline to use national resources through a common interface that is configured for optimal use. Researchers can focus on their scientific goals and less on assembling the cyberinfrastructure they require. Gateways can also foster collaborations and the exchange of ideas among researchers”. MD-Paedigree thus intends to reuse the latest Service Oriented Architecture (SOA) based Gateway released in Sim-e-Child,

which enables secure and reliable access to abstracts from and integrates all forms of applications and data useful to users. The Gateway materializes as a layered architecture of standard secure (generic medical) services running on top of a grid infrastructure, which is physically installed at the participating clinical centres. Thanks to these on-site access points, users can transparently utilize a number of heterogeneous computing resources, ranging from local databases, to the distributed grid infrastructure regardless of their location and available connectivity. The Gateway supports the major principles of an SOA and exposes a significant set of biomedical utilities to date.

MD-Paedigree will extend the Gateway and demonstrate a reasonably well-scoped use-case of the Service Oriented Knowledge Utility (SOKU) vision, as published by the European Commission in the Future for European Grids: Grids and Service Oriented Knowledge Utilities report, to address the challenge of delivering personalised care to patients.

MD-Paedigree will implement the SOKU vision, to facilitate the design and development of innovative predictive models as reusable and adaptable workflows of data mining applications and turning the latter into clinically validated decision support tools, made available at the point of care. This is what is illustrated in the following figure, where reusable VPH models (in the centre) are incubated in the system with progressive semantic enrichment and model transformations in a cycle (the yellow spiral) witnessing the intervention of both automated database-guided learning and data integration and knowledge experts validation. As these models become more mature, they are then clinically validated by participating centres and concerned clinical researchers, and ultimately made available at the point of care thanks to the physical distribution and computational nature of the MD-Paedigree model-driven infostructure.

Taking its roots from a well-established distributed digital repository, the MD-Paedigree VPH Infostructure will thus hatch in a plethora of breakthrough decision support applications, as is illustrated with the petals of the SOKU flower (top-left of Figure below).

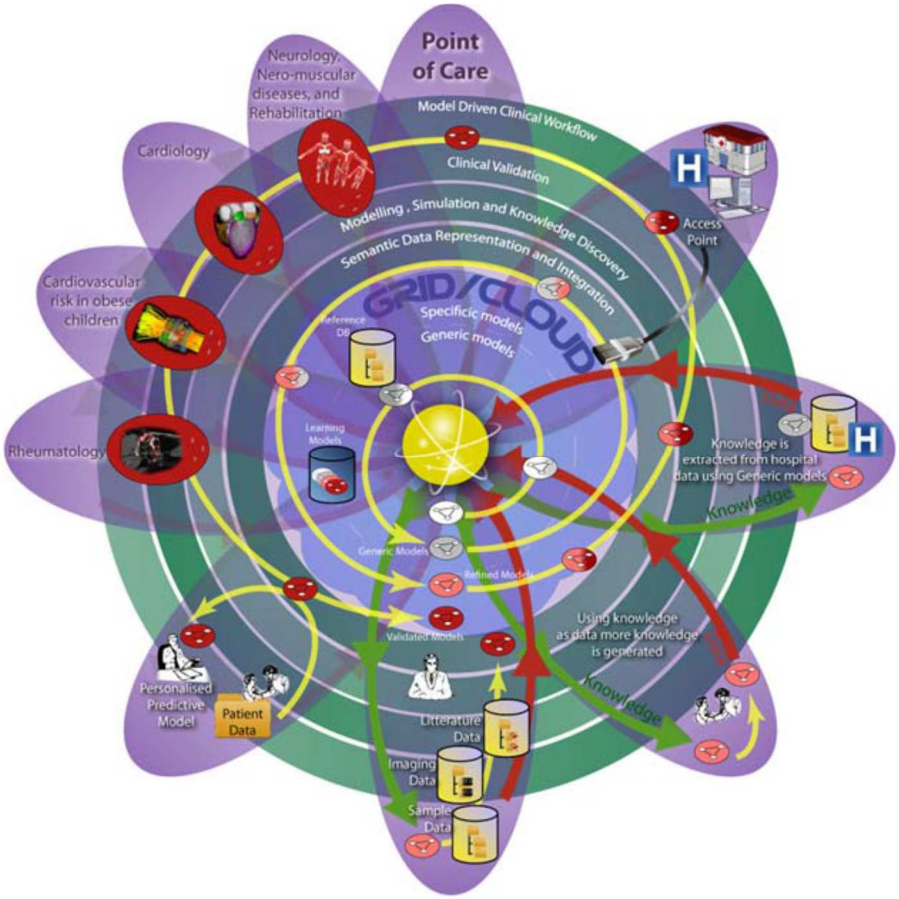


Figure 1: SOKU Flower

4 Inherited PCDR infrastructure as a CRO

Installed on top of the existing PCDR infrastructure, the current solution provides 2 Nodes having the ability to host science gateways, some central services and a web portal. The architecture is composed of hardware described in section 5 and of different software described in section 6 to 10. The infrastructure provides all the functionalities of connection and query on present data and some applications are already integrated.

The current system is already fully functional and able to acquire data as soon as protocols and data formats are defined.

4.1 Advantages

The CRO architecture for the whole project has a main advantage: it is already installed and functional. Added to this it provides a significant calculation capacity as described in section5.

The current power and configuration of the system can provide and run gateways for each institution in order to create cluster data as if it would be in different locations. Moreover, it will be easy to move virtualised gateways from OPBG to other sites if created.

4.2 Drawbacks

One of the main functionality of the concept of distributed gateways consists of hosting data at the place of the owner, allowing the system to work at least with local data if connection to other sites is lost. Centralised CRO breaks this and creates a single point of failure from outside OPBG.

Depending on the size of data to import, data transfer using importers can be huge. According to this statement, it is preferable to reduce as much as possible the network distance between data sources and data storage. It is preferable to have local gateways that first filter data for queries in order to only transfer the necessary data for a given query.

In PCDR, the storage has been calibrated for OPBG data only. The storing of information from so many different sources may quickly overload the current storage elements.

4.3 Conclusion

Such an installation can provide a complete solution for the beginning of the project providing the full functionality set and allowing quick development and testing. However, the system would be much more efficient if some other storage nodes were installed in some other institutions, particularly those institutions providing data as set out in **Errore. L'origine riferimento non è stata trovata.** of this Document.

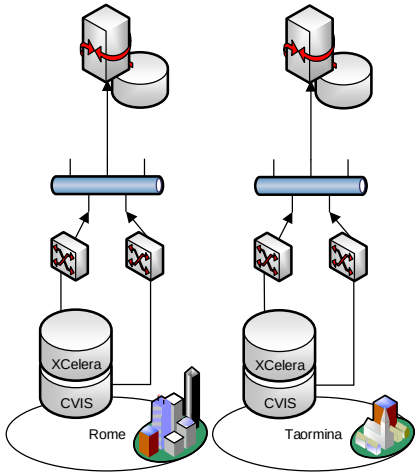
5 Hardware Architecture

5.1 Node Architecture (Rome and Taormina)

5.1.1 Current installed node connectivity schema

Current architecture is composed of:

- 1 Node at OPBG Rome
- 1 Node at CCPM Taormina
- 1 Portal
- 1 Central Server



The nodes are currently installed and connected together through a FastWeb secured connection. This allows both sites to share information with ease.

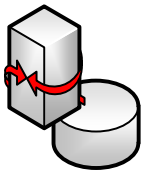
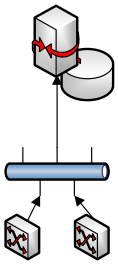
Due to important latency in administrative rights and choice and to the fact that no hardware has been funded within the project, no new gateways have currently been installed since D14.1. However, UCL and DHZB are supposed to install on site a gateway within the next few weeks.

5.1.2 Node composition

The current nodes are composed of different hardware as defined in the following table:

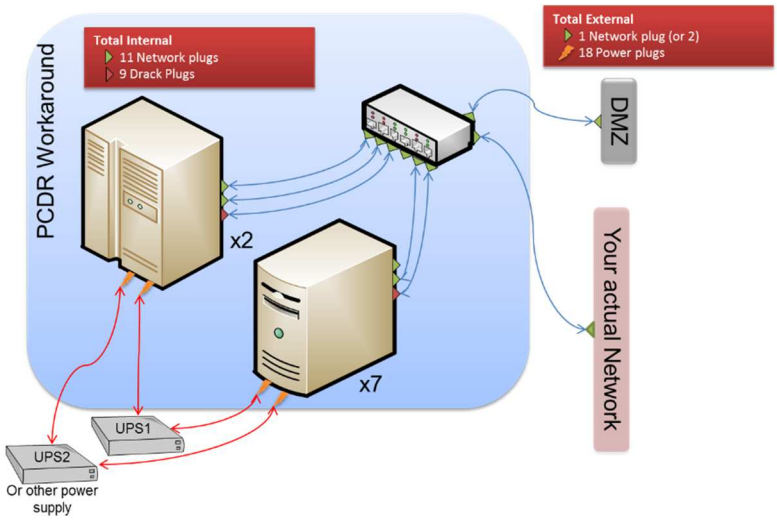
		Quantity	CPU	MEM	DISK
Service Container	PER515-12-16-6T	2	12	16	6
Computing Node	PER415-12-16-250	7	12	16	0,25
Data Dumper	Virtual Machine	1	4	4	0,25
Switch	PC5524	1	0	0	0
Total			112	148	14
			Cores	GB	TB

In the following, node's elements will be represented according to the key below:

		
Gateway	Importer	Node

5.1.3 Node physical installations

This configuration is installed as presented in the schema on the right.



The server installation is quite standard and simple managing redundant power supply, secured network connection for computer management, and dedicated external connection through a DMZ for the in the cloud virtual machines installed on the servers. Standard servers are deployed to host as many gateway as needed and 2 storage elements are managing the data storage. For the project's beginning PCDR storage elements should be sufficient to store the project data, but if a systematic storage of anonymised DICOM data require much more space, adding a heavy storage solution (SAN or NAS) should be considered.

5.2 Portal and Central Servers

The current installation is also based on a few central components that are needed for the correct behaviour of the whole system. These components are:

- AMGA central
- Portal
- Query dedicated node

5.2.1 AMGA central

AMGA Central is a server physically localised in Archamps in France that provides the synchronisation abilities of the tables that need it. It is necessary for the system to perform correctly, but, if stopped, does not block local updates of information nor queries.

5.2.2 Portal

The portal is the main access point of the system; currently it is physically installed at OVH at Compiègne. It provides the web front-end, and so if stopped, the main user interfaces would not be available, however the core system and web-services will remain functional. In the future framework this single point of failure may be removed by the installation of other portal servers at the main node sites.

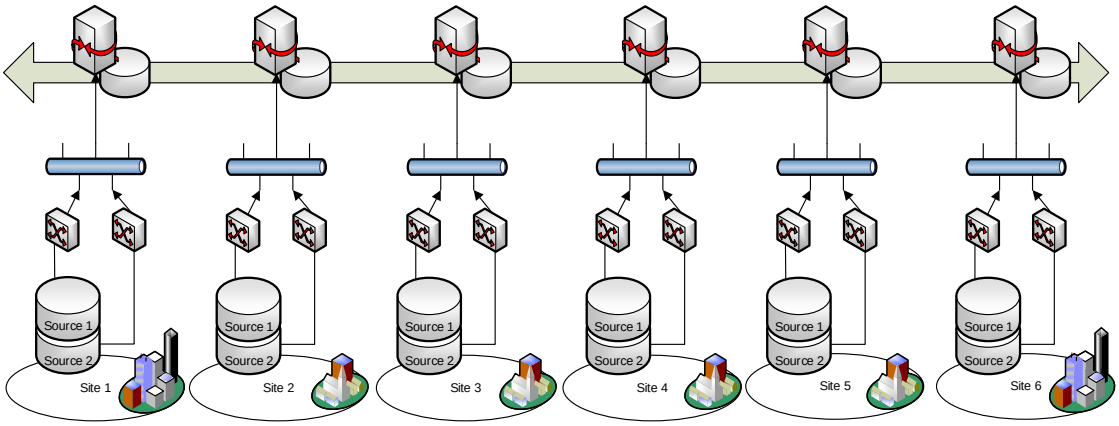
5.2.3 Query dedicated node

This node is a temporary node that is physically installed at Rome. This node provides database access to the portal runs the distributed query on the global system. In future installations, this node should be replaced by cloud-based calculation nodes that will be automatically instantiated on demand.

5.3 Target hardware deployment

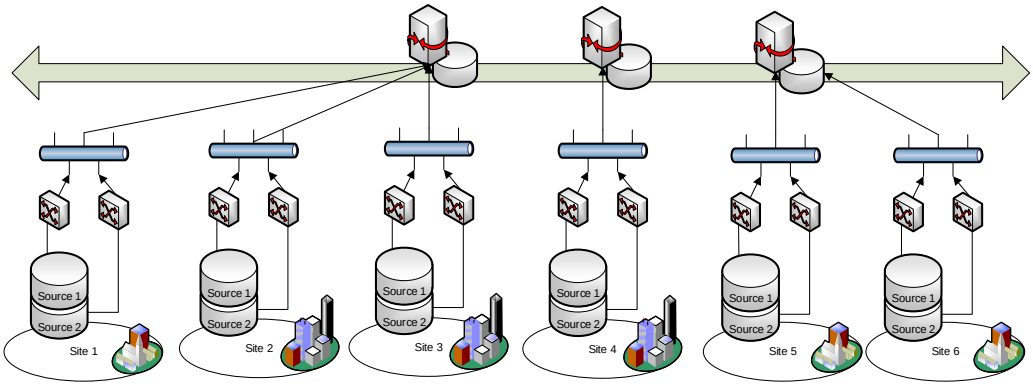
5.3.1 Optimal implementation

The current system is functional as is but is not yet optimal. To be optimal, each information provider should have their own node onsite. This would mean that all importers could run locally and store data locally. Most of the traffic would be internal and all the data would stay physically on site when not queried. This would also permit any centre to have fine-grained control of its own data particularly in terms of access rights. Also, it is important to note that by choosing this implementation, with each node being a source provider and the data being physically on site, isolating data of a particular site is as simple as unplugging a wire or shutting down a server.



5.3.2 Probable implementation

While the optimal technical implementation is as stated above, a compromise may be made in order to meet budget and time constraints. In this implementation, only few nodes would store data and provide it to the whole system. These node would be managing groups and rights and physically store the data, while the others would just have importers (that in this case are exporters from their point of view). The drawback of this implementation is that some institutions would host other institution's data and in case of a hardware problem the data provider would not have access to their data anymore. Also, in a case of disagreement between the host and the hosted, the hosted won't have any way to erase its data from the system.



In this example,

- Site 3 host data of site 1 and 2
- Site 4 is a standard node
- Site 5 is hosting the data of site 6

For the first step, OPBG have proposed to act as a CRO (Contract Research organization) providing resources.

5.4 Load capacity

5.4.1 Storage

Storage Load capacity has been evaluated and PCDR system have the capacity to store it.

CMP

Patient average storage size : 6 GB

Number of patient expected : 180

Total expected size : 1 TB

CVD

Patient average storage size : 2GB

Number of patient expected : 180

Total expected size : 0.54 TB

JIA

Patient average storage size :4 GB

Number of patient expected : 300

Total expected size : 1.2 TB

NND

Patient average storage size :9 GB

Number of patient expected : 290

Total expected size : 2.6 TB

Total Estimated Size : 5.34 TB

PCDR has a storage capacity of 6 x 1 Terabyte by storage machine that makes 4TB with backup (5 drives RAID 5 and 1 for automatic recovery). 2 storage servers are installed in Rome and 2 in Taormina for a total of 16TB capacity. This largely covers the need for the MD-Paedigree project. However, some other considerations have to be taken into account (see 5.4.3)

5.4.2 Power

The PCDR project has provided 12 computational servers to MD-Paedigree. For optimal functioning, each gateway from the system has to be hosted by one physical machine. Some computational server for query execution and tests environment are also required. For network speed and system administrative reasons, OPBG Rome node has been loaded first and some gateways are created on OPBG CCPM Taormina's node. The current system is powered enough for the standard computational needs (everyday work, queries, importation). Nevertheless, for big challenges external cloud resources may be needed, and for network speed issues we may need to consider installing some resource on different locations at acquisition center.

5.4.3 Other considerations

Legal and ethics considerations imply that some centers are required to host their data by themselves physically on-site. The MD-Paedigree infostructure team works on the installation of the gateway at these places.

6 FedEHR: Storage infrastructure

FedEHR is a Pandora service that provides storage abilities for medical events and patient information. As a Pandora service, all the interfaces of Pandora are exposed as web services consumable by other applications, and user interfaces are integrated in the Portal.

6.1 Patient centric data structure

In accordance with to the latest data modelling concepts in the literature, FedEHR proposes a storage model that is fully centred on the patient. All the data that is stored in the system is organised around a data structure representing a patient model. The current description of FedEHR architecture provides an evolutionary structure of data starting from the patient. Currently the data structure is oriented around medical concepts of medical events and clinical variable. These abstract models can be refined and specialised using metadata definitions created from physicians' descriptions of diseases and exams.

FedEHR is a highly secure patient-centric and vendor-neutral EHR (Electronic Health Records) repository. Leveraging on the cloud, FedEHR provides a distributed database for heterogeneous medical information integration from different geographical locations. It gives a unique and integrated view of data and offers a variety of tools to navigate and analyse data.

FedEHR is composed of 3 modules: Cloud, Big data, Analytics

6.2 FedEHR Cloud services

FedEHR Cloud establishes cross-enterprise security and common virtualized environments for computing resources management. Such innovative exoskeleton information systems allow healthcare professionals to rapidly, securely and anonymously share medical information across the healthcare enterprise, while setting a robust ground for demanding applications to develop limitlessly.
The following paragraph presents modules proposed by the solution, the Cloud mechanisms are detailed in paragraph 0

MD-Paedigree Cloud Integration.

6.2.1 Desktop Fusion and Terminal Service

The Desktop Fusion and Terminal Service makes it possible to expose and interact with advanced applications in the cloud. This expert access is destined to users in need of powerful functions, from advanced GUIs to UNIX-like command line interfaces.

6.2.2 Pandora Amga Node Configuration

The Pandora Gateway is using a distributed architecture where the whole Information System is kept in interconnected AMGA databases. This Information System contains the gateway information which allows them to inter-operate.

6.2.3 Pandora Pipeline Service

The Pipeline Service is a Web Service responsible for executing tasks inside the Pandora Gateway.

6.2.4 Pandora Gateway Saga Service

The Saga Service is a Web Service providing an abstraction layer allowing interacting with various grid or even non-grid middleware or infrastructure.

6.2.5 Pandora tomcat6 configuration

Pandora Gateway specific tomcat configuration files.

6.2.6 Pandora Gateway utilities:

- Pandora management utilities: data management-related Java archives
- Pandora misc utilities: miscellaneous utilities
- Pandora misc jSaga utilities: Saga-related jars containing jSaga Saga implementation and various adapters

6.2.7 Persistency Service

The Persistency Service is a backend service that indexes the datasets (images and clinical study data) in the Analysis Base.

6.3 Big Data Federation Service

To design, instantiate and manage patient-centric vendor-neutral distributed big data silos.

The Big Data Federation Service allows to setup silos of medical sensitive data in the private cloud network and to federate them into a single database. Aggregated data can then be queried, filtered, processed securely and irrespectively of its geographical location and complexity. At the very heart of FedEHR, Big Data makes it possible for healthcare professionals to access massive amounts of heterogeneous medical data, to analyse trends, patterns, simulate and test treatments, or even advise on similar cases and associated outcomes found in the network of connected electronic health records silos.

6.3.1 OPBG Importer

Some importers are written in java and are used in the export of data from cardiovascular routine systems of OPBG (Rome) and CCPM (Taormina) to provide numerical metadata information about echocardiography and textual information about examination conclusions.

The importers are:

- taking data from the routine system,
- normalising the data (using a hospital specific structure), and

- pseudonymising it (using 3 part storage).

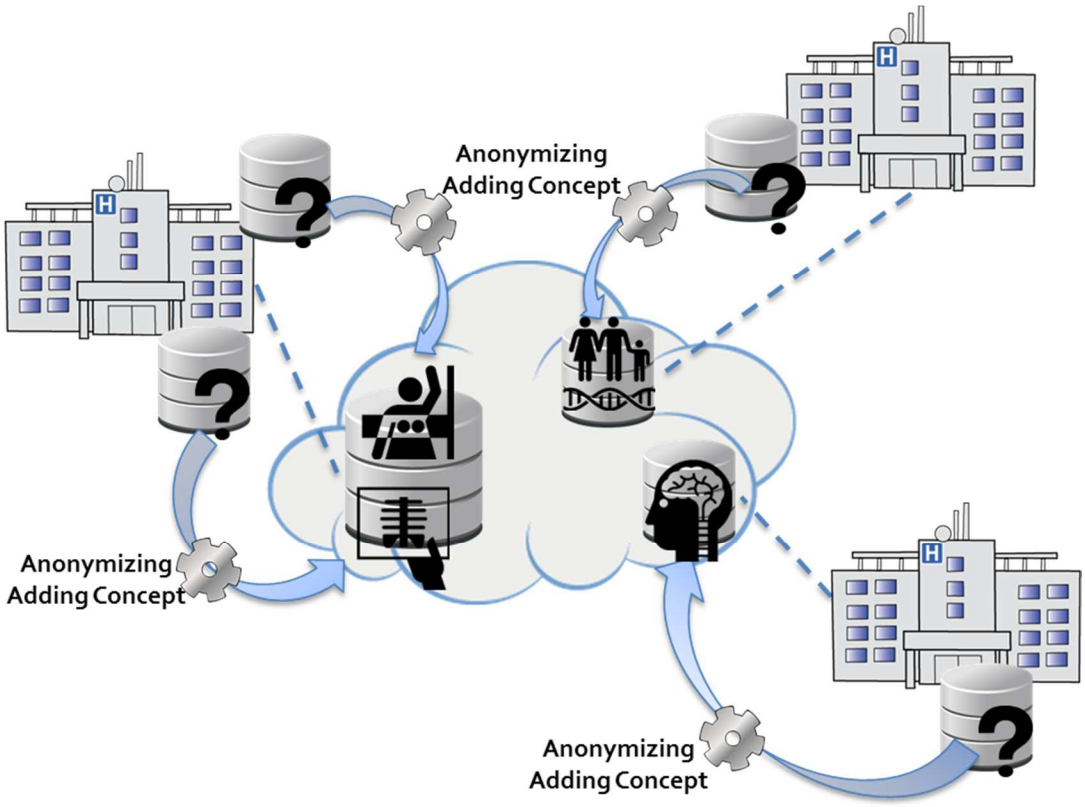


Figure 2: Importers

In the future some ontological information such as ICD or SNOMED will be added. This is in order to provide a uniform concept-based query capacity, keeping the ability to have a personal easy understandable data structure at each node.

6.3.2 DPS generic Importer

USFD and MAAT partners have developed a generic importer based on The Data Publication Suite (DPS) connection abilities. A pivot exporting XML format that can manage both data structure, semantic annotations and data values has been defined. Depending of the needs, anonymisation can be processed by the DPS transport, done by the java complementary importer or using a hybrid plugin called by the DPS.

This generic connector allows to reuse the DPS software developed for VPH-Share for connection to a hospital routine system. The DPS graphical interface is used to model data structure as a tree and some annotations are added to indicate to the FedEHR repository which element of FedEHR repository of the XML corresponds to a patient and which one corresponds to a medical event. Once defined the connector is configured to run automatically periodically to enrich the repository.

```
<?xml version="1.0" encoding="utf-8"?>
<DataInstance xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Tables xmlns="http://vph-share.eu/dms/">
    <Table>
      <Name>Walks</Name>
      <D2rName>Walks</D2rName>
      <Fields>
        <Field>
          <Name>ankle_angle_r_z</Name>
          <D2rName>ankle_angle_r_z</D2rName>
          <Size>13156</Size>
          <Type>string</Type>
        </Field>
        <Field>
        ...
        </Field>
      </Fields>
      <Key>WalkID</Key>
    </Table>
    <Table>
     ...
    </Table>
  </Tables>
  <Name xmlns="http://vph-share.eu/dms/">mdptest</Name>
  <Guid xmlns="http://vph-share.eu/dms/">2a4ba63b-1c9c-4e9e-b4d4-a338415757f9</Guid>
  <PublishAddress xmlns="http://vph-
share.eu/dms/">vphsharedatatest.sheffield.ac.uk</PublishAddress>
  <Prefixes xmlns="http://vph-share.eu/dms/" />
  <Relationships xmlns="http://vph-share.eu/dms/">
    <Relationship PrimarySourceName="Protocol name" PrimaryTableName="Walks"
PrimaryFieldName="WalkID" SecondarySourceName="Protocol name" SecondaryTableName="Signals"
SecondaryFieldName="WalkID" Type="OneToMany" />
    <Relationship PrimarySourceName="Protocol name" PrimaryTableName="Demographics"
PrimaryFieldName="PatientID" SecondarySourceName="Protocol name" SecondaryTableName="Walks"
SecondaryFieldName="PatientID" Type="OneToMany" />
    <Relationship PrimarySourceName="Protocol name" PrimaryTableName="Signals"
PrimaryFieldName="SignalID" SecondarySourceName="Protocol name" SecondaryTableName="Parameters"
SecondaryFieldName="SignalID" Type="OneToMany" />
  </Relationships>
...
...
...
```

Additionally to the DPS, a generic importer reading the pivot XML has been developed. This java importer analyses the pivot XML to define if the type of the data declared already exists or not. Then it creates new types when needed and pushes data into the repository. The java importer is directly invoked by the DPS periodical execution making the process fully invisible for the final physician user.

6.3.3 Dicom Importer from exported pre-anonymised data

6.3.3.1 Context

In order to accelerate modelling tasks and permit to modellers to start working before the infostructure is able to store data, a File Sharing system has been used. This data represents a significant amount of data and users do not want to repeat work again to export the same data into the final repository. So in order not to add extra work to physicians, it has been chosen to develop an importer for data in the repository. According to the fact the data in the repository is constituted at 99% of DICOMs, a special importer has been developed to push all this data into the repository.

6.3.3.2 Challenge

According to the law of most countries, the data shared in the file sharing system has to be first anonymised. Unfortunately, instead

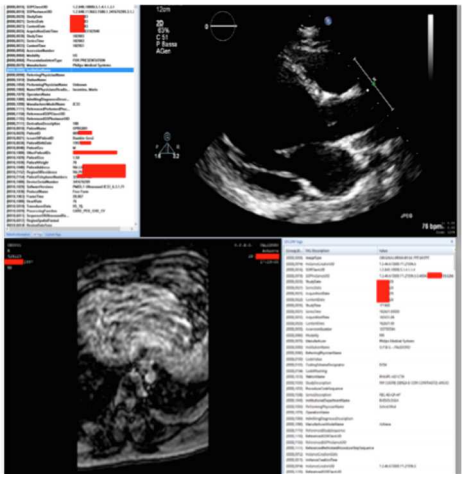


Figure 3 - Example of images that have been anonymised by PACS system. Red boxes hide data that are not anonymised enough

of simplifying the work, it considerably increased the complexity of the process. Indeed, anonymising data has two side effects:

- As far as anonymisation is done by different tools (the ones already existing at hospitals), it is impossible to know whic patient is concerned, and then it is difficult to reunify patients when having different exams.
- When patients are anonymised, the names and dates are altered, and it is difficult to find possible derived data from not standardised elements.

So the challenge was to import data with a good anonymisation level and the ability to enrich patient information with future exams.

6.3.3.3 Implementation

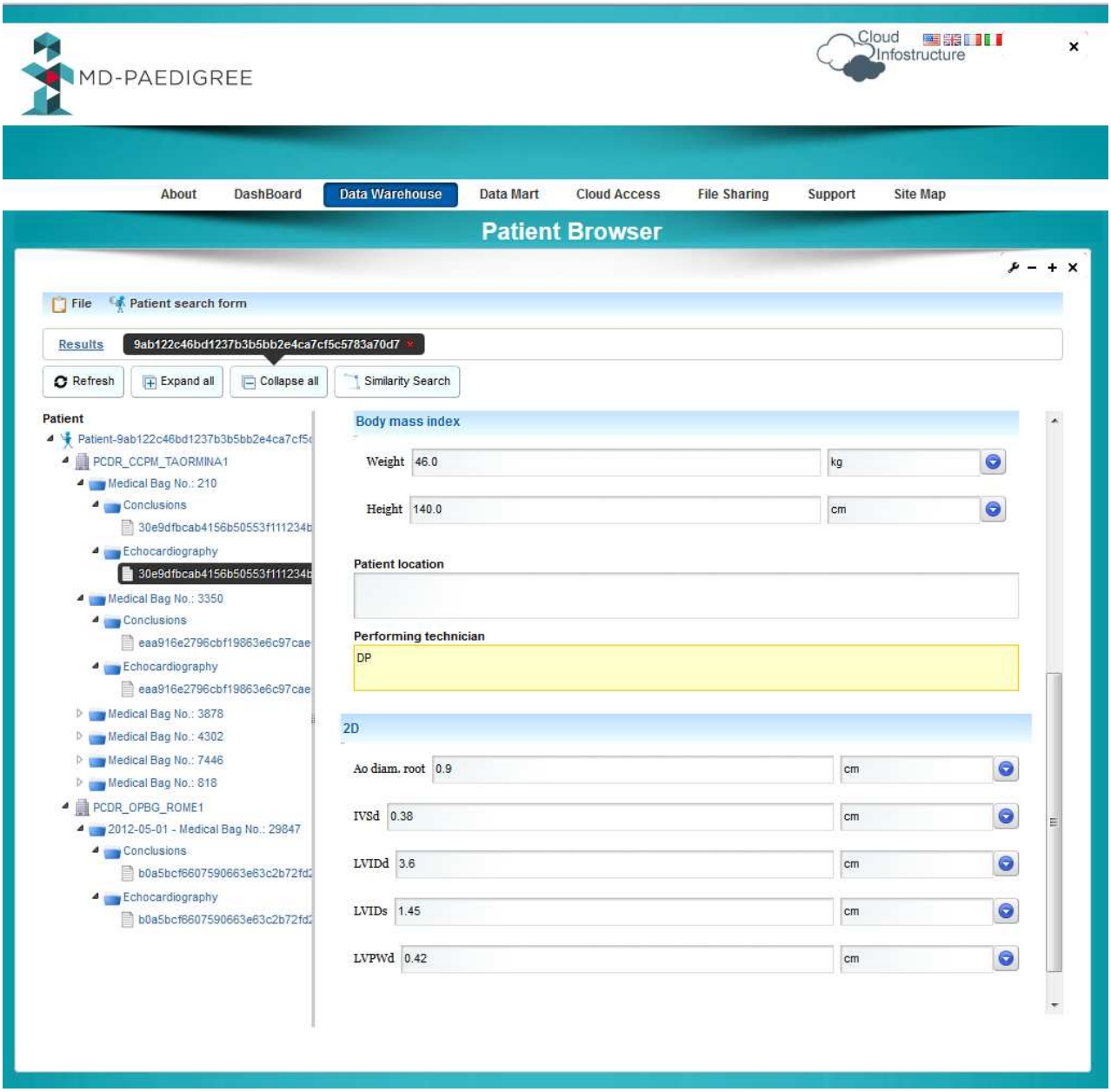
Even if anonymised by PACS systems depending on the configuration, the awareness of the technician that exports and the different needs of anonymity of each country, anonymised data in the file sharing system are not necessarily at a sufficient level of anonymisation for European constraints.

In accord with physicians and validated by the ethics comity, an arbitrary chosen ID has been defined for all MD-Paedigree patients. In the file sharing system, patient information has been arranged in such a way that the folder is named by this ID. From this folder a java importer executes a strong arbitrary anonymisation process removing all fields that are not defined in a whitelist and forcing the Patient ID in DICOMS to the chosen ID. Added to this, all importer writer partners will ensure that in the future the routine patient IDs will always be translated in the anonymisation process to this ID.

This protocol ensures at the same time a high level of anonymity for patient data and the perennial ability to enrich patients with future data.

6.3.4 Patient and Cohort Browser

The Patient Cohort Browser allows navigating through the data in an integrated and harmonized manner. The data is structured around the patient and patients are grouped into cohorts of interest. The Patient Browser as its names indicates, is a portal integrated feature that allows physicians to access full information about a patient from any the nodes of the system. It provides a complete medical history of the patient regardless of the physical location of data.



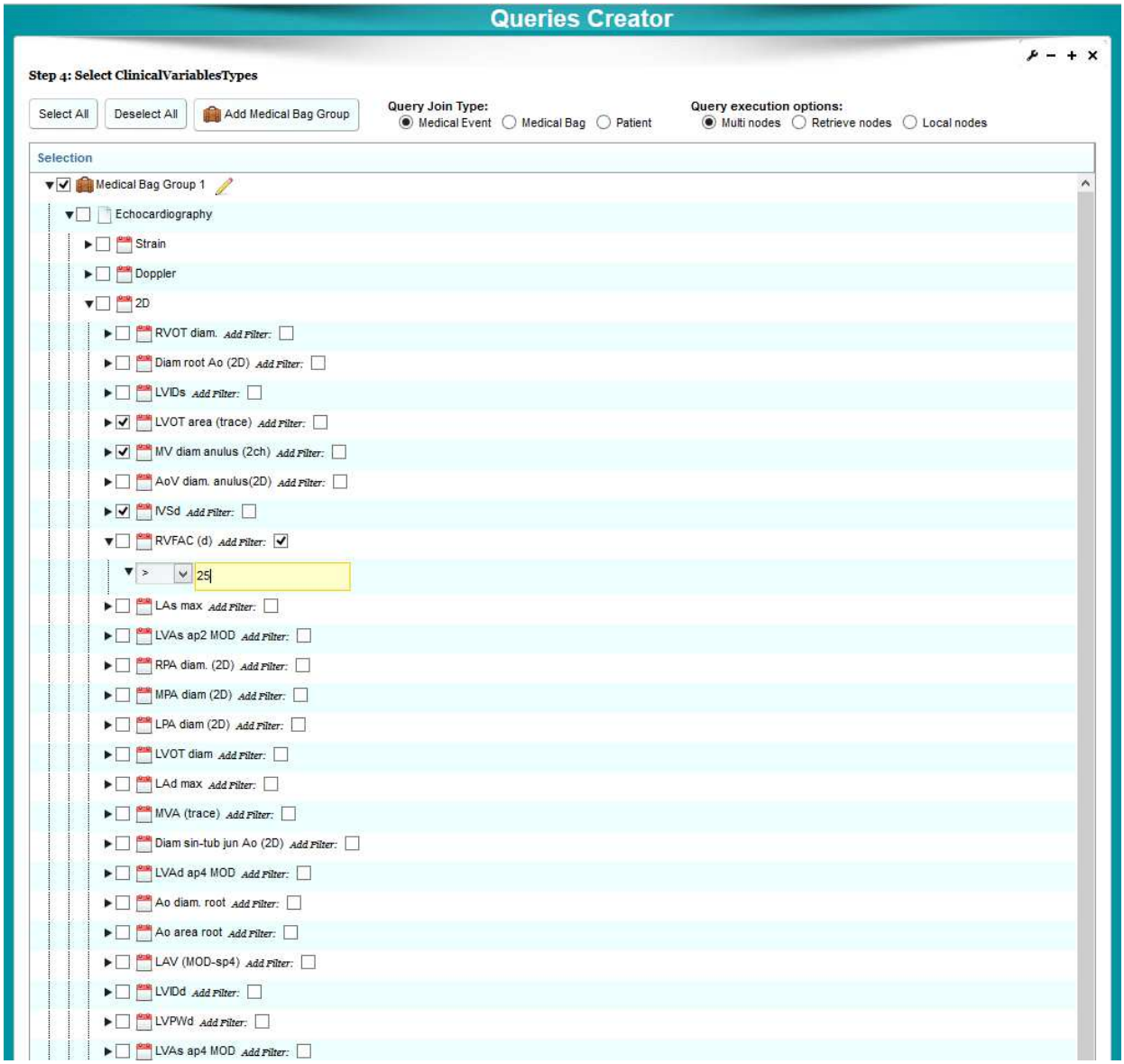
In the alpha version the patient browser has been updated to add DICOM and file specific restitution with integrated viewer for DICOM and ability to download DICOM images. Improvements have also been added, giving the ability to model with multiple instances of elements to be correctly rendered.

6.4 FedEHR Analytics

At the top of FedEHR-powered e-infrastructures, Analytical tools can be enacted to provide scientific insights onto large volumes of medical data, as well as advanced visualisation techniques to revolutionize our understanding of complex pathophysiological phenomena.

6.4.1 Query System

Stored data is not useful without a query system. FedEHR provides an inter-site query system presented as an SQL query for end users. These queries are managed by a query management system which generates a resultset that can be downloaded in a variety of formats and a graph visualisation tool.



Alpha version adds some functionalities for generating queries. New options propose different choices in query execution.

- "Query Join Type" : the choice is given between Medical Event, Medical Bag and Patient
  - Medical Event will provide the query that will look for all restrictions in the same Medical Event (same exam)
  - Medical Bag provide the query that will look for all restrictions in the same Medical Bag (same visit or follow up depending of the modelling choices)

- Patient provides the query that will look for all restrictions in the same Patient (during his entire life with all the stored exams)

- "Query execution options" : the choice is given between Multi nodes, Retrieve nodes, Local node
  - Multi nodes will generate a fast executing query that will query all nodes one by one to get data corresponding to the chosen restrictions in each node.
  - Retrieve nodes will generate a query that will be slow but will arrange the system to join information between all the nodes. This allows to query information at patient level even if the patient is followed up in different hospitals of the network.
  - Local node will execute the query on the selected node, depending of the gateway chosen by the user at configuration time.

The last version of query tool also provides the ability to run an R script at different levels. R scripts are possible to run on each site result and/or on the aggregated result set of all nodes.

6.4.2 FedEHR Similarity Search

The similarity search algorithm and patient cluster rendering based on query results are provided by the Portal. This mechanism enables physicians to travel through a similarity network graph and find clusters of similar patients. They can then retrieve all the anonymised information of any of the displayed patients. In this way they can identify interesting cases through the Patient Browser.

This first implementation is only based on pre-calculated queries and needs some more work to permit end users to use it by defining on demand criteria of similarities.

According to the unplanned amount of work represented by the data collection process, it has been decided to abandon this interface for the current reporting period and the following. It will be updated in the future if the allowed budget allows it.

6.5 Packages Version

6.5.1 Gateways

pandora-gateway-desktopfusion-management 2.1.0

pandora-gateway-desktopfusion-splash-translational-medecine 1.1-1

pandora-gateway-gateone-management 2.1.0

pandora-gateway-idal-amga-node-configuration 2.1.0

pandora-gateway-idal-fedehr 2.2.0

pandora-gateway-sal-desktopfusion 2.1.1

pandora-gateway-sal-gateone 2.1.1

pandora-gateway-sal-pipeline 2.1.0

pandora-gateway-sal-saga 2.1.0

pandora-gateway-sl-core 2.1.3

pandora-gateway-sl-utils-management 2.1.6

pandora-gateway-sl-utils-misc 2.1.12

pandora-gateway-sl-utils-misc-jsaga 2.2.1

pandora-wbar 1.1-1

6.5.2 Portal

data-management-portlet 1.6.0

desktopfusion-portlet 1.6.0

fedehr-chart-portlet 1.6.0

fedehr-repository-supervision-portlet 1.6.0

fedehr-similarity-search-portlet 0.0.1

google-earth-portlet 1.6.0

pandora-ext 1.0

pandora-global-lib 1.6.0

pandora-services-portlet 1.1

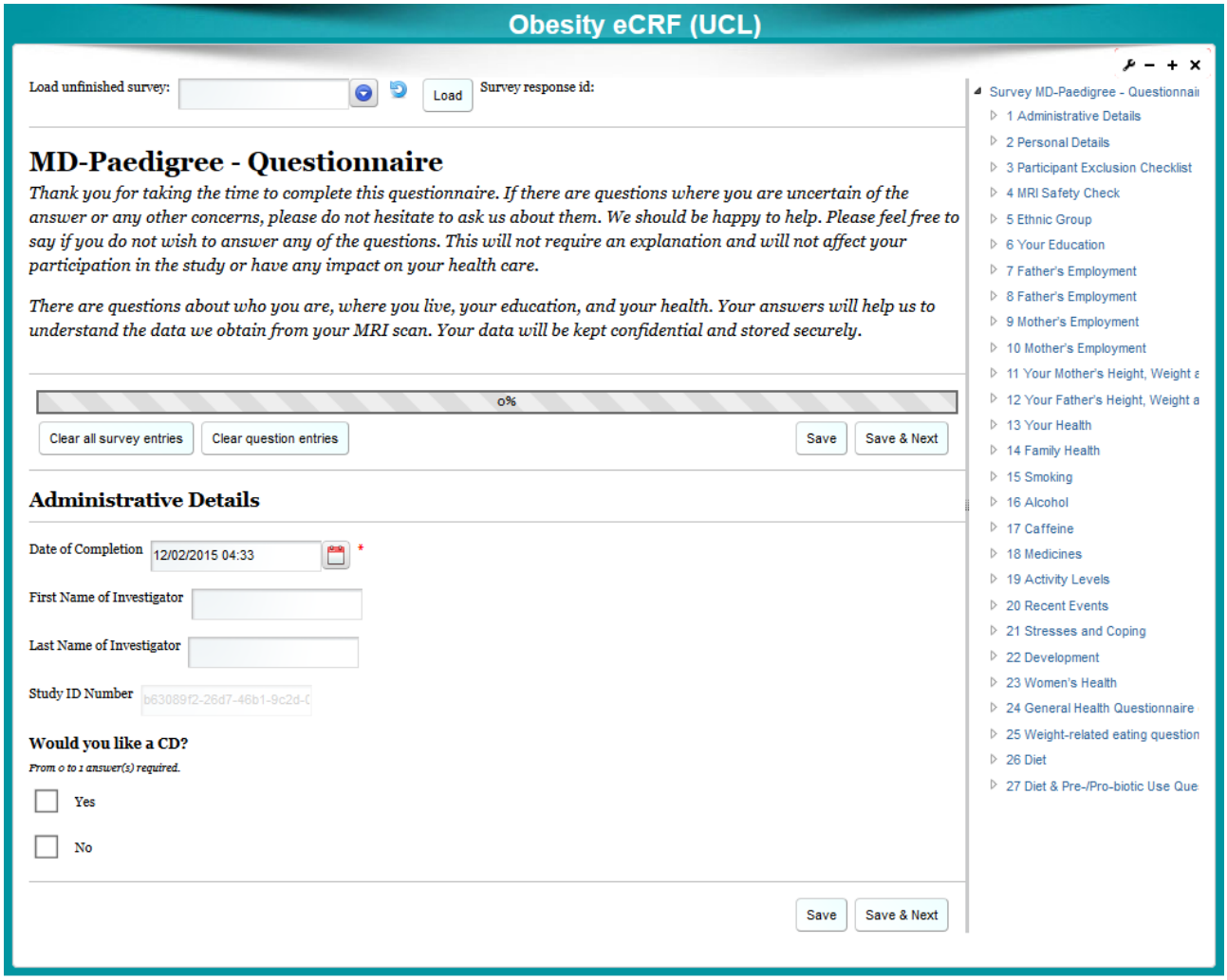
patient-browser-portlet 1.6.0

terminal-portlet 1.6.0

7 eCRF

During the project user specifications, a need of a new acquisition tool for cases has been registered. According to the need and in order to simplify the process of importation of this new data, gnubila decided to develop and provide to the project a tool called eCRF (electronic Clinical report form). This application consists of a configurable survey exposed through a web interface hosted by a server installed in each data

acquiring centre. This tool has been conceived and developed with the assistance of UCL ,following a semi-agile process whilst the end-users have been consulted at each step of definition. This development has achieved the implementation of one survey of 120 pages for CMP domain adapted to Italy and UK.



8 DCV: Data Curation and Validation

The new web-based DCV tool uses a client-server architecture. The following figure shows its architecture.

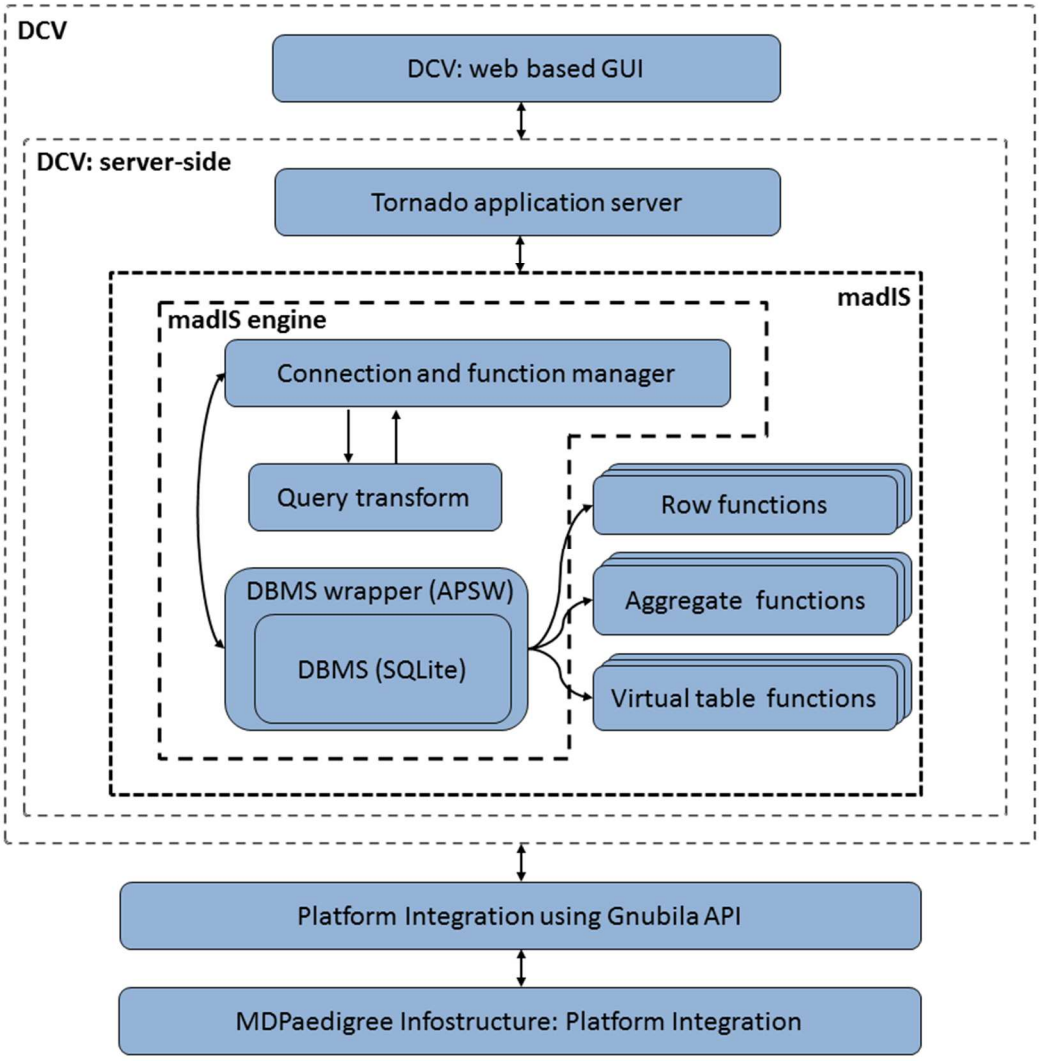


Figure 4: Architecture of the new DCV tool. It is connected with the MD-Paedigree infrostruction through Gnubila's API

For more information about DCV's implementation please consult deliverable "D15.2 - DCV curation tools and services to automatically and manually acquire high-quality curated data"

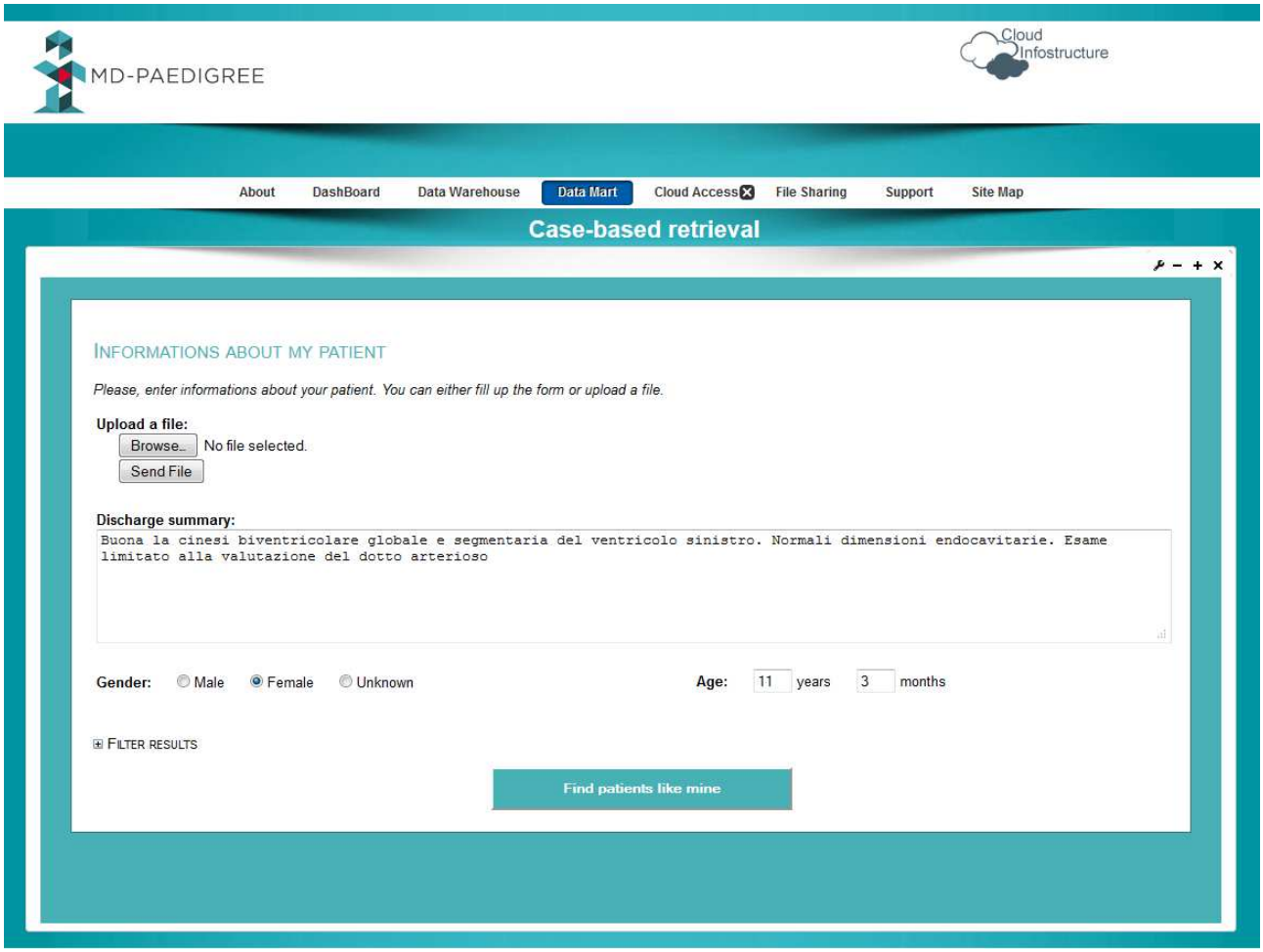
9 AITON: Statistical Models

In the current implementation, AITON is provided as a standalone application that is able to get information from a FedEHR query and provide statistical models in the form of graphical probabilistic models dealing with correlation finding and criteria impacts.

At this time, the integration is based on file transfer and program installation. As with the previous tool, the full integration process should provide a direct streamed data acquisition. The infostructure may also provide a cloud access to the application thus avoiding a local installation.

10 Case-based retrieval

Case Base retrieval is currently lightly integrated to the Portal, and it is able to run on some extract from FedEHR queries containing clinical narratives. Using onto-terminologies, it is able to classify the cases and find similar cases in the database providing back the ID of the patients corresponding to the criteria. A dynamic functional link back to Patient browser is provided.



Currently the case-based retrieval is integrated as an IFrame in the portal, but a better integration is needed. To do so, the portal will have to integrate the case-base server and its dependent services.

A Web service call to the case based retrieval is implemented and functional but the new fully integrated interface has not been developed yet.

11 GPU based processing and computation

Graphics Processing Units (GPUs) are dedicated processors, designed originally as graphic accelerators. Since CUDA (Compute Unified Device Architecture) was introduced in 2006 by NVIDIA as a graphic application programming interface (API), the GPU has been used increasingly in various areas of scientific computations due to its superior parallel performance and energy efficiency [Kirk et al., 2010].

The GPU is viewed as a compute device which is able to run a very high number of threads in parallel inside a kernel (a function, written in C language, which is executed on the GPU and launched by the CPU). The threads of a kernel are organized at three levels: blocks of threads are organized in a three dimensional (3D) grid at the top level, threads are organized in 3D blocks at the middle level, and, at the lowest levels, threads are grouped into warps (groups of 32 threads formed by linearizing the 3D block structure along the x, y and z axes respectively).

11.1 Optimized computation of stencil based algorithms

11.1.1 Introduction

Stencil computation is a computational pattern on an *n*-dimensional grid, whereas each location is updated iteratively as a function of its neighbouring locations. This pattern is found in several application domains, like image processing, computational fluid dynamics, weather prediction, etc.. Previous studies have shown that, if regular Cartesian grids are used, GPU based implementations are able to significantly speed up the execution compared to regular CPU based implementations [Phillips et al., 2010], [Shimokawabe et al., 2011].

The goal of the herein reported work was to evaluate the performance of 3D stencil based algorithms on a series of recent GPUs. Previous research activities have focused on single precision computations. To meet the high accuracy requirements, inherent for scientific computations [Nita et al., 2013], [Zaspel et al., 2013], we focus on double precision computations.

11.1.2 Methods and implementation

For studying 3D stencil based algorithms implemented on graphics processing units, we consider the 3D unsteady heat conduction problem which is modeled as a second order partial differential equation describing the distribution of heat over time over a given 3D space. For the numerical solution we apply a finite difference method on a 3D grid of points. The solution scheme is fully explicit: the computation of the new value at any grid point is fully independent from the computations at the other grid points.

In the following we introduce two baseline GPU-based implementations of the unsteady heat diffusion problem. For the first baseline implementation (called in the following *3DBase*), each grid point is handled by a separate thread. Two buffers are allocated, one for the values at the previous time step and one for the values at the new time step. To eliminate the memory copy requirement from one buffer to the other, the buffers are swapped at the end of each time step. To compute the new value at a grid point each thread performs seven global memory read operations at each time step. Since global memory operations are very slow, this represents a severe limitation of the kernel performance.

To allow for a better memory usage, we also consider a more efficient approach, whereas threads and thread-blocks are organized into 2D structures. The computational grid is divided into x-y planes and the subdomains are assigned to separate thread blocks. Each 2-D slice is represented through the grid points in the *x* and *y* directions, providing for the threads the (*i*,*j*) indices of the grid points. A loop is then used to traverse the grid in the z-direction and obtain the final *k* coordinate (this kernel version is called in the following *2DBase*). Unlike the *3DBase* implementation, for which a thread updates a single point, herein the same thread operates on several grid points. Next, we describe a series of optimization techniques for the two baseline implementations. We focus mainly on minimizing warp divergence and global memory accesses. Besides global memory, the GPU architecture provides fast on-chip memory, registers and shared memory, which is distributed between threads and thread blocks respectively.

The starting point for the new kernel is the *3DBase* implementation. Since shared memory is allocated at thread block level, threads can cooperate when populating data blocks allocated in the shared memory. Shared memory arrays of size *blockXDim* · *blockYDim* · *blockZDim* are allocated. Each thread within a block loads the value of the grid point it handles from global memory to shared memory. With this technique, threads lying at the border of a thread block do not have access to all their neighbors and cannot compute the corresponding new values. Hence, the execution configuration is designed so as to ensure block overlapping in all directions (*3DShMOverL*). Starting again from the *3DBase* implementation, a different shared memory based strategy is developed. The shared memory arrays are padded with an additional slice on each side of the 3D block leading to a total size of (*blockXDim* + 2) · (*blockYDim* + 2) · (*blockZDim* + 2).

First, each thread populates the value of the grid point it handles in shared memory. Next, the threads located on the boundary of the block load the remaining data slices from global memory to the shared memory. To load points located outside of the block, conditional operations are introduced which cause branch divergence. Thus, each thread of a thread block has access to all its neighbors and is able to update the corresponding grid point (no overlapping between thread blocks is required - *3DShMNoOverL*). *T*he *2DBase* implementation can be optimized by storing redundant data (values from different slices) in registers - *2DReg*. As for the kernels with 3D thread blocks, shared memory can also be used to reduce global memory accesses for the kernels with 2D thread blocks. The size of the shared memory array chosen for this kernel version is (*blockXDim* + 2) · (*blockYDim* + 2). To allow each thread of the thread block to compute the new value of the corresponding grid point, additional slices are populated at each border of the 2D shared memory array.

For the *2DShM* implementation version the loading of the central section of the shared memory does not introduce any divergent branches since it is not conditioned. The loading of the slices with *y* index equal to 0 or *blockYDim* + 2 introduces a maximum of two divergent branches, one for each half-warp, depending on the compute capability of the GPU. On the other hand, the slices with *x* index equal to 0 or *blockXDim* + 2 lead to divergent branches and only one thread of the entire half-warp performs a read operation. This aspect may be alleviated by the cache memory, but this depends on the size of the slices.

To reduce branch divergence, the shared memory array is used only for the central section and for the slices with index equal to 0 or *blockYDim* + 2, while the other values are read from the global memory and stored into registers. Only the threads lying at the left or right border perform separate global memory reads (Figure 5 - *2DShMReg*), while the other values are safely read from the shared memory.

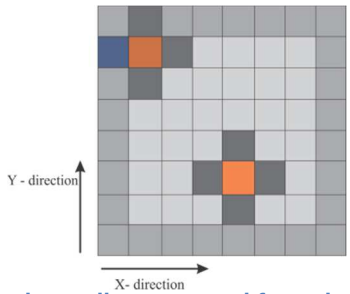


Figure 5: 2DShMReg: Northern and southern slices are read from the shared memory, eastern and western values from the global memory

Results

To evaluate the performance of the different strategies for running 3D stencil based algorithms on GPUs, we used three different NVIDIA GPU cards: GeForce GTX 480, GeForce GTX 660M and GeForce GTX 680 (the first one is based on the Fermi architecture, while the other two are based on the Kepler architecture), and the CUDA toolkit version 5.5. The numerical solution was obtained on a grid of 128x128x128 and was identical for all three GPU cards and for all implementation versions down to the 15th decimal, i.e. close to the precision of the double-type representation in computer data structures.

Table 1 displays the execution times for one time step for the three above-mentioned GPU cards, obtained for the seven different kernel versions introduced in the previous section. The GTX660M card leads to the largest execution times although it has been considerably later released compared to the GTX480 card. This can be explained however by the fact that this card was specifically designed for low power consumption, so as to be used in notebook PCs The GTX680 is the best performing card. The ratio of the execution times for the GTX660M and GTX680 cards varies between 4.26 and 5.56 for different kernel versions. This roughly reflects the inverse of the power consumption ratio, which is equal to 3.9.

Method	GTX480	GTX660M	GTX 680
3DBase	1.7	3.45	0.62
3DShMOverL	3.5	6.17	1.13
3DShMNoOverL	1.8	3.78	0.73
2DBase	1.2	3.09	0.63
2DReg	0.9	2.47	0.58
2DShM	1.2	2.87	0.59
2DShMReg	1.09	2.32	0.48

Table 1: Execution time [ms] for a single time step, obtained for the seven different implementation versions on three different GPU cards

Interestingly, whereas for the GTX660M and the GTX680 cards the *2DShMReg* kernel performs best, for the GTX480 card the *2DReg* kernel leads to the smallest execution time. Shared memory based optimizations were particularly important for pre-Fermi GPU cards. For the Fermi architecture these optimizations were not always leading to a better performance due to the fact that the global memory read operations were cached at L1 level. Even though the cache size is regularly small, it is efficient for algorithms based on Cartesian grids where data access patterns are regular [Shimokawabe et al., 2011]. For the Kepler architecture however the L1 cache is no longer used for caching global memory read operations, but only for register spilling. Hence, for the GTX480 card (Fermi), since the L1 cache is intensively used for caching global memory read operations, the *2DReg* kernel outperforms the *2DShMemReg* kernel. On the other hand, for the GTX660M and the GTX680M, since the L1 cache functionality is limited to register spilling, shared memory usage became more important, illustrated by the better performance of the 2DShMReg kernel.

11.2 Single-GPU solution of very large systems of linear equations using the preconditioned conjugate gradient method

11.2.1 Introduction

Our focus is on developing fast solutions for very large sparse linear systems of equations of the type $A \cdot x = b$, using parallel methods, where A is an NxN symmetric positive definite matrix. Such systems routinely appear when computing numerical solutions to PDEs, such as but not limited to the Finite Element Method. A widely-used iterative approach for solving such linear systems is the Conjugate Gradient (CG) method [Hestenes et al., 1952]. In each iteration, the CG method performs a Sparse-Matrix Vector multiplication, a process that converges in at most N iterations to the exact solution. While current GPU technology excels in fast processing of a large number of parallel computational threads, its global memory size can still create a bottleneck in solving large linear systems. We have developed a methodology for overcoming this limitation using a streaming based algorithm. Parallel algorithms for iterative solutions to the above system have been proposed already, e.g. [Ortega, 1988], and using the CUDA framework [Verschoor et al., 2012]. However, such solutions do not take into account RAM memory limitations on the size of the solution.

11.2.2 Methods and implementation

We propose a streaming based algorithm in order to overcome the global memory size limitation of the GPU, so as to be able to solve large systems arising in numerical solutions of various biomechanical PDEs. These may include but are not limited to fluid flow, bone or soft tissue deformation, etc. GPU cards have limited RAM memory (currently up to 12GB), which limits the size of the system of equations (currently to around 12 million equations). To alleviate this limitation we introduce a streaming based solution whose core idea is to store the matrix *A* on the CPU RAM, and to transfer it slice by slice to the GPU during the matrix-vector multiplication step of the PCG method (Figure 6 & Figure 7). Our streaming based strategy can be used either in the context outlined in the next section, or in the more general context of iterative methods that need to handle during each iteration an operation involving data that exceeds the GPU memory.

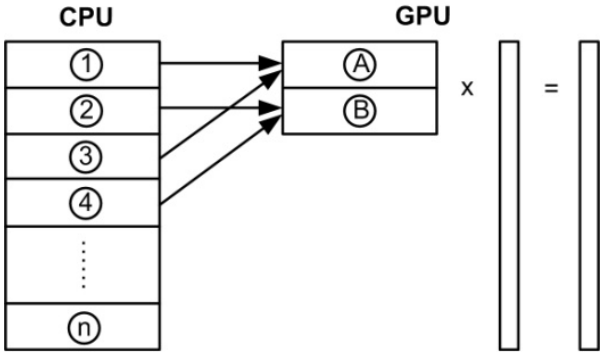


Figure 6: Slicing strategy for matrix A and succesive memory transfer to the GPU

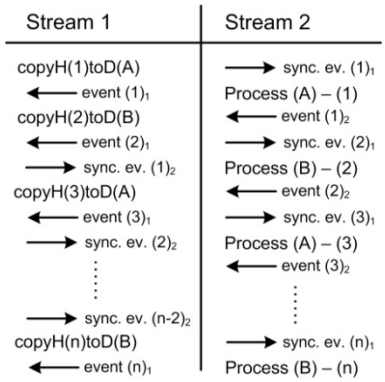


Figure 7: Streaming-based execution of the matrix-vector multiplication

When applying the PCG method, the majority of the memory required for the solution is occupied by matrix *A* and by the matrix used for the preconditioning [Saad, 2003]. To reduce the memory requirements, here we apply a Jacobi (diagonal) pre-conditioner which is stored as a vector, while *A* is stored in a sparse matrix format (e.g. ELLPACK) [Bell et al., 2008]. The other operations of the PCG method are either vector-vector or scalar-vector operations. The seven vectors are stored throughout the entire execution on the GPU due to their limited memory size. Figure 6 displays the slicing strategy of *A*. To ensure coalesced memory accesses by the threads of the same warp, *A* is stored in column major order. The slicing however is performed on a row basis. To limit the number of copy operations to two for one slice, we still store data in column major order, but only at slice level (all values of one slice are stored in consecutive locations). To reduce the execution time, the memory transfer operation of one slice is overlapped with the processing of another slice. To implement the overlapping behavior, two memory slices are allocated on the GPU (marked 'A' and 'B' in Figure 6): while data is copied into one slice, the other one is processed.

The matrix-vector multiplication is performed by using two different streams: one for the memory transfer operations – host (CPU) to device (GPU) – and one for the processing of the slices – kernel execution (figure 1.2.1(b)). The operations of different streams are executed out of order and need to be synchronized. Therefore we use CUDA events: an event $(i)_1$ is recorded after copying slice *i* from host to device, while an event $(i)_2$ is recorded after processing slice *i*. The synchronization at events $(i)_1$ is used to enable the processing of one slice only after the corresponding memory transfer operation is finished. The synchronization at events $(i)_2$ is used to enable the overwriting of one slice only after it was processed. This ensures a correct synchronization between the streams, irrespective of the relative duration of the memory transfer and processing operations.

11.2.3 Results

This section presents specific results for a bone structure analysis application, where Cartesian Finite Element (FE) models are typically used, whereas 3D bone tissue voxels obtained from microCT (computed

tomography) are converted into equally sized hexahedral finite elements [van Rietbergen, 2001]. To test the method from section 2, we used four different FE models. For each model, a volume of cube was meshed with 8-node hexahedral elements, with each node having 3 DOFs (translation in *x*, *y* and *z* dimensions). The four linear systems were first solved with the commercial software ANSYS (Release 14.0.3, ANSYS, Inc) on a six-core processor (Intel (R) Xeon (R) E-5-2670. 2.60 GHz) with 256 GB of RAM. Next, we solved the systems with the streaming based GPU algorithm on an eight-core i7 processor, 3.4GHz, with 8GB of RAM and a NVIDIA Kepler GTX680 graphics card, with 2GB of RAM. Table 2 compares the execution times of the CPU and GPU based algorithms. The speed-up varies between **44.2x** and **181.2x** for the largest system of equations.

Config.	Nr. of equations	CPU - Ansys	Streaming based GPU algorithm			
		Exec. time [s]	Exec. Time [s]	Nr. iter.	Number of slices in *A*	Speed-up
Test 1	2.260.713	3441	77.8	525	30	**44.2**
Test 2	4.102.893	21334	175.9	647	30	**121.3**
Test 3	5.582.601	35125	268.2	721	18	**131.0**
Test 4	7.057.911	66046	363.1	785	44	**181.9**

Table 2: Execution time [ms] for a single time step, obtained for the seven different implementation versions on three different GPU cards

11.3 Multi-GPU solution of very large systems of linear equations using the preconditioned conjugate gradient method

The method described in the previous section was also used to evaluate multi-GPU based implementation of the preconditioned conjugate gradient method.

11.3.1 Multi-node multi-GPU based implementation of the preconditioned conjugate gradient

The strategy we implemented for running the PCG method on multiple GPUs is:

One node is considered to be the master and performs all operations which do not involve the matrix-vector multiplication (initialization, copy operations, vector-vector operations, scalar-vector operations).

All nodes, including the master node, perform the matrix-vector multiplication step:

Each GPU stores a section of matrix A, which includes multiple slices (data transfer to the various GPUs is performed during initialization). All sections are approximately equal.

At each iteration, before starting the matrix-vector multiplication, each GPU receives the vector values used during the multiplication.

Each GPU performs in parallel the matrix-vector multiplication for the section of the matrix which was assigned to it.

Each GPU sends the resulting vector section back to the master node.

The Message Passing Interface (MPI) is used for transferring data between the nodes. The implementation described in this section is motivated by two goals:

Reduce the execution time of a single-GPU based implementation

Handle cases when matrix A does not fit into the RAM memory of the CPU

Table 3 displays the results for a two-node configuration for the *Test 2* and *Test 3* configurations used in the previous section. As one can observe, if a 1 Gbit/s node-to-node transfer speed is used the execution time

increases compared to the single-node implementation. This is given by the time required to transfer the vectors at each iteration. Conversely, if a 10 Gbit/s node-to-node transfer speed is used, the execution time decreases by a factor of around 1.5, which means that the time spent during data transfer is overcompensated by the time saved through the parallel computation of the matrix-vector product.

Configuration	1 Node	2 Nodes (1Gbit/s)	2 Nodes (10 Gbit/s)
Test 2	176s	450s	116s (1.51x)
Test 3	268s	704s	179s (1.50x)

Table 3: Comparison of single-node and multi-node GPU based implementation of the PCG method

11.3.2 Single-node multi-GPU based implementation of the preconditioned conjugate gradient

The strategy described in the previous section was also used for a hardware configuration containing multiple GPUs in a single node. The major difference is that instead of using MPI to transfer data from one node to another, all data transfers are performed through the PCI Express bus. The specific hardware configuration used for testing is: E6989 Rampage IV Extreme Main Board which has 4 PCI slots (2 at x16, 2 at x8), 3 GPUs: 1xGTX Titan Black, 2x GTX680. Due to the different transfer speeds over the PCI bus, the ideal partitioning of slices to different GPUs depends on the PCI bus to which each GPU is physically connected. Hence, we implemented a methodology for automatically determining the number of slices for each GPU so as to obtain an execution time for the matrix-vector multiplication, which requires approximately the same time on each GPU (so as to avoid idle times for the processors). Table 4 displays the results for three different configurations. One can observe that the multi-GPU implementation leads to smaller execution times. However, the measured speed-up is smaller than the theoretical value due to transfer of data between GPUs.

Configuration	Slice distribution	Exec. Time [s]	Speed-up	
			Measured	Theoretical
Titan (x16)	55	164.1 ± 0.87	-	-
Titan(x16), 680(x16)	35/20	120.0 ± 0.64	1.37	1.57
Titan(x16), 680(x8), 680(x8)	35/10/10	137.1 ± 0.69	1.20	1.57

Table 4: Comparison of single-node single-GPU and multi-GPU based implementation of the PCG method

11.4 Random forest based classification

11.4.1 Introduction

Machine Learning algorithms have been proven to be useful in a variety of application domains [Zhang, 2000]. Herein we focus on one of the most common machine learning applications: classification. We study how to effectively implement a random forests (RF) algorithm for data classification on GPUs by evaluating the performance of the algorithm in terms of execution time, compared to the CPU-based version. The random forest consists of multiple decision trees which can be generated and evaluated independently and can classify large amounts of data, described by a large number of attributes. Therefore, the random forest algorithm is very well suited for a massively parallel approach (implemented on GPUs).

11.4.2 Methods and implementation

Random forest is an ensemble classifier consisting of decision trees that combines two sources of randomness to generate base decision trees: bootstrapping instances for each tree and considering a random subset of features at each node [Breiman, 2001]. It is a supervised learning method: the training data consists of a set of training examples; each example is a pair consisting of an input object (typically a vector of features) and the corresponding desired output value. A supervised learning algorithm analyzes the training data and produces a model, which is then used to predict the output for new examples.

During the learning phase, the data that has reached a given leaf is used to model the posterior distribution. During the test phase, these posterior distributions enable the prediction for new unseen observations reaching a given leaf.

Because the training phase is done offline, the time required by this phase is not critical. Therefore, we only focus on the acceleration of random forest classification since in most of the cases this phase is done online and the execution time may be critical. The algorithm behind the testing phase is based on the fact that each internal node of a tree contains a simple test that splits the space of data to be classified and each leaf contains an estimate based on training data of the posterior distribution over the classes. The input data transformed into a feature vector is classified by propagating the information through all the trees and performing an elementary test at each node that directs it to one of the child nodes. Each decision node contains a test function that compares a feature response with a threshold to generate a binary decision. Once the sample reaches the leaf in each three in the forest, the posterior probabilities are combined (voting or averaging) to compute the final posterior probability. Traversing a large number of decision trees sequentially is ineffective when they are built independently of each other. Since the trees in the forest are independently built and the only interaction is the final counting of the votes, the voting part (classification) of the RF algorithm can be efficiently parallelized [Grahn et al., 2011].

The first step of the GPU based implementation of the RF classifier is to load all decision tree data structures of a RF into the GPU memory. Prediction is performed in a loop over all pixels of the input image. We determine the feature response for each pixel which will become the input vector for each tree. Each decision tree is then traversed in parallel to retrieve the probability distribution over all classes for the given pixel. The probability distributions from every tree in the forest are averaged and, finally, the result is copied back into the CPU memory.

To accelerate prediction on the GPU, we use multiple threads to process each image and multiple threads to process the RF trees in parallel. After images are loaded, we calculate the integral image in a pre-processing step. Calculating image integrals is expensive with respect to processing time. We accelerate it by calculating the integral for each of the five image channels in parallel with separate threads on the GPU. Prediction time depends on the complexity of the features but scales linearly with the number of trees, depth of the trees and the number of pixels in the input image. To take advantage of the massively parallel computing power of the GPU, instead of pre-computing all values for all possible features, we sampled the feature space at runtime and calculated the feature responses on demand.

Our strategy for storing the RF decision trees involves the mapping of the data structure describing the RF to a 2-D texture array which is stored in the GPU texture memory. These texture arrays are read only, and, since they are cached, this improves the performance of reading operations [Grahn et al., 2011]. The data associated with a tree is laid out in a four-component float texture, whereas the data of each node is stored on three separate columns in each channel of the texture array. We store the data of each node of a tree in the forest in sequential horizontal positions and different trees on separate rows. Data stored in the texture memory contains the position of the left and right child nodes, threshold values and all feature parameters required to evaluate the test for a node. If a node cached in the texture memory is a leaf node, then we add the probability distribution learned during training and the index of the leaf. To navigate through the tree during the evaluation, we use the tex2D function wich performs a texture lookup in a given 2-D sampler based on 2-D node coordinates and channel information (Figure 8: Algorithm for the binary decision tree

evaluation). Our strategy involves launching a kernel which evaluates the probability of the random forest with the number of threads equal to the number of candidates. As the number of feature candidates can exceed the maximum number of threads per block with a maximum of 1024, 1536 and 2048 for compute capability 1.2/1.3, 2.x and 3.x, respectively, several thread blocks are launched.

```
Algorithm. Random Forest prediction on the GPU
1. for all trees in the forest do
2.       while curNode has valid children do
3.              if children_found then
4.              float right  = evaluate_Harr_feature(boxCenter, curNode);
5.                 if (right < 0)
6.                    curNode <- leftChildPosition
7.                 else
8.                    curNode <- rightChildPosition
9.              else if leaf_node_reached then
10.             probability + = leaf_Node_Histogram
11.             total ++
12.             end if
13.      end while
14. end for
15. return probability / total
```

Figure 8: Algorithm for the binary decision tree evaluation

11.4.3 Results

This section presents specific results for a spine structure detection application, whereas the 3D input image was obtained from microCT (computed tomography) and where the random forest classifier is used to detect lesions. After the system was trained, tests were run with three different implementations of the classification algorithm. The implementation was tested with two available data sets and performance benchmarks for our implementation have been compared with two CPU based implementations. The experimental results indicate that our GPU-based implementation of the Random Forest algorithm outperforms the two CPU based algorithms (CPU single-core and CPU multi-core).

To test the method for the bone lesion detection, we used a Random Forest detector consisting of 100 trees, 27K normalized candidates (voxels) for each vertebra and 12K features. The execution configuration of the classification kernel specifies that the number of threads is equal to the number of candidates (27K), each block contains 128 threads (27K/128 blocks are used) and each thread traverses a candidate through all trees of the random forest.

Prediction time was compared on a machine equipped with Intel Core i7 CPU and a NVIDIA GPU GeForce GTX Titan Black. Table 5 compares the execution times of the CPU and GPU based algorithms. The speed-up varies between **170x** and **190x** compared to the single-core implementation and between **24x** and **28x.** Table 6 displays the total execution time which includes various other non-parallelized steps like the loading of the input image, etc. The RF based position detector classifier occupies the vast majority of the total execution time (~60%).

Implementation	Patient 1	Patient 2
CPU Multi-core	27.62 ± 0.39s	48.60 ± 0.41s
CPU Multi-core + GPU	17.85 ± 0.32s	22.57 ± 0.34s

Table 5 : Comparison of execution times of CPU and GPU based algorithms for the RF classifier

Implementation	Patient 1	Patient 2
CPU Single-core	73.01 ± 0.58s	182.33 ± 0.64s
CPU Multi-core	10.19 ± 0.15s	27.00 ± 0.25s
CPU Multi-core + GPU	0.421 ± 0.02s	0.979 ± 0.03s

Table 6: Total execution time for the detection system

11.5 Non-invasive assessment of aortic coarctation

11.5.1 Introduction

The Lattice Boltzmann Method (LBM) has been introduced in the 80's, and has developed into an alternative powerful numerical solver for the Navier-Stokes (NS) equations for modelling fluid flow. Due to the high computational requirements, there has been a lot of interest in exploring high performance computing techniques for speeding up the LBM algorithms. Herein we introduce a parallel implementation of the LBM designed for blood flow computations. To meet the high accuracy requirements of blood flow applications, computations are performed with double precision. The method is used for computing blood flow in a patient-specific aorta geometry with coarctation (CoA), containing the descending aorta and the supra-aortic branches. CoA is a congenital cardiac defect usually consisting of a discrete shelf-like narrowing of the aortic media into the lumen of the aorta, occurring in 5 to 8% of all patients with congenital heart disease [Ringel et al., 2007].

11.5.2 Methods and implementation

We considered the single relaxation time version of the equation, based on the Bhatnagar-Gross-Krook (BGK) approximation, which assumes that the macroscopic quantities of the fluid are not influenced by most of the molecular collisions: For a more detailed description of the Boltzmann equation and the collision operator we refer the reader to [Succi, 2001]. The current study focuses on 3D flow domains: we used the D3Q15 lattice structure.

The boundary conditions (inlet, outlet and wall) are crucial for any fluid flow computation. For the LBM, the macroscopic quantities (flow rate/pressure) cannot be directly imposed at inlet and outlet. Instead, the known values of the macroscopic quantities are used for computing the unknown distribution functions near the boundary. For the inlet and outlet of the domain we used Zou-He [Zou et al., 1997] boundary conditions with known velocity. For the outlet we used homogeneous Neumann boundary condition. The arterial geometry has complex boundaries in patient-specific blood flow computations, and hence, for improving the accuracy of the results, we used advanced bounce-back boundary conditions based on interpolations [Bouzidi et al., 2001]. The solid walls are defined as an isosurface of a scalar field, commonly known as the level-set function.

In the following we focus on the GPU based parallelization of the above-described LBM. The LBM is both computationally expensive and memory demanding [Astorino et al., 2012], but its explicit nature and the data locality (the computations for a single grid node require only the values of the neighbouring nodes) make it ideal for parallel implementations. Each node can be computed at each time step independently from other nodes. A first important difference between the CPU and the GPU implementation of the LBM is the memory arrangement. Regularly, on the CPU, a data structure containing all the required floating-point values for a grid node is defined, and then an array of this data structure is created (the Array Of Structures approach – AOS). This approach is not a viable solution on the GPU because the global memory accesses would not be coalesced and would drastically decrease the performance. Instead of AOS, the Structure Of Arrays (SOA) approach has been considered [Astorino et al., 2012]: a different array is allocated for each variable of a node, leading to a total of 35 arrays. The memory access patterns for the AOS and SOA approaches are displayed in Figure 9 for the three velocity components.

Two different kernels have been defined and are called at each iteration. Kernel 1 first computes the macroscopic quantities (velocity and density), by iterating through the 15 probability distribution functions. Then it applies the Zou-He boundary conditions at the inlet of the domain and it performs the collision step: first the equilibrium distribution function is computed and then the new probability distribution functions are determined. The second kernel focuses on the streaming step, the interpolated bounce-back boundary condition and the outlet boundary condition. All these operations require information from the neighbouring nodes. The operations of the second kernel are more complex since the grid nodes located at the boundary require a different treatment than the other nodes. Due to the high accuracy requirements of blood flow computations, and unlike previous researches, all computations were performed with double precision. Because the arrays and the execution configuration are one-dimensional, it is necessary to map the three-dimensional coordinates inside the grid to a global index used to access the data from the arrays.

The LBM is usually applied for a rectangular grid. For blood flow computations, the rectangular grid is chosen so as to include the arterial geometry of interest. In this case though, the fluid nodes represent only 1/5 or less of the total number of nodes. Hence, if the nature of the nodes (fluid/solid) is not taken into account, around 80% of the allocated memory is not used and around 80% of the threads do not perform any computations. To avoid this problem, we used an indirect addressing scheme, displayed in fig. Figure 10: Indirect addressing. Memory is only allocated for the fluid nodes and an additional array (called fluid index array) is introduced for mapping the global index determined with (1.5.7) to the fluid nodes arrays (negative values in the fluid index array correspond to solid nodes). The content of the fluid index array is determined in the preprocessing stage on the CPU and is required only during the streaming step.

11.5.3 Results

To compare the performance of the CPU based implementation of the LBM with the GPU based implementation for double precision computations, we considered three different NVIDIA GPU cards: GeForce GTX 460, GeForce GTX 650 and GeForce GTX 680. The CPU based implementation was run on an eight-core i7 processor using both single and multi-threaded code. Parallelization of the CPU code was performed using OpenMP. Three different 3D benchmark applications were first considered for determining the best performing GPU card: Poisseuille flow, lid-driven cavity flow and flow in an elbow shaped domain. The performance improvements were significant (speed-up varying between **11** and **44x** when compared to the multi-threaded CPU-based implementation), demonstrating that a GPU based implementation of the LBM is superior to a multi-core CPU implementation. The best performance is obtained for the GTX 680.

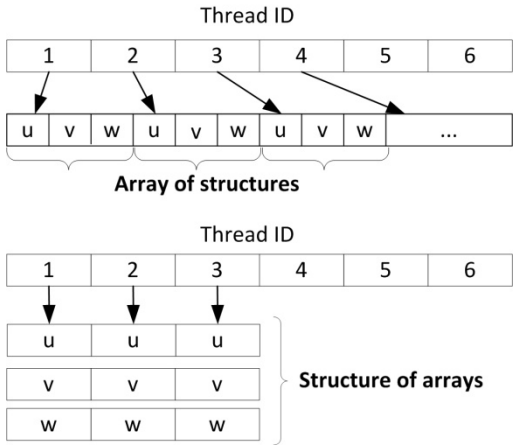


Figure 9: Memory access patterns: Array of Structures (top), Structure of Arrays (bottom);

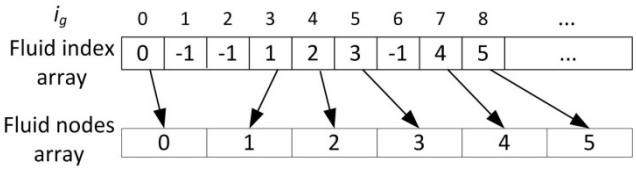


Figure 10: Indirect addressing

Once the GTX680 was determined as best performing GPU card for double-precision 3D computations, we used it to compute blood flow in a patient-specific aorta model with coarctation. To obtain the correspondence between the lattice units and the physical units, we used the method described in [Latt, 2007]. The computations were initialized with the equilibrium distribution function, and for the current research activity we focused on steady-state computations, i.e. we imposed the average value of the flow rate profile specified in the challenge. The grid size was set to 92x156x428 (6142656 nodes), of which only 518969 represented fluid nodes (less than 10%). Figure 11 displays the computation results obtained after 10000 time steps (the converged solution). The best performing execution configuration is with 128 threads per block and the speed-up compared to the execution time of the multi-threaded CPU implementation is of **19.42x**.

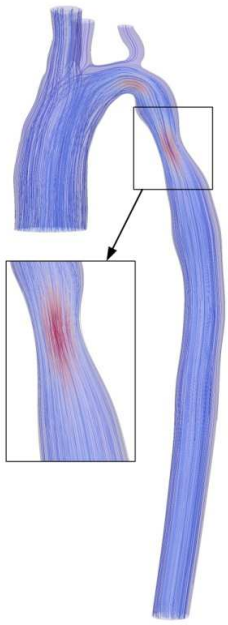


Figure 11: Computation result (streamlines) for the patient-specific coarctation geometry

12 MD-Paedigree Cloud Integration

12.1 Introduction

Cloud computing is a way to provide and to use a large number of computers connected through a real-time communication network. In science, cloud computing is a synonym for distributed computing over a network, and means the ability to run a program or application on many connected computers at the same time.

In common usage, the term "the cloud" is essentially a metaphor for the Internet. Marketers have further popularized the phrase "in the cloud" to refer to software, platforms, and infrastructure that are sold "as a service", i.e. remotely through the Internet. Typically, the seller has actual energy-consuming servers which host products and services from a remote location, so end-users don't have to; they can simply log on to the network without installing anything.

The major models of cloud computing service are known as Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS).

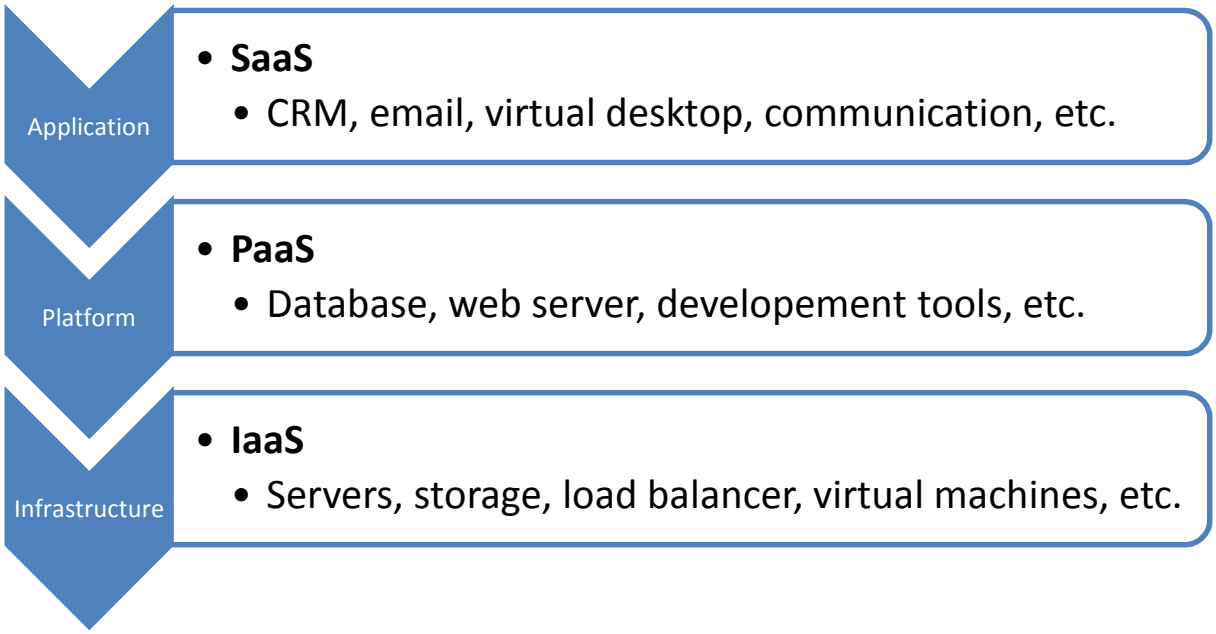


Figure 12: Cloud Architecture

Each concept will be explained in the next sub-sections.

12.1.1 Infrastructure as a Service (IaaS)

Infrastructure as a Service is a provision model in which an organization outsources the equipment used to support operations, including storage, hardware, servers, and networking components. The service provider owns the equipment and is responsible for housing, running and maintaining it. The client typically pays on a per-use basis.

12.1.2 Platform as a Service (PaaS)

Platform as a Service (PaaS) is a way to rent hardware, operating systems, storage, and network capacity over the Internet. The service delivery model allows the customer to rent virtualized servers and associated services for running existing applications or developing and testing new ones.

PaaS is an outgrowth of Software as a Service (SaaS), a software distribution model in which hosted software applications are made available to customers over the Internet. PaaS has several advantages for developers. With PaaS, operating system features can be changed and upgraded frequently. Geographically distributed development teams can work together on software development projects. Services can be

obtained from diverse sources that cross international boundaries. Initial and on-going costs can be reduced by the use of infrastructure services from a single vendor rather than maintaining multiple hardware facilities that often perform duplicate functions or suffer from incompatibility problems. Overall expenses can also be minimized by unification of programming development efforts.

On the downside, PaaS involves some risk of "lock-in" if offerings require proprietary service interfaces or development languages. Another potential problem is that the flexibility of offerings may not meet the needs of some users whose requirements rapidly evolve.

12.1.3 Software as a Service (SaaS)

Software as a Service (SaaS) is a software distribution model in which applications are hosted by a vendor or service provider and made available to customers over a network, typically the Internet.

SaaS is becoming an increasingly prevalent delivery model as underlying technologies that support Web services and service-oriented architecture (SOA) mature and new developmental approaches, such as Ajax, become popular. Meanwhile, broadband service has become increasingly available to support user access from more areas around the world.

Benefits of the SaaS model include easier administration, automatic updates and patch management, easier collaboration, global accessibility etc.

12.2 MD-Paedigree into the Cloud

MD-Paedigree has a relatively complex architecture composed of different layers. The Cloud is one of them (c.f. the introduction). This layer has mainly three purposes:

1. increase the computing power and/or storage capacities of the infrastructure when an activity peak is detected;
2. increase before the activity peak the computing power and/or storage capacities of the infrastructure (i.e. a user will need more resources than the system can provide in two weeks);
3. run experiments/pipelines on private resources which are not shared with anybody for security reasons (i.e. really sensitive datasets).

These three cases share several specificities but are quite different in many ways. Indeed, the first case needs a near real-time monitoring of the infrastructure usage in order to detect if more resources are needed or if we can remove some of them. The second and third ones concern activities which are planned and which have to be prepared in advance. In any case, all share the fact that resources have to be allocated/removed and configured in order to be integrated into the infrastructure. This implies, then, the ability to:

- provision/decommission virtual machines (VM) into a specific infrastructure (IaaS) ;
- configure automatically the VMs so that they can be used into the MD-PAEDIGREE infrastructure.

12.3 Virtual Machine provisioning

Cloud provisioning is the allocation of a cloud provider's resources to a customer. When a cloud provider accepts a request from a customer, it must create the appropriate number of virtual machines (VMs) and allocate resources to support them. In this context, the term provisioning simply means "to provide".

Managing such a provisioning is unfortunately, most of the time, specific to every single cloud infrastructure provider (IaaS). The Open Cloud Computing Interface (OCCI) [1], which was created in March 2009, aims at delivering a set of specifications through the Open Grid Forum (OGF) [2] for cloud computing service providers. OCCI was originally initiated to create a remote management API for IaaS model based services, allowing for the development of interoperable tools for common tasks including deployment, autonomic scaling, and monitoring. Unfortunately, only a couple of IaaS have implemented it [3].

Many commercial and open-sourced solutions target the management of multiple IaaS clouds in order to support multi-cloud operations in the IaaS layer. RightScale [4], which is the pioneer in this area, followed by other companies/solutions like scalr [5], enStratus [6], kaavo [7], and SlipStream [8] offer multi-vendor IaaS management solutions. In order to achieve this, each system supports a set of different IaaS providers and translates higher-level management commands to the IaaS specific commands.

In the context of MD-PAEDIGREE, the Slipstream solution was chosen for different reasons. First of all, this is a really powerful product which has already been used in many real environments. Moreover, the MAAT team has already been in touch with most of the developers for many years: this was a great advantage for the tests and integration of the system. Last but not least, the Slipstream developers know also very well the EMI/gLite middleware that is used into MD-PAEDIGREE so the integration was also easier.

12.3.1 SlipStream

SlipStream by SixSq, is a multi-cloud coordinated provisioning engine, provided as a PaaS. It provides a multi-machine provisioning system for defining and executing deployments based on cloud provider agnostic recipes. From within a unified web-based service, application users have the capability to deploy different parts of their application on separate Cloud providers to accommodate various resilience and redundancy of even running cost requirements. Slipstream uses connector architecture to communicate to a number of open-source (i.e. OpenStack) and proprietary (i.e. Amazon AWS) IaaS providers.

12.3.1.1 Main features

As stated in the documentation [9], SlipStream main features are:

- **Multi-machine Provisioning System**: it allows users to define and execute deployments, based on high-level recipes, independently from the cloud on which the recipes will be applied. Deployments include coordination and orchestration of virtual machines, including ordering and synchronisation of services ;
- **Multi-cloud Provisioning:** it supports multi-cloud deployments. This means users have the choice of a number of cloud service providers and technologies (public and/or private) when deploying virtual machines, from within the same SlipStream service. Furthermore, users can choose to deploy different parts of a deployment on different cloud services or regions, such that redundant and resilient behaviour is reached. It supports different clouds using a connector design ;
- **Support continuous integration processes with continuous deployment:** it encourages users to parameterise image creation and deployment recipes, such that key parameters (e.g. software version, package location, dependencies, inter-relationships) can be provided at runtime. This means that it is easy to integrate SlipStream with continuous integration servers to provide a full deployment chain ;
- **Independence from specific IaaS interfaces and hypervisors:** the SlipStream recipes are independent of specific IaaS interfaces, therefore avoiding vendor lock-in, allowing you to focus on configuration and deployment, instead of the specifics of each IaaS ;
- **Community sharing platform:** the SlipStream data model allows users to decide to share their image and deployment recipes with other users, thus contributing to a community effort.

12.3.1.2 Supported Cloud

SlipStream communicates to IaaS clouds services via a connector architecture. A growing number of connectors are available. The connectors talking to open source IaaS implementations are released under the same open source license as the SlipStream core, while connectors to proprietary solutions are closed source and available via a commercial license.

At the time of writing the list is the following:

- StratusLab (incl. OpenNebula): open source, available ;
- OpenStack: open source, available ;
- Abiquo: proprietary, available ;

- CloudSigma: proprietary, available ;
- Amazon EC2: proprietary, available ;
- VMWare vCloud: proprietary, available ;
- OCCI: open source, available ;
- Physical/Fixed: open source, available ;
- CloudStack: proprietary, coming soon ;
- IBM Smart Cloud Entry: proprietary, coming soon ;
- Microsoft IaaS Azure: proprietary, coming soon.

12.4 Virtual Machines configuration

12.4.1 Concepts - systems administrators needs help!

In order to powerfully take advantage of the elasticity of the cloud platform specificities, having an automated way of configuring VMs instances is crucial. Manual configuration is error prone and, in a reactive environment, where VMs can be quickly created and destroyed, it is not even reasonable and doable as the task would be too huge.

Within the Cloud, VMs can be very short-lived, as starting and creating VMs is a task happening very frequently – either for testing something new or for adding dynamically more computation power or service isolation/replication – the creation (depicted in the previous section) and the configuration of the VMs have to be automatized. Especially when adding a new instance of a particular service, it has to be configured in the exact same way as other running instances, and, if needed, it has to be automatically taken into account by the rest of the infrastructure.

One old, and hopefully deprecated way of managing multiple servers was to create custom home made scripts to assist the System Administrators in these huge boring tasks, but this has proven to have a lot of drawbacks:

- Each organization is “reinventing the wheel”, there is no capitalization on the knowledge of others ;
- It is really organization-specific (non-standard) ;
- It is not easy to design and to adapt to other use cases (reusability) ;
- It is not easy to test ;
- It is very time consuming.

But this error-prone method has been superseded by the emergence of a new kind of software tailored to address this exact use case: the configuration management tools and their ecosystem, with its best practices and dedicated tools.

It was widely adopted due to the real needs it helps to address, and that are shared by many organizations.

12.4.2 Configuration Management to the rescue

Efficient and productive VMs configuration is accomplished by the usage of the so-called configuration management tools whose adoption has emerged at first for non-cloud infrastructure and has been pushed forwards by the massive increase of Cloud deployments.

There is now a plethora of configuration management tools, from the venerable C-based CFengine [10] born in 1993 to the recent python-based Ansible [11] available since 2012. All these tools are based on different languages and frameworks allowing the system administrators to select a tool whose language they are comfortable or at least familiar with.

These tools work mainly with one or more repositories where the configuration is defined and later applied to the corresponding node/nodes. It is possible to create templates for a certain type of server and then apply it easily to one or more nodes by just configuring the specific part.

Configuration management tools are often associated with the use of a revision control tool for keeping the history of all the modifications, allowing to easily control, track, and rollback changes.

With automation a system administrator can manage a lot more servers than manually doable, and this will be achieved in a much less error-prone way as everything is version controlled and automated, reducing IT costs as well as improving infrastructure availability and reactivity. Deploying a new server is much more predictable, quicker, and safer when automated.

After a market evaluation, puppet has been chosen due to its openness, massive adoption, active community, great versatility (a lot of modules for managing a lot of services are already freely available), and its ruby nature (yes this is just a matter of personal taste, but admins have to like what they do to be efficient).

12.4.2.1 Puppet IT automation software

12.4.2.1.1 Introduction

Puppet [14] is a ruby-based Open Source software provided by Puppetlabs [12] and really open to the community that can publicly report issues or feature requests [20] and even have some code merged in the official code base if proven appropriate.

The code is hosted in GitHub [15] and can be easily reviewed, forked, adapted, and merged back in the main source code tree if the changes or fixes are useful for the community and follow the official guidelines.

Puppetlabs also provides a commercial product, Puppet Enterprise [13], offering custom proprietary high level interfaces as well as support.

Puppet is ruby-based but the configurations are expressed in a specific declarative language allowing to express the desired state of the target system. An extensive official documentation [16] is provided allowing any administrator to gain a quick knowledge of the features and internals.

Puppet is made of several components working in concert to allow for a seamless management of the infrastructure, from Hiera [17] for storing the hierarchical data corresponding to the infrastructure to Marionette Collective for the server orchestration.

12.4.2.1.2 Architecture

Puppet's architecture can be adapted to a lot of different workflows and deployment schemes, but a main standard configuration deployment prevails and will be presented.

In the standard configuration, there is a central service, called the puppet master, to which every node connects using their puppet agent to retrieve the configuration they have to apply locally.

Here is a quick overview of the anatomy of a traditional puppet run on a node - concepts and tools presentation will follow.

- The facts are collected on the node using facter [22] ;
- The facts are sent to the puppet master ;
- Based on the facts, the puppet master computes a *catalog* of *resources* configuration having to be applied on the agent ;
- The catalog is sent to the agent ;
- The agent applies the catalog ;
- The agent reports about the run to the puppet master.

Communication, authentication, and authorization is done using SSL certificates, everything is encrypted before getting over the wire making puppet usable across a public network.

Configuration is organized in reusable modules that can be shared among the users' community, even Puppetlabs is sharing a lot of their modules with the community.

Configuration is associated to a server (nodes in the puppet world) using node manifests that includes classes provided by modules and the appropriate classes' configuration.

Managed servers - being physical or virtual - provide a set of facts to puppet allowing puppet to make decision on the configuration or include them in it. Facts are server-specific variables, like hostname, network interfaces information, kernel-related information, it can be either software-related or hardware-relate. If needed it is even possible to manage custom facts. Facts are handled by Puppetlabs' facter [22] software, puppet's closest companion.

The node's *catalog* is a Directed Acyclic Graph (DAG) of the resources configuration (and their relationship) having to be applied to a server.

Configurations are text files saved in a source code revision control tool, as in the software engineering world, git [23] is used in MD-PAEDIGREE. It allows systems administrators to easily branch and test features on some specific node (real or local) and to keep all the changes history.

12.4.2.1.3 Language

Puppet uses a custom declarative language to express the desired state of the system, the configuration have to be organized in classes allowing a set of related resource configuration to agglomerate in one logical unit, for an easy reuse and concern-separation.

In puppet's terminology, a resource is "an independent atomic unit" with a specific type that can vary from a simple file to a service, software package, user, or cron job. Custom types can also be created if needed.

A *resource* can be declared and configured using *type*-specific *parameters* as well as *meta-parameters* applying to every type.

Resources are abstracted by *providers* from their operating system-specific implementation: marking a package resource as having to be installed on a system will transparently call the corresponding operating system-specific tool (yum on RedHat-like systems, aptitude or apt-get on Debian-like systems and so on) making it relatively easy to manage a set of heterogeneous systems.

Puppet's language also offers a number of features from conditional statements to relationship expression – the resource order in the catalog is not the sequential order of the resource in the puppet manifests and dependencies can be explicitly specified if needed.

12.4.2.1.4 Modules

Modules are a way of organizing a set of related-files in an easily distributable form, an example can be a Network Time Protocol (NTP) module allowing for the configuration of a NTP server to keep the systems' clocks synchronized. It could contain several classes, such as one for configuring a node acting as a client or one acting as a server for the other clients.

A module can be of any sort, i.e. it can configure only specific software, or even just a file but they can also configure a full server by gluing together different specific modules. Puppet modules aim at being independent, reusable and adaptable in order to be easily reused for others' needs.

Modules are mostly developed using the Puppet Language [18] but they can also include puppet libraries developed in pure ruby (they can also embed any script or any binary created with any possible language that the target host is able to execute).

The puppet ecosystem is very vast and lots of modules are freely available from multiple sources. Thus, to allow for an easier module search, a forge has been put on air by Puppetlabs [19].

Instead of writing all modules from scratch, a common practice is to use a module provider to avoid re-inventing the wheel, a lot of modules provider exists, but quality, coherence, and adaptability do vary a lot.

MD-PAEDIGREE uses mostly Example42 [21] modules as well as some Puppetlabs and various third-parties modules, and when no module was available or complete enough some homemade modules have been developed.

Example42 was chosen over other modules providers as they provide a set of coherent and standardized modules, covering a wide range of functionalities with a particular and expressed goal of making them reusable and standardized.

The main point of Example42 modules is the author attention to write quality, standardized and re-usable modules that follow a well-designed and documented structure and leave all room possible to allow modules' users to adapt the module to their needs without needing to alter the module's internals.

We also faced the case where some modules were missing a feature we required or some little bugs, and the author was really open and reactive: we have a number of fixes and new features having been integrated using the standard GitHub workflow (i.e. forking a project, making required changes, and sending a pull request to the original project to get code reviewed and merged back into the main source code tree).

All these modules covers our needs but it can be quite complicated to use different modules from multiple providers as some modules are dependent on others and usage can largely differ (i.e. example42's apache module won't allow to configure Puppetlabs' puppet module apache Virtual Host).

When developing custom modules the community best practices are followed as much as possible, but as the puppet ecosystem is a moving part in constant evolution keeping up dated with the latest best practices can be time-consuming.

The main rules that are followed are to create reusable and autonomous specific modules that are extensible.

The most complex modules that had to be developed and supported for MD-PAEDIGREE are those used to manage the grid nodes, allowing for the creation and automatic configuration of a grid node.

The MD-PAEDIGREE grid-specific modules started as a fork of an old publicly available gLite module [24] placed in the public domain.

The module was first ported to EMI [25] then to UMD [26] and the required fixes were added, the modules do not have been published back to GitHub but it will soon be the case, once they will have been cleaned for specificities and full reviewed.

Profile and roles modules can be created to add a higher level of encapsulation of modules and abstraction: a server can include a specific role that will include multiple profiles allowing for great reusability and following the "Don't Repeat Yourself" (DRY) principle.

The different modules used in MD-PAEDIGREE allow for a configuration from the NTP server to the grid node going through the package manager and shell configuration.

12.4.2.1.5 Node manifests

The node manifests are used to assign and configure classes found in modules to a specific node, they link the classes expressed in the modules to the server promoting one anonymous generic useless server to one specific well-crafted server design for a precise and deterministic usage.

In order to allow to represent the infrastructure organization to a hierarchy – like a global general configuration with default package repositories and base configuration, a specific location with custom network configuration and finally a particular node operating at this specific location – an inheritance mechanism is provided (for nodes and classes) but it is quite buggy and has been proven quite unhelpful regarding the representation of the infrastructure as an organized hierarchy.

This leads to the creation of Hiera, a tool allowing to express the infrastructure and its configuration in a more robust way, while allowing a complete separation of the data and the code: the configuration is now data-centric and can be purely expressed using the data.

12.4.2.1.6 Data centric configuration using Hiera

As clearly advertised by its name, Hiera is a hierarchical database allowing us to express the infrastructure structure, avoiding data duplication as much as possible and allowing for a greater flexibility in modules configuration as well as a better separation between code and data.

Without using Hiera, modules have to embed some data in them, as all the operating system's specific parts have to be configured accordingly to the operating system type, the location of a service configuration file on a Debian-based system is not the same as on a RedHat-based system, so both file paths have to be stored into the module. This forces developers to use a lot of conditionals statements (case, if…) into the classes, making them uselessly complicated to read and maintain. Supporting a new operating system or altering its default configuration required to edit the code which is a much more error prone practice than editing the data and potential newly introduced bugs could affect all the infrastructure, not only a specific subset or even only a single node.

Cleaning the manifests from all that conditional code, make them shorter, cleaner, easier to understand and maintain.

With Hiera, the modules can separate their code from the data, and users can easily tweak the modules without having to edit them improving module reusability.

Hiera is a quite recent player in the puppet ecosystem and the community is still testing different possibilities of integration.

One of the interesting setups is having a node-less deployment, where all the classes and their configuration are assigned from the data sources provided by Hiera. There is no specific per-node manifest, only one default node manifest used by all nodes, thus the data stored in Hiera will determine what classes to load and how to configure them. This node-less setup allows us to have an automatic configuration of the new nodes, based on their site or role, without having to edit the puppet manifests and minimizing the need for data editions.

Hiera configuration specifies an ordered and prioritized list of data sources, from the most specific to the least specific: the MD-PAEDIGREE infrastructure is using mainly the following sources:

- Node-specific data source ;
- Location-specific data source ;
- Role-specific data source ;
- Common data source.

Hiera provides several query tools which allow us to retrieve information, to either merge all the results in one set or to pick only the most specific value, making overloading of a configuration parameter really easy, i.e. the common data source can define a default NTP server, but the location-specific data source can provide the nearest and best NTP server for this particular location and if needed a node can have a specific "personal" configuration.

The Hiera data sources names can contains facts' values that will be expanded. This allows us to have data sources correlated to a special server fact. If a data source is not present it will be skipped without failure.

Data sources are simple text files containing data that can be specified using the JSON or YAML formats. As they are also stored as pure text files, a complete changes tracking is possible.

12.4.2.2 The Marionette Collective – making the puppet runs dancing

Historically puppet agent was intended to be running continuously on the servers, and was supposed to contact the puppet master on a defined time interval. It was also possible to configure the agent to listen

for requests from the master asking the agent to request a new catalog and run a pass, providing a sort of a push model, in order to force agent to run when changes were pushed to the master. The main problem of this approach was that the agent was very memory hungry and even could be stuck eating resources.

In order to avoid this problem different models have been used and developed, from running the agent using cron or even using a Nagios plugin, but it was quite hack-ish or at least less feature-full (as an example, using cron no kick option was available). This leads an active community member (R.I.Pienaar [27]) to develop a tool intended to address this problem as well as dealing with other distributed and/or wide infrastructure-related use cases: the Marionette Collective, a client/server software which allows administrators to orchestrate and inventory a set of servers.

The Marionette Collective, mcollective, is built around some modern concepts like publishers/subscribers using a Message Bus and designed in a modular way with a plugin-based extension system. Multiple plugins are provided by default. This allows us to execute different kinds of task on remote servers, such as listing available servers or issuing Nagios NRPE commands. The main plugin that is interesting with puppet is the puppet plugin, allowing us to launch and monitor puppet runs on remote servers. It even allows operators to filter the servers list using facts gathered from nodes or classes that got assigned to nodes. As an example it is possible to request a puppet run on all the Debian-based nodes having the NTP class included. This plugin also provides a way of ensuring that the puppet master won't be taken down due to too much simultaneous connections of servers: it can be told that only 10 concurrent access are permitted, the plugin will ensure that no more than 10 servers are running a puppet pass in parallel.

The mcollective agent runs as a service and is also a lot lighter than the puppet agent service, even if some important fixes have been done lately to make it less resource hungry.

The Marionette Collective was found so relevant to the puppet community that Puppetlabs did even hire the original developer, financing the support and active development of mcollective and the kick option was removed from the puppet codebase. Finally, the original author, R.I.Pienaar, became the software architect of the aforementioned Hiera database.

12.4.2.3 Puppet/mcollective drawbacks/problems

Nonetheless, using puppet, Hiera and mcollective can also cause problems, as it becomes quite easy to propagate an error impacting every server of the infrastructure, on twitter, @devops_borat depicted this using the humoristic formula:

> "To make error is human. To propagate error to all server in automatic way is #devops."

The ruby nature of puppet and mcollective (Hiera is quite different, it is clojure-based requiring to run in a Java Virtual Machine), can also be problematic as the ruby world tends to be very reactive and the production-grade operating systems tends to have outdated libraries (called gems) or even non-packaged at all in some cases. In order to ease things for the users, Puppetlabs provides public packages repositories but it is currently required to install some specific gems on the server, and those gems are out of the control of the distribution's package manager.

Modules publicly available have not the same quality and sometimes it is difficult to make them interoperate as two modules of different providers could depend on two different dependencies having the same name. Therefore, evaluating, selecting, integrating, and updating modules can be a quite complex and cumbersome task.

In order to prevent all the related problems, it is really important to have a strong (and of course automated) way of understanding if the agent really does apply the desired configuration without any side effects. A first step towards a more sustainable infrastructure can be achieved by using different environment (production, development) allowing for the separation of development from production code. It also has to be enforced by another practice adopted and adapted from the software engineering world,

more specifically from the Test Driven Development movement: viewing, managing, using infrastructure as code and testing it proactively, extensively, and thoroughly.

12.4.3 Testing

In order to setup a trustful deployment of a Virtual Machine on the Cloud that can be used with confidence, it is essential to understand if the configured computer can address the need it is supposed to. This can be accomplished by using automated test assessing the reliability and accuracy of a specific module or node configuration.

In order to cover the most possible bugs, tests have to operate at different levels, it is possible to do unit testing of modules to ensure that the atomic parts of the module are working right, as well as some sort of integration tests using serverspec and Vagrant [28].

Some of the modules publicly available are already doing some unit tests, mostly using rspec and commonly used ruby testing tools, but test quality goes from absent, incomplete, outdated to an almost exhaustive coverage.

Unit tests help a lot as they allow us to understand quickly if all the parts of a module behave as expected in a timely efficient manner. But in order to validate a full node configuration, the complete assessment of server integration is required, which consists of more complex tests.

The final configuration of a node can be tested using the serverspec tool to apply some Behaviour Driven Development (BDD) and Test Driven Development (TDD) principles to server configuration and infrastructure testing.

Serverspec tests allow us to express a server configuration, the tests will validate if the server matches its expected behaviour. Checks are run through a Secure Shell (ssh) remote connection and will assess that the puppet configuration provides the expected result. As a side effect it also documents the server's configuration.

Vagrant is a tool that was developed to easily manage local virtual machines creation and automatic configuration, providing an automated way to efficiently create a local development or test environment perfect for disposable re-use.

Initially, Vagrant aimed at deploying machines only locally, using a VirtualBox provider, but due to its plugin-based architecture offering in a great flexibility and extensibility, a large amount of virtual machines providers have been developed by the community giving administrators the possibility to easily create virtual machines on VMWare, OpenStack, and other cloud providers.

For MD-PAEDIGREE, the nodes' configuration is tested using serverspec and Vagrant makes testing theme in local virtual machines easier. Vagrant allows to effectively setup and teardown a virtual machine to ensure that tests are done on a clean production-like environment.

By using a Continuous Integration tool such as Jenkins [29] it will even be possible to launch the tests on every push to the puppet repository to easily track what any change might cause.

12.4.4 Conclusion

The usage of configuration management tools helps increasing the productivity as well as the overall quality of service.

System administrators are shifting to a developer role, because the infrastructure is now managed as a code (versioning, testing, automation…), and they have to interact, communicate, and exchange actively with the developers to fix the gap between development and operations, making the deployments on the production infrastructure more reliable, easy, and predictable. Using tools is one aspect of this emerging cultural and technical movement called *devops* – a compound word made of development and operations – but one of the most important thing, at least as important as the tools used, is ensuring that there is a good and frequent communication between the developers and the operators to ensure that the infrastructure

meets its applications needs. The infrastructure has also to be taken into account during the development phase to ensure that the move to production will be a smooth as possible. Obviously system and users' monitoring reports are also a key to the overall success, as they assess if the infrastructure and applications are operating as intended.

Puppet creates a development environment that is as close as possible to the production environment, reducing the window for unseen bugs. Together with Vagrant, it even allows developers to have a complete and coherent local development environment as close as possible to the production one.

Puppet can be seen as an infrastructure documentation as it represents the desired state of the infrastructure, the use of Hiera to store data will only enforce such a view making it very useful to determine what is used for a node or which node is used for a task.

Using tools such as puppet, provide a way of managing the heterogeneous individual servers like a coherent heard of anonymous individuals allowing for an easy and smooth growth and enforcing the infrastructure sustainability, reliability, and resilience.

It is also a great help in tracking down and reacting to problems, since it leads to a new iteration of infrastructure configuration and management improvements.

As all this tools are meant to be automated, they could easily integrate in a continuous integration/delivery setup, having tests automatically played in a production-like virtual machine: if tests are successful there is an automated deployment of the build to the development infrastructure.

Additionally, to reduce the management costs and improving the infrastructure reliability, one of the target goals is to be able to handle automatically all sorts of system failures, leading to automatically commission or decommission nodes, according to the infrastructure requirements. Another goal is to have the smallest possible infrastructure downtime and maximum adaptability to a required usage peek/growth allowing for a seamless and elastic adaptation to the infrastructure workload.

13 Self-assessment Criteria

13.1.1 Self-assessment estimation

14.1	**The grid network is installed and configured for MD-Paedigree**		
	Upper limit : 100% within Month 24	Lower limit: 75% within Month 24	100%
	The grid is installed and functional as needed		
14.2	**Load Capacity analysis is provided**		
	Upper limit : 100% within Month 24	Lower limit: 75% within Month 24	75%
14.3	**The number of gateway corresponds to what the analysis defines / the workload is supported / the application of WP15 and 16 are compatible with the system.**		
	Upper limit : 100% within Month 24	Lower limit: 50% within Month 24	75%
	Enough Gateways are installed for the platform optimal operating but important gateways are not yet installed. WP15 and WP16 tools are not fully in the infostructure platform environment.		
14.4	**Cloud provider and APIs have been chosen and qualified to be integrated the MD-Paedigree solution / GPU computing Technology has been chosen for the GPU processing layer.**		

	Upper limit : 100% within Month 12	Lower limit: 75% within Month 12	100%
	Cloud API has been chosen and tested. CPU technology is chosen and ready to run for some modelling experiments.		
14.5	**All the application from WP15-16 can be deployed onto the cloud / the imaging calculation solution from WP15-16 can run using GPU.**		
	Upper limit : 100% within Month 12	Lower limit: 50% within Month 12	70%
	Cloud infrastructure and machine provisioning are ready to use. Tests from WP15-16 are still expected. GPU processing has significant result on image analysis.		
~~**14.6**~~	~~**ADP is able to decompose its dataflow management to be able to manage data distributed as it is in the system.**~~		
	~~Upper limit : 60% within Month 24~~	~~Lower limit: 40% within Month 24~~	
	Removed		
~~**14.7**~~	~~**ADP is decomposable enough to run onto the grid using calculation distribution and parallelization abilities.**~~		
	~~Upper limit : 60% within Month 24~~	~~Lower limit: 40% within Month 24~~	
	Removed		
14.6 Replaces old 14.6 and 14.7	EXAREME is integrated into the system providing distributed processing and parallelization of resourse/time consuming algorithms		
	Upper limit : 3 within Month 24	Lower limit: 2 within Month 24	2.5
	1. Design and develop a Complex Dataflow Processing Engine with native UDF support (madIS integrated with ADP) 2. Develop complex UDFs to be used in SQL-based dataflows executed by the Complex Dataflow Processing Engine. 3. Provide distributed processing and parallelization of resourse/time consuming algorithms 4. Integration with the platform		
14.7 Ex 14.8	**VPH-Share recommendations that the solutions have to follow are defined for all the applications**		
	Upper limit : 100% within Month 24	Lower limit: 75% within Month 24	50%
	MD-Paedigree reuse directly some part of VPH-Share project. However, VPH-Share constraints for applications have not been clearly defined.		
14.8 Ex 14.9	**All the applications that are developed for MD-Paedigree (in particular the ones from WP15-16) respect the VPH-Share defined recommendations and will manage to use Pandora security, privacy and integration functionalities to be servable as a service Through the gateways.**		
	Upper limit : 60% within Month 24	Lower limit: 40% within Month 24	30%
	Different applications have put efforts to integrate with Pandora services. However, most of them are not integrated as services.		

13.1.2 Corrective action

A special attention will be paid to the correction on self-assessment criteria 14.7 and 14.8 in order to restore a status that is up to the level needed. Synchronization meetings with VPH-Share members to clearly define constraints on application will be introduced during the agile methodology process (cf D17.1) and a particular attention will be put on application integration as a service for all the developed applications.

14 Conclusion

Alpha version implements most of the functionalities, in line with the original estimated advancement. All partners work in concordance to provide the most adapted and usable system to physicians. Current system is well sized for the project purpose but some non-technical aspects impacts the original assumption that all gateways could be stored on OPBG PCDR system. Cloud and GPU integrations are done and ready to be used, but no funds have been planned for infrastructure empowerment, so the consortium will have to find a way to provide these resources if modelling challenges are submitted to the system.

The recent decision to use agile methodology, as described in D17.1, will involve the user in the development choices and testing, resulting in better communication and understanding between the infostrucure group, physician and modellers. This better communication and synchronisation between software providers and users should improve the project's efficiency, resulting in a user-oriented system ready to be used.

Annexe 1 – GPU Publications

The methods, algorithms and implementations described above have lead to the publication of the following papers:

1. Vizitiu, A., Itu, L.M., Lazar, L., Suciu, C. *Double precision stencil computations on Kepler GPUs*, Proc. of the 18th Inter. Conf. on System Theory, Control and Computing - ICSTCC 2014, Sinaia, Romania, October 15-17, 2014.

2. Vizitiu, A., Itu, L.M., Nita, C., Suciu, C. Optimized *Three-Dimensional Stencil Computation on Fermi and Kepler GPUs*, 18th IEEE High Performance Extreme Computing Conference, Waltham, MA, USA, Sept. 9-11, 2014.

3. Itu, L. M., Sharma, P., Georgescu, B., Kamen, A., D., Suciu, C., Comaniciu, D. *Model Based Non-invasive Estimation of PV Loop from Echocardiography*, Proc. of the 36th Annual Inter. Conf. of the IEEE Engineering in Medicine & Biology Society - EMBC 2014, Chicago, USA, August 26-30, 2014.

4. Itu, L. M., Suciu, C. *A method for modeling surrounding tissue support and its global effects on arterial hemodynamics*, IEEE International Conference on Biomedical and Health Informatics, Valencia, Spain, June 1-4, 2014.

5. Nita, C., Chen, Y., Lazar, L., Mihalef, V., Itu, L.M., Viceconti, M., Suciu, C., *GPU Accelerated Finite Element Analysis of Trabecular Bone Tissue*, Proc. of the Virtual Physiological Human Conference, Trondheim, Norway, Sept 9-12, 2014.

Annexe 2 – GPU References

[Alastruey et al., 2009] J. Alastruey, S. Nagel, B. Nier, A. Hunt, P. D. Weinberg, and J. Peiro, "Modelling pulse wave propagation in the rabbit systemic circulation to assess the effects of altered nitric oxide synthesis", Journal of Biomechanics, vol. 42, pp. 2116–2123, 2009.

[Astorino et al., 2012] M. Astorino, J. Becerra Sagredo, and A. Quarteroni, "A modular lattice Boltzmann solver for GPU computing processors", SeMA journal, vol. 59, pp. 53-78, 2012.

[Bell et al., 2008] Bell N, Garland M. Efficient sparse matrix-vector multiplication on CUDA. NVIDIA Technical Report NVR-2008-004. 2008.

[Bessems, 2008] D. Bessems, "On the propagation of pressure and flow waves through the patient-specific arterial system", PhD Thesis 2008, Techincal University of Eindhoven, Netherlands.

[Bouzidi et al., 2001] M. Bouzidi, M. Firdaouss, and P. Lallemand, "Momentum transfer of a Boltzmann-Lattice fluid with boundaries," Physics of Fluids, vol. 13, pp. 452-3459, 2001.

[Breiman, 2001] L. Breiman, "Random forests," Machine Learning, vol. 45, pp. 5–32, 2001.

[Formaggia et al., 2013] L. Formaggia, A. Quarteroni, and C. Vergara, "On the physical consistency between three-dimensional and one-dimensional models in haemodynamics", Journal of Computational Physics, vol. 244, pp. 97–112, 2013.

[Grahn et al., 2011] Grahn, H.; Lavesson, N.; Lapajne, M. H. & Slat, D. (2011), CudaRF: A CUDA-based implementation of Random Forests., in Howard Jay Siegel & Amr El-Kadi, ed., 'AICCSA' , IEEE, , pp. 95-101

[Hestenes et al., 1952] M. Hestenes, E. Stiefel, Methods of conjugate gradients for solving linear systems, J. Res. Nat. Bur. Stand. 49 (1952) 409–436

[Kirk et al., 2010] D. Kirk, and W.M. Hwu, Programming Massively Parallel Processors: A Hands-on Approach, London: Elsevier, 2010.

[Latt, 2007] J. Latt, "Hydrodynamic limit of lattice Boltzmann equations", PhD Thesis, Universite de Geneve, Geneve, Switzerland, 2007.

[Liu et al., 2007] Y. Liu, C. Dang, M. Garcia, H. Gregersen, and G. S. Kassab, "Surrounding tissues affect the passive mechanics of the vessel wall: theory and experiment", American Journal of Physiology - Heart and Circulatory Physiology, vol. 293, pp. 3290-3300, 2007.

[Nita et al., 2013] C. Niţă, L. M. Itu, and C. Suciu, "GPU Accelerated Blood Flow Computation using the Lattice Boltzmann Method", IEEE High Performance Extreme Computing Conference, pp. 1-6, Sept. 2013.

[Olufsen et al., 2000] M. Olufsen, C. Peskin, W. Y. Kim, E. M. Pedersen, A. Nadim, and J. Larsen, "Numerical simulation and experimental validation of blood flow in arteries with structured-tree outflow conditions", Annals of Biomedical Engineering, vol. 28, pp. 1281-1299, 2000.

[Ortega, 1988] 5. Ortega, J. Introduction to parallel and vector solution of linear systems, Plenum Press, 1988.

[Phillips et al., 2010] E. Phillips, and M. Fatica, "Implementing the Himeno benchmark with CUDA on GPU clusters", IEEE Intern. Parallel & Distributed Processing Symposium, pp. 1-10, April 2010.

[Ringel et al., 2007] R.E. Ringel, and K. Jenkins, "Coarctation of the aorta stent trial (coast)", 2007, http://clinicaltrials.gov/ct2/show/NCT00552812.

[Saad, 2003] 2. Saad Y, editor. Iterative methods for sparse linear systems. 2nd ed. Philadelphia: Society for Industrial and Applied Mathematics; 2003.

[Shimokawabe et al., 2011] T. Shimokawabe, T. Aoki, T. Takaki, T. Endo, A. Yamanaka, N. Maruyama, A. Nukada, and S. Matsuoka, “Peta-scale phase-field simulation for dendritic solidification on the TSUBAME 2.0 supercomputer”, Intern. Conf. for High Performance Computing, Networking, Storage and Analysis, pp. 13-18, 2011.

[Succi, 2001] S. Succi, The Lattice Boltzmann Equation - For Fluid Dynamics and Beyond. New York: Oxford University Press, 2001.

[van Rietbergen, 2001] 4. van Rietbergen, B. 2001. "Micro-FE analyses of bone: state of the art." Adv Exp Med Biol no. 496:21-30.

[Verschoor et al., 2012] 6. Verschoor, M., Jalba, A. Analysis and Performance Estimation of the Conjugate Gradient Method on Multiple GPUs, Parallel Computing 38 (2012), pp. 552-575

[Zaspel et al., 2013] P. Zaspel, M. Griebel, “Solving incompressible two-phase flows on multi-GPU clusters”, Computers & Fluids 2012, vol. 80, pp. 356-364, 2013.

[Zhang, 2000] Zhang, D. 2000. Applying machine learning algorithms in software development, In Proc. of Monterey Workshop on Modeling Software System Structures, Santa Margherita Ligure, Italy, pp. 275–285.

[Zou et al., 1997] Q. Zou, and X. He, “On pressure and velocity boundary conditions for the Lattice Boltzmann BGK model,” Physics of Fluids, vol. 9, pp. 1591-1598, 1997.

Annexe 3 – CLOUD References

[1] OpenCloud Computing Interface (OCCI) : http://occi-wg.org/
[2] Open Grid Forum: http://www.ogf.org/
[3] OCCI implementations: http://occi-wg.org/community/implementations/
[4] Rightscale: http://www.rightscale.com/
[5] Scalr: http://www.scalr.com/
[6] enStratus: https://www.enstratius.com/
[7] kaavo: http://www.kaavo.com/
[8] SlipStream: http://sixsq.com/products/slipstream.html
[9] SlipStream doc.: https://slipstream.sixsq.com/documentation
[10] CFengine: http://cfengine.com/
[11] Ansible: http://www.ansible.com
[12] Puppetlabs: https://puppetlabs.com
[13] Puppet enterprise: https://puppetlabs.com/puppet/puppet-enterprise
[14] Puppet open source: https://puppetlabs.com/puppet/puppet-open-source
[15] Puppet source code: https://github.com/puppetlabs/puppet
[16] Puppet documentation: https://docs.puppetlabs.com
[17] Hiera: http://projects.puppetlabs.com/projects/hier
[18] Puppet Language: http://docs.puppetlabs.com/puppet/latest/reference/lang_summary.html
[19] Puppet Forge: https://forge.puppetlabs.com
[20] Puppet bugs tracker: https://tickets.puppetlabs.com
[21] Example42: http://example42.com
[22] Facter: https://puppetlabs.com/facter
[23] Git: http://git-scm.com
[24] Puppet gLite module: https://github.com/cristim/glider-glite
[25] European Middleware Initiative (EMI): http://www.eu-emi.eu/
[26] Unified Middleware Distribution (UMD): http://repository.egi.eu/category/umd_releases/
[27] R.I. Pienaar : http://www.devco.net
[28] Vagrant: http://www.vagrantup.com/
[29] Jenkins: http://jenkins-ci.org