STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXIII**, Number 2, 2018
DOI: 10.24193/subbi.2018.2.03

APPLICATION FUNCTIONS PLACEMENT OPTIMIZATION IN A MOBILE DISTRIBUTED CLOUD ENVIRONMENT

ANNA REALE, PÉTER KISS, CHARLES FERRARI, BENEDEK KOVÁCS, LÁSZLÓ SZILÁGYI, AND MELINDA TÓTH

Abstract. Distributed Computing in 5G Mobile Networks is a potential requirement for certain applications that depends on low latency and information sharing through or with data information sources. Such applications may be observed as a distributed application. We present a tool and method to optimize the deployment of distributed applications, dividing it into Modules, in a 5G Mobile Network environment. To do so we apply an approximation algorithm for the Path Computation and Function Placement Problem described in [1]. We show that under certain circumstances it is beneficial to deploy parts of such applications in a Cloud Computing environment with Distributed Cloud resources at the Mobile Network Edge. We verify our findings with an example, an Augmented Reality application.

1. Introduction

5G mobile networks promise high bandwidth and low latency on the radio interface for both downlink and uplink data [2] which capability will enable new type of applications and services. Such mobile applications include Augmented Reality (AR), Virtual Reality, Gaming and many other bandwidth heavy and latency sensitive applications, potentially applied for critical use cases such as Intelligent Transportation Systems or Surveillance.

Deploying an application on a 5G network with distributed cloud capabilities involves the choice of were to allocate what parts of the applications. It depend both on the application itself and on the involved network. In this

Received by the editors: March 31, 2018.

2010 *Mathematics Subject Classification.* 68U20,68U35,90C35.

1998 *CR Categories and Descriptors.* C.4[**PERFORMANCE OF SYSTEMS**]: Modeling techniques– ; C.2.4[**COMPUTER-COMMUNICATION NETWORKS**]:Distributed Systems– *Distributed applications* .

Key words and phrases. 5G, Distributed Cloud,Distributed Computing, Application Partitioning, Edge Computing, Augmented Reality.

This paper was presented at the 12th Joint Conference on Mathematics and Computer Science, Cluj-Napoca, June 14-–17, 2018.

work we propose a method and tool to facilitate planning of refactoring of an existing applications into modules. To do so we calculate the best placement of the modules on the network compute servers, given user profile and context conditions informations such as: typical user request, policy per type of user (SLA), available bandwidth, network node types, available computation power and cost of computation on the device, in the distributed edge and central cloud.

We have chosen a resource demanding AR application for our test. We assume that such applications can benefit both from involving low latency external computation power and significant sized, but affordable, storage capabilities. To validate our assumptions we apply the mentioned tool and measure the application properties under certain circumstances and network constraints.

Main contributions of this work is the proposal of a method to automatize application partitioning and placement in a 5G/Edge environment. We introduce a possible tool-set implementing our method, its experimental setup and evaluation.

Most works on this topic focuses only on the problem of task partitioning and placement, while they seldom addresses issues of multiple users and load balancing. For this purpose in our work we integrate an approach from network service placements and apply a variation of the approximation algorithm for the Path Computation and Function Placement Problem described in [1].

2. Background

In the following section we give an informal description of the problem we address and contextualize it by referring to related works.

2.1. Problem Statement.

To partition an application and to deploy its modules in a 5G network with edge computing resources, we need to calculate what is the (sub)optimal grouping of the components and their placement that maximizes the usage of network capacities in a given instant. Giving a flexible method to automate this process enables applications to adapt to environment changes through dynamical reallocation of resources.

This task can be reduced in three main steps:

(1) Model the application through hybrid analysis (using both static analysis and heuristics from dynamic profiling of the given application);
(2) Calculate a partition to divide the application in modules minimizing their interactions and communication cost while maximizing the responsiveness and perceived performances;
(3) Decide best placements of the modules in the given network;

2.2. **Related Works.** State-of-the-art Application Partitioning Algorithms (APAs), applied to distributed processing, still face many issues and challenges. An extensive summary concerning APAs in Mobile Cloud Computing is proposed in [3]. Based on the model used as an input to the partition, the authors identify three main categories of existing solution: graph-based, Linear Programming (LP) or hybrid solutions between the two.

2.2.1. *Application Partitioning.* Solutions based on graph representations of the applications may use data flow graph to represent data dependencies between operations [4, 5, 6],

while class dependency graphs can be used to describe the structure of an application [7, 4].

The authors in[8] partition object-oriented programs by generating an Object Relation Graph (ORG) to estimate the runtime objects and their interactions, and then applying graph partitioning to this ORG. In [9] a two-layer graph structure is used, in which a second graph, the Target Graph (TG) accounts for the various target infrastructures and distribution objectives.

Graph-based APAs require efficient manual annotation techniques, it is up to the programmer to balance the metrics and specify metrics function. In addition, a greatresource overhead is generated in case of applications with a large number of components. Finally the performance of graph-based solutions depends on the application characteristics: the analysis is easily performed if the applications is already modularized somehow. On the other hand, LP based solutions always produce optimal results for a particular objective function [10, 11, 12]. LP APAs need dynamic scheduling techniques, extra profiling and resource monitoring, thus they also cause high overheads.

Hybrid solutions extract the important features of graph-based APAs and LP-based APAs in order to improve the performance and mitigate overheads but, in most cases, at the expenses of generating only a sub-optimal partition [13, 14, 6, 15, 16].

In this work we plan to create a tool to help to define a set of candidates for partitions according to different network conditions. Such partition database could be used in the future to allow dynamic reallocation of the application, based only on light dynamic profiling of the context it runs in.

2.2.2. *Application Modeling and graph partitioning.* The NP-hard graph partitioning problem is a fundamental issue in many other domains of computer science, such as parallel processing [17] and load balancing [18]. In grid computing the graph partitioning problem has been used to define parallel tasks to be deployed on heterogeneous infrastructures. As stated by [16] many proposed algorithms, such as MiniMax, VHEM, QM, PaGrid, and MinEX, use a multilevel paradigm, while others use simulated annealing [19].

In the literature, decomposition techniques based on graphs [20] involve three macro steps: (1) Identify level of granularity for the partitions elements or tasks; (2) Analyze the application with task dependency and interaction graphs, (3) Map possible valid partitions.

Properties of tasks that affect the quality of mapping are: feasibility of task generation, size of tasks and size of data handled by the task or passed between two of them.

In fact, one needs to take in consideration the interaction between the partitioned task: they often share data and may have a precise sequential order [21, 22].

In scheduling the interaction graph is used to represent the application dividing it into tasks. Nodes in the graph are the tasks while their weights denote the amount of work to be performed by the task. Edges represent the interactions between tasks. Generally edges are undirected, when directed they are used to show the direction of the flow of data (if the flow is unidirectional). Weights on edges contain the cost of communication. Shared data may imply synchronization protocols (mutual exclusion, etc) to ensure consistency.

In distributed systems theory, the interaction graph is also referred as the Control Flow Graph (CFG). A CFG is a representation, using graph notation, of all paths that might be traversed through a program during its execution. The graph provides the structure of the program as a whole, among others, making explicit all of the paths that are induced by a conditional branch. A function dependency graph, for example, is a sub-graph of this graphs, having has partition granularity the function. Dependency between functions implies interaction (calls or data passing) between them.

A Call Graph (CG) is a dependency graph that represents calling relationships between functions in a computer program. Each node denotes a procedure and each $edge(f, g)$ indicates that procedure f calls g. Thus, a cycle in the graph indicatesrecursive calls.

Call graphs are results of a basic program analysis, that can be used for model programs, or as a basis for further analyses. Call graphs can be dynamic or static. A dynamic call graph is a record of an execution of the program, for example as output by a profiler. Thus, a dynamic call graph can be exact, but only describes one execution of the program.

In object-oriented languages the potential target method(s) of many calls cannot be precisely determined solely by an examination of the source code

[23]. Thus, to build the call graph, it is necessary to have inter-procedural data and control-flow analysis of the program.

Granularity. Program partitioning has been used in application offloading for resource-constrained devices. Previous works propose computation offloading at different levels of granularity: Module level [5], Method level [13], Object level [8, 9], Thread level [14, 6], Component level [10, 16]. Various metrics can help to decide a good level of granularity for the partition of a graph. For instance, the critical path, being the longest directed path between any start and finish nodes, indicates what is the shortest time needed to execute. The time can be calculated from it's length, computed by sum of the weights of the traversed nodes. The average degree of concurrency, that is the total amount of work divided by critical path length is also a common metric. Related to the size of the partitions we consider important the size of the data associated with tasks, because it helps to minimize volume of data-exchange and maximize data locality. Also the size of context is an indicator of how affordable or expensive the communication between tasks can be.

2.2.3. *Placement Models.* Appropriate resource allocation is a very old issue in different disciplines. In this section, we present two resource allocation problems in computer networks: placement of Virtual Machines (VM) in cloud computing and placement of Virtual Networks Function (VNF).

VM Placement. With the term VM placement we refer to the process of selecting the most appropriate physical machines for VMs. According to [24], objectives of VM placement are maximizing resource utilization, reliability and availability. There are several approaches to VM placement in the literature [25, 26, 27], some variants even consider dynamic placement and multi-clouds placement. For instance, [25] uses traffic-aware VM placement to improve the network scalability in data center, defining it as an hard optimization problem solved by a two-tier approximation algorithm to overcome very large sizes.

Service Chain Placements in NFV. Service Function Chaining (SFC) [28] aims to overcome the limitation of static deployment models applying algorithms that can optimally map SFC to substrate network. This category of algorithms is referred as "Virtual Network Functions Placement (VNFP)" algorithms [29]. As explained in [30], in this category of placement problems, we are given a physical network, VNF specifications, and a set of service requests. The algorithm performs the three following steps:

(1) Calculate an optimal number of needed VNF types, all the VNFs that should be instantiated compose a set.
(2) Place VNFs to physical nodes such that the demand of VNFs do not exceed the capacity of physical nodes;

(3) Assign service requests to VNFs such that the demand of service requests do not exceed the capacity of VNFs.

However, the three steps are not independent, and their order depends from the implementation of the algorithm and the problem statement. For example, [31, 32] give an Integer Linear Programming(ILP) formulation; many others, preferring fast heuristics to allow real time decisions [33], propose a dynamic programming; [34] gives a Mixed ILP formulation and a heuristic algorithm that solve the problem incrementally, which can solve the problem for incoming flows without impacting existing flows. Among the meta-heuristic solutions, [35] proposes a method based on genetic algorithms while [36] considers a greedy algorithm and a tabu search-based algorithm.

Although in our example we will not work with VNF specific algorithms, we claim that our methods may be applied to them as well. This is especially true for network functions such as User Plane Function and special observability, monitoring, tracing, logging and analytics VNFs.

3. A model for application partitioning and deployment

In the following sections we provide formal description of the models and methods used to construct our simulation toolset, followed by a description of the real application we used as first input for it.

3.1. **Models.**

To map an application based on functions granularity we construct a function dependency graph. In scheduling and load balancing, this method is used when the application can be described from the static definition of the dependency graph and the function sizes are known.

Determining an optimal mapping of the function dependency graph becomes solvable if there are good heuristics available to estimate the data flow and a structured call graph. In our case we use a static analyzer tool to generate the function call graph from the source code. Then we run the application and collect for each function, using a non-intrusive dynamic profiler, the percentage of runtime spent in it. In addition, for each link between two functions, we collect the number of times the callers calls the callee. We normalize those results and store them in the call graph as node and edge weights. The normalized edge weights will define the dependency between the two connected functions, thus to estimate how to separate the application to reduce such interactions it will be enough to use a minimum edge-cut strategy. The node weight is a useful information to estimate the complexity of the computations handled by the function, this value can be used to balance the partitions or to deploy different optimization strategies. For example if we want for the User Equipment (UE) accessing the application to save energy, we could want to concentrate the computational load on the Edge or on the Cloud the UE can connect to.

It is important to stress how we are deploying a context-insensitive construction of the application call graph. In fact, each node will represent exactly a single contour: an analysis-time representation of a function.

3.2. **Application Partitions.** Having our weighted graph, we partition it using Multilevel [37] version of the Kernighan - Lin [38] algorithm (MLKL). We choose the multilevel strategy to be able to handle potentially large function call graphs. After running the algorithm, the application will be divided into a number of Modules where the ceiling for the number of partitions can be selected by the user, and the weight of each Module and the interaction frequencies between them are derived from the original graph. The directed graph resulting as an output of the partition steps represents the due interactions between the modules. This new level of abstraction means that we lose information like when and for how long two specific functions in two modules will interact at running time. Such information also depends on the user interaction with the application itself and can vary from instance to instance.

We decided to adopt a pessimistic approach in the module deployment phase, taking as the weight of the assumption making that we want to instantaneously run all the modules.

3.2.1. *MLKL and METIS.* MLKL is a Multilevel Version of the KL algorithm. It means that the algorithm is applied in three repeated phases: Coarsen, Partition and Uncoarsen.

First, the algorithm coarsen down the graph by merging connected vertices until a small graph is obtained. Then this graph is partitioned and uncoarsened again, while optimizing the partition in each uncoarsening step using KL as refinement function.

The KL algorithm is iterative. It starts with an initial partition and in each iteration it finds two subsets which guarantee a smaller edge-cut. If such subsets exist, then it moves them to the other part and this becomes the partition for the next iteration. The algorithm continues by repeating the entire process. In the implementation proposed by [39] the KL algorithm computes for each vertex v a quantity called gain which is the decrease (or increase) in the edge-cut if v is moved to the other part. The algorithm terminates when the edge-cut does not decrease after x number of vertex moves and those last moves are undone to get the maximum edge cut.

3.3. **Network Model.** Now we test our partition behavior in our network to see what configuration gets the maximum out of the same network conditions.

Thus we deploy all tasks in all nodes and see what are used the most to satisfy network demand considering network capacities.

The network is a fixed set of computational resources and communication links. The network is represented by a graph $N = (V, E)$, where V is the set of nodes and E is the set of edges.

We classify nodes in three categories: UE, Edge Cloud Servers and Central Cloud Servers. Note that the classes are disjoint and our proposed method works with other types of disjoint classification of nodes as well.

Nodes and edges have capacities. The capacity of an edge $e \in E$ is denoted by $c(e)$, and the capacity of a node $v \in V$ is denoted by $c(v)$. All capacities are positive integers. $c(e)$ represents the available bandwidth between the two network nodes; $c(v)$ depends on the amount of available computational resources and the cost of accessing them. We suppose several UEs

that request services from the application. Each of these services may be different on the *Service type* and the *Location* of the involved nodes. Examples of such services can be a video upstream or augmented downlink video. Each Module is a part of the application that, combined, can solve a certain service request.

3.4. **Service Request.** A service request for user j is specified by a tuple $s_j = (G_j, d_j, b_j, U_j)$, where the components are as follows:

$G_j = (M_j, Y_j)$ is a directed (acyclic) graph called the place-and-route graph (pr-graph). There is a single source and a single sink, that corresponds to the node requesting the service. We denote the source and sink nodes in G_j by $ns_j \in M_j$ and $nt_j \in M_j$, respectively. The other vertices correspond to services or processing stages of a request. The edges of the pr-graph are directed and indicate precedence relations between pr-vertices.

The demand of a request s_j is d_j and its benefit is b_j. Demand is computed from the cost of running a complete module. The benefit is the benefit of serving that precise request of service. It should be calculated from the SLA, but it depends on the network owner as well. By scaling, we may assume that $min_j\{b_j\} = 1$.

We map the User Equipment service request s_j as the realization of a path trough the directed partition graph representing the application. In this case the demand of a Module can be calculated over the cost of each function composing the Module that composes the specific service request. The routing cost from one Module to the other become than the overhead or transmission cost brought by the selected Module interaction scheme. For example, the size of the data to be transferred from one Virtual Machine to the other to keep the state consistent trough all their network instances [40]. The impact of the service request on the network thus can vary only based on the location of the Modules. To specify the possible realization of a pr-graph in the physical network we use a function $U_j : M_j \cup Y_j \to 2^V \cup 2^E$ where $U_j(m)$ is a set of "allowed" nodes in N that can perform module m, and $U_j(y)$ is a set of "allowed" edges of N that can implement the precedences and routing requirement that corresponds to y. We now define for each service request s_j the product network $pn(N, s_j)$. The node set of $pn(N, s_j)$, denoted by V_j, is defined as $V_j \triangleq \cup_{y \in Y_j}(U_j(y) \times y)$. We refer to the subset $U_j(y) \times y$ as the y-layer in the product graph. The edge set of $pn(N, sj)$, denoted E_j, consists of two types of edges $E_j = E_{j,1} \cup E_{j,2}$ defined as follows:

(1) Routing edges connect vertices in the same layer, they represent the physical links in the network.
$E_{j,1} = \{((u, y), (v, y)) \mid y \in Y_j, (u, v) \in U_j(y)\}$
(2) Processing edges connect two copies of the same network vertex in different layers, representing the move from one Module to the consecutive one in the service chain specified in Y.
$E_{j,2} = \{((v, y), (v, y')) \mid y \neq y' \in Y_j$ edges with common endpoint m, and $v \in U_j(m)\}$

PCPF problem. The substrate network $N = (V, E)$ and a set of service requests $\{s_i\}_{i \in I}$ described as stated before, are the necessary input for the solution we used for Path Computation and Function Placement Problem (PCFP). The goal is to compute valid realizations $\tilde{P} = \{\tilde{p}_i\}_{i \in I'}$ for a subset of the requests $I' \subseteq I$ so that $\tilde{P}$ satisfies the capacity constraint of N and maximize the total benefit $\sum_{i \in I'} b_i$. For our work, we apply the fractional relaxation of PCFP-problem described in [1]. This is a variation of Raghavan's randomized rounding algorithm for general packing problems [41].

3.4.1. *Experiment Setup.* We created a generic setup for Multi Access Edge Computing partitioning and distribution. It is composed by four resource constrained devices connected with an edge server through redundant networks, where different network setups can be tried. The application has initially all the processing activities done in the server, which collects information from the four connecting devices and performs the processing.

The connections used for the experiment explained in this article were carried with wireless 5 GHz and Ethernet connections, where the client devices were equipped with 100 megabits network shields.

The client devices were equipped with cameras using Sony IMX219 sensors, streaming real-time video to the server. The camera was configured to create frames of 640x480 pixels, 25 frames per second and 4:3 aspect ratio. The connection between the clients and the server was an UDP connection.

3.4.2. *Measurement Tools.* The measurements used to configure the tool are grouped in three independent areas namely: (1) Network performance; (2) Computational performance, and (3) Software processing cost. Each of them composed as described below:

- *Network Performance*: available resources, Jitter between nodes (Latency variation), Locality UE-Edge-Cloud;
- *Computational Performance*: Machine capabilities, Network connection speed, Processor capabilities, Memory availability;
- *Software processing cost*: Dependency between two functions (number of calls), Resource usage from App (Average memory Usage Mb per function), Cost of the software execution (processor cycles that are required to execute each function of the software).

The software measurements were taken using instrumented profiling tools, Valgrind [42] and our self-produced tools.

3.5. **Modeling the example application.** During our experiment, we chose to start at a function level granularity for our applications, to be able to partition it into Modules. A typical AR application has the following chain of services: *capture, preprocessing, detection, recognition, tracking, rendering.* Each of this service calls a sequence of *Modules.* Note that these *Modules* may be different for different applications. Another example can be a partitioning of a Linear Unicast service which may have the following modules: Streaming, Origination, Manipulation, Encapsulation, Encryption, Encoding, according to [43].

In our first example (Figure 1) we show the result of running the partitioning only on the call graph of the *capture* service (involving camera calibration), where different colors refers to different modules and the number of requested partitions was 5. As second example (Figure 2), we show the Function Call graph generated only by the camera calibration part of the application 2 on the edges the calls between functions and on the nodes the CPU clocks.

The result of the whole AR application partition is shown in Figure 3a. The *Start* node represents the interface with the User Equipment, the $Main$ node is the partition in which the known entry point of the program execution is located.

The arrows are the interaction between $Modules$. For example, we know that $Main$ can receive data and be called by $M1$, but every call from the $Main$ goes either to $M2$ or to $M4$. In the construction of service requests we kept the following interaction constraints: if the service needed by UE is contained in $M1$ the shortest possible request path became $\{(Start, Main)(Main, M2)$ $(M2, M1)(M1, Main)\}$.

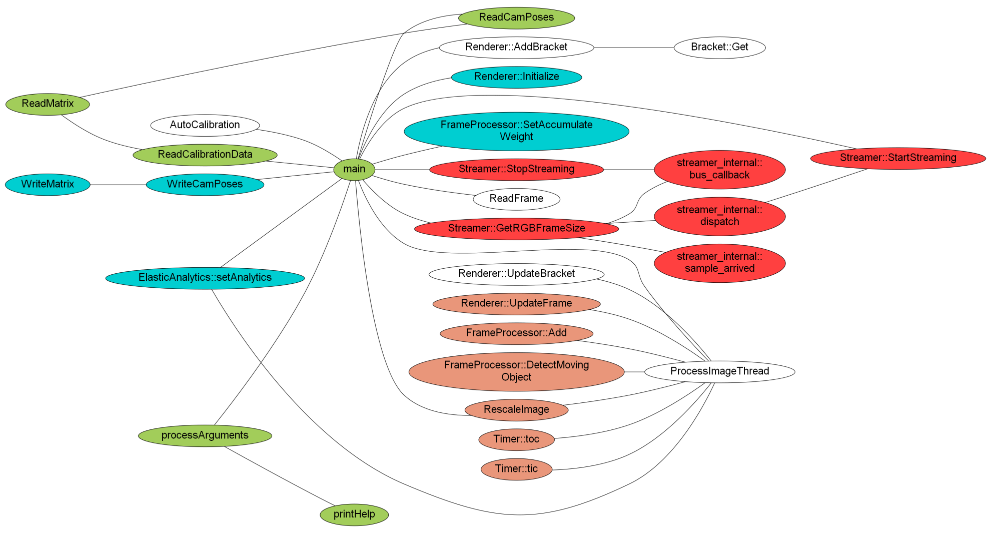


FIGURE 1. Partitioned Call Graph for an Image Capture Service

The Simulation Network will represent the possible interactions between nodes. In our simulation, we decided to allow both direct UE to Edge and UE to Cloud communications (Figure 3b). We consider an average transmission overhead in the range of few ms (1 or 2) between UE and Edge nodes, of 25 ms betweens Edge and Local Clouds and of the sum of the two (26 or 27) between UE and Cloud. The resulting pr-graph

is shown in Figure 3c. We normalize the capacities of the network nodes based on available memory. We experimented on a SLA scenario where we want to reduce the computation time at a minimum overhead.

For the same computation demand we define the benefit of a chosen deployment path based on the computation cost (we estimated the Edge to be four times more expensive than the cloud) and the average transmission overhead. Both weights were calculated as the coefficient of variation of the relative measures registered on the Experiment Setup.

In all the generated simulations a deployment was proposed for which 12 contemporary simulated user requests where served, respecting the capacity constraints of different networks, obtaining maximal benefit flows like the one shown in Figure 3d. On average, the benefit was higher than running everything on the device: a complete run on the single device lasted on average 9444 ms, while the average run on our simulations saved from 667 ms up to 3904 ms with maximum average communication overhead per request being 1152 ms (Table 1).

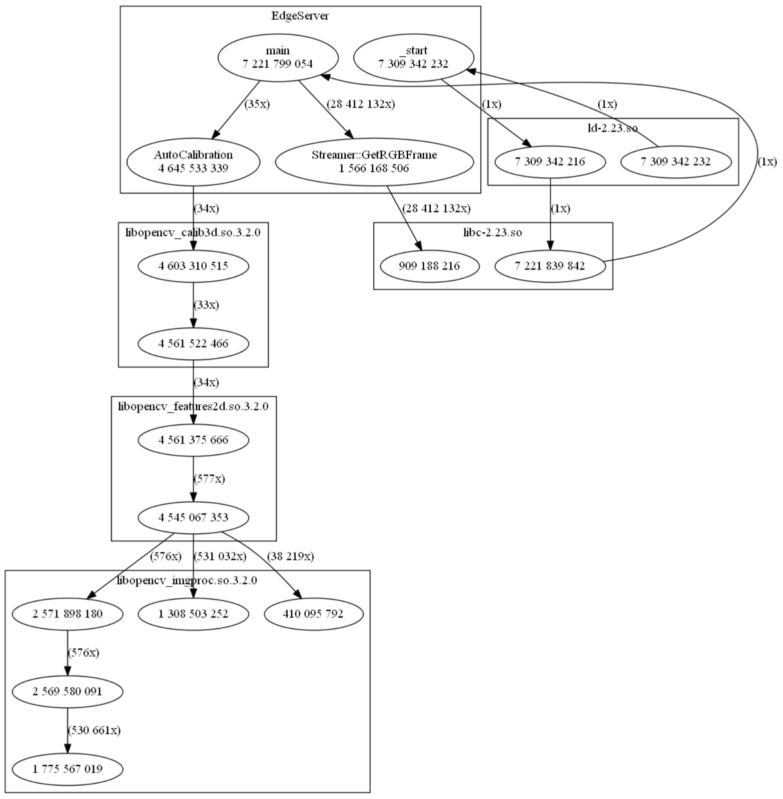


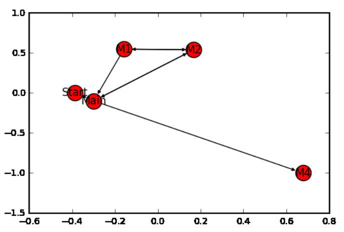
Figure 2. Camera calibration call graph

	Run1	Run2	Run3	Run4	Run5	Run6	Run7	Run8
Edges	4	2	5	5	5	3	4	4
UEs	3	6	3	3	3	5	4	4
Local Cloud	1	1	1	1	1	1	1	1
Average overhead per request (ms)	37	58	144	29	1152	583	84	148
Average benefit per request (ms)	3941	1348	3743	3348	1820	2841	3340	2395
Average final benefit (ms)	3904	1289	3598	3319	667	2258	3255	2246

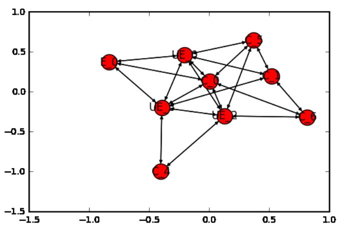
Table 1. Experiment result: benefits of partitioning and deployment of the same application on different networks topologies

4. Conclusions and Future Work

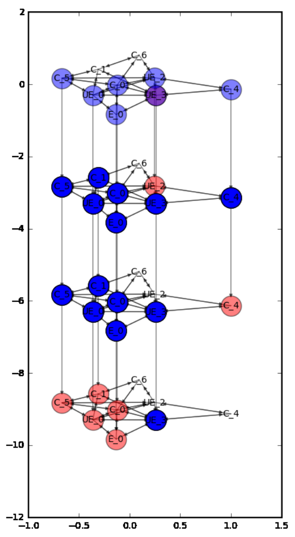
In this work we described the methods and the algorithms we used to develop a first prototype of our tool to partition and deploy an application in a



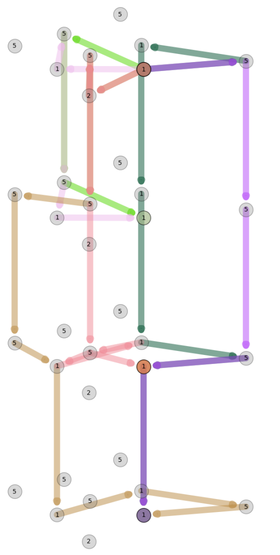
(A) AR Application after partition



(B) Simulation Network



(C) Complete AR application pr-graph



(D) Maximal benefit flows satisfying all user requests

FIGURE 3. Models of a real application

5G distributed network. We believe this is the minimum analysis to be performed to, in the future, be able to implement a dynamic reallocation of the applications based on variation of the context conditions. The problem was divided in three steps. First selecting the application granularity and construct a graph model. Than reduce it into Modules by solving the NP-hard graph partitioning problem it represents; finally implement and apply a fractional relaxation of the Path Computation and Function Placement Problem as described by [1].

Simulation were run with various simultaneous request of service. For our specific set up and our AR application, there is a possibility to implement a distributed scenario with a reasonably low overhead.

The next step would be to implement the new application partition suggested by the framework and locate them in the physical network to verify how close our simulations are to reality. We hope by running the new deployment to be able to perfect the parameters we used to describe the network capacities and the benefits of the distributed execution. Interesting measures to validate the outcome on different AR applications could be quality and efficiency related measures: for example Video Quality as Average Bit rate expressed in Kbps.

5. Acknowledgements

This research has been co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013). We would like to thanks our university supervisors and the 5G Ericsson Garage team: Gábor Fábián, Zoltán Gera.

References

[1] G. Even, M. Rost, and S. Schmid, "An approximation algorithm for path computation and function placement in sdns," in *International Colloquium on Structural Information and Communication Complexity*, pp. 374–390, Springer, 2016.

[2] E. AB, "5g systems, enabling the transformation of industry and society," white paper, ERICSSON, 2017.

[3] J. Liu, E. Ahmed, M. Shiraz, A. Gani, R. Buyya, and A. Qureshi, "Application partitioning algorithms in mobile cloud computing: Taxonomy, review and future directions," *Journal of Network and Computer Applications*, vol. 48, pp. 99–117, 2015.

[4] I. Giurgiu, O. Riva, D. Juric, I. Krivulev, and G. Alonso, "Calling the cloud: enabling mobile phones as interfaces to cloud applications," in *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*, p. 5, Springer-Verlag New York, Inc., 2009.

[5] L. Yang, J. Cao, Y. Yuan, T. Li, A. Han, and A. Chan, "A framework for partitioning and execution of data stream applications in mobile cloud computing," *ACM SIGMETRICS Performance Evaluation Review*, vol. 40, no. 4, pp. 23–32, 2013.

[6] R. Newton, S. Toledo, L. Girod, H. Balakrishnan, and S. Madden, "Wishbone: Profile-based partitioning for sensornet applications.," in *NSDI*, vol. 9, pp. 395–408, 2009.

[7] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, and D. Milojicic, "Adaptive offloading inference for delivering applications in pervasive computing environments," in *Pervasive Computing and Communications, 2003.(PerCom 2003). Proceedings of the First IEEE International Conference on*, pp. 107–114, IEEE, 2003.

[8] A. Messer, I. Greenberg, P. Bernadat, D. Milojicic, D. Chen, T. J. Giuli, and X. Gu, "Towards a distributed platform for resource-constrained devices," in *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pp. 43–51, IEEE, 2002.

[9] L. Wang and M. Franz, "Automatic partitioning of object-oriented programs for resource-constrained mobile devices with multiple distribution objectives," in *Parallel and Distributed Systems, 2008. ICPADS'08. 14th IEEE International Conference on*, pp. 369–376, IEEE, 2008.

[10] L. Yang, J. Cao, and H. Cheng, "Resource constrained multi-user computation partitioning for interactive mobile cloud applications," *Technical reort. Department of Computing, Hong Kong Polytechnical University*, 2012.

[11] D. Kovachev, "Framework for computation offloading in mobile cloud computing," *IJIMAI*, vol. 1, no. 7, pp. 6–15, 2012.

[12] M.-R. Ra, B. Priyantha, A. Kansal, and J. Liu, "Improving energy efficiency of personal sensing applications with heterogeneous multi-processors," in *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, pp. 1–10, ACM, 2012.

[13] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pp. 49–62, ACM, 2010.

[14] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*, pp. 301–314, ACM, 2011.

[15] M. Goraczko, J. Liu, D. Lymberopoulos, S. Matic, B. Priyantha, and F. Zhao, "Energy-optimal software partitioning in heterogeneous multiprocessor embedded systems," in *Proceedings of the 45th annual design automation conference*, pp. 191–196, ACM, 2008.

[16] T. Verbelen, T. Stevens, F. De Turck, and B. Dhoedt, "Graph partitioning algorithms for optimizing software deployment in mobile cloud computing," *Future Generation Computer Systems*, vol. 29, no. 2, pp. 451–459, 2013.

[17] B. Hendrickson and T. G. Kolda, "Graph partitioning models for parallel computing," *Parallel computing*, vol. 26, no. 12, pp. 1519–1534, 2000.

[18] H. Meyerhenke, B. Monien, and S. Schamberger, "Graph partitioning and disturbed diffusion," *Parallel Computing*, vol. 35, no. 10-11, pp. 544–569, 2009.

[19] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[20] F. Berman, "High-performance schedulers," *The grid: blueprint for a new computing infrastructure*, vol. 67, pp. 279–309, 1999.

[21] D. L. Long and L. A. Clarke, "Task interaction graphs for concurrency analysis," in *Proceedings of the 11th international conference on Software engineering*, pp. 44–52, ACM, 1989.

[22] M. Naghibzadeh, "Modeling workflow of tasks and task interaction graphs to schedule on the cloud," *CLOUD COMPUTING 2016*, p. 81, 2016.

[23] D. Grove and C. Chambers, "Ibm research report an assessment of call graph construction algorithms," 06 2000.

[24] M. Masdari, S. S. Nabavi, and V. Ahmadi, "An overview of virtual machine placement schemes in cloud computing," *Journal of Network and Computer Applications*, vol. 66, pp. 106–127, 2016.

[25] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *INFOCOM, 2010 Proceedings IEEE*, pp. 1–9, IEEE, 2010.

[26] R. A. da Silva and N. L. da Fonseca, "Algorithm for the placement of groups of virtual machines in data centers," in *Communications (ICC), 2015 IEEE International Conference on*, pp. 6080–6085, IEEE, 2015.

[27] J. Chase, R. Kaewpuang, W. Yonggang, and D. Niyato, "Joint virtual machine and bandwidth allocation in software defined network (sdn) and cloud computing environments," in *Communications (ICC), 2014 IEEE International Conference on*, pp. 2969–2974, IEEE, 2014.

[28] J. Halpern and C. Pignataro, "Service function chaining (sfc) architecture," tech. rep., 2015.

[29] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on*, pp. 171–177, IEEE, 2015.

[30] Y. Xie, Z. Liu, S. Wang, and Y. Wang, "Service function chaining resource allocation: A survey," *arXiv preprint arXiv:1608.00095*, 2016.

[31] T. Taleb, M. Bagaa, and A. Ksentini, "User mobility-aware virtual network function placement for virtual 5g network infrastructure," in *Communications (ICC), 2015 IEEE International Conference on*, pp. 3879–3884, IEEE, 2015.

[32] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pp. 1346–1354, IEEE, 2015.

[33] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On orchestrating virtual network functions," in *Network and Service Management (CNSM), 2015 11th International Conference on*, pp. 50–56, IEEE, 2015.

[34] A. Mohammadkhan, S. Ghapani, G. Liu, W. Zhang, K. Ramakrishnan, and T. Wood, "Virtual function placement and traffic steering in flexible and dynamic software defined networks," in *Local and Metropolitan Area Networks (LANMAN), 2015 IEEE International Workshop on*, pp. 1–6, IEEE, 2015.

[35] M. Bouet, J. Leguay, and V. Conan, "Cost-based placement of virtualized deep packet inspection functions in sdn," in *Military Communications Conference, MILCOM 2013-2013 IEEE*, pp. 992–997, IEEE, 2013.

[36] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and S. Davy, "Design and evaluation of algorithms for mapping and scheduling of virtual network functions," in *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*, pp. 1–9, IEEE, 2015.

[37] B. Hendrickson and R. W. Leland, "A multi-level algorithm for partitioning graphs.," *SC*, vol. 95, no. 28, pp. 1–14, 1995.

[38] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations research*, vol. 21, no. 2, pp. 498–516, 1973.

[39] G. Karypis and V. Kumar, "Metis – unstructured graph partitioning and sparse matrix ordering system, version 2.0," tech. rep., 1995.

[40] K. Ha, P. Pillai, W. Richter, Y. Abe, and M. Satyanarayanan, "Just-in-time provisioning for cyber foraging," in *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pp. 153–166, ACM, 2013.

[41] P. Raghavan, *Randomized rounding and discrete ham-sandwich theorems: provably good algorithms for routing and packing problems.* University of California. Computer Science Division, 1986.

[42] N. Nethercote and J. Seward, "Valgrind: a framework for heavyweight dynamic binary instrumentation," in *ACM Sigplan notices*, vol. 42, pp. 89–100, ACM, 2007.

[43] W. Scott, "Content delivery networks (cdn): Caching principles, architecture, and resource optimization." `https://www.slideshare.net/hacktivism/cisco-live-content-delivery-networks-cdn`, 2017. Online; accessed 29-March-2018.

Faculty of Informatics, ELTE university, Budapest, Pázmány Péter stny. 1/C., 1117 Hungary

Ericsson Hungary Research and Development Center, Budapest, Magyar tudósok körútja 11, 1117 Hungary

Email address: {anna.reale, axx6v4, svu938, toth_m}@inf.elte.hu

Email address: {benedek.kovacs,laszlo.szilagyi}@ericsson.com