A Resource Discovery Framework for Cloud-based Genomics Computing

Mauro Femminella, Gianluca Reali, Dario Valocchi
Department of Engineering
University of Perugia
Perugia, Italy
mauro.femminella@unipg.it, gianluca.reali@unipg.it, dario.valocchi@gmail.com

Emilia Nunzi
Department of Experimental Medicine
University of Perugia
Perugia, Italy
emilia.nunzi@unipg.it

Abstract—**In recent years scientific computing has evolved into a massive usage of cloud computing, due to its flexibility in managing computing resources. In this paper, we focus on genomic data processing, which is rapidly gaining momentum in research and medical activities. The main characteristics of these data sets is that not only the number of available genome files is becoming extremely large, but also each individual data set is significantly large, in the order of tens of GB. Hence, a wide diffusion of cloud-based genomic data processing will have a significant impact on network resources, since each processing request will require the transfer of tens of GBs into computing nodes. To face this issue, in this paper we propose a resource discovery framework which provides decision agents with the needed information for selecting the most suitable computing nodes. We have implemented this resource discovery function in a distributed fashion, and extensively tested it in a lab testbed consisting of about 70 nodes. We found that the overhead of the proposed solution is negligible in comparison with the amount of transferred data.**

Keywords—cloud computing; genomics processing; distributed framework; resource discovery protocols; off-path signaling; NSIS

I. Introduction

In recent years, the implementation of scientific computing platforms have evolved from the use local cluster-based computing, to the use of distributed grid computing and, more recently, of cloud computing infrastructure [13]. This evolution is due to the improved flexibility in managing computing resources that this latest paradigm offers over the other ones. However, scientific computing is very different from typical cloud-based applications, ranging from hosting multimedia servers to deploying storage facilities. The main differences include the volume of data managed by scientific applications and/or CPU requirements, which could be of some orders of magnitude larger than in other types of applications. In addition, scientific computing applications are highly heterogeneous. For example, they may consist of platforms used to storage and process the output of high energy physics experiments, which need to be co-located with experimental facilities in order to timely collect the huge amount of data, or climate changes simulations, which needs high performance computing architectures, or genomic data sets processing, which may require more modest amounts of computing resources for a single execution, with a highly increasing number of executions to be implemented, with a size of the input files easily reaching tens of GBs.

This work is supported by EU under the project ARES, accepted in the first open call of the GN3plus project (grant EC/FP7/605243).

In this paper, we focus on genomic data processing, which is rapidly gain momentum in research and medical activities, due to the reduction in the DNA sequencing costs [1]. The main characteristics of these data sets is that not only the number of available genome files is becoming extremely large, but also each data set is significantly large, in the order of tens of GB. This problem is referred to as Big^2 data problem, and its importance increases over time. In fact, it is expected that in the next few years all newborns will be sequenced and the medical science will build upon genome-processing outcomes. Clearly, it is not realistic to assume that each hospital will be able to acquire a large computing facility (private cloud) to cope with the internal processing demand. Thus, the use of public cloud-based processing services will be the obvious solution [11]. Hence, it is evident that the suitable management of genomic data will be essential not only for storage services, but also for their processing and their transfer. In fact, a wide diffusion of cloud-based genomic data processing will have a significant impact on network resources, since each processing request will require the transfer of many GBs of data into computing nodes in data centers of cloud providers.

In this paper we propose a discovery protocol of cloud resources [22][29], by providing decision agents with a number of context information. This information may consist of position and availability of processing resources (CPU, RAM, storage), input data to be processed, auxiliary files (e.g. annotations [12]), and image files of the virtual machines (VMs) hosting the genomic software packages used to process genomics data. In fact, the types of genomic processing (implemented through the combination of different programs, referred to as *genomic pipelines*) that can be executed over a single genome is quite large; just to have an idea, the interested reader can find a non-exhaustive list reported in Table 1 of [12]. Thus, it is not realistic to expect that each datacenter of a cloud provider can simultaneously host all VMs needed for any possible request.

Our discovery protocol leverages the functions offered by the Next Steps in Signaling (NSIS) framework [2][3]. In more detail, we use the functions offered by the recently defined off-path extension to the NSIS protocol suite, initially presented in [8] and further refined in [9]. This solution allows disseminating signaling over network areas of nearly arbitrary shape. By leveraging the interception capabilities of the signaling transport layer of NSIS, this signaling dissemination protocol is highly efficient, and may allow finding *resources* which are *close* to a given network path. In the considered scenario, the path under consideration could be, for instance, the one connecting the repository storing the needed VM images and the data center storing the input data. Since computing clusters in datacenters able to host VMs *cannot* be on the IP path connecting two servers (only routers are located on path), a signaling protocol with off-path discovery capabilities is used to discover data centers with both sufficient computing capabilities and a position suitable in order to minimize the overall network traffic exchanged. When this information is reported back to a decision agent, the latter can execute an optimization algorithm for selecting a datacenter.

Our solution can be classified as a peer-to-peer (P2P) resource discovery solution [25], with the feature that, being based on NSIS, it is able to couple with specific network paths. Classic P2P-based approaches [28][23] offer little or no support to provide proximity-based solutions, as stated in [26], trying just to limit the network overhead. Hierarchical solutions enable the efficient discovery of the resources belonging to one super-peer, but when more than one super-peers are involved, the problem is not trivial. Other solutions, recently proposed, make use of a proper ontology for service discovery [18][19][20], without considering path proximity issues. In [24], the authors propose an abstraction layer to discover the most appropriate infrastructure resources, which is then used in constraint-based approach applied to a multi-provider cloud environment. However, proximity issues are loosely handled, by simply referring to location requirements. Other solutions have been specifically designed for grid computing platforms, such as [21][27]. In [21], routing hops are considered, but they are relevant to the grid overlay. In [27], again the effort is to minimize the amount of message on the grid overlay, and not to find nodes close to specific locations or paths.

We have implemented the proposed discovery solution in a lab testbed consisting of about 70 nodes, and have executed extensive tests. The obtained results prove that the network overhead of the proposed solution is negligible when compared with the size of data files to be exchanged.

The paper is organized as follows. Section II presents some background on NSIS, and the reference scenario, which is represented by the project ARES [10]. Section III provides algorithmic and protocols details of our solution, and Section IV presents the results of lab experiments. Finally, we draw our concluding remarks in Section V.

II. Background and Reference Scenario

A. Background on NSIS protocols and off-path extension

The NSIS protocol suite divides the signaling functions into two layers [3]. The upper layer, called NSIS Signaling Layer Protocol (NSLP), implements the application signaling logic. The lower layer, called NSIS Transport Layer Protocol (NTLP), delivers the NSLP messages to the next NSIS node on path towards a given destination. The General Internet Signaling Transport (GIST) is the IETF-defined version of the NTLP [5]. GIST transport services make use of existing protocols in the TCP/IP stack. 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 session, GIST peers have to create a Message Association (MA) by using the information transported in the GIST packets, such as the unique identifier of the GIST node (Peer Identity) and one of its IP addresses.

Although IETF has standardized only two NSLP protocols (Quality-of-Service signaling and NAT/firewall traversal), others have been implemented , such as the NetServ NSLP [7].

NSIS has been primarily designed for managing states over 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 message routing methods (MRMs). Two MRMs are currently specified in GIST RFCs: (i) Path-Coupled MRM, which routes signaling messages through the data path, and (ii) Loose End MRM, used for preconditioning states in middleboxes when data destinations lie behind them.

GIST messages include a Message Routing Information (MRI) object, which 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.

The modular architecture of NSIS and GIST allows extending them, also defining new MRMs, as specified in [4]. In [8][9] we have extended the signaling capabilities of NSIS with off-path features by defining a new MRM. This extension requires the definition of a GIST peer discovery protocol for making each GIST node aware of its GIST peers. By using the information collected by this protocol, we defined new delivery strategies of signaling messages, which exploit the signaling interception capabilities of GIST.

In this work, we use a specific off-path delivery strategy defined in [8][9], the so-called hose mode. It consists of delivering NSIS messages to all NSIS nodes that are at a distance, expressed in IP hops, lower than or equal to the hose radius from each node of the data path connecting the sender with the signaling destination. In practice, each NSIS node on the data path is responsible of disseminating signaling to nodes at a distance no larger than the radius (bubble mode). Hence, the hose consists of a sequence of adjacent bubbles. As for the algorithm used to implement the proposed functions, in this paper we consider the GIST discovery protocol named Leaf, which provides stability and low overhead, whereas the signaling dissemination is carried out with the GIST Flooding strategy. The interested reader can find more details in [9].

B. Reference scenario: the ARES project

The reference scenario of this paper is the research project

ARES (Advanced networking for EU genomic RESearch [10]), which aims to implement cloud-based processing facilities in the points of presence (PoPs) of the Géant network [17]. Thus, the ARES framework can be regarded as an application delivery network (ADN, [14]), and the Géant network as a cloud provider with multiple datacenters.

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 [16]. In this virtualized infrastructure, we also assume that a specific node of the computing cluster in the PoP executes a VM that hosts the NetServ environment [7]. This solution has the advantage of being plug&play, since adding just a VM is straightforward and do not require any change in the management of the computing infrastructure. In addition, it allows deploying the same service architecture also with different cloud management systems, since it is necessary to adapt just a NetServ service.

NetServ is a solution for in-network service modularization and virtualization. The node architecture is suitable for any type of nodes, such as routers, servers, set-top boxes, and user equipment. Its architecture is targeted for in-network virtualized service containers and a common execution environment for network services. In ARES, NetServ services are used as content caches [15], to limit the overhead of transferring large genomic related files, and to implement service management functions, to interact with the OpenStack framework. NetServ caching service can be also hosted in routers (an effort to port NetServ into Juniper routers is currently in progress). The control plane of NetServ uses the NSIS signaling to coordinate service execution and to hot deploy service modules in surrounding NetServ nodes.

III. Discovery Framework

The proposed discovery framework is based on the off-path extension of NSIS, and specifically of GIST. In more detail, we designed an NSLP with at least one type of message, named Query, and the related answer (QueryResponse). It is used to collect binary information (*true* or *false*) on the status of the recipients, which are all the NSIS enabled nodes in the hose surrounding the IP paths between the sender and the destination. Clearly, this behavior can be extended to enable more sophisticated types of queries. The structure of these messages is schematized in Fig. 1. The payload of the QueryResponse is returned to the local application that triggered the signaling.

Query	=	NSLP-header [Application-requirement list] [Content-Id list]
QueryResponse	=	NSLP-header ResponseWrapper stack
ResponseWrapper	=	common-header Node-Id Depth Response code

Fig. 1 – NSLP packets format.

From Fig. 1, it is clear that this type of Query can be used for searching both nodes storing a desired content (caches or original repositories), and datacenters suitable for executing computations, thus satisfying specific, minimum requirements.

Fig. 2 shows the inter process communication between the involved framework entities of a forwarder node, when it receives an Query message, routed through an hose by GIST. The involved entities are the NTLP (GIST), the newly designed NSLP, and the application managing the data.

After the NTLP handshake, described in [5], the NSLP data are delivered to GIST, which provides to send them to the NSLP. The NSLP identifies the message type, extracts the application payload and passes it to the Application instance (steps 1-4 in Fig. 2). The Application processes the query and returns the response code to the NSLP, which provides to store it (steps 5-7 in Fig. 2). Meanwhile, the NTLP is in charge of forwarding the Query message through the hose. It identifies the next-hop destinations and starts a GIST hand-shake with them. Each hand-shake triggers the delivery of a MessageStatus from the NTLP to the NSLP [5]. 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 Fig. 2).

During the GIST handshake, if a next-hop destination has received the same App-Query message from another node, it will answer the GIST query message with a GIST off-path-duplicate error, thus aborting the handshake [9]. This error is reported to the NSLP using another MessageStatus, which triggers the decrease of the response counter (steps 11-13 in Fig. 2). If the NTLP starts *m* GIST hand-shakes and receives *k* GIST off-path-duplicate errors, the NSLP will wait for *m-k* responses before forwarding its response upstream (Fig. 2, 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 Fig. 2). 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 (Fig. 2, steps 20-22), i.e. to the GIST peer from which it initially received the Query.

During the responses collection, the NSLP is also in charge of maintaining the Depth field of each ResponseWrapper object it receives, depicted in Fig. 1. This object, along with the Node-Id object and the RespondeCode, allows the signaling initiator to compute a data structure which contains both the application information and the hose topology. During the steps 7 in Fig. 2, the NSLP initializes its ResponseWrapper with data provided by the local Application (step 6 in Fig. 2), its own Node-Id, and a Depth field containing the value 0. During the step 19 of Fig. 2, the Depth field of all the received wrappers is increased by 1, before adding it to the local stack. Fig. 3 shows the stacks contained in each Response message exchanged when a Query is sent through an 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

Fig. 3, the router R1 can infer the hose structure which is depicted with explicit indentation. In this figure, the label DC*i* indicates a datacenter (thus an off-path node), whereas a label R*j* indicates a router.

With reference to the scenario of the ARES project, described in section II.B, we consider a quite refined search process, consisting of three steps. The first two steps use the hose signaling illustrated above.

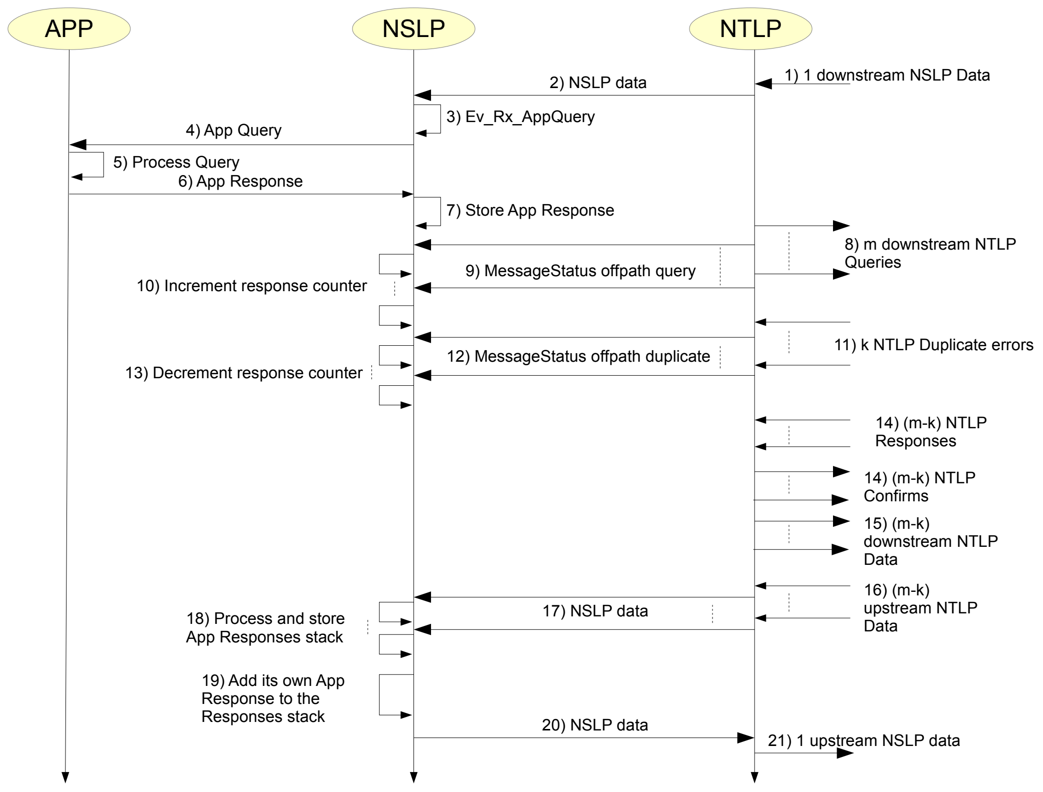
In the first step, the ARES decision agent, known as GCM (genomic CDN manager [10]), triggers a hose signaling, on the VM repository application, towards the node storing the genomic data sets to be processed. In addition, a further signaling is triggered also on the repository storing the auxiliary files, needed for pipeline computation [10], towards both VM repository and genomics data sets storage. This first set of hose signaling is used to identify the potential datacenters close to any of the three involved paths, and satisfying the minimum requirements indicated in the payload of the Query message (see the *Application-requirement list* field in Fig. 1). When the application which has sent the signaling obtains the QueryResponse messages, it forwards it to the GCM, which can thus know all the identities of the datacenters 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 *Content-Id list* field of the Query message (see again Fig. 1). Again, when the application which has sent the signaling obtains the QueryResponse messages, 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 datacenter controller (local OpenStack interface bundle, LOIB, in the ARES architecture, see [10]), 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 III.A of [9] for details). 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, 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 datacenter for executing the needed processing.

IV. Performance Evaluation

In this section, we present the overhead of a single hose Query, measured on a topology of 71 real nodes. We used 71 virtual machines running the Ubuntu 12.04 operating system.



APP
NSLP
NTLP
1) 1 downstream NSLP Data
2) NSLP data
3) Ev_Rx_AppQuery
4) App Query
5) Process Query
6) App Response
7) Store App Response
8) m downstream NTLP Queries
9) MessageStatus offpath query
10) Increment response counter
11) k NTLP Duplicate errors
12) MessageStatus offpath duplicate
13) Decrement response counter
14) (m-k) NTLP Responses
14) (m-k) NTLP Confirms
15) (m-k) downstream NTLP Data
16) (m-k) upstream NTLP Data
17) NSLP data
18) Process and store App Responses stack
19) Add its own App Response to the Responses stack
20) NSLP data
21) 1 upstream NSLP data

Fig. 2 – Sequence diagram describing the interactions between the application, the NSLP, and the NTLP inside a node.

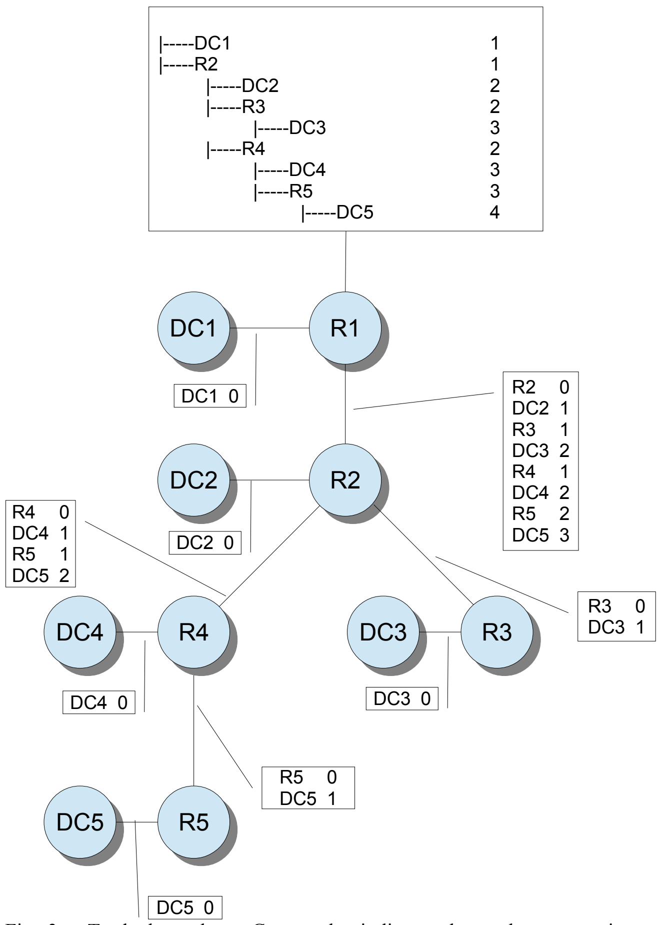


Fig. 3 – Testbed topology. Grey nodes indicates that a data center is co-located with the PoP at 1 IP hop. Each PoP is realized with one router, NSIS enabled.

Fig. 4 shows the testbed topology. The network reflects the PoP level topology of the Géant network [17]. The network is composed of 41 PoPs, represented as a single router, and 30 datacenters. Each grey node in the figure is connected with a single datacenter (not shown in the figure to improve its readability), laying 1 IP hop away from it. Each node runs NSIS-ka [6], along with our off-path extension, and NetServ. We used the NetServ NSLP Probe/ProbeResponse messages to evaluate the overhead of the Query/QueryResponse signaling, since, basically, they use the same logic (see [7]). 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 hose, the radius of which ranges from 1 to 3 IP hops. 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.

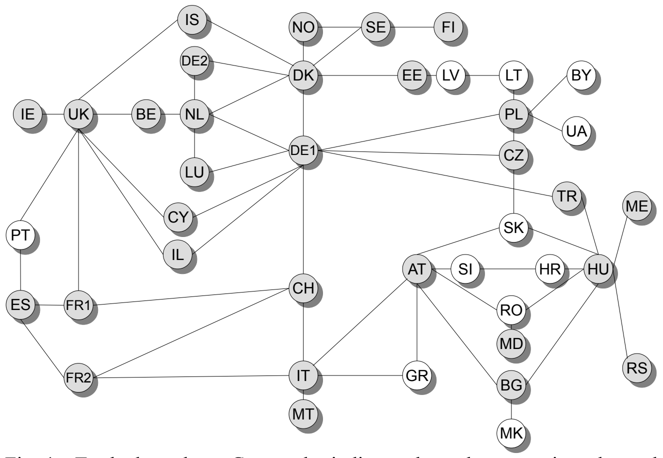


Fig. 4 – Testbed topology. Grey nodes indicates that a datacenter is co-located with the PoP, at 1 IP hop distance. Each PoP is realized with one router running NSIS. Each datacenter is modeled by a single node, running NSIS.

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 [9].

Fig. 5 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). In addition, such an overhead is not only limited in absolute, but also when compared with size of the file to be moved. In fact, as reported in [10], 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 [10], 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 [10]. Thus, not only the overhead of the service is negligible with respect to the amount of traffic associated to the processing service, but also the relevant benefit is definitely worth its deployment

V. Conclusion

In this paper, we have presented a signaling framework for resource discovery of cloud resources for genomic processing applications. The proposed framework can be used not only to search datacenters able to host the processing, but also caches able to provide with a lower overhead the desired content. The specific characteristics of the proposed solution is the capability to provide results with a controlled proximity degree with respect to the data path connecting involved entities.

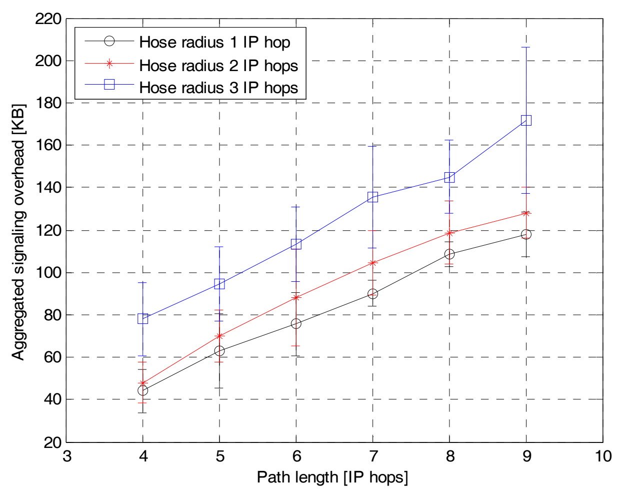


Fig. 5 – Signaling overhead over the whole network as a function of the path length, for different value of the hose radius.

This aspect is particularly important to take decisions able to minimize the impact of genomics processing on the underlying network infrastructure. The results, in terms of network overhead, confirm that the signaling overhead is definitely negligible and thus affordable for any network speed.

Clearly, even if we explicitly referred to the ARES frameworks, this type of signaling is suitable for all network scenarios in which large amounts of data have to be moved towards cloud sites for later processing. Thus, this solution is applicable also outside the genomic scenarios.

Future work will consider a complete system description and deployment, including the definition of a suitable optimization function in the decision agent and the measurement of not only signaling overhead, but also network resource consumption during data transfer.

REFERENCES

[1] Wetterstrand KA. DNA Sequencing Costs: Data from the NHGRI Genome Sequencing Program (GSP) Available at: www.genome.gov/sequencingcosts. Accessed on December 11, 2013.

[2] R. Hancock, G. Karagiannis, J. Loughney and S. Van den Bosch, "Next Steps in Signaling (NSIS): Framework" IETF, RFC 4080, June 2005.

[3] X. Fu et al., "NSIS: a new extensible IP signaling protocol suite", IEEE Communications Magazine, 43(10), 2005, pp. 133- 141.

[4] J. Manner et al., "Using and Extending the NSIS Protocol Family", IETF, RFC 5978, October 2010.

[5] H. Schulzrinne, R. Hancock, "GIST: General Internet Signalling Transport", IETF RFC 5971, October 2010.

[6] NSIS-ka, open source NSIS implementation by Kalsruhe University, available at: https://projekte.tm.uka.de/trac/NSIS/wiki/.

[7] M. Femminella, R. Francescangeli, G. Reali, J.W. Lee, H. Schulzrinne, "An Enabling Platform for Autonomic Management of the Future Internet", IEEE Network, Nov./Dic. 2011, pp. 24-32.

[8] M. Femminella, R. Francescangeli, G. Reali, H. Schulzrinne, "Gossip-based signaling dissemination extension for next steps in signaling," IEEE/IFIP Network Operations and Management Symposium (NOMS 2012), April 2012, Maui, HW, USA.

[9] M. Femminella, R. Francescangeli, G. Reali, H. Schulzrinne, D. Valocchi, "Off-path Signaling Extension for General Internet Signaling Transport Protocol", submitted for journal publication. Available at http://arxiv.org/abs/1406.7650.

[10] Mauro Femminella, Emilia Nunzi, Gianluca Reali, Dario Valocchi, "Networking issues related to delivering and processing genomic big data", International Journal of Parallel, Emergent and Distributed Systems, 2014, DOI: 10.1080/17445760.2014.929685

[11] A. O'Driscoll a, J. Daugelaite, R. D. Sleator, "'Big data', Hadoop and cloud computing in genomics", Journal of Biomedical Informatics, vol. 46, 2013, pp. 774–781.

[12] M. Yandell and D. Ence, "A beginner's guide to eukaryoticgenome annotation", Nature Reviews Genetics, vol. 13, May 2012.

[13] 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.

[14] P. Romano, F. Quaglia, "Design and Evaluation of a Parallel Invocation Protocol for Transactional Applications over the Web", IEEE Transactions on Computers, 63(2), 2014, pp. 317-334.

[15] Femminella M., Reali G., D. Valocchi, Francescangeli R., Schulzrinne H, "Advanced caching for distributing sensor data through programmable nodes", IEEE LANMAN 2013, Bruxelles, Belgium, 2013.

[16] OpenStack, http://www.openstack.org/, accessed on 30 April 2014.

[17] The GÉANT pan-European research and education network. Available at http://www.geant.net

[18] Satyen Abrol, Latifur Khan and Bhavani Thuraisingham, "An Ontology-based System for Cloud Service", Collaboratecom 2012, October 14–17, 2012 Pittsburgh, United States.

[19] Taekgyeong Han, Kwang Mong Sim, "An Ontology-enhanced Cloud Service Discovery System", Proc. of International MultiConference of Engineers and Computer Scientists (IMECS 2010), March 17-19, 2010, Hong Kong.

[20] Jaeyong Kang, Kwang Mong Sim, "Towards Agents and Ontology for Cloud Service Discovery", International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC 2011), October 2011, Beijing, China.

[21] Rajiv Ranjan, Lipo Chan, Aaron Harwood, Rajkumar Buyya, Shanika Karunasekera, "Decentralised Resource Discovery Service for Large Scale Federated Grids", E-SCIENCE '07, Bangalore, India, December 2007.

[22] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, Cesar A. F. De Rose, Rajkumar Buyya1, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms", Softw. Pract. Exper. 2011; 41:23–50.

[23] Wu-Chun Chung, Chin-Jung Hsu, Kuan-Chou Lai, Kuan-Ching Li, Yeh-Ching Chung, "Direction-Aware Resource Discovery Service in Large-Scale Grid and Cloud Computing", IEEE International Conference on Service-Oriented Computing and Applications (SOCA 2011), December 2011, Irvine, United States.

[24] Wright et al., "A constraints-based resource discovery model for multi-provider cloud environments", Journal of Cloud Computing: Advances, Systems and Applications 2012, 1:6.

[25] Ranjan R., Harwood A., Buyya R., "Peer-to-peer-based resource discovery in global grids: a tutorial", IEEE Communications Surveys & Tutorials, vol. 10, No. 2, 2008.

[26] Gregor Pipan, "Use of the TRIPOD overlay network for resource discovery", Future Generation Computer Systems, Vol. 26, No. 8, October 2010, Pages 1257–1270.

[27] Leyli Mohammad Khanlia, Saeed Kargarb, "FRDT: Footprint Resource Discovery Tree for grids", Future Generation Computer Systems, Vol. 27, No. 2, February 2011, Pages 148–156.

[28] Guan Le, Ke Xu, Junde Song ,"Gossip-based Hybrid Multi-attribute Overlay for Resource Discovery in Federated Clouds", Ninth IEEE International Conference on e-Business Engineering (ICEBE 2012).

[29] Alhamazani K., Mitra K., Lizhe Wang, Rabhi F., Ranjan R., "Cloud monitoring for optimizing the QoS of hosted applications", IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom), 2012, pp. 765-770, 2012.