Mapping the allocation of resources for 5G slices to the k-MUTEX with n instances of m resources problem

Guillaume Fraysse*†
*Orange
France
Email: guillaume.fraysse@orange.com

Jonathan Lejeune†, Julien Sopena†, Pierre Sens†
†Sorbonne Université, UPMC Univ Paris 06
CNRS, Inria, LIP6
F-75005, Paris, France
Email: firstname.lastname@lip6.fr

***Abstract*—The introduction of slicing in 5G networks requires to consider the infrastructures as shared between not only users but also between independent services with heterogeneous requirements. This problem can be seen as a variation of the k-mutex problem. In this paper we introduce two contributions. First we propose a formulation of the problem of allocation of resource for 5G slicing. Then we present adaptation of an existing algorithm for a more general k-mutex problem to our problem and the experimental results of tests run in our simulator showing the performances compared to other algorithms.**

***Keywords*—Computer network management, Distributed algorithms, Network Slicing, Distributed Systems, k-mutex, drinking philosophers, deadlock**

I. Introduction

Fifth Generation (5G) mobile networks raise an allocation of resources problem for Slicing. 5G Slicing leverages the virtualization, Cloud Computing, Software-Defined Networking (SDN) and Network Functions Virtualization (NFV) technologies to allow the creation of network slices isolated from each other. This approach allows each of the slices to run different Network Services that have distinct objectives. But the different slices share the same underlay resources (e.g. compute, storage, network). 5G slicing requires the possibility for multiple clients to use the same infrastructures for different services.

Resource allocation in the context of 5G Slicing can be seen as a Distributed Mutual Exclusion problem. Several variations of the problem exist for system with a single instance of multiple resources or for system with multiple instances of a single resource but few work has addressed the mutual exclusion of systems such as 5G Slices with multiple instances of multiple resources.

Our contributions: First, we define in Section II the problem of the concurrent allocation of multiple resources in the 5G Slicing context as a generalized k-mutex problem where multiple processes try to allocate N instances of M resources. Second, we present, in Section IV, and evaluate, in V, several algorithms that solves the problem using our our discrete event simulator. We survey methods currently used to address related problems in Section III Finally, Section VI introduces future work to address the full scope of the problem.

II. Problem Statement

A. 5G Slicing

The Fifth Generation of Mobile Networks will be the first generation that is designed from the start to address usages other than the traditional Mobile BroadBand usage that provides network connectivity to mobile devices.It addresses a broader range of usages, the three main use cases that are the focus on the standardization bodies are Enhanced Mobile Broadband (eMBB), Massive Machine Type Communications (mMTC) and Ultra-reliable and Low Latency Communications (URLLC). These usages have different requirements that can be seen as combinations of three basic components: bandwidth, latency and density.

The 5G slicing concept was first introduced by the Next-Generation Mobile Networks (NGMN) in a Technical Document [1] in January 2016. Since then several Standards Developing Organization (SDO) and Industry Fora have launched works to analyze the use-cases and impact of 5G Slicing. 5G slices allow to host several independent end-to-end logical networks on a shared physical infrastructure. For that it capitalizes on the recent evolution of infrastructures detailed above: IaaS, SDN and NFV that allow to share a physical infrastructure for multiple Network Functions.

The SLICENET European project addresses the specific topic of Slice Management in Virtualised Multi-Domain, Multi-Tenant 5G Networks and referenced the different on-going or published standards in their deliverables [25].

B. Resource allocation for 5G Slicing

Telecommunications Networks are split across several domains, e.g. datacenters, on a large geographical zone (usually a country or a large region). Each of these domains hosts computing and network resources. These resources are part of the NFV Infrastructure. The servers are running hypervisors to host virtual machines (VMs), the network can be composed of SDN nodes. 5G Slicing offers the possibility to allocate slices of these resources to different usages on this shared infrastructure.

All these resources will be treated as abstract resources and there will be no focus on their nature in the rest of this paper. Instead the focus is on how to dynamically allocate requests for shared resources that are distributed across multiple domains.Each of these resources can be used by only one user (e.g. VM) at any time, so we will treat this as a Distributed Mutual Exclusion problem. Additionally these resources can be split into different types (e.g. RAM, CPU, bandwidth, network physical port), and several resources are available of each type.

C. *Resource allocation as a distributed k-mutex problem*

E.W. Dijkstra introduced the mutual exclusion problem [9] as the guarantee that only one process will access a single shared resource ay any time when multiple processes need to access it. The process that accesses the shared resource is then said to be in the critical section (CS). Since then this work has been generalized.

A first generalization to systems with multiple resources is by Dijkstra himself when he defined the "AND-synchronization problem", also known as the "dining philosophers problem" [8]. This focused on systems were requests were *static*: each process always request the same resources. It has further been generalized as the "drinking philosophers problem" by Chandy et Misra [5] to systems where the requests were *dynamic*. These problems consider situations where there is a single instance of each resource.

Other works focus on systems where there are multiple instances of a single resource. Among these works some focus on systems where multiple processes try to allocate an instance of the resource, this is known as the k-mutex problem [20], [15], [4]. Other works focus on the allocation of k-out of-M resources [21].

Fewer works have addressed a more general variant of this k-mutex problem where there is N instances of M resources.

In the context of 5G slicing defined above, there is more than one resource in the system, and there are multiple instances of these resources that need to be allocated by multiple processes. We consider that the allocation of resources for 5G Slices is a k-mutex problem where there are multiple resources of multiple types.

III. Related work

Cloud Computing Infrastructure as a Service (IaaS) provides Virtual Machines or containers to customers. In this paradigm, resources usually refer to an abstraction of computing resources exposed by the Physical Machines to the hypervisor as defined by [10]. Optimization of resources in this field has led in recent years to a lot of papers and surveys, among them [18], [10], [27] focused on the placement of VM. Their common goal is to improve the way VMs are placed on the Physical Machines that compose a Cloud infrastructure. Several approaches have been tried, [16] compared 18 VM placement algorithms inspired by bin-packing literature which led them to conclude that the choice of the algorithm for VM placement had no significant impactbut that it was the algorithm for the selection of cluster which led to the most significant impact on resource optimization. Some papers focused on VM placements for specific application profiles for example scientific computations in [24]. Other papers focused on finding other heuristics such as respect of Service Level Agreements (SLA) in [14] or security rules in [11]. This bin-packing approaches do not consider dynamic requests, i.e. not known apriori, and do not consider the properties for mutual exclusion we will introduce later in this Section.

Virtual Network Embedding (VNE) is described as a resource management problem to *efficiently map VNs requests from SPs on an SN*. Measurability of such efficiency depends on the objective, typical objectives are maximizing the number of mapped VNs as in [6] or minimizing the resource provisioning cost on the SN as in [22], [2], and [7].

The problems are mostly addressed with Integer Linear Programming (ILP) formulations, a common method for optimization. This method requires an apriori knowledge of the requests and the models used don't address the mutual exclusion properties.

As none of the approaches referenced above offers a solution to the problem described in Section II we consider focusing on k-mutex algorithms.

Multiple algorithms [12], [23], [26], [17] have been proposed to address the classical mutual exclusion problem exposed by Dijkstra but focused on a single shared resource. Algorithms for classical mutual exclusion needs to enforce two properties:

- ***safety***: there is at most one process in the Critical Section
- ***liveness***: each request has to be satisfied within a finite time (under the hypothesis that the allocated resources are eventually released).

If the *liveness* properties is not satisfied, the system can ***deadlock***: multiple processes are left waiting indefinitely and never enter the CS. For systems with one and only one type of resource and with multiple resources of that type, they can be decomposed in two classes of problems.The first is known as the k-mutex problem. It is solved by algorithms such as the one proposed by Raymond et al. [19]. The second is when one process tries to allocate multiple resources of the same type called the k-out of-M resources allocation problem [21]. This requires that the algorithms satisfies the same two properties, *safety* and *liveness*, than the classical mutual exclusion problem. The definition for *liveness* is the same but the definition of the *safety* property is different: the number of resources which are allocated to the processes at any time is always less than or equal to M. (Each resource being allocated to only one process at a time).

For algorithms that address systems where there is one instance of M resources (also known as the dining and drinking philosophers problems) need to address that two requests can **conflict**. A conflict occurs when two request tries to allocate a common resource. The algorithms still need to satisfy the *safety* and *liveness* properties but with new definitions :

- *safety*: only one process can allocate the resource at any given time
- *liveness*: the basic property detailed above stay valid but we need an additional requirement *distinguishability*: when two requests conflict, it is always possible to define an order that satisfies both requests.

They also need to satisfy an additional property, ***concurrency***: if two requests don't conflict, they are not forbidden to enter their critical section simultaneously.

In systems with multiples resources of multiple types, the presence of a conflict between two requests is not sufficient to prevent their allocation as we have multiple resources of each type.

Bouabdallah and Laforest propose in [3] a distributed token-based algorithm for what they call the general resources allocation problem: at any moment, any process may request several resources of several resources types. This is similar to our problem. Their algorithm does not require an apriori knowledge of the requests but requires the locking of resources. Lejeune et al. [13] demonstrate that this is not efficient in terms of the usage rate of resources and introduce an algorithm for k-mutex that does not require a lock but only addresses systems with a single resource of each type. We propose in Section IV an extension of this algorithm to address our problem with systems with multiple resources of each type.

IV. Algorithms and their simulation

We developed a discrete event simulator to evaluate the performances of different algorithms. The simulator is a centralized scheduler of requests which are run against the simulated distributed system. The simulator is composed of Python 2.7 scripts. As an input it takes the parameters that characterize the system:

- The number C of types of resources
- The number N of requests emitted by the system at each time t, called *tick* in the simulator
- The maximum $max\_request\_size$ number of resources in a request
- the maximum time max_time during which resources are allocated. $\forall i, \alpha_i \leq max\_time$
- the duration t_{max} of the simulation

As a first step, we assume that k is the number of resources of each type.

At each tick t, N requests are simulated. Each request has a size calculated by a linear random function that is between 1 and $max\_request\_size$ and the resources are requested for a time between 1 and max_time that is also calculated by a linear random function.

A. No_Algorithm

A first algorithm, called *No_Algorithm* is showcased here to illustrate the behavior of the simulator. With this algorithm when a request is emitted by the system, if is satisfied if it can be satisfied. If it can't be satisfied the algorithm discards the request and increment a counter to count the number of requests that can not be satisfied.

One of the outputs of the simulator is the number of requests that cannot be satisfied as soon as they are emitted by the system. We run multiple tests, based on several system configurations and produce a graph with the resulting data shown in Figure 1. The results illustrate the degree of occurrence of

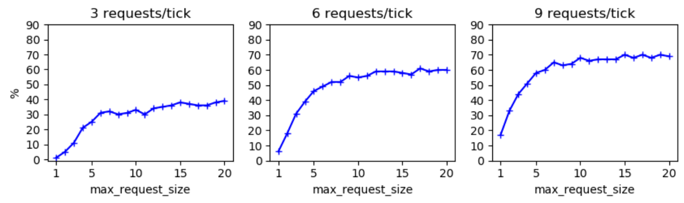

Fig. 1. Comparison of concurrency for systems where $C = 10$ with $k = 4$

requests that can not be allocated upon emission, according to three different system loads $N = 3$, $N = 6$ and $N = 9$ when $C = 10$ and $k = 4$:

- On X-axis the $max\_request\_size$ varies from 1 to 20,
- On Y-axis the degree (in %) of occurrence of resources that can not be satisfied as soon as they are emitted.

These graphics show that if the algorithm just discards the requests that can not be satisfied upon emission, we end up discarding a large part of them. More than 30% of them get discarded under a load of 3 requests per tick when the $max\_request\_size$ is around 7. More than 70% get discarded under heavier load in the test configuration.

B. Naive_Algorithm

The next algorithm is called *Naive_Algorithm*. This algorithm showcases that the properties defined in Section III need to be addressed otherwise the system eventually enters a deadlock and no request can be satisfied anymore.

The algorithm is similar to the No_Algorithm but when a request that can not be satisfied upon emission appear, instead of discarding the request, the algorithm locks all the available resources present in the request and appends the request to a *pending* array. After the request resolution, it tries to allocate the requests present in the *pending* array following the same algorithm: it satisfies the request if it can be, otherwise it will lock the available resources that are not already locked. This algorithm does not distinguish the resources and thus does not guarantee the *liveness* property.

C. Incremental_Algorithm

The next algorithm that is implemented in our simulator is a textbook distributed mutual exclusion algorithm that we call *Incremental_Algorithm*. It is based on the linear ordering of the resources to guarantee the *liveness* property defined in section III. Like the Naive_Algorithm is uses a locking mechanism on resources when a request cannot be satisfied upon emission. when there is only one resource of each type, the classical algorithm locks each resource type following the defined order. In this algorithm a request cannot lock a resource of a type class c_i if it also needs to lock a resource of a type c_j when $j < i$ where $<$ is the order function defined. To have the algorithm work when there are multiple resources of a same type, all the resources of that type are locked at the same time. If not enough resources are available to lock them all, then none are locked.

D. LASS_Algorithm

We finally describe the LASS algorithm. The name comes from Lejeune, Arantes, Sopena and Sens who described it in [13]. Contrary to No_Algorithm and Naive_Algorithm, it is not based on a locking function but computes vectors of counters for each resource. Each resource has a counter that is incremented at each request. The vectors define a total order on the requests and this order allows the algorithm to satisfy the *liveness* property. The algorithm has been described for systems with a single resource of each type. Our contribution here is to adapt it so that it can manage multiple resources of each types. Without modifications the algorithm does not satisfy the distinguishability requirement of the *liveness* property which definition is recalled in Section III. We propose to compute a vector per type of resources. We then get a total order of the requests by type of resources which satisfies the *liveness* property.

V. Experimental results

We use our simulator to run the algorithmspresented in Section IV on a system configuration with 40 resources from 10 different types. Each request spends in average 5 ticks in the CS.

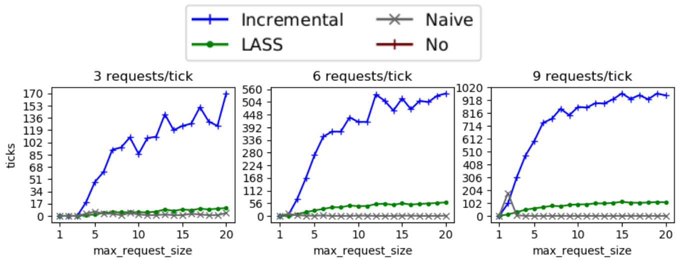

Fig. 2. Comparison of Average Delay, $C = 10$

We run tests with these two configurations and compare two metrics. As for Figure 1 the X-axis contains the values for $max\_request\_size$ which vary from from 1 to 20.

Figure 2 shows the values for the **Average Delay** on the Y-axis according to different loads. The Average Delay is the average time spent by requests between the times when they are emitted and when they are actually allocated. Figure 3 show the values, for the same tests, of the **Usage Rate**. The usage rate on the duraiton of the test is the sum over each tick of the rate of resources that are used compared to the total size of the system.

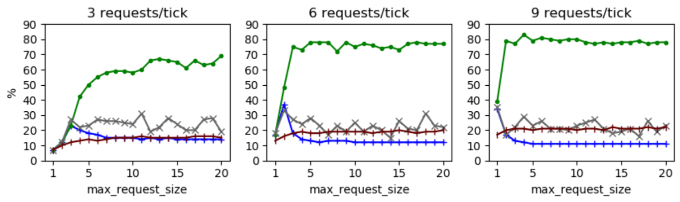

Fig. 3. Comparison of Usage Rate, $C = 10$

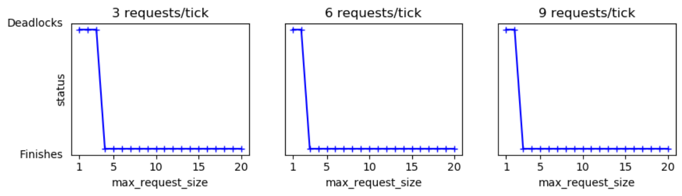

Fig. 4. Completion status of algorithm Naive_Algorithm, $C = 10$

No_Algorithm has no Average Delay as the requests are either allocated upon emission or discarded. Naive_Algorithm does nothing to guarantee the *liveness* property.Figure 4 shows that Naive_Algorithm deadlocks in most of the configuration tested as it does nothing to guarantee the *liveness* property, cf. IV. So its resultsare not significant.

The results show that the LASS_Algorithm outperforms No_Algorithm and Incremental_Algorithm for both metrics while guaranteeing the *liveness* property. No_Algorithm discards a large part of the requests, cf. Section IV. This explain why the usage rate 3 to 4 times worse than with LASS_Algorithm.The Usage Rate of LASS_Algorithm is also 3 to 4 times higher than the one of the Incremental_Algorithm. This can be explained by the fact that resources are not locked when they are not used. This is consistent with the results showcased in the original paper about the LASS_Algorithm [13] where the algorithm was applied to systems with only one resource of each type.

VI. Future works

The experimental results in Section V showcase that distributed mutual exclusion algorithms allow for an efficient allocation of resources in systems like 5G Networks slices with multiple types of resources (e.g. CPU, RAM, Disk, Network Bandwidth) and multiple resources available of a same type (e.g. in a situation where we have multiple servers available, or multi-CPU servers). They also showcase that it is important to carefully select an efficient algorithm according to the metrics that are important to the system as results can differ greatly between algorithms. We also shown that algorithms designed to guarantee the properties for mutual exclusion in system with one resource per type can be adapted to systems where there are multiple resources of each type.

Next step is to propose a new k-mutex algorithm that addresses additional constraints such as the localization of resources while satisfying the properties required by the system. We will then implement it in our simulator and if possible in an orchestrator where it can be tested against a running system.

References

[1] NGMN Alliance. NGMN 5G P1 Requirements & Architecture Work Stream End-to-End Architecture Description of Network Slicing Concept. https://www.ngmn.org/publications/technical-deliverables/page/3/, 2016-01-13.

[2] J. F. Botero, X. Hesselbach, M. Duelli, D. Schlosser, A. Fischer, and H. de Meer. Energy Efficient Virtual Network Embedding. 16(5):756–759, 2012-05.

[3] A. Bouabdallah and C. Laforest. A Distributed Token-based Algorithm for the Dynamic Resource Allocation Problem. 34(3):60–68, 2000-07.

[4] S. Bulgannawar and N. H. Vaidya. A distributed K-mutual exclusion algorithm. In *Proceedings of 15th International Conference on Distributed Computing Systems*, pages 153–160.

[5] K. M. Chandy and J. Misra. The Drinking Philosophers Problem. 6(4):632–646, 1984-10.

[6] N. M. M. K. Chowdhury, M. R. Rahman, and R. Boutaba. Virtual Network Embedding with Coordinated Node and Link Mapping. In *IEEE INFOCOM 2009*, pages 783–791, 2009-04.

[7] S. R. Chowdhury, R. Ahmed, M. M. A. Khan, N. Shahriar, R. Boutaba, J. Mitra, and F. Zeng. Protecting virtual networks with DRONE. In *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pages 78–86, 2016-04.

[8] E. W. Dijkstra. Hierarchical ordering of sequential processes. 1(2):115–138.

[9] E. W. Dijkstra. Solution of a Problem in Concurrent Programming Control. 8(9):569–, 1965-09.

[10] Brendan Jennings and Rolf Stadler. Resource Management in Clouds: Survey and Research Challenges. 23(3):567–619, 2015-07.

[11] R. Jhawar, V. Piuri, and P. Samarati. Supporting Security Requirements for Resource Management in Cloud Computing. In *2012 IEEE 15th International Conference on Computational Science and Engineering*, pages 170–177, 2012-12.

[12] Leslie Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. 21(7):558–565, 1978-07.

[13] J. Lejeune, L. Arantes, J. Sopena, and P. Sens. Reducing Synchronization Cost in Distributed Multi-resource Allocation Problem. In *2015 44th International Conference on Parallel Processing*, pages 540–549, 2015-09.

[14] F. Machida, Masahiro Kawato, and Y. Maeno. Redundant virtual machine placement for fault-tolerant consolidated server clusters. In *2010 IEEE Network Operations and Management Symposium - NOMS 2010*, pages 32–39, 2010-04.

[15] K. Makki, P. Banta, K. Been, N. Pissinou, and E. K. Park. A token based distributed k mutual exclusion algorithm. In *[1992] Proceedings of the Fourth IEEE Symposium on Parallel and Distributed Processing*, pages 408–411.

[16] K. Mills, J. Filliben, and C. Dabrowski. Comparing VM-Placement Algorithms for On-Demand Clouds. In *2011 IEEE Third International Conference on Cloud Computing Technology and Science*, pages 91–98, 2011-11.

[17] Mohamed Naimi, Michel Trehel, and André Arnold. A Log (N) Distributed Mutual Exclusion Algorithm Based on Path Reversal. 34(1):1–13, 1996-04-10.

[18] Anshul Rai, Ranjita Bhagwan, and Saikat Guha. Generalized Resource Allocation for the Cloud. In *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, pages 15:1–15:12. ACM, 2012.

[19] Kerry Raymond. A Tree-based Algorithm for Distributed Mutual Exclusion. 7(1):61–77, 1989-01.

[20] Kerry Raymond. A distributed algorithm for multiple entries to a critical section. 30(4):189–193, 1989-02-27.

[21] Michel Raynal. A distributed solution to the k-out of-M resources allocation problem. In *Advances in Computing and Information - ICCI '91*, pages 599–609. Springer, Berlin, Heidelberg, 1991-05-27.

[22] A. Razzaq and M. S. Rathore. An Approach towards Resource Efficient Virtual Network Embedding. In *2010 2nd International Conference on Evolving Internet*, pages 68–73, 2010-09.

[23] Glenn Ricart and Ashok K. Agrawala. An Optimal Algorithm for Mutual Exclusion in Computer Networks. 24(1):9–17, 1981-01.

[24] N. Sfika, A. Korfiati, C. Alexakos, S. Likothanassis, K. Daloukas, and P. Tsompanopoulou. Dynamic Cloud Resources Allocation on Multidomain/Multiphysics Problems. In *2015 3rd International Conference on Future Internet of Things and Cloud*, pages 31–37, 2015-08.

[25] SLICENET. Deliverables - slicenet. https://slicenet.eu/deliverables/, 2018.

[26] Ichiro Suzuki and Tadao Kasami. A Distributed Mutual Exclusion Algorithm. 3(4):344–349, 1985-11.

[27] A. Widjajarto, S. H. Supangkat, Y. S. Gondokaryono, and A. S. Prihatmanto. Cloud computing reference model: The modelling of service availability based on application profile and resource allocation. In *2012 International Conference on Cloud Computing and Social Networking (ICCCSN)*, pages 1–4, 2012-04.