



27-03-2015

Open Call Deliverable OCA-DS1.1

Advanced Networking for the EU genomic research (ARES)

Open Call Deliverable OCA-DS1.1

Grant Agreement No.: 605243
Activity: NA1
Task Item: 10
Nature of Deliverable: R (Report)
Dissemination Level: PU (Public)
Lead Partner: University of Perugia
Document Code: GN3PLUS14-1284-49
Authors: Gianluca Reali, Mauro Femminella, Emilia Nunzi, Dario Valocchi, Matteo Picciolini, Valerio Napolioni

© GEANT Limited on behalf of the GN3plus project.

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7 2007–2013) under Grant Agreement No. 605243 (GN3plus).

Abstract

The technical motivation that have brought to the ARES proposal is essentially the expected unsuitability of current data networks for supporting genomic processing services in the near future. The essential aims of the project was to design, implement, and evaluate experimentally network solutions for exchanging genomic data sets necessary for executing genomic medical services.

ARES combines some cutting-edge networking technologies, data management policies, virtualization techniques, and distributed computing for realizing an overall architecture which provides: simple, cloud based, access to medical and research personnel; scalable network footprint, through combining dynamic content distribution networking, optimized cache management, adaptation to the user practices, overlay networking, and advanced signal protocols able to discover available resources off-path; differentiated network services in accordance to different degree of seriousness of the situation needing genomic processing.

The ARES experiments were configured the so as to be totally compliant with the GÉANT infrastructure, and experiments have provided results both in terms of service differentiation and scalability, which are the main metrics of assessing the suitability of the ARES service architecture. All the project achievements, in terms of research, implementation, experiments, dissemination and standardization have met the project expectations.

Abstract

The technical motivation that have brought to the ARES proposal is essentially the expected unsuitability of current data networks for supporting genomic processing services in the near future. The essential aims of the project was to design, implement, and evaluate experimentally network solutions for exchanging genomic data sets necessary for executing genomic medical services.

ARES combines some cutting-edge networking technologies, data management policies, virtualization techniques, and distributed computing for realizing an overall architecture which provides: simple, cloud based, access to medical and research personnel; scalable network footprint, through combining dynamic content distribution networking, optimized cache management, adaptation to the user practices, overlay networking, and advanced signal protocols able to discover available resources off-path; differentiated network services in accordance to different degree of seriousness of the situation needing genomic processing.

The ARES experiments were configured the so as to be totally compliant with the Géant infrastructure, and experiments have provided results both in terms of service differentiation and scalability, which are the main metrics of assessing the suitability of the ARES service architecture. All the project achievements, in terms of research, implementation, experiments, dissemination and standardization have met the project expectations.

Table of Contents

Executive Summary	9
1 Introduction	12
2 Project Management	15
2.1 Operational and financial coordination	15
2.1.1 Overall coordination	16
2.1.2 ARES Management Objectives	17
2.1.3 ARES Management Achievements	17
2.2 Project evaluation and risk assessment	18
2.3 Chapter Conclusions	22
3 Scenarios, Requirements, and Architecture	23
3.1 Definition of scenarios and requirements	24
3.1.1 General requirements	24
3.2 Genomic Pipelines	29
3.2.1 Copy Number Variation (CNV) pipeline	29
3.2.2 Differential Expression (DE) pipeline	30
3.2.3 Computing requirements	31
3.3 System architecture and technologies	35
3.3.1 Network Architecture and Components	35

3.3.2	Selected Technologies	45
3.4	Chapter Conclusions	51
4	Distributed Networking and Applications	52
4.1	Advanced signaling protocols	52
4.1.1	NSIS Signaling	54
4.1.2	NetServ signaling	74
4.1.3	RESTful signaling	81
4.1.4	Preliminary Experimental and simulation results	85
4.2	Network and Service Management	97
4.2.1	Resource management through OpenStack APIs	98
4.2.2	GCM operation	98
4.2.3	LOIB operation	98
4.2.4	Network management through OpenStack APIs	99
4.2.5	Network configuration management	99
4.2.6	Numerical results	111
4.3	Chapter Conclusions	112
5	Implementation and Integration	113
5.1	Deployment and integration of hw and sw infrastructure	113
5.1.1	ARES Software Architecture	113
5.1.2	ARES Hardware Architecture	115
5.1.3	ARES System Integration	117
5.2	Deployment of the genomic processing software	120
5.3	Chapter Conclusions	125
6	Experiments and Evaluation	126
6.1	Plan for experimental assessment	126
6.1.1	Plans for Assessing the ARES Individual Components	127
6.1.2	Plans for Assessing the ARES Integrated System	137
6.2	Execution of experiments and analysis of results	138
6.2.1	Genomic Pipelines	138
6.2.2	Network performance	167
6.3	Chapter Conclusions	184
7	Dissemination, Standardization, and Exploitation	185
7.1	Dissemination and standardisation	185

7.1.1	ARES Public dissemination	185
7.1.2	ARES Standardization Activities	191
7.2	Exploitation Plans	191
8	Conclusions	193
	References	196
	Glossary	199

Table of Figures

Figure 3.1:	ARES general requirements	24
Figure 3.2:	Sketch of the interactions happening on interface ui.	25
Figure 3.3:	Sketch of the ARES differentiated schedule period	26
Figure 3.4:	Sketch of the interactions happening on interface ni.	29
Figure 3.5:	Copy Number Variation (CNV) Pipeline	30
Figure 3.6:	Differential Expression (DE) Pipeline	31
Figure 3.7:	Processing time of the CNV pipeline vs. number of allocated virtual cores, for different values of the allocated RAM size.	33
Figure 3.8:	Processing time of the DE pipeline vs. number of allocated virtual cores, for different values of the allocated RAM size. Use of the STAR aligner [18] instead of Bowtie2 [17] in the DE pipeline (Figure 3.6).	34
Figure 3.9:	Architectural components of the ARES experimental infrastructure.	37
Figure 3.10:	PoP architecture.	38
Figure 3.11:	ARES Cloud management.	41
Figure 3.12:	Networking and service paradigms contributing to the optimized solution.	46
Figure 3.13:	The NetServ Architecture.	49
Figure 3.14:	A graphical sketch of the main OpenStack components.	50
Figure 4.1:	ARES Cloud signaling	53
Figure 4.2:	PoP internal architecture.	55
Figure 4.3:	Q-mode gossip encapsulation with 1) Q-forward, and 2) Q-drop.	57
Figure 4.4:	Example of possible subset of the universe and solution for the discovery problem for node 1. The tree T_1 is drawn in bold.	60

Figure 4.5: Pseudocode of the main loop of a Gossip Initiator.	62
Figure 4.6: GIST extension packet format.	64
Figure 4.7: Pseudocode of a Gossip Session of the Gossip Initiator.	65
Figure 4.8: Pseudocode of the Daemon Loop of a Gossip Responder/Forwarder.	66
Figure 4.9: off-path bubble of radius 2 exploded from the node X.	70
Figure 4.10: off-path balloon of radius 2 sent from the node X to the node Y.	71
Figure 4.11: off-path hose sent from the node X to the node Y.	72
Figure 4.12: GIST off-path MRI.	73
Figure 4.13: Pseudocode of the GIST Broadcast algorithm.	74
Figure 4.14: Pseudocode of the GIST Multicast algorithm.	74
Figure 4.15: NetServ SETUP example message.	75
Figure 4.16: NetServ + off-path hose within the Ares scenario.	77
Figure 4.17: Sequence diagram describing the interactions between the application, the NSLP, and the NTLP inside a node.	79
Figure 4.18: PROBE responses aggregation in a hose signaling session from R1 to R5 with radius of 1 IP hop.	80
Figure 4.19: Messages flow between a GPVM and the LOIB.	84
Figure 4.20: Géant PoP-level logical topology (January 2014). Grey nodes include a data centre, laying 1 IP hop away from the PoP. The relevant physical PoP-level topology is depicted in Figure 4.21.	86
Figure 4.21: Géant PoP-level physical topology (January 2014). The connections with GE, AR, AZ, and RU are not shown, thus we have not included these nodes in the logical PoP-level topology depicted in Figure 4.20.	87
Figure 4.22: Transient experimental results vs. size of the PTS list: (a) convergence time on full topology, (b) signaling overhead on full topology, (c) convergence time on sparse topology, and (d) signaling overhead on sparse topology. For One-hop (OH), $B=\infty$ indicates a memory size larger than the number of PEs in the network, whilst $B=10$ indicates a memory size equal to 10 PEs. 95% confidence intervals are shown.	88
Figure 4.23: Full and sparse topology (a) convergence time and (b) steady phase overhead as a function of H.	92
Figure 4.24: Full topology signaling bandwidth: (a) steady phase experimental data for all solutions, (b) Leaf experimental data and models, (c) Random experimental data and models, (d) One-hop experimental data and models.	93
Figure 4.25: Convergence time expressed in gossip cycles for the Leaf solution. 95% confidence intervals are shown.	94
Figure 4.26: Off-path signaling overhead in full topology. Light colour indicates TCP traffic, solid colour indicates UDP traffic.	95
Figure 4.27: Off-path signaling overhead in sparse topology ($P=0.8$). Light color indicates TCP traffic, solid color indicates UDP traffic.	96

Figure 4.28: Signaling overhead over the whole network as a function of the path length, for different value of the hose radius. Network configuration with parameter P=1.	97
Figure 4.29: Sketch of a time realization of service request process.	100
Figure 4.30: Sketch of the ARES differentiated schedule period.	100
Figure 4.31: Sketch of the ARES distributed requests	101
Figure 4.32: Graphical interpretation of the optimization procedure involving two cost functions.	103
Figure 4.33: Example of a network making evident the presence of local minima in cost function.	105
Figure 4.34: Flow diagram of the algorithm used for identifying disjoint paths	107
Figure 4.35: Sample network architecture for demonstrating the proposed routing algorithm	108
Figure 4.36: Steps of the routing algorithm enabling parallel download.	110
Figure 4.37: Simulations results: ARES vs. baseline cloud computing paradigm, as a function of the input load.	112
Figure 5.1: the full testbed functional architecture, and the relevant mapping on the physical hardware.	118
Figure 5.2: the large scale testbed architecture (Géant topology), and the relevant mapping on the physical hardware.	119
Figure 5.3: Copy Number Variation (CNV) Pipeline	121
Figure 5.4: Differential Expression (DE) Pipeline	122
Figure 5.5: Genomic reads used for executing the CNV pipeline.	123
Figure 5.6: Genomic reads used for executing the DE pipeline	124
Figure 6.1: Resource occupancy of the CNV pipeline vs. time. Input read size of 2.6 GB, taken from the site of the 1000genomes projects.	128
Figure 6.2: Flow diagram of the experiments to be used for assessing the ARES integrates system.	131
Figure 6.3: GÉANT IP network topology.	133
Figure 6.4: PoP architecture.	134
Figure 6.5: Simulations results: ARES vs. baseline cloud computing paradigm, as a function of the input load.	136
Figure 6.6: Flow diagram of the experiments to be used for assessing the ARES integrates system.	138
Figure 6.7: Execution times in hours versus number of CPU cores allocated to the VM for the CNV pipeline; input size is indicated close to each curve.	165
Figure 6.8: Execution times in hours versus amount of memory allocated to the VM for the CNV pipeline; input size is indicated close to each curve..	165
Figure 6.9: Execution times in hours versus number of CPU cores allocated to the VM for the DE pipeline; input size is indicated close to each curve.	166
Figure 6.10: Execution times in hours versus amount of memory allocated to the VM for the DE pipeline; input size is indicated close to each curve.	166
Figure 6.11: Network footprint in TB versus requests.	174

Figure 6.12: Gain (the ratio between baseline footprint and ARES footprint), evaluated by means of a moving average of the 5 latest samples.	175
Figure 6.13: Box plot of the ratio between network time and processing time for both ARES and baseline schemes.	176
Figure 6.14: Box plot of the ratio between network time and processing time for both ARES and baseline schemes.	177
Figure 6.15: Estimated probability density function for the path stretch for ARES (for baseline it is always 1).	178
Figure 6.16: Network footprint in TB versus requests.	180
Figure 6.17: Gain (the ratio between baseline footprint and ARES footprint), evaluated by means of a moving average of the 5 latest samples.	181
Figure 6.18: Box plot of the ratio between network time and processing time for both ARES and baseline schemes.	182
Figure 6.19: Box plot of the data throughput for both ARES and baseline schemes.	183
Figure 6.20: Estimated probability density function for the path stretch for ARES (for baseline it is always 1).	184

Table of Tables

Table 2.1: List of the project milestones successfully achieved	20
Table 2.2: Evaluation of risks.	21
Table 3.1: Information exchanged through the ui-interface and the corresponding data exchanged at the ni-interface	25
Table 3.2: Medical Requirements	27
Table 3.3: report of information exchange through the ni-interface	28
Table 3.4: Configuration and minimum resource requests of VMs implementing the genomic pipelines.	32
Table 3.5: Requirements and functions of the ARES experimental platform.	45
Table 4.1: List of NetServ message parameters.	76
Table 4.2: List of the fields of a submitTask JSON payload.	83
Table 4.3: List of the fields of a Download JSON payload.	85
Table 6.1: Configurations of the VM that have been set for evaluating timing performance of the CNV pipeline	129
Table 6.2: Sample of results collected by the assessment of the CNV pipeline.	130

Table 6.3: Experiments grid for CNV pipeline.	139
Table 6.4: Experiments with 2 CPU cores and 4 GB of RAM.	140
Table 6.5: Experiments with 4 CPU cores and 4 GB of RAM.	141
Table 6.6: Experiments with 8 CPU cores and 4 GB of RAM.	142
Table 6.7: Experiments with 16 CPU cores and 4 GB of RAM.	143
Table 6.8: Experiments with 6 CPU cores and 8 GB of RAM.	144
Table 6.9: Experiments with 4 CPU cores and 8 GB of RAM.	145
Table 6.10: Experiments with 2 CPU cores and 8 GB of RAM.	146
Table 6.11: Experiments with 16 CPU cores and 8 GB of RAM.	147
Table 6.12: Experiments with 2 CPU cores and 12 GB of RAM.	148
Table 6.13: Experiments with 4 CPU cores and 12 GB of RAM.	149
Table 6.14: Experiments with 8 CPU cores and 12 GB of RAM.	150
Table 6.15: Experiments with 16 CPU cores and 12 GB of RAM.	151
Table 6.16: Experiments grid for DE pipeline.	152
Table 6.17: Experiments with 16 CPU cores and 32 GB of RAM.	153
Table 6.18: Experiments with 8 CPU cores and 32 GB of RAM.	154
Table 6.19: Experiments with 4 CPU cores and 32 GB of RAM.	155
Table 6.20: Experiments with 2 CPU cores and 32 GB of RAM.	156
Table 6.21: Experiments with 16 CPU cores and 40GB of RAM.	157
Table 6.22: Experiments with 8 CPU cores and 40 GB of RAM.	158
Table 6.23: Experiments with 4 CPU cores and 40 GB of RAM.	159
Table 6.24: Experiments with 8 CPU cores and 40 GB of RAM.	160
Table 6.25: Experiments with 16 CPU cores and 64 GB of RAM.	161
Table 6.26: Experiments with 8 CPU cores and 64 GB of RAM.	162
Table 6.27: Experiments with 4 CPU cores and 64 GB of RAM.	163
Table 6.28: Experiments with 2 CPU cores and 64 GB of RAM.	164
Table 6.29: File size and processing for all emulated genomics pipeline.	173
Table 6.30: Normalized signaling overhead for ARES and baseline system.	179
Table 6.31: Normalized signaling overhead for ARES and baseline systems.	182
Table 7.1: List of scientific (peer reviewed) publications	187
Table 7.2: Comprehensive list of dissemination activities during the reporting period.	190

Executive Summary

This document contains a comprehensive description of the motivations, activities, results and future expected activities of the Géant Open-Call project ARES.

The technical motivation that has brought to the ARES proposal is essentially the expected unsuitability of current data networks for supporting genomic processing services in the near future. The first sign of this unsuitability is observed at the Beijing Genomics Institute, which delivers genomic data, stored in memory devices, through express couriers. Due to the so called “big drop” of sequencing cost, genomic data sets have been flooding data centres, and a widespread belief is that in the near future a lot of medical activities will be based on genome processing. In this scenario, specific network solutions are needed for the exchange of massive set of genomic data in the network so as to allow most of medical centres, and not only the most prestigious ones, to make use of such services. In addition, even research activities are being highly affected by the increasing use of genomes and, under this perspective, a further massive exchange of genomic data are needed for supporting networked genome processing services.

The ARES project has combined some cutting-edge networking technologies, data management policies, virtualization techniques, and distributed computing for realizing an overall architecture which provides:

- Simple, cloud based, access to medical and research personnel
- Scalable network footprint, through combining dynamic content distribution networking, optimized cache management, adaptation to the user practices, overlay networking, and advanced signal protocols able to discover available resources off-path.
- Differentiated network services in accordance to different degree of seriousness of the situation needing genomic processing.

Research and implementation activities in ARES have been carried out through a logical project organization and evolution, from service specifications, through research and implementation phases, and the final experimental assessment. This evolution has been monitored by project management organization which has checked to production of the project milestones by the times scheduled in the proposal.

The ARES implementation has made a large use of open-source components, so as to stimulate the diffusion of the software components on research communities. In addition, the software design has been accomplished by selecting the same hypervisors and cloud management tools actually used in Géant, so as to make their instantiation simple and promote their usage by the Géant community. In this regard, a collaboration with the Géant personnel managing the Géant Testbed Service (GTS) has been activated. At the time of writing, the GTS is not equipped for importing virtual machines (VMs) from external with the resource configuration needed for the ARES VMs. Nevertheless, these capabilities will be made available soon and specific agreement has been established for the use of GTS after the formal end of the project for executing further ARES experiments.

The essential content of Chapter 2 is the description of how the management of ARES has assured compliancy with the GN3plus practices and has coordinated the personnel working on ARES for achieving the desired results in due time. The following Chapter 3, named “Scenarios, Requirements, and Architecture”, introduces all the technical entities making part of the ARES architectures, defines their interfaces formally, including the information flow, functions, and requirements. Thus, this chapter is the baseline of all the subsequent research and implementation activities. The optimized algorithms that provides the improvements of the ARES project are illustrated in Chapter 4, “Distributed Networking and Applications”. In this chapter we illustrate the mathematical background of the optimized algorithms, and the pseudo-code of the procedures and protocols that have been implemented and allow all the entities introduced in Chapter 3 to cooperate for implementing a networked processing environment. In addition, we also show the cloud-based user interface that allow medical and research personnel to use the ARES capabilities in a simplified manner, without the need of having to deal with the typical issues of network scientists. The implementation of both genomic processing tools and the network components are illustrated in Chapter 5. This implementation is largely based on open-source components. We include the hardware configuration hosting the ARES VMs. The experimental assessment of ARES is shown in Chapter 6. In these experiments, we configured the test-bed so as to be totally compliant with the Géant infrastructure. We deployed 72 VMs, interconnected through an overlay network, having the same topology of the Géant network. Although this choice does not impact on the general validity of results, the selection of the Géant topology was done either in consideration of the desirable future use of the software components, and to highlight any possible issues related to the Géant topology. Actually, bottleneck effects due to the Géant topology result in very extreme case studies and overloaded networks. Our experiments have provided results both in terms of service differentiation and scalability, which are the main metrics of assessing the suitability of the ARES service architecture. The final chapter focuses on the ARES results in terms of dissemination and standardization, with a final description of the exploitation plans of the ARES partners. As for the dissemination activities, a detailed list of events show an intense dissemination, made of research papers, magazine papers for not very specialized people, and general dissemination events such as interviews that appear in web pages. In summary the dissemination activity, even in consideration of the short duration of the project, has largely met expectations. For what concerns standardization, ARES has also produced a proposal of a signaling protocol submitted to the IETF.

While drawing the final conclusion of the documents, the final chapter indicates a number of possible future activities that can leverage on the ARES achievements.

The ARES achievements can be evaluated against the principal objectives of the 1st Géant Open Call as follows:

- *Deliver world-class services to research and education communities, building on GN3's success.*
The optimized delivery of genomic services through advanced networking has the potential to increase the Géant service portfolio, and the Géant-compliant implementation allows the research and education community to make use of the ARES experimental technology easy, beginning with the Géant Testbed Service.
- *Support the growth of R&E communities within Europe in both breadth and depth, and expose them to talent elsewhere.*
The development of the ARES components through open-source packages allows them to be distributed on the research communities, and the broad scope of genomic processing makes the ARES packages of interest also for other future activities other than medical applications.
- *Innovate to meet the needs of the community, and act as a catalyst to translate this into a competitive European ICT sector.*
The need of pursuing the ARES objectives has been deeply illustrated, and it is also demonstrated by the large number of initiatives worldwide that aim at making the networked genomic processing affordable, reliable, and increasingly popular. Thus, this sector is considered highly strategic for a number of medical and business sectors and its growth in the EU is worth of being considered a primary and strategic objective. ARES has given an appreciable contribution that can be leveraged for further strategic objectives for the EU.
- *Collect and share knowledge about network technologies and services through cooperation and community gathering.*
Finally, it is worth to highlight that the ARES achievements have been made possible by the cross-fertilization of two research areas, namely genomics and data networking. This interdisciplinary approach is believed to be a key feature that is recommended for future research activities promoted by the Géant community.
- *Showcase innovation in the area of data communications and telecommunications by enabling the use of innovative GÉANT network technologies, services and infrastructure.*
The advanced Géant technologies, in particular the large bandwidth connections accessible by the bandwidth on demand capabilities are suitable for hosting the ARES components. We have indeed struggled to make them compliant with the Géant Infrastructure, and the first scheduled step is the porting ARES in the Géant Testbed Service shortly, as agreed with the relevant personnel, when the needed capabilities for uploading and suitably configuring VMs in the Géant Testbed Service will be deployed.
- *Achieve maximum market visibility for the technologies and services developed by GÉANT.*
The intense dissemination activity that has accompanied the project in its lifetime is illustrated in the document, and appears to fully meet expectations. In particular, the contribution to standardization and the research papers can guarantee a significant diffusion in the research and technical communities related to the Géant activities.

- *Undertake focused work packages that further the continued enhancement and ongoing operation of the leading-edge GÉANT network.*

The ARES work packages have been defined so as to efficiently direct the project activities towards its objectives. Among them, work packages 3, 4, and 5 deal with cutting-edge technologies of the networking and services areas and has the potentials to be successfully exploited for further enhancing the Géant service deployment introduce resource management techniques targeted to genomic computing, which is indeed one of the most challenging and promising research and business areas.
- *Enhance the combined GEANT and NRENs ability to provide world class connectivity and services to the knowledge community and to push the state of the art in innovation in Research and Education networking.*

The overlay ARES networking is designed to have an inter-domain scope. Thus, the EU researchers served by different Géant NRENs can easily interoperate for making use of the genomic processing tools made available through the ARES case studies and deliver further services by leveraging on the ARES distributed networking.

1 Introduction

This document is the final deliverable of the Géant open-call project ARES (**A**dvanced Networking for the EU Genomic **R**esearch).

The objective of the document is to illustrate all the phases of the projects, the achieves results, and delineate the expected future activities that can benefit from the ARES results.

The background of the project consists of a rapidly increasing production and usage of genomic data for medical and research purposed, that is going to produce a big-data challenge for data networks. In fact, the costs for sequencing either a unit of DNA or the whole human genome have been decreasing over the time faster than the Moore's law. This cost evolution is referred to as "the big drop". It has begun in 2008, and is due to the introduction of novel sequencing machines. Under a very practical viewpoint, this means that the cost per unit data produced decreases more rapidly than the cost for storing a unit data and, more importantly, for

distributing a unit data. Hence, the bottleneck of the process of effectively using the genome information will reside on the ICT side, in particular in data networks.

After this introduction, the document is organized in chapters, each corresponding to a project Work Package (WP), and chapter titles correspond to WP titles, plus a concluding chapter.

Chapter 2, named “Project Management”, illustrates the ARES management organization, which had both to be compliant with the GN3plus organization and to guarantee suitable internal working condition through lightweight management procedures leading to the full achievements of the contractual requirements. The coordination of only two partners, with sound expertise in international collaborations, has highly helped the ARES management.

Chapter 3, named “Scenarios, Requirements, and Architecture”, establishes the foundation of the ARES research and experiments through a detailed definition of entities, interfaces, information flow, functions, and requirements. The resulting research challenges have been faced through the definition of the service requirements for genomic data exchange, deployment of a suitable architectural framework, for modular and virtualized services, able to provide suitable insights for the service operation and relevant perception, deployment of mechanisms and protocols for service description, discovery, distribution, and composition.

Chapter 4, named “Distributed Networking and Applications”, provides a comprehensive description of the complete signaling exchange necessary to establish a genomic processing services. We illustrate how all the entities forming the ARES distributed networking environment interact, which messages they exchange, and how results are provided to requesting users. We show that from the user perspective the ARES services are accessible through a simple cloud interface. In addition, we detail the underlying procedures that are executed in order to accomplish a request, based on CDN content delivery and advanced off-path NSIS signaling.

The second objective of the chapter is to show the “intelligence” that governs this exchange. Since one of the ARES objectives is to provide genomic services with a quality depending on both the service delivery time and network resource optimization, we have introduced suitable optimization algorithms, along with the needed mathematical details for a rigorous description.

Chapter 5, named “Implementation and Integration”, illustrates the implementation, deployment and integration of the ARES hardware and software infrastructure, along with the final layout of the ARES experimental test-bed. This description includes the ARES genomic experiments that constitute the application scenarios deployed, and the software components of the ARES experimental scenarios. The description of the used hardware infrastructure is included. The whole experimental system has been finalized at the month 16 since the beginning of the project, in accordance with expectations. The illustrated infrastructure has been widely tested in operational scenarios representing the Géant situation, in terms of topology and capabilities. All the components of the testbed have been implemented by using the virtualization techniques that allows porting them into the Géant testbed service (GTS) and Géant PoPs.

Chapter 6, named “Experiments and Evaluation”, is dedicated to the ARES experimental activities. It begins with the description of the experimental approach and methods, based on the GUM (Guide to the expression of

uncertainty in measurement) specifications. A full assessment of the genomic pipelines follows, the results of which have been used for the comprehensive ARES experiments that demonstrate the effectiveness of the ARES proposal. In particular, the experimental results show significant benefits in terms of scalability, network footprint and service delivery time in comparison with general cloud approaches. These results have been found through real experiments executed by deploying virtual machines and protocols over a real overlay network having a topology equal to the Géant one.

Chapter 7, named “Dissemination, Standardization, and Exploitation”, illustrates the dissemination events in ARES, contribution to standardization, which consists of a resource discovery protocol submitted to IETF, and the exploitation plans of the partners of ARES.

Finally, Chapter 8 draws conclusions about the ARES expectations, activities, and results, and sketched the expected future activities that will leverage on the ARES achievements.

In conclusion, the impact of the ARES results can be checked against the potential impact illustrated in the project proposal in terms of research, operational costs, diffusion of genomic services, standardization, software availability, European research support, and benefits for other applications beyond medicine:

2 Project Management

The description of the management activities of the project ARES are organized in a specific WP dedicated specifically to *project management* (WP1).

It is worth to mention preliminarily that the management of ARES have been quite simple, since the project size is relatively small and all administrative issue have been continuously supported by the Géant personnel. In particular, the mailing lists opencalls@geant.net and gn.finance@dante.net have allowed receiving valuable supports for general issues and administrative issues, respectively. In addition, the mailing list gn3plus-all@geant.net has allowed to be informed of the ongoing Géant activities, and the mailing list tf-noc@terena.org has allowed to coordinate with the Géant personnel to organize to use of the Géant Testbed Service.

The ARES internal management organization has guaranteed total compliance with the GN3plus organization.

As for the internal ARES management, the best management practices already used in previous projects, in which the ARES personnel was involved, have been used. In any case, since the ARES project includes two partners only, the general goal of ARES project management effort has provided the project with a light-weight, flexible management service capable of ensuring an intensive, flexible, open-intended dialog among the two partners concerning key strategic and scientific issues, with rapid and effective decision-making on technical and organizational issues, full and effective compliance with contractual requirements.

2.1 Operational and financial coordination

The coordinating partner of the project ARES is the University of Perugia (UoP), which has a long and successful experience on management and development of large international research projects, both at technical and administrative levels. Although the two partners of ARES have joined the GN3plus consortium as formally independent beneficiaries, each providing independent financial reports, the University of Perugia has always monitored the correct progresses of the activities in the project, as appears in the monthly RAG reports uploaded in the GEANT portal by the 15th day of each subsequent month.

2.1.1 Overall coordination

The coordinating partner was in charge of coordinating reporting activities on the project activities. Regular progress reports are derived from contributions from all the partners.

In addition, the coordinating partner organised periodic meeting of the ARES partners.

The project coordinator ensured that the periodic reports from partners were completed, on time, and that the consumption of resources and progress are compatible with the technical achievements and reflect the actual ARES progress. In addition, the project coordinator have actively given an major contribution for writing and integrating the various contribution to the milestones documents and to the final project deliverable.

The overall strategic vision for the project was established and analyzed by the Management Committee (MC) on the basis of the achieved results. Due to the presence of two only partners, the Management Committee is the only committee having a management role in ARES. It includes the project coordinator, Gianluca Reali, and the technical coordinator of the partner GGB, Emilia Nunzi. It regularly happened that other personnel was invited to the meetings of the Management Committee in order to report about the specific achievements and needs for a successful outcome of the project. The Central Project Office was in charge of assisting the ARES personnel in administrative issues. Given the small size of the project, the administrative service of the Dipartimento di Ingegneria of the University of Perugia was found suitable to this aim.

2.1.1.1 Management structures in individual partner organizations

Each partner has appointed:

A Project Manager, with overall responsibility for the successful completion of the activities assigned to the partner, including administrative requirements. The main role for project managers was to represent the partner within the MC and define (and, if necessary, revise) the strategies pursued by individual partners and to ensure that these were backed by adequate resources. The role of Project Manager was a part time job. Project Managers were not involved themselves (while executing the MC management activities) in the day to day management of the project except where major decisions were required.

A Technical Project Leader, responsible for the day to day technical activities..

2.1.1.2 Project technological infrastructure

The technological infrastructure includes:

A project Web-site open to access by the general public

A mailing list server

A SVN server for code management

Project pages in social networks Twitter and Facebook.

The main ARES Web site is provided by Géant, and includes sections providing a general description of the project (<http://www.geant.net/opencall/Applications/Pages/ARES.aspx>), profiles of project partners, information on events involving ARES, on-line access to ARES documents, and contact information (in the Géant intranet, reserved page accessible from <https://intranet.geant.net/JRA0>). Another Web page dedicated to the project, for achieving a larger dissemination, has been developed and available at: <http://conan.diei.unipg.it/lab/index.php/research/ares>.

To facilitate the exchange of documents within the project, the ARES personnel made use of an internal document server. The document server has provided a document repository organized into “folders”. There are folders for each WP plus separate folders for Monthly Reports, Milestone documents, and for informal exchange of documents not related to a specific Work Package.

In addition to the document server, the ARES personnel makes use of a list server, allowing project participants to address email to groups of participants subscribing to specific lists. The software platforms used for all these activities are OSS.

2.1.2 ARES Management Objectives

The main goal of the ARES management was to provide the project with a light-weight, flexible management service capable of ensuring:

Objective 1: An intensive, flexible, open-intended dialog among the partners concerning key strategic and technical issues;

Objective 2: Rapid and effective decision-making on technical and organizational issues;

Objective 3: Full and timely compliance with contractual requirements.

2.1.3 ARES Management Achievements

The following tasks have been accomplished for achieving the objectives listed in the previous section. These tasks are illustrated in a way compliant with the WP1 description of the proposal, “Project Management”.

2.1.3.1 Task 1.1 Operational and financial coordination

This task allowed to setting the ARES project in motion. It has included the following activities, totally accomplished:

Setting up the Central Project Office. The administrative personnel of the Dipartimento di Ingegneria was made aware of all administrative issues related to the project,

Kick-off meeting. During this meeting, the administrative and reporting procedures, collected at the GN3plus Symposium in Vienna, October 2014, have been illustrated, along with the format for project documentation. In addition, the rules and procedures for scheduling future internal meetings have been defined.

Periodical meetings for the Management Committee.

Follow-up meetings with the project coordinator

the Management Committee has a continuous view of the project vision, by considering the technical results achieved and achievable. All technical issues have been faced and solved in the best feasible way. Currently, no open issues exist.

Even if individual partners are responsible of their financial statements, the coordinating partner has monitored their production by the deadline, and the compliance with the GN3+ format requirements.

The coordinating partner has promptly notified the Géant/GN3+ coordinating and administrative personnel about the change of LEAR and department name of the University of Perugia. The position of the new UoP LEAR has been officially established.

2.2 Project evaluation and risk assessment

This task consists with monitoring the project objectives in relation to the project milestones. All the document milestones have been produced indicated in the project proposal have been prepared by the deadline indicated. Some documents, to be delivered by the end of March, are in the phase of the final editing.

Milestone number	Milestone name	Work package involved	Expected date 1	Means of verification2
------------------	----------------	-----------------------	-----------------	------------------------

¹ Measured in months from the project start date (month 1).

Milestone number	Milestone name	Work package involved	Expected date 1	Means of verification ²
1	Set of system requirements completed.	WP2	Month 4	Internal Milestone report IM2.1 is available
2	System architecture and first release of algorithms and CDN procedures completed	WP2 - WP3	Month 6	Internal Milestone reports IM2.2 and IM3.1 are available
3	Reports on standardization, dissemination, and management	WP1 - WP6	Month 9	Internal Milestone reports IM1.1 and IM6.1 are available
4	CDN implemented together with first set of experiments	WP4 - WP5	Month 10	Internal Milestone reports IM4.1 and IM5.1 are available + HW/SW infrastructure running
5	Finalization of protocols and CDN management services.	WP3	Month 12	Internal Milestone report IM3.2 is available
6	Project testbed fully operational.	WP4	Month 16	Internal Milestone report IM4.2 is available + All ARES components

Milestone number	Milestone name	Work package involved	Expected date 1	Means of verification ²
				properly working
7	All ARES activities completed	WP1, WP5, and WP6	Month 18	Internal Milestone reports IM1.2, IM5.2, and IM6.2 are available + Final deliverable available + ARES CDN testbed finalized, operational, and assessed

Table 2.1: List of the project milestones successfully achieved

In addition to the set of milestone documents, this **final deliverable** illustrates all the project phases and achievements in details. The delivery deadline of this document is the end of Month 18, corresponding to March 2015.

The Management Committee has reviewed the overall progress of the project against the agreed project plan. No deviations with respect to the initial plans have been detected.

As for the possible risks identified in the project proposal, we comment all of them in Table 2.2 in the light of the actual project execution.

Risks indicated in the ARES proposal and relevant evaluation

Inadequate coordination: No problem have emerged by the project coordination.

Risks indicated in the ARES proposal and relevant evaluation
<p>Conflicts among the partners: No conflicts have emerged. In fact, the GGB partners have contributed through genomic pipelines, definition of specifications for genomic services, definition of genomic experiments, and their evaluation. UoP has contributed to all the technical activities related to networking. The interdisciplinary nature of the consortium has promoted collaboration and no conflicts have arisen.</p>
<p>Drop-out by a partner: No partner drop-out happened.</p>
<p>Loss of a partner team leader or a critical technician: No such events has happened.</p>
<p>Lack of resources for prototype implementation in the Géant network: The implementation has been successfully accomplished. Porting to the Géant network will be done as soon as the GTS needed resources will be made available. These resources were not made available during the project lifetime and, in accordance to the proposal, external resources have been used for implementation. Specific agreements with the relevant Géant personnel have been taken. In any case, since the ARES components have been implemented in full compliance with the Géant virtualization system, this porting is expected to be very easy and fast.</p>
<p>Difficulties in recruiting: No such difficulties have emerged.</p>
<p>Significant delays in implementing the ARES distributed environment: As mentioned above, the implementation of the ARES Components has proceeded successfully by using external networks. Porting to the Géant network will be done as soon as the GTS needed resources will be made available.</p>
<p>Problems due to a wrong selection of the middleware platform: The middleware virtualization platform selected is the one used in the Géant infrastructure. No issues for this choice have emerged during the project lifetime.</p>
<p>Residual latency problems after deep research activities: The technical activities of the project have addressed all latency problems and no additional latencies beyond expectations have emerged.</p>
<p>Security levels below requirements: No security problems have emerge. In particular, the use of publicly available data sets have eliminated any concerns about security issues.</p>
<p>Standards bodies not accepting ARES's work: The standard body being examined the ARES contribution is the IETF. A document has been submitted, and the final evaluation of its acceptance will happen in a time frame much longer than the project duration. We are confident on the goodness and the quality of our proposal.</p>
<p>It may happen that notwithstanding success in delivering the ARES framework, the project does not succeed in exploiting it: The real experiment deployed over real networks have demonstrated the feasibility of the ARES framework.</p>
<p>Failure to produce desired outcomes: All the expected outcomes have been produced.</p>

Table 2.2: Evaluation of risks.

2.3 Chapter Conclusions

This chapter is a report of the ARES management activities undertaken during the whole duration of the projects.

What emerges in this chapter is that the management support to ARES, made easy by the small size of the project and the presence of two partners only, has been suitable and no management issues has been observed. The Management Committee has continuously monitored the actual project progresses, and no deviations of the initial project plans have been observed.

All milestone documents and the final deliverable have been produced by the scheduled deadline.

In conclusion, all the project activities have been implemented according to the expectations indicated in the ARES proposal and all the management objectives mentioned in section 2.1.2 have been successfully achieved”.

3 Scenarios, Requirements, and Architecture

This chapter has the purpose of establishing the foundation of the ARES research and experiments through a detailed definition of entities, interfaces, information flow, functions, and requirements. The general objectives of ARES are:

To gain a deep understanding of network problems relating to a sustainable increase in the use of both genomes and relevant annotations for diagnostic and research purposes.

To allow extensive use of genomes through the collection of relevant information available on the network for the purpose of diagnosing diseases by means of gene sequencing.

To identify, at network level, suitable management policies of genomes data sets and annotations, in terms of efficiency, resiliency, scalability, and QoS in a distributed environment using a multi-user Content Delivery Network (CDN) approach and Network Functions Virtualization (NFV).

To make available the achieved results to extend the service portfolio of the GÉANT network.

We aim to address these challenges through the definition of the service requirements for genomic data exchange, deployment of a suitable architectural framework, for modular and virtualized services, able to provide suitable insights for the service operation and relevant perception, deployment of mechanisms and protocols for service description, discovery, distribution, and composition.

Experimental activities address performance evaluation, in terms of QoE, scalability, total resource utilization, signaling load, performance improvement and coexistence with legacy solutions, dynamic deployment and hot re-configurability.

For these purposes, it was necessary to plan experiments in detail with a clear and detailed view of the hardware and software requirements, information exchange protocols both in the user and control planes, and network and service management functions.

The objective of this document is indeed to provide such information available at the current stage on the activities.

3.1 Definition of scenarios and requirements

3.1.1 General requirements

Figure 3.1 shows the highest level of the ARES system, used as root element for detailing requirements. It includes a user, representing the medical/biologist user of the ARES system. The user interacts with the medical tool, used to access ARES services, through the user interface (ui). The medical tool is a web interface designed with the goal of simplifying the interaction between the researcher and the overall processing system, masking as much as possible all computation machinery which is the core of the project. Exchanges through the ui interface consists of:

- a. Service requests (i.e.type of bioinformatics analysis required for diagnostic needs).
- b. Service data.
- c. Results of processing.

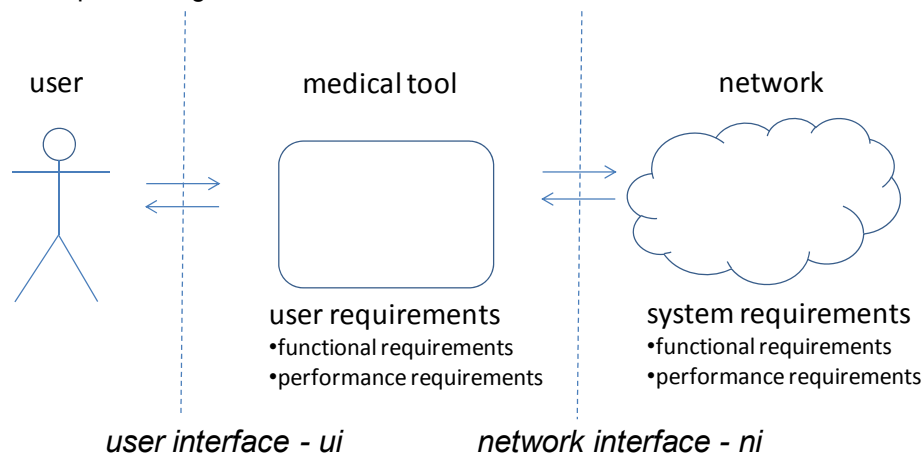


Figure 3.1: ARES general requirements

Information exchanged through the ui-interface is reported in Table 3.1 together with the corresponding type of data exchanged at the network interface (ni). In more detail, in each row, the Table reports the type of information exchanged between the user and the medical tool, the type of request together with a brief description, including parameters of the information exchange, and the corresponding exchange triggers on the ni interface.

Category of ui-exchange	Type	Category/Descriptors	Corresponding category of ni-exchange
Service requests	Bioinformatics analysis of the genome	<ul style="list-style-type: none"> - Guaranteed time (class #1, ...class #n) - Minimum cost 	- Genomic pipeline indication
Service data	Reference to patient genomic data	<ul style="list-style-type: none"> - Size - Quality level (resulting from sequencing files) - Mnemonic indicator 	Data transfer
Visualization of the status of the processing	Link to processing results	Reliability level (H,2,3,...,n,Q0)	Returned data
	Service progress indication	Percentage of expected computing time	
	Service rejection	Error code	

Table 3.1: Information exchanged through the ui-interface and the corresponding data exchanged at the ni-interface

The obtained message exchange, from the user point of view, can be roughly summarized by the Figure 3.2. The Private Genome database, as explained later in this document, contains the genomic files of the patient to be processed.

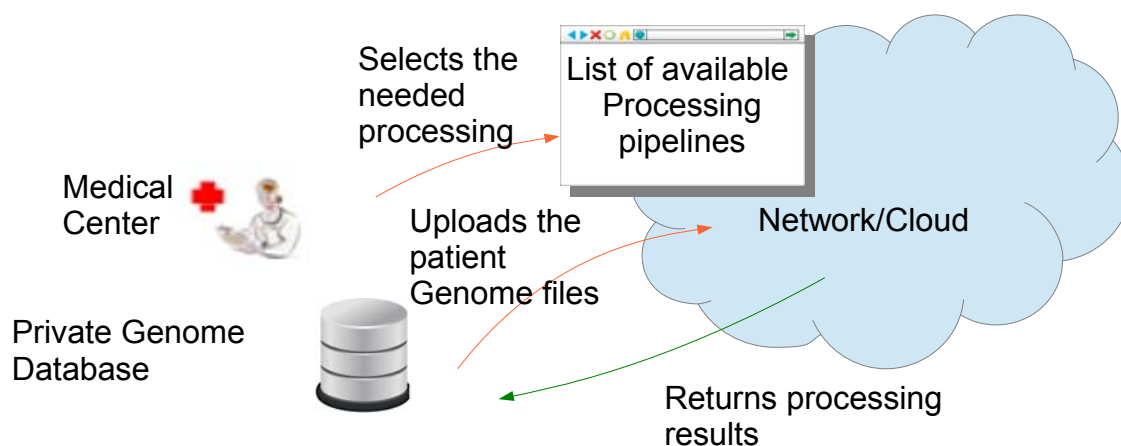


Figure 3.2: Sketch of the interactions happening on interface ui.

The service schedule time is the time interval during which service requests are used to deliver requests to the network (see Figure 3.3). ARES provides different schedule time for each service request based on the optimal analysis time indicated in Table 3.2.

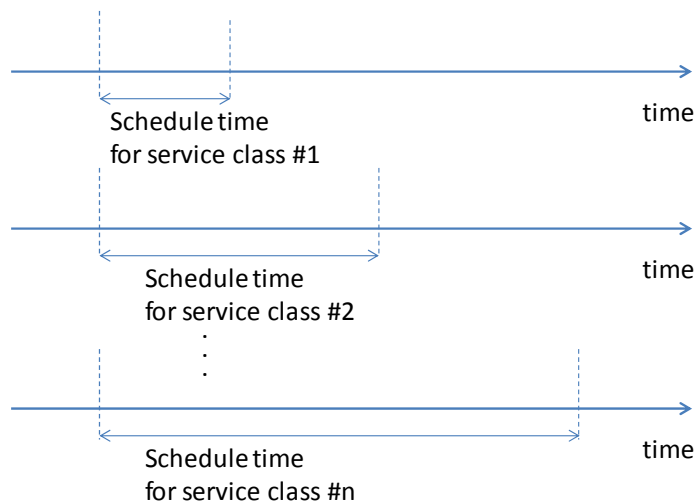


Figure 3.3: Sketch of the ARES differentiated schedule period

Differential Expression RNA-Seq			
Clinical Phenotype	Target	Reference	Optimal analysis time (in days)
T-Cell Acute Lymphoblastic Leukemia	Classify patients into T-ALL subtypes	[7]	4
Renal cell carcinoma	Classify patients into Renal Cell Carcinoma subtypes	[8]	7
Chronic lymphocytic leukemia	Classify patients into Chronic lymphocytic leukemia subtypes	[9]	7
Breast Cancer	Predicting outcome of HER2-positive breast tumors	[10]	7
CNV detection by WGS			
Lung Cancer	Circulating tumor cells-based cancer diagnostics	[11]	7
Pregnancy	Fetal abnormalities	[12]	4

Differential Expression RNA-Seq			
Clinical Phenotype	Target	Reference	Optimal analysis time (in days)
Retinal dystrophies	Genetic classification	[13]	10
Lung Cancer	Drug treatment resistance	[14]	7
Neurodevelopmental disabilities	Genetic classification	[15]	15

Table 3.2: Medical Requirements

Exchanges through the network interface (ni-interface) consist of the exchange of different types of data between the medical tool and the network. In turn, inside the network data can be exchanged with either the service orchestration point, labelled as Genomic CDN Manager (GCM), which is the software entity taking decisions about the where executing service requests, or the data centre in the point of presence (PoP) selected to perform the computation by the GCM. These data are:

- a. Genomic pipeline indication (to GCM).
- b. Data transfer(to PoP).
- c. Returned data (from PoP).

Data exchanges through the ni-interface are detailed in Table 3.3.

ni-data exchange category	Type	Category/Descriptors	Corresponding ui-information exchange category
Indication of the bioinformatics pipeline for genome data set processing	Bioinformatics analysis selected	- Guaranteed time (class #1, ...class #n) - Minimum cost	Service requests
Data transfer	Patient genomic data	- Size - Quality level (resulting from sequencing operation)	Service data
URL transfer	Patient genomic data	- Size	Link to service data

ni-data exchange category	Type	Category/Descriptors	Corresponding ui-information exchange category
	reference	- Quality level (resulting from sequencing operation)	
Returned data	Results	Reliability level(H,2,3,...,n,Q0)	Visualization of the status of the processing
	Service progress indication	Percentage of expected computing time	
	Service rejection	Error code	

Table 3.3: report of information exchange through the ni-interface

The resulting message exchange, from the medical tool point of view, can be roughly summarized by the Figure 3.4.

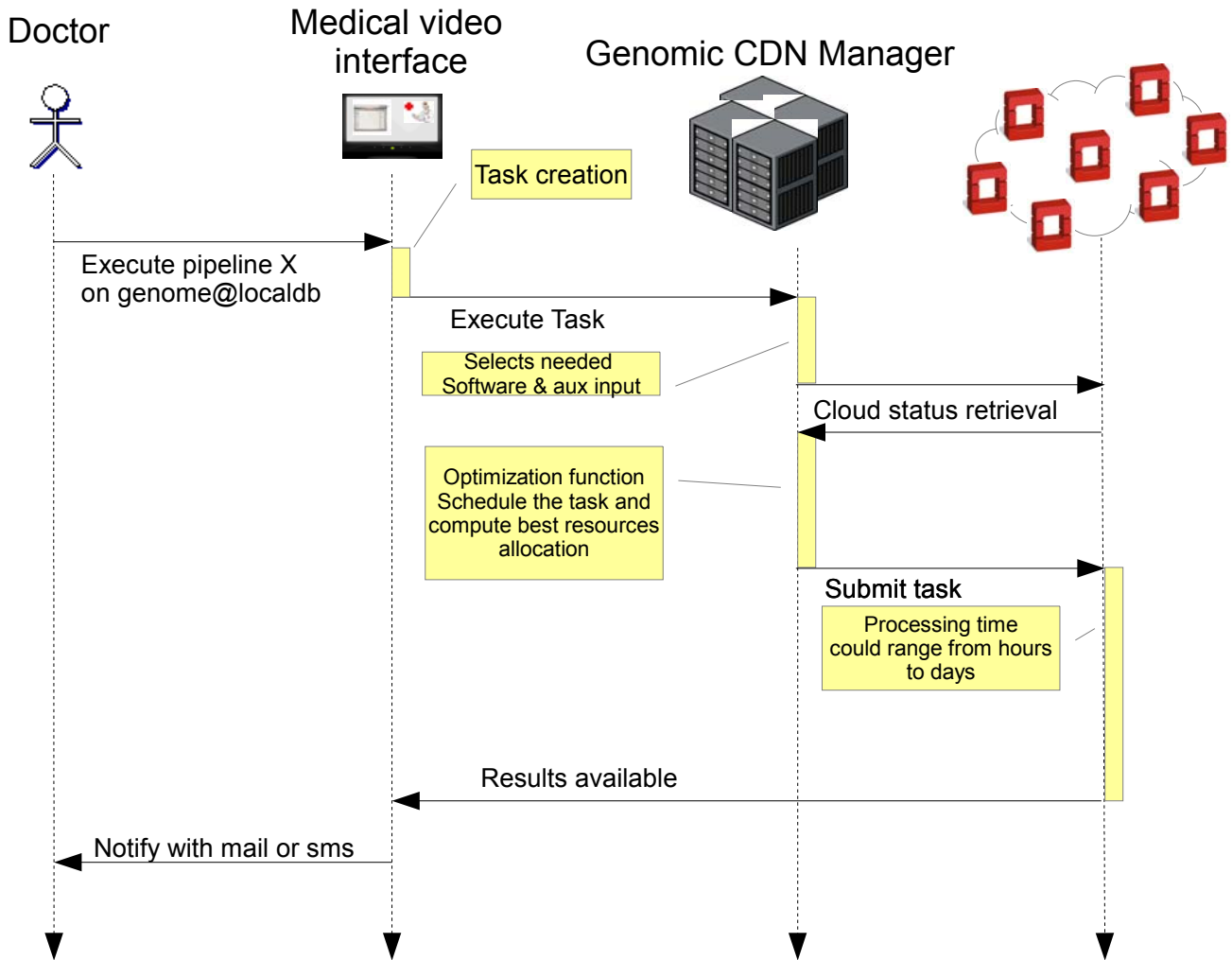


Figure 3.4: Sketch of the interactions happening on interface ni.

3.2 Genomic Pipelines

Experiments in ARES made use of two genomic pipelines, named Differential Expression (DE) and Copy Number Variation (CNV). These pipelines have been deployed in two different configurations, thus resulting in four different case studies that make a different use of network and computing resources.

3.2.1 Copy Number Variation (CNV) pipeline

Figure 3.5 shows the flow diagram of the CNV pipeline, along with the name of its software components. The bioinformatics processing functions implemented are beyond the scope of this document. For what concerns

the ARES experiments, the resource requirements to be deployed, in two different configurations, are reported in Table 3.4. Further details on the actual configuration of the pipeline for the final ARES experiments are given in Section 6.2.1.

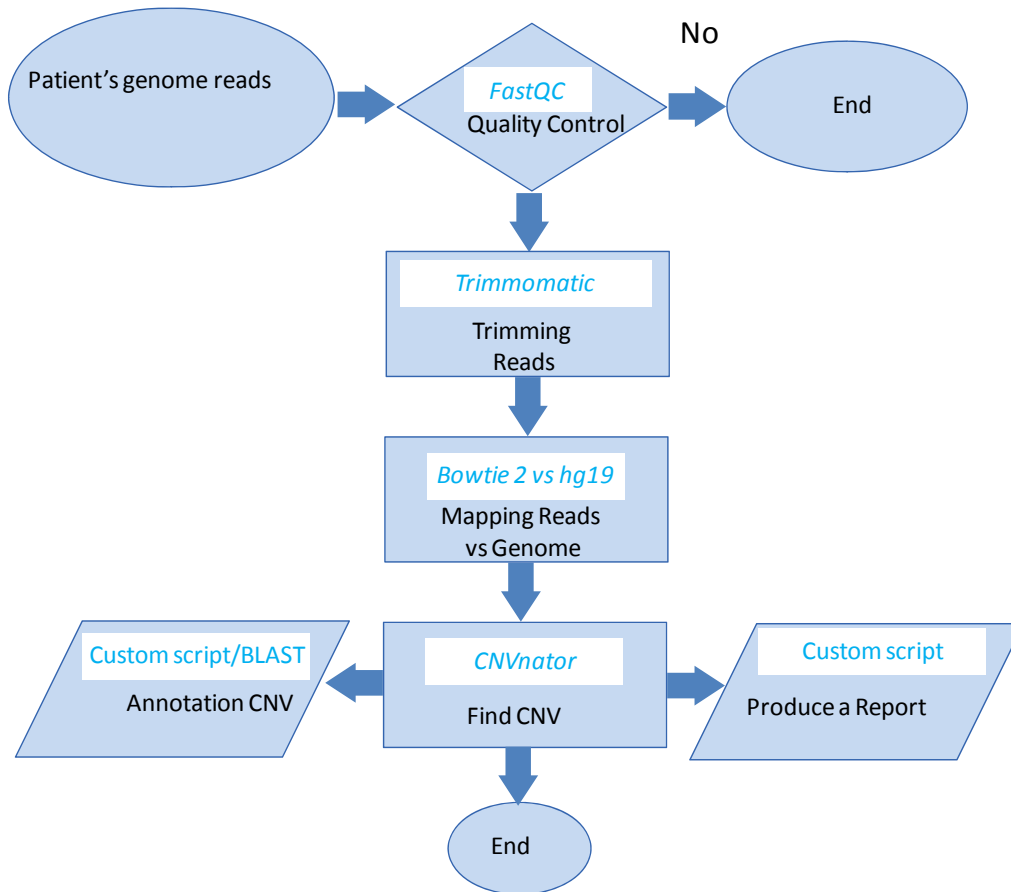


Figure 3.5: Copy Number Variation (CNV) Pipeline

3.2.2 Differential Expression (DE) pipeline

Figure 3.6 shows the flow diagram of the DE pipeline, along with the name of its software components. The bioinformatics processing functions implemented are beyond the scope of this document. For what concerns the ARES experiments, the resource requirements to be deployed, in two different configurations, are reported in Table 3.4. Further details on the actual configuration of the pipeline for the final ARES experiments are given in Section 6.2.1.

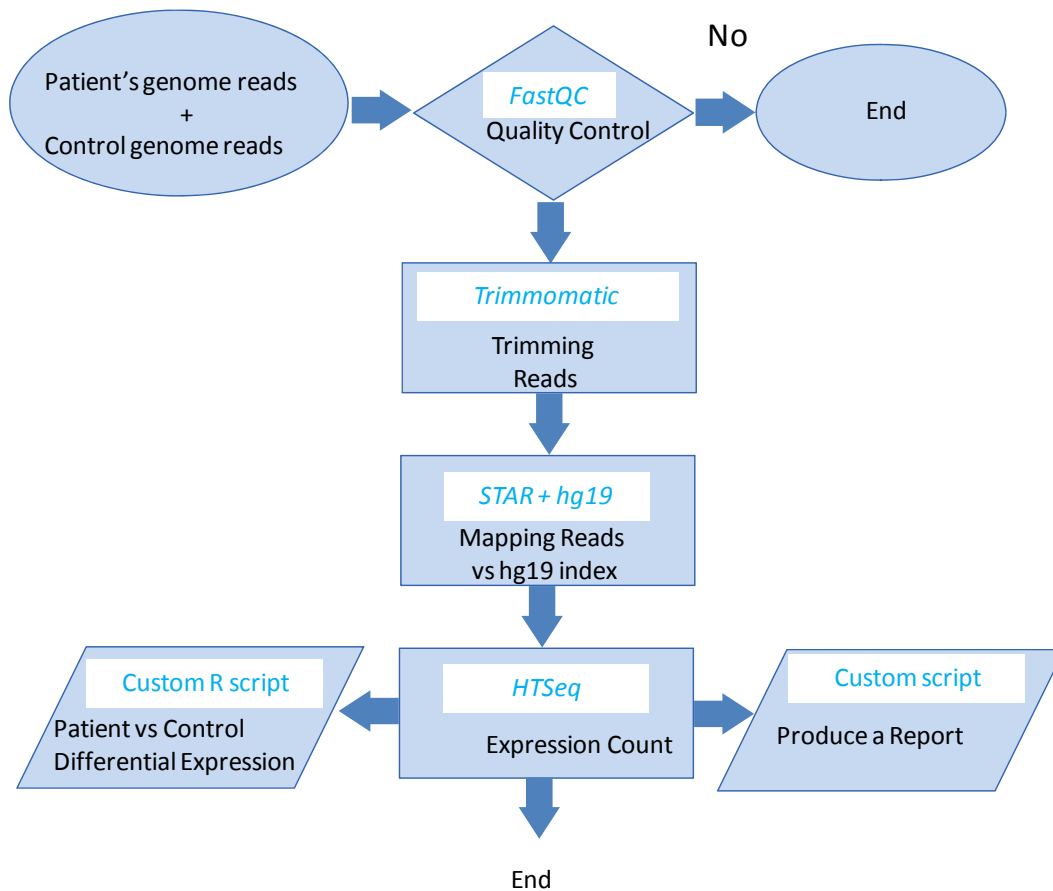


Figure 3.6: Differential Expression (DE) Pipeline

3.2.3 Computing requirements

The VM size and the auxiliary files size (e.g. the genome indexes) reported in Table 3.4 represent the total size of the files to be uploaded into the PoP selected for the computation, in order to execute the relevant genomic pipeline. The RAM size and the number of CPU cores reported in Table 3.4 are the minimum amount of RAM and the minimum number of CPU cores, respectively, necessary to execute the relevant genomic pipeline. As for the disk size allocated to the VM during the execution of the pipeline, this has been estimated with a specific input file. These values have been obtained by performing extensive computation with different amounts of RAM and number of CPU cores allocated to the VM, which represent reference values to be used for associating different service levels with RAM size and allocated number of CPU cores.

Figure 3.7 and Figure 3.8 show the experimental processing time obtained by using a different number of virtual cores and a different amount of RAM space to the CNV and DE pipelines, respectively. We can observe that, by increasing the RAM size and the number of processor cores, it is possible to reduce the execution time of the genomic pipeline. However, the performance improvement associated with the increase of the memory

size allocated to the VM is often negligible. Only in Figure 3.7 the curve with 4 GB of RAM is slightly slower with respect all the other ones, due to the different way the computation is performed to cope with a limited size of available memory, as explained in Table 3.4, and only when the number of allocated CPU cores is equal to 2. Instead, the variability associated with a different amount of CPU cores is more significant. Said this, by observing Figure 3.7 it is quite evident that the gain in allocating more than 4 CPU cores is almost negligible and thus not recommended, and, in any case, doubling the number of CPU cores (from 2 to 4), we can save "only" the 27% of computing time, which is about the half of what expected. Thus, this has to be done only when strictly necessary. Thus, a final, general comment is that the processing time of the CNV pipeline is dependent of the number of CPU cores used, whilst it appears to be quite insensitive to the amount of allocated RAM size beyond 8GB. It is possible to execute the CNV pipeline by using 4GB of RAM, but the achievable performance results to be affected of this choice.

Pipeline	Configuration	Hypervisor	VM image size	RAM size min	# CPU cores min	VM storage size ³	Auxiliary files size
CNV	BOWTIE aligner (the computing is performed on the whole human genome)	KVM	3.1GB	8 GB	1	50 GB	3.5 GB
CNV	BOWTIE aligner (the computing is performed chromosome by chromosome)	KVM	3.1 GB	4 GB	1	50 GB	3.5 GB
DE	BOWTIE aligner	KVM	3.1 GB	4 GB	1	80 GB	3.5 GB
DE	STAR aligner	KVM	3.1 GB	32 GB	1	100 GB	26 GB

Table 3.4: Configuration and minimum resource requests of VMs implementing the genomic pipelines.

³The requirement in terms of storage size allocated to the VM executing the processing pipeline (in GB) has been estimated by using as inputs two compressed files of 1.2 GB each. It takes into account all intermediate outputs of the pipeline, and leaves enough spare disk space to avoid problems in the operating system.

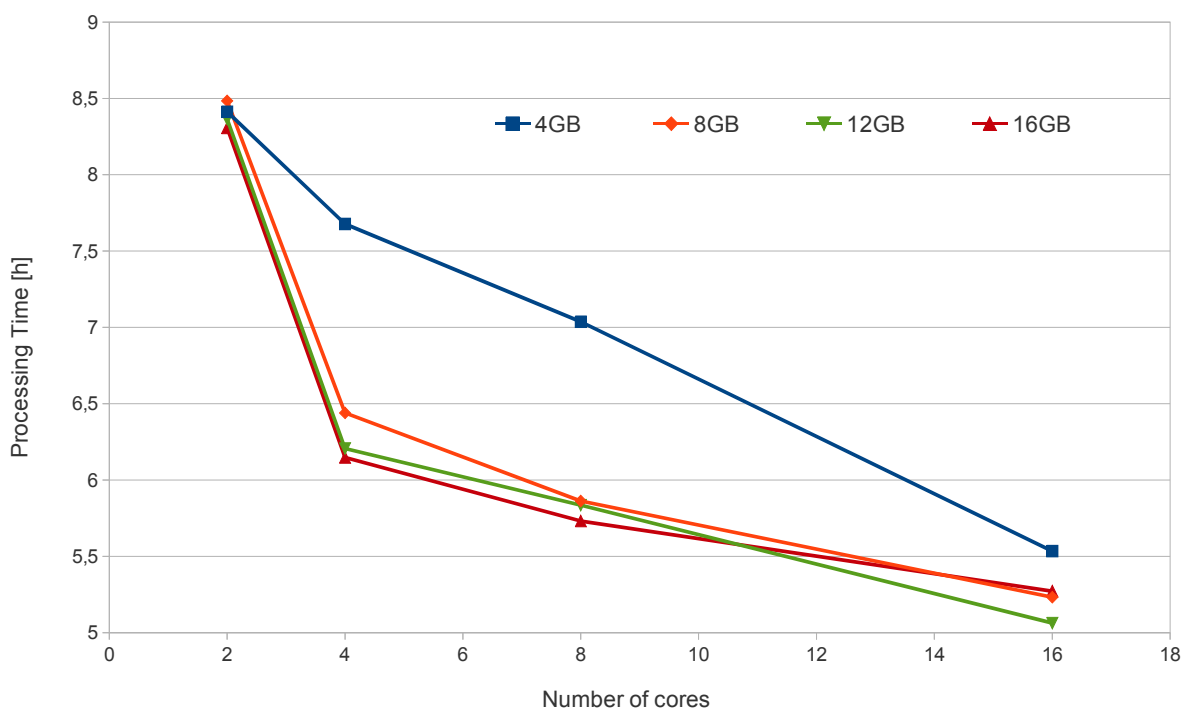


Figure 3.7: Processing time of the CNV pipeline vs. number of allocated virtual cores, for different values of the allocated RAM size.

The situation is similar in Figure 3.8: there is not a clear, significant advantage to allocate more than the minimum computing resources (reported in Table 3.4) to the VM performing a DE pipeline. In fact, the performance of the DE pipeline shows a substantial independence of the RAM size when the STAR aligner [18] is used instead of the Bowtie 2 aligner [17]. The sequence alignment is a bioinformatics function that arranges the sequences of DNA, RNA, or protein to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between the sequences [16]. This functions immediately follows the trimming one in pipelines (see the pipelines flow diagrams in Figure 3.5 and Figure 3.6). Depending on the type of aligner used, the requirements in terms of computing resource can vary a lot. This aligner requires a minimum amount of 32GB RAM, and any further increase of this large value is useless. In addition, a very large amount of RAM could give some problem due to the dynamic memory management and garbage collection of the Java components of the pipeline. Also for the allocation of CPU cores, multiplying by 8 the their number, the gain is around 20%, a result which does not definitely recommend allocating more that 2 CPU cores.

A final note is relevant to the hypervisor choice. Even if we have used KVM, since it is open source and widely used both in the research community and in operational data centres, especially for its easy integration with cloud management systems like OpenStack, the obtained results do not depend on the specific hypervisor used, which could be also Xen, Hyper-V, or VMWare ESXi.

These simple considerations make evident the possibility of making different trade-offs when the two pipelines are used. The traded resources essentially consist of the RAM size and CPU cores, which may have an impact on cache deployment and management, essentially on the download time, number of parallel-executed pipelines and their processing time. Clearly even the processing time may be part of a trade-off in order to deploy differentiated service categories.

The results shown will be used in operation to adapt the performance requirement to the seriousness of the medical scenario being executed.

The results shown so far are not representative of the wide scenarios that can be found in operation. Even during the execution of the activities of the ARES project, different genomic components have been introduced, according to considerations deriving from both the ICT and medical areas. Thus, even if what was done so far is well representative of the ARES experimental activities, a continuous monitoring of the suitability of individual functions and requirement has been done.

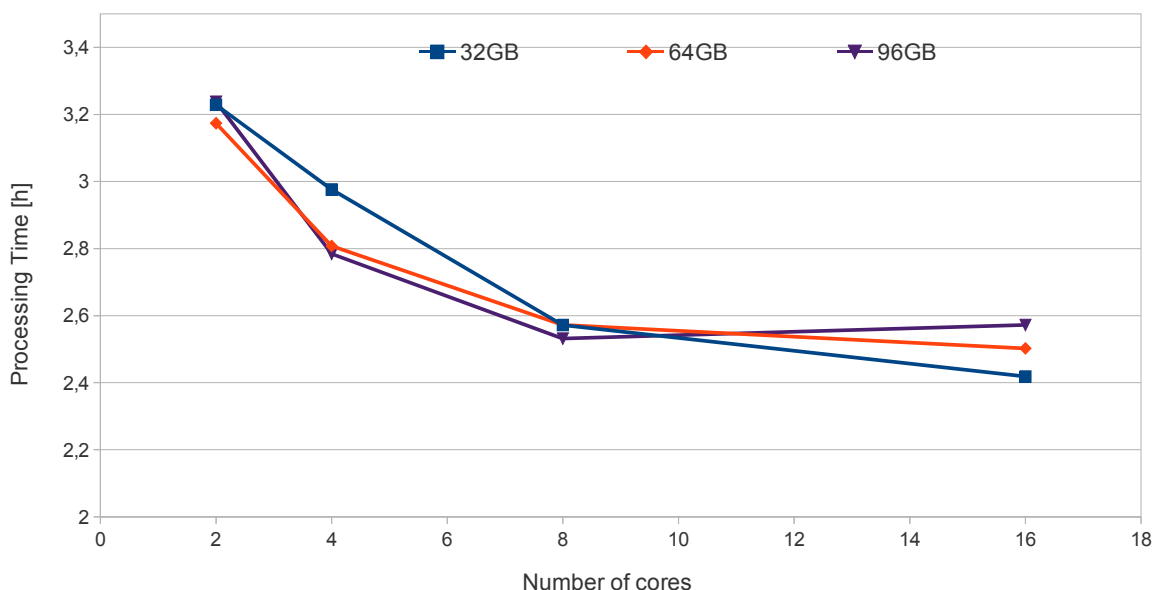


Figure 3.8: Processing time of the DE pipeline vs. number of allocated virtual cores, for different values of the allocated RAM size. Use of the STAR aligner [18] instead of Bowtie2 [17] in the DE pipeline (Figure 3.6).

We stress that the processing results shown in this chapter are preliminary, and are shown to provide the interested reader some glue with the expected genomic processing time. A very detailed assessment of the genomic pipelines in different operative configurations is illustrated in Chapter 6.

3.3 System architecture and technologies

3.3.1 Network Architecture and Components

This section illustrates the elements interconnected in the ARES experimental architecture and their configuration in their deployment and operating conditions. Figure 3.9 sketches these components. Some of these components are individual functional entities, directly mapped on the relevant physical entities. Others are different functional entities, which are deployed together within a single physical entity, as detailed in what follows. The specific technologies selected to implement the system are sketched in this Section and described in more detail in Section 3.3.2.

Local DB: Local database including the patient's genomic files. They are be transferred to the processing elements, but not cached in the network due to privacy issues. For the same reason, when a final reply is sent back to the requesting user, being it either the processing results or a service rejection notification, the patient's genomic files are removed from the network entities where they were transferred. This is both a functional and a physical entity.

Repository: this is the location where genomic VMs and their supporting files (which are the human genomic index and the hg-19 human genome reference model for the considered genomic pipelines), are originally stored. From these repositories, VMs and supporting files are transferred to the PoPs selected for executing computation. The PoPs staying on the path of the transmitted files are able to cache these files by running NetServ [1] CDN cache modules for fulfilling further requests without transferring them from the original repositories.

GCM: Genomic CDN manager, having the tasks of receiving service requests from service users, orchestrating the overall network operations, and running an optimization process whose output is the selection of the PoP where the genomic processing will take place. This entity can be either executed in a dedicated physical machine, or as one of the service modules (NetServ bundles, see Section 3.3.2) running in a server of a PoP, whose architecture is illustrated below.

PoP: Point of Presence, indicating an aggregate of networking, computing, and storage resources (i.e. a data centre connected to a backbone router of the considered network). A simplified picture of the architecture of a PoP is sketched in Figure 3.10 (see also [19]). In order to preserve the readability of the figure, we have not shown any form of redundancy, in terms of both duplicated routers and redundant connections. For the same reason, also layer 2 details are omitted. A PoP is composed of one or more backbone (BB) routers, which are connected to the BBs of the neighbouring PoP. In addition, these BBs provide connectivity services to a number of access routers (ARs), which, in turn, provide connectivity to both other networks (in the case of

Géant, to the NRENs) and to the data centre hosting the computing resources deployed in the PoP. For each service request, a PoP is selected by the GCM as the result of an optimization process to host the genomic processing. PoP resources are abstracted through cloud management systems interfaces. We have selected OpenStack [5] as cloud management system, since it is open source and widely adopted in operational deployments. This way, PoPs make available resources for implementing CDN *caches* and *genomic computing nodes*. Both are implemented in virtualized frameworks: genomic computing nodes are VMs, whereas CDN cache modules are implemented as NetServ services, running either in VMs or routers. Also data centre management functions have been implemented through the NetServ framework, described in some details in section 3.3.2.2. In more detail, the functional entities deployed in each PoP are:

The genomic processing modules. Each of these modules is able to perform a specific processing on genomics data [20], and is implemented through a specific pipeline. Each processing pipeline, as illustrated by the two case studies presented in section 3.2, consists of several software tools which operate in cascade. Since the goal of this project is to allow the personnel which already uses these software tools to simplify their usage, the only viable solution is to run them in virtual machines, as may already happen in the data centres of their institutions. In this way, the overall functioning of the ARES machinery is hidden to the final users. We label these processing modules as genome processing virtual machines (**GPVMs**).

The cache module. Its function is to cache contents so as to serve them upon future requests. These contents can be both GPVM images, or auxiliary files needed to perform genomic computations (e.g. see Table 3.4). This type of entity can be deployed both on routers and on servers. In the ARES proposal, this functional entity is implemented as a NetServ bundle (see section 3.3.2.2 for more details). For this reason, we label it as cache bundle (**CB**). Caching can be done both in memory and on disk, depending of the size of the content. A CB can also act as a proxy for a Repository storing auxiliary files or GPVM images.

The OpenStack interface module. This functional entity is in charge to interact with the OpenStack deployment in a PoP. Since also this module has been implemented as a NetServ bundle, we label it as local OpenStack interface bundle (**LOIB**). The LOIB has multiple functions.

The first function is to interact with the GCM during the discovery phase in order to provide information about the amount of resources available in the local PoP for processing genomes. In fact, the only entity in the ARES system able to interact with OpenStack through the OpenStack APIs is the LOIB.

The second function is to drive the execution of the GPVM, by interacting with the GCM. Thus, it receive the command from the GCM to start the computation, and trigger all the machinery needed to start the VM with the required resources, among those available in the locally defined OpenStack flavours. Optionally, the LOIB can also drive the resize of the resources allocated to a GPVM, once the processing phase involving the alignment (see e.g. Figure 3.5 and Figure 3.6), which is the more demanding in terms of resources, is ended.

The third function is to interact with the CB. Since the OpenStack platform does not have a client able to retrieve a VM image from a remote URL, it is necessary that an external entity (the LOIB in this case) retrieves the VM image from a remote or local (the CB in this case) location and injects it in the OpenStack storage module (see section 3.3.2.3 for details).

The fourth function is to interact with the GPVM. Once the GPVM is turned on, the LOIB passes to the GPVM the URLs of the auxiliary files to be downloaded in order to carry out the computation, as well as the URL of the patients' genomes, previously communicated.

It is worth noting that the LOIB provides a number of functions which are much more general than those relevant to the specific ARES scenario (genomic processing). Thus, a possible option is to split the LOIB entity in two entities. The first is specific to the ARES scenario (ARES application bundle, **AAB**), and takes care of all aspects dealing with genomics processing (thus just the fourth function listed above). The second is an underlying entity (the real LOIB), which mediates the access of the AAB with OpenStack. In this way, it is possible to easily reuse the overall ARES system for a completely different CDN and processing service, to be carried out in the Géant network, by designing another application bundle specific for another application scenario.

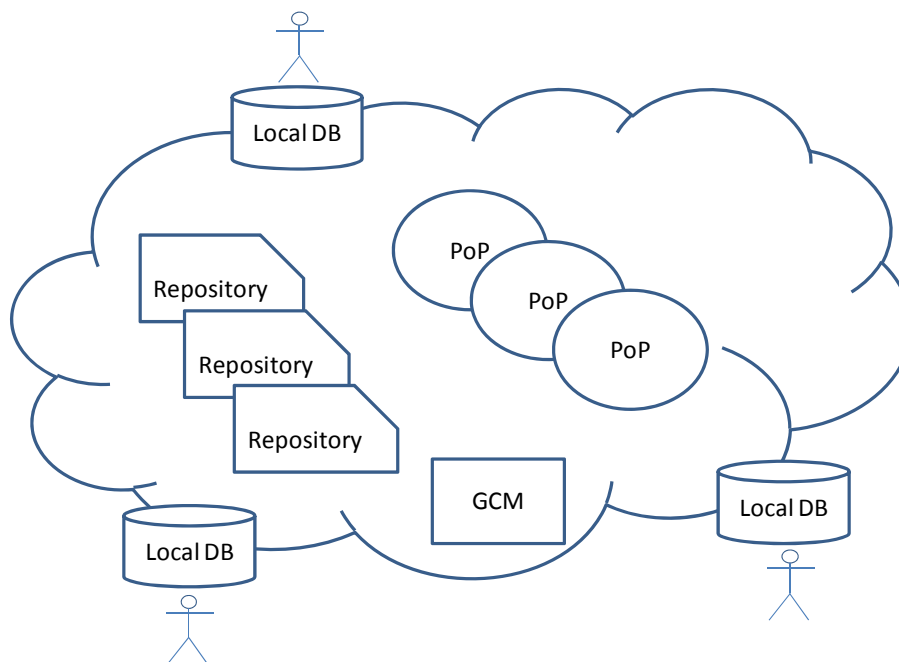


Figure 3.9: Architectural components of the ARES experimental infrastructure.

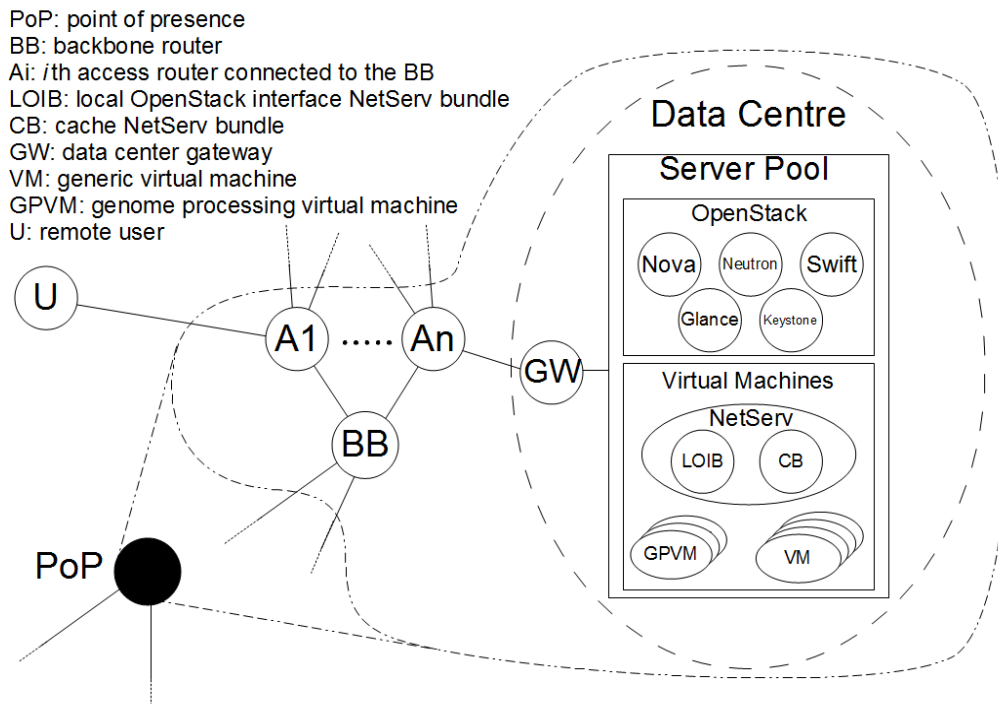


Figure 3.10: PoP architecture.

The detailed interaction of the entities shown in Figure 3.9 is shown in Figure 3.11. The access to the ARES medical service happens through a web interface. Resource abstraction in PoPs is achieved by the use of OpenStack [5]. The detailed explanation of each individual message is beyond the scope of this document, and it provided in the IM3.1 report. What is needed is the abstraction of the functions and requirements needed to support the exchange shown in Figure 3.11.

Below, we summarize the main interactions among the defined functional entities:

Interaction Medical tool – GCM: it allows the users to ask for a service to the system. It also allows the user to be notified about the completion of service request, and to visualize processing results.

Interaction Medical tool – Local DB: it allows the users to specify the genomics file to be analysed during the requested processing service.

Interaction Medical tool – GPVM: it allows to send the processing output from the GPVM to the Medical tool. An alternative solution is to send to the Medical tool just the URL from which it is possible to download the processing result. In this case, the process may be mediated by the LOIB/GCM.

Interaction GCM – LOIB: it allows:

Retrieving the information about the available resources in the PoP, among those assigned to the ARES system (see discussion in section 3.3.2.3).

Providing the LOIB with the URL (of a CB or of a Repository) from which to download the GPVM image to be used.

Starting the computation of the genomics processing service and check the status of the current computation.

Providing the VM with all the files needed to perform the computation, by means of the communications of the URLs from which to download them to the LOIB.

Interaction GCM – CB: it allows:

Finding out which cache has the required content.

Deploying a new CB on a NetServ node, acting as a proxy for a Repository.

Interaction GCM – Repository: it allows understanding if a repository has a specific content.

Interaction LOIB – OpenStack: it allows:

Retrieving the amount of resources allocated to the ARES system in a PoP are currently available for performing a new processing task.

Injecting a VM image (GPVM image) into the OpenStack image storage module.

Starting/stopping a GPVM.

(Optional) Resizing the current resources allocated to a GPVM, by selecting a different flavour.

Interaction LOIB – CB: it allows temporary retrieving a GPVM image to be injected into an OpenStack storage module.

Interaction LOIB – GPVM: it allows:

Communicating the URLs of all the files (including auxiliary files and patients genomes) to be processed.

Checking the status of the current computation.

Communicating the location where uploading the processing output.

Interaction GPVM – Local DB: it allows the GPVM to retrieve the genomics files to be processed.

Interaction GPVM – CB: it allows the GPVM to download one or more auxiliary file from a CB. If the file is still not present in the CB, this interaction causes the CB to retrieve it and then serve to the GPVM.

Interaction GPVM – Repository: it allows the GPVM to download one or more auxiliary file from a Repository, without using the intermediate caching functions of CBs. This can also trigger the installation of a number of CB on the downstream path between the Repository and the GPVM. The consequent action is that the Repository will redirect the GPVM to download those file from a CB.

Interaction CB – CB: it allows a CB to retrieve a file from another CB, in chained download process.

Interaction Repository – CB: it allows a Repository NetServ-enabled to

Installing a CB bundle on a NetServ installation.

Instructing a CB on the parent CB from which a specific content has to be retrieved.

Fill a CB with a specific content.

If the Repository is not NetServ-enabled, the first two functions are provided by a CB acting as a proxy for the Repository. Such a CB can be installed by the GCM.

Table 3.5 reports these functions and requirements. It is very easy to map these requirements with the exchange shown in Figure 3.11 and with the interactions among functional entities reported above. For each requirement, Table 3.5 reports its description, the involved entities, the involved WPs and Tasks of the ARES proposal, and general comments.

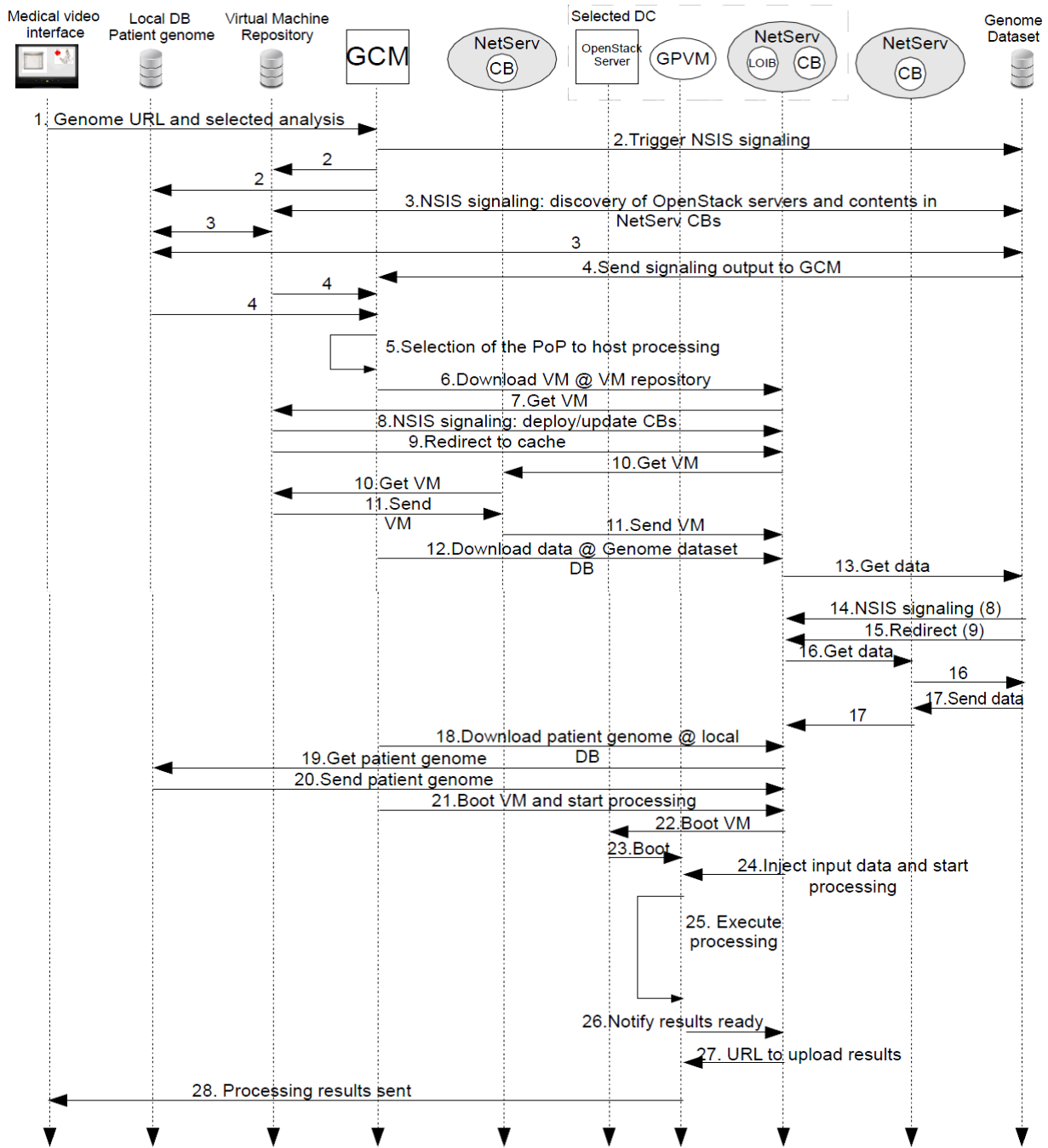


Figure 3.11: ARES Cloud management.

Requirement ID	Function description	Involved entities	WP/Task	Comments
GENERAL REQUIREMENTS				
GREQ1	Execute a given pipeline on the file genome@localdb	Medical Tool	WP2/Task 2.1 WP4/Task 4.1	Simple implementation. Marginal impact on experiments.
GREQ2	Request of task execution	Medical Tool GCM	WP3/Task 3.1 WP4/Task 4.1	Simple implementation. Marginal impact on experiments (message 1 in Figure 3.11)
GREQ3	Selection of neededsoftware & auxiliary input	GCM	WP2/Task 2.2 WP3/Task 3.1 WP4/Task 4.1, 4.2	Simple implementation. Significant impact on subsequent phases.
GREQ4	Cloud status retrieval	GCM PoPs	WP3/Task 3.1, 3.2 WP4/Task 4.1	Advanced NSIS signaling [2]. Significant impact on experiments (messages 2, 3, and 4 in Figure 3.11)
GREQ5	Execution of the optimization function to select the most suitable PoP to host pipeline	GCM	WP2/Task 2.1 WP 4/Task 4.2	Schedule the task and compute best resource allocation. Significant impact on experiments (phase 5 in Figure 3.11)
GREQ6	Task Submission	GCM PoPs	WP 2/Task 2.2 WP 3/Task 3.2 WP 4/Task 4.2	Simple implementation. Marginal impact on experiments (messages 6, 12,

Requirement ID	Function description	Involved entities	WP/Task	Comments
				18, and 21 in Figure 3.11)
GREQ7	Return of processing results	PoPs Medical Tool	WP 3/Task 3.2 WP 4/Task 4.2	Final step of experiments (message 28 in Figure 3.11)
SIGNALLING REQUIREMENTS				
SREQ1	Session Setup	Entities on path	WP 3/Task 3.1	NSIS signaling [2] (sub-function used by on-path and off-path signaling)
SREQ2	Probing and path construction	Entities on path and off-path close to on path	WP 3/Task 3.1	NSIS signaling (messages 8 and 14 in Figure 3.11)
SREQ3	Content Search	Entities on path and off-path close to on path	WP 3/Task 3.1 WP 4/Task 4.1	Off-path advanced NSIS signaling (message 3 in Figure 3.11)
SREQ4	Resource Search	Entities on path and off-path close to on path	WP 3/Task 3.1 WP 4/Task 4.1	Off-path advanced NSIS signaling (message 3 in Figure 3.11)
SREQ5	Trigger remote bubble signaling	Entities off-path close to receiver	WP 3/Task 3.1	Off-path advanced NSIS signaling (to instantiate a NetServ node acting as proxy for an external repository, not show in Figure 3.11)
SREQ6	End-to-end signaling	End-to-end entities	WP 3/Task 3.1	HTTP/NSIS signaling (messages 1, 4, 6, 9, 12, 15, 18, 21, 22, 24, 26, 27, and

Requirement ID	Function description	Involved entities	WP/Task	Comments
				28 in Figure 3.11)
SREQ7	Trigger signaling	End-to-end entities	WP 3/Task 3.1	Off-path advanced NSIS signaling (message 2 in Figure 3.11)
APPLICATION REQUIREMENTS				
AREQ1	Redirection	End to end entities	WP 3/Task 3.2	Service management function (messages 9 and 15 in Figure 3.11)
AREQ2	Download Command	Selected PoP VM Repository Pre-cached PoPs	WP 3/Task 3.2	Network and application specific service function (message 6 in Figure 3.11)
AREQ3	Get VM Request	Selected PoP VM Repository Pre-cached PoPs	WP 3/Task 3.2	Network and application specific service function (messages 7 and 10 in Figure 3.11)
AREQ4	Datatransfer phase	Selected PoP Repositories Local DB Patient Genome Pre-cached PoPs (when applies)	WP 3/Task 3.2	Network and application specific service function (messages 11, 17, 20, and 24 in Figure 3.11)
AREQ5	Download data @ genome dataset DB command	Selected PoP GCM	WP 3/Task 3.2	Network and application specific service function (message 12 in Figure 3.11)
AREQ6	Get Genomic Data	Selected PoP	WP 3/Task 3.2	Network and

Requirement ID	Function description	Involved entities	WP/Task	Comments
	command	Genome Dataset Repository Local DB Patient Genome Pre-cached PoPs (when applies)		application specific service function (message 13, 16, and 19 in Figure 3.11)
AREQ7	Download patient genome @ local DB	Selected PoP GCM	WP 3/Task 3.2	Network and application specific service function (message 18 in Figure 3.11)

Table 3.5: Requirements and functions of the ARES experimental platform.

3.3.2 Selected Technologies

The main technologies selected to implement the ARES components have been identified at the time of the project proposal, and have been confirmed after a careful analysis of the service scenarios and of the requirements extracted from their analysis. These technologies are:

- NSIS [2] for the signaling,
- NetServ for service modularization and virtualization,
- OpenStack for cloud service management.

All of them are open source and freely available. The obtained view point that is obtained is depicted in Figure 3.12. From the point of view of the final user, the service offered by the framework designed in ARES appears like a Software as a Service (SaaS) cloud service. From the point of view of the Géant service administrator, the designed system allows to add to the Géant service portfolio a dynamic Content Delivery Network (CDN), built over an overlay through the usage of NetServ modules. Finally, from the point of view of the Géant personnel managing computing resources in PoPs, the offered service is simply an Infrastructure as a Service (IaaS) cloud service.

The optimization algorithms, implemented as NetServ modules as well, interact with all the other system components by the NSIS signaling [2], which acts as the glue which maintain functioning all these pieces.

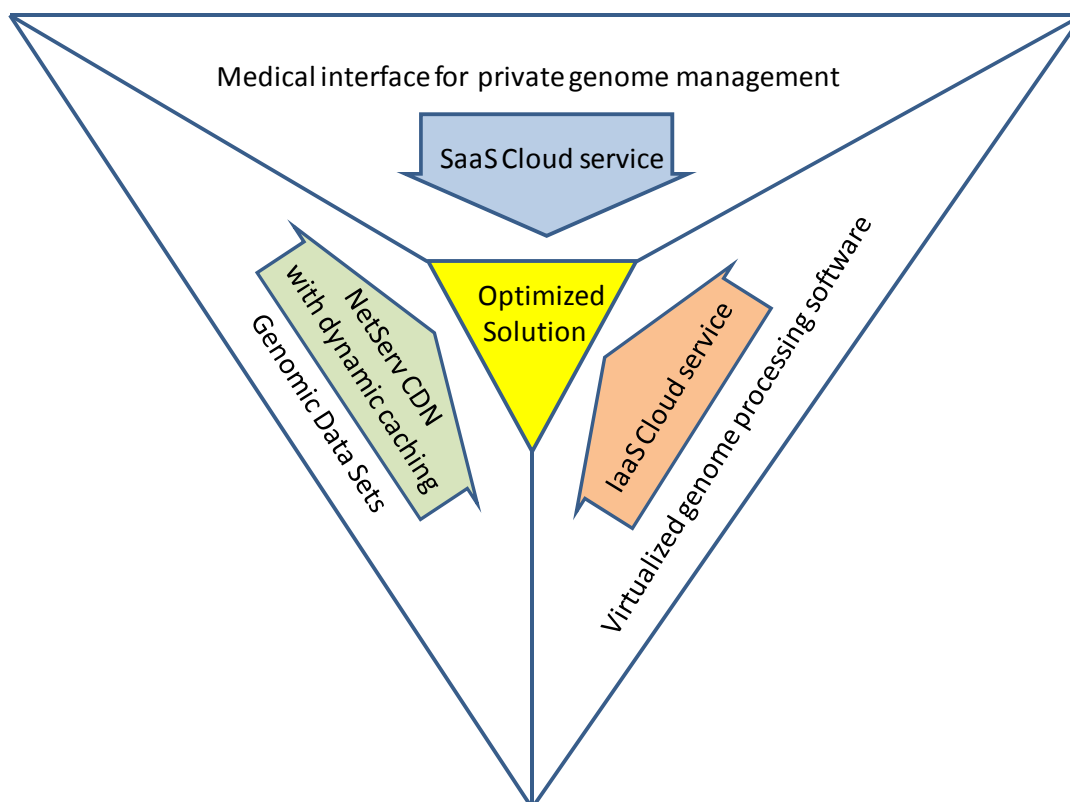


Figure 3.12: Networking and service paradigms contributing to the optimized solution.

3.3.2.1 NSIS

NSIS [2] is a suite of protocols specified by the IETF RFC 4080. It was defined for signaling information about a data flow along its path in the network, in order to allow signaling applications to install and manage states in the network. It consists of two layers:

- NSIS transport layer protocol (NTLP), which is a generic lower layer used for node discovery and message sending, which is regarded as independent of any signaling application;
- NSIS signaling layer protocol (NSLP), the upper layer, which defines message format and sequences; it contains the specific signaling application logic.

GIST (General Internet Signaling Transport protocol [3]), is a widely used implementation of NTLP. GIST makes use of the existing transport and security protocols to transfer signaling (i.e. NSLP) messages on behalf of the served upper layer signaling applications. It provides a set of easy-to-use basic capabilities, including node discovery and message transport and routing. Since its original definition, GIST allows transporting signaling message according to two routing paradigms. It provides end-to-end signaling, that allows sending signaling messages towards an explicit destination, and path-coupled signaling, that allows installing states in all the NSIS peers that lie on the path between two signaling peers.

In [4], we have illustrated our implementation of a third routing paradigm, named off-path signaling, that allows sending signaling message to arbitrary sets of peers, totally decoupled from any user data flow. A peer set is called off-path domain. We have defined and implemented three different off-path signaling scheme: (i) *Bubble*, which allows to disseminate signaling around the sender, (ii) *Balloon*, which allows disseminating signaling around the receiver, and (iii) *Hose*, which allows discovering and exchanging signaling messages with peers close to the network path connecting information source and destination. Thanks to the extensibility of GIST, other off-path domains can be implemented as they become necessary for different signaling scenarios. Details about off-path signaling are provided in report IM3.1.

3.3.2.2 NetServ

The objectives of ARES have been pursued by resorting to the NetServ *service modularization and virtualization* [1]. The former consists of providing well-defined building blocks, and use them to implement other services. The runtime environment executing services is provided by a virtual services framework, which manages access to building blocks and other resources on network nodes. Applications use building blocks to drive all network operations, with the exception of packet transport, which is IP-based. The resulting node architecture, designed for deploying *in-network services*, is suited for any type of nodes, such as routers, servers, set-top boxes, and user equipment. This architecture is targeted for in-network virtualized service containers and a common execution environment for both packet processing network services and traditional addressable services (e.g. a Web server).

It is necessary to specify the main entities of the implemented NetServ architecture in some details (see Figure 3.13).

Service containers, which are user-space processes. Each container includes a Java Virtual Machine (JVM), executing the OSGi framework for hosting service modules. Each container may handle different service modules, which are OSGi-compliant Java archive files, also referred to as *bundles*. The OSGi framework allows for hot-deployment of bundles. Hence, the NetServ controller may install modules in service containers, or remove them, at runtime, without requiring JVM reboot. Each container includes system modules, library modules, and wrappers of native system functions. The current prototype uses Eclipse Equinox OSGi framework. Service modules are OSGi bundles deployed in a service container.

Any incoming packet is routed from the network interface, through the kernel, to a service container process being executed in user space. Each NetServ module may act as either a *server module* or a *packet processing module*. This original NetServ feature overcomes the traditional distinction between router and server by sharing each other's capabilities. In the ARES system, we used only server modules, since the designed system architecture does not need the functions provided by packet processing modules.

Signaling module: The implementation of the NetServ signaling daemons is based on an extended version of NSIS-ka, an open source NSIS implementation by the Karlsruhe Institute of Technology⁴. The NetServ NSLP is

⁴ <https://projekte.tm.uka.de/trac/NSIS/wiki/>.

able to manage the hot-deployment of service bundles on remote nodes. NetServ NSLP handles 3 types of messages:

- SETUP message, which triggers the installation of a specific bundle on the remote node. Its header includes NetServ version, the requesting NetServ user id, the bundle time-to-live, and an arbitrary series of <key,value> pairs which are initialization parameters for the bundle.
- REMOVE message, which explicitly removes an existing bundle from the remote node.
- PROBE message, which is used to check if a specified bundle is instantiated on the remote node.

NetServ controller, which coordinates the NSIS signaling daemons, the service containers, and the node transport layer. It receives control commands from the NSIS signaling daemons, about the installation or removal of service bundles and filtering rules in the data plane. It makes use of the *netfilter* library through the *iptables* tool. The NetServ controller is also in charge of setting up and tearing down service containers, authenticating users, fetching and isolating modules, and managing service policies.

The NetServ repository, which stores a pool of modules (either building blocks or applications) deployable through NetServ signaling in the NetServ nodes present in the managed network.

NetServ hosts an OpenStack management bundle (the NetServ interface to OpenStack, LOIB) able to interact with OpenStack to retrieve the information on the available VMs and on the hardware capabilities, so as to offer this information to the decision system, the GCM in Figure 3.11. In addition, another NetServ bundles implements a cache (CB), used to manage the local cache for storing and managing the contents.

The CB bundle can either be executed on the same NetServ container hosting the LOIB, or on an instance of NetServ deployed within a different VM. The CB is in charge of managing the restful interface for distributing both the genomic data files and the VM image files. It implements the HTTP server function by using the NetServ native support for Jetty⁵, a light-weighted HTTP server implemented in Java, which allows creating restful interfaces.

⁵ Servlet engine and HTTP server, <http://www.eclipse.org/jetty/>.

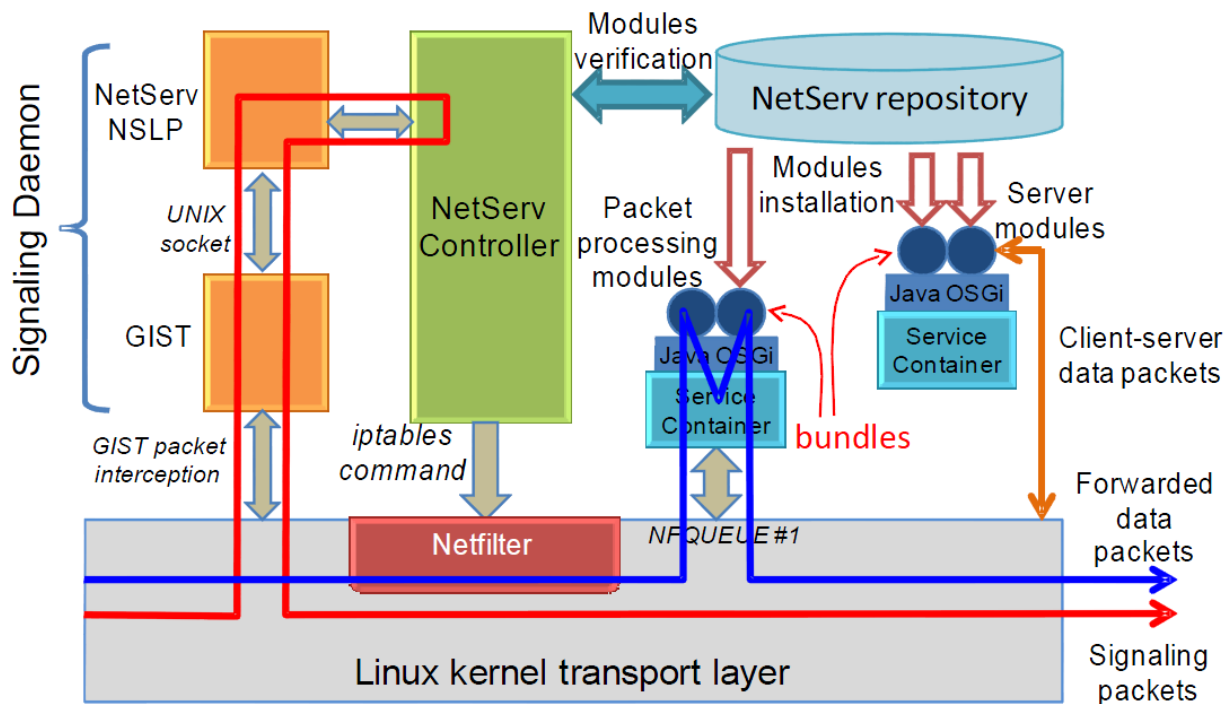


Figure 3.13: The NetServ Architecture.

3.3.2.3 OpenStack

OpenStack [5] is a cloud management system that allows using restful APIs to control and manage the execution of virtual machines (VMs) on a server pool. It allows retrieving information about the computational and storage capabilities of both the cloud and the available VMs. It also allows triggering the boot of a selected VM with a specific hardware configuration (OpenStack flavor). Thus, it is possible to wrap, inside a certain number of VMs, different existing software packages processing genome files. Each VM could expose, through OpenStack, the required configuration, which allows mapping the minimum required computational capabilities for a specific processing service into the hardware configuration for the VM that executes the service. Virtual machine image files are stored within the storage units of the PoP managed by Open Stack. In fact, OpenStack allows also using a restful API to inject new VMs into the pool, thus allowing the system to retrieve the relevant files from other locations (i.e. other GÉANT PoPs). Furthermore, OpenStack is not bounded to the use of a single hypervisor, but it can make use of different drivers to support different formats of virtual machines and hypervisors, such as VMWare, KVM, Xen, and Hyper-V.

The detailed description of the OpenStack components shown as circles in Figure 3.10 is beyond the scope of this document. The interested reader may refer to [5]. The main OpenStack modules used in our system are (see also Figure 3.14):

- *Keystone* (Identity Service): Provides an authentication and authorization service for other OpenStack services. Provides a catalog of endpoints for all OpenStack services.

- *Glance* (Image Service): Stores and retrieves virtual machine disk images. OpenStack Compute makes use of this during instance provisioning.
- *Nova* (Compute Service): Manages the lifecycle of compute instances in an OpenStack environment. Responsibilities include spawning, scheduling and decommissioning of virtual machines on demand.
- *Neutron* (Networking Service): Enables network connectivity as a service for other OpenStack services, such as OpenStack Compute. Provides an API for users to define networks and the attachments into them. Has a pluggable architecture that supports many popular networking vendors and technologies.

In the current stage of the analysis, we assume that each PoP has a certain amount of storage space and computational capabilities hosted in one or more servers and managed through OpenStack. In this virtualized infrastructure, we also assume that a specific node executes a virtual machine that hosts the NetServ environment. This solution has the advantage of being really plug&play with respect to a frequent development solution in data centre (i.e. OpenStack-based management), since adding just a VM is straightforward and do not require *any* change in the management of the computing infrastructure.

In order to share an OpenStack deployment with other users, we assume that in the OpenStack configuration the cloud administrator created a dedicated tenant, the ARES tenant. Inside this tenant, the ARES modules need administration rights. These credentials can be configured inside the LOIB module. With these credentials, the LOIB can connect to the Keystone API and request authentication tokens, granting the access to all OpenStack API, thus accessing to the resources within the resource pool assigned to the ARES tenant.

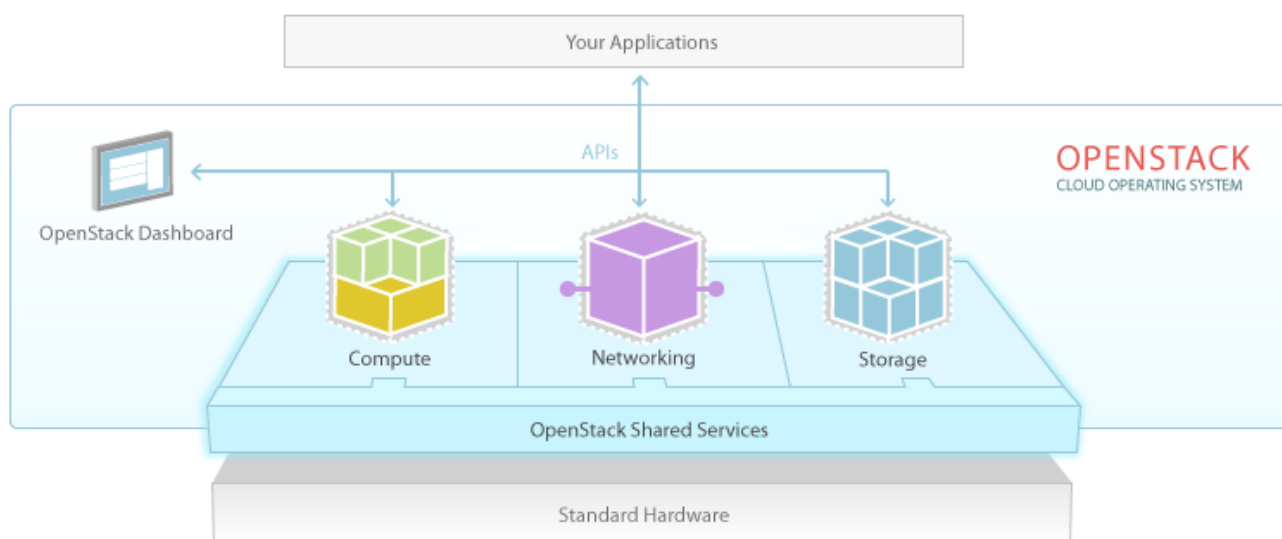


Figure 3.14: A graphical sketch of the main OpenStack components.

3.4 Chapter Conclusions

This chapter includes a detailed description of the components, functions and requirements deemed necessary for implementing the experimental research of the project ARES. For this purpose, we have abstracted all the basic components and protocols in order to outline first the performance requirements, which translate in hardware and software requirements of the genomic processing, then the functional requirements needed in the ARES distributed infrastructure.

For what concerns the performance requirements, and the corresponding resource requirements, we have also executed experiments in local servers with the aim of finding reference results usable for assessing the proper operation of the ARES distributed system currently being implemented. Through these experiments, we had the possibility of investigating the minimum required RAM space and number of CPU cores. In addition we had the possibility to investigate the effects of over-provisioning virtual machines implementing the genomic software packages. It emerges that the number of CPU cores is more effective than the available RAM, provided that a minimum amount, sometime considerable, is provided. The computing times have been perfectly compliant with the medical requirements associated to their usage and reported in Table 3.2.

Subsequently, the network architecture has been considered, with aim of identifying the so-called “General Requirements”, “Signalling Requirements” and “Application Requirements”. Implementing and deploying the ARES infrastructure meets these requirements. In particular, the signaling requirements reflect the ambitious goals of the project, since they can be accomplished through a deep revision of the on-path semantic of the NSIS function, by the introduction of the off-path signaling distribution, which allows advanced CDN functions which are the original core of the ARES proposal.

4 **Distributed Networking and Applications**

The scope of this chapter is twofold.

The first objective is to give a comprehensive description of the complete signaling exchange necessary to establish a genomic processing service. To this aim, we illustrate how all the entities forming the ARES distributed networking environment interact, which messages they exchange, and how results are provided to requesting users. We show that from the user perspective the ARES services are accessible through a simple cloud interface. In addition, we detail the underlying procedure that are executed in order to accomplish a request, based on CDN content delivery and advanced off-path NSIS signaling.

The second objective to show the “intelligence” that governs this exchange. Since one of the general objectives of ARES is to provide genomic services with a quality depending on both the service delivery time and network resource optimization, it is necessary to design suitable optimization algorithms. We show algorithms that allow identifying portions of the contents to be transferred from caches and databases so as to minimize the download time. In addition we also show an algorithm allowing the use of disjoint paths for implementing parallel downloading.

The overall objective of this chapter is indeed to give a complete and global picture of the ARES networking system.

4.1 **Advanced signaling protocols**

The ARES network entities are detailed in Chapter 3. Figure 4.1 shows the CDN signaling that will be detailed in what follows.

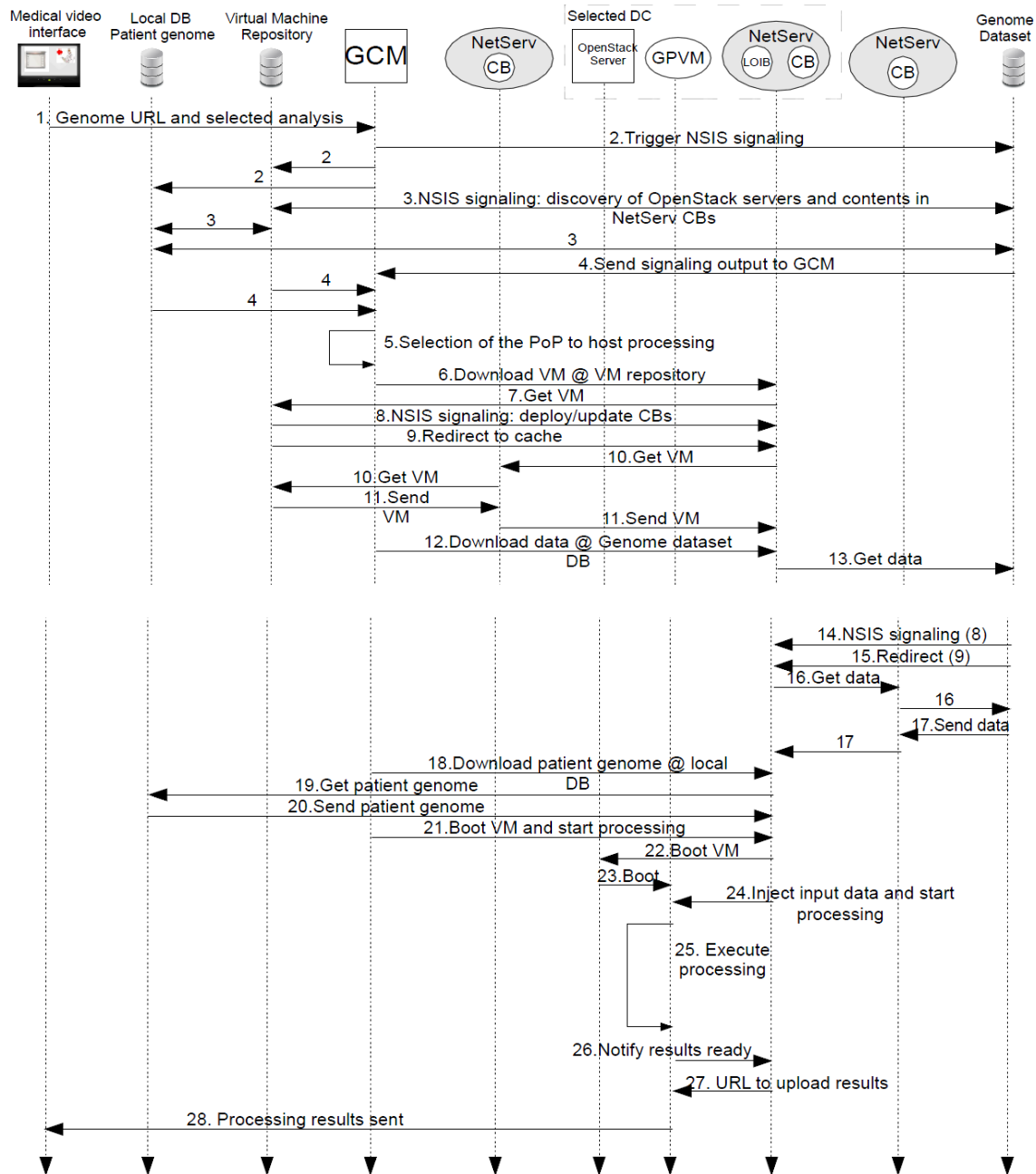


Figure 4.1: ARES Cloud signaling

In order to simplify the figure, Local DB patient genome, VM repository, and genome dataset repository have to be intended as the NetServ [1] CB acting as proxies for them. These CBs are instantiated at runtime by the CGM at the first request for download contents from them, using an appropriate off-path NSIS signaling

(remote bubble around the IP address of the selected repository [4]. Even the interactions between the OpenStack entities and the LOIB bundle are shown in a simplified way. Figure 4.1 illustrates a worst case scenario, since contents are retrieved by the original repository and subsequently cached on the path to the selected PoP. Future requests can be served by retrieving data (VM and/or genome auxiliary data) from the caches close to the selected PoP. These caches are discovered at step 3, and used by the GCM in the PoP selection process (step 5). First, the service request is issued by the medical interface to the HTTP server. In our experiments, it consists of providing the URL of a patient's genome and the selection of the "disease", corresponding to software pipeline for analyzing the patient's genome. This request is captured by the GCM, hosted by a dedicated NetServ bundle, implementing the intelligence of the system. This Controller queries the database management system for being informed about the location of repositories storing genomic data sets and VMs implementing the desired software pipeline, likely implemented through a NoSQL approach for scalability reasons. Once this information is returned to the GCM, an NSIS [6] signaling is started by the repositories towards each other, triggered by the GCM. The NSIS session will deliver a probe message according to the Hose off-path signaling that will be detailed in Section 4.1.1. Each NSLP probe message will follow the "virtual" path and collects the data on the computing and storage capabilities of the PoP servers hosting NetServ instances along the "virtual" path between the repositories storing data and the VM with the processing software. When the NetServ nodes have collected this information, they forward it to the GCM. Then, it uses the following information to find the suitable position of the cloud nodes (e.g. the Géant PoPs) where both data and VM image have to be loaded. When data are downloaded, an additional NSIS signaling allows locating on-path caches (or also slightly offpath ones), indicated by steps 8 and 14 in Figure 4.1. The on-path caches are populated by the crossing traffic, and available to provide data for any further request. When all files have been transferred, the LOIB, by interacting with OpenStack, inject the VM image in the OpenStack storage, drives the VM boot, and inject the auxiliary files in the VM, which executes the software pipeline. Then, the obtained results are returned to the requesting medical personnel. Then, the used VM is shutdown and sensible data (patient genome) are deleted, whereas the VM image and auxiliary files are maintained in the local CB for future use. Further details on the signaling protocol are given in the subsequent sections. Section 4.1.1 deals with strict NSIS signaling and deepens the algorithms and the operations implemented for Ares through the NSIS protocol suite. Section 4.1.2 introduces the NetServ integration with the NSIS framework, and the signaling logic implemented in the NetServ NSIS Signaling Layer Protocol (NetServ NSLP). Section 4.1.3 gives some details on the signaling between the GCM, the Medical Tool and the LOIB selected for the processing, which make use of HTTP messages and JSON text coding. Section 4.1.4 provides some sample simulation results. Section 4.2 is dedicated to Network and Service management. Final chapter consideration can be found in Section 4.3.

4.1.1 NSIS Signaling

In order to perform the selection of the most suitable PoP to host the computation, the GCM needs to gather information on the location of OpenStack clouds, on their resources availability, on contents location and relative distances. We assumed that each PoP of the Géant network can be represented as shown in Figure 4.2. In order to discover OpenStack cloud that may lay beyond backbone routers (BB in Figure 4.2) belonging to the network paths that connect the different data sources, a scalable off-path signaling protocol is needed.

Thus we extended the NSIS signaling protocol with new off-path capabilities, together with a gossip-based network discovery.

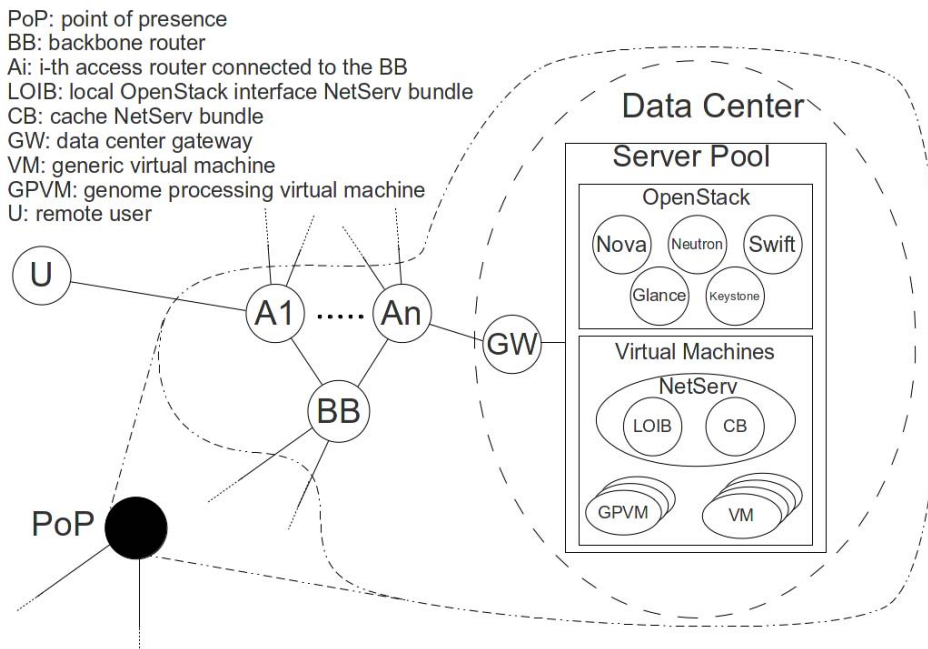


Figure 4.2: PoP internal architecture.

The IETF Next Steps in Signaling (NSIS) [21] Working Group was formed in 2001. It had the goal of designing a new generic Internet signaling protocol suite, in order to overcome the shortcomings of other signaling protocols, such as SIP and SS7, the design of which was targeted to specific needs, and may not be easily adaptable to the needs of new applications.

The NSIS framework was designed to be modular, easily extendable, secure, and customizable. However, since NSIS was primarily designed to solve RSVP issues, its QoS-aware nodes discovery process is tied to the datapath. In fact, it can be used to install, modify, and remove states both on the two communicating hosts (end-to-end signaling), and on intermediate nodes (path-coupled signaling). This is typically achieved by exchanging end-to-end signaling messages, having an IP Router Alert Option (RAO) [22] in their header, intercepted by NSIS-capable nodes along their path to the destination. However, the NSIS architecture is flexible, and signaling message routing is controlled by the so-called Message Routing Method (MRM). An MRM is the algorithm used by NSIS to both discover peers and route signaling messages. In order to decouple signaling messages from datapath a new MRMs has been designed. In this way, new signaling applications can be made available to NSIS nodes.

In ARES we have proposed a new off-path signaling MRM for the General Internet Signaling Transport (GIST) protocol [3], which is the IETF-defined version of the NSIS Transport Layer Protocol. This solution retains all

the benefits introduced with the NSIS signaling architecture, and provides off-path signaling capabilities within scopes specified by any suitable metrics, such as the number of IP hops. The proposed scheme makes use of the packet interception capabilities of GIST. Since GIST nodes may be incrementally deployed in autonomous systems (ASs), they tend to be scattered, forming an overlay. Each GIST node needs to know the GIST overlay where its signaling messages must be distributed. To solve this issue, we propose also a gossip-based GIST node discovery protocol, which (once again) makes use of the packet interception capabilities of GIST in order to minimize network overhead and discovery time. This protocol, which is the second contribution of this paper, is used to dynamically and asynchronously bootstrap any GIST network by collecting node capabilities and calculating different metrics.

The NSIS protocol suite divides the signaling functions into two layers. The upper layer, called NSIS Signaling Layer Protocol (NSLP), implements the application signaling logic. The lower layer, called NSIS Transport Layer Protocol (NTLP), is in charge of delivering the NSLP protocol messages to the next NSIS node on path. It is thus necessary to discover both the next-hop NSIS node, which may differ from the next IP node, and the transport and security services available for fulfilling the signaling application requirements.

GIST transport services make use of existing protocols in the TCP/IP suite. Normally UDP is used for delivering unreliable signaling, TCP is needed when reliability is required, and TLS over TCP solution provides secure and reliable signaling transport. GIST only delivers NSLP messages hop-by-hop between pairs of neighboring NTLP signaling nodes, whereas the end-to-end signaling functions, if needed, are provided by NSLP. Before starting a signaling session, GIST peers have to create a Message Association (MA) by using the information transported in the so-called Network Layer Information (NLI) object of GIST packets, such as the unique identifier of the GIST node (Peer Identity, PI) and one of its IP addresses.

NSIS has been primarily designed for managing states on nodes lying on data paths. For this purpose, NTLP messages may be intercepted at NSIS-capable nodes on path. In particular, the GIST protocol allows specifying messages routing policies through MRMs. Two MRMs are currently specified:

- *Path-Coupled MRM, which routes signaling messages through the data path;*
- *Loose End MRM, used for preconditioning firewalls and NAT states when data destinations lie behind them.*

GIST messages include a Message Routing Information (MRI) object, that allows NSIS nodes to identify the MRM to be used. For example, in case of a Path-Coupled MRM, GIST packets are intercepted by NSIS nodes on-path and then re-sent towards the destination, after being processed by the NSLP entities. Three encapsulations have been defined for GIST:

- *D-mode: UDP transport without packet interception;*
- *C-mode: TCP transport without packet interception;*
- *Q-mode: UDP transport with port 270 assigned by IANA(IP RAO optional [22]), and packet interception at GIST nodes. Legacy IP nodes transparently forward packets towards their destination.*

In order to limit the complexity of the signaling architecture, the NSIS WG imposed two restrictions: initial focus only on MRMs for path-coupled signaling and no multicast support. These restrictions can be lifted by taking advantage of both the modular architecture of NSIS and GIST, and the possibility of defining new MRMs.

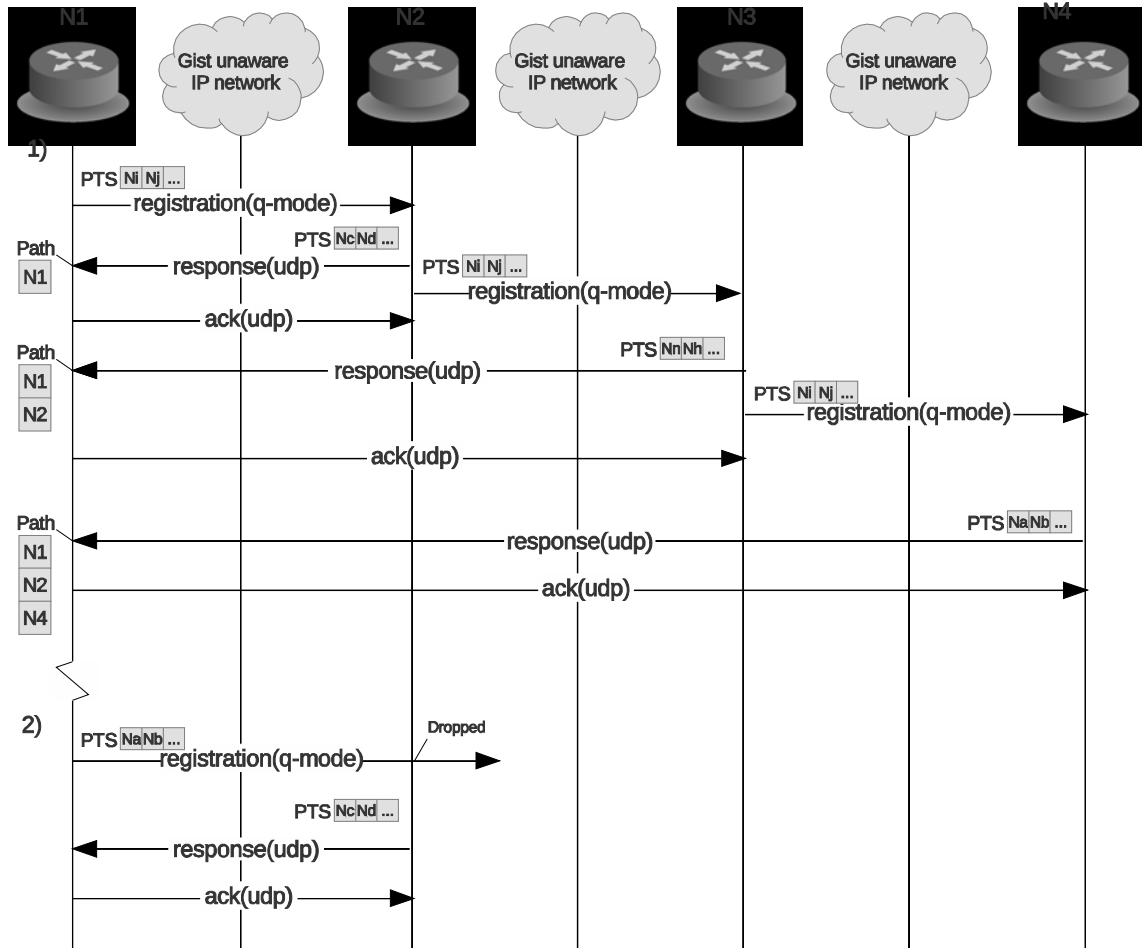


Figure 4.3: Q-mode gossip encapsulation with 1) Q-forward, and 2) Q-drop.

4.1.1.1 Gossip Based Network Discovery

The approach used to discover GIST nodes consists of a gossip protocol extension to GIST, which leverages the GIST packet interception. Gossip sessions are round-based and asynchronous. They are established between two nodes, an initiator and a responder, through a three-way handshake, which makes use of three new GIST messages: Registration, Response, and ACK. At the beginning of each round the initiator sends a

Registration message to the responder. When the responder receives this message, it replies with a Response. The handshake is closed by a final ACK message sent by the initiator.

As in other gossip protocols (e.g., see [6]), both the Registration and the Response messages include a list of GIST peers that the initiator and the responder may want to share with each other, referred to as peers to share list (PTS). Therefore, each node can establish gossip sessions with other (possibly unknown) nodes on subsequent rounds.

Differently from other gossip protocols the GIST packet interception capabilities allows the Registration message, encapsulated in Q-mode, to be received and processed not only by the responder, but also by the GIST nodes on its path (see Figure 4.3). This way, these nodes can participate to the discovery process by sending Response messages towards the initiator, sharing their own PTS list.

We envision and analyze two managing policies for intercepted Registration messages in intermediate GIST nodes. The first one, referred to as Q-forward, consists of forwarding the message towards the responder after having sent a Response back to the initiator (Figure 4.3.1). By handshaking with each node, the initiator can evaluate its downstream distance from all nodes along the path, in terms of both GIST/IP hops and, roughly, latency for reaching each of them. Since IP routing could be asymmetric, these distances may not be valid in the reverse direction. Thus, the only reliable upstream information, collected by the responder and intermediate GIST nodes, is the initiator identity.

The second policy, denoted Q-drop, allows only the first GIST node on the path to intercept the Registration and return the Response. Then, the Registration is discarded and not forwarded downstream, as illustrated in Figure 4.3.2.

The design goals of these two approaches are different. The Q-forward policy aims to discover overlay paths and evaluate the associated metrics. This information can be used by the off-path signaling distribution algorithms presented in Section 4.1.1.2. The goal of the Q-drop policy is simply to allow the initiator to collect the identities of the GIST neighbors, that is GIST nodes at one GIST hop away, together with the number of IP hops within this GIST hop, and a rough estimation of the communication latency. Although the Q-drop policy allows achieving a partial knowledge of the network, it scales well in networks with many nodes. In addition, it provides enough information to run the flooding-based offpath distribution strategy illustrated in section 0.5. We now illustrate the (common) procedure used to bootstrap the GIST protocol when a node is turned on. It is inspired by [24]. Then, we detail two gossip solutions, called Leaf and One-hop, based on Q-forward and Q-drop policies, respectively.

Gossip bootstrap procedure for GIST nodes

When a GIST node is turned on, its list of reachable GIST nodes is empty. Thus, it is necessary to ask another always-on node, called tracker, an initial list of active GIST nodes to gossip with. Thus, the tracker acts as the first GIST responder (node N4 in Figure 4.3). A tracker address must therefore be statically configured in all GIST nodes. A different solution consists of making use of an IP-to-ASN public mapping service. Any turned-on GIST node can use its own IP address to retrieve the Autonomous System Number (ASN), referred to as *asni*, from this mapping service. Then, the node can obtain the tracker IP address by issuing a DNS query to resolve

asni.nsis.org. This procedure needs the creation of a generic container domain, such as “nsis.org” to which AS providers can register the IP addresses of the trackers of their networks, similarly to [25]

After this initial procedure, a GIST node knows (at least) one additional GIST node, and can periodically establish a gossip session with it, in order to update its PTS list. Clearly, a GIST node cannot know when it has discovered all the other GIST nodes in the network (end of discovery phase). It assumes that when it does not learn the identity of any new GIST node for a given number of subsequent cycles, it enters the steady phase. In this case, it can increase the gossip session period, in order to limit the consumption of network resources.

Leaf-based gossip protocol

Before entering the protocol details, we present a mathematical model of the GIST gossip protocol based on the Q-forward policy. Our network model consists of a graph of GIST nodes only, referred to as GIST overlay, denoted as $G=(\mathbf{V},\mathbf{E})$. \mathbf{V} is the set of GIST nodes with cardinality $K=|\mathbf{V}|$, and \mathbf{E} is the set of the undirected edges. The IP routing of GIST packets, which determines the elements in \mathbf{E} connecting GIST peers, makes use of the underlying IP routing protocols, which we assume to be based on shortest-path algorithms. GIST can intercept packets when the Q-mode encapsulation is used, and applies the Q-forward policy to these packets. We define a path $\pi_{ij} = \{i, k, \dots, j\}$ as the ordered sequence of GIST nodes on the IP path from i to j , and we denote by $s_{ij} = \pi_{ij} - \{i\}$ the path without the source node, that is the sequence of GIST nodes visited by a packet sent by the peer i towards the peer j . We define $\mathbf{S} = \{s_{ij} \mid i, j \in \mathbf{V}\}$. For each GIST node $i \in \mathbf{V}$, we also define the set of its GIST neighbors as $\mathbf{N}_i = \{j \in \mathbf{V} \mid s_{ij} = \{j\}\}$, with $A_i = |\mathbf{N}_i|$.

Let us focus on the discovery phase. It allows all GIST nodes in \mathbf{V} to receive the identities of the other GIST nodes and to evaluate the relevant metrics. The minimization of the discovery time translates in minimizing the number of gossip sessions. We model this problem as a *set covering problem* (which is a class of problems known to be NP-hard, see [26]): given a node $i \in \mathbf{V}$ and the associated universe $\mathbf{U}_i = \mathbf{V} - \{i\}$, the set $\mathbf{S}_i = \{s_{ik}, k \in \mathbf{D}_i \subseteq \mathbf{U}_i\}$, $\mathbf{S}_i \subseteq \mathbf{S}$, is a cover for \mathbf{U}_i if the union of its elements contains all elements in \mathbf{U}_i . Thus, it is possible to formulate the following problem (\mathbf{C}_1):

$$\min \sum_{i=1}^K |\mathbf{D}_i|, \quad (1)$$

subject to

$$\mathbf{U}_i = \bigcup_{s_{ik} \in \mathbf{S}_i} s_{ik} = \bigcup_{k \in \mathbf{D}_i} s_{ik}, \forall i \in \mathbf{V}. \quad (2)$$

The solution of this problem, that is the identification of the minimum sets $D_i, i \in V$, provides a solution of the GIST discovery problem for all GIST peers. In fact, $|D_i|$ is the minimum number of gossip rounds necessary to the node i to contact *all* the other GIST nodes in V by leveraging the interception capability of GIST. In addition, for each GIST node $i \in V$ we define the single-source shortest-path tree T_i rooted at i . T_i identifies the GIST nodes on the (shortest) IP path towards any other node $k \in V$. An example of T_i for a very simple graph G is drawn in bold in Figure 4.4, where $i=1$. We say that a node $h \in V$ is a leaf for i if it is a leaf for the tree T_i , that is $h \in s_{ij} \Leftrightarrow h = j$. We denote as L_i the set of leaf nodes for i , and $M_i = |L_i|$. Paths associated to leaves for node 1 are shown by red dashed arrows in Figure 4.4.

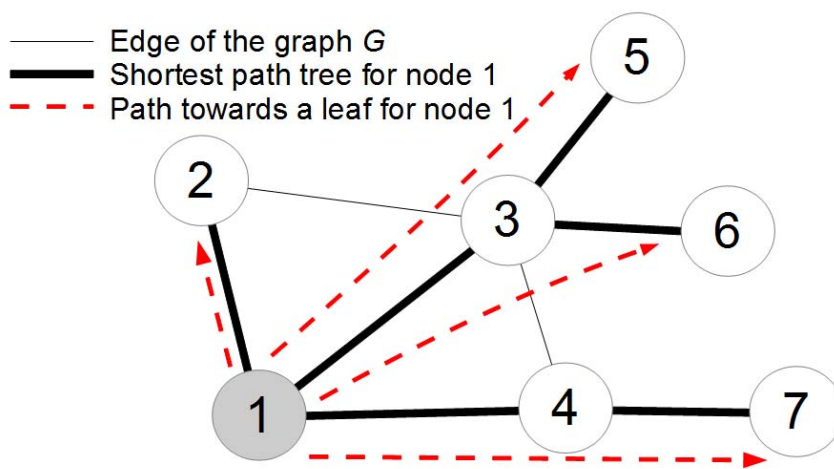


Figure 4.4: Example of possible subset of the universe and solution for the discovery problem for node 1. The tree T_1 is drawn in bold.

Our proposed solution of the problem (C_1) is based on the following consideration. If a node i executes a gossip session with all the leaves of its T_i tree, it certainly discovers all the GIST nodes in G , together with the relevant metrics, thanks to the Q-mode encapsulation and Q-forward policy. Thus, our solution is aimed to quickly discover all Leaf Peers (LPs) of the tree associated with each node in V . A formal proof of this statement is provided by *Theorem 1*.

Theorem 1: The optimal solution D_i^* to the set cover problem (C_1) is given by the sets of leaves for each node in the overlay, that is $D_i^* = L_i, i \in V$.

Proof: By definition of leaf, each node $h \in L_i, i \in V$, belongs to the optimal set of solution D_i^* , that is $L_i \subseteq D_i^*$, otherwise U_i is *not covered*. We show that $D_i^* = L_i$. Assume, by contradiction, that $\exists z \in D_i^* - L_i$. Then, since z is not a leaf, $\exists y \in L_i \mid z \in s_{iy}$. Thus, from the shortest path routing assumption it follows that $s_{iz} \subset s_{iy}$. Since

$y \in L_i \subseteq D_i^*$, thus $D_i^* - \{z\}$ is still a solution of (C_1) (see (2)) but with a lower cost than D_i^* (see (1)). Consequently D_i^* cannot be an optimal solution for i . \square

Similarly, it is very easy to show that L_i is the solution also for the optimization problem modeling the network resource consumption during the discovery phase, denoted as (C_2) :

$$\min \sum_{i=1}^K \sum_{j \in B_i} [p_{ij}q + \sum_{z \in S_{ij}} (p_{zi}r + p_{iz}a)], \quad (3)$$

subject to

$$U_i = \bigcup_{k \in B_i} S_{ik}, \forall i \in V, \quad (4)$$

that is $(C_2) \equiv (C_1)$, since they have the same solution $B_i = D_i = L_i$, where

p_{ij} is the distance, defined as the number of IP hops, from the node i to the node j on the tree T_i , i.e. on S_{ij} .

q, r , and a are the the size, at IP layer, of the Registration, Response, and Ack messages, respectively.

Thus, to cope with all these issues, we resort to a heuristic which is aimed to quickly discover LPs of each GIST node, since all the other GIST nodes will be discovered by intercepting Registration messages. Even if the optimal solution of the network discovery problem is known, it is difficult to implement it in practice, since the system works in a distributed fashion with an incomplete knowledge of the overlay. In fact, even if the GIST node is an IP router, by simply inspecting its routing table it is not possible to identify the leaves of the GIST overlay. In order to infer the whole network topology (not only the overlay), each GIST node should behave as a passive listener at GIST layer for routing protocols messages, which seems to be much beyond its scope. In addition, GIST nodes are not necessarily routers, but they can also be end nodes, which cannot execute routing functions. Last but not least, the identities of GIST nodes are not known at bootstrap, but they are discovered during the execution of the discovery phase, which we want to optimize by contacting only specific nodes (often still unknown) to limit the number of necessary gossip rounds, for LPs, as in Figure 4.3.1. For this reason, we call this solution Leaf-based. In addition, each node, in order to let other nodes quickly discover *their* LPs, exchanges only the identities of *potential* leaves in the PTS field of Registration and Response messages. To this aim, we have defined simple, lightweight, and soft-state structures storing GIST peer information at each GIST node. The former, called peer table (PeT), stores the identities of the other GIST peers, together with their associated metrics (peer element, PE). The latter, called path table (PaT), stores in node i the *ordered* sequence of PEs in S_{ij} , as new GIST nodes are discovered and contacted. The PaT is

computed by each *initiator* i by inspecting the Response messages sent by any intermediate node k that has intercepted the Registration message destined to a *responder* j .

It is worth to note that a given node z , which is a leaf for i , not necessarily is a leaf also for j , which receives the PTS list sent by i . Finally, it may happen, especially in the initial gossip rounds, that a newly activated GIST node knows just a limited number of peers, thus the identities it shares could not be *true* leaves.

We now detail the algorithms executed in GIST nodes.

The main loop executed by a GIST *initiator* is shown in Figure 4.5, which includes the management procedure of the gossip timer T_{gossip} . T_{gossip} is increased if no PeT changes are done for a number *maxCounter* of subsequent gossip sessions (see lines 8-16), and restored to its initial value when a change happens (lines 5 and 6). When the timer expires, this procedure calls the function *gossipSession*, which is the core procedure executed by an *initiator*, illustrated in Figure 4.7 and detailed below.

Procedure mainLoop

Input: $minT_{gossip}, maxT_{gossip}, maxCounter$

```

1:  $T_{gossip} \leftarrow minT_{gossip}$ 
2:  $gossipSessionCounter \leftarrow 1$ 
3: loop
4:   if new peer arrived during past gossip session  $\wedge$ 
      PeT contains uncontacted peers then
5:      $T_{gossip} \leftarrow minT_{gossip}$ 
6:      $gossipSessionCounter \leftarrow 1$ 
7:   else
8:     if  $gossipSessionCounter \leq maxCounter$  then
9:        $gossipSessionCounter ++$ 
10:    else
11:      if  $2T_{gossip} \leq maxT_{gossip}$  then
12:         $T_{gossip} \leftarrow 2T_{gossip}$ 
13:      else
14:         $T_{gossip} \leftarrow maxT_{gossip}$ 
15:      end if
16:    end if
17:  end if
18:  start  $T_{gossip}$  timer
19:  call Procedure gossipSession
20:  if  $T_{gossip}$  timer is not expired then
21:    wait  $T_{gossip}$  timer expiration
22:  end if
23: end loop

```

Figure 4.5: Pseudocode of the main loop of a Gossip Initiator.

The *initiator* must *select* two types of peers stored in the PeT: the so-called *peer to gossip* (PTG), which is the *responder*, selected by invoking the function *getPeerToGossip* (lines 1-7,), and the PTS list, which includes the identities of the PEs to share with the PTG and any intercepting nodes (lines 8-10), through the invocation of the function *getPeers*. Since gossip sessions must be established only with leaves, the algorithms implemented to select PTG and PTSs aim at this goal. In the PE entry of the PTG stored in the initiator, the flag *isTried*, used for managing soft states, is set to “true”. The gossip Responses received by the initiator are managed by the loop described in lines 14-36. When the initiator receives the PTS list within a Response sent by a remote peer, it performs two actions:

each element of the received PTS not already present in the PeT is added, and its flags *<isGossiped, isContacted>* are set to *<true, false>*. This is important for subsequent selection of PTG and PTS. In fact, since each node tries to share just LPs, an identity received in a PTS list is a good candidate for being selected as future PTG;

each GIST intercepting peer on the path to the PTG is added to the PeT, if not present, together with its metrics, and the relevant flag *isContacted* is set to “true”. This peer is not a good candidate for future selection of the PTG or a PTS element, since it is not a leaf peer (LP) for the initiator.

Two issues are still open. The first is the metric evaluation and path construction in the PaT maintained by the *initiator*. The second is the selection of PTG and PTS elements.

For what concern the first issue, the Registration message includes an NLI element (Figure 4.6), used to compute the distance between the *initiator* and the PTG in terms of IP and GIST hops. The *initiator* initializes a *TTL* value in the IP hop field of the NLI (TTL_{NLI}) of the Registration message, which cannot be modified by intercepting nodes. This value is copied in the TTL field of the IP header (TTL_{IP}), which is decremented at each IP hop. Finally, the *TTL* value is also copied in the GIST hop field of the GIST header (HOP_{GIST}), which is decremented at each GIST hop.

```
Registration =  Common-Header
                [ NAT-Traversal-Object ]
                Message-Routing-Information
                Session-Identifier
                Network-Layer-Information
                Supported-NSLPs
                [ Node-List ]
```

```
Response =     Common-Header
                [ NAT-Traversal-Object ]
                Message-Routing-Information
                Session-Identifier
                Network-Layer-Information
                Supported-NSLPs
                [ Node-List ]
```

```
Ack =          Common-Header
                [ NAT-Traversal-Object ]
                Message-Routing-Information
```

Session-Identifier
Network-Layer-Information

Figure 4.6: GIST extension packet format.

Once the Registration packet is intercepted by a GIST node, either a forwarder or a responder, it computes its IP and GIST distances from the *initiator*. The IP distance is evaluated as $TTL_{NLI} - TTL_{IP}$ and written in the NLI of the Response, whereas the GIST distance is $TTL_{NLI} - HOP_{GIST}$ and written in the GIST hop field (lines 2-8 of pseudo-code shown in Figure 4.8). The Response message is sent by using the D-mode encapsulation, so as to not be intercepted and, consequently, to keep the GIST hop field untouched.

In turn, the *initiator* updates a temporary path list (*peerList*) as it receives Responses. The position in the *peerList* of a peer is exactly its distance in GIST hops. The procedure is completed when the GIST hop distance of the PTG is equal to the number of received responses (size of the *peerList*), and the last element of that list is exactly the PTG (note that the function *last(·)* returns the last element of the list). If this condition is not met, at T_{gossip} expiration, the path is truncated at the last peer having a position equal to its distance. In any case, the new path is added to the *pathList* stored in the PaT (function *updatePath List*). Finally, the *initiator* sends an ack message back, which does not include any PTS. Through this procedure, the *initiator* can also roughly estimate the round trip latency to any responding peer.

Procedure gossipSession**Input:** (*peerList*, *pathList*)**Output:** *pathList*

```

1: if this is the first gossip session then
2:   peerToGossip ← tracker
3: else
4:   call procedure getPeerToGossip
5:     Input: peerTable
6:     Output: peerToGossip
7:   end if
8: call procedure getPeers
9:   Input: Number of peers to share, peerToGossip
10:  Output: peersToShare: list of peers to share
11: Send peerToShare list to peerToGossip
12: peerToGossip.isTried ← true
13: peerList: empty list of element
14: while  $T_{gossip}$  is not expired do
15:   Receive sharedPeers from peer in a Response
16:   for all element ∈ sharedPeers do
17:     if element ∉ peerTable then
18:       add element to peerTable
19:       element.isGossiped ← true
20:       element.isContacted ← false
21:     end if
22:   end for
23:   if peer ∉ peerTable then
24:     add peer to peerTable
25:     peer.isContacted ← true
26:   else
27:     update peer metrics
28:   end if
29:   append peer to peerList
30:   order peerList basing on GIST Hop
31:   if last(peerList) == peerToGossip ∧
|peerList| == peerToGossip.gistHop then
32:     send gossip ack to peer
33:     break while
34:   else
35:     send gossip ack to peer
36:   end if
37: end while
38: call procedure updatePathList

```

Figure 4.7: Pseudocode of a Gossip Session of the Gossip Initiator.

Procedure gossipDaemonLoop

```

1: loop
2:   Receive a gossip Registration message
3:    $TTL_{IP} \leftarrow$  packet IP header TTL field
4:    $HOP_{GIST} \leftarrow$  message.GISThopCount
5:    $TTL_{NLI} \leftarrow$  message.nli.ipttl
6:   create a gossip Response message gossipResponse
7:    $gossipResponse.nli.ipttl \leftarrow TTL_{NLI} - TTL_{IP}$ 
8:    $gossipResponse.GISThopCount \leftarrow$ 
    $TTL_{NLI} - HOP_{GIST}$ 
9:   call procedure getPeers
10:    Input: (Number of peers to share, gossip initiator)
11:    Output: PTS: list of peers to share
12:     $gossipResponse.peersToShare \leftarrow PTS$ 
13:    send gossipResponse to the gossip initiator
14:    if message.destination is not  $\in$  myAddresses then
15:      forward message toward destination
16:    end if
17:    for all element  $\in$  message.sharedPeers do
18:      if element  $\notin$  peerTable then
19:        add element to peerTable
20:        element.isGossiped  $\leftarrow$  true
21:        element.isContacted  $\leftarrow$  false
22:      end if
23:    end for
24:    if initiator  $\notin$  peerTable then
25:      add initiator to peerTable
26:      initiator.isGossiped  $\leftarrow$  false
27:      initiator.isContacted  $\leftarrow$  false
28:    end if
29:    loop
30:      Receive a gossip acknowledgement
31:      break inner loop
32:    end loop
33: end loop

```

Figure 4.8: Pseudocode of the Daemon Loop of a Gossip Responder/Forwarder.

The selection of PTS elements is a common process to *initiator*, *forwarders*, and *responder*. Assume that the maximum size of the PTS list is H , which is a protocol design parameter. Since the Leaf-based gossip protocol aims at LPs, and shared identities are good candidates to be gossiped, if the PaT includes at least H paths, H randomly selected LPs of these paths are used. Otherwise, the node tries to fill the PTS list by using peers already discovered but still not contacted, that are identifiable by the flag *isContacted* $=$ *false* in their PEs. Such peers are those that have already contacted the selecting node, or those whose identities have been shared by other nodes (i.e. they have also the flag *isGossiped* $=$ *true*). Since uncontacted peers might

also be LPs, this approach is preferable to making use of peers already contacted, which are not LPs, given that all nodes should preferably share LPs.

PTG is selected randomly from three priority lists. The first list, referred to as *high priority*, includes uncontacted PEs with the flag *isGossiped* set true, since they are most likely LPs. The second list, referred to as *low priority*, includes uncontacted PEs with the flag *isGossiped* set false, that are not likely to be LP. Finally, the third list, referred to as *no priority*, includes all LPs of the PaT. Thus, uncontacted peers are preferably selected, in order to quickly accomplish network discovery. When all peers have been contacted (priority lists are empty), peers enter the steady phase, during which just LPs are gossiped, in order to update the status of the highest possible number of peers by a single Registration message.

Since PE states are soft, they are removed if not refreshed. The lifetime value depends on the number of paths in the PaT (*pathList*), and is set equal to $\max T_{gossip} \times \max(|pathList|, 1) \times (1 + \Delta)$, where Δ is a parameter used to avoid accidental PE cancellations.

The PaT consistency is guaranteed by updates done when a new path is collected during a gossip session. For space limitations, we do not report details of the function *updatePathList*, which merges, updates, or truncates paths already present in the *pathList* as a consequence of a gossip session (line 38 of Figure 4.7)

One-hop gossip protocol

An alternative network discovery approach is to limit the scope of the gossip exchanges to the neighboring GIST peers only, in order to limit the communication overhead. This approach will be referred to as One-hop. In the One-hop strategy, Registration messages are encapsulated in Q-mode and managed by the Q-drop policy by GIST forwarders (see Figure 4.3.2). We can again model the network discovery problem as a set covering problem (C_3):

$$\min \sum_{i=1}^K |P_i|, \quad (5)$$

subject to

$$N_i \subseteq \bigcup_{j \in P_i} S_{ij}, \forall i \in V, \quad (6)$$

where $P_i \subseteq U_i$ is the set of destinations of the Registration messages sent by node i . N_i is clearly an optimal solution of (C_3), since the inclusion in (6) becomes an equality. However, any set of destinations $F_i \neq N_i, F_i \subseteq U_i$ respecting the condition (6) with the additional constraint

$$s_{ij} \cap s_{ik} = \emptyset, \forall j, k \in F_i, \quad (7)$$

is also an optimal solution of (C_3) , without any additional network overhead with respect to the trivial solution N_i due to the usage of the forwarding policy Q-drop. In fact, while the condition (6) ensures that the union of sequences s_{ij} with destinations $j \in F_i$ is a cover of the set N_i , the condition (7) ensures that each GIST neighbor is counted just once, and thus the result of the sum (5) is $\sum_{i=1}^K A_i$. Nevertheless, even if the problem can be easily solved by an “oracle” having a complete knowledge of the overlay G , it is all but trivial to solve in operation, for the same reasons related to the access to routing information mentioned in section 3.1.2.

In order to efficiently discover destinations while respecting both (6) and (7), each peer classifies PEs as follows:

- Neighbor peer: a peer at 1 GIST hop distance, with metric values obtained by a complete gossip session. Its flags $\langle \text{isContacted}, \text{isTried} \rangle$ are set to $\langle \text{true}, \text{true} \rangle$;
- Unreachable peer: a peer located at a GIST hop distance greater than 1, that the peer unsuccessfully tried to contact in a gossip session. Metric values are thus undefined. Its flags $\langle \text{isContacted}, \text{isTried} \rangle$ are set to $\langle \text{false}, \text{true} \rangle$;
- Uncontacted peer: a peer with no contact attempt. Its flags are set to $\langle \text{false}, \text{false} \rangle$.

A node tries to contact still uncontacted peers first (PTG in discovery phase), and then all other ones. The use of distinct flags for neighboring and unreachable peers allows contacting only neighbors in the steady phase, but not unreachable peers that are behind already discovered neighbors. The transition from the discovery to the steady phase is managed as in the Leaf-based solution, according to the algorithm illustrated in Figure 4.5. The PTS selection is random, since there is no value in limiting the choice to one or two of the above categories. In fact, differently from leaves, the sets of GIST neighbors could be quite different for two neighboring peers exchanging their PTS list each other.

Although the number of peers to contact is significantly lower than in the leaf-based approach (in general $A_i = M_i$), the fact that at each gossip cycle it is possible to contact only one peer (i.e., the first on the route to the destination) could be disadvantageous, since it allows to classify just one peer as unreachable/neighbor at a time.

An important feature of the One-hop discovery process is the management of the unreachable peer identities. In order to avoid storing useless PEs with no valid metrics associated, we limit the maximum number of unreachable peers in the PeT. The unreachable peers in the PeT are managed according to a Least Recently Used (LRU) strategy. Without this limitation, the size of the PeT could increase up to the number of GIST nodes in the network. However, storing a small number of unreachable peer identities is useful. In fact, it allows avoiding to start gossip handshakes towards unreachable peers already checked, which would only increase the network overhead and, eventually, the discovery time of neighboring peers. In addition, having unreachable peers to check would not allow the algorithm to switch from the discovery to the steady phase (i.e. to relax the T_{gossip} timer, lines 4-6 in Figure 4.5), even if all neighbors for that node have been discovered.

Another important feature of the One-hop discovery is the management of multiple IP addresses of GIST peers, as it happens for routers or multihomed servers. When the NLI of a received gossip message includes the PI of a peer already present in the PeT and labeled as unreachable, if the IP address in the NLI is different from the one stored in the PeT, this address updates the stored one, and the PE is set as uncontacted. Consequently, a new attempt to contact the PE may be done by using the newly inserted IP address in subsequent gossip sessions. This feature is necessary since IP nodes with multiples interface in densely connected networks can be reached through different paths depending of the used IP address, when the path cost is equal. PE are modeled through soft states also in this solution.

4.1.1.2 *Off-path GIST dissemination*

We propose a new MRM that allows sending signaling messages decoupled from the data path. Our objective is to distribute GIST signaling messages within off-path distribution domains, including peers which meet specific conditions. Clearly, distribution domains considered in what follow are related to data paths, but not coincide with them.

We will make use of the following notation, where r is a radius expressed in IP hops:

$$d_i(r) = \{j \in V : p_{ij} = r\}, i \in V;$$

$$c_i(r) = \{j \in V : p_{ij} \leq r\} = \bigcup_{l=1}^r d_i(l), \text{ with } i \in V.$$

The proposed off-path dissemination types are as follows:

Bubble: a region around an initiator i . Signaling messages are delivered to $c_i(r)$;

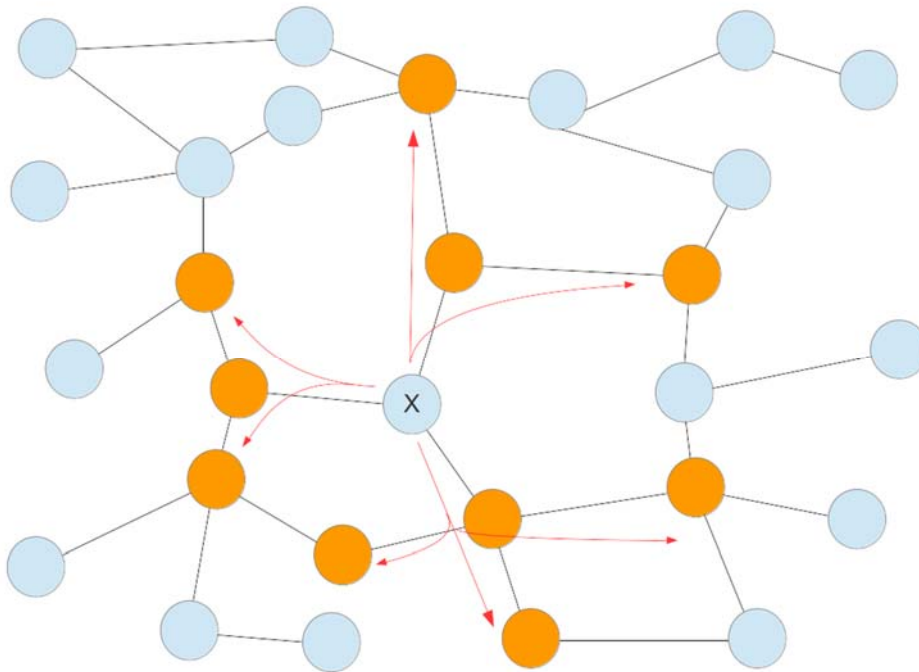


Figure 4.9: off-path bubble of radius 2 exploded from the node X.

Balloon: signaling messages are delivered from an initiator i to a node y . They are intercepted by GIST nodes $z \in s_{iy}$; y has to deliver the message to $c_y(r)$. Hence, signaling messages are received by all GIST nodes $z \in s_{iy} \cup c_y(r)$;

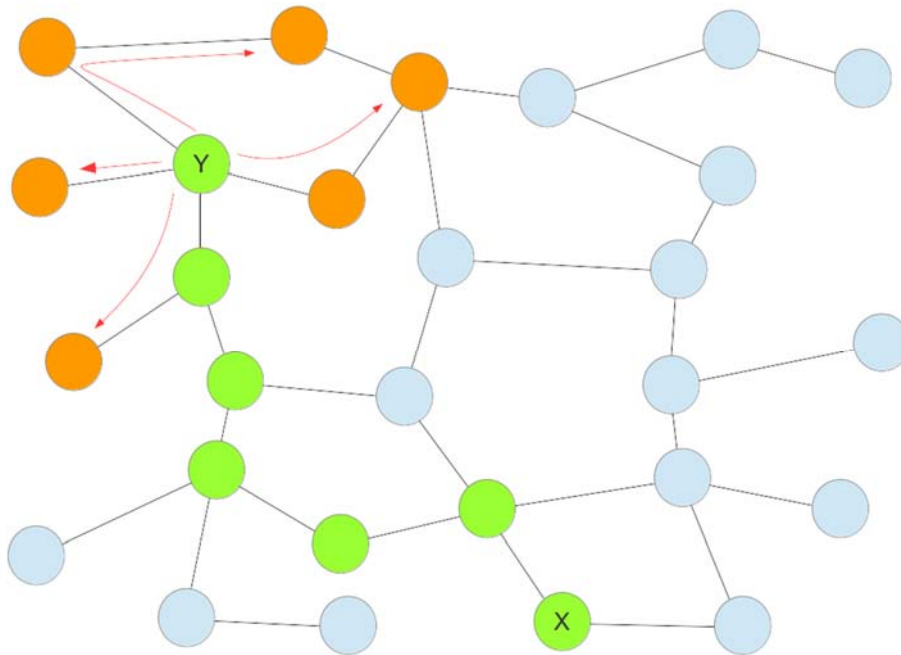


Figure 4.10: off-path balloon of radius 2 sent from the node X to the node Y.

Hose: signaling messages are delivered from i to y . Each GIST node $z \in \pi_{iy}$ delivers the message to $c_z(r)$. Hence, signaling messages are received by all GIST nodes $k \in \bigcup_{z \in \pi_{iy}} c_z(r)$.

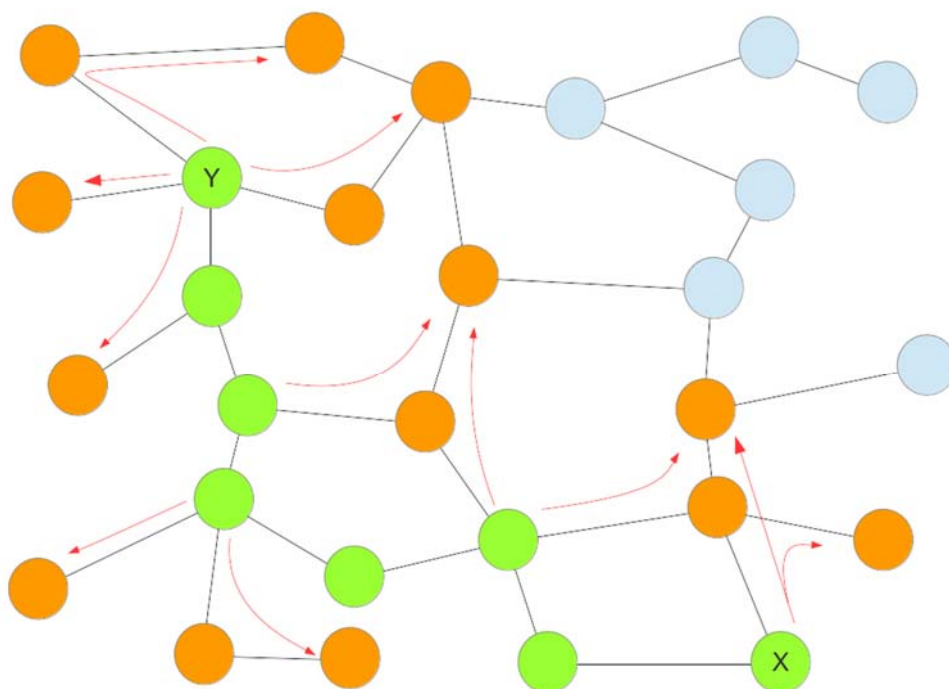


Figure 4.11: off-path hose sent from the node X to the node Y.

The definitions above could also make use of different metrics, such as GIST hops count. Without loss of generality, in what follows we will make use of the IP hop count.

Since the proposed off-path domains are built by using Bubble domains, we focus on the distribution of signaling messages within a Bubble.

The off-path MRI for the proposed MRM is shown in Figure 4.12. When an NSLP protocol requests the delivery of an off-path signaling message, it must indicate the metric specifying the radius of the off-path domain. Such a request is processed by the GIST *initiator*, which identifies the set of the destination peers compliant with this metric. The matching condition is verified for each element of the tuple $\langle \text{off-path domain type, metric type, value} \rangle$. Before sending signaling messages to the selected destinations it is necessary to execute the GIST three-way handshake to create MA between peers and, if required by the NSLP protocol, the underlying transport protocol handshake [3].

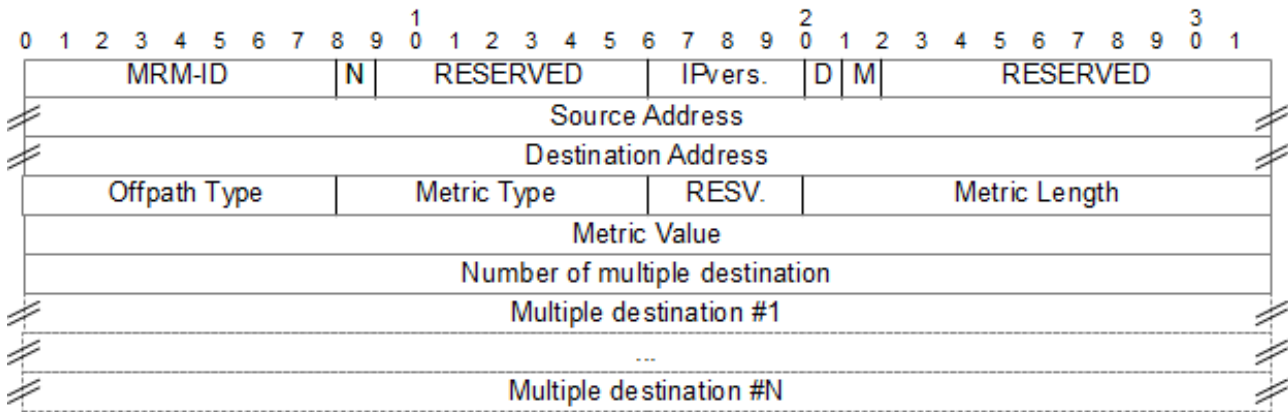


Figure 4.12: GIST off-path MRI.

For implementing the Bubble, we propose three algorithms for selecting the signaling destinations:

GIST Flooding

The flooding algorithm can make use of the information made available by either One-hop-based or Leaf-based discovery. The selection of destinations is done locally at each peer, as signaling spreads over the Bubble.

Each node receiving an off-path message reads the metric value m in the MRI. If $m > 0$, the node searches in its PeT all the peers 1 GIST hop away whose IP distance is not larger than m . For each destination, it decrements m by the IP distance of the selected next GIST hop, and then sends the message. Clearly, any forwarder does not send the message back to the sender. In order to avoid the packet duplication problem, typical of flooding protocols, we have introduced a new GIST error message that may be sent during the three-way GIST handshake. When a peer receives a GIST query, it stores the identifier of the served NSLP, the GIST session identifier, and the value of m used to build the bubble, called m_{prev} . Then, if it receives a further GIST query [3]. from another peer with the same GIST session value, it replies with the error message and aborts the session if $m \leq m_{prev}$, otherwise it rebuilds the bubble by using m .

GIST Broadcast

This algorithm has already been proposed in [4], and it is used here as performance baseline. In this algorithm the destination selection is done by *initiators*, which aim to cover the complete set of nodes which meet the metric condition. An initiator i identifies the sub-tree $\tau_i(r)$ included in the Bubble of radius r (i.e. $\tau_i(r) = T_i \cap c_i(r)$), and sends a signaling message to each leaf of this sub-tree, denoting by $\lambda_i(r)$ this new set of leaves. Since each GIST node i stores in its PaT all the discovered paths s_{ij} , in order to obtain $\tau_i(r)$ it is enough to truncate them by excluding the elements with a metric larger than r . The final element of each truncated path will provide the elements of $\lambda_i(r)$, discarding repetitions. If some paths to the destination peers share some GIST nodes, these nodes will receive replicated queries and data messages. However, in order to reach all the selected destinations, these nodes have to intercept and forward signaling messages without generating error messages. The pseudocode of the algorithm is reported in Figure 4.13.

Procedure GIST Broadcast

- 1: $b_i = \emptyset$ //this set contains the signaling destinations
- 2: $w_i(r) = c_i(r)$ //temporary set
- 3: **for** $j = r \rightarrow 1$ **do**
- 4: $b_i(r) = b_i(r) \cup \{d_i(j) \cap w_i(j)\}$
- 5: $w_i(j-1) = w_i(j) - \bigcup_{y \in \{d_i(j) \cap w_i(j)\}} \{s_{iy} \cap w_i(j)\}$
- 6: **end for**

Figure 4.13: Pseudocode of the GIST Broadcast algorithm.

GIST Multicast

In this algorithm, shown in Figure 4.14, peers use the Multiple Destination field of the MRI shown in Figure 4.12, in order to aggregate multiple signaling destinations within a single signaling session. An initiator i identifies $\lambda_i(r)$ defined above, and initializes $l_i(r) = \lambda_i(r)$. Then, for each leaf $y \in l_i(r)$, it checks if y shares the next-hop peer with any other leaf $z \in l_i(r)$. Any leaf z matching this condition is inserted into the Multiple Destination field of the MRI which includes y as destination address. All the leaves sharing the next-hop peer with y are removed from $l_i(r)$, together with y itself. The process goes on by grouping destinations until $l_i(r)$ is emptied. When the message is intercepted by a forwarder, the latter executes the same procedure, by using the destinations included in the MRI instead of computing $\lambda_i(r)$. The algorithm pseudocode is shown in Figure 4.14.

Procedure GIST Multicast

- 1: $l_i(r) = \lambda_i(r)$
- 2: **for all** $y \in l_i(r)$ **do**
- 3: $mri_mdest_y = \emptyset$
- 4: **for all** $z \in l_i(r) - \{y\}$ **do**
- 5: **if** $s_{iy} \cap s_{iz} \neq \emptyset$ **then**
- 6: $mri_mdest_y = mri_mdest_y \cup \{z\}$
- 7: **end if**
- 8: **end for**
- 9: send message to y with mult. dest. $z \in mri_mdest_y$
- 10: $l_i(r) = l_i(r) - (mri_mdest_y \cup \{y\})$
- 11: **end for**

Figure 4.14: Pseudocode of the GIST Multicast algorithm.

4.1.2 NetServ signaling

As we mentioned in Section 4.1.1, the actual signaling logic of an application making use of NSIS is implemented in the so called NSIS Signaling Layer Protocol (NSLP). NSLPs and NTLP

The NetServ NSLP is designed to allow a NetServ node to control the deploying of NetServ bundles into remote nodes of the network.

There are three type of messages in the NetServ NSLP:

- *Setup*: message used to trigger the setup of a bundle into the remote NetServ node. It carries the so called NetServ Control Message (NCM) and a list of key:value elements that specify the setup configuration. The key “properties” in the NetServ Setup message could be used to send a list of bundle-specific configuration parameters, organized as a key=value list, that can be retrieved by the bundle itself after the deployment. The setup message is shown in Figure 4.15.
- *Remove*: the remove message triggers the remove of the specified bundle from the NetServ node. It contains only the NCM specifying the bundle to remove.
- *Probe*: this message is used to retrieve information about the status of a specific bundle inside a NetServ node. It carries the NCM and a probe ID, which is a number specified from the client of the NetServ NSLP daemon and identifies the specific probe message.

Responses to these messages are given in the classic HTTP response code format: <Status code, Status message>.

Table 3.5 shows the list of all the configuration parameter that can be sent inside a the Netserv messages after the NCM. The Message column indicate whether the field is found in a Setup (S), a Remove(R) or a Probe (P) and if it is mandatory(*).

```

SETUP NetServ.apps.application_name NETSERV/0.1
dependencies:
filter-port:5060
filter-proto:udp
notification:
properties:visualizer_ip=1.2.3.4,visualizer_port=5678
ttl:3600
user:user_name
url:http://content-publisher.com/modules/module_name.jar
signature:4Z+HvDEm2WhHJrg9UKovwmMutxGibsA71FTMFykVa0Y\xGclG8o=
<blank line>

```

Figure 4.15: NetServ SETUP example message.

Parameter	Message	Description
dependencies	S	List of bundle dependencies necessary to run the bundle
filter-port	S	Destination port for the packet that will be intercepted by the bundle
filter-proto	S	Type of protocol for the packet that will be intercepted by the bundle
notification	S, R	XML-RPC URL invoked after the setup of the module
node-id	S, R, P	Address of the explicit destination of a message (end-to-end signaling)
probe	P*	Code of the probe request
properties	S	List of key=value properties that are passed to the bundle
ttl	S*	Time to live of the bundle (s)
signature	S*, R*, P*	Digital signature of the message
user	S*, R*, P*	The user owning the NetServ Container

Table 4.1: List of NetServ message parameters.

All the bundles deployed inside a NetServ container can use the API of the NetServ NSLP daemon to forge a message and send it to a remote NetServ node. By this mean a NetServ bundle can control the deployment and the execution of other bundles throughout the network.

All NetServ messages use the transport service provided by the underlying GIST layer. Hence our off-path extension, and in particular the Hose dissemination, can be exploited to send probe messages reporting the status of the CBs and the LOIBs which lies few hop from the path between data sources (Figure 4.1, message 3). Figure 4.16 shows the joint use of probe messages and off-path signaling in the Ares scenario:DC1 and DC2 are data centre hosting two virtual machines, VM1 and VM2, which are NetServ node hosting a CB. R1 and R2 are backbone routers. N1 and N2 are NetServ-enabled data repositories. The probe message is delivered from node N1 to node N2, but with the Hose dissemination it reaches also the data centre behind backbone router R1 and R2, allowing N1 to retrieve information about the data contained in the CBs inside VM1 and VM2.

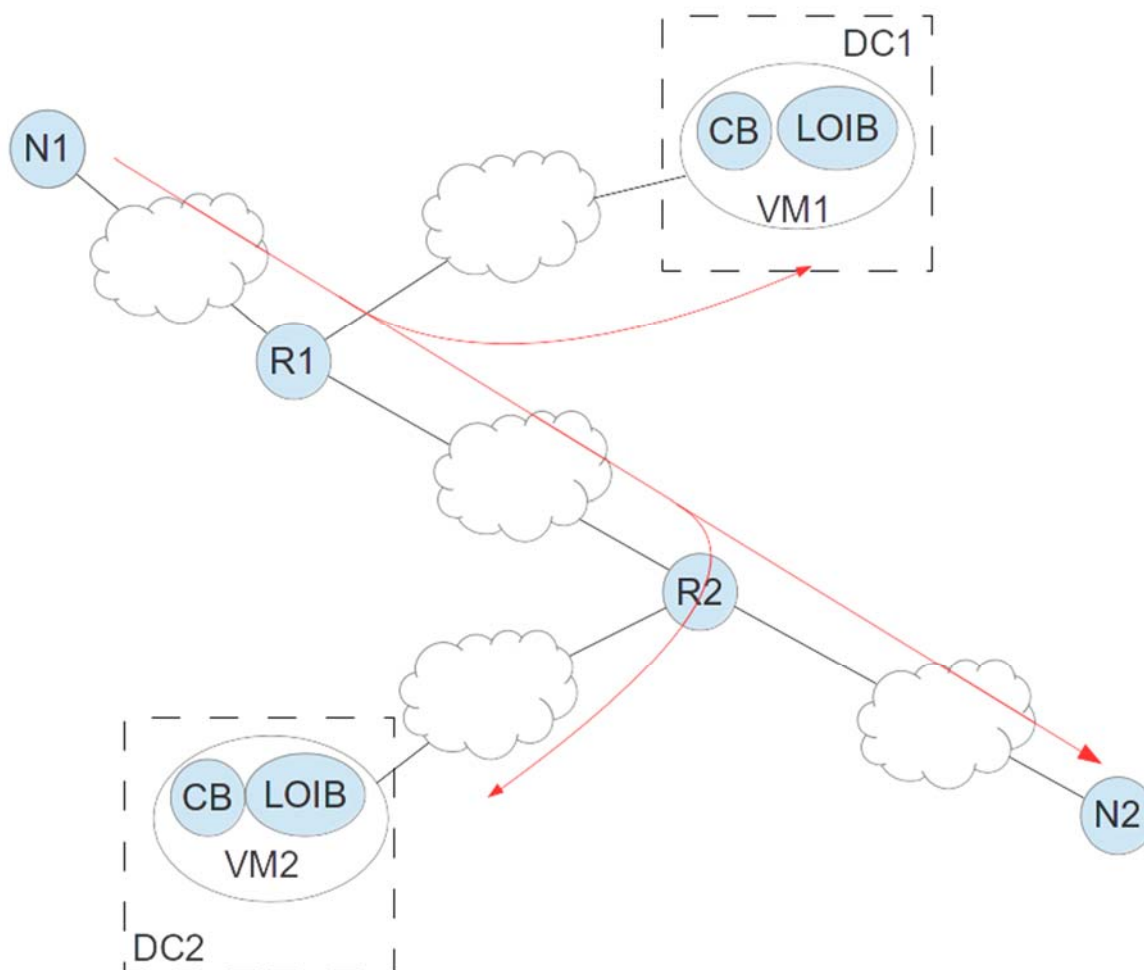


Figure 4.16: NetServ + off-path hose within the Ares scenario.

The same procedure can be used also to probe the presence of the LOIB bundle inside the data centre DC1 and DC2.

In more details, we implemented a resource discovery framework leveraging on both the off-path extension of NSIS, and specifically of GIST, and the NetServ NSLP. The NetServ SETUP message is used to implement a query to a NetServ bundle. In fact SETUP message could carry a list of setup properties, formatted as <key,value> pair. When a NetServ node receives a SETUP for a bundle that is already installed, it passes this list of properties to the bundle itself. Therefore the GCM can use the properties list of the SETUP message to pass to the bundle the list of desired contents (in the case of the CB) or the resource requirements (in the case of the LOIB). Upon receiving this information, the bundle can use either the NSIS signaling API or the RESTful interface of the signaling source to answer the query.

The second message we used to implement the resource discovery is the PROBE message. It is used to locate the instances of a specific bundle in the desired off-path domain. Figure 4.17 shows the inter process communication between the involved framework entities of a forwarder node, when it receives a PROBE message, routed through an hose by GIST. The involved entities are the NTLP (GIST), the NetServ NSLP, and the bundle managing the data.

After the NTLP handshake, described in [3], the NSLP data are delivered to GIST, which provides to send them to the NSLP. The NSLP identifies the message type, extracts the application parameters and passes them to the NetServ bundle (steps 1-4 in Figure 4.17). The bundle processes the query and returns the response code to the NSLP, which provides to store it (steps 5-7 in Figure 4.17). Meanwhile, the NTLP is in charge of forwarding the Query message through the hose. It identifies the next-hop destinations and starts a GIST handshake with them. Each hand-shake triggers the delivery of a MessageStatus from the NTLP to the NSLP (Figure 4.17, step 9). This message notifies the NSLP that the Query has been forwarded toward a new destination and that a Response message is expected from that destination. By this mean, the NSLP can maintain a counter, which stores the number of App-Response messages the NSLP must receive before it can forward its response upstream (steps 8-10 in Figure 4.17).

During the GIST handshake, if a next-hop destination has received the same SETUP message from another node, it will answer the GIST query message with a GIST off-path-duplicate error, thus aborting the handshake, as illustrated in Section 4.1.1.2. This error is reported to the NSLP using another MessageStatus, which triggers the decrease of the response counter (steps 11-13 in Figure 4.17). If the NTLP starts m GIST handshakes and receives k GIST off-path-duplicate errors, the NSLP will wait for $m-k$ responses before forwarding its response upstream (Figure 4.17, steps 1-16). Each time the NSLP receives an App-Response, it adds the received stack to a local response stack and decrements the response counter (steps 17-19 in Figure 4.17). Upon receiving the last response, which decreases the response counter to zero (i.e., all expected responses arrived), the NSLP adds its own ResponseWrapper to the stack and forwards the Response message upstream (Figure 4.17, steps 20-22), i.e. to the GIST peer from which it initially received the Query.

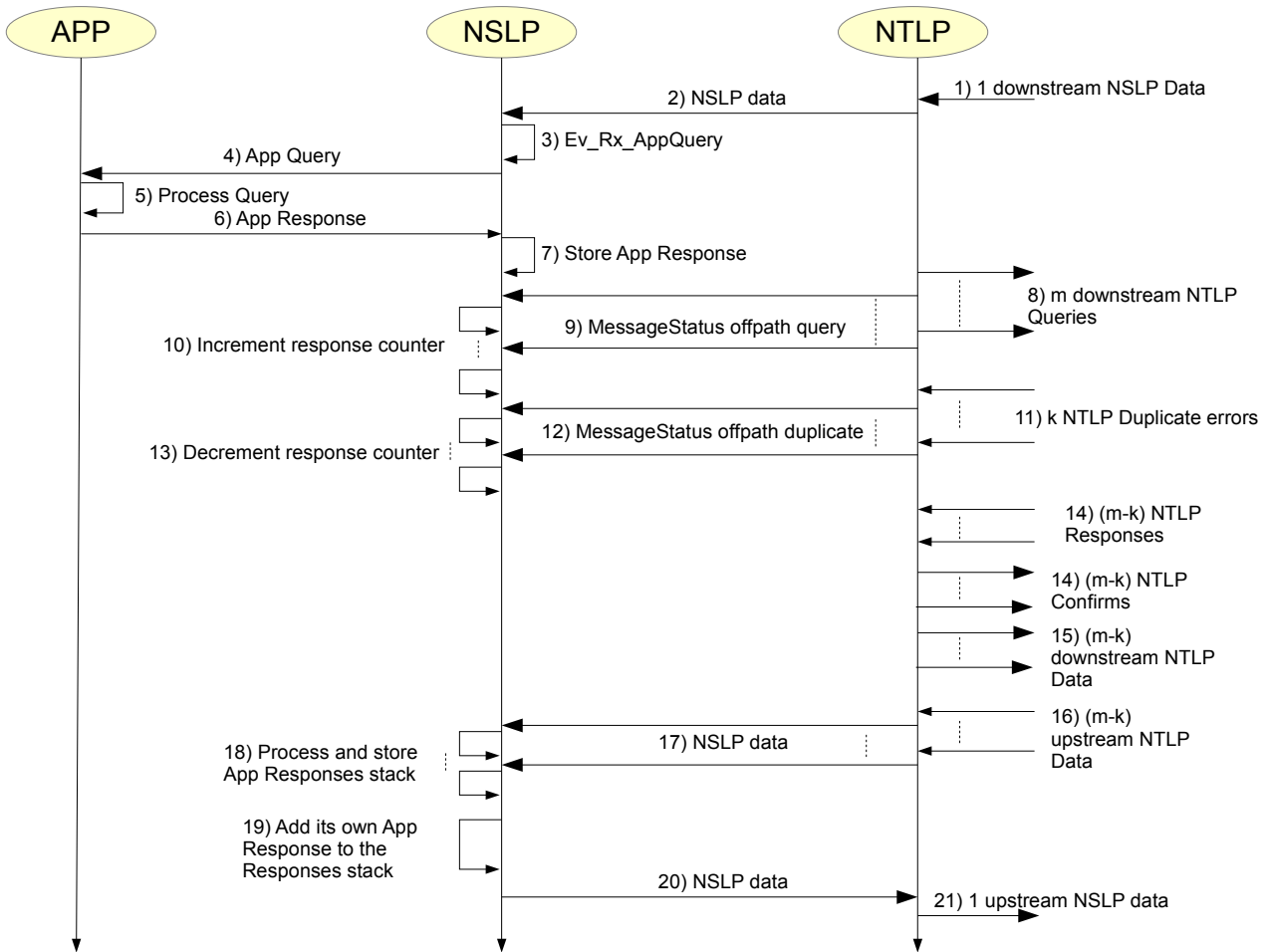


Figure 4.17: Sequence diagram describing the interactions between the application, the NSLP, and the NTLP inside a node.

During the PROBE responses collection, the NSLP is also in charge of maintaining an integer field, called Depth field, which is present in each ResponseWrapper object it receives. This object, along with the Node-Id object and the ResponseCode, allows the signaling initiator to compute a data structure which contains both the bundle status information and the hose topology. During the steps 7 in Figure 4.17, the NSLP initializes its ResponseWrapper with data provided by the local NetServ application (step 6 in Figure 4.17), its own Node-Id, and a Depth field containing the value 0. During the step 19 of Figure 4.17, the Depth field of all the received wrappers is increased by 1, before adding it to the local stack. Figure 4.18 shows the stacks contained in each Response message exchanged when a Query is sent through a hose with radius 2 from R1 to R5. The response code field is omitted for readability. It is clear that, using this simple rule on the tree of Figure 4.18, the router R1 can infer the hose structure which is depicted with explicit indentation. In this figure, the label DC_i indicates a data centre (thus an off-path node), whereas a label R_j indicates a router.

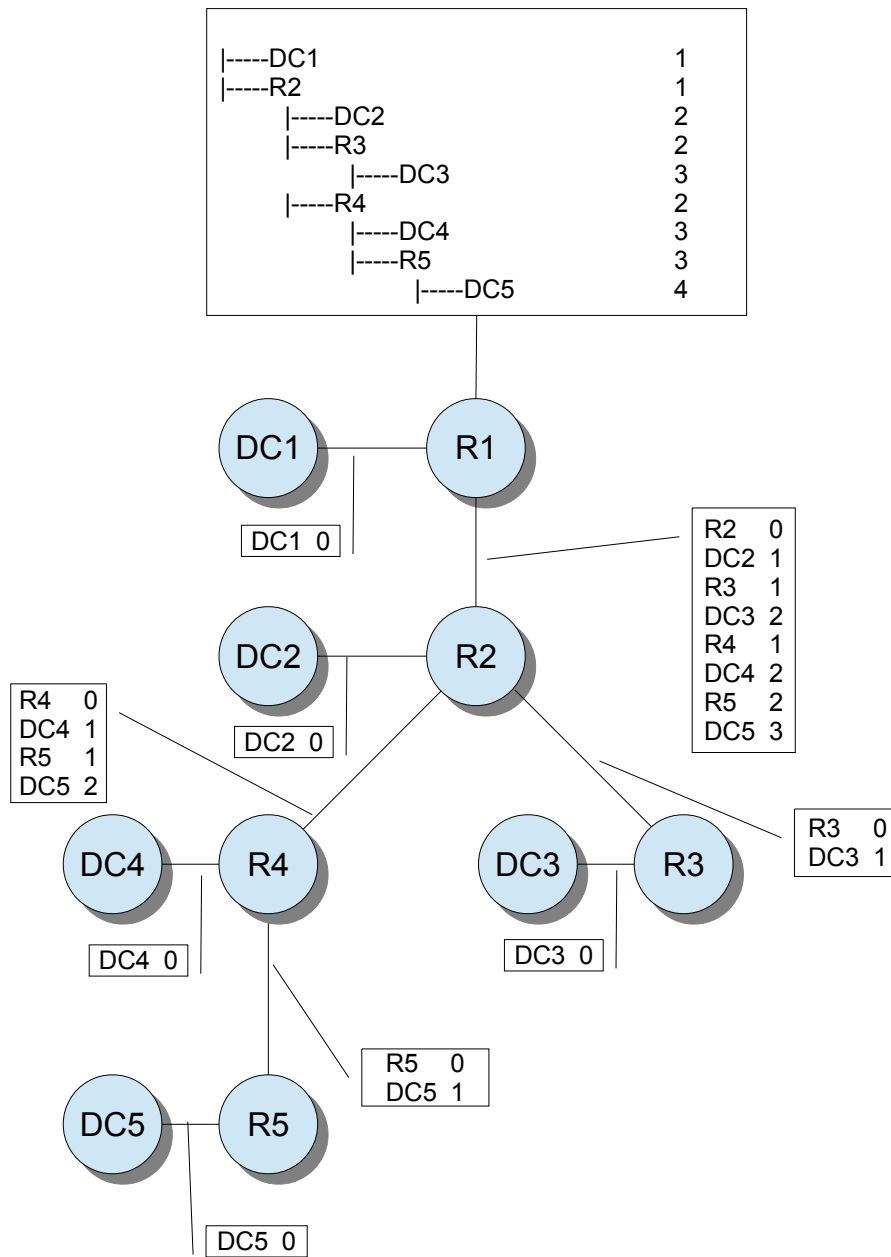


Figure 4.18:PROBE responses aggregation in a hose signaling session from R1 to R5 with radius of 1 IP hop.

With reference to the ARES scenario, we designed a quite refined search process, consisting of three steps. The first two steps use the hose signaling illustrated above.

In the first step, the GCM triggers a hose signaling on the VM repository application towards the node storing the genomic data sets to be processed. In addition, a further signaling is triggered also on the repository storing the auxiliary files, needed for pipeline computation, towards both VM repository and genomics data sets storage. This first set of hose signaling carries a PROBE message for the LOIB bundle. Hence it is used to

identify the potential datacentres close to any of the three involved paths. When the application which has sent the signaling obtains the Response messages, it forwards it to the GCM, which can thus know all the identities of the datacentres able to support the requested processing.

The further step is to trigger, always from the GCM to the same nodes of the previous step, additional hose signaling to identify the potential locations (original repository or caches) where cacheable contents (VM image files or auxiliary files) could be stored. In this case, the searched content is carried in the properties list of the SETUP message, along with the address of the RESTful interface provided to collect the responses (see details of RESTful messages/interfaces in the next Section 4.1.3). Again, when the application which has sent the signaling obtains the responses through its RESTful interface, it forwards it to the GCM, which can thus know all the locations where the data needed for the computation reside.

In the third step, the GCM communicates the identities of the nodes storing data to each LOIB collected in the first step, which will measure the distance (in terms of IP hops or network latency) from each data location. Note that in most of cases this measure is already present in the node at the GIST layer, since it is collected during the initial gossip discovery phase of the GIST protocol itself (see Section 4.1.1.1). Thus, the application bundle (LOIB) will simply use a GIST API to locally retrieve this information. Then, the LOIB selects, for each type of data (genomics data set, auxiliary files, and VM image files) the location at the lowest distance, and returns it to the GCM. When the GCM has retrieved all the response, it has a quite complete picture of the network, including computing resources, network distance, and content availability, and it can run an optimization function to select the data centre for executing the needed processing.

4.1.3 RESTful signaling

As for the simple communication between the different entities of the Ares Framework (Medical Tool, GCM, LOIB, GPVM) the use of NSIS+NetServ is not always the best choice. The Medical Tool and the GPVM are not NetServ entities, hence it is impossible to use the NetServ NSLP to create and send message. Furthermore part of the messages exchanged between these entities are very small. For this reasons we decided to demand part of the signaling to the classic RESTfull paradigm, along with the JSON metadata format. RESTful interfaces are implemented inside the GCM, the LOIB and the GPVM. All these interfaces are implemented embedding Jetty servlets inside a Java application. The next section will detail the messages exchanged during all the phase shown in Figure 4.1, which makes use of the RESTful paradigm.

4.1.3.1 Ares Medical Tool and GCM messages

There are two kind of messages that Ares Medical Tool can send to the GCM:

SubmitTask: This message is sent via an HTTP POST to the URL <http://ares.gcm.eu:2311/submitTask>. The message carried by the POST is forged from the following JSON schema:

```
{“taskID”:String, “Prio”:String, “Analysis”:String, “InputURL”:String, “InputSample”:[{“folder”:String}]}
```

When the GCM receives this message it will start all the necessary routines to select the appropriate PoP and perform the selected analysis. The “TaskID” is the unique identifier that represent the request of the medical personnel. The “Prio” field is a code expressing the maximum service time allowed for the task. It will be mapped into a specific service class. The “Analysis” field specifies the type of analysis request by the medical personnel. The “InputURL” specifies the URL of the repository where the patient genome data resides. The URL must point to a parent directory containing patients genome reads divided in different folders, one for each patient. The GCM will include in the processing only the patient folders specified in the array “InputSample”.

getTask: This message is sent via an HTTP POST to the URL <http://ares.gcm.eu:2311/getTask>. It is used by the Ares Medical Tool to ask for updates on the status of a list of tasks. It carries a JSON messages which follows the subsequent schema:

```
{“TaskList”:[{“TaskId”:String}]}
```

that is a simple list of task identifier.

Both the messages receive as a response from the GCM a 200_OK message which carries a JSON payload which follows the subsequent schema:

```
{“TaskList”:[{“TaskID”:String, “Prio”:String, “Analysis”:String, “InputURL”:String, “InputSample”:[{“folder”:String}], “start”:long, “stop”:long, “LOIBAddress”:String, “output”:String,}]}
```

Each element of the array is almost equal to the schema of the submitTask message, but with the addition of four fields: the “start” and “stop” field that are used for test purpose, the “LOIBAddress” that represents the address of LOIB in the selected PoP, and the “output” field that is filled with the path of the output file when the processing is finished. The combination of the LOIB address and the output path will be fetched by the Ares Medical Tool to the medical personnel in the end of the computation.

4.1.3.2 GCM and LOIB messages

SubmitTask: The GCM sends to the LOIB of the selected PoP a single message via the RESTful API. This message is used to submit the processing with the specific configuration of resources and input files. The message is sent to the URL <http://x.x.x.x:2205/submitJob> with the HTTP POST method. It carries a JSON payload following the subsequent schema:

```
{“TaskID”:String, vmURL:String, minRAM:String, minCPU:String, minDisk:String, vmFormat:String, vmContainer:String, vmName:String, Downloads:[{URL:String, Type:String, Subfolder:String}]}
```

Parameter	Type	Description
TaskID	String	Unique identifier of the Task
vmURL	String	URL of the Virtual Machine image to download
minRAM	String	Minimum amount of memory requested for the processing (MB)
minCPU	String	Minimum number of core requested for the processing
minDisk	String	Minimum amount of storage requested for the processing (GB)
vmFormat	String	Format of the Virtual Machine image
vmContainer	String	Hypervisor type requested for the Virtual Machine image
vmName	String	Name of the Instance that will be created
URL	String	URL of the input/auxiliary file to download
Type	String	Type of file to download (INDEX, INPUT, CHROMOSOMES)
Subfolder	String	Patient subfolder that will store patient genome (only valid when Type=INPUT)

Table 4.2: List of the fields of a submitTask JSON payload.

UpdateStatus: The LOIB can send to the GCM periodic updates on the status of a task sending a message to the address `http://ares.gcm.eu:2311/updateStatus?taskID=0000&status=newStatus`. The parameters will be parsed from the URL and updated inside the GCM database. When the processing is finished the LOIB can use the same message also to communicate the path of the output file using the address `http://ares.gcm.eu:2311/updateStatus?taskID=0000&status=FINISHED&output='/url/target/to/file'`.

4.1.3.3 LOIB and GPVM messages

During the GPVM startup, a Java HTTP server is started as an upstart job. The server implement the RESTful paradigm and provides the main communication interface between the LOIB and the GPVM. All the processing operation are triggered and controller through RESTful command. Figure 4.19 sketches the flow of messages between the LOIB and a GPVM during the whole life-cycle of a processing job. Each message in Figure 4.19 reports the URL and the HTTP message type used.

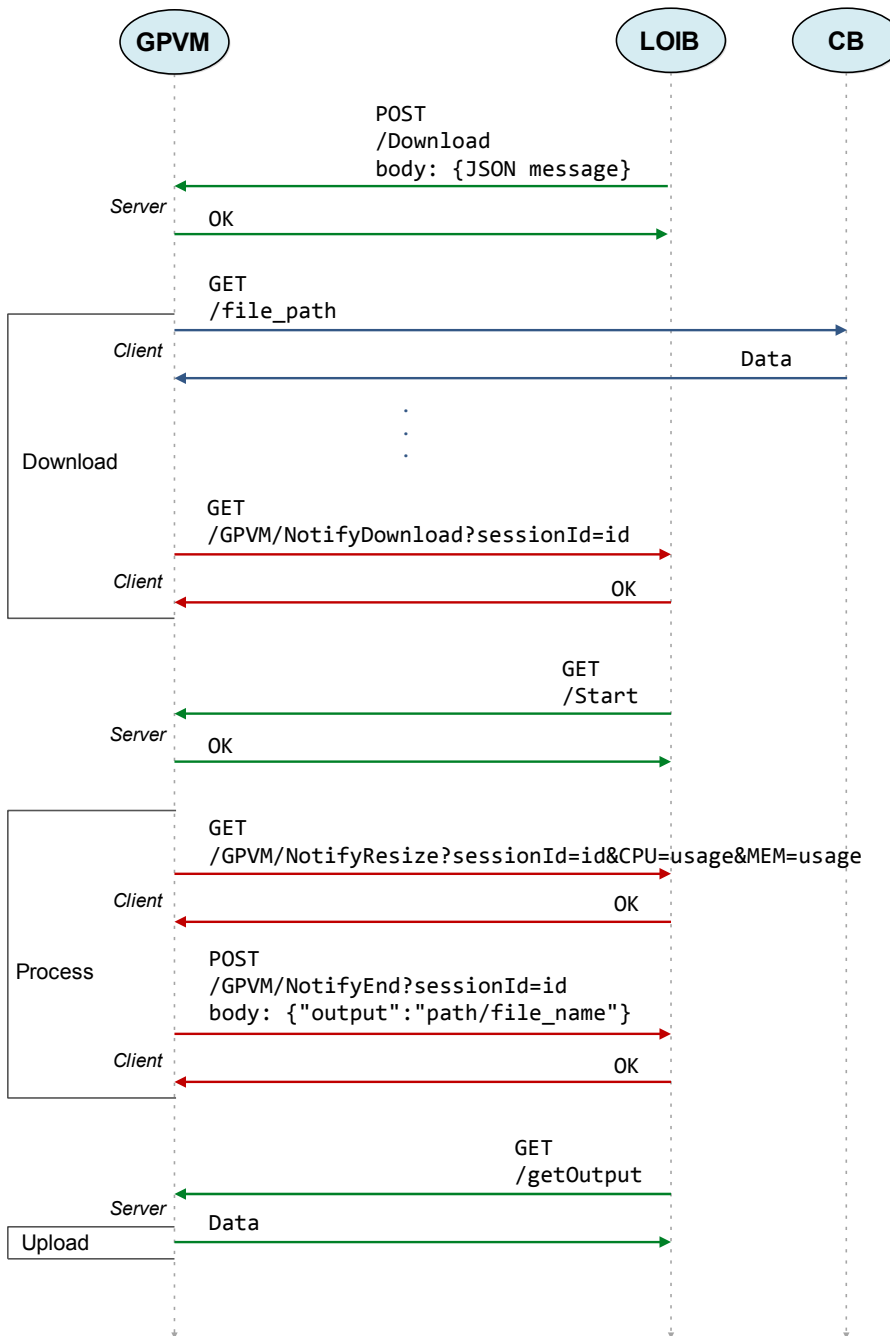


Figure 4.19: Messages flow between a GPVM and the LOIB.

There are several messages involved in the communication:

Download: Through this message the LOIB communicate to the GPVM the complete list of the files to download. This list will contain input files provided from the medical user, genome indexes, whole genome file, chromosomes and annotation. The HTTP POST message carries a JSON payload that follows the schema:

```
{“sessionId”:String, “Downloads”:[ {“URL”:String, “Type”:String, “Subfolder”:String} ]}
```

Parameter	Type	Description
sessionId	String	Unique identifier of the GPVM.
URL	String	URL of the input/auxiliary file to download
Type	String	Type of file to download (INDEX, INPUT, CHROMOSOMES)
Subfolder	String	Patient subfolder that will store patient genome (only valid when Type=INPUT)

Table 4.3: List of the fields of a Download JSON payload.

notifyDownload: Through this message the GPVM notifies to the LOIB that the download procedure is finished and that it is ready to start the processing.

start: This message, sent from the LOIB to the GPVM triggers the execution of the processing pipeline.

notifyResize: Through this message the GPVM notifies to the LOIB that the most intensive part of the processing is finished and that the LOIB can try to reallocate part of the resources allocated to the GPVM. The GPVM sends to the LOIB the amount of RAM and the percentage of used CPU core as parameters, hence the LOIB can find a new resource arrangement for the specific GPVM. Session identifier is used to authenticate the GPVM.

notifyEnd: Through this message the GPVM notifies to the LOIB that the pipeline processing is finished and that the output file is available for the download. The output file path is sent in the JSON payload illustrated in Figure 4.19.

getOutput: Through this message the LOIB request the output file to the GPVM in order to store it for future request of the medical personnel.

4.1.4 Preliminary Experimental and simulation results

In this subsection, we present the results of an experimental campaign executed on a local testbed consisting of 60 nodes. The network reflects the PoP-level topology of the Géant network at January 2014. Our topology is composed of 41 PoP routers, represented as a single node, and by 19 datacentres. Figure 4.20 shows the experimental topology we used, which has been derived by the physical PoP-level topology Géant topology depicted in Figure 4.21. Each gray node in Figure 4.20 includes a PoP router connected with a single data centre (not shown to preserve figure readability), laying 1 IP hop away from the PoP router. In our local testbed, distributed over multiple physical hosts, each logical node has been implemented by means of a single virtual

machine (VM) running Ubuntu Server v.10.04. Each VM models either a PoP router or an entire data centre (see Figure 4.2). Clearly, in the network topology we considered there are two types of nodes: core nodes, which are PoP routers connected with at least another two nodes, or stub, which are PoP routers connected only to one PoP core node (e.g. MT in Figure 4.20), or data centres.

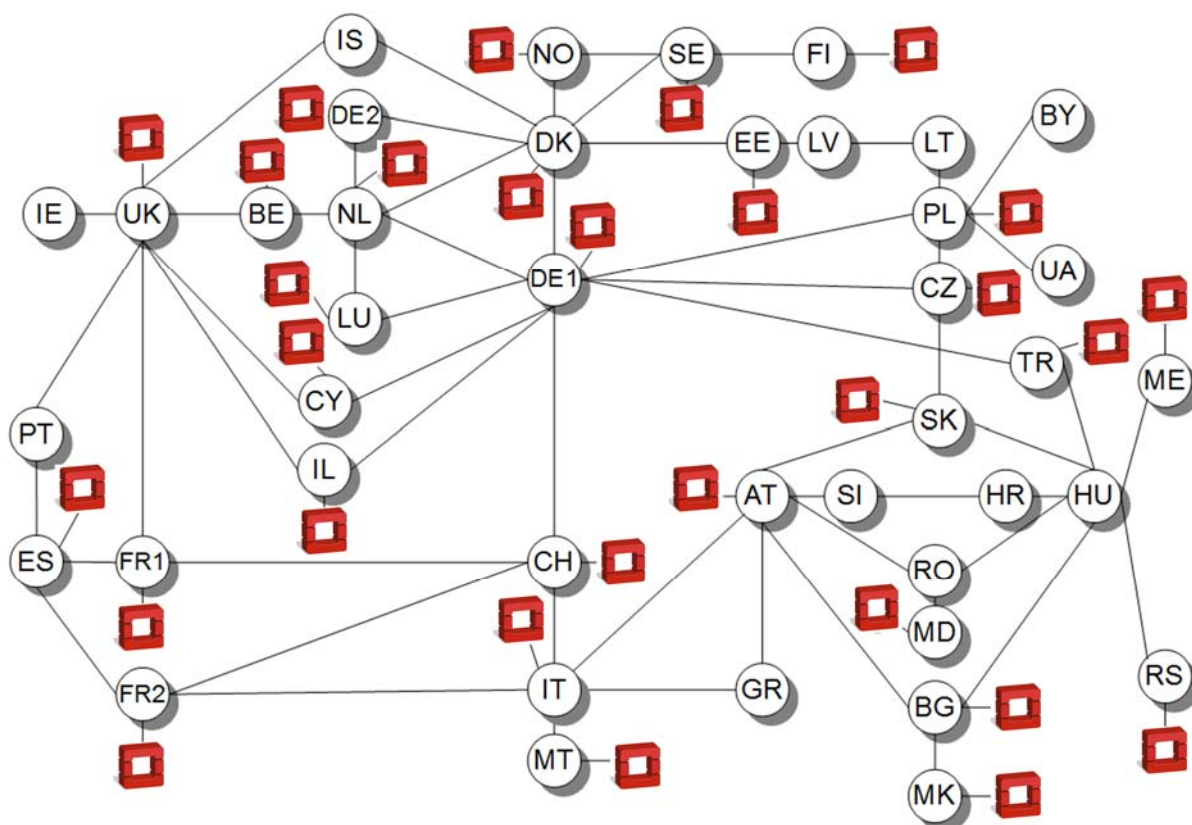
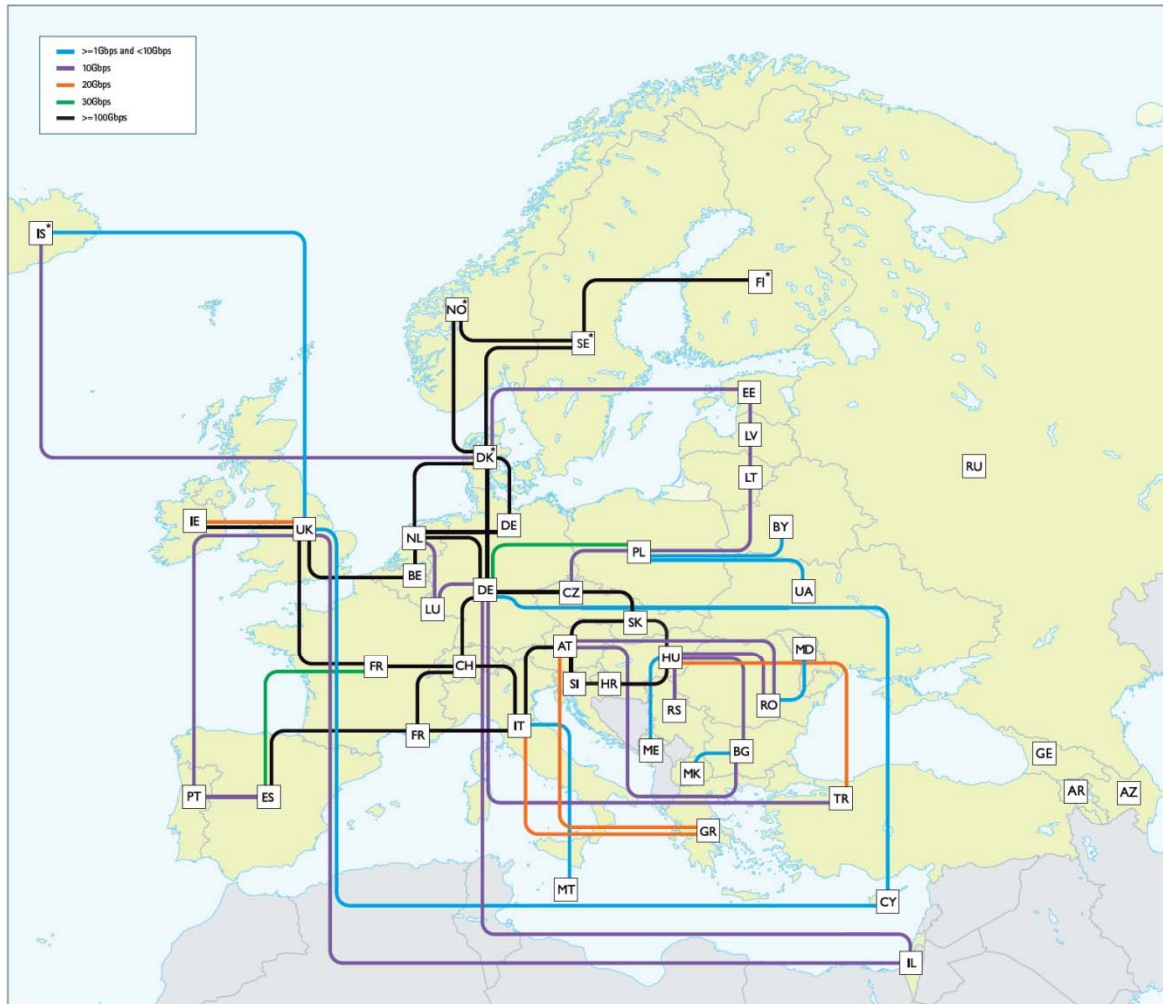


Figure 4.20: Géant PoP-level logical topology (January 2014). Grey nodes include a data centre, laying 1 IP hop away from the PoP. The relevant physical PoP-level topology is depicted in Figure 4.21.

Our proposals have been implemented as an NSIS-ka extension [6]. We have implemented two overlay networks:

- *Full overlay topology*: All nodes execute NSIS.
- *Sparse overlay topology*: we have disabled NSIS from 12 core nodes. Thus, these nodes act as legacy IP routers, not running NSIS. All the other nodes are NSIS compliant. This configuration is aimed at evaluating the impact of an incremental deployment of our extended NSIS. For the sparse topology, we define the parameter P which indicates the number of the NSIS enabled nodes (overlay nodes) divided by the total number of IP nodes. For the case analyzed here, $P=0.8$:

Now, we present the results relevant to network discovery, in order to compare the proposals illustrated in section 4.1.1.1, and off-path distribution, in order to evaluate the performance of algorithmic alternatives presented in section 4.1.1.2.



GÉANT connectivity as at January 2014. GÉANT is operated by DANTE on behalf of Europe's NRENs.



*Connections between these countries are part of NORDUnet (the Nordic regional network)

Figure 4.21: Géant PoP-level physical topology (January 2014). The connections with GE, AR, AZ, and RU are not shown, thus we have not included these nodes in the logical PoP-level topology depicted in Figure 4.20.

4.1.4.1 Network discovery results

The performance of the network discovery algorithms was evaluated by executing two sets of tests. The first one allowed evaluating performance during the transient phase, when all nodes are turned on. The second one was used to evaluate the overhead in the steady phase, when the GIST node discovery has been completed. We define the convergence time t_i of the node i as the time taken for executing the transient phase, that is when at least one complete gossip session has been executed with all NSIS nodes for the Leaf-based solution, and with all the NSIS neighbors for the One-hop based solution, respectively. Hence, the network convergence time is $\max_{i \in V} \{t_i\}$. During the transient, we have evaluated also the overall network overhead.

We compared the network discovery performance of our proposals with that of the Random solution [4], where the PTG and PTS lists are randomly selected in the complete set of PEs stored in each node, using the Q-forward policy.

Figure 4.22 shows the convergence time and signaling overhead during the transient phase as a function of H , the number of shared PIs in a gossip message. In all experiments, differently from the Random discovery, both the Leaf and One-hop (labeled as “OH”) solutions provide satisfactory convergence time and signaling overhead. The label “OH, $B = \infty$ ” indicates the usage of a memory size larger than the number of PEs in the network, and the label “OH, $B = 10$ ” indicates memory size of 10 PEs.

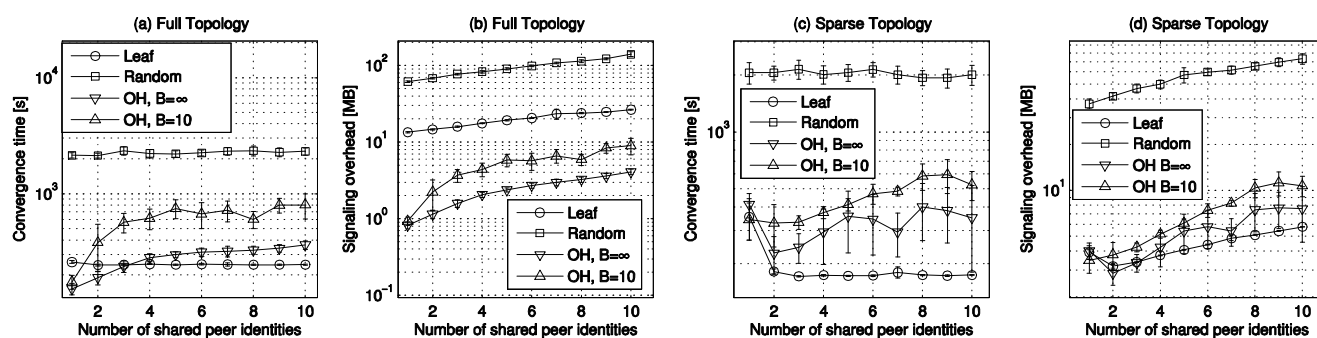


Figure 4.22: Transient experimental results vs. size of the PTS list: (a) convergence time on full topology, (b) signaling overhead on full topology, (c) convergence time on sparse topology, and (d) signaling overhead on sparse topology. For One-hop (OH), $B = \infty$ indicates a memory size larger than the number of PEs in the network, whilst $B = 10$ indicates a memory size equal to 10 PEs. 95% confidence intervals are shown.

From Figure 4.22.a and Figure 4.22.c it emerges that when the Leaf solution is used both on sparse and full topologies, the convergence time does not decrease when more than two PIs are shared. This result is generally valid since sharing 2 identities is enough to fill the PeT of each peer with a sufficient number of uncontacted peers. This way, uncontacted peers can be selected as a PTG in most of subsequent cycles. By using the One-hop solutions, the convergence time increases with the number of shared identities H . Sharing many peer identities makes the set of PEs, selectable for the next PTG, large. In addition, the number of possible PTGs is much higher than the number of next hop routers. When these routers are also NSIS nodes, it

is disadvantageous to test a large set of PEs, most of which are unreachable due to the use of the Q-drop policy.

When the paths to most of PTGs share the same next hop, the interception of other next hop routers happens less frequently. This process is even hampered by the presence of multiple IP addresses of GIST peers and by the small size of B . In fact, multiple destination IP addresses could cause packets sent for contacting a neighboring peer passing through another GIST node, and the initiator would label this peer as unreachable.

Only when the identity of this deemed unreachable peer will be associated with another of its IP addresses, the initiator will try to contact it again.

The process goes on until the “right” IP address for that peer is communicated to the initiator. When the B value is lower than the number of NSIS nodes in the network, some unreachable nodes could be deleted from the memory, and, if they are shared again by a different peer, they are regarded as uncontacted peers and candidate for a new gossip session. Clearly, these operations can further increase the convergence time.

For the full overlay topology, we observe that when the number of shared peer identities is 1 or 2, the “OH, $B = \infty$ ” is preferable. In a sparse topology, the Leaf solution is preferable in all situations. Note that in a sparse topology the discovery time of the One-hop solution could slightly increase. In order to better highlight this behavior, Figure 4.23.a shows the discovery time of all implemented solutions, by black solid lines for the full topology, and by red dotted lines for the sparse topology. As expected, in the sparse topology the convergence time of the Leaf and Random solutions slightly decreases, due to a reduced number of NSIS nodes. Since the Leaf solution is designed to discover leaves, a significant decrease of the convergence time is expected when the number of leaves decreases. Instead, in the case of the One-hop solutions, while the implementation with $B = 10$ slightly benefits from the reduced number of NSIS nodes, such a result cannot be achieved when $B = \infty$. In this case, a slight performance degradation is due to the increased average number of nodes being 1 GIST hop away, as it happens in the sparse topology.

Figure 4.22.b and Figure 4.22.d show the signaling overhead during the transient phase in the full and sparse topologies, respectively. The overhead of both Leaf and Random solutions increases with the number of shared PEs, as the payload of gossip messages. In addition, when duration of the transient phase increases (Figure 4.22.a and Figure 4.22.c), the amount of exchanged NSIS traffic increases as well (Figure 4.22.b and Figure 4.22.d).

In a full topology, the overhead generated by the One-hop solutions is significantly lower than that generated by the Leaf and Random solutions, since the One-hop solutions limit the scope of messages to 1 GIST hop, corresponding in this case to 1 IP hop. Differently, in a sparse topology, the average distance of the GIST nodes being 1 hop away increases, thus causing an increase of the overhead. In this case, the Leaf solution outperforms the One-hop. Leaf and OH solutions always outperform the Random one.

As for the steady state, we have both executed experiments and defined a theoretical model for the full topology. We denote by δ the IP network diameter, and by ξ_{ij} the probability that a peer i selects an LP $j \in L_i$ as a PTG. The length of the paths p_{ij} in the PaT of i is modeled as a discrete random variable distributed in the

range $[p_i^{min}; p_i^{max}]$, with mean μ_i and variance σ_i^2 . The overhead generated by the Leaf solution during a gossip session between nodes i and j is equal to:

$$\phi_{ij} = p_{ij}q + (r + a) \sum_{i=1}^{p_{ij}} i = p_{ij}q + (r + a) \frac{p_{ij}(p_{ij} + 1)}{2} \quad (8)$$

thus, the average overhead generated by the i th node is:

$$\Phi_{i,Leaf} = \sum_{j \in \mathcal{L}_i} \xi_{ij} \phi_{ij} = q\mu_i + (r + a) \left(\frac{\mu_i}{2} + \frac{\mu_i^2 + \sigma_i^2}{2} \right) \quad (9)$$

where $\xi_{ij} = 1/M_i$ is assumed to be in the steady state.

We also assume that the mass probability function of p_{ij} is uniformly distributed between $p_i^{min}=1$ and p_i^{max} . Thus, by a simple mathematical derivation, it follows that

$$\Phi_{i,Leaf} = q\mu_i + (r + a) \left[\frac{\mu_i}{2} + \frac{1}{2} \left(\mu_i^2 + \frac{1}{3}(\mu_i^2 - \mu_i) \right) \right] \quad (10)$$

Thus, the total network signaling overhead is:

$$\Phi_{Leaf} = \sum_{i=1}^K \Phi_{i,Leaf} = \sum_{i=1}^K \left[q\mu_i + \frac{(r + a)}{3} (\mu_i + 2\mu_i^2) \right] \quad (11)$$

In order to provide an upper bound depending on global network parameters only (i.e. number of nodes K and network diameter δ), without requiring any knowledge of μ_i , in (8) we replace $p_{ij} < \delta$ by δ . Thus, it follows that:

$$\Phi_{Leaf,UB} = K\delta \left[q + (r + a) \frac{(\delta + 1)}{2} \right] \geq \Phi_{Leaf} \quad (12)$$

which is a useful expression for evaluating the GIST overhead. The overhead rate is found by dividing both $\Phi_{Leaf,UB}$ and Φ_{Leaf} by the gossip period T_{gossip} . An estimate of the bandwidth consumption during the transient phase is obtained by normalizing $\Phi_{Leaf,UB}$ and Φ_{Leaf} by the minimum value of T_{gossip} , and an estimate of the steady state bandwidth consumption is obtained by normalizing them by the maximum value of T_{gossip} . In fact, in the steady state PeT does not change, and thus T_{gossip} increases up to its maximum value (see Figure 4.5). The values of $\min T_{gossip}$, $\max T_{gossip}$, and $\max Counter$ are equal to 5s, 40s, and 10 iterations, respectively. The length of Registration (q) and Response (r) messages is 156 bytes when a single PI is shared. It increases by 28 bytes for any further PI shared. The length of the ack message (a) is 112 bytes.

This model can be easily adapted to evaluate the mean overhead of the Random solution. It is enough to replace, in the equations above, the mean IP path length, μ_i , with the mean IP distance, γ_i , from the node i towards all other PEs.

For the One-hop solutions, we can provide a more general model. Let v_i be the mean IP distance between the i th GIST node and its neighboring GIST peers. Since each peer executes a gossip session only with a neighboring peer, the resulting total signaling overhead is:

$$\Phi_{One-hop} = \sum_{i=1}^K \Phi_{i,One-hop} = \sum_{i=1}^K (q + r + a)v_i \quad (13)$$

As in the full topology $v_i = 1$, the total signaling overhead is:

$$\Phi_{One-hop,full} = K(q + r + a) \quad (14)$$

Figure 4.24.a shows the experimental signaling bandwidth consumed by all strategies in the steady phase, as a function of H . For all solutions, bandwidth increases linearly with the number of shared identities. It emerges that the most efficient schemes are the One-hop ones, due to the limited scope of their signaling exchanges. The Random solution consumes less bandwidth than the Leaf one, since the PTG in the Random solution has a mean distance $\gamma_i < \mu_i$. However, the convergence time of the Random solution is higher than the one of the Leaf solution by one order of magnitude. In any case, considering the results relevant to the transient phase, which suggest to use $H = 2$, the total signaling overhead over the whole network is 70 Kbit/s, which is definitely negligible.

Figure 4.24.b shows the signaling bandwidth consumption of the Leaf solution versus H , including theoretical results labeled as “theor.” (11), the upper bound labeled as “UB” (12), and the experimental results, labeled as “exp.”. Results are shown for both the transient phase, labeled as “TP”, and the steady phase, labeled as “SP”. The first comment is that theoretical and experimental results of the steady state are in an excellent agreement, and also the rough upper bound is very close to the experimental results. As for the transient phase, the difference between theoretical and experimental results is due to the very initial phase of the discovery process. In this phase, not all exchanged PIs are LPs, thus some sessions span over paths shorter than those used in the steady state, thus making the overall signaling bandwidth lower than the theoretical one. The upper bound is about twice the experimental results, thus acceptable for a coarse estimation.

Figure 4.24.c shows the theoretical and experimental signaling bandwidth of the Random solution as a function of H , in both transient and steady phases. Labels are the same as in Figure 4.24.b. The agreement between the theoretical and experimental results in the steady state is excellent. For what concerns the transient phase, the overestimation of the theoretical model is due to the approach used for selecting the PTG and the PTS list by the Random solution. In fact, the mere random strategy could lead to stalls, in which a GIST node does not receive new PIs for several consecutive cycles and starts increasing T_{gossip} , thus decreasing the overall signaling bandwidth.

Figure 4.24.d shows the theoretical and experimental signaling bandwidth of the One-hop solutions versus H , including both theoretical and experimental results for both the transient and steady phases. Differently from the Leaf and Random solutions, an excellent agreement between the theoretical model and experimental results for the transient phase is found, for both values of B . Instead, the theoretical model of the steady phase significantly underestimates the bandwidth consumption, which is close to the estimate for the transient phase. This is due to the management of T_{gossip} in the One-hop solutions. In fact, differently from the other two solutions, multiple IP addresses of neighboring peers cause frequent changes in PeT. This problem could be avoided by sharing all IP addresses associated with a PI in each gossip message, but we did not proceed in this way since the resulting overhead would significantly increase. Thus, for most of the time, $T_{gossip} = \min T_{gossip}$. In addition, also the usage of a limited buffer could trigger a PeT change, and thus the usage of the minimum T_{gossip} value. The use of a large buffer for storing unreachable peers produces small benefits, since the probability that a node increases its T_{gossip} value is slightly higher. Thus, the overhead of the solution “OH, $B = 10$ ” is slightly worse than that of the solution “OH, $B = \infty$ ”.

Figure 4.23.b shows the consumed bandwidth in the steady state, found experimentally, of all implemented solutions, in both the full and the sparse topologies. As expected, for sparse topologies, the bandwidth overhead of Leaf and Random solutions is lower. This is not true for the One-hop solutions. In fact, although K decreases, the average IP distance between neighbors increases (see (13)). The resulting effect is an increase of the bandwidth overhead, even larger than the one generated by the Leaf solution.

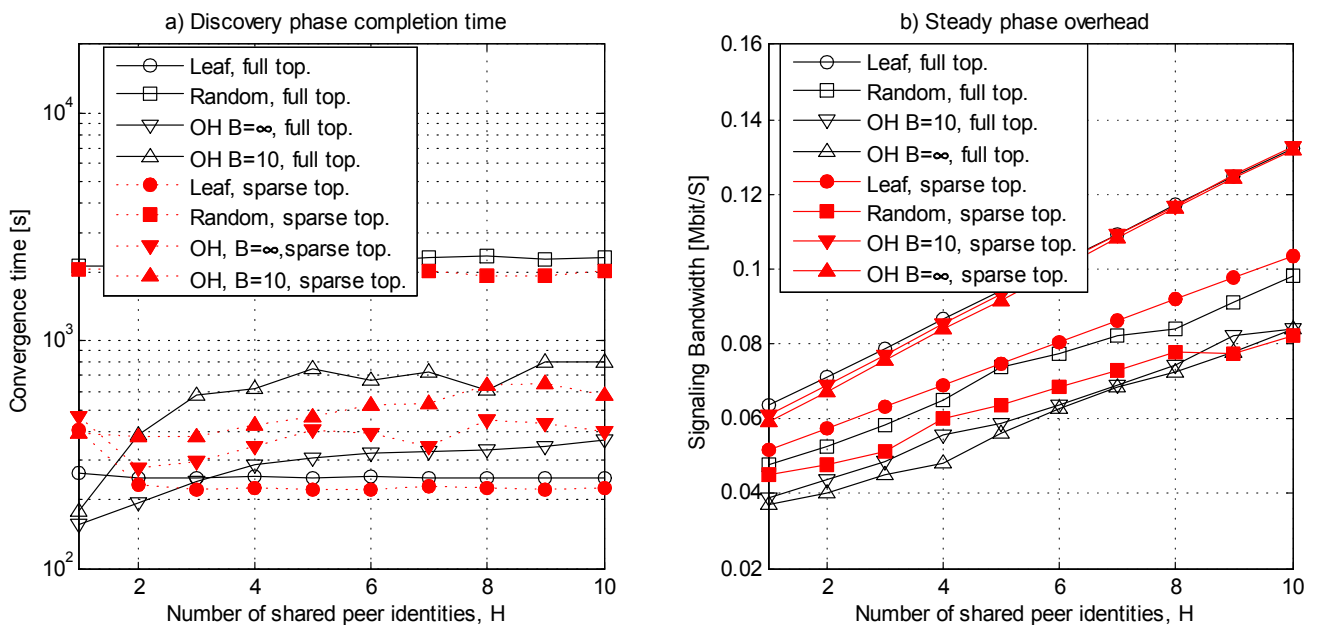


Figure 4.23: Full and sparse topology (a) convergence time and (b) steady phase overhead as a function of H .

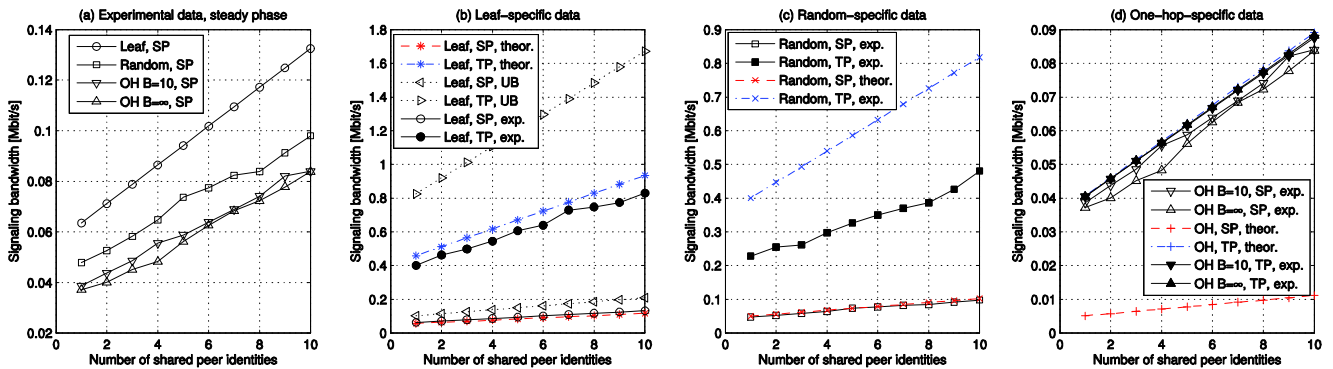


Figure 4.24: Full topology signaling bandwidth: (a) steady phase experimental data for all solutions, (b) Leaf experimental data and models, (c) Random experimental data and models, (d) One-hop experimental data and models.

In order to evaluate the convergence time of the Leaf solution in larger networks, we made use of a custom Matlab simulator. We have generated 10 graphs, by using the model proposed in [27], with a number of nodes that ranges from 100 to 1000. The convergence time is expressed in number of gossip cycles. Figure 4.25 shows the simulation results, a lower bound of the convergence time, obtained through the maximum number of leaves in a single tree T_i with $i \in V$, and an upper bound, obtained by K . The distance between the lower bound and the Leaf solution is due to the initial cycles of the algorithm, when PTS is selected from a small set of peers, and each node may share peers that are not leaves in its tree. Since these shared peers may be chosen by other peers in subsequent cycles, useless gossip rounds are executed thus slowing the algorithm down. In order to limit discovery time and overhead in very large networks, it is necessary to partition the network and limit the scope of each zone by setting suitable values of the IP TTL field for Leaf-based solutions. In summary, One-hop solutions are acceptable in some cases, but their performance is not easily predictable. For sparse topology, which is the most realistic one, the Leaf solution is preferable.

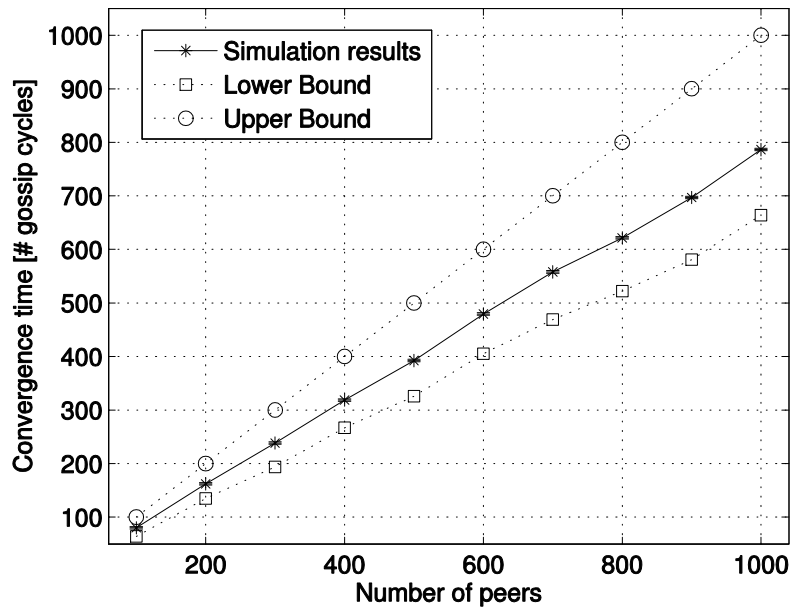


Figure 4.25: Convergence time expressed in gossip cycles for the Leaf solution. 95% confidence intervals are shown.

Finally, it is easy to show that the time needed to react to a topology change (node insertion or removal) is bounded by the discovery time.

4.1.4.2 Off-path signaling distribution results

Off-path signaling dissemination algorithms have been analyzed by using the SETUP message of the NetServ NSLP. During the experiments we measured all the NSIS traffic at the IP layer, including both TCP and UDP protocols. We used two different configurations. In the first, the SETUP message was sent by a stub node, while in the second it was sent by a core node. The message was distributed over a Bubble with radius r ranging between 1 and 5 IP hops. Experiments have been repeated by using full and sparse topologies, with SETUP message sizes of 355 and 1024 bytes. Both values are typical of the considered NSLP. Figure 4.26 shows the total signaling traffic generated by the off-path signaling experiments, using the full topology, as a function of the radius r .

Each bar shows the amount of traffic generated by the transmission of a single SETUP message over the Bubble. The lower part of each bar is relevant to the TCP traffic, whilst the the upper portion refers to the UDP contribution.

The GIST Broadcast algorithm generates significantly larger traffic than the GIST Flooding and the GIST Multicast ones, since queries (UDP), responses (UDP), confirm (TCP), and data messages (TCP), sent to LPs sharing common sub-paths, are received and re-transmitted more than once by the GIST nodes on common sub-paths. We refer to these GIST nodes as network forwarders, NFs. On the contrary, for both GIST Multicast

and Flooding, each node processes NSLP data only once. Even though the presence of multiple destinations in the GIST header increases the GIST Multicast packets size, GIST Flooding increases traffic. In fact, in GIST Multicast the network initiator (NI) selects the nodes to be included in the off-path domain at the beginning of the signaling session.

In GIST Flooding, the selection is done hop by hop by NFs, on the basis of their own PeT and PaT, possibly resulting in sub-optimal paths. In addition, the GIST Flooding algorithm generates a larger amount of UDP traffic due to error messages sent upon reception of duplicate queries. These differences are significant only for large r values, and, more importantly, when the NI is a stub and the message is small. In fact, when the message is small, the impact of duplicated queries and the relevant error messages is higher. In addition, when the NI is a stub, the bubble includes fewer nodes, thus the impact of these inefficiencies is higher.

As shown in Figure 4.27, these differences increase in the sparse topology, since each node has a larger number of neighbors, with a larger IP distance between them. Hence, the GIST Flooding generates a lot of queries and error messages for each node involved in signaling. On the contrary, the GIST Multicast optimizes the distribution over the sub-trees of the overlay network, thus reducing the number of GIST handshakes. As the radius r increases, it compensates the overhead generated by multiple destinations in the Multicast GIST packet header.

In summary, GIST Multicast and GIST Flooding are nearly equivalent for r up to three IP hops in all different configurations, and the advantages of the GIST Multicast algorithm become significant for a radius of five IP hops. The usage of randomized approaches, such as the Blind Broadcasting proposed in [28], would not reduce the additional traffic generated by the GIST Flooding algorithm, and they would largely increase the delivery time, especially in sparse topologies.

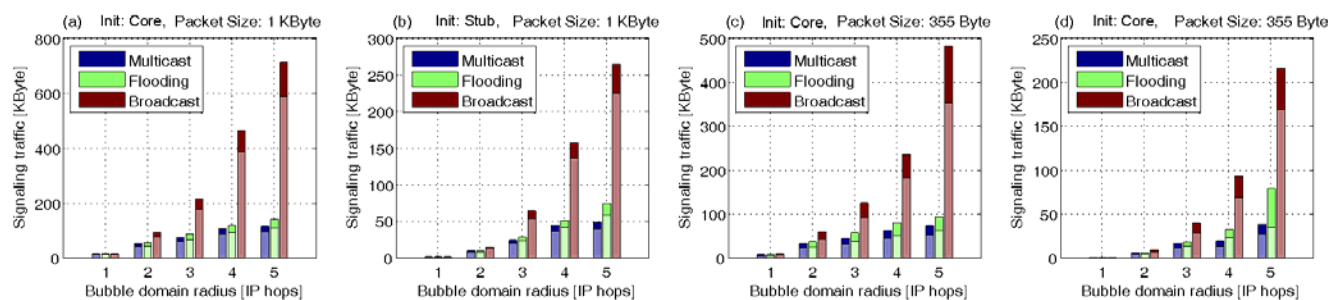


Figure 4.26: Off-path signaling overhead in full topology. Light colour indicates TCP traffic, solid colour indicates UDP traffic.

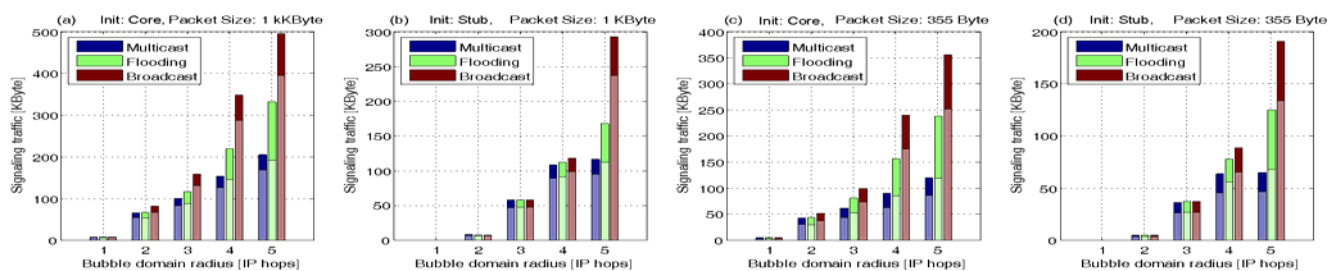


Figure 4.27: Off-path signaling overhead in sparse topology (P=0.8). Light color indicates TCP traffic, solid color indicates UDP traffic.

Now, let us present results relevant to signaling distributed according to the Hose topological domain, which is the domain of highest interest in the ARES framework, as explained in the previous sections. We used the NetServ NSLP Probe/ProbeResponse messages to evaluate the overhead of an ARES query. In fact, NetServ Probe message queries NetServ nodes for application dependent information, and the corresponding ProbeResponse carries back this information to the requesting client.

We sent query messages from a group of sources to destinations with the same distance in terms of IP hops, and then we have averaged results, providing also 95% confidence intervals. The messages was routed through different hoses, the radius of which ranges from 1 to 3 IP hops. The network configuration used is the full overlay topology (i.e. P=1). We selected paths with lengths ranging from 4 to 9 IP hops. We measured the aggregated network overhead, calculated by counting the size of each message (Probe or ProbeResponse or Error Message) which crossed any topology link at the IP layer, by using the logging facilities provided by iptables.

In this work, we do not consider the overhead associated to GIST gossip messages, which has a negligible impact on network capacity, as already shown in the previous Section 4.1.4.1.

Figure 4.28 shows the aggregated overhead as a function of the path length, with the radius of the bubbles composing the hose as a figure parameter. As expected, the overhead increases with both the path length and the bubble radius. The general comment is that the aggregated overhead is definitely negligible (below 200 KB in the worst case, i.e. a path length of 9 IP hops and hose radius of 3 IP hops). Consider that an hose with radius equal to 3 IP hops, with an IP path length equal to 9 IP hops, can include more than half of the network nodes.

In addition, such an overhead is not only limited as absolute value, but also when compared with size of the file to be moved. In fact, the average size of a single genome is about 3.2 GB, whereas compressed VM images are in the order of 3 GB. This means that the total traffic due to a single signaling step is lower than 0.1% than the size of the smaller file transferred. In addition, since each computation last for a few hours, the impact of the associated signaling bit rate on network performance is negligible as well.

It is worth underlying that by using the information on network and nodes status, provided by the proposed discovery service, the average amount of traffic to be moved within the network can be reduced by a factor 6,

as shown in the simulation results presented in section 4.2.6. Thus, not only the overhead associated to service/resource discovery is negligible with respect to the amount of traffic associated to the processing service, but also the relevant benefit is definitely worth its deployment.

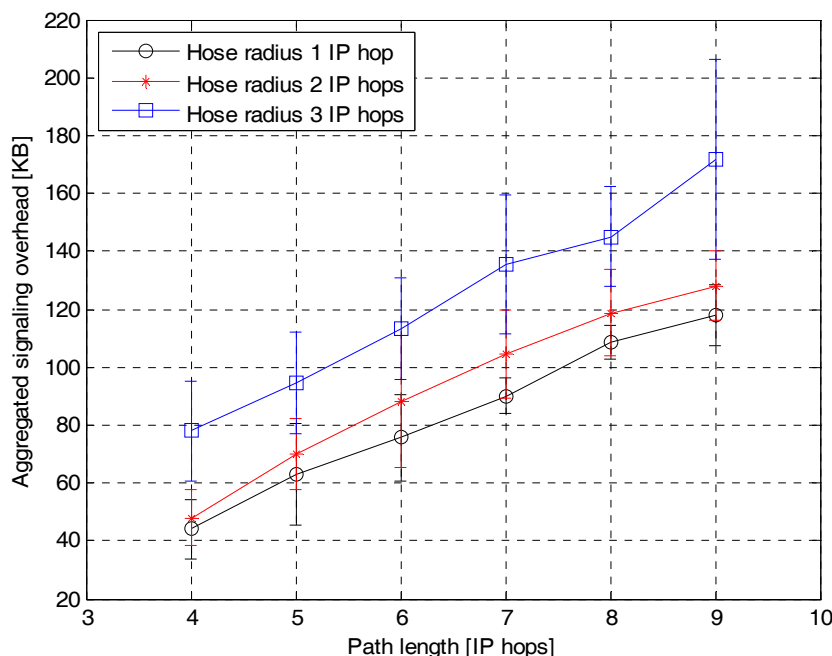


Figure 4.28: Signaling overhead over the whole network as a function of the path length, for different value of the hose radius. Network configuration with parameter $P=1$.

4.2 Network and Service Management

This section illustrates the ARES management framework for the control of the virtual machines hosting the NetServ instances and the software packaged for genomic processing. This includes a set of algorithms and procedures that will be used to perform all the operations illustrated in Section 4.1. These operations are carried out by executing novel algorithms, which are genuine research products of ARES. In particular we will detail:

- *algorithms to dimension the amount of resource allocated to VMs with respect to the workload;*
- *algorithms for the optimal deployment and displacement of VMs over the CDN;*
- *algorithms and procedures for migrating VMs, or migration of application bundles between NetServ instances;*
- *algorithms to optimally dimension overlay links.*

In Section 4.2.1 we present the resource management operation managed by the LOIB through the OpenStack [5] APIs. In Section 4.2.4 we will focus on the Network management executed by the LOIB using Neutron APIs.

4.2.1 Resource management through OpenStack APIs

In this section we will detail the operation executed by the Ares entities during a processing task flow.

4.2.2 GCM operation

The most relevant part of the operations executed by the GCM are:

- *Retrieving of the information on the OpenStack cloud position and on content availabilities inside the CB scattered around the network.*
- *Executing the optimization function to select the PoP to host the computation.*

The signaling involved in these operations is described in Section 4.1.2 and the algorithms for the optimization procedure will be detailed in Section 4.2.5.

After the GCM has selected the PoP to host the process, it enumerates the input files provided by the medical personnel in the message described in Section 4.1.3.1. We suppose that the input folders can be accessed through the FTP protocol. Hence the GCM can use the FTP list command to enumerate all the files present in each folder specified by the user, and use them to fill the download list. It will also map the priority requirements into a combination of minimum resources amount (memory, number of cores, storage). Then it sends the submitTask command to the selected LOIB (see Section 4.1.3.2).

4.2.3 LOIB operation

The LOIB receives from the GCM the complete list of parameters necessary to carry out the processing (see Section 4.1.3.2). To allow the LOIB to use all the OpenStack API, we supposed that inside the OpenStack cloud the cloud administrator created a dedicated tenant, called Ares tenant. Inside this tenant an admin user is needed. Username and password of this user can be set inside the LOIB configuration file. With these credentials the LOIB can connect to the Keystone API and request authentication tokens, granting the access to all OpenStack API.

First step is injection of the virtual machine image inside the Glance repository. The OpenStack API does not allow downloading the Image file directly inside the Image repository. For this limit the LOIB downloads the image from the CB and injects it inside the Glance Image Repository through its RESTful interface. After the upload, the LOIB uses the Glance API also to retrieve the status of the image uploaded and checks for error.

Then the LOIB uses the resources requirement specified in the task submission to find a suitable OpenStack Flavor. An OpenStack Flavor is a combination of memory amount, number of cores and storage amount that represent the resources allocated for a specific instance of an image. The best flavor is selected from the LOIB in order to respect the condition and to save resources.

After the Flavor Selection a new instance of the GPVM can be created. The LOIB uses the Nova API to create the new instance of the Image uploaded to the Glance repository, using the selected flavor. Once the instance is created, OpenStack will carry out the necessary operation to spawn the instance in the cloud. The duration of the spawning operation depends from the image size and from the cloud status, hence the LOIB will use the Nova API to poll the cloud until the instance is spawned and running. Operation to gain connectivity for the new GPVM will be detailed in Section 4.2.4. After the virtual machine is running and connected the LOIB can use the RESTful interface inside the GPVM to trigger the execution of the pipeline, as detailed in Section 4.1.3.3.

4.2.4 Network management through OpenStack APIs

GPVM connectivity is provided through the Neutron module of the OpenStack framework. In order to communicate with the external entities of Ares, the GPVM need to have a so called Floating-IP assigned to it. Floating-IP allows an OpenStack Instance to be addressed from clients outside the OpenStack virtual network. To allow the creation and allocation of Floating-IP, a so called *floating network* needs to be provisioned inside the Neutron framework. Hence the cloud administrator needs to reserve a slice of addresses belonging to the external network of the data centre to the Ares tenant, and create the appropriate floating network. An internal network has to be created for the Ares tenant as well. The Ares private network and the floating external network needs to be connected through a Neutron router, setting the external network as a gateway. All Ares instances will be connected to the Ares network automatically by the LOIB, using the Nova API. Once the image is running, the LOIB can use the Neutron API to search for a free Floating-IP inside the floating network, it can create one if none is present, and allocate the floating-IP to the new Instance. After this procedure the GPVM will be addressable from all the entities outside of the data centre, and communication between the GPVM and the LOIB will be allowed.

4.2.5 Network configuration management

Assume the existence of different n classes of services, define according to the tightness of a request. We assume that for each class, the number of service requests arrived within a schedule interval time is considered, as shown in Figure 4.29.

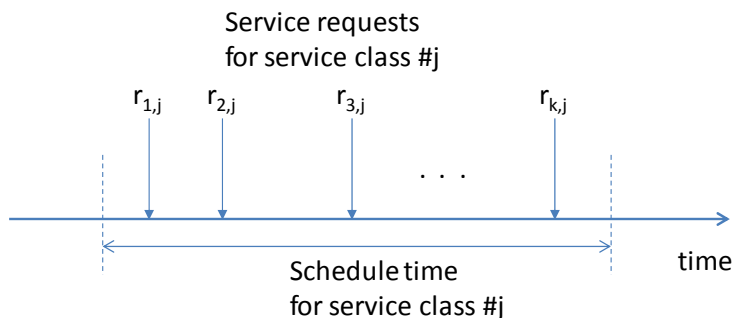


Figure 4.29: Sketch of a time realization of service request process.

The tightness of the services requested in different classes translates into the length of the schedule time, as shown in Figure 4.30: (repeated Figure 3.3 for ease of reading)

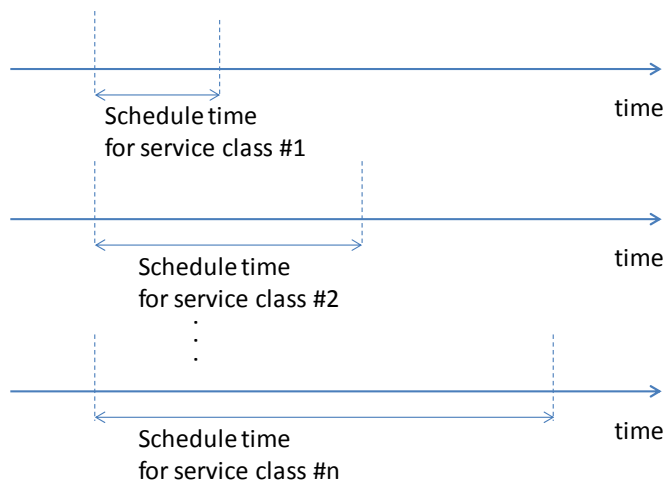


Figure 4.30: Sketch of the ARES differentiated schedule period.

We focus on a the service requests having the shortest schedule time. Service requests r_i are issued from border nodes of the network, as shown in Figure 4.31.

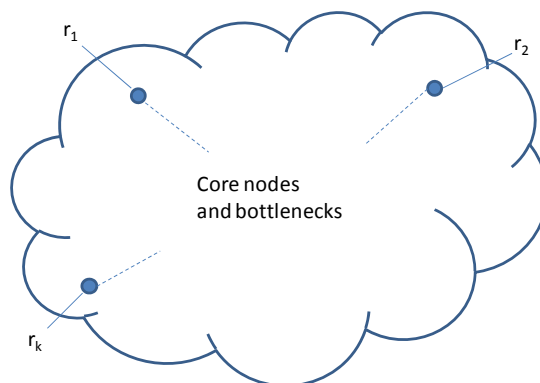


Figure 4.31: Sketch of the ARES distributed requests

Given the seriousness of the requests, the optimization is done for each individual request with the only objective of minimizing the service delivery time.

We assume that the files to be downloaded are large and split in chunks, available in a number K of *end-nodes*. All chunks are available in each end node. End nodes are both repositories and temporary caches that happen to store the desired information. The optimization problem consists of *determining the portion of the file to be downloaded from each end node in parallel*.

The cost function for downloading a portion of size x_i of a file of size S from the i th end node is the download time, which is equal to $g_i(x) = x_i/b_i = a_i x_i$, being b_i the actual bandwidth for downloading the file portion.

We begin by considering a case study with $K=2$. The optimization problem consists of finding the optimum x value that minimizes the download time, as follows:

$$f_{g_1, g_2}(S) = \min_{0 \leq x \leq S} \max(g_1(x), g_2(S-x)) \quad (15)$$

Similarly, for a generic number of end nodes, is follows that:

$$f_k(S) = \min_{x_1, \dots, x_k | \sum_{i=1}^k x_i = S} \max(g_i(x_i)) \quad (16)$$

We will show that the optimum vector $\underline{x} = (x_1, \dots, x_k)$ is the one producing the similar cost for all end nodes. The intuition behind this conclusion is sketched in Figure 4.32 for $K=2$, which shows how an optima value $x_{opt} = a_1/(a_1 + a_2)$ is found.

In synthesis, given K end-points, the optimum is found by solving the following system of equations:

$$\begin{cases} a_1 x_1 = a_2 (S - x_1 - x_3 - \dots - x_K) = a_2 x_2 \\ a_2 x_2 = a_3 (S - x_1 - x_2 - x_4 \dots - x_K) = a_3 x_3 \\ \vdots \\ a_{K-1} x_{K-1} = a_K (S - x_1 - x_2 - x_3 \dots - x_{K-1}) = a_K x_K \\ x_1 + x_2 + x_3 + \dots + x_K = S \end{cases} \quad (17)$$

The general solution of this system is easily found as follows:

$$\begin{cases} a_1 x_1 = a_2 x_2; x_1 = \frac{a_2}{a_1} x_2 = \frac{a_2}{a_1} \frac{a_3}{a_2} x_3 = \frac{a_2}{a_1} \frac{a_3}{a_2} \frac{a_4}{a_3} x_4 \dots = \frac{a_K}{a_1} x_K \\ a_2 x_2 = a_3 x_3; x_2 = \frac{a_3}{a_2} x_3 = \dots = \frac{a_K}{a_2} x_K \\ \vdots \\ a_{K-1} x_{K-1} = a_K x_K; x_{K-1} = \frac{a_K}{a_{K-1}} x_K \\ \frac{a_K}{a_1} x_K + \frac{a_K}{a_2} x_K + \dots + \frac{a_K}{a_{K-1}} x_K + x_K = S \end{cases} \quad (18)$$

Thus:

$$x_K = \frac{S}{a_K \sum_{i=1}^K \frac{1}{a_i}} \quad (19)$$

Again

$$x_{K-1} = \frac{a_K}{a_{K-1}} \frac{S}{a_K \sum_{i=1}^K \frac{1}{a_i}} = \frac{S}{a_{K-1} \sum_{i=1}^K \frac{1}{a_i}} \quad (20)$$

and so on. Thus, the general solution is:

$$x_{opt-j} = \frac{S}{a_j \sum_{i=1}^K \frac{1}{a_i}}, \quad 1 \leq j \leq K \quad (21)$$

The overall cost of the transfer of the set of data ξ from node Ω is therefore equal to

$$C_{\xi, \Omega} = a_{\xi, \Omega, i} x_{\xi, \Omega, opt-i} \quad \forall i, 1 \leq i \leq K,$$

where $x_{\xi, \Omega, opt-i}$ is the size of the portion of ξ downloaded by the i th end-node when the requesting node in Ω and the data set to be downloaded is ξ .

The selected node Ω from which the set of data ξ is downloaded is the node which minimizes the cost

$$C_{\xi} = \min_{1 \leq \Omega \leq K} C_{\xi, \Omega} \tag{22}$$

This problem may be solved for both transferring genome annotations and virtual machines. In case a potential site is collocated with the content, the cost is 0 since the relevant cost function is always null.

Figure 4.32 shows a graphical interpretation of the optimization procedure when two cost functions are involved. The extension to a generic number of functions is straightforward.

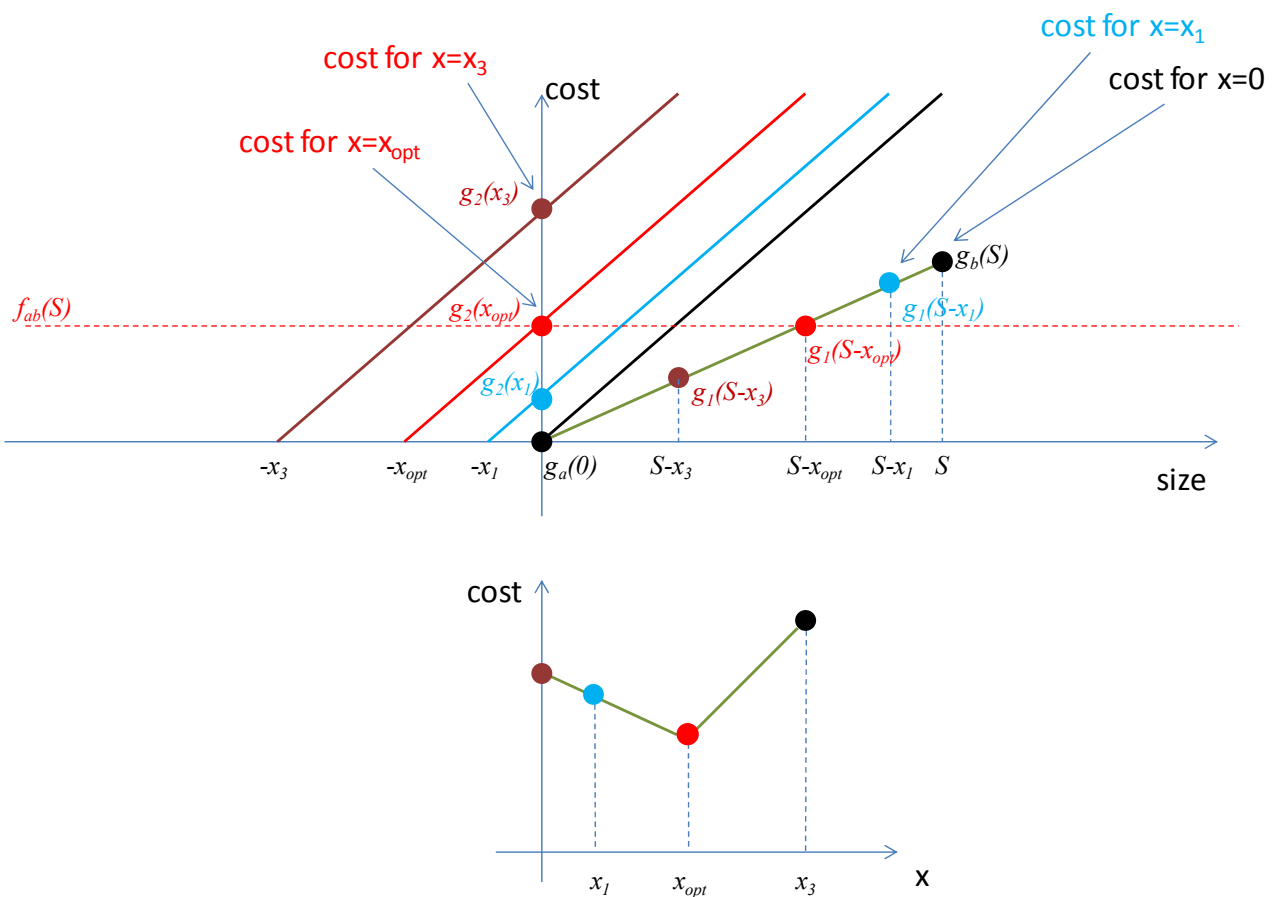


Figure 4.32: Graphical interpretation of the optimization procedure involving two cost functions.

For what concerns the project ARES, we have to select the node that allows minimizing the overall cost for downloading all contents needed for processing genome data. In particular the genome annotations (A), the virtual machines (M), and the patient's genome (G). Thus, the optimization problem consists of selecting the node (or POP) of the network that minimizes the following cost:

$$c_{\xi} = \min_{1 \leq \Omega \leq K} \max(c_{A,\Omega}, c_{M,\Omega}, c_{G,\Omega}) \quad (23)$$

We can assume that the propagation time can be neglected in comparison with the transmission time. In other words, what really counts is the set of bottlenecks for each network path.

The latter assumption implies that all nodes upstream a bottleneck of a path towards a data set ξ generate the same cost value for downloading ξ . Only nodes downstream the bottleneck can provide better results.

1st case study: downloading each content type from a single server.

To solve the optimization problem we can assume that the end nodes where data reside are usable nodes for the optimization algorithm.

It suffices to determine the cost for all nodes and selecting the node corresponding to the minimum one.

For what concerns the search of the optimum node, we observe that the metric c_{ξ} can produce local minima. For example, observe the situation shown in the figure below.

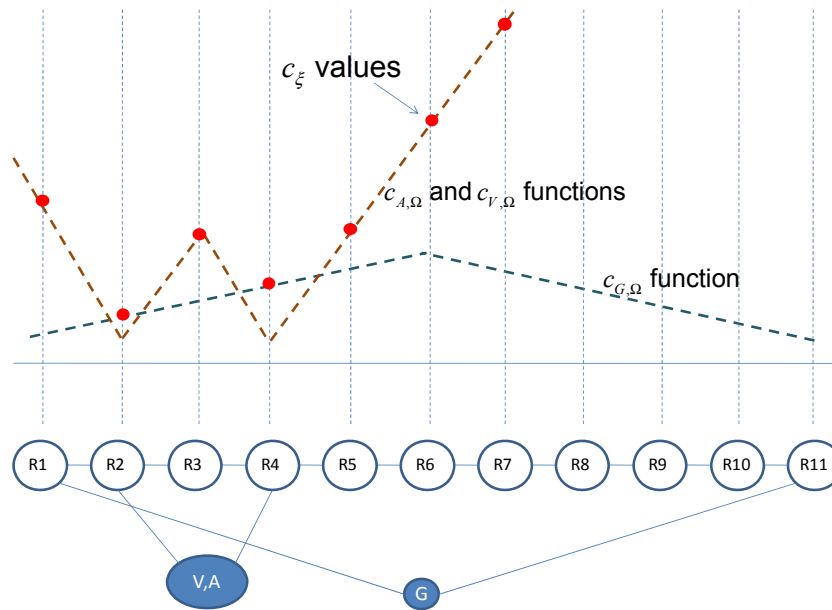


Figure 4.33: Example of a network making evident the presence of local minima in cost function.

It is easily observed that a heuristic search of the minimum implemented by exploring the network from the node R11 in the direction of the maximum descent metric value, the search gets stuck into the node R4, which is a local minimum, whilst the optimum node selection is R2. This results is clearly determined by the specific metrics used, which are increasing with the distance from the contents, which is anyway realistic since the available bandwidth is decreasing with the length of the network connection.

In order to avoid this possibility we propose a different approach. Our proposal consists of using the maximum bandwidth tree over the network spanned by each node. A branch of the tree stops growing when all the needed contents are retrievable through it. This way it is possible to consider also the caches already populated by the contents. Clearly, a branch connecting the node to the medical site is necessary since the patient’s genome is not cached and used just once.

The detailed approach will be illustrated in what follows, when the protocol for establishing parallel downloading will be illustrated and the minimum costs determines the optimum node.

CDN Network Service Categories:

Among the multiple possibilities for differentiating network services, we draft three categories that can be used as a baseline for any further differentiation:

- *Top CDN category:* it requires the minimum service delivery time. It is implemented through all defined mechanisms. Used for the most serious situations.

- *Average CDN category*: it requires a low service delivery time. It is implemented through all defined mechanisms with the exception of parallel downloading. From the networking viewpoint, this service aims at reducing the number of network entities involved.
- *Best effort category*: Makes use of the mechanism if the average category, with the only difference consisting of the use of the min-hop metric to select the node where processing is executed. This category definition aims to minimize the consumed resources.

2nd case study: parallel downloading

We now show an algorithm usable for finding different paths. A path is represented as a path vector P , which is the ordered list of the addresses of the network interfaces traversed by proceeding from the path source to the path destination.

Our routing algorithm makes use of the NSIS signaling and its modified version making use of the Bubble/Balloon signaling.

We will use the following further notation in text and figures:

- $V(x)$: set of nodes neighboring the node x .
- $P(x,y,i)$: i th path vector from a node x to a node y .
- $R(s,d)$: set of path vectors from the source node s to the destination node d .
- $I(P)$: first component of the path vector P .
- N : set of nodes in any path.
- $PPath$: set of partial paths.

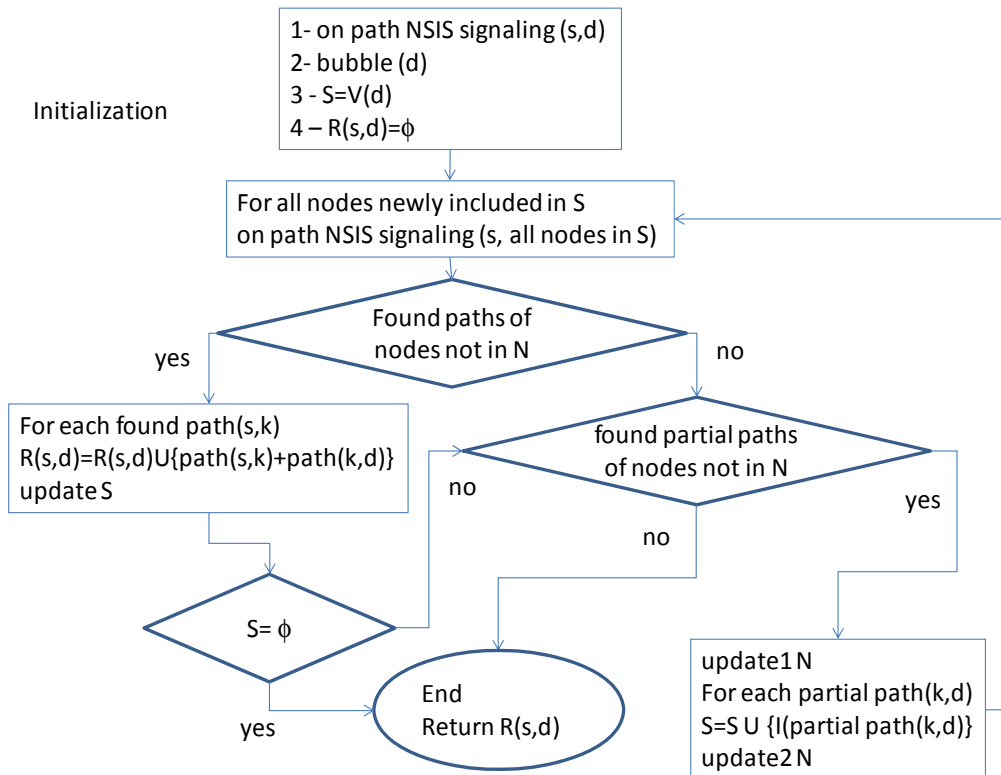


Figure 4.34: Flow diagram of the algorithm used for identifying disjoint paths

Update S means removing from S all elements that was included in a path from s to d.

Update1 N means to remove from N all nodes that were in partial paths that are not joint to a recently found partial path.

Update2 N means to include in N all nodes that are present in a partial path recently found.

Example 1:

Consider the network topology shown in Figure 4.35. The steps of the algorithm shown above, applied to this network, are summarized in Figure 4.36 below.

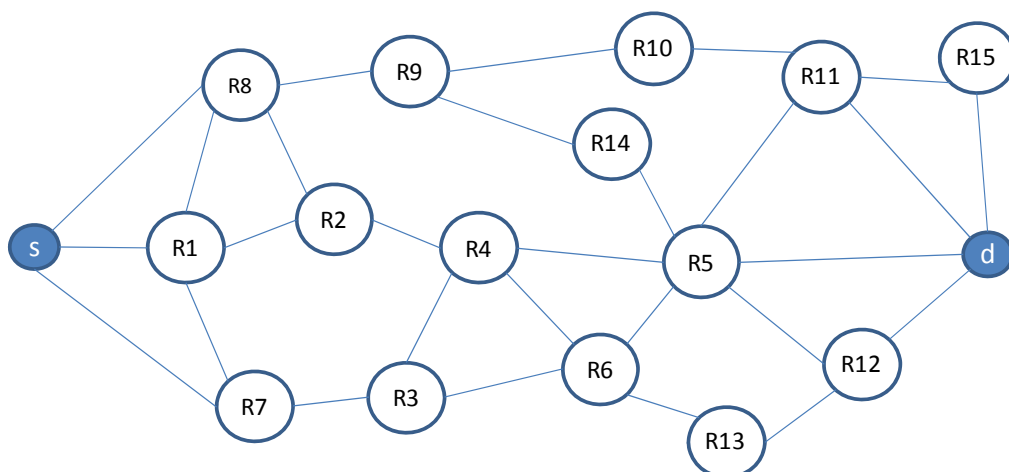


Figure 4.35: Sample network architecture for demonstrating the proposed routing algorithm

It is worth to note that it could be sufficient to make use of bottleneck disjoint paths, and not totally disjoint paths. Nevertheless, identifying a bottleneck is a quite challenging task in operation, and a bottleneck position may also change over time. For this reason, we preferred to follow the approach of making use of totally disjoint paths, so as to also guarantee the possibility of implementing independent control operations over paths.

Having illustrated how parallel downloading can be implemented, we go back to the problem of finding the network configuration for providing the desired processing service.

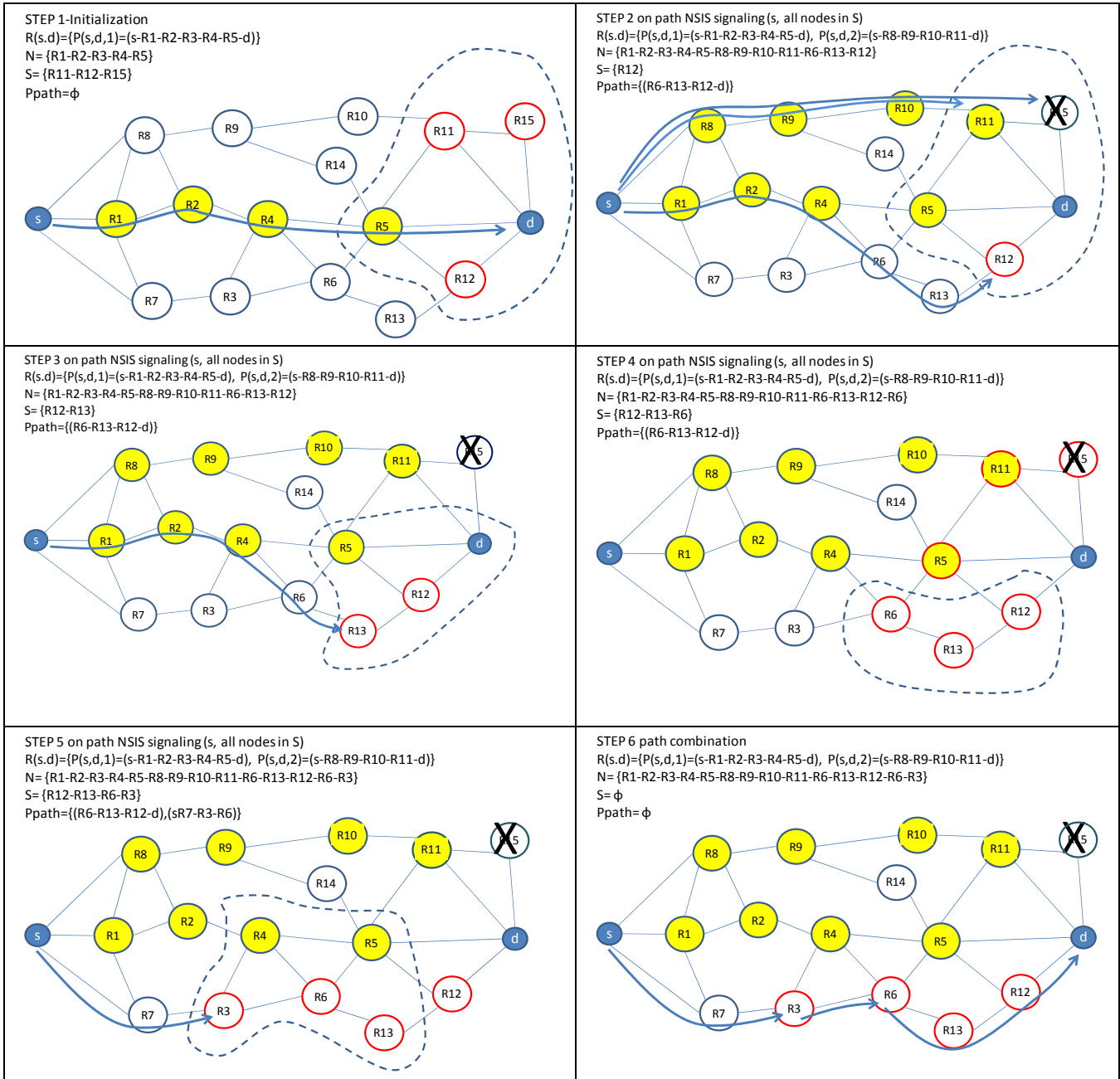
Remember that the final aim is to allocate network bandwidth to the contents to be downloaded.

Let SA, SG, and SV be the size of the contents to be downloaded, annotations, patient's genome and virtual machine, respectively. By applying the same concepts deriving from the inf-max convolution, we have to allocate bandwidth to download these contents so as to make the download time identical for all contents.

From the procedure illustrated above for parallel downloading, for each individual node we can find the set of path vectors, associated with each content, leading to that node, with a given bottleneck bandwidth. Some elements of the path vectors relevant to different contents could be shared, and in that case the relevant bandwidth must be allocated to the contents with the aim of achieving the same download time.

We will use the following further notation:

- $b(x,y,i)$: bandwidth of the i th path vector from a content site x to a node y .
- NS: number of path vectors having components in common with other path vectors.
- $J_{a,b,T}$: index valued 1 if the bandwidth a is taken from the path vector b to send the chunk T , otherwise 0.
- NS_x , with x in $\{A,G,V\}$, is the number of servers storing the content x , thus corresponding to the number of chunks downloaded for the content x .



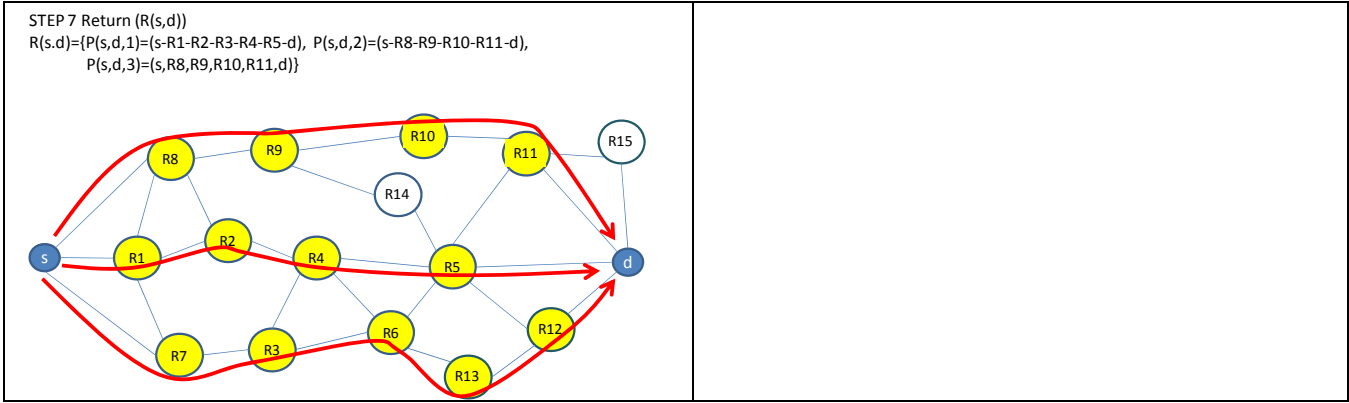


Figure 4.36: Steps of the routing algorithm enabling parallel download.

The solution of the following system of equations must be found.

$$\begin{cases}
 \frac{1}{b_{1,A,T}} x_{i,A,T} = \frac{1}{b_{2,A,T}} x_{i+1,A,T}, \forall T = 1, \dots, NS_A \\
 \sum_{T=1}^{NS_A} x_{T,A} = S_A \\
 \frac{1}{b_{1,G,T}} x_{i,G,T} = \frac{1}{b_{2,G,T}} x_{i+1,G,T}, \forall T = 1, \dots, NS_B \\
 \sum_{T=1}^{NS_G} x_{T,G} = S_G \\
 \frac{1}{b_{1,A,V}} x_{i,A,V} = \frac{1}{b_{2,V,T}} x_{i+1,V,T}, \forall T = 1, \dots, NS_V \\
 \sum_{T=1}^{NS_V} x_{T,V} = S_V \\
 b_{i,A,T1} J_{i,k,T1} + b_{j,G,T2} J_{j,k,T2} + b_{f,V,T3} J_{f,k,T3} \leq b(S_W, node, k), \forall W \in \{A, G, V\}, \forall i, j, k, T1, T2, T3.
 \end{cases} \tag{24}$$

The last equation of the system above is a congruity equation. It verifies that the bandwidth of the shared connections is not exceeded.

We stress that the algorithm shown by the flow diagram shown in Figure 4.34 and subsequent figures can be implemented by making use of the NSIS off-path extensions already designed and being currently implemented.

4.2.6 Numerical results

For what concerns the achievable performance, at this stage of the project (we are finalized the debugging of the overall platform) they have been analyzed by means of simulations. We have selected an optimization function which aims at minimizing the network traffic.

Service requests are modeled as a Poisson process, with average rate λ , generating from 5 to 47 requests per hour. For each value of λ , we have simulated 10000 genomic processing requests. Each request comes from a user connected to one of the 41 PoPs of the topology. The users are randomly selected using a uniform distribution. We considered the execution of three different genomic processing pipelines. Each of them uses a VM, whose image file is 3 GBs large. The auxiliary and reference files, together with the necessary annotations for the three different processing pipelines are equal to 1, 10, and 20 GBs, respectively. The relevant execution times have been set to 3, 4, and 5 hours, including also the transfer time. Data flow can be shaped in the hypervisor or by another NetServ bundle, so as to avoid overwhelming the network and thus causing starvation to other type of traffic. According to the GÉANT map available at [30] and illustrated in Figure 4.21, the minimum bandwidth between PoPs has been set to 1 Gb/s. We assumed 10 Gigabit Ethernet connections within PoPs. Each PoP which hosts a data centre (see Figure 4.20) can serve up to 5 genomic requests contemporarily, regardless the selected pipeline. This is equivalent to configure a tenant dedicated to ARES within an OpenStack deployment. The routers are all NetServ-enabled routers. This NetServ deployment can be accomplished either by resorting to a software defined networking solutions, which has been successfully tested and described in [31], by porting the NetServ platform in a real router, or by using Linux-based software router solutions. We assumed that the caching capabilities (RAM plus disk space) in NetServ nodes is equal to 50 GBs. Only VM images and annotations/auxiliary files are cached for future usage, but not genomes, due to privacy constraints.

The proposed service architecture is compared with a basic cloud computing paradigm (see, e.g., the cloud solution described in [32]), in which each PoP with a connected data centre includes a computing node with the same resources of the ARES solution, in order to perform a fair comparison. When a new service request arrives, the computing node is selected randomly among those available, and it is fed by VM image and input files (genome plus auxiliary files).

Figure 4.37 shows the simulation results, which consist of the aggregated bit rate resulting from file transfer as a function of the service request rate. The gain provided by the ARES solution is evident: it allows decreasing the network traffic rate up to a factor of about 6, while providing the same application services. This result is due to the dynamic caching strategy implemented by the proposed solution, which strongly reduces the transmission of very large files on the backbone network. We have also evaluated the impact of deploying our solution only in VMs in data centres. The effect is that, without the caching in router, the solution is slightly less effective (the difference is about 5%). However, the difference is limited since we have considered unlimited storage resources in data centres, which is not realistic. When the number of different types of genomic processing increases and the storage space in data centres is limited, the additional storage in routers can significantly improve the overall effectiveness of the system, especially if suitable deduplication techniques [33] are deployed.

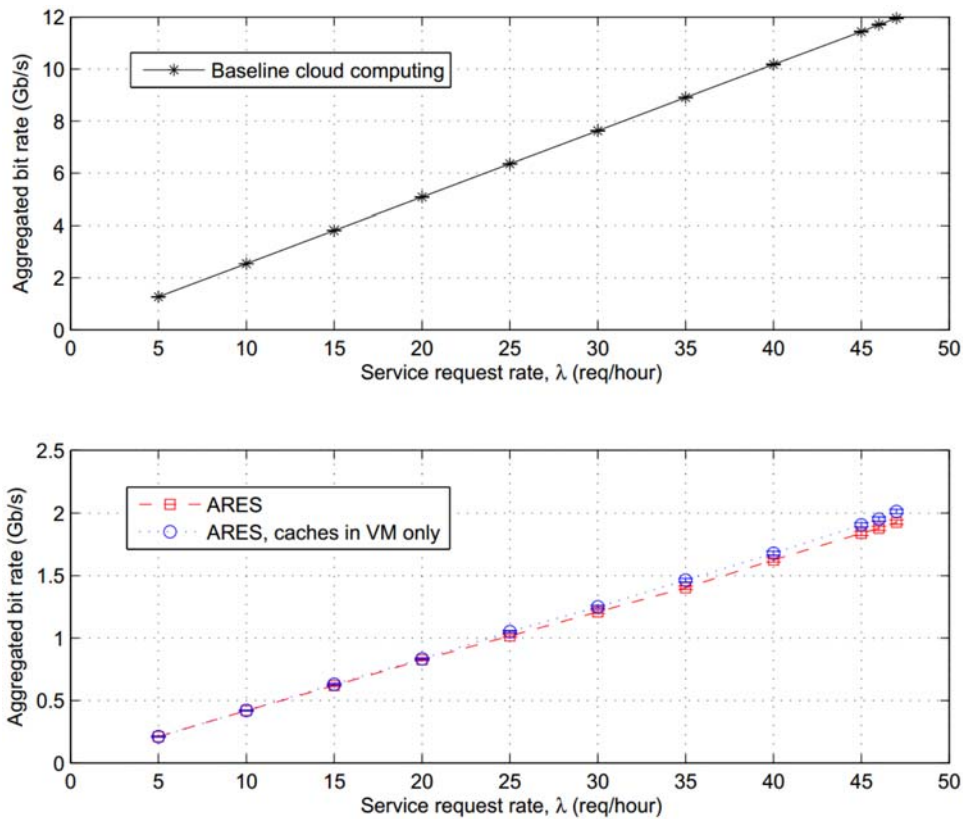


Figure 4.37: Simulations results: ARES vs. baseline cloud computing paradigm, as a function of the input load.

4.3 Chapter Conclusions

This document illustrates the complete signaling procedures in ARES, together with the results of an experimental and simulation campaign used to validate our procedures. For this purpose, we have detailed, step by step, all interactions between the ARES functional and physical entities. In addition we have shown the optimization procedures used to select the available genomic contents and their replicas so as to minimize the download time and usage of network resources. Again, we have sketched a protocol for identifying disjoint paths for implementing parallel download of genomic data sets and virtual machines. Obtained results show that the impact of signaling is really negligible, whereas the benefit introduced are significant, with a potential saving in network bandwidth up to 6 times when the proposed procedure are deployed.

5 Implementation and Integration

The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog.

5.1 Deployment and integration of hw and sw infrastructure

5.1.1 ARES Software Architecture

In this section, we illustrate all software components/tools used for the deployment and assessment of the ARES project. The main platform used are OpenStack [5] and NetSev [4]. In particular, In our implementation, we use the KVM hypervisor. We make heavily reference to the functional architecture depicted in Section 3, which we report below to ease the reader task:

The cache module (also referred to as CB). Its function is to cache contents so as to serve them upon future requests. These contents can be both VM images, or auxiliary files needed to perform genomic computations. This type of entity can be deployed both on routers and on servers. In the ARES proposal, this functional entity is implemented as a NetServ bundle. For this reason, we label it as cache bundle (CB). Caching can be done both in memory and on disk, depending of the size of the content.

The OpenStack interface module (LOIB). This functional entity is in charge of interacting with the OpenStack deployment in a PoP. Since also this module has been implemented as a NetServ bundle, we label it as local OpenStack interface bundle (LOIB). In addition, this module is in charge to interact also with VM performing the genomic processing.

GCM: Genomic CDN manager, having the tasks of receiving service requests from service users, orchestrating the overall network operations, and running an optimization process whose output is the selection of the PoP where the genomic processing will take place. This entity can be either executed in a dedicated physical machine, or as one of the service modules (NetServ bundles) running in a

server of a PoP. It uses a MySQL database to store session information. The DB can be deployed on a physical or virtual machine, and suitable replicated.

Let us start with the VMs running DE or CNV pipelines. In addition to the bioinformatics software, we added a *restful* web interface, implemented in Jetty, a well known Java tool for running web services. It allows the LOIB module to interface to the VM without the human intervention. This web server is able to interface with a Python script, which manages the execution of the pipeline implemented by the VM, as well as the ingestion of auxiliary files (e.g. h19 human genome reference model) and or input files to be processed.

As for the deployment of NetServ, it runs in PCs or VM running Ubuntu Server 12.04 LTS. A NetServ installation consists of a number of modules, interacting each other. The basic modules are

- NetServ controller (C++): it is the component which orchestrates all NetServ modules, including signaling, containers, and so on.
- NetServ container (C++): it is responsible to launch a Java Virtual Machine (JVM), compliant with OSGi framework.
- OpenJDK Runtime Environment (IcedTea 2.4.7): the JVM used.
- OSGI 3.6.0.v20100517: The Open Service Gateway initiative (OSGi) specification describes a modular system and a service platform for the Java programming language that implements a complete and dynamic component model, something that does not exist in standalone JVM environments. Applications or components, coming in the form of bundles for deployment, can be remotely installed, started, stopped, updated, and uninstalled without requiring a reboot; management of Java packages/classes is specified in great detail. Application life cycle management is implemented via APIs that allow for remote downloading of management policies. The service registry allows bundles to detect the addition of new services, or the removal of services, and adapt accordingly.
- NsisKa + Offpath extension (C++): the NSIS version used in ARES [6], enhanced with the off-path extension [4].
- NetServ NSIS NSLP (C++): the interface between the low level NSIS signaling framework and the NetServ system. The controller is in charge to use the NSLP interface to manage bundle set up, tear down, and configuration changes.

The libraries necessary to develop and deploy the NetServ application bundles used in the project are:

GSON 2.2.2: a parser for the JSON format. JSON (JavaScript Object Notation) is a lightweight data-interchange format, and it has been chosen since it is easy for humans to read and write. It is also easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

Jetty 6.1.24: Jetty (Servlet Engine and Http Server) provides a Web server and javax.servlet container, plus support for SPDY, WebSocket, OSGi, JMX, JNDI, JAAS and many other integrations. These components are open source and available for commercial use and distribution. Jetty is used in a wide variety of projects and products, both in development and production. Jetty can be easily embedded in devices, tools, frameworks, application servers, and clusters

In our lab, we deployed OpenStack by installing all modules on up to three server/PC, with enough computing resources. We installed OpenStack on Ubuntu 12.04.3 LTS 64 bit systems. We used the following modules of the OpenStack Havana version:

- Keystone: the IDM module, to manage authentication.
- Glance: the VM Repository module
- Nova: the compute module.
- Neutron: the module in charge of managing the networking functions.

Each OpenStack installation needs a NetServ VM, which orchestrates all ARES specific services. This NetServ VM will run the LOIB bundle, in addition to a CB bundle. The other NetServ VMs (i.e. those emulating NetServ enabled routers) will run just a CB bundle.

As for the interaction between the LOIB and OpenStack, the Keystone module needs to be configured to accept API instruction from the LOIB. To this purpose it is necessary to create a new OpenStack tenant. In the OpenStack environment, a tenant is a container used to group or isolate resources and/or identity objects. Depending on the service operator, a tenant may map to a customer, account, organization or project. In our deployment it is necessary to create an ARES tenant, which groups the NetServ VM and all the GPVM instances, and isolates the private network between them. An administrator user must be created inside the tenant and its login credentials must be configured inside the LOIB, using the LOIB configuration file. Thus the LOIB can use Keystone API to request an authorization token and use it to authenticate with the needed OpenStack modules.

5.1.2 ARES Hardware Architecture

The hardware infrastructure used to deploy the lab testbed is described in what follows. It consists of two sites at the University of Perugia, interconnected by a shared, IPv4 routed network infrastructure with bottleneck at 100 Mb/s.

In one site it is deployed our lab (TLC laboratory), which is equipped with the following devices available to test the ARES components:

- 5 personal computers (PCs) running Ubuntu server 12.04.3 LTS, 64 bit, with the following characteristics: 24 GB of RAM, 2 hard disks with 1 TB capacity each, 1 processor Intel(R) Core(TM) i7-

3770 CPU @ 3.40GHz (4 physical CPUs + 4 hyperthreaded virtual CPUs). 2 gigabit Ethernet network interfaces.

- 1 PCs running Ubuntu server 12.04.3 LTS, 64 bit, with the following characteristics: 16 GB of RAM, 2 hard disks with 1 TB capacity each, 1 CPU Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz (4 physical cores + 4 hyperthreaded virtual cores). 2 gigabit Ethernet network interfaces.
- 16 ports Gigabit Ethernet switch, VLAN enabled, plus an additional 48 ports Gigabit Ethernet switch, VLAN enabled as well;
- 1 server running Ubuntu server 12.04.3 LTS, 64 bit, with the following characteristics: 128 GB of RAM, 4 AMD Opteron (TM) Processors 6128 @ 2.00 GHz (total 32 physical cores), 2 hard disks (SAS) with capacity of 146 GB each for running the system + 1 external (USB 3.0) SATA storage device with capacity 4 TB. 4 gigabit Ethernet network interfaces;
- 2 servers running Ubuntu server 12.04.3 LTS, 64 bit, with the following characteristics: 22 GB of RAM, 1 hard disk (SAS) with capacity of 146 GB + 1 additional SATA storage device with capacity 2 TB, 2 processors Intel(R) Xeon(R) CPU E5410 @ 2.33GHz (total 8 physical cores each server). 3 gigabit Ethernet network interfaces;
- 2 servers running Ubuntu server 12.04.3 LTS, 64 bit, with the following characteristics: 64 GB of RAM, 1 hard disk (SAS) with capacity of 146 GB + 1 external (USB 3.0) SATA storage device with capacity 4 TB, 4 processors Six-Core AMD Opteron(tm) Processor 8425 HE 2.1 GHz 64 GB (total 24 physical cores each server), 3 gigabit Ethernet network interfaces;
- 1 workstation running Ubuntu server 12.04.3 LTS, 64 bit, with the following characteristics: 16 GB of RAM, 1 hard disk (SAS) with capacity of 500 GB + 2 external (USB 3.0) SATA storage device with capacity 1 TB, 2 processors quad-core Intel(R) Xeon(R) CPU E5620 @ 2.40GHz (total 8 physical cores+ 8 hyperthreaded virtual CPUs).
- 1 PC running Ubuntu server 12.04.3 LTS, 64 bit, with the following characteristics: 4 GB of RAM, 1 hard disks with 1 TB capacity, 1 processor Intel(R) Core(TM) 2 Quad Processor Q9650 (4 physical CPUs @3 GHz). 1 gigabit Ethernet network interfaces.

In the other site it is deployed a shared lab, used mainly for teaching purposes (Software Engineering laboratory), which is equipped with the following devices available to test the ARES components:

- 2 PCs running Ubuntu server 12.04.3 LTS, 64 bit, with the following characteristics: 32 GB of RAM, 2 hard disks with 1 TB capacity each, 1 processor Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz (4 physical CPUs + 4 hyperthreaded virtual CPUs). 2 gigabit Ethernet network interfaces.

- 8 PCs running Ubuntu server 12.04.3 LTS, 64 bit, with the following characteristics: 8 GB of RAM, 1 hard disks with 500 GB capacity, 1 processor Intel(R) Core(TM) i5-2300 CPU @ 2.8GHz (4 physical CPU cores). 1 gigabit Ethernet network interface and 1 100 Mb/s Ethernet network interface.
- 4 PCs running Ubuntu server 12.04.3 LTS, 64 bit, with the following characteristics: 12 GB of RAM, 1 hard disks with 500 GB capacity, 1 processor Intel(R) Core(TM) i5-2300 CPU @ 2.8GHz (4 physical CPU cores). 1 gigabit Ethernet network interface and 1 100 Mb/s Ethernet network interface.
- Two 48 ports Gigabit Ethernet switch, both of them VLAN enabled. One of them is also compliant with OpenFlow specifications 1.0.

5.1.3 ARES System Integration

This section illustrates the two testbed architectures used to test our system. The software and hardware configurations have been described in the previous section and will not be repeated. The first testbed architecture is a complete system deployment (**full testbed, FT**), in which we have installed in our infrastructure all software entities described in Chapter 3. This has allowed us not only to demonstrate the feasibility of the system, but also to characterize the genomic workload in terms of consumption and efficiency of use of CPU cores, amount of memory, disk space, and bandwidth. This characterization has been also used to evaluate the requirements of the above resources during the execution of genomic pipelines, providing additional insight on the needed resources. This characterization has been used to deploy another testbed architecture, the **large scale testbed (LST)**, in which we deployed just network entities on a large scale network, emulating a large part of the Géant one, in order to evaluate the network impact of the proposed service on a geographical network.

5.1.3.1 Full testbed (FT)

Figure 5.1 illustrates the FT architecture and the mapping of the physical hardware of our labs. We organized the network connectivity by inspiring to a subsection of the Géant network (label next to routers identifies Géant PoPs). Inside each OpenStack deployment, there is always a VM running NetServ (not illustrated in order to preserve figure readability) with LOIB and CD bundles. In addition, each OpenStack can run a number of VMs with genomic pipelines, depending on the currently available amount of network resources. The detailed algorithms have been detailed in Section 4. This testbed is accessible from the public internet through a web server. The FT has been used to perform a number of genomic pipelines analyses, so that we have accurately characterized genomic workload in terms of bandwidth, CPU, RAM, and storage for all the duration of the processing.

5.1.3.2 Large scale testbed (LST)

The Figure 5.2 illustrates the LST architecture and the mapping of the physical hardware of our labs. In this case, we have enlarged the portion of Géant network we emulated, arriving to 32 PoPs. In the figures, the black circles close to some PoPs indicate that such PoPs are "equipped" with a data centre. These data centre are not full OpenStack deployment, but VMs with installed the LOIB and the CB. Clearly, we modified the LOIB in order to continue doing all its functions even without a real OpenStack deployment. This large scale testbed has been used to test on a larger scale the impact on the network of the ARES service, and which are the advantages of deploying ARES with respect to a plain cloud computing environment, measured from the viewpoint of a network engineer.

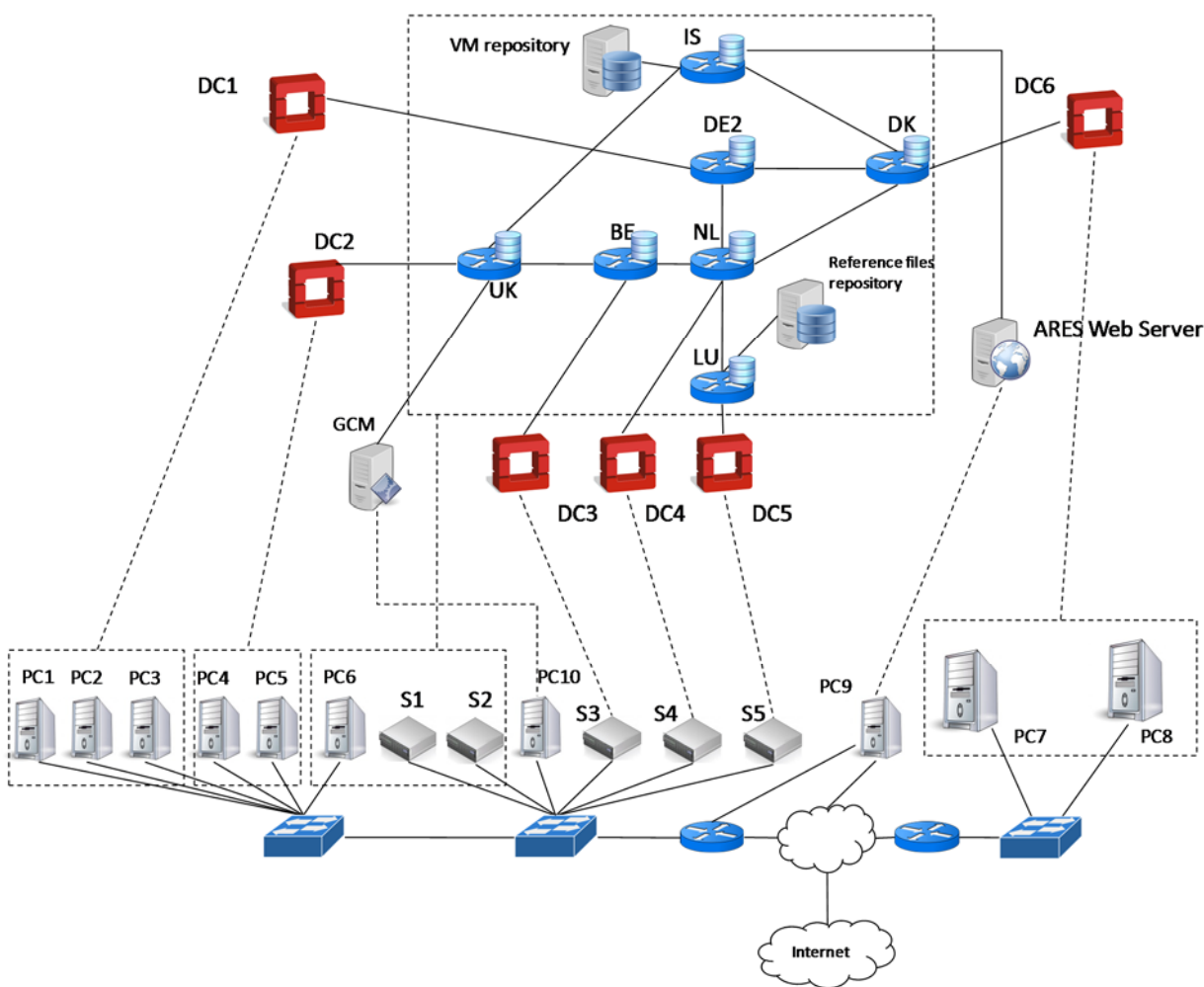


Figure 5.1: the full testbed functional architecture, and the relevant mapping on the physical hardware.

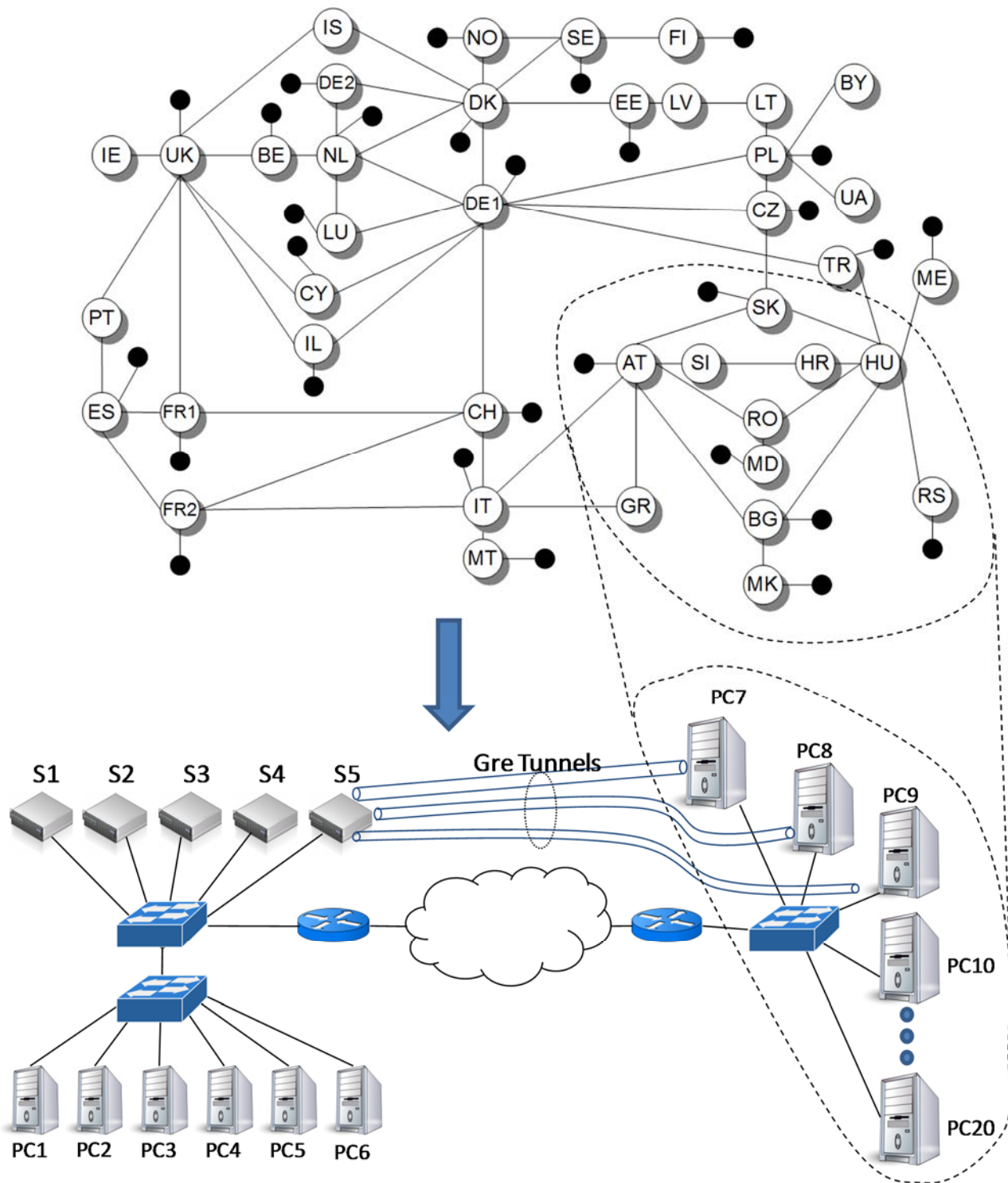


Figure 5.2: the large scale testbed architecture (Géant topology), and the relevant mapping on the physical hardware.

5.2 Deployment of the genomic processing software

The flow diagram of the CNV and DE pipelines was already introduced in Section 3.2, and shown again in Figure 5.3 and Figure 5.4 for ease of reading. The following software tools are used for implementing the CNV pipeline:

- FastQC, which aims to provide a simple way to do some quality control checks on raw sequence data coming from high throughput sequencing pipelines.
- Samtools v 0.1.16: SAM/BAM file format conversion tool.
- Trimmomatic (v 0.32) performs a variety of useful trimming tasks for paired-end and single ended data. The selection of trimming steps and their associated parameters are supplied on the command line.
- Bowtie 2 (v 2.1.0) is an ultrafast and memory-efficient tool for aligning sequencing reads to long reference sequences. It is particularly good at aligning reads of about 50 up to 100s or 1,000s of characters, and particularly good at aligning to relatively long (e.g. mammalian) genomes. In addition, also the latest human genome reference model 19 (hg19) is used.
- CNVnator v 0.2.7: CNV discovery and genotyping from read-depth analysis of personal genome sequencing.

All these software packages are open source. The operating system version is Ubuntu server 12.04.3 LTS. The CNV VM needs at least 4 GB of RAM, and 1 vCPU is enough to carry out the computation (minimum requirements to run a VM implementing the CNV pipeline). The impact of larger size of RAM and/or an higher number of vCPU on computation time has been mentioned in Chapter 3, and deeply analyzed in the framework of the project. Detailed plans for CNV assessment tests are described in Section 6, which includes also the analysis with variable input size.

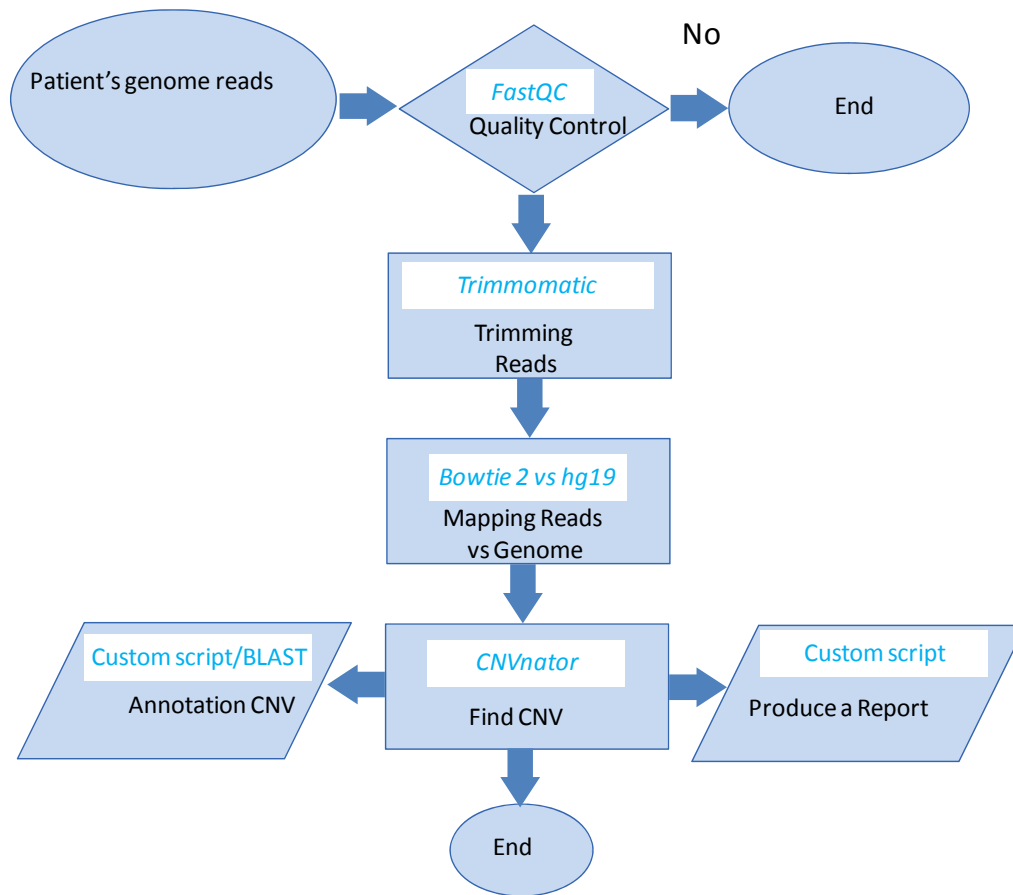


Figure 5.3: Copy Number Variation (CNV) Pipeline

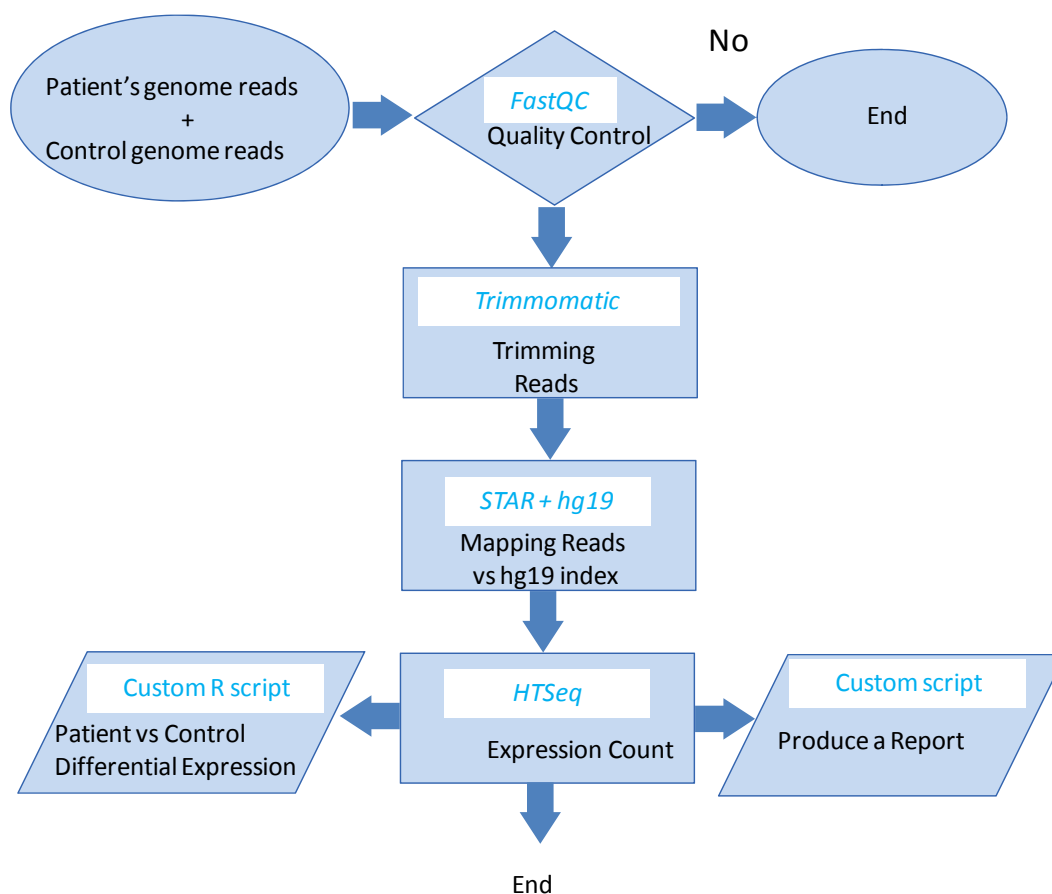


Figure 5.4: Differential Expression (DE) Pipeline

Some of the experiments have been carried out in order to evaluate timing execution of the implemented CNV pipeline when different VM resources, in terms of number of CPU cores and Memory RAM, are allocated when input files of different size are processed.

To this purpose, sequenced data available on the ftp site of the 1000genomes projects (<ftp://ftp.genomes.ebi.ac.uk/vol1/ftp/data/>) have been used. In particular, data files with different sizes related to individuals labelled as HG00097 and HG00358 have been downloaded and processed with the CNV VM. The name of the individual genomic reads and the relevant combination is illustrated in Figure 5.5.

Patient name	Input files size [GB]	In [GB]
HG00097	SRR765989_1 (2.8) SRR765989_2 (2.8)	5.6
HG00097	SRR741384_1 (6.1) SRR741384_2 (6.1)	12.2
HG00097	SRR741384_1 (6.1) SRR741384_2 (6.1) SRR765989_1 (2.8) SRR765989_2 (2.8)	17.8
HG00097	SRR741385_1 (7.1) SRR741385_2 (7.0) SRR741384_1 (6.1) SRR741384_2 (6.1)	26.3

Figure 5.5: Genomic reads used for executing the CNV pipeline.

For what concerns the DE pipeline, the following software tools are used:

- FastQC, which provides a simple way to do some quality control checks on raw sequence data coming from high throughput sequencing pipelines.
- Samtools v 0.1.16: SAM/BAM file format conversion tool.
- Trimmomatic (v 0.32) performs a variety of useful trimming tasks for paired-end and single ended data. The selection of trimming steps and their associated parameters are provided via command line.
- STAR (rna-star v 2.3.0.1) is an ultrafast tool for aligning sequencing reads to long reference sequences. STAR outperforms other aligners by a factor of >50 in mapping speed, while at the same time improving alignment sensitivity and precision.
- Gene Expression Count (HTSeq v 0.5.4p5) is a Python package that provides infrastructure to process data from high-throughput sequencing assays. It provides API and libraries to perform DE analysis. In addition, also the latest human genome reference model 19 (hg19) is used.
- Differential Expression (DESeq R Library v 1.14.0): custom script in R language to analyze results of expression count.

All these software packages are open source. The operating system version is Ubuntu server 12.04.3 LTS. The DE VM would needs at least 4 GB of RAM, and 1 vCPU is enough to carry out the computation (minimum requirements to run a VM implementing the DE pipeline with Bowtie2 aligner). However, in our test we have used the version with the STAR aligner, thus the minimum requirement in terms of amount of RAM is 32 GB. Again, the impact of larger size of RAM and/or an higher number of vCPUs on computation time has been analyzed in the framework of the project. Detailed plans for DE assessment tests are described in Section 6, which includes also the analysis with a variable input size.

Some of the experiments have been carried out in order to evaluate timing execution of the implemented DE pipeline when different VM resources, in terms of number of CPU cores and Memory RAM, are allocated when input files of different size are processed.

To this purpose, sequenced data available on the ftp site of the 1000genomes projects (ftp://ftp.genomes.ebi.ac.uk/vol1/ftp/data/) have been used. In particular, data files with different sizes related to individuals labelled as HG00097, HG00259, and HG003334 have been downloaded and processed with the DE VM. The name of the individual genomic reads and the relevant combination is reported in Figure 5.6.

Patient name	Input files size [GB]	In [GB]	Patient name	Input files size [GB]
HG00358	ERR188089.1 [2.10] ERR188089.2 [2.12]	4.22	None	_____
HG00097	ERR188231.1 [3.15] ERR188231.2 [3.18]	6.33	None	_____
HG00097	ERR188231.1 [3.15] ERR188231.2 [3.18]	6.33	HG00358	ERR188089.1 [2.10] ERR188089.2 [2.12]
HG00097 HG00259	ERR188231.1 [3.15] ERR188231.2 [3.18] ERR188378.1 [3.70] ERR188378.2 [3.73]	13.76	None	_____
HG00097 HG00259 HG00334	ERR188231.1 [3.15] ERR188231.2 [3.18] ERR188378.1 [3.70] ERR188378.2 [3.73] ERR188127.1 [3.70] ERR188127.2 [3.73]	17.87	None	_____

Figure 5.6: Genomic reads used for executing the DE pipeline

5.3 Chapter Conclusions

This chapter illustrates the deployment of the ARES hardware and software infrastructure. The current status of deployment is in accordance with expectations and project plans.

Future deployment activities will include, when available, the use of GTS.

The current deployment of the GTS does not allow integrating virtual machine images loaded from the exterior. Nevertheless, as soon as this function will be made available, the experimental testbed will be transferred into the Géant network, either within or after the formal end of the project. In this regard, an relevant agreement with the Géant personnel has been taken.

6 Experiments and Evaluation

6.1 Plan for experimental assessment

The *purpose* of the plans of assessing is manifold. They must provide evidence that the project has produced the expected outcomes, in terms of deployed services in a way compliant with the proposal expectations. They must also provide useful information for identify any technical problem and the relevant solution.

The *approach* of the assessment plans is based on experiments aiming at identifying strengths and weaknesses of the implemented components and the overall system.

The *information* to be collected consists of both simulated and measured performance of the implemented ARES components. Each subsection in what follows will illustrate the information to be collected for each individual assessment step.

The *personnel* carrying out simulations and experiments is from the University of Perugia for what concerns the networking technical aspect and from the Polo GGB for what concerns the genomic pipelines.

The *methods* for executing experiments followed the metrological approaches, based on the **GUM** (Guide to the expression of uncertainty in measurement) specifications.

The *times* for collecting the desired information have been:

- End of ARES design for executing simulations.
- End of individual component implementation for the assessment of individual components.
- End of the ARES implementation for assessing the implemented services.

The *analysis* and *processing* of the collected information have been done after each individual collection.

The *access* to the assessment activity will be made available to the Géant community and possibly be published.

6.1.1 Plans for Assessing the ARES Individual Components

6.1.1.1 *Plans for assessing the genomic processing tools.*

Genomic processing tools and their deployment have already be illustrated in Section 3.2 and in Section 5.2 For the reader convenience we recall the main features the processing tools, namely Copy Number Variation (CNV) and Differential Expression (DE). They were implemented in Virtual Machines (VMs). The VM size and the auxiliary files size (e.g. the genome indexes) are reported in Table 3.4. They represent the total size of the files to be uploaded into the PoP selected for the computation, in order to execute the relevant genomic pipeline. The RAM size and the number of CPU cores reported in the table are the minimum amount of RAM and the minimum number of CPU cores, respectively, necessary to execute the relevant genomic pipeline. As for the disk size allocated to the VM during the execution of the pipeline, this has been estimated with a specific input file. These values have been obtained by performing extensive computations with different amount of RAM and number of CPU cores allocated to the VM, which represents reference values to be used for associating different service levels with RAM size and allocated number of CPU cores.

Before illustrating some sample results of an extensive experimental analysis of the execution time in different case studies, it is necessary to have an idea about how the implemented pipelines make use of the computing resources in the different steps of their execution. Figure 6.1 shows an example relevant to the CNV pipeline. It shows how the CNV makes of the available computing and memory resources, for a relatively small input size (patient file) of 2.4 GB. Vertical red bars delimitate different phases. It is evident that the pipeline is quite inefficient in the usage of computing resources. Just in the initial phases the multi-core architecture is exploited, and most of software packages included in the pipeline just makes use of a single core. Similar considerations can be done also for what concerns the amount and RAM used. It is arguable that the reason of this behaviour is that, initially, the bioinformatics programmers had not the need of using multi-core machines. Nevertheless, given the increasing diffusion of multicore architectures, including both for CPUs and GPUs (graphical processing units), and that we have used one of the most popular genomic processing software (CNVnator), this issue is significant and need a specific solution. In addition, we have observed a very similar behaviour even in the DE processing. Thus, the inefficient software design is actually an open issue in genomics.

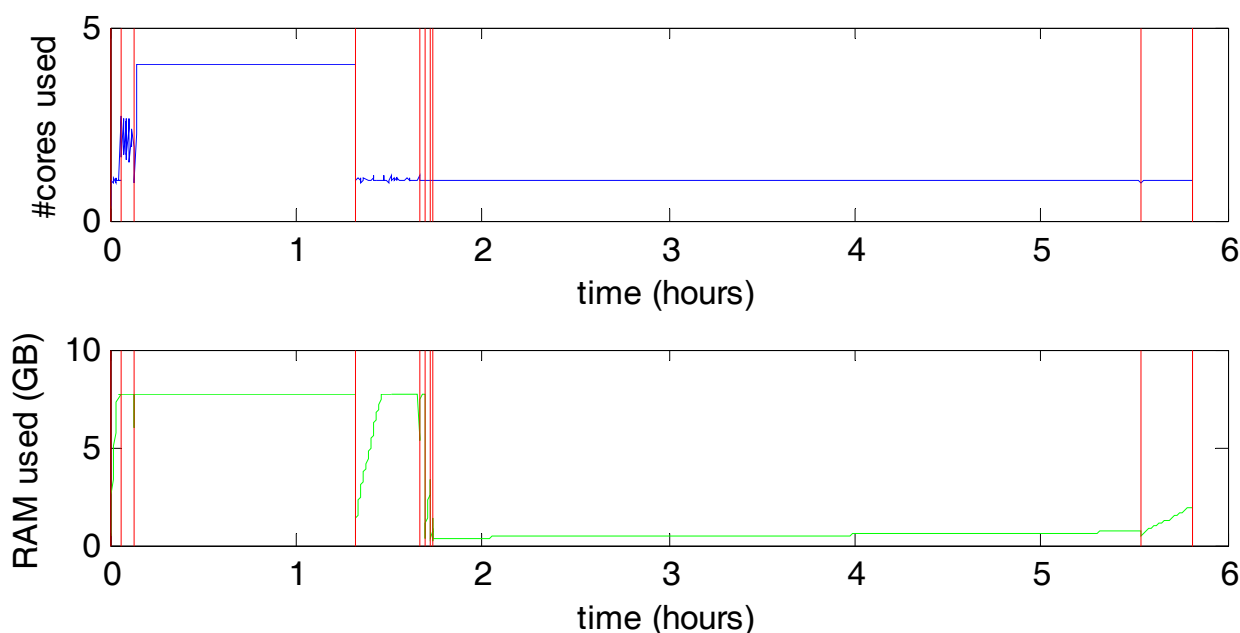


Figure 6.1: Resource occupancy of the CNV pipeline vs. time. Input read size of 2.6 GB, taken from the site of the 1000genomes projects.

Experimental setup:

Size and type of involved files:

- Initial VM size: 14 GB;
- Maximum allowable VM size: 300 GB;
- Genome files taken from <ftp://ftp.genomes.ebi.ac.uk/vol1/ftp/data/>

Experiments have been carried out in order to evaluate the execution time of the implemented pipelines when different VM resources, in terms of number of CPU cores and Memory RAM, are allocated when input files of different size are processed.

To this purpose, sequenced data available in the site of the 1000genomes projects (<ftp://ftp.genomes.ebi.ac.uk/vol1/ftp/data/>) have been used. In particular, data files with different sizes related to individuals labelled as “HG00097” and “HG00358” have been selected for the CNV VM. Analogous choices have been done for the DE pipeline.

The VM with the CNV pipeline has been executed by a workstation with 16 logical CPU cores (2quad-cores Intel Xeon CPUs with 2 threads, i.e. a total of 8 physical CPU cores and 8 virtual CPU cores) and 16 GB RAM. Different VM configurations that use 2, 4, 8 or 16 cores number and 4 GB, 8 GB or 16 GB RAM have been identified. For each of the 12 possible VM configurations, indicated in Table 6.1, a set of experiments have

been executed by processing the input files of different size and the corresponding required execution time and final VM size have been evaluated.

# CORES	RAM		
	04 GB	08 GB	12 GB
02	Conf. 1	Conf. 7	Conf. 9
04	Conf. 2	Conf. 6	Conf. 10
08	Conf. 3	Conf. 5	Conf. 11
16	Conf. 4	Conf. 8	Conf. 12

Table 6.1: Configurations of the VM that have been set for evaluating timing performance of the CNV pipeline

Results related to each VM configuration will be reported in an individual table. Just to provide an example, we have started executing experiments for the CNV pipeline. Table 6.2 refers to the VM configuration with maximum disk space that VM can allocate equal to 300 GB, 2 cores and 4 GB RAM. Moreover, each row of the table reports the experiment number, the patient label to which sequenced data refer, the file used and the corresponding size, the number of cores and the maximum allocated RAM set in the VM, the time needed for processing indicated files and the final VM size. File sizes used for this experiments set are respectively equal to 5 GB, 12 GB, 18 GB and 26 GB. Similar analysis have been done for all pipelines and all experimental configurations of interest in order to provide criteria for determining the optimal working point.

Exp #	Patient name	Input files size [GB]	IN [GB]	Cores #	VM Mem.[GB]	Time [h]	Final VM [GB]
01	HG00097	SRR765989_1 (2.8) SRR765989_2 (2.8)	5.6 GB	2	4096	7.15	73 GB
02	HG00097	SRR741384_1 (6.1) SRR741384_1 (6.1)	12.2 GB	2	4096	17.11	158 GB
03	HG00097	SRR741384_1 (6.1) SRR741384_1 (6.1) SRR765989_1 (2.8) SRR765989_2 (2.8)	17.8 GB	2	4096	22.62	217 GB
04	HG00097	SRR741385_1 (7.1) SRR741385_1 (7.0) SRR741384_1 (6.1) SRR741384_1 (6.1)	26.3 GB	2	4096	36.78	300 GB

Table 6.2: Sample of results collected by the assessment of the CNV pipeline.

6.1.1.2 Plans for assessing the medical interface.

Assessing the medical interface consists of verifying that the genomic services implemented in ARES can be started and executed correctly through this interface. It can be accessed through the web. Just to give a sample of the implemented interface, Figure 6.2 shows a snapshot of it.

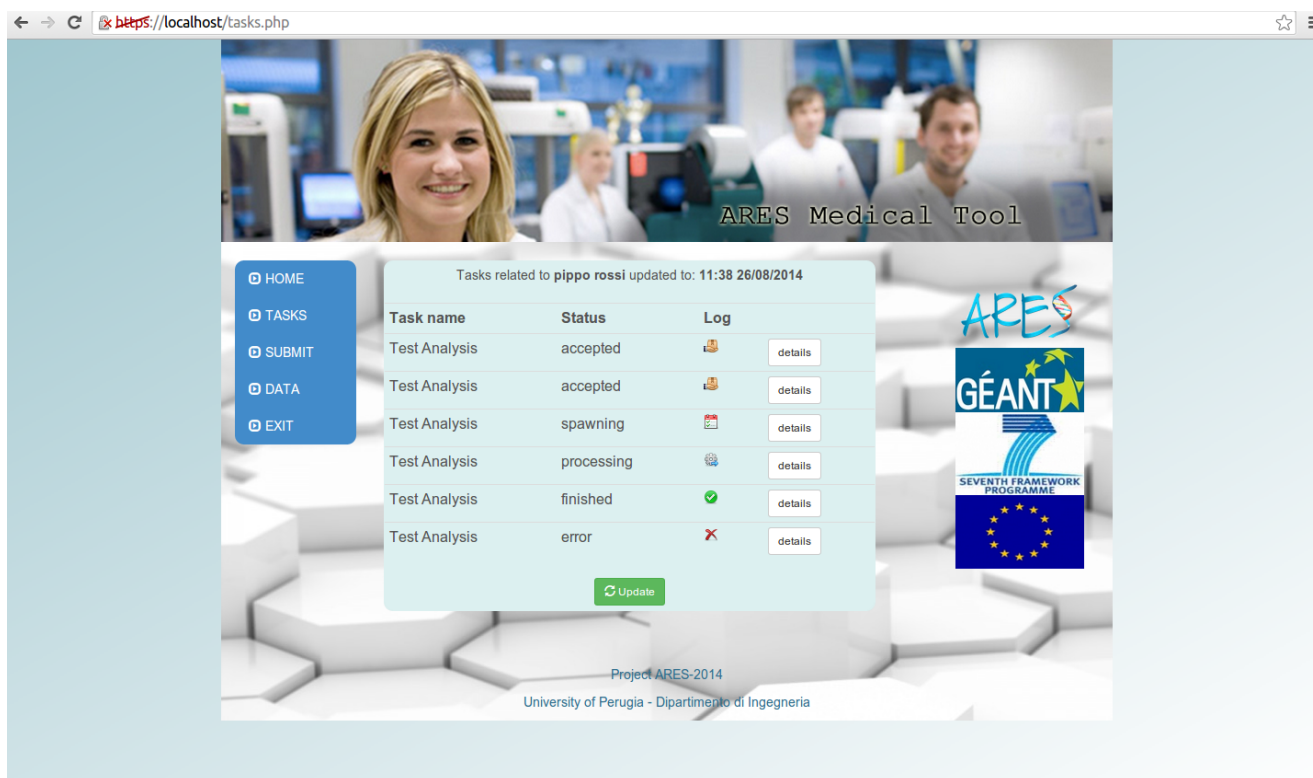


Figure 6.2: Flow diagram of the experiments to be used for assessing the ARES integrates system.

It is worth to consider that ARES will not produce a market-ready commercial tool, but rather a research tool usable by researchers. The assessment plans consists of verifying that:

- The interface is intuitive and user friendly. The individual evaluation of the GGB personnel allowed this assessment. Specific assessment has been carried through interview to the GGB personnel in order to collect general comments in order to improve the interface, which has been done. The usage of these end-users feedback contributed to improve significantly the usability of the interface.
- Correct execution of all the implemented services.
- Correct storage and retrieval of the output data by medical personnel.

6.1.1.3 Plans for assessing the cloud environment.

Plans for assessing the cloud environment refers to the cloud deployment in the software engineering laboratory of the Department of Engineering of the University of Perugia. At the time of writing, no Géant resources have been allocated to ARES yet. If access to some Géant resources will granted to ARES, these plans will be updated accordingly.

The cloud environment in ARES was designed by considering that its implementation is based on OpenStack [5].

OpenStack (<http://www.openstack.org/>) is a cloud management system that allows using both restful interface and APIs to control and manage the execution of virtual machines (VMs) on a server pool. It allows retrieving information about the computational and storage capabilities of both the cloud and the available VMs. It also allows triggering the boot of a selected VM with a specific hardware configuration. Thus, it is possible to wrap, inside a certain number of VMs, different existing software packages processing genome files. Each VM could expose, through OpenStack, the required configuration, which allows mapping the minimum required computational capabilities for a specific processing service into the hardware configuration for the VM that executes the service.

Virtual machine image files are stored within the storage units of the PoP managed by OpenStack and also available for download through a restful interface. In fact, OpenStack allows also to automatically inject, in the managed server pool, new VMs, thus allowing the system to retrieve the relevant files from other locations (i.e. other GÉANT PoPs). Furthermore, OpenStack is not bounded to the use of a single hypervisor, but it can make use of different drivers to support different formats of virtual machines and hypervisors, such as VMWare, KVM, Xen, and Hyper-V. In our implementation, we use the KVM hypervisor.

Assessment plans of the cloud environment are organized in *two phases*. The *first* phase consists of a simulation of the ARES cloud environment. The objective of this simulation phase is to analyze the suitability of the designed network architecture from the service perspective without having the possibility of accessing the Géant infrastructure. For this purpose, we have executed some simulations using the topology of the GÉANT IP network (Géant), illustrated in Figure 4.20 and shown also in Figure 6.3 for convenience. In this figure, each circle represents a national PoP, whose internal architecture is shown in Figure 4.2 and shown also in Figure 6.4 for convenience. We have assumed that the VM repository is reachable through the IT PoP, whereas the repository for annotations and auxiliary, reference genome files can be reached through the UK PoP.

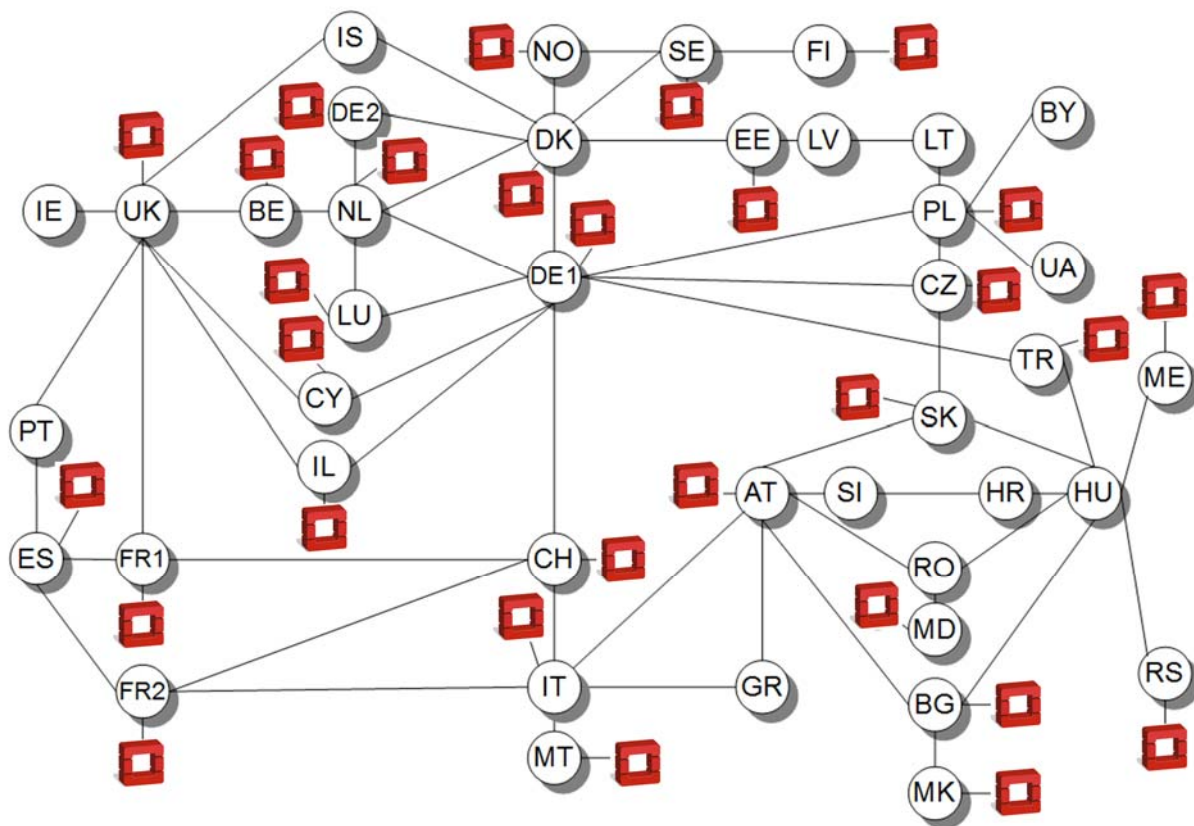


Figure 6.3:GÉANT IP network topology.

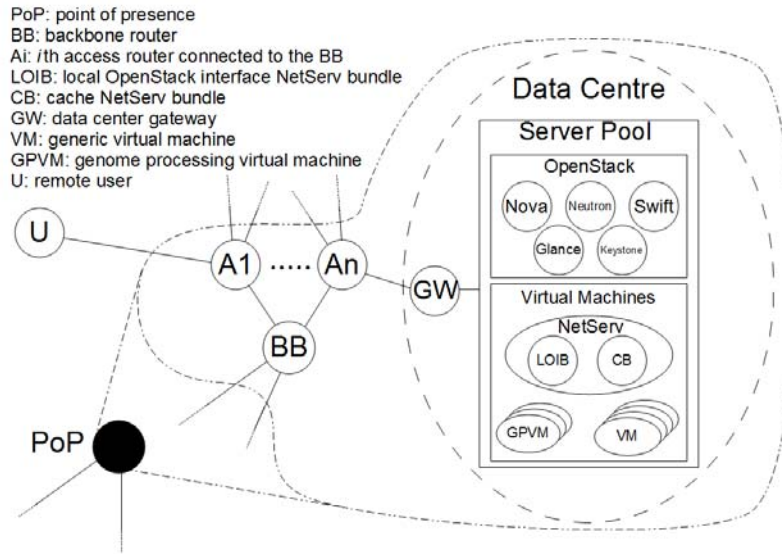


Figure 6.4: PoP architecture.

Service requests are modeled as a Poisson process, with average rate λ , generating from 5 to 45 requests per hour. For each value of λ , we have simulated 10000 genomic processing requests. Each request comes from a user connected to one of the PoPs of the topology. The users are randomly selected using a uniform distribution. We considered the execution of three different genomic processing pipelines. Each of them uses a VM, whose image file is 3 GBs large. The auxiliary and reference files, together with the necessary annotations for the three different processing pipelines are equal to 1, 10, and 20 GBs, respectively. The relevant execution times have been set to 3, 4, and 5 hours, including also the transfer time. Data flow can be shaped in the hypervisor or by another NetServ bundle, so as to avoid overwhelming the network and thus causing starvation to other type of traffic. According to the Géant map, the minimum bandwidth among PoPs has been set to 1 Gb/s. We assumed 10 Gigabit Ethernet connections inside PoPs. Each PoP can serve up to 5 genomic requests contemporarily, no matter of the selected pipeline. We assumed that all routers depicted in Figure 6.3 are NetServ-enabled routers. This can be accomplished either by resorting to a software defined networking solutions, which has been successfully tested and described in [4], or by porting the NetServ platform in a real router. We assumed that the caching capabilities (RAM plus disk space) in NetServ nodes is equal to 50 GBs. We assumed that only VM images and annotations/auxiliary files can be cached for future usage, but not genomes, for privacy constraints.

The proposed service architecture is compared with a basic cloud computing paradigm, in which in each PoP there is a computing node with the same characteristics of that in our solution. When a new service request arrives, the computing node is selected randomly among those available, and it is fed with VM image and input files (genome plus auxiliary files).

Figure 6.5 shows an initial set of results. It emerges that the ARES solution performs quite well. In fact, it allows decreasing up to about 5/6 times the network traffic in comparison to the reference solution, the basic cloud

computing paradigm, while providing the same service. This result is due to the ARES dynamic caching strategy, which strongly reduce the transmission of very large files on the backbone network.

Being the simulation results promising, in order to go further it is necessary to test the individual NetServ modules we have implemented: the Genome CDN manages (GCM), the cache bundle (CB), and the local OpenStack interface bundle (LOIB). For each of them, we consider functional and, when applicable, performance tests, in order to assess the correct functioning of each element of the overall system architecture.

The second phase of the *assessment plans of the cloud environment* consists of testing the real individual cloud components of OpenStack. The overall cloud environment testing is done automatically by the test involving the implemented genomic pipelines.

Experimental assessment investigated the following aspects:

- *Access transparency*: the set of CDN services are accessible regardless the user locations, to be verified experimentally. Success= successful verification for all locations.
- *Location transparency*: the NSIS signaling provides transparency to any change of the repository locations. Success=transparency verified for all PoPs.
- *Availability*: according to the CAP theorem, a distributed information system cannot guarantee consistency, availability, and partition-tolerance at the same time. The achievable availability for all CDN classes have been investigated in relation to the tolerable service time and the metrics illustrated below.
- *Failure transparency or Partition tolerance*: CDN service are robust to PoP and router failures. We show how the system can manage and overcome node failures. In particular, the client programs operate correctly after a server or repository failure. Repeated failures have been emulated so as to investigate and maximize the actual robustness. This metric is strictly related to access transparency.
- *Consistency*: the cache instantiation and update procedures guarantee data consistency. This metric is strictly related to location transparency. Repeated experiments, also in the presence of node failures, have been executed. Any experiment has been considered successful if all caches are synchronized with the relevant metadata.
- *Scalability*: CDN services allow increasing the tolerable network load and also scale gracefully to huge ones. Scalability has been analyzed and optimized in relation to the suitable trade-off induced by the CAP theorem.

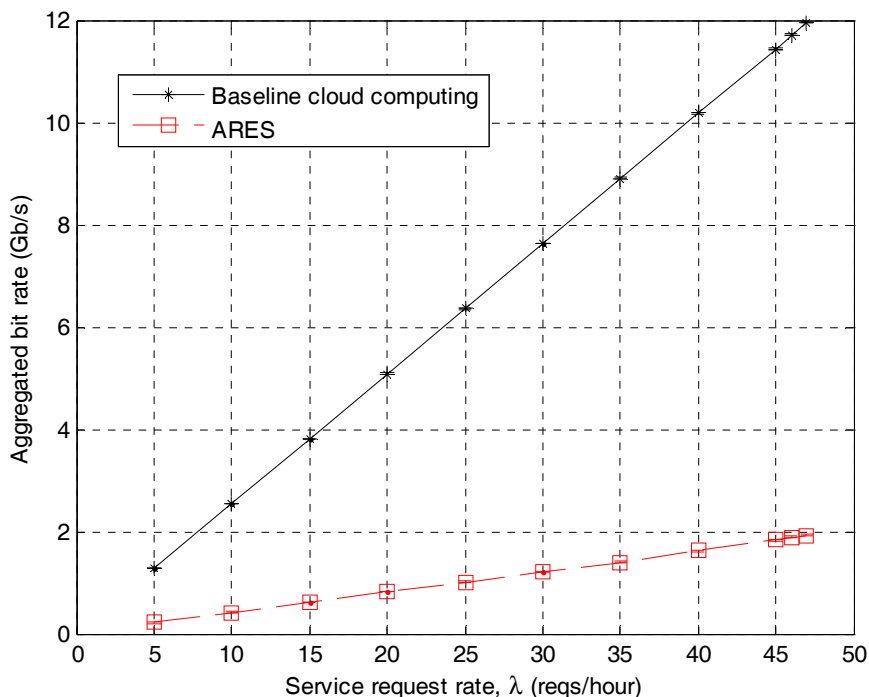


Figure 6.5: Simulations results: ARES vs. baseline cloud computing paradigm, as a function of the input load.

6.1.1.4 Plans for assessing the advanced offpath signalling.

This section illustrates the plan for assessing the NSIS offpath signalling[4].

ARES makes use of an extension of the IETF Next Steps in Signaling (NSIS) protocol suite[2][3]. Our proposal updates the NSIS transport layer. This way, the design of an NSIS-compliant application can leverage it without having to deal with low level signaling transport issues. In particular, we have proposed an extension of the General Internet Signaling Transport Protocol (GIST), including path decoupled signaling capabilities[3].

This extension, executable with none or minimal IP node configuration, requires the definition of a GIST peer discovery protocol for making each GIST node aware of its GIST peers. A further novelty consists of the delivery strategies of signaling messages, depending on the domain topological features. These strategies make use of the information collected during the GIST peer discovery phase and exploit the signaling interception capabilities of GIST.

Assessment has proceeded *experimentally*, in two phases. The *first phase* is devoted to assess the functions of the signaling protocol in an unloaded network, and it is supported by a theoretical analysis. This experimental assessment has been executed immediately after the protocol development. The *second phase* is devoted to assess if the signaling protocol is still able to provide its functions when the network is loaded by transfer of

genomic data sets, which can imply delays, timeouts, and retransmissions which were not present during the initial testing. The results of this type of assessment is reported in Section 6.2.2, together with all other system performance figures.

As for the first phase, tests have been carried out with different topologies, including the Géant one, shown in Figure 6.3 and Figure 6.2. The performance of the off-path signaling and dissemination protocols has been evaluated by implementing nodes running Ubuntu Server v.10.04. Each node can act as a backbone router in a PoP of an autonomous system (AS).

Two types of nodes have been implemented, *stub* and *core* nodes. Each stub is connected to one of the core nodes. NSIS extensions have been implemented within NSIS-ka [6].

Two overlay networks have been implemented:

- Full overlay topology: All nodes execute NSIS.
- Sparse overlay topology: Core nodes are legacy IP routers, not running NSIS. All the other nodes are NSIS compliant. This configuration is aimed to evaluating the impact of a partial introduction of our extended NSIS.

The first set of experiments evaluates the network discovery time and overhead. Some experiments are used to evaluate performance during the transient phase, when all nodes are turned on. The second set of experiments analyses the steady phase, when the GIST node discovery has been completed.

We define the convergence time of a node as the time taken for executing the transient phase, which is when at least one complete gossip session with allNSIS nodes has been executed. During the transient, we have evaluated also the overall network overhead.

Experiments completely characterize the signaling behaviour, in terms of time of accomplishment, and upper bound to the signaling load. The results of this phase have been highly successful, and they have been reported in Section 4.1.4. Not only the signaling protocol is highly efficient in terms of overhead, and able to converge in short times, but also its performance are easily predictable, as witnessed by the theoretical analysis presented in Section 4.1.4.

6.1.2 Plans for Assessing the ARES Integrated System

The ARES integrated system has been assessed though metrological validation tests. They consists of the repeated execution of data processing requiring different service time specifications. This approach allows stressing both the network and the individual components implemented in operation.

Assessment has been done for each implemented service. Tests proceeded according to the diagram shown in Figure 6.6. A service time is specified according to the service time specifications reported in Section 3. The specified service time must be lower than or equal to the service time specified in IM2.1. For the analyzed

service, repeated user requests have been generated. These requests trigger GCM service mapping over the network resources (PoP selection, CPU cores, RAM). After that, all files are transferred into the selected PoP and genomic processing starts. At the end of the execution, the service time is evaluated and compared with the specified service time. If the time requirements are satisfied, this time is decreased by an amount of time to be determined yet. Otherwise it is increased. Experiments are repeated for a number of times that allow full service characterization.

Note that the use of different service time specifications can induce the ARES optimization algorithm executed by the GCM to produce different results and different CDN mapping. This means that the repeated trials for each individual experiments are necessary.

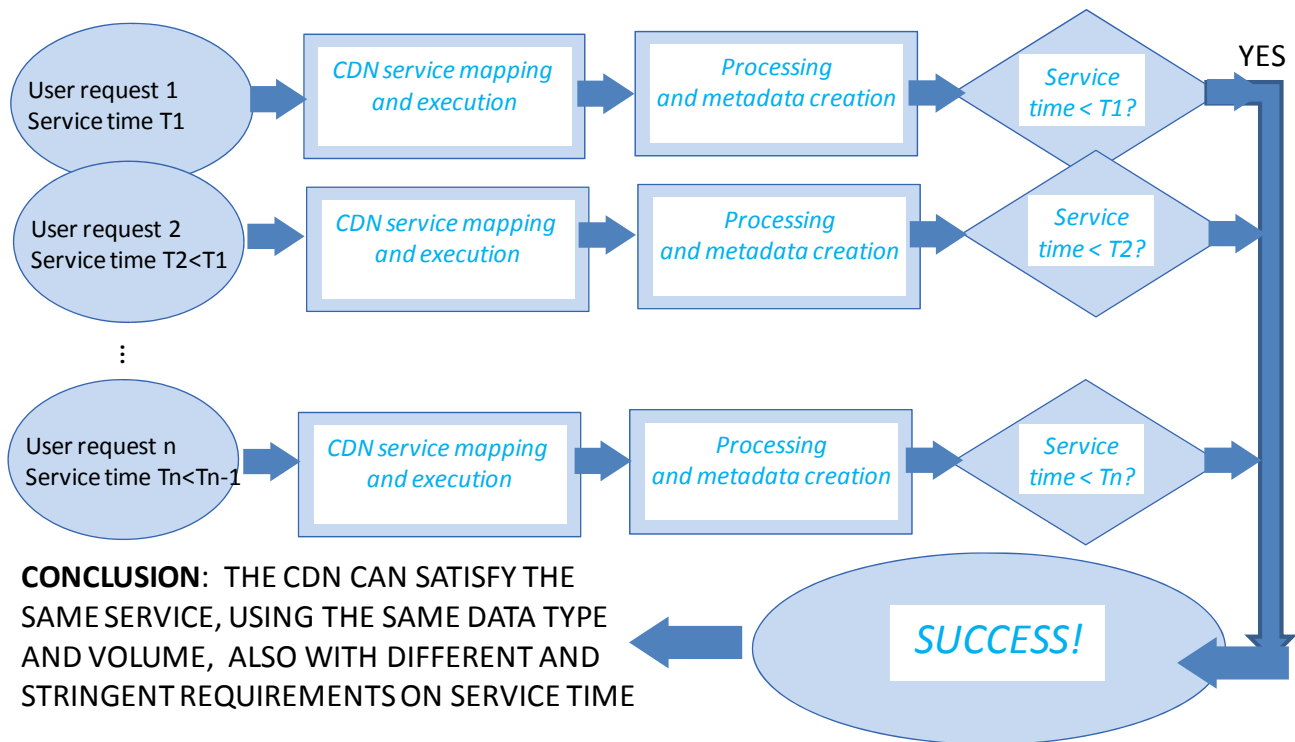


Figure 6.6: Flow diagram of the experiments to be used for assessing the ARES integrates system.

6.2 Execution of experiments and analysis of results

6.2.1 Genomic Pipelines

6.2.1.1 CNV Experimental setup

Size and type of involved files:

Open Call Deliverable - Advanced Networking for the EU genomic research (ARES)
D1:
Final ARES deliverable
 Document Code: GN3PLUS14-1284-49

- VM starting size: 14 GB;
- Maximum allowable VM size: 300 GB;
- archive for download anonymized genomes: <ftp://ftp.genomes.ebi.ac.uk/vol1/ftp/data/>

Experiments reported in this section have been carried out in order to evaluate timing execution of the implemented CNV pipeline when different VM resources, in terms of number of CPU cores and Memory RAM, are allocated when input files of different size are processed.

To this purpose, sequenced data available on the ftp site of the 1000genomes projects (<ftp://ftp.genomes.ebi.ac.uk/vol1/ftp/data/>) have been used. In particular, data files with different sizes related to individuals labelled as HG00097 have been downloaded and processed with the CNV VM.

The VM with CNV pipeline run on a workstation with 16 logical CPU cores (2quad-cores Intel Xeon CPUs with 2 threads) and 16 GB RAM. Different VM configuration that use 2, 4, 8 or 16 cores number and 4 GB, 8 GB or 16 GB RAM have been set. Table 6.3 reports the grid of each executed experiments. For each VM configuration, the input file size has been changed and the corresponding required execution time and final VM size have been evaluated.

# cores	RAM Size (GB)		
	04	08	12
02	Proc. 1	Proc. 7	Proc. 9
04	Proc. 2	Proc. 6	Proc. 10
08	Proc. 3	Proc. 5	Proc. 11
16	Proc. 4	Proc. 8	Proc. 12

Table 6.3: Experiments grid for CNV pipeline.

Results related to each VM configuration have been reported in a table. In particular, Table 6.4 refers to the the VM configuration with maximum disk space that VM can allocate equal to 300 GB, 2 cores numbers and 4 GB RAM. Moreover, each row of the table reports the experiment number, the patient label to which sequenced data refer, the file used and the corresponding size, the number of cores and the maximum allocated RAM set in the VM, the time needed for processing indicated files and the final VM size. File sizes used for this experiments set are respectively equal to 5 GB, 12 GB, 18 GB and 26 GB.

Processing 1

Exp.	Patient	Input files	In	Cores	VM mem	Time	Final VM
#	name	size [GB]	[GB]	#	[MB]	[h]	[GB]
01	HG00097	SRR765989_1 (2.8) SRR765989_2 (2.8)	5.6	2	4096	7.15	73.39
02	HG00097	SRR741384_1 (6.1) SRR741384_2 (6.1)	12.2 2	2	4096	17.11	158.01
03	HG00097	SRR741384_1 (6.1) SRR741384_2 (6.1) SRR765989_1 (2.8) SRR765989_2 (2.8)	17.8	2	4096	22.62	216.89
04	HG00097	SRR741385_1 (7.1) SRR741385_2 (7.0) SRR741384_1 (6.1) SRR741384_2 (6.1)	26.3	2	4096	36.78	300.55

Table 6.4: Experiments with 2 CPU cores and 4 GB of RAM.

Processing 2

Exp.	Patient	Input files	In	Cores	VM mem	Time	Final VM
#	name	size [GB]	[GB]	#	[MB]	[h]	[GB]
01	HG00097	SRR765989_1 (2.8) SRR765989_2 (2.8)	5.6	4	4096	6.48	73.39
02	HG00097	SRR741384_1 (6.1) SRR741384_2 (6.1)	12.2 2	4	4096	14.04	158.01
03	HG00097	SRR741384_1 (6.1) SRR741384_2 (6.1) SRR765989_1 (2.8) SRR765989_2 (2.8)	17.8	4	4096	16.87	216.89
04	HG00097	SRR741385_1 (7.1) SRR741385_2 (7.0) SRR741384_1 (6.1) SRR741384_2 (6.1)	26.3	4	4096	25.81	300.55

Table 6.5: Experiments with 4 CPU cores and 4 GB of RAM.

Processing 3

Exp.	Patient	Input files	In	Cores	VM mem	Time	Final VM
#	name	size [GB]	[GB]	#	[MB]	[h]	[GB]
01	HG00097	SRR765989_1 (2.8) SRR765989_2 (2.8)	5.6	8	4096	6.00	73.39
02	HG00097	SRR741384_1 (6.1) SRR741384_2 (6.1)	12.2 2	8	4096	12.17	158.01
03	HG00097	SRR741384_1 (6.1) SRR741384_2 (6.1) SRR765989_1 (2.8) SRR765989_2 (2.8)	17.8	8	4096	14.58	216.89
04	HG00097	SRR741385_1 (7.1) SRR741385_2 (7.0) SRR741384_1 (6.1) SRR741384_2 (6.1)	26.3	8	4096	20.64	300.70

Table 6.6: Experiments with 8 CPU cores and 4 GB of RAM.

Processing 4

Exp.	Patient	Input files	In	Cores	VM mem	Time	Final VM
#	name	size [GB]	[GB]	#	[MB]	[h]	[GB]
01	HG00097	SRR765989_1 (2.8) SRR765989_2 (2.8)	5.6	16	4096	6.05	73.54
02	HG00097	SRR741384_1 (6.1) SRR741384_2 (6.1)	12.2 2	16	4096	12.04	157.91
03	HG00097	SRR741384_1 (6.1) SRR741384_2 (6.1) SRR765989_1 (2.8) SRR765989_2 (2.8)	17.8	16	4096	14.35	216.89
04	HG00097	SRR741385_1 (7.1) SRR741385_2 (7.0) SRR741384_1 (6.1) SRR741384_2 (6.1)	26.3	16	4096	20.65	300.53

Table 6.7: Experiments with 16 CPU cores and 4 GB of RAM.

Processing 5

Exp.	Patient	Input files	In	Cores	VM mem	Time	Final VM
#	name	size [GB]	[GB]	#	[MB]	[h]	[GB]
01	HG00097	SRR765989_1 (2.8) SRR765989_2 (2.8)	5.6	8	8192	4.31	73.39
02	HG00097	SRR741384_1 (6.1) SRR741384_2 (6.1)	12.2 2	8	8192	10.08	157.96
03	HG00097	SRR741384_1 (6.1) SRR741384_2 (6.1) SRR765989_1 (2.8) SRR765989_2 (2.8)	17.8	8	8192	11.63	216.84
04	HG00097	SRR741385_1 (7.1) SRR741385_2 (7.0) SRR741384_1 (6.1) SRR741384_2 (6.1)	26.3	8	8192	16.95	300.52

Table 6.8: Experiments with 6 CPU cores and 8 GB of RAM.

Processing 6

Exp.	Patient	Input files	In	Cores	VM mem	Time	Final VM
#	name	size [GB]	[GB]	#	[MB]	[h]	[GB]
01	HG00097	SRR765989_1 (2.8) SRR765989_2 (2.8)	5.6	4	8192	4.74	73.39
02	HG00097	SRR741384_1 (6.1) SRR741384_2 (6.1)	12.2 2	4	8192	12.17	157.96
03	HG00097	SRR741384_1 (6.1) SRR741384_2 (6.1) SRR765989_1 (2.8) SRR765989_2 (2.8)	17.8	4	8192	14.22	216.84
04	HG00097	SRR741385_1 (7.1) SRR741385_2 (7.0) SRR741384_1 (6.1) SRR741384_2 (6.1)	26.3	4	8192	21.47	300.53

Table 6.9: Experiments with 4 CPU cores and 8 GB of RAM.

Processing 7

Exp.	Patient	Input files	In	Cores	VM mem	Time	Final VM
#	name	size [GB]	[GB]	#	[MB]	[h]	[GB]
01	HG00097	SRR765989_1 (2.8) SRR765989_2 (2.8)	5.6	2	8192	5.89	73.49
02	HG00097	SRR741384_1 (6.1) SRR741384_2 (6.1)	12.2 2	2	8192	16.69	158.10
03	HG00097	SRR741384_1 (6.1) SRR741384_2 (6.1) SRR765989_1 (2.8) SRR765989_2 (2.8)	17.8	2	8192	19.96	216.87
04	HG00097	SRR741385_1 (7.1) SRR741385_2 (7.0) SRR741384_1 (6.1) SRR741384_2 (6.1)	26.3	2	8192	32.55	300.53

Table 6.10: Experiments with 2 CPU cores and 8 GB of RAM.

Processing 8

Exp.	Patient	Input files	In	Cores	VM mem	Time	Final VM
#	name	size [GB]	[GB]	#	[MB]	[h]	[GB]
01	HG00097	SRR765989_1 (2.8) SRR765989_2 (2.8)	5.6	16	8192	4.27	73.39
02	HG00097	SRR741384_1 (6.1) SRR741384_2 (6.1)	12.2 2	16	8192	9.93	158.01
03	HG00097	SRR741384_1 (6.1) SRR741384_2 (6.1) SRR765989_1 (2.8) SRR765989_2 (2.8)	17.8	16	8192	11.51	216.89
04	HG00097	SRR741385_1 (7.1) SRR741385_2 (7.0) SRR741384_1 (6.1) SRR741384_2 (6.1)	26.3	16	8192	16.41	300.56

Table 6.11: Experiments with 16 CPU cores and 8 GB of RAM.

Processing 9

Exp.	Patient	Input files	In	Cores	VM mem	Time	Final VM
#	name	size [GB]	[GB]	#	[MB]	[h]	[GB]
01	HG00097	SRR765989_1 (2.8) SRR765989_2 (2.8)	5.6	2	12288	6.09	73.39
02	HG00097	SRR741384_1 (6.1) SRR741384_2 (6.1)	12.2 2	2	12288	17.01	158.01
03	HG00097	SRR741384_1 (6.1) SRR741384_2 (6.1) SRR765989_1 (2.8) SRR765989_2 (2.8)	17.8	2	12288	20.25	217.18
04	HG00097	SRR741385_1 (7.1) SRR741385_2 (7.0) SRR741384_1 (6.1) SRR741384_2 (6.1)	26.3	2	12288	33.00	300.56

Table 6.12: Experiments with 2 CPU cores and 12 GB of RAM.

Processing 10

Exp.	Patient	Input files	In	Cores	VM mem	Time	Final VM
#	name	size [GB]	[GB]	#	[MB]	[h]	[GB]
01	HG00097	SRR765989_1 (2.8) SRR765989_2 (2.8)	5.6	4	12288	4.76	73.39
02	HG00097	SRR741384_1 (6.1) SRR741384_2 (6.1)	12.2 2	4	12288	12.27	158.02
03	HG00097	SRR741384_1 (6.1) SRR741384_2 (6.1) SRR765989_1 (2.8) SRR765989_2 (2.8)	17.8	4	12288	14.36	216.89
04	HG00097	SRR741385_1 (7.1) SRR741385_2 (7.0) SRR741384_1 (6.1) SRR741384_2 (6.1)	26.3	4	12288	21.95	300.54

Table 6.13: Experiments with 4 CPU cores and 12 GB of RAM.

Processing 11

Exp.	Patient	Input files	In	Cores	VM mem	Time	Final VM
#	name	size [GB]	[GB]	#	[MB]	[h]	[GB]
01	HG00097	SRR765989_1 (2.8) SRR765989_2 (2.8)	5.6	8	12288	4.43	73.63
02	HG00097	SRR741384_1 (6.1) SRR741384_2 (6.1)	12.2 2	8	12288	10.41	158.11
03	HG00097	SRR741384_1 (6.1) SRR741384_2 (6.1) SRR765989_1 (2.8) SRR765989_2 (2.8)	17.8	8	12288	12.13	216.89
04	HG00097	SRR741385_1 (7.1) SRR741385_2 (7.0) SRR741384_1 (6.1) SRR741384_2 (6.1)	26.3	8	12288	17.52	300.60

Table 6.14: Experiments with 8 CPU cores and 12 GB of RAM.

Processing 12

Exp.	Patient	Input files	In	Cores	VM mem	Time	Final VM
#	name	size [GB]	[GB]	#	[MB]	[h]	[GB]
01	HG00097	SRR765989_1 (2.8) SRR765989_2 (2.8)	5.6	16	12288	4.24	73.83
02	HG00097	SRR741384_1 (6.1) SRR741384_2 (6.1)	12.2 2	16	12288	10.00	157.63
03	HG00097	SRR741384_1 (6.1) SRR741384_2 (6.1) SRR765989_1 (2.8) SRR765989_2 (2.8)	17.8	16	12288	11.74	216.89
04	HG00097	SRR741385_1 (7.1) SRR741385_2 (7.0) SRR741384_1 (6.1) SRR741384_2 (6.1)	26.3	16	12288	16.60	300.56

Table 6.15: Experiments with 16 CPU cores and 12 GB of RAM.

6.2.1.2 DE Experimental setup

Experiments reported in this section have been carried out in order to evaluate timing execution of the implemented DE pipeline when different VM resources, in terms of number of CPU cores and Memory RAM, are allocated when input files of different size are processed.

To this purpose, sequenced data available on the ftp site of the 1000genomes projects (<ftp://ftp.genomes.ebi.ac.uk/vol1/ftp/data/>) have been used. In particular, data files with different sizes related to

individuals labelled as HG00097, HG00259, HG00334, and HG00358 have been downloaded and processed with the DE VM.

The VM with CNV pipeline run on a workstation with 24 logical CPU cores (4 processors Six-Core AMD Opteron(tm) Processor 8425 HE 2.1 GHz) and 64 GB RAM. Different VM configuration that use 2, 4, 8 or 16 cores number and 32 GB, 40 GB or 64 GB RAM have been set. Table 6.16 reports the grid of each executed experiments. For each VM configuration, the input file size has been changed and the corresponding required execution time and final VM size have been evaluated.

Results related to each VM configuration have been reported in a table. The format is the same used for the CNV tables.

# cores	RAM Size (GB)		
	32	40	64
02	Proc. 4	Proc. 8	Proc. 12
04	Proc. 3	Proc. 7	Proc. 11
06	Proc. 2	Proc. 6	Proc. 10
16	Proc. 1	Proc. 5	Proc. 9

Table 6.16: Experiments grid for DE pipeline.

Processing 1

Exp.	Patient	Input files	In	Cores	VM mem	Time	Final VM
#	name	size [GB]	[GB]	#	[MB]	[h]	[GB]
01	HG00358	ERR188089_1 [2.10] ERR188089_2 [2.12]	4.22	16	32768	2.77	105.72
02	HG00097	ERR188231_1 [3.15] ERR188231_2 [3.18]	6.33	16	32768	3.72	126.20
03	HG00097	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188089_1 [2.10] ERR188089_2 [2.12]	10.55	16	32768	6.32	176.63
04	HG00097 HG00259	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188378_1 [3.70] ERR188378_2 [3.73]	13.76	16	32768	8.57	215.77
05	HG00097 HG00259 HG00334	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188378_1 [3.70] ERR188378_2 [3.73] ERR188127_1 [3.70] ERR188127_2 [3.73]	17.87	16	32768	11.18	260.70

Table 6.17: Experiments with 16 CPU cores and 32 GB of RAM.

Processing 2

Exp.	Patient	Input files	In	Cores	VM mem	Time	Final VM
#	name	size [GB]	[GB]	#	[MB]	[h]	[GB]
01	HG00358	ERR188089_1 [2.10] ERR188089_2 [2.12]	4.22	8	32768	2.88	105.84
02	HG00097	ERR188231_1 [3.15] ERR188231_2 [3.18]	6.33	8	32768	5.75	176.49
03	HG00097	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188089_1 [2.10] ERR188089_2 [2.12]	10.55	8	32768	5.77	176.49
04	HG00097 HG00259	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188378_1 [3.70] ERR188378_2 [3.73]	13.76	8	32768	9.00	215.74
05	HG00097 HG00259 HG00334	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188378_1 [3.70] ERR188378_2 [3.73] ERR188127_1 [3.70] ERR188127_2 [3.73]	17.87	8	32768	10,79	260.95

Table 6.18: Experiments with 8 CPU cores and 32 GB of RAM.

Processing 3

Exp.	Patient	Input files	In	Cores	VM mem	Time	Final VM
#	name	size [GB]	[GB]	#	[MB]	[h]	[GB]
01	HG00358	ERR188089_1 [2.10] ERR188089_2 [2.12]	4.22	4	32768	3.04	105.84
02	HG00097	ERR188231_1 [3.15] ERR188231_2 [3.18]	6.33	4	32768	3.85	126.20
03	HG00097	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188089_1 [2.10] ERR188089_2 [2.12]	10.55	4	32768	6.79	176.38
04	HG00097 HG00259	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188378_1 [3.70] ERR188378_2 [3.73]	13.76	4	32768	9.12	260.65
05	HG00097 HG00259 HG00334	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188378_1 [3.70] ERR188378_2 [3.73] ERR188127_1 [3.70] ERR188127_2 [3.73]	17.87	4	32768	11,20	260.65

Table 6.19: Experiments with 4 CPU cores and 32 GB of RAM.

Processing 4

Exp.	Patient	Input files	In	Cores	VM mem	Time	Final VM
#	name	size [GB]	[GB]	#	[MB]	[h]	[GB]
01	HG00358	ERR188089_1 [2.10] ERR188089_2 [2.12]	4.22	2	32768	2.83	105.83
02	HG00097	ERR188231_1 [3.15] ERR188231_2 [3.18]	6.33	2	32768	3.90	126.09
03	HG00097	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188089_1 [2.10] ERR188089_2 [2.12]	10.55	2	32768	6.69	215.59
04	HG00097 HG00259	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188378_1 [3.70] ERR188378_2 [3.73]	13.76	8	32768	9.00	215.74
05	HG00097 HG00259 HG00334	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188378_1 [3.70] ERR188378_2 [3.73] ERR188127_1 [3.70] ERR188127_2 [3.73]	17.87	2	32768	11,67	260.62

Table 6.20: Experiments with 2 CPU cores and 32 GB of RAM.

Processing 5

Exp.	Patient	Input files	In	Cores	VM mem	Time	Final VM
#	name	size [GB]	[GB]	#	[MB]	[h]	[GB]
01	HG00358	ERR188089_1 [2.10] ERR188089_2 [2.12]	4.22	16	40960	2.95	105.83
02	HG00097	ERR188231_1 [3.15] ERR188231_2 [3.18]	6.33	16	40960	3.54	125.99
03	HG00097	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188089_1 [2.10] ERR188089_2 [2.12]	10.55	16	40960	5.80	176.41
04	HG00097 HG00259	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188378_1 [3.70] ERR188378_2 [3.73]	13.76	16	40960	8.78	215.64
05	HG00097 HG00259 HG00334	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188378_1 [3.70] ERR188378_2 [3.73] ERR188127_1 [3.70] ERR188127_2 [3.73]	17.87	16	40960	10,97	260.70

Table 6.21: Experiments with 16 CPU cores and 40GB of RAM.

Processing 6

Exp.	Patient	Input files	In	Cores	VM mem	Time	Final VM
#	name	size [GB]	[GB]	#	[MB]	[h]	[GB]
01	HG00358	ERR188089_1 [2.10] ERR188089_2 [2.12]	4.22	8	40960	2.43	105.71
02	HG00097	ERR188231_1 [3.15] ERR188231_2 [3.18]	6.33	8	40960	3.81	125.98
03	HG00097	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188089_1 [2.10] ERR188089_2 [2.12]	10.55	8	40960	6.36	176.48
04	HG00097 HG00259	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188378_1 [3.70] ERR188378_2 [3.73]	13.76	8	40960	8.95	215.74
05	HG00097 HG00259 HG00334	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188378_1 [3.70] ERR188378_2 [3.73] ERR188127_1 [3.70] ERR188127_2 [3.73]	17.87	8	40960	10,63	260.54

Table 6.22: Experiments with 8 CPU cores and 40 GB of RAM.

Processing 7

Exp.	Patient	Input files	In	Cores	VM mem	Time	Final VM
#	name	size [GB]	[GB]	#	[MB]	[h]	[GB]
01	HG00358	ERR188089_1 [2.10] ERR188089_2 [2.12]	4.22	4	40960	3.33	105.71
02	HG00097	ERR188231_1 [3.15] ERR188231_2 [3.18]	6.33	4	40960	4.24	125.98
03	HG00097	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188089_1 [2.10] ERR188089_2 [2.12]	10.55	4	40960	6.95	176.39
04	HG00097 HG00259	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188378_1 [3.70] ERR188378_2 [3.73]	13.76	4	40960	8.82	215.73
05	HG00097 HG00259 HG00334	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188378_1 [3.70] ERR188378_2 [3.73] ERR188127_1 [3.70] ERR188127_2 [3.73]	17.87	4	40960	10,63	241.96

Table 6.23: Experiments with 4 CPU cores and 40 GB of RAM.

Processing 8

Exp.	Patient	Input files	In	Cores	VM mem	Time	Final VM
#	name	size [GB]	[GB]	#	[MB]	[h]	[GB]
01	HG00358	ERR188089_1 [2.10] ERR188089_2 [2.12]	4.22	2	40960	2.94	105.82
02	HG00097	ERR188231_1 [3.15] ERR188231_2 [3.18]	6.33	2	40960	3.72	126.10
03	HG00097	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188089_1 [2.10] ERR188089_2 [2.12]	10.55	2	40960	6.20	176.38
04	HG00097 HG00259	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188378_1 [3.70] ERR188378_2 [3.73]	13.76	2	40960	8.95	215.72
05	HG00097 HG00259 HG00334	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188378_1 [3.70] ERR188378_2 [3.73] ERR188127_1 [3.70] ERR188127_2 [3.73]	17.87	2	40960	11,63	260.64

Table 6.24: Experiments with 8 CPU cores and 40 GB of RAM.

Processing 9

Exp.	Patient	Input files	In	Cores	VM mem	Time	Final VM
#	name	size [GB]	[GB]	#	[MB]	[h]	[GB]
01	HG00358	ERR188089_1 [2.10] ERR188089_2 [2.12]	4.22	16	65536	2.26	105.72
02	HG00097	ERR188231_1 [3.15] ERR188231_2 [3.18]	6.33	16	65536	3.67	125.99
03	HG00097	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188089_1 [2.10] ERR188089_2 [2.12]	10.55	16	65536	6.05	176.41
04	HG00097 HG00259	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188378_1 [3.70] ERR188378_2 [3.73]	13.76	16	65536	9.30	215.64
05	HG00097 HG00259 HG00334	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188378_1 [3.70] ERR188378_2 [3.73] ERR188127_1 [3.70] ERR188127_2 [3.73]	17.87	16	65536	11,93	260.70

Table 6.25: Experiments with 16 CPU cores and 64 GB of RAM.

Processing 10

Exp.	Patient	Input files	In	Cores	VM mem	Time	Final VM
#	name	size [GB]	[GB]	#	[MB]	[h]	[GB]
01	HG00358	ERR188089_1 [2.10] ERR188089_2 [2.12]	4.22	8	65536	2.98	105.83
02	HG00097	ERR188231_1 [3.15] ERR188231_2 [3.18]	6.33	8	65536	3.72	125.98
03	HG00097	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188089_1 [2.10] ERR188089_2 [2.12]	10.55	8	65536	6.82	176.26
04	HG00097 HG00259	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188378_1 [3.70] ERR188378_2 [3.73]	13.76	8	65536	8.77	215.74
05	HG00097 HG00259 HG00334	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188378_1 [3.70] ERR188378_2 [3.73] ERR188127_1 [3.70] ERR188127_2 [3.73]	17.87	8	65536	11,48	260.73

Table 6.26: Experiments with 8 CPU cores and 64 GB of RAM.

Processing 11

Exp.	Patient	Input files	In	Cores	VM mem	Time	Final VM
#	name	size [GB]	[GB]	#	[MB]	[h]	[GB]
01	HG00358	ERR188089_1 [2.10] ERR188089_2 [2.12]	4.22	4	65536	3.60	105.87
02	HG00097	ERR188231_1 [3.15] ERR188231_2 [3.18]	6.33	4	65536	4.17	126.25
03	HG00097	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188089_1 [2.10] ERR188089_2 [2.12]	10.55	4	65536	7.84	176.58
04	HG00097 HG00259	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188378_1 [3.70] ERR188378_2 [3.73]	13.76	4	65536	10.46	215.79
05	HG00097 HG00259 HG00334	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188378_1 [3.70] ERR188378_2 [3.73] ERR188127_1 [3.70] ERR188127_2 [3.73]	17.87	4	65536	14.57	260.65

Table 6.27: Experiments with 4 CPU cores and 64 GB of RAM.

Processing 12

Exp.	Patient	Input files	In	Cores	VM mem	Time	Final VM
#	name	size [GB]	[GB]	#	[MB]	[h]	[GB]
01	HG00358	ERR188089_1 [2.10] ERR188089_2 [2.12]	4.22	2	65536	3.60	106.20
02	HG00097	ERR188231_1 [3.15] ERR188231_2 [3.18]	6.33	2	65536	4.17	126.06
03	HG00097	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188089_1 [2.10] ERR188089_2 [2.12]	10.55	2	65536	7.84	176.47
04	HG00097 HG00259	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188378_1 [3.70] ERR188378_2 [3.73]	13.76	2	65536	10.46	215.72
05	HG00097 HG00259 HG00334	ERR188231_1 [3.15] ERR188231_2 [3.18] ERR188378_1 [3.70] ERR188378_2 [3.73] ERR188127_1 [3.70] ERR188127_2 [3.73]	17.87	2	32768	14.57	260.64

Table 6.28: Experiments with 2 CPU cores and 64 GB of RAM.

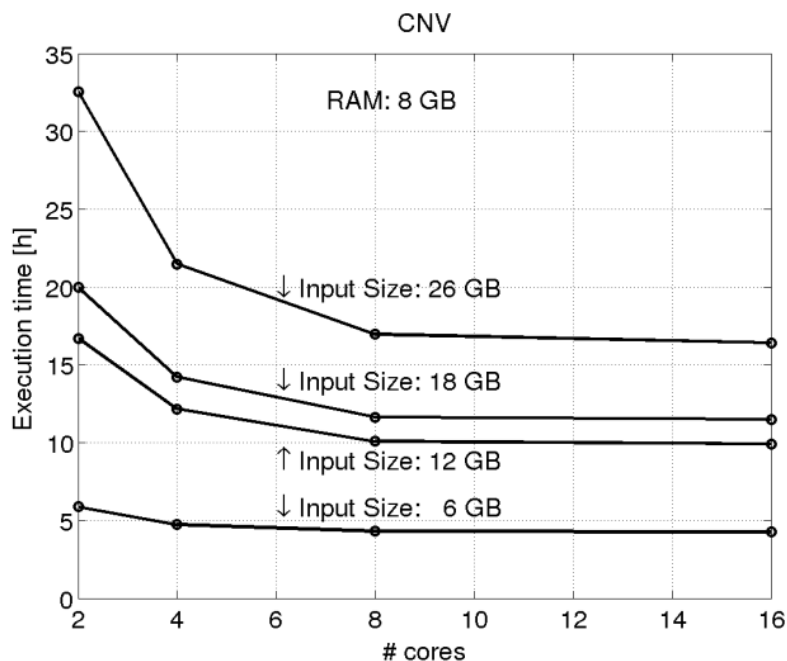


Figure 6.7: Execution times in hours versus number of CPU cores allocated to the VM for the CNV pipeline; input size is indicated close to each curve.

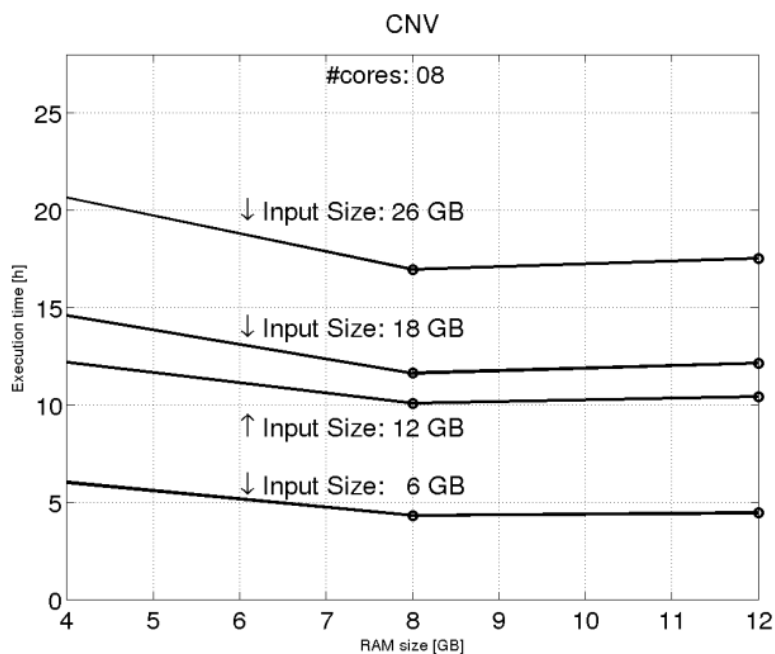


Figure 6.8: Execution times in hours versus amount of memory allocated to the VM for the CNV pipeline; input size is indicated close to each curve..

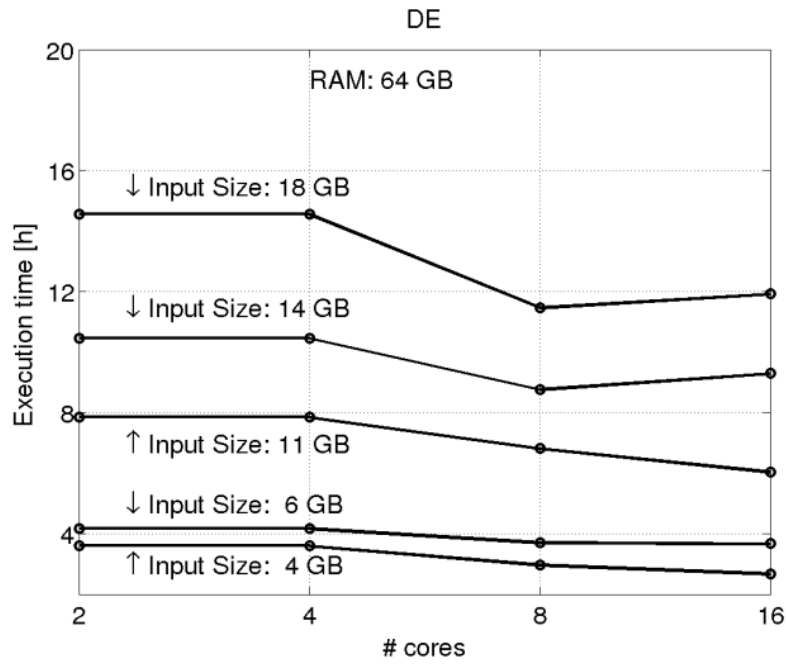


Figure 6.9: Execution times in hours versus number of CPU cores allocated to the VM for the DE pipeline; input size is indicated close to each curve.

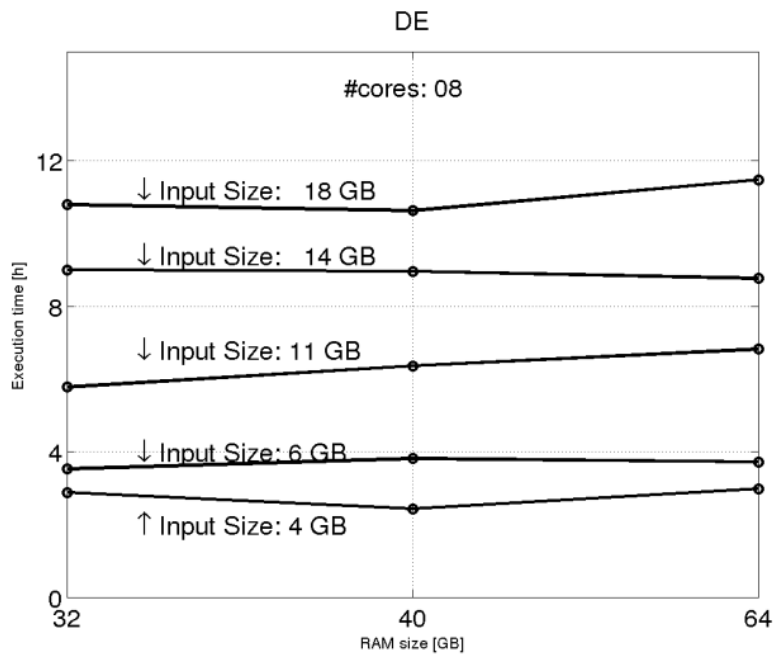


Figure 6.10: Execution times in hours versus amount of memory allocated to the VM for the DE pipeline; input size is indicated close to each curve.

In addition to the resource usage efficiency, also the processing time is of paramount importance, since it is the amount of time to be subtracted from the total desired service time for obtaining the maximum amount of time usable for handling all network issues. Figure 6.7, Figure 6.8, Figure 6.9, Figure 6.10 shows some sample results of an extensive experimental campaign using the CVN and DE pipelines. We report the name of them read files downloaded from 1000 genomes project site, the combination of which allows obtaining the input size for each curve. More details can be found the tables reported above. The execution time is shown as a function of both the number of cores and the amount of RAM initially allocated to the processing VM. What emerges from figures is that both CNV and DE pipelines are quite unaffected by the allocation of an amount of RAM larger than the minimum requirements. This rises further suspects that the software design is quite inefficient also in the RAM management. The slight performance improvement due to an increased number of CPU cores is limited to the initial processing phases of the pipelines. The clear usable information deriving from these results is that for introducing different service classes, distinguished by the service time, the most effective approach consists of making use of different input sizes. In fact, the execution time scales almost linearly with the input size, whilst is quite unaffected by over-allocation of RAM and CPU cores. This conclusion clearly holds when some popular genomic software packages are used and can be reconsidered when most of bioinformatics will make use of highly parallelized software currently being implemented. In more practical terms, it means that in the near future, when serious medical situations have to be handled, the patient's genome has to be processed singularly, since the processing time of aggregated multiple genomic reads, as it is frequently done in genomic computing, cannot be significantly reduced by over-allocating computing resources. We stress that this approach is a *transient* solution that minimizes the effect of an inefficient software design, which actually is an open issue in genomics. In case of very large amount of genomic data sets to process, a viable alternative to strongly reduce the processing time is the usage of Hadoop, in case the whole desired pipeline is implemented according to the Map-Reduce framework. In fact, currently, even if the number of bioinformatics tools implemented in Hadoop is increasing, a large number of them has not still been ported.

6.2.2 Network performance

In this section we illustrate the results of large scale tests, used to assess the performance of the system as a whole. We have done two types of tests:

1. Functional tests of the complete system over the testbed illustrated in Figure 5.1. We have used this testbed to perform a complete assessment of the functions developed in the ARES project, and to carry out the extensive genomic processing tests by using the cloud management functions offered by OpenStack. The results of these test have been illustrated in section 6.2.1.
2. Network performance tests used to measure the effectiveness and efficiency of the developed networked system. In this case, we have used the large scale testbed illustrated in Figure 5.2, where all operations regarding network transfer are executed, but final genomic processing are emulated. The results of these tests are reported in this section.

In order to correctly estimate the duration of these emulated processing, we have numerically fitted the duration of real processing presented in section 6.2.1. In this way, we have found a number of empirical laws able to realistically map the input size into the corresponding processing time. As for the input size, we have chosen file size randomly selected in the set $\text{Input}=\{1, 2, \dots, 15\}$ GB. The node issuing the service request is randomly selected, with an uniform distribution. The arrival rate of these requests has been generated according to a Poisson process, with average rate of 5 requests/hour.

In order to exploit the (inefficient) behaviour of the genomic pipelines shown in Figure 6.1, we have assumed that OpenStack will resize the number of CPU cores and amount of RAM after the 30% of the processing time has elapsed. In particular, the number of CPU core is reduced to 1 CPU core, whereas the amount of RAM is reduced as 25% of original allocation.

As for the large scale test, in this section we present two different datasets.

1. The first is relevant to a single type of request (homogeneous tests), corresponding to the CNV processing pipeline (pipeline 2 in Table 6.29). We have assumed that this pipeline has been scheduled with the maximum priority, and thus these results are useful to understand the system performance seen by the class of service with maximum priority
2. The second dataset is instead relevant to a large scale test, where the usage of 20 different pipelines have been emulated. Some of these pipelines share some reference/auxiliary files needed for computation. In this case, we assumed that all different types of requests have the same priority. The size of all involved reference/auxiliary/VM image files is reported in Table 6.29. The pipeline popularity has been modeled by the Mandelbrot-Zipf distribution, typically used in content distribution models [34][35][36], with parameter $\alpha=1$, and plateau factor $q=4$. This parameter values allow obtaining a popularity rank with a modest imbalance between requests.

In both cases, we have compared the performance of ARES system with that of a baseline cloud computing system, in which the selection of the PoP where performing the computation follows this algorithm:

1. If the PoP from which the request is originated has enough computing capability (CPU cores, RAM, disk size), the request is allocated to the data centre connected to that PoP;
2. if the condition above is not verified, the request is randomly allocated to a data centre which has enough computing resources to handle the request.

Since ARES caches files (auxiliary files, reference files, VM image files) of large files (several GB, see data in Table 6.29), we handle this process by dividing those file in multiple chunks of equal size, selected to 2 GB.

The cache in both routers and data centre is implemented by disk size, since the RAM allocated to that nodes, due to the constraints imposed by our testbed, is used to handle signaling, decision logic and, above all, data transfer (Java buffers for streaming data). Each cache, both on router or data centre node, has the capability to cache data up to 40 GB, which reside in a storage module which most of times is connected on an USB 3.0

port. Clearly, using RAM or internal SSD modules to cache contents would imply much better performance in terms of download times. Thus, performance reported here in terms of download time has to be considered a sort of upper bound.

All emulated data centres have the same service capacity, which is equal to

1. 16 CPU cores;
2. 64 GB of RAM;
3. 2 TB of disk size. The storage is partitioned in this way:
 - a. 20 GB allocated to store VM image files in the Glance module of OpenStack [5],
 - b. 180 GB to handle the VM which runs both the cache (CB) and the local manager (LOIB),
 - c. 1.8 TB to store processing VM disks.

The overall cacheable data set consists of 186 GB.

Pipeline index	# CPU cores	RAM (GB)	VM size (GB)	Execution time (h)	Auxiliary/Reference/VM image Files id	File Size (GB)
1	2	32	Min 107 Max 205	Min 1.6 Max 18.4	File1	24
					File2	0.88
					File3	0.88
					File4	0.006
					VM 1	3
2	2	8	Min 110 Max 250	Min 1.6 Max 18.4	File5	3.8
					File2	0.88
					File3	0.88
					VM 2	3
3	2	8	Min 113 Max 295	Min 1.6 Max 18.4	File5	3.8
					File2	0.88
					File3	0.88
					File6	2.9
					VM 3	3

Pipeline index	# CPU cores	RAM (GB)	VM size (GB)	Execution time (h)	Auxiliary/Reference/VM image Files id	File Size (GB)
4	2	5	Min 105 Max 175	Min 1.6 Max 18.4	File7	24
					File8	0.88
					File9	0.88
					File10	0.006
					File11	9.6
					VM 4	3
5	2	32	Min 107 Max 205	Min 1.6 Max 18.4	File12	3.8
					File8	0.88
					File9	0.88
					File10	0.006
					File13	8.6
					VM 5	3
6	2	8	Min 110 Max 250	Min 1.6 Max 18.4	File14	5.1
					File8	0.88
					File9	0.88
					File10	0.006
					File15	6.2
					VM 6	3
7	2	8	Min 113 Max 295	Min 1.6 Max 18.4	File12	3.8
					File8	0.88
					File9	0.88
					File16	2.9
					VM 7	3
8	2	5	Min 105	Min 1.6	File1	24

Pipeline index	# CPU cores	RAM (GB)	VM size (GB)	Execution time (h)	Auxiliary/Reference/VM image Files id	File Size (GB)
			Max 175	Max 18.4	File2	0.88
					File3	0.88
					File4	0.006
					File17	9.6
					VM 8	3
9	2	32	Min 107 Max 205	Min 1.6 Max 18.4	File5	3.8
					File2	0.88
					File3	0.88
					File4	0.006
					File18	8.6
					VM 9	3
10	2	8	Min 110 Max 250	Min 1.6 Max 18.4	File19	5.1
					File2	0.88
					File3	0.88
					File20	6.2
					VM 10	3
11	2	8	Min 113 Max 295	Min 1.6 Max 18.4	File6	3.8
					File2	0.88
					File3	0.88
					File6	2.9
					VM 11	3
12	2	5	Min 105 Max 175	Min 1.6 Max 18.4	File7	24
					File8	0.88
					File9	0.88
					File10	0.006

Pipeline index	# CPU cores	RAM (GB)	VM size (GB)	Execution time (h)	Auxiliary/Reference/VM image Files id	File Size (GB)
					File11	9.6
					VM 12	3
13	2	32	Min 107 Max 205	Min 1.6 Max 18.4	File12	3.8
					File8	0.88
					File9	0.88
					File10	0.006
					File13	8.6
					VM 13	3
14	2	8	Min 110 Max 250	Min 1.6 Max 18.4	File14	5.1
					File8	0.88
					File9	0.88
					File10	0.006
					File15	6.2
					VM 14	3
15	2	8	Min 113 Max 295	Min 1.6 Max 18.4	File12	3.8
					File8	0.88
					File9	0.88
					File16	2.9
					VM 15	3
16	2	5	Min 105 Max 175	Min 1.6 Max 18.4	File1	24
					File2	0.88
					File3	0.88
					File4	0.006
					File17	9.6

Pipeline index	# CPU cores	RAM (GB)	VM size (GB)	Execution time (h)	Auxiliary/Reference/VM image Files id	File Size (GB)
					VM 16	3
17	2	32	Min 107 Max 205	Min 1.6 Max 18.4	File5	3.8
					File2	0.88
					File3	0.88
					File4	0.006
					File18	8.6
					VM 17	3
18	2	8	Min 110 Max 250	Min 1.6 Max 18.4	File19	5.1
					File2	0.88
					File3	0.88
					File20	6.2
					VM 18	3
19	2	8	Min 113 Max 295	Min 1.6 Max 18.4	File6	3.8
					File2	0.88
					File3	0.88
					File6	2.9
					VM 19	3
20	2	5	Min 105 Max 175	Min 1.6 Max 18.4	File7	24
					File8	0.88
					File9	0.88
					File10	0.006
					File11	9.6
					VM 20	3

Table 6.29: File size and processing for all emulated genomics pipeline.

6.2.2.1 Results for high priority requests

In this section, we report the results relevant to test carried out with a single pipeline. Each service request comes with an input files with size randomly selected from the Input set. The PoP from which the requests are issued are randomly selected as well.

Figure 6.11 shows the network footprint for the baseline versus the ARES approach. It is quite evident that, after the initial transient, the slope of the ARES curve decreases significantly. In the end, the total footprint for the baseline system is nearly 3 times the ARES one, thus the gain is really significant in terms of total exchanged traffic. We stress that also in the ARES case the footprint increases with the number of service requests, since requests arrives from PoPs, whereas processing happens in data centres. This means that at least the transfer of input files, which cannot be cached, always happens.

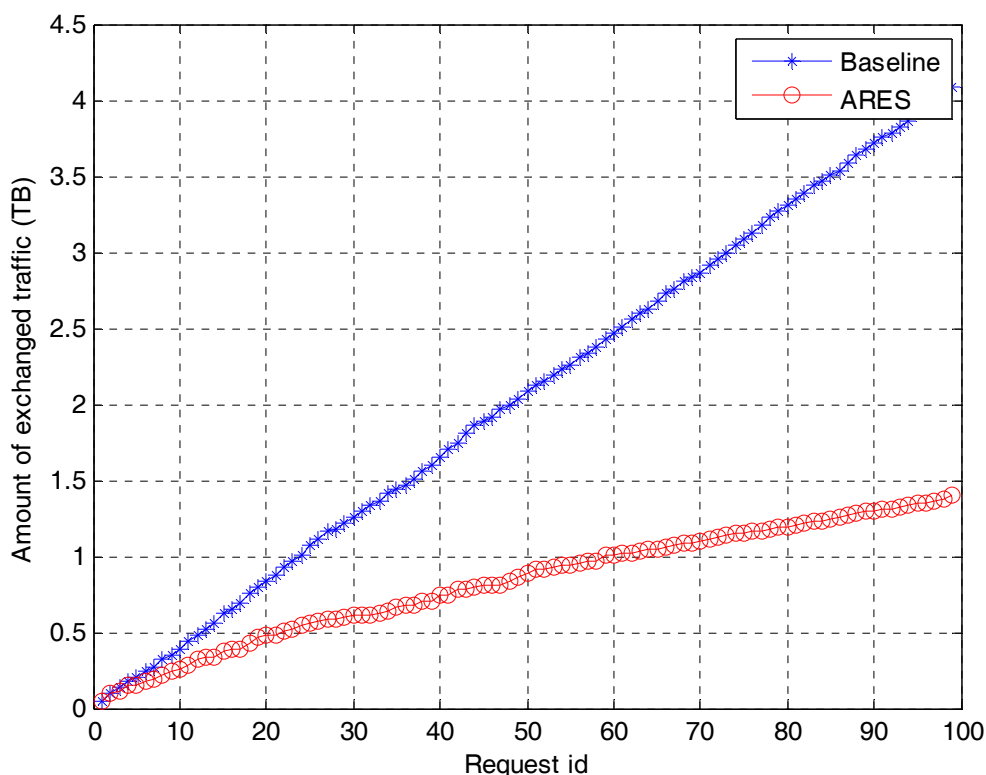


Figure 6.11: Network footprint in TB versus requests.

In order to gain a more detailed view of what happens during tests execution, in Figure 6.12 we show the gain of the ARES system in terms of traffic footprint (defined as mount of exchanged traffic in the baseline over the one in ARES), evaluated by means of a moving average using the latest 5 samples. This is necessary, in order to capture the correlation between service request coming from PoPs close each other and positive peaks in

footprint gain. In fact, when two close PoPs request the execution of the pipeline on different input files, if the GCM decides to allocate the processing in the same data centre or in close data centres, the network benefits of data already there, or being very close to it. This implies a strong advantage with respect to the baseline, where the manager tries just to co-locate service requests with processing, and auxiliary files and VM image files are always downloaded from the repository, which are located in IT (VM repository) and UK (auxiliary and reference file repository) PoPs. As we can see, these positive correlations generate peaks in the gain up to more than 7, which means that the traffic exchanged in the baseline computing model is more than 7 times the one in the ARES model. After the initial transient, the gain never goes below the value of 2, which implies that the ARES system at least saves 50% of traffic, which is really significant. This is due to the fact that the network has initiated to cache chunks around, and the amount of traffic to transfer files toward the selected PoP is no more than half that required by the baseline, including also not cacheable files. It is important to say that ARES is able to gain over the baseline in two different ways. First, it minimizes the traffic exchanged by selecting the data centre where hosting the processing after having found where contents, or chunks of them, are stored. Second, during the transfer, it caches content chunks in the nodes laying on the path between source and destination, including also the cache in the destination data centre. This distributes contents on the network as much as requests arrive. In addition, after half of the test, the gain never drops below a value of 3, which is an excellent result.

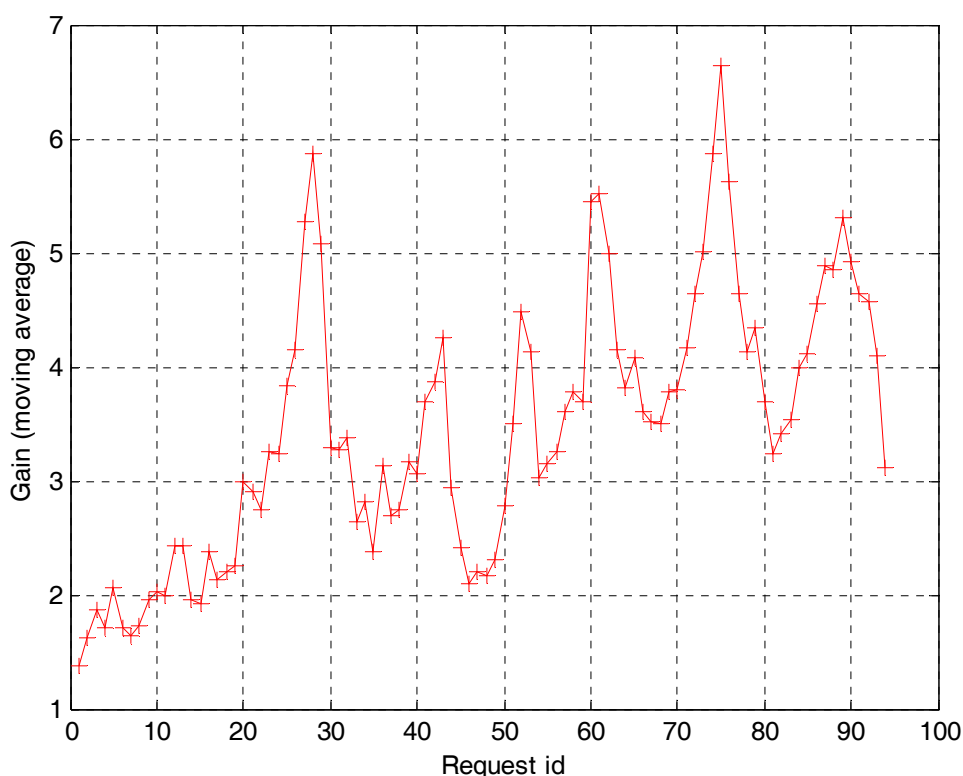


Figure 6.12: Gain (the ratio between baseline footprint and ARES footprint), evaluated by means of a moving average of the 5 latest samples.

Figure 6.13 shows the boxplot of the ratio between the network service time of a request and the processing time of a request. The boxplot is a convenient way to report variable information, since it provides quartiles and data samples which are considered outliers. As we can see, in more than 75% of cases the ARES system ensures that the network time (signaling time + optimization algorithm execution + download time + spawning time in emulated OpenStack) is less than 20% of the pipeline processing time. Thus, the network service offered by ARES is really satisfying, since the system bottleneck is on the cloud part, and not on the network part. In addition, this also means that the network occupation due to ARES file transfer is small not only in terms of amount of exchanged traffic, but also in terms of time. Thus, the ARES service can be ported into a real network, like Géant, with a minimal impact on the current provided services. Clearly, in order to execute this step, it is necessary to set a number of well provisioned data centres, in order to be able to timely execute genomic processing requests. In addition, it is evident that, even if also the baseline has a good behaviour, the time of the ARES system are more compact, which implies less variability and thus a more predictable service. In the case of heterogeneous service requests (section 6.2.2), this aspect will result more evident.

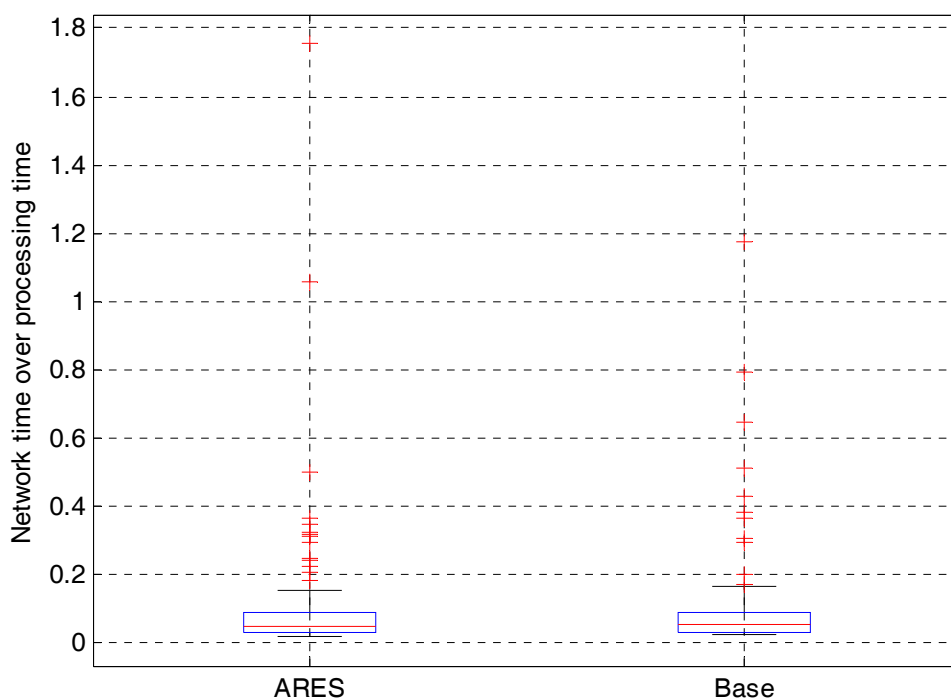


Figure 6.13: Box plot of the ratio between network time and processing time for both ARES and baseline schemes.

Figure 6.14 shows the transfer bandwidth per service request of the two systems, ARES and baseline. This is defined as the time needed to transfer all the needed data (VM image file, auxiliary/reference files, and input file) into the data centre selected for executing the processing. We used the box plot format, which is useful to

identify the range within which most of samples occur. The benefit introduced by ARES is evident: the average values for ARES is 320 Mb/s, whereas that for the baseline system is 88 Mb/s. Thus, not only ARES allows saving a lot of bandwidth by reducing the network footprint, but also definitely outperforms the baseline system during the transfer. These benefits are also due to the possibility of parallel downloading, which is possible in the ARES system due to the fact that each content chunk is downloaded separately, from the most convenient source.

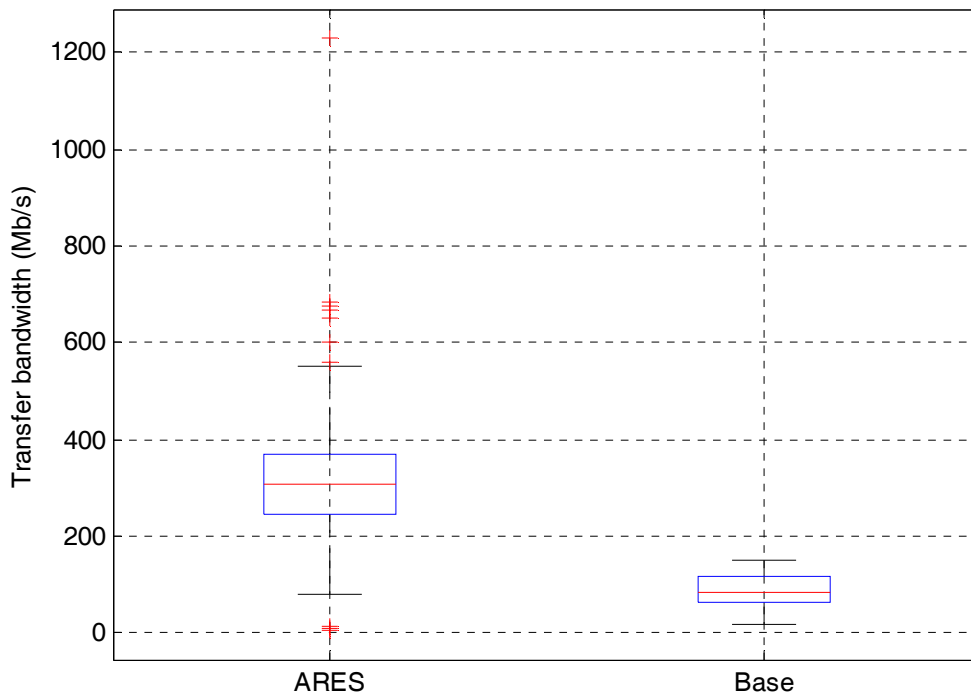


Figure 6.14: Box plot of the ratio between network time and processing time for both ARES and baseline schemes.

Finally, in Figure 6.15 we present a metric that is relevant only for ARES system, the path stretch. The path stretch is defined as the ratio between the path length (measured in IP hops) used to download cacheable files, and the length of the path that would result to download the same file from the repository, from the same location. Since the cache search procedure deployed in the ARES system ensures that the content will be always downloaded by the most convenient node, this metric is a number in the range $[0, 1]$. Depending on the location selected by the ARES decision logic for hosting the processing, in some cases the most convenient location can be also very close to the repository. In addition, in the initial phases, contents have to be downloaded necessarily from the repository. When the number of requests increases, caches are populated in the networks, and it is always less frequent to download the content from the repository. When approaching to the steady state condition, most of data centres already have all files necessary to perform computation. Figure

6.15 shows the estimated probability density function for this parameter. It is evident that the dominating value is 0, which implies that the selected data centre already has all needed files (VM image file + auxiliary/reference file) necessary for hosting the computation, and it has just to download the Input, which usually is the file representing patient genomes and clearly cannot be cached, and thus it is not considered for path stretch.

Since the baseline scheme does not use caching, the path stretch is, by definition, 1, and clearly it is not shown.

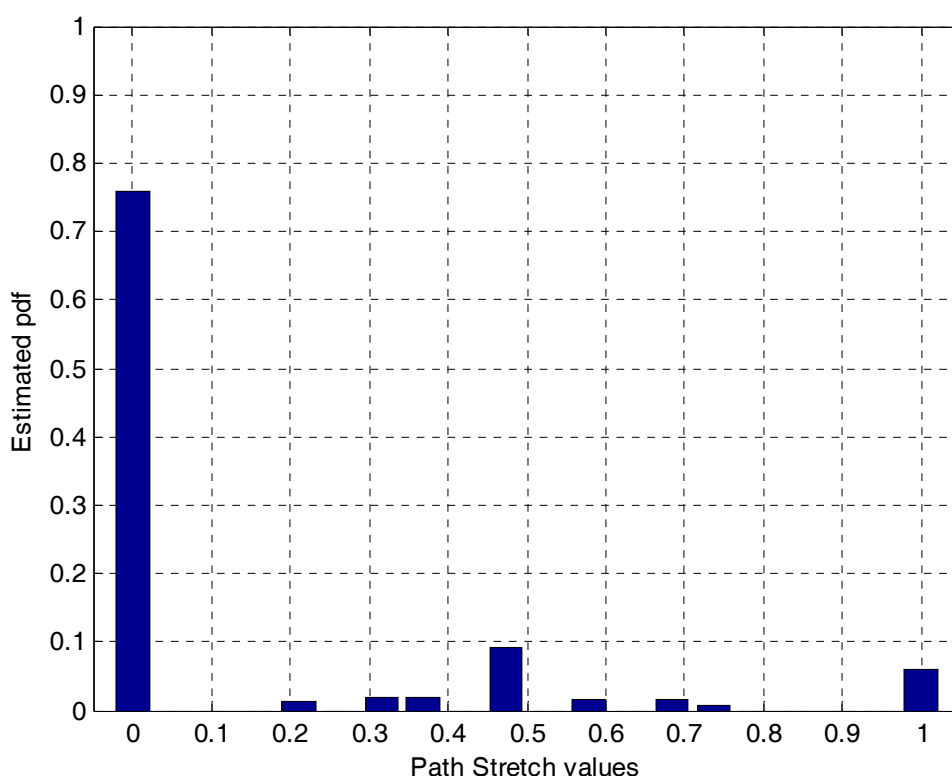


Figure 6.15: Estimated probability density function for the path stretch for ARES (for baseline it is always 1).

Finally, Table 6.30 reports the cost of signaling in terms of exchanged traffic normalized to the data traffic (genome files + auxiliary files + VM image files). As expected, the ARES logic requires more signaling traffic than the baseline one. In fact, NSIS is used only by ARES. In addition, since the traffic exchanged in the baseline system is much higher, it is clear that the comparison is a bit unfair. However, what emerges is that the overhead is largely balanced by the gain shown in the previous performance figures. In any case, the ARES overhead is really negligible with respect to the amount of exchanged traffic, thus it does not represent a system bottleneck, which thus scales very well by increasing network size.

System	HTTP signaling (GCM)	HTTP signaling (LOIB)	NSIS signaling (UDP)	NSIS signaling (TCP)
ARES	$1.24 \cdot 10^{-6}$	$4.38 \cdot 10^{-5}$	$2.15 \cdot 10^{-3}$	$4.30 \cdot 10^{-5}$
Baseline	$6.82 \cdot 10^{-7}$	$9.16 \cdot 10^{-7}$	-	-

Table 6.30: Normalized signaling overhead for ARES and baseline system.

6.2.2.2 Results for request with equal priority

The type of traffic handled in this scenario is much more challenging, due to the heterogeneity present not only in the node issuing the service request (already present in the other scenario), but also in the type of service request. The overall dataset is much larger than the amount of cache deployed in each node, which manages the eviction of contents according to LRU. This means that, in this scenario, cached chunks are replaced much more rapidly with respect to the other scenario, modeling high priority requests. Nevertheless, the gain due to the ARES system is still significant, and completely justifies the system deployment.

Figure 6.16 shows the total footprint for both ARES and the baseline system. This case is more challenging for the network footprint, due to the above mentioned heterogeneity in the request pattern, and the limited cache size with respect cacheable file sizes (see Table 6.29). Nevertheless, results are encouraging, since the total footprint for ARES is *half* that of the baseline system. This is due to the two level of optimization performed by ARES. The first one is that, once executed the signaling, the system knows where contents are stored, and can perform an optimization towards minimization of exchanged traffic. This is one of the reasons leading to part of the gain. The other reason is that, during file transfer, contents chunks are stored in caches, thus spreading these contents all over the network, thus potentially moving them towards multiples data centres. However, due to storage constraints in caches, what happens is that these contents cannot survive in caches for very long times, since if they are not requested again, they are evicted to leave space to new contents to be cached. Clearly, if a chunk in a cache is currently being downloaded from an upstream cache, or it is serving a downstream cache, it cannot be evicted. If all the content chunks in a cache are in these conditions, that is they are locked and cannot be removed, a content chunk "crossing" that node could be not stored in that cache, which is on the path between the source and the destination data centres. Clearly, this phenomenon is more evident when contents chunks are large. However, too small chunks are not recommended, in order to keep the signaling traffic at very low volume. In addition, this sort of hysteresis in removing contents increases the chance of a cache hit given the low value of the ratio chunk size over cache size, thus from our viewpoint is beneficial.

Figure 6.17 shows the gain the ARES system in terms of footprint with respect to the baseline system (defined as traffic for the baseline over traffic for ARES), evaluated by means of a moving average using the latest 5 samples as a function of the request number. Also in this service scenario performance is very good. Average values are more than doubled. Peaks, correlates with service requests having the same id (that is the same type of processing), or with different ids having some of the reference/auxiliary files in common. In these cases, it is possible to have a very significant gain, with gain in saved network traffic up to three times.

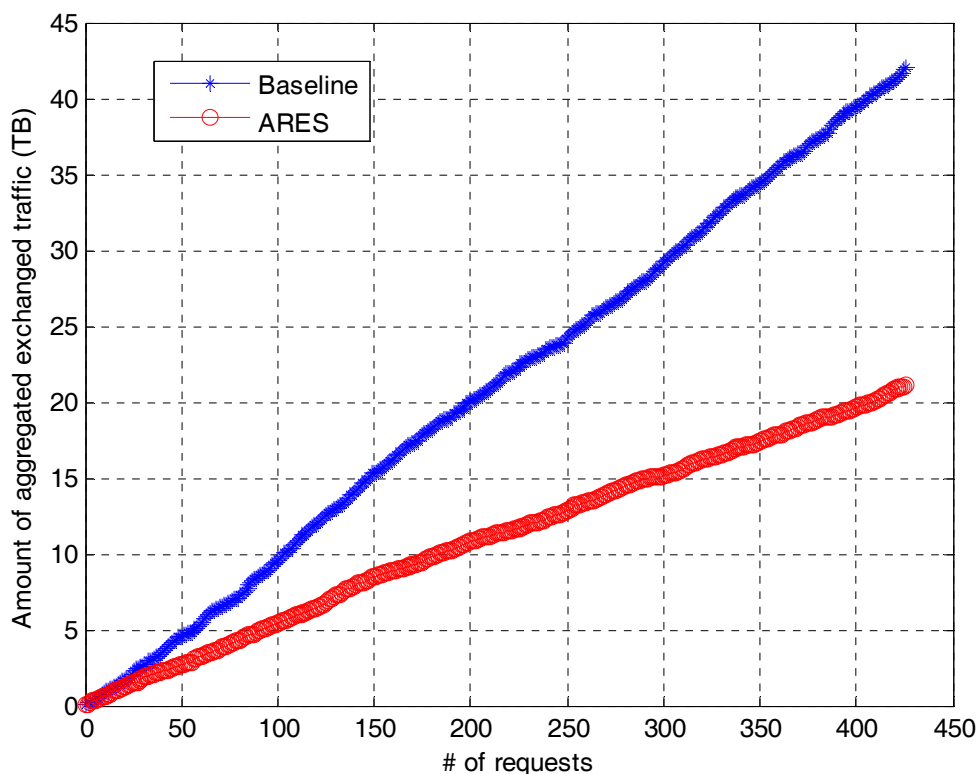


Figure 6.16: Network footprint in TB versus requests.

As for service time, also in this performance figure the ARES system provides an evident benefit, as shown by the box plot in Figure 6.18. The saving in network time is really significant, since in average using ARES allows shrinking the normalized network service time by 55%. In this case, by focusing on ARES performance, 75% of requests complete their data transfer phase within 50% of time necessary to the processing. In the baseline system, 75% of requests complete their data transfer phase within the same time devoted to the processing. This is a very important result, since the variability of service time is an undesired features. ARES allows limiting this phenomenon by pushing contents close to the data centres as a consequence of content caching. In this scenario, the impact of network service is in average higher, since the average size of files to be downloaded is higher (see Table 6.29) than the case with homogeneous requests. In turn, since there are also service requests involving small genome files (e.g. 1 GB), depending on the type of processing to be applied, the processing time can become comparable to the network service time, and sometimes also much small (see outliers in Figure 6.18).

In Figure 6.19, we report the box plot of the data throughput per service request. Also in this case, as for the case with high priority requests, the ARES approach outperforms the baseline one. We can note that absolute values are lower with respect to the case analyzed in section 6.2.2.1. This can be explained by the fact that, for the case with homogeneous requests, the size of cacheable files is fixed to 8.56 GB, plus a variable input size (non cacheable) with average value equal to 8 GB. As for the case analyzed in this section, given the

heterogeneity in service requests, the average size of cacheable files is equal to 20.97 GB, plus a variable input size (non cacheable) with average value equal to 8 GB. This implies that the network is much more congested due to the larger amount of traffic to transport, and thus the net data throughput decreases. Nevertheless, the ARES approach is able to guarantee an improvement of 33%. As for the outliers in the ARES approach, they represent the case in which most of (or all) the cacheable files associated to a single request are already stored in the cache of the data centre selected to host the processing. In this case, the data transfer happens at very high rates, and the only potential bottleneck is the upload of the genome file.

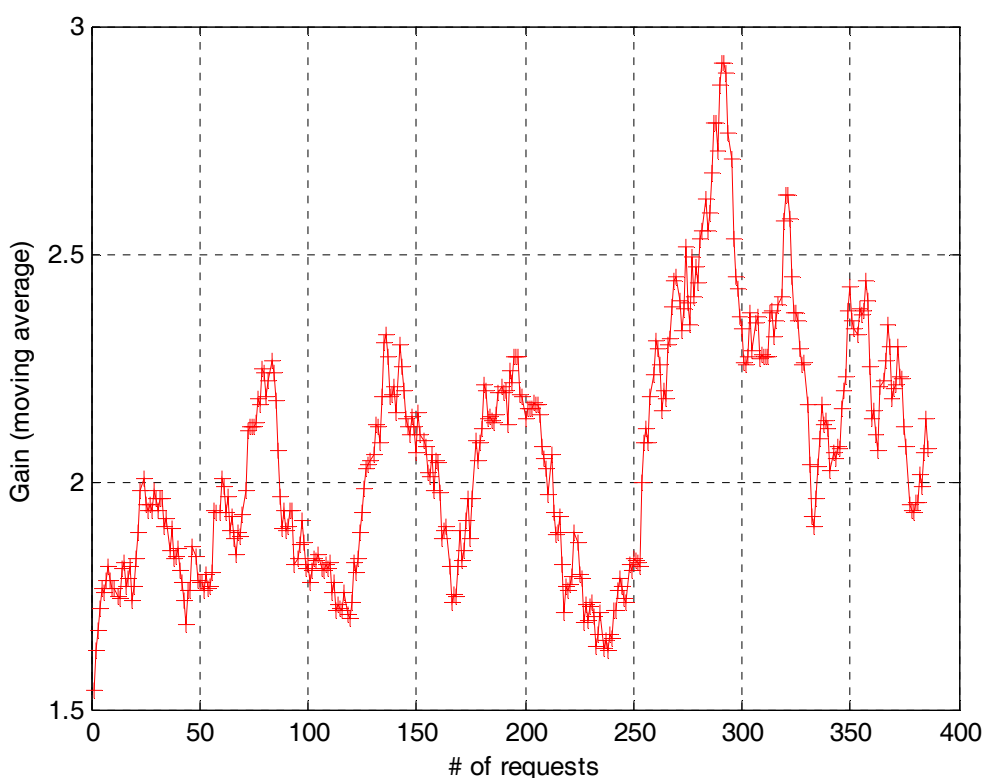


Figure 6.17: Gain (the ratio between baseline footprint and ARES footprint), evaluated by means of a moving average of the 5 latest samples.

Finally, the path stretch is shown in Figure 6.20. In this case, due to the heterogeneity of service requests, the probability density function is less biased towards the 0 value, even if this is the most frequent value (45% of times a chunk is found in the local cache). However, most of values are in the left side of the plot, with an average value equal to 0.24. Thus, not only the ARES logic is able to select the most convenient datacentre where running the processing, thus optimizing the consumption of network resources, but also the caching mechanism is effective. Clearly, increasing the cache size we would reach results in the order of those shown for the case of homogeneous service requests.

As for the impact of signaling on the overall traffic, also in this case it is negligible. In particular, given the fact that the average file size has increased (see Table 6.29), the normalized overhead has decreased as well. Thus, the scalability of the system is guaranteed.

System	HTTP signaling (GCM)	HTTP signaling (LOIB)	NSIS signaling (UDP)	NSIS signaling (TCP)
ARES	$2.63 \cdot 10^{-7}$	$7.55 \cdot 10^{-6}$	$6.41 \cdot 10^{-4}$	$6.20 \cdot 10^{-6}$
Baseline	$1.6 \cdot 10^{-7}$	$3.9 \cdot 10^{-7}$	-	-

Table 6.31: Normalized signaling overhead for ARES and baseline systems.

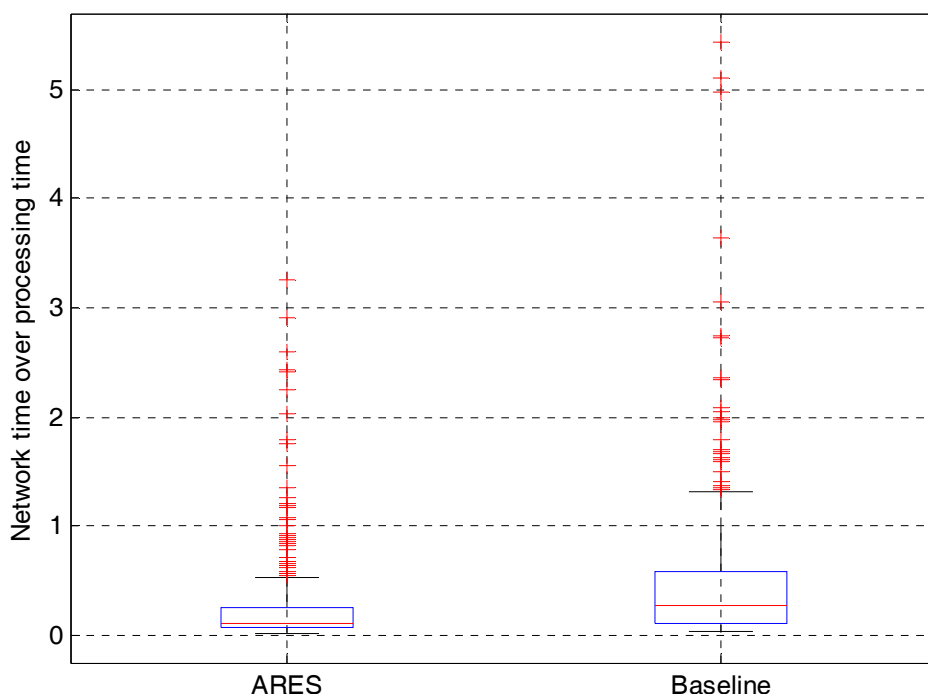


Figure 6.18: Box plot of the ratio between network time and processing time for both ARES and baseline schemes.

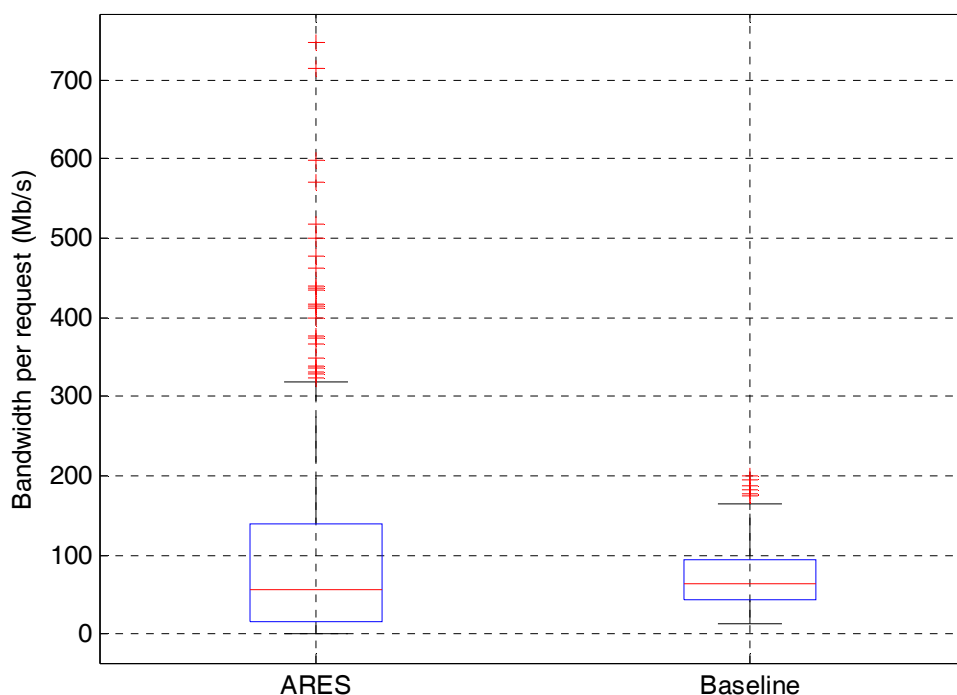


Figure 6.19: Box plot of the data throughput for both ARES and baseline schemes.

Finally, we have verified on this system that all targets for the metrics defined in section 6.1, that is access transparency, location transparency, availability, failure transparency, consistency, and scalability are successfully met. In particular, we have done a number of functional tests in which we have selectively shut down repositories, routers, links, and data centre nodes. As for GCM and repositories, we have previously duplicated them before shutting down the pre-defined one, in order to emulate access and location transparency. We also verified that upon system saturation (computing resources required by service requests exceed system capacity), content caches are consistent and the system simply rejects some service requests, without freezing the ongoing processing.

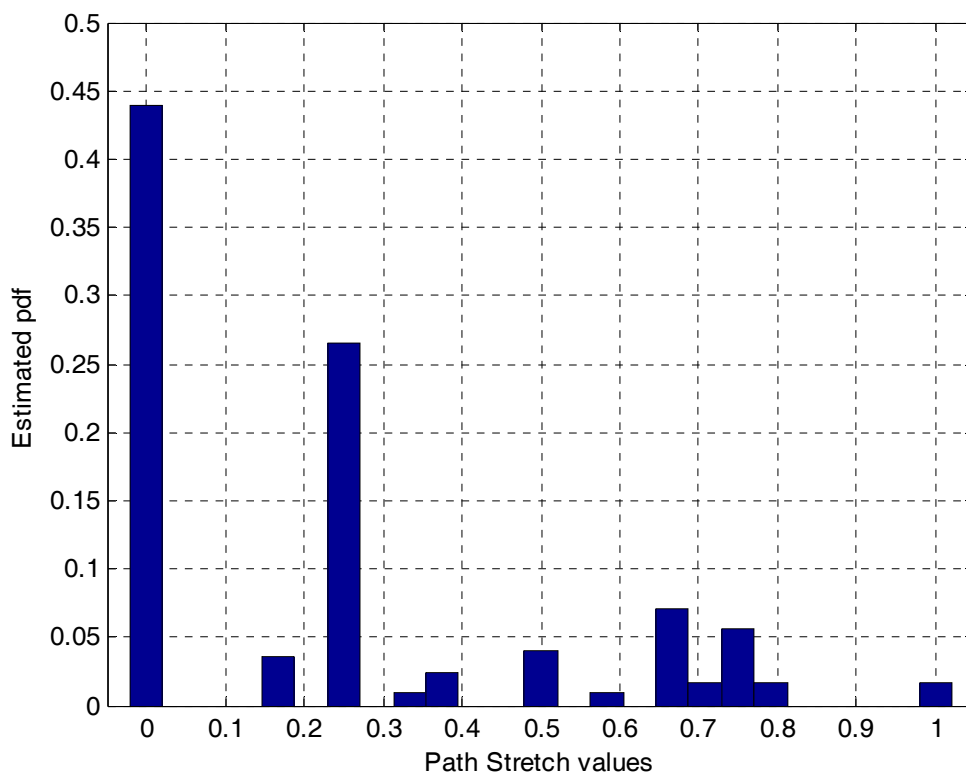


Figure 6.20: Estimated probability density function for the path stretch for ARES (for baseline it is always 1).

6.3 Chapter Conclusions

This chapter is a report of the ARES assessment plans and evaluation. These activities have been accomplished in accordance to expectations and project plans. The experimental results are in line with expectation in terms of scalability, service time, and signaling load.

7 Dissemination, Standardization, and Exploitation

7.1 Dissemination and standardisation

7.1.1 ARES Public dissemination

7.1.1.1 Research papers.

Table 7.1 reports a list of research papers that have been either published or accepted or just submitted, and waiting for the relevant editorial decisions, during the ARES project lifetime.

N o.	Title	Authors	Title of Periodical or Series	Publisher	Year	Permanent Identifiers ¹ (if available)	Is/Will Open Access ² Be Provided to This Publication?
1	The ARES Project: Network Architecture for Delivering and Processing Genomics Data	Mauro Femminella Gianluca Reali Dario Valocchi Emilia Nunzi	IEEE 3rd Symposium on Network Cloud Computing and Applications (NCCA) 2014.	IEEE	2014	ISBN 978-0-7695-5168-5, DOI: 10.1109/NC CA.2014.12	No
2	The ARES Project: Cloud Services for Medical Genomics	Mauro Femminella Gianluca Reali Dario	IEEE 3rd Symposium on Network Cloud Computing and Applications	IEEE	2014	ISBN 978-0-7695-5168-5, DOI: 10.1109/NC CA.2014.11	No

N o.	Title	Authors	Title of Periodical or Series	Publisher	Year	Permanent Identifiers ¹ (if available)	Is/Will Open Access ² Be Provided to This Publication?
		Valocchi Emilia Nunzi Valerio Napolioni Matteo Picciolini	(NCCA) 2014.				
3	ARES: Advanced Networking for Distributing Genomic Data	Mauro Femminella Gianluca Reali Dario Valocchi Emilia Nunzi	TERENA Networking Conference (TNC) 2014	Terena	2014	ISBN 978-0-7695-5168-5	Yes
4	A Resource Discovery Framework for Cloud-based Genomics Computing	Mauro Femminella Gianluca Reali Dario Valocchi Emilia Nunzi	IEEE CloudNet 2014	IEEE	2014	ISBN 978-1-4799-2730-2, DOI: 10.1109/CloudNet.2014.6968960	No
5	Networking issues related to delivering and processing genomic big data	Mauro Femminella Emilia Nunzi Gianluca Reali Dario Valocchi	International Journal of Parallel, Emergent and Distributed Systems	Taylor & Francis	2015	DOI: 10.1080/17445760.2014.929685	No
6	A signalling protocol for service function localization	Mauro Femminella Gianluca Reali Dario	Paper submission to IEEE Communications Magazine	IEEE	2015	Not yet	No

N o.	Title	Authors	Title of Periodical or Series	Publisher	Year	Permanent Identifiers ¹ (if available)	Is/Will Open Access ² Be Provided to This Publication?
		Valocchi Emilia Nunzi					
7	Networking for Genomic Computing	Mauro Femminella Gianluca Reali Dario Valocchi Emilia Nunzi Athanasios Vasilakos	Paper submission to Proceedings of the IEEE.	IEEE	2015	Not yet	No

Table 7.1: List of scientific (peer reviewed) publications

Notes:

1. A permanent identifier should be a persistent link to the published version (full text if open access or abstract if article is pay per view) or to the final manuscript accepted for publication (link to article in repository).
2. Open Access is defined as free of charge access for anyone via Internet. Please answer “yes” if the open access to the publication is already established and also if the embargo period for open access is not yet over but you intend to establish open access afterwards.

7.1.1.2 Other Dissemination Activities

Table 7.1 reports a comprehensive list of all dissemination activities in ARES, including events other than those related to research papers.

No .	Type of Activities ¹	Description	Name of Event	Date /Period	Place	Type of Audience ²	Countries Addressed
1	Workshop (Doctorate Seminar)	Processing and management of biometric human	Seminars at Doctorate in Genetics, Oncology	20 March 2014	Siena, Italy	Scientific Community (higher)	Italy (researchers from

No	Type of Activities ¹	Description	Name of Event	Date /Period	Place	Type of Audience ²	Countries Addressed
		data: EU supported research	and Clinical Medicine (GenOMeC)			education, Research)	Universities of Siena, Pisa, and Florence.
2	Workshop	Advanced Networking for the EU genomic research ARES	2nd TERENA Network Architects Workshop	13-14 November 2013	Prague, Czech Republic	Scientific Community (higher education, Research), Industry.	Géant Community
3	Paper publication & presentation	The ARES Project: Network Architecture for Delivering and Processing Genomics Data	IEEE NCCA 2014	5 February 2014	Rome, Italy	Scientific Community (higher education, Research).	Various Countries
4	Paper publication & presentation	The ARES Project: Cloud Services for Medical Genomics	IEEE NCCA 2014	5 February 2014	Rome, Italy	Scientific Community (higher education, Research).	Various Countries
5	Workshop	Ares presentation	1 st FOCUS GROUP on Technologie Sociali @ Confindustria Perugia http://www.confindustria.umbria.it	29 October 2013	Perugia, Italy	Industry, Policy makers	Italy
6	Interview	Interview to Gianluca Reali -	GN3 Symposium 2013 GARR TV Web Site - GARR TV	8-10 October 2013	Wien, Austria	Civil Society, Media	Italy
7	Web (ARES dissemination tools accessible via Internet, Web and Social	Project direct dissemination activity	General dissemination activity	October - December 2013	Perugia, Italy	Civil Society	Various Countries

No	Type of Activities ¹	Description	Name of Event	Date /Period	Place	Type of Audience ²	Countries Addressed
	Networks)						
8	Interview to Gianluca Reali by Diana Cresti.	Description of ARES (official title still to be released).	Article on GARR NEWS Magazine. ISSN 2039-8271.	10 April 2014	Perugia, Italy	Scientific Community (higher education, Research), Industry.	Italy
9	Workshop	ARES: Advanced Networking for Distributing Genomic Data ARES	ARES presentation at the Vrije Universiteit Brussel.	13 May 2014	Brussels Belgium	Scientific Community (higher education, Research).	Various Countries
10	Interview to Gianluca Reali by Tamsin Henderson	Title: ARES – A GÉANT Open Call project aims to address urgent genome data set challenges	http://www.geant.net/MediaCentreEvents/news/Pages/ARES.aspx	2014	On-line	Civil Society, Media	Various Countries
11	Workshop Presentation of Emilia Nunzi	Progetto ARES: Advanced networking for EU genomic REsearch	XXXI Congresso Nazionale Misure Elettriche ed Elettroniche, http://www.diism.univpm.it/gmee2014	Sept. 11-13, 2014.	Ancona, Italy	Scientific Community (higher education, Research).	Italy
12	Article on Connect Magazine. Issue #18	Title: Genomic Analysis in Géant: A Distributed Cloud Approach	http://www.geant.net/MediaCentreEvents/CONNECT/Pages/default.aspx	2015	On-line	Civil Society, Media	Various Countries
13	Standardization	Submission of a standardization document to the IETF Title: Off-path Signaling Protocol for Service Function Chaining	Contribution to Standardization. draft-femmreali-sfc-offpath-signaling-00 http://datatracker.ietf.org/doc/draft-femmreali-sfc-offpath-signaling/	March 2015	On-line	Scientific Community (higher education, Research).	Various Countries

No	Type of Activities ¹	Description	Name of Event	Date /Period	Place	Type of Audience ²	Countries Addressed
14	Paper publication & presentation	ARES: Advanced Networking for Distributing Genomic Data	TERENA Networking Conference (TNC) 2014	19 May 2014	Dublin, Ireland	Scientific Community (higher education, Research), Industry.	TERENA Community
15	Paper publication & presentation	A Resource Discovery Framework for Cloud-based Genomics Computing	IEEE CloudNet 2014	8 October 2014	Luxembourg	Scientific Community (higher education, Research).	Various Countries
16	Special session organization	Networks for Intensive Computing and Massive Storage Applications in Biology, Genetics, and Genomics	IEEE CloudNet 2014	8-10 October 2014	Luxembourg	Scientific Community (higher education, Research).	Various Countries
17	Paper publication	Networking issues related to delivering and processing genomic big data	International Journal of Parallel, Emergent and Distributed Systems	January 2015	On-line	Scientific Community (higher education, Research).	Various Countries
18	Paper submission	A signalling protocol for service function localization	Journal: IEEE Communications Magazine	March 2015	On-line	Scientific Community (higher education, Research).	Various Countries
19	Paper submission	Title: Networking for Genomic Computing	Journal: Proceedings of the IEEE.	March 2015	On-line	Scientific Community (higher education, Research).	Various Countries

Table 7.2: Comprehensive list of dissemination activities during the reporting period.

7.1.1.3 ARES Web Site

The main ARES Web site is provided by Géant, and includes sections providing a general description of the project, profiles of project partners, information on events involving ARES, on-line access to ARES documents, and contact information. Another Web page dedicated to the project, for achieving a larger dissemination, has been developed and available at: <http://conan.diei.unipg.it/lab/index.php/research/ares>. From this site, connections to the ARES contents distributed through social networks is available.

7.1.1.4 ARES poster

In the initial phase of the project, a poster of ARES has been prepared, under request, and made available to the GN3+ personnel for any future dissemination purpose.

7.1.2 ARES Standardization Activities

The ARES standardization activities consists of a preparation and submission of an IETF (Internet Engineering Task Force) Draft document (M. Femminella, G. Reali, D. Valocchi, draft-femmreali-sfc-offpath-signaling-00, March 23, 2015, <http://datatracker.ietf.org/doc/draft-femmreali-sfc-offpath-signaling/>) . This document includes off-path signaling protocols, as part of the IETF NSIS signaling architecture. In fact, these off-path signaling protocols play a central role in the deployment and operation of the ARES CDN. The relevant IETF working group is Service Function Chaining (SFC) WG.

This submitted standardization document describes the off-path signaling protocol (OSP), designed for distributing signaling messages over portions of data networks around data paths (configurable network scope). The main issue that is addressed by this document is the need of discovering network nodes, hosting virtualized service functions, within network scopes destined to deploy a wide class of services. The protocol is designed to be distributed, autonomic, and easy to deploy solution. This protocol leverages on two main network functions, namely, on-path packet interception and off-path signaling distribution.

7.2 Exploitation Plans

The exploitation plans of the ARES partners follows their working areas.

The University of Perugia intends to leverage on the cultural and technical ARES achievement in different directions. First of all, a continuous interaction with the Géant personnel, in particular with the one managing the Géant Testbed Service (GTS), has allowed the UoP team to prepare a virtualized distributed genomic computing environment ready to be host in the GTS. Since the latter is not still available for hosting external VMs with the needed resource allocation, this porting has been postponed by few months. As soon as it will be possible to proceed, regardless the ARES project will be formally finished, a porting and experimental activity will be executed with the aim of demonstrating the actual capability of increasing the Géant service portfolio. Having implemented all the ARES components with the primary target of being compliant with the Géant

network, we believe that this porting will be done in a very short time, through a simple customization of VMs in terms of access rights.

In addition, UoP will exploit the ARES achievement in two main research strands. First of all, since the achieved results are quite encouraging, further research activities, and relevant research projects, will be pursued. The aim of these projects will explore new networking potentials, in particular by making use of CCN/ICN services. This is a quite challenging research, since naming of genomic contents is not sufficiently standardized so as to explore the ICN/CCN potentials. Nevertheless, this is a research direction worthy of further exploration. The second research direction that will be explored is the support of genomic services for business activities different from medicine (for example, food production and tracking, management of industrial processes based on bacteria meta-genomics).

A further exploitation activity of UoP will be an enriched student training within a sector widely recognized as one of the most promising areas of the 21th century: data science and engineering.

GGB intends to use scientific results for increasing performance of the already available network with an ultra-wideband network in order to improve the service of bioinformatic analysis in terms of the delivery times of the product. In particular, the use of the Géant network, if available in the future, will allow an easier upgrade to a faster and efficient network infrastructure that will guarantee shorter service time due to the Géant network that already interconnects national research and education networks across Europe. This performance improvement will facilitate the extension of diagnostic services that GGB is about to deliver to hospitals in the local and neighbouring area, which are already connected by an existent network infrastructure that is different from Géant.

Moreover, research project currently ongoing at GGB involved in genome sequencing and that produce a massive amount of data, will consider results of the ARES project in order to improve performance of timing delivery of bioinformatic services between research groups in Europe. In particular, GGB will update the way computing is being done in order to exploit network services and to optimize management of network-based databases. Criteria derived within the ARES project for optimizing network timing delivery of bioinformatic processes aimed to diagnostic scopes could be extended and integrated to applications fields that require cloud based functions and network on-demand services. GGB is interested in introducing skills of network systems management in order to optimize existing network resources and improve the productivity of processes that makes intensive use of bioinformatic analyses to be shared between centres delocalized over large geographical areas.

8 Conclusions

This deliverable is a comprehensive description of the activities carried out in the framework of the ARES project. The essential aims of the project was to design, implement, and evaluate experimentally network solutions for exchanging genomic data sets necessary for executing genomic medical services. These objectives have been met through a research organizations, illustrated through a work-package organization and a temporal sequence of activities designed to optimize the efficiency of the research and implementation efforts.

The progress of the project activities has been checked against the production of the milestones by the dates scheduled during the preparation of the ARES proposal.

A significant feature of the project was the interdisciplinary expertise brought by the two collaborating partners, UoP and GGB. Whilst UoP has carried out all the research, implementation, and experimental activities related to the network solutions, GGB has contributed with the needed expertise for managing genomic data set. In particular, the networked genomic processing implemented in the project have been selected so as to represent a significant portions of the actual genomic services used for research and medical purposes. This contribution has been particularly useful in the perspective of designing network services since it has allowed to:

- Identifying the medical needs in terms of service time for different importance of the genomic analyses.
- Evaluating experimentally the processing time for different genomic processing pipelines in different operative configurations in a cloud environment, along with the processing resources (RAM size, number of computing cores) needed for their deployment.
- Identifying the actual network specification in the light of processing time and medical needs.
- Evaluating and making use of the most recent networking technologies for designing a solution compliant with the features of the Géant network with the objective of integrating the Géant service portfolio.
- Designing scalable solutions for managing genomic data in the network by making use of the original feature of genomic data sets and the network adaptation to the medical practices. In particular, the latter consists of using caching algorithms able to adapt to the flow of genomic data throughout the network.

For all these reasons we consider ARES a fruitful process that has paved the way for further initiatives in different directions.

- A specific agreement has been taken with the Géant personnel to port all the ARES components in the Géant Testbed Service, as soon as the configuration tools of the latter will allow allocating memory space and number of cores to virtual machines in order to execute genomic processing. We stress that under to functional point of view, these components have been developed for them to be compliant with the Géant virtualization environment.
- GGB intends to take advantage of the ARES results for enriching its networked genomic services.
- UoP intends to extend the scope of the ARES networked services towards different areas, including business ones, making use of genomes, such as meta-genomics, food production, and synthetic biology.

A particular comments is needed in relation of the lesson learned related to the networked deployment of genomic processing. Most of genomic software packages, among which a lot of them are open-source, have been implemented in the perspective of a private use in local computers. A very limiting factor that has been found in optimizing the ARES services was their differentiation for guaranteeing differentiated quality for handling situations more or less serious. For this reason, the most effective way of implementing differentiated genomic services was to suitably organize the input data to the genomic pipelines. Re-designing genomic software packages was out of the scope of ARES, and requires a major research effort significantly larger than the ARES one. For this reason, a further significant progress is envisaged in co-designing genomic distributed pipelines operating in geographically distributed computing clusters, organized by suitably designed network file systems. This is a research topic is indeed pursued in also for other applications making use of the mostly used file systems, such as the HFS, which do not scale well over geographical networks. Leveraging on the ARES achievements, this direction is promising of further significant benefits that will be the main object of future research projects.

In conclusion, the impact of the ARES results can be checked against the potential impact illustrated in the project proposal:

Research and dissemination: A major effort has been produced and appears for disseminating the research achievements. These activities include contributions to research journals and conferences, research magazine, general magazine, and generic dissemination events such as interviews available online.

Costs: The ARES framework has been developed in the perspective of commoditization of human genomes for medical purposes. The demonstrated scalability of the ARES solutions results in cost reductions of the used resources due to the reduces network footprint.

Diffusion of genomic services: Cost reduction implies higher affordability, and in the medium terms this means that the ARES solutions and the relevant further extensions can allow small medical centres accessing currently very expensive genomic services. This direction will be taken by partner GGB, that will take immediate advantage of the ARES achievements.

Standardization: The ARES activities have produced a proposal of standardizing a protocol for resource discovery. This proposal has been submitted to the IETF. UoP will be taking care of the process, which has a long duration.

Software availability: The ARES software components have been largely implemented by using open-source components. When these components will be deployed in the Géant Testbed Service they will represent a

usable baseline for increasing the Géant service portfolio and will strengthen the Géant position as leading European infrastructure to foster strategic network solutions.

European research support: The Géant vocation for supporting research in Europe can be strengthened by making the mentioned software components and services available EU-wide for research purposes.

Benefits for other applications beyond medicine: Genomic computing is the basis of a plethora of services, including industry (e.g. DNA engineering of bacteria used in industrial processes), meta-genomics (used, for example, for tracking goods), food production (development of genetically modified crops), and others. Data management shown some commonalities with the ARES framework, and by leveraging on the ARES achievements it is possible to increase the technical potentials also in other areas.

References

- [1] NetServ project home page, <http://www.cs.columbia.edu/irt/project/netserv/>. Visited on February 18, 2014.
- [2] X. Fu et al., "NSIS: a new extensible IP signaling protocol suite", *IEEE Communications Magazine*, 43(10), 2005, pp. 133- 141.
- [3] H. Schulzrinne, R. Hancock, "GIST: General Internet Signalling Transport", IETF RFC 5971, October 2010.
- [4] M. Femminella, R. Francescangeli, G. Reali, H. Schulzrinne, "Gossip-based signaling dissemination extension for next steps in signaling", *IEEE/IFIP NOMS 2012*, Maui, US, 2012.
- [5] OpenStack web site, <http://www.openstack.org/>, visited on February 18, 2014.
- [6] NSIS-ka, open source NSIS implementation by Karlsruhe University, available at: <https://projekte.tm.uka.de/trac/NSIS/wiki/>.
- [7] Atak ZK, Gianfelici V, Hulselmans G, De Keersmaecker K, Devasia AG, Geerdens E, Mentens N, Chiaretti S, Durinck K, Uyttebroeck A, Vandenberghe P, Wlodarska I, Cloos J, Foà R, Speleman F, Cools J, Aerts S. Comprehensive analysis of transcriptome variation uncovers known and novel driver events in T-cell acute lymphoblastic leukemia. *PLoS Genet*. 2013 Dec;9(12):e1003997.
- [8] Pflueger D, Sboner A, Storz M, Roth J, Compérat E, Bruder E, Rubin MA, Schraml P, Moch H. Identification of molecular tumor markers in renal cell carcinomas with TFE3 protein expression by RNA sequencing. *Neoplasia*. 2013 Nov;15(11):1231-40.
- [9] Ferreira PG, Jares P, Rico D, Gómez-López G, Martínez-Trillos A, Villamor N, Ecker S, González-Pérez A, Knowles DG, Monlong J, Johnson R, Quesada V, Djebali S, Papasaikas P, López-Guerra M, Colomer D, Royo C, Cazorla M, Pinyol M, Clot G, Aymerich M, Rozman M, Kulis M, Tamborero D, Gouin A, Blanc J, Gut M, Gut I, Puente XS, Pisano DG, Martin-Subero JI, López-Bigas N, López-Guillermo A, Valencia A, López-Otín C, Campo E, Guigó R. Transcriptome characterization by RNA sequencing identifies a major molecular and clinical subdivision in chronic lymphocytic leukemia. *Genome Res*. 2014 Feb;24(2):212-26.
- [10] Kalari KR, Necela BM, Tang X, Thompson KJ, Lau M, Eckel-Passow JE, Kachergus JM, Anderson SK, Sun Z, Baheti S, Carr JM, Baker TR, Barman P, Radisky DC, Joseph RW, McLaughlin SA, Chai HS, Camille S, Rossell D, Asmann YW, Thompson EA, Perez EA. An integrated model of the transcriptome of HER2-positive breast cancer. *PLoS One*. 2013 Nov 1;8(11):e79298.
- [11] Ni X, Zhuo M, Su Z, Duan J, Gao Y, Wang Z, Zong C, Bai H, Chapman AR, Zhao J, Xu L, An T, Ma Q, Wang Y, Wu M, Sun Y, Wang S, Li Z, Yang X, Yong J, Su XD, Lu Y, Bai F, Xie XS, Wang J. Reproducible copy number variation patterns among single circulating tumor cells of lung cancer patients. *Proc Natl Acad Sci U S A*. 2013 Dec 24;110(52):21083-8
- [12] Carss KJ, Hillman SC, Parthiban V, McMullan DJ, Maher ER, Kilby MD, Hurles ME. Exome sequencing improves genetic diagnosis of structural fetal abnormalities revealed by

ultrasound. *Hum Mol Genet.* 2014 Feb 11.

- [13] Eisenberger T, Neuhaus C, Khan AO, Decker C, Preising MN, Friedburg C, Bieg A, Gliem M, Charbellssa P, Holz FG, Baig SM, Hellenbroich Y, Galvez A, Platzer K, Wollnik B, Laddach N, Ghaffari SR, Rafati M, Botzenhart E, Tinschert S, Börger D, Bohring A, Schreml J, Körtge-Jung S, Schell-Apacik C, Bakur K, Al-Aama JY, Neuhann T, Herkenrath P, Nürnberg G, Nürnberg P, Davis JS, Gal A, Bergmann C, Lorenz B, Bolz HJ. Increasing the yield in targeted next-generation sequencing by implicating CNV analysis, non-coding exons and the overall variant load: the example of retinal dystrophies. *PLoS One.* 2013 Nov 12;8(11):e78496.
- [14] Jia P, Jin H, Meador CB, Xia J, Ohashi K, Liu L, Pirazzoli V, Dahlman KB, Politi K, Michor F, Zhao Z, Pao W. Next-generation sequencing of paired tyrosine kinase inhibitor-sensitive and -resistant EGFR mutant lung cancer cell lines identifies spectrum of DNA changes associated with drug resistance. *Genome Res.* 2013 Sep;23(9):1434-45
- [15] Talkowski ME, Maussion G, Crapper L, Rosenfeld JA, Blumenthal I, Hanscom C, Chiang C, Lindgren A, Pereira S, Ruderfer D, Diallo AB, Lopez JP, Turecki G, Chen ES, Gigek C, Harris DJ, Lip V, An Y, Biagioli M, Macdonald ME, Lin M, Haggarty SJ, Sklar P, Purcell S, Kellis M, Schwartz S, Shaffer LG, Natowicz MR, Shen Y, Morton CC, Gusella JF, Ernst C. Disruption of a large intergenic noncoding RNA in subjects with neurodevelopmental disabilities. *Am J Hum Genet.* 2012 Dec 7;91(6):1128-34.
- [16] DM. Mount, *Bioinformatics: Sequence and Genome Analysis* (2nd ed.), Cold Spring Harbor Laboratory Press: Cold Spring Harbor, NY, 2004, ISBN 0-87969-608-7.
- [17] Langmead B, Salzberg S. Fast gapped-read alignment with Bowtie 2. *Nature Methods.* 2012, 9:357-359.
- [18] A. Dobin et al, "STAR: ultrafast universal RNA-seq aligner", *Bioinformatics* 2012; doi: 10.1093/bioinformatics/bts635.
- [19] N. Spring et al., "Measuring ISP topologies with rocketfuel," *Networking, IEEE/ACM Transactions on*, vol. 12, no. 1, pp. 2–16, 2004.
- [20] M. Yandell and D. Ence, "A beginner's guide to eukaryotic genome annotation", *Nature Reviews, Genetics*, vol. 13, May 2012.
- [21] Hancock et al., "Next Steps in Signaling (NSIS): Framework," IETF, RFC 4080, Jun. 2005.
- [22] D. Katz, "IP Router Alert Option," RFC 2113, Internet Engineering TaskForce, Feb. 1997, updated by RFCs 5350, 6398.
- [23] B. Haeupler et al., "Discovery through gossip," in *ACM SPAA '12*, 2012, pp. 140–149.
- [24] M. Jelasity, A. Montresor, and O. Babaoglu, "T-man: Gossip-based fast overlay topology construction," *Comput. Netw.*, vol. 53, no. 13, pp. 2321–2339, Aug. 2009.
- [25] Ntp pool project. [Online]. Available: <http://www.pool.ntp.org/>
- [26] T. H. Cormen et al., *Introduction to Algorithms*, 2nd ed. McGraw-Hill Higher Education, 2001
- [27] A.-L. Barabasi and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.
- [28] L. Gargano et al., "Efficient communication in unknown networks," *Networks*, vol. 38, no. 1, pp. 39–49, 2001.
- [29] G. Reali et al. IM2.2 Milestone report "Milestone IM2.2: Report on scenarios, requirements and system architecture ", ARES report.
- [30] The GÉANT pan-European research and education network , <http://www.geant.net>.
- [31] E. Maccherani et al., "Extending the NetServ Autonomic Management Capabilities using OpenFlow", *IEEE/IFIP NOMS 2012*, Maui, US, 2012.
- [32] Eric E. Schadt, Michael D. Linderman, Jon Sorenson, Lawrence Lee, Garry P. Nolan, "Computational solutions to large-scale data management and analysis", *Nature Reviews*

Genetics, vol. 11, September 2010.

- [33] Shruti Sanadhya, Raghupathy Sivakumar, Kyu-Han Kim, Paul T. Congdon, Sriram Lakshmanan, Jatinder Pal Singh, "Asymmetric Caching: Improved Network Deduplication for Mobile Devices", Mobicom 2012, Istanbul, Turkey.
- [34] M. Hefeeda, O. Saleh, "Traffic modeling and proportional partial caching for Peer-to-Peer systems", IEEE/ACM Transactions on Networking, 16 (2008), pp. 1447–1460.
- [35] Konstantinos Katsaros, George Xylomenos, George C. Polyzos, "MultiCache: An overlay architecture for information-centric networking", Computer Networks, Volume 55, Issue 4, 10 March 2011, Pages 936-947.
- [36] Dario Rossi, Giuseppe Rossini, "Caching performance of content centric networks under multi-path routing (and more)", Technical report, Telecom ParisTech, 2011, <http://perso.telecom-paristech.fr/~drossi/paper/rossi11ccn-techrep1.pdf>.

Glossary

AAB	ARES Application Bundle
AR	Access Router
AREQ	Application REquirement
ARES	Advanced networking for EU genomic REsearch
AS	Autonomous System
BB	Backbone
CB	Cache Bundle
CDN	Content Distribution (or Delivery) Network
CNV	Copy Number Variation
CPU	Central Processing Unit
DB	Data Base
DE	Differential Expression
DNA	DeoxyriboNucleic Acid
EU	European Union
FT	Full testbed
GB	Giga Byte
GCM	Genomic CDN Manager
GGB	Polo di Innovazione sulle Genomica, Genetica e Biologia.
GIST	General Internet Signaling Transport (protocol)
GPVM	Genome processing VM
GREQ	General REquirement
GTS	Géant Testbed Service
GUM	GUide to the expression of uncertainty in Measurement
HTTP	HyperText Transfer Protocol
IANA	Internet Assigned Numbers Authority
laaS	Infrastructure-as-a-Service
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
ID	IDentifier
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
LP	Leaf Peer
LOIB	Local OpenStack Interface Bundle

LTS	Large Scale Testbed
MA	Message Association
MC	Magagement Commettee
MRI	Message Routing Information
MRM	Message Routing Method
NCCA	Network Cloud Computing and Applications
NCM	NetServ Control Message
NFV	Network Functions Virtualization
ni	network interface
NI	Network Initiator
NLI	Network Layer Information
NREN	National Research and Education Network
NSIS	Next Step in Signaling
NSLP	NSIS Signaling Layer Protocol
NTLP	NSIS Transport Layer Protocol
OH	One Hop
OSGi	Open Service Gateway initiative
OSP	Off-path Signaling Protocol
PaT	Path Table
PeT	Peer Table
PI	Peer Identity
PoP	Point of Presence
PTG	Peer to Gossip
PTS	Peer to Share
QoE	Quality of Experience
QoS	Quality of Service
RAM	Random Access Memory
RAO	Router Alert Option
RNA	Ribonucleic acid
SaaS	Software-as-a-Service
SAS	Serial Attached SCSI
SCSI	Small Computer System Interface
SREQ	Signalling REquirement
TCP	Transmission Control Protocol
TNC	Terena Networking Conference
TTL	Time-to-Live
UB	Upper Bound
UDP	User Datagram Protocol
ui	user interface
UoP	University of Perugia
USB	Universal Serial Bus
VM	Virtual Machine
WP	Work Package