
**Università degli Studi di Ferrara**

DOTTORATO DI RICERCA
IN SCIENZE DELL'INGEGNERIA

CICLO **XXXIII**
COORDINATRICE/COORDINATORE **Prof. TRILLO STEFANO**

# *TOWARD ELASTIC PARTITIONING OF MULTI-TENANT COMPUTING SYSTEMS AT THE EDGE*

Settore Scientifico Disciplinare **ING-INF/01**

Tutore
**Prof. Bertozzi Davide**

Dottoranda/o
**Dott. Turki Mariem**

**Anni 2017/2020**

# Abstract

Fog computing is gaining momentum to extend Cloud resources in close proximity to data sources, end users or both. Among the explored Fog deployment models, the Public Fog offers compute and memory resources for open use to IoT service providers, and is emerging as a fundamental component for an Edge-Fog-Cloud complete compute continuum along which IoT services can be flexibly instantiated.

The multi-tenant nature of public Fog nodes represents a major design and management challenge at the intersection of different yet related research disciplines, ranging from dynamic mapping of manycore architectures to resource management for Cloud and Fog resources, and from computing acceleration to software virtualization. The fundamental challenge is to efficiently share the limited pool of Fog resources among multiple consolidated IoT services sharing the same hardware platform.

This thesis revolves around the key intuition that multi-tenancy could be reconciled with limited resource capacity through an elastic provisioning of Fog resources. As a result, the thesis proposes **a holistic support for elastic Fog computing**, following a bottom-up methodology.

The support is fundamentally rooted in the capability of the on-chip interconnection network (network-on-chip, NoC) to spatially and temporally isolate communication flows originated by different IoT services. The isolation in space enables to partition the Fog node architecture into spatially-isolated execution environments that provide enhanced security with respect to software-only isolation, and the strictest notion of service composability. At this level, the thesis proposes pLBDR, a lightweight routing mechanism that prevents functional and non-functional interference of intra-partition communication flows with one another. Above all, it combines low complexity with fast dynamic reconfigurability of the partitioning pattern, thus delivering a NoC-supported elastic partitioning in space that is out-of-reach for current NoC technology.

In space-multiplexed parallel computing architectures, some communications unavoidably break the spatial locality, especially those associated with memory controller and system configuration traffic. For these flows, this thesis provides efficient time-multiplexing while meeting the distinctive requirements of an elastic Fog environment: low-latency communication scheduling in time, and runtime reconfigurability of the number of time slots. This new set of requirements make the proposed time-multiplexed NoC a unique design point in the open literature.

In compliance with its bottom-up approach, the thesis finally tackles the resource management challenge to master the elasticity properties of the underlying compute and memory partitions. In line with mainstream approaches to resource management for manycore systems, the thesis assumes a hierarchical framework where virtual resource reassignments are dynamically changed into the actual reallocation of physical resources. At this level, the shape, size and location of space partitions have to be adjusted in a non-overlapping way to fulfil the variations. The thesis proposes an Integer Linear Programming shape-based

model that strives to deliver prioritized latency guarantees to IoT services while perturbing the system state the least possible. The modest running times enable the deployment of the proposed Partition Manager for online use, in combination with a "prior provisioning prompt allocation" scheme for resource utilization in Fog computing.

Overall, this thesis is a highly interdisciplinary piece of work that provides an integrated hardware/software support for elastic Fog computing, and paves the way for a dynamically-orchestrated Edge-Fog-Cloud continuum serving as a seamless hosting environment for the next generation of smart IoT services.

**Keywords:** Internet of Things (IoT), Fog computing, Network-On-Chip (NoC), Routing mechanism, Elastic Computing, Cloud continuum, Time-Division-Multiplexing (TDM), Space-Division-Multiplexing (SDM), Resource Management, Security, Strong Isolation, Real-Time applications, Dynamic reconfiguration.

# Contents

# List of Figures

# List of Tables

# Introduction

All along the past years, there has been a tremendous increase in the popularity of the Internet-of-Things (IoT) to become one of the most utilized innovative technologies at present. It has pervaded almost every industry, from health care to automotive. Despite the facilities and benefits IoT has provided, the industry is facing abundant obstacles in its adoption and implementation.
In fact, the early IoT scenarios were doing all of their processing in the Cloud using Big Data methodologies and tools, and returning the results to the interested users.
Current advances in IoT technologies are making an unprecedented number of immersive applications available at the infrastructure Edge, featuring unmatched computing power and storage capacity requirements with respect to the old generation of IoT services [131, 148].
As a result, novel computation paradigms are emerging from the convergence of IoT, wireless sensor networks, mobile computing, Edge computing and Cloud computing. Among them, Fog computing is having the most rapid growth, due to its capability to bring computing paradigms from the Cloud to the Edge and enable sophisticated data processing in closer proximity to end users [25, 28, 90].
Populating the current gap between Smart Objects at the infrastructure Edge and datacenter-powered Cloud with emerging Fog computing nodes, and enabling their tight interplay, is a promising approach to set up a novel "compute continuum" along which the next generation of IoT services could be flexibly instantiated [117, 133]. However, the implementation of efficient Fog computing platforms represents a challenging task. The main challenges include the following:

- First and foremost, this is still a constrained execution environment, that needs to cope with a massive amount of data with scarce computational, bandwidth and memory resources with respect to the Cloud counterpart [147].

- Second, the criticality of the capacity limitations is amplified by the shared (i.e., multi-tenant) nature of most Fog nodes, which enables different IoT service providers to amortize the infrastructure cost while still taking advantage of more powerful devices than their in-house built/developed IoT devices [147].

- Third, IoT applications are context-sensitive [65, 116] and exhibit fluctuating workloads over time. Not only the execution context changes continuously [101], but also the business logic of the application and the pricing model of the Fog node [43], in addition

to user preferences. As a result, Fog nodes should allow for the flexible provisioning of resources to applications over time [127].

- Last but not least, security is a primary design goal for Fog computing [150], where services should be strongly isolated from one another.

State-of-the-art hardware/software platforms for Fog computing are not keeping up with these emerging requirements. Common implementations rely on commercial off-the-shelf parallel computing architectures, rely on software-driven virtualization without any hardware extensions for efficient and secure multi-tenancy, and are typically mastered through statically-defined resource management policies.

This thesis revolves around spatial partitioning of manycore architectures as an effective solution to deliver secure-grade isolation of IoT services and platform composability. It is in fact well-known that the most promising mapping of a workload to a parallel computing architecture consists of reserving a spatial region of contiguous computing and memory resources. By combining this with the isolation of such spatial "partitions" (e.g., through a strict control over partition boundaries in hardware), the isolated execution environment allows IoT service providers to develop their applications and to verify their timing properties in isolation, since the final consolidation in the target platform with an unknown and runtime-varying number of sharers will not have any functional and non-functional interference over its operation. At the same time, segregated spatial partitions are easier to protect against security threats [95], including denial-of-service attacks or even the formation of timing channels [124].
The thesis takes its steps from the observation that:
(i) no state-of-the-art COTS platform exhibits hardware support for spatial-division multiplexing of its manycore resources, as a key enabler for efficient multi-tenancy;
(ii) no state-of-the-art platform is optimized for the frequent dynamic reconfiguration of resource allocation in a multi-tenant platform (resource elasticity).
(ii) no state-of-the-art resource manager explicitly focuses on the control of spatial partition attributes, such as shape, size and location, thus failing to deliver contiguity, compactness, high execution performance and control over system state perturbation in the presence of a dynamically reconfigurable system.

The key technical intuition behind this thesis consists of identifying the pivotal role that the routing mechanism of the on-chip interconnection network (NoC) plays with respect to the fulfilment of the efficient, flexible and secure partitioning requirements of manycore processors/accelerators. In fact, the secure-grade isolation of spatial partitions of parallel compute and memory resources depends ultimately on the capability to constrain routing paths of communication flows on the NoC within the spatial partition boundaries.

A related challenge consists of flexibly reconfiguring the partitioning pattern without having to adapt the NoC routing framework each time to contain communications within the newly defined partition shapes.

The solution proposed by this thesis consists of *Partition-Enabling Logic-Based Distributed Routing (pLBDR)*, a lightweight routing mechanism for SDM-capable NoCs that combines implementation efficiency with flexibility.

pLBDR guarantees routability of spatial partitions leveraging a unique, unaffected global routing function. Thus, as the partition configuration needs to be modified at runtime, only the new partition boundaries need to be reprogrammed in the routing mechanism (limited to just a single 4-bit register per switch), while avoiding any routing path management overhead in software and in hardware. For the sake of fast reconfigurability, the proposed mechanism poses some restrictions to the irregularity of legal partition shapes, while still enabling those shapes that can deliver internal high-performance communications.

However, not all communications in a multi-tenant platform exhibit spatial locality, such as the routing paths connecting spatial partitions to the memory controllers, which would end up crossing the spatial extension of intermediate partitions and breaking the composability and isolation properties of the system as a whole. Therefore, temporal multiplexing (i.e., isolation) of non-local communication flows from/to different partitions is also required. Indeed, many distributed operating systems for manycore processors envision partitioning of the resources in both **space** and **time**, thus inspiring the approach of this thesis to equip Fog computing nodes with both space- and time-division multiplexing of chip-scale communications. Our solution is based on a dual Network-On-Chip concept. The first networking layer serves intra-partition traffic without mutual interference between spatial partitions, and uses the pLBDR routing framework to route packets. The second layer serves non-local traffic by scheduling domains on the network over time using time-division multiplexing (TDM). More precisely, we build on the concept of *Token-based TDM*, which takes advantage of the properties of the network Channel Dependency Graph (CDG) defined by the topology and the routing algorithm. From it, we derive the generic requirements to have all input ports of all routers serving only packets from the same domain at each time slot. This would allow contention between packets from the same domain only, while at the same time delivering secure-grade isolation between domains.
The approach is generalized to an arbitrary number of domains by selectively and deterministically placing propagation stops at specific points in the NoC, which preserves the strong isolation property and delivers more latency-efficient NoC communications than state-of-the-art of solutions. Finally, we provide support for runtime modifications of the system configuration through a flexible architecture that does not require substantial changes to support the new configuration, but rather modifies the scheduling commands distributed to the network switches through a token-based notification mechanism.
With respect to state-of-the-art, our work provides a flexible architecture from the ground up, and extends performance and implementation efficiency to the whole configuration space.

Our Dual-NoC solution ensures that concurrent IoT services running onto the same Fog computing node can be allocated secure-grade, isolated and flexible compute and memory partitions. But those allocated resources may be then remodulated over time depending on

application load and execution context variations. Along this direction, IoT providers may update their service models to demand resources based on a "use-just-what-you-need" basis, and the Fog resource management framework may have to cope with conflicting requests following decision policies inspired by priorities and pricing schemes.
The support for the illustrated "resource-elastic" computing paradigm in Fog platforms raises new challenges for traditional resource management schemes:

- First, the underlying hardware platform model needs to be revised to formulate the resource allocation problem. In fact, in manycore processors the applications are typically assigned virtual resources, not physical ones, since the focus on "how many" resources to provide is currently dominating over the definition of "which ones" to provide. The mapping of virtual resources to physical ones is typically overlooked, that is, considered as an implementation detail or, in the most accurate papers, as the focus of future work. Indeed, in an elastic computing environment brokering resources among consolidated services, identifying "which" exact resources to assign and/or revoke is of the utmost important, since it affects application latency and the amount of task migrations needed to evolve the system from one configuration to another.

- Second, the resource management framework becomes multi-layer: a top-level Resource Manager reconciles potentially conflicting resource allocation demands leveraging an abstract view of both application requests and available (virtual) resources, while a lower-level Partition Manager enforces the final allocation determining the new shape, size and location of concrete physical space partitions, as well as their non-overlapped composition on the computation grid.

The latter is a new optimization problem, and is the focus of the last part of our research work. We propose an innovative management framework for elastic partitioning of manycore accelerators/processors for Fog computing platforms under dynamic operational behavior. Based on predefined demands for more or less resources for the consolidated services harmonized by application-level managers, our partition manager fulfils the demands through the readjustment of partition shapes, size and location, in a way that trades optimality of the execution state with the reconfiguration overhead. On the one hand, the solver minimizes the number of task migrations of those services that are "neutral" (i.e., do not have a demand, or are not selected for resource revocation) during a reconfiguration event. On the other hand, it aims at correlating the proximity of a partition/service to the memory controller with its performance criticality.

Overall, the design methods and tools delivered by this thesis, that address the "elastic computing" paradigm for Fog nodes at several abstraction layers, are only the first steps toward materializing a novel Things-Fog-Cloud compute continuum. More research is needed to fulfil this vision, which represents an ambitious target for the future requiring the work of many PhD students.
The specific milestone achieved by this thesis consists of engineering a dual-NoC architecture

for Fog nodes enabling the flexible partitioning of its compute and memory resources, and of a resource management framework capable of mastering the exposed flexibility. As such, the thesis presents contributions that are logically synergistic and meant to work in a vertically-integrated way.

## Thesis Organization

The thesis is composed of six technical chapters, in addition to the introduction, conclusions and future work.

**Chapter 1** defines the vision for a new Things-Cloud compute continuum, serving as the ecosystem for the next-generation Internet of Things (IoT) services.

**Chapter 2** reviews the state of the art by giving a background about Elastic systems and Fog management. Also, it reveals the limitations of existing solutions to deliver secure and composable multi-tenancy in emerging Fog nodes, and to manage it.

**Chapter 3** illustrates the target architecture.

**Chapter 4** presents our proposed Time-Division-Multiplexing (TDM) NoC for strong isolation and low-latency.

**Chapter 5** explores the innovative routing framework for flexible space-division multiplexing.

**Chapter 6** reports on the implementation of a resource management framework to deal with the elasticity and dynamism of emerging IoT applications consolidated on the same Fog node.

In closing, we summarize the thesis outcomes, provide final remarks on the presented research works, and envision possible future works.

# Chapter 1

# New System Concept and Contribution of the Thesis

Internet of Things (IoTs) represents a notable first pace towards the networked society. It provides gigantic social and economical opportunities while at the same time it raises significant challenges regarding the infrastructure underpinning it and the deluge of data generated from it. This chapter explores these challenges and presents new system concepts capable to integrate nowadays IoT services.

## 1.1 Basic Definitions

This section introduces the terminology and concepts related to the three components of the emerging IoT-Fog-Cloud ecosystem.

- Internet of Things (IoT): the term IoT encloses all physical objects "things" that are connected to the internet, collecting and sharing data. Next, we present two computing paradigms that can be utilized together to fulfill the heterogeneous requirements associated with IoT applications: cloud and fog computing.

- Internet of Things (IoT) services/applications: IoT services/applications promise to bring immense value into our lives. With newer wireless networks, superior sensors and revolutionary computing capabilities, the Internet of Things could be the next frontier in the race for its share of the wallet. IoT applications are expected to equip billions of everyday objects with connectivity and intelligence. The fields where the most important IoT applications have been developed are: Wearables, Smart Home Applications, Health Care, Smart Cities, Agriculture and Industrial Automation.

- Cloud Computing: is a model which enables the on-demand availability of a shared pool of configurable computer system resources (e.g. services, storage, servers, networks, and applications) with minimal management effort by the user. However, in light of the proliferation of IoT connected devices, a huge volume of data is being generated

rapidly. Thus, data management, computation and storage becomes laboured in the cloud. This has led to the emergence of fog computing.

- Fog Computing: is a layered model that extends services offered by the cloud to the edge of the network (e.g. routers, switches, etc..) to run information-processing services on top of relatively powerful nodes in proximity of either raw data sources, information consumers, or both [28, 90]. Thus, ameliorates efficiency and minimizes the distance across the network by reducing the amount of data needed to transport to the cloud for computation, storage, and management.

## 1.2 Motivation: Cloud-Things Continuum challenges

Today we have several levels of computing ranging from IoT sensors and smart objects all the way to the Cloud datacenter infrastructure through IoT gateways and networks of private Fog nodes. The problem is that such layers are loosely coupled today, which stymies innovation in IoT services and limits the creation of an ecosystem for the large scale adoption of IoT technologies.

Clearly, an effort should be made to couple such layers more tightly together. The goal of a tight integration of heterogeneous computing layers and devices all along the data path is currently giving rise to an unprecedented effort in industry and academia to materialize a complete and innovative distributed «computing continuum».

Historically, large-scale processing for early IoT services was taking place in the Cloud, thus leading to a strongly polarized Edge-Cloud integrated system. However, as massive amounts of data (so-called Big Data) are being generated in unusual variety and volumes, and need to be processed under tighter latency, security and cost requirements, the main goal is to enable Big Data analytics in closer proximity to the raw data sources or to information consumers. This has raised a surge of interest in power-efficient computing architectures for deployment at the infrastructure Edge, especially coping with the computation horsepower required by artificial intelligence under tight resource and power budgets. Nonetheless, there is still a large gap in the Things-Cloud continuum that computing acceleration efforts are not able to bridge, as illustrated in Fig. 1.1.

On the one hand, there are private Edge infrastructures, hosting low latency and isolated services, with no resource sharing, high Capital Expenditure (CAPEX), time-to-market, and Total Cost of Ownership (TCO). Despite the progress in power-efficient computing, there is still a mismatch between the computational requirements of thorough Big Data analytics and the capability offered by such infrastructures.

On the other hand, the public Cloud infrastructure comes with the opposite characteristics. At the time of this writing, there is no consolidated hardware/software platform capable of bridging this fundamental gap, nor it is clear which player will effectively bridge it in the future.

Fog computing is a fast growing technology that might play a fundamental role to bridge

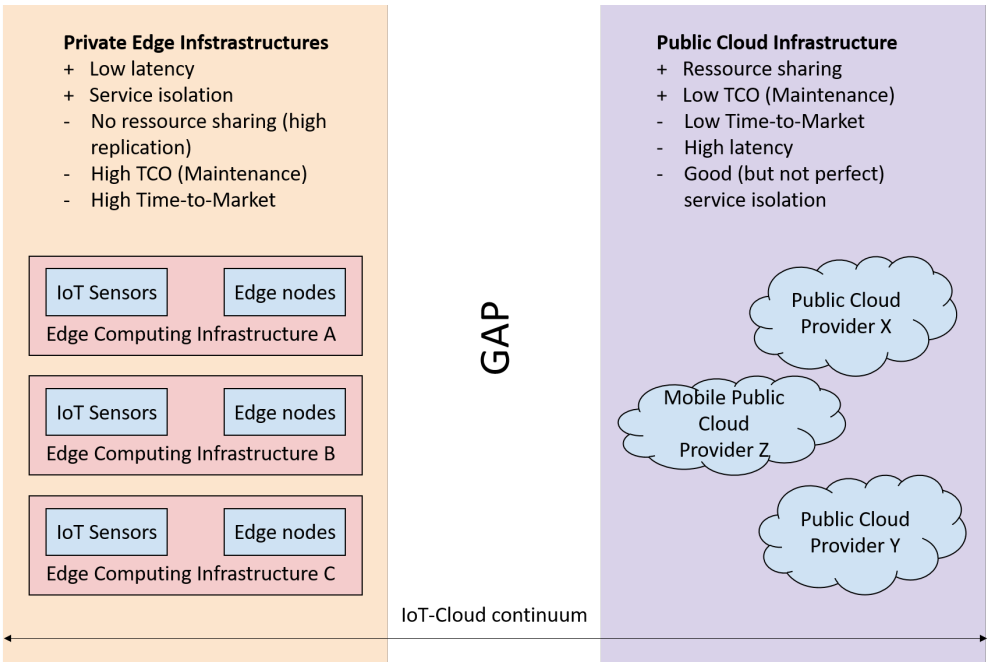



**Figure 1.1:** Current gap in the IoT-Cloud continuum.

this gap. Its purpose is to bring Cloud-like computing capability to the Infrastructure Edge, which is a much more constrained execution environment. The concept has been around for many years but it is still lacking of a concrete large-scale implementation. For this reason, developers have not spent too much time so far in developing applications capable of taking full advantage of the Fog paradigm. In turn, this is slowing down further progress towards concrete realizations of the paradigm, since they have to be customized for the applications that will make use of them. Ultimately, there is a vicious circle in Fog computing that calls for further research investments to unfold its potential and enable an effective Things-Fog-Cloud continuum. This thesis is a contribution to break this circle.

The main challenge to be tackled consists of matching high computational requirements with capacity-constrained hardware platforms (although more equipped than smart sensor nodes). Moreover, there are functional challenges, consisting of the computing orchestration throughout the devices of the compute continuum, and non-functional challenges including cost, power, reliability, security and privacy.

At the same time, current hardware/software platforms are not well suited for Fog and Edge computing, due to the relatively static policies for resource allocation. In fact, the main characteristics of an integrated Edge/Fog Computing environment are the relative resource shortage as well as the IoT workload dynamicity, which call for innovative computing and service models revolving around flexible dynamic resource allocation strategies.

In a nutshell, Fog nodes could bridge a missing gap between smart objects and Cloud computing toward a complete and innovative "computing continuum", provided relevant functional and non-functional challenges are addressed by research. This thesis is a contribution in this direction.

## 1.3 Concepts for a new computing continuum

The first goal of this thesis was to work out guiding principles for the Edge-Fog-Cloud computing continuum, serving as the context for the technical research activities presented later. In particular, the following concepts contribute to deliver a vision of such continuum, and were elaborated through fruitful brainstorming sessions with other groups at University of Ferrara (distributed systems, operating research), with other italian universities (Politecnico di Milano, Università di Modena/Reggio Emilia, Università di Messina), with european academic partners (EPFL Lausanne, Universidad Politecnica de Valencia) and industrial players (Leonardo, Camlin, Everis, Sysgo, Dunavnet):

- A compromise must be established between the increased resource requirements of future IoT services and the cost considerations of underlying execution platforms. This naturally tends to create intermediate levels of hierarchy along the Cloud-Things computing continuum.

- A Fog node will be necessarily shared among many IoT services, at least for public Fog nodes. Thus, the adoption of fine-grain resource allocation strategies and high frequency pricing schemes at the infrastructure Edge become mandatory to make an effective use of resources, and for their efficient sharing.

- Since Fog nodes are placed in closer proximity to IoT devices, context awareness should be a fundamental design and management concern. Exploiting context information enables improved services, reduces their cost for resource utilization and leads to efficient resource sharing. For this reason, the runtime optimization of the deployment of the so-called application topologies cannot be statically defined any more, based for instance only on the data traffic analysis between the tasks of the topology and their computational complexity. First of all, most IoT applications have fluctuating workloads. Also, most end nodes will be mobile, thus changing the association of processing workloads to Fog nodes over time, and causing time-dependent congestion on such nodes. As a result, context-aware computing will play a fundamental role in determining which data to process, its computational requirements, its criticality for the end user and where to process it along the Cloud-Things continuum.

- Management strategies must be employed to solve any possible conflict when applications are consolidated onto the same underlying hardware platform, harmonizing their potentially conflicting requirements. Compute and memory resources will have to be dynamically split into virtual platforms whose resource allocation depends on application goals and platform state.

- Services' demand for resource allocation do not depend only on the context and/or platform state, but also on the business logic of the application and the user preferences. For this, quality-of-execution requirements (response time, latency, accuracy,

throughput, etc.) need to be traded off with the cost (data transfer, virtual resource usage, storage or monitoring cost). Not only the execution context changes continuously, but also the business/pricing models and the user preferences may evolve throughout the lifecycle of the service/underlying platforms.

- For the applications whose deployment topology changes over time, a flexible provisioning should be employed due to the offload of components from/to Edge from/to the cloud. To achieve this, we envision a global Service Supervisor for each service as a whole, capable of mapping and remapping service components all along the Edge-Cloud continuum, interacting with a local Resource Manager for the Fog platform negotiating access to local resources.

- For the sake of managing and instantiating service components in the Cloud or on the Fog devices, a Global Orchestrator running in the Cloud will coordinate between these services. For each service, the Supervisor component will notify the Global Orchestrator of the desired location for the instantiation or the migration of service components. The Global Orchestrator will thus coordinate with the Cloud and Fog devices to translate the requests into a concrete course of mapping or remapping action.

- In order to allow the runtime synthesis of IoT services, dynamic resource allocation is needed. However, the lack of unified standards and the coexistence of open and proprietary solutions currently limit device interoperability. As a result, participating devices will need to have predefined interfaces, accessible through standardized APIs.

- Service components semantics have to be identified in such a way that the system can automate the tasking and reasoning process without the need for hardwired configurations and service specifications. The semantics will enable intelligent response to the business requirements and policies enabling the hardware and software components to understand run-time requirements and execute the corresponding service strategies.

- Since IoT services will move (part of) their execution from the cloud to the Fog, security becomes one of the most challenging properties that has to be satisfied. Thus, the need for computing platforms ensuring strongly isolated execution environments for the running services will be a priority. Basically, the requirement is to implement a trusted compute base for the "elastic" and migratable applications to deploy and run. In addition, many threats will arise from platform sharing. In fact, resource dependencies and synchronization between the running services are exposed during runtime, which raises the need for secure communication channels (both on-chip and off-chip), authentication and authorization methods.

- In Fog node architectures, data privacy is essential. To ensure it, direct and indirect accesses to the data of the different applications that share the resources of the same Fog node must be forbidden whether these data are stored or processed on the main

host processor or in an companion accelerator. In addition, the owner/operator of the Fog node must not be able to access the application data. As a solution, we could use an hypervisor-based isolation system, coupled with hardware mechanisms to ensure isolation at the accelerator side and to ensure the authenticity of the hypervisor itself.

The concepts of this vision are pictorially illustrated in Fig.1.2, where they are combined to draft the scheme for a novel Things-Fog-Cloud compute continuum.

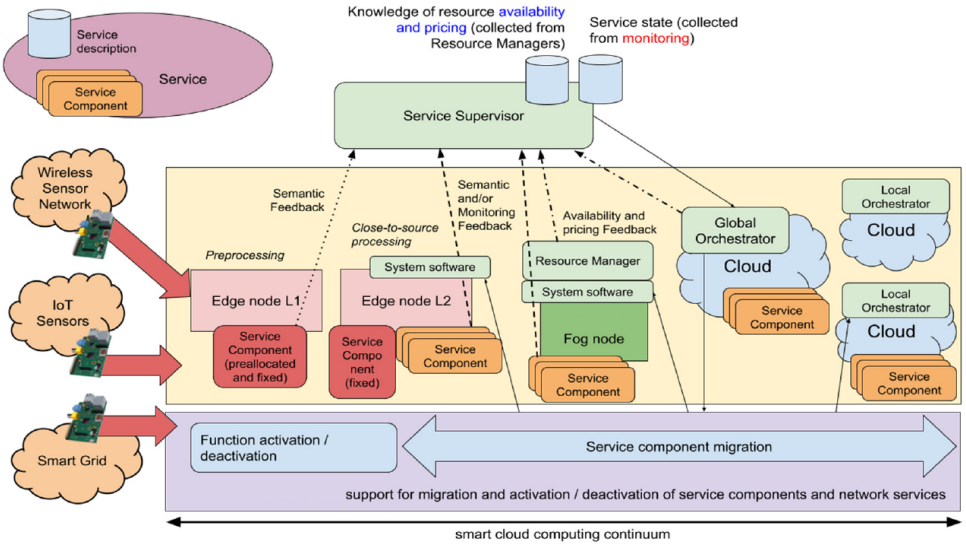



**Figure 1.2:** Vision for a novel Things-Fog-Cloud continuum.

The platform will be capable of executing service components on different locations: in the Cloud, on Fog Nodes, and on COTS Edge devices. Federated Cloud environments could be part of the picture, allowing to build flexible software platforms that leverage multiple Cloud providers. At the Edge level, the vision includes 2 different types of COTS devices: level 1 (L1) devices that have limited resources and can support dynamic workloads only through the activation/deactivation of pre-installed software components, and level 2 (L2) devices that have more resources and can implement dynamic workloads also through the migration of service components to/from the Fog or the Cloud.
At the service management level, a multi-layer architecture is envisioned. Each Fog service has an associated Service Supervisor: a software component that is in charge of instantiating the Fog service, of monitoring the performance of each service component, and of overseeing resource allocations/remodulations both at the entire service and at the single service component levels. Service Supervisor interfaces with the Resource Manager (RM) component running on each Fog node to have an always up-to-date knowledge of the corresponding local resource availability and pricing scheme (which will typically change according to resource availability). At the same time, Service Supervisor interfaces with service components to monitor their performance, according to both technical (response times, resource consumption, etc.) as well as business-level (revenue, OPEX, etc.) key performance indicators. Leveraging the knowledge of resource availability and pricing and service performance, as

well as an interoperable service description provided by developers, Service Supervisor will control resource allocation and remodulation. Service component migration requests are fulfilled by a Global Orchestrator component, which interacts with System Software components running on each Fog node and L2 Edge nodes, as well as with Local Orchestrator components running on other Cloud platforms, connected to the platform in order to perform migration of containers corresponding to the interested service components.

## 1.4 Components for a novel compute continuum

From an implementation viewpoint, we envision a framework that revolves around the following fundamental new components with respect to state-of-the-art:

- **A shared multi-tenant Fog node:** it ensures a secure and composable compute environment for the applications that share the very same Fog node and run on top of it. *This thesis will contribute interconnect-centric hardware design methods for multi-tenant Fog nodes.*
- **Elastic allocation of Fog resources to elastic IoT services:** instead of allocating resources on the Fog node for the worst-case, IoT services will contribute to efficient resource sharing by demanding resources based on a "use-just-what-you-need" paradigm. This way, resources will be dynamically assigned to those IoT services that need them the most at any given point in time. *This thesis will specialize the context of elastic computing for a public Fog node serving low-latency and user-centric applications.*
- **Dynamic resource manager in the cloud-Fog-Things continuum:** it will ensure a hierarchical resource management framework structured into communicating layers. The top layer will take care of dynamic topology deployment of IoT services by (re-)assigning service components all along the data path of the continuum. At the bottom layer, the manager will master the elasticity of Fog resource allocation to IoT services on the local Fog node. *This thesis will focus on the bottom management layer and will deliver a hardware-dependent resource manager mapping virtual resource assignments in a Fog node into the concrete readjustment of physical resources.*

### 1.4.1 A shared Fog node

A multi-tenant Fog platform shared by IoT services represents the innovative hardware component of the proposed vision. It serves as a connecting platform across the Edge-Cloud continuum by bringing Cloud-like computations and elastic resource provisioning capability at the Edge. We envision an architecture revolving around a high performance host multi-core processor together with a dynamically reconfigurable array fabric of homogeneous or heterogeneous accelerator cores. The fundamental relevance of this Fog node is to serve as the first-level computing platform sharing among IoT services across the Things-Fog-Cloud

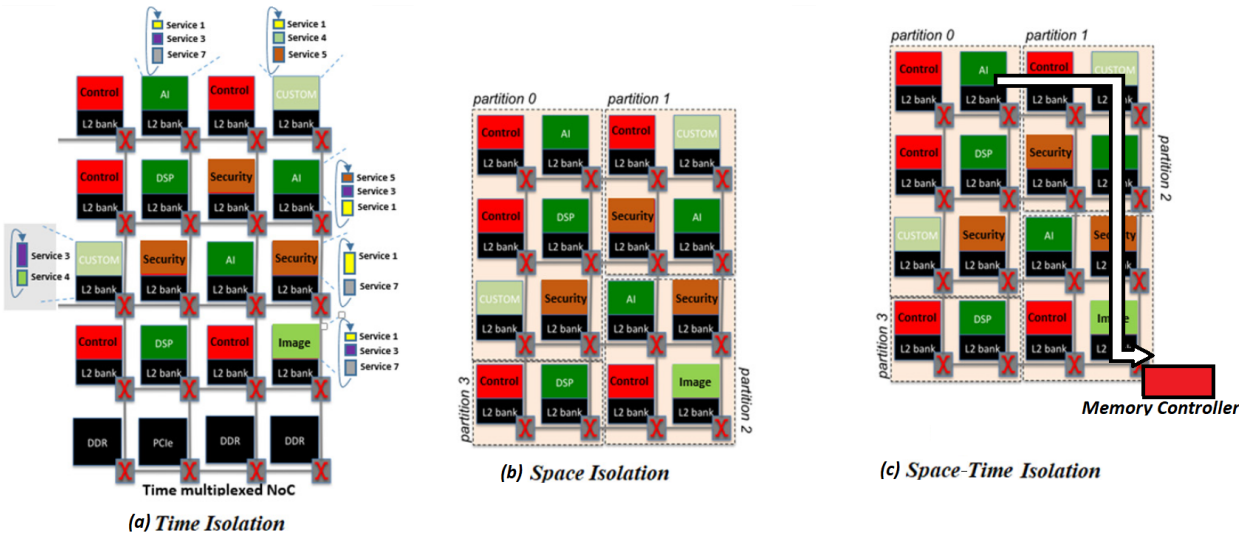



**Figure 1.3:** Platform partitioning strategies put at work on an array fabric of heterogeneous processing elements or accelerators.

continuum.

In order to ensure composability and security in a shared multi-tenant systems, a common solution is to divide the system into a set of domains. Each domain represents an isolated environment along the time or space dimensions, where an IoT service can be kept separated from the other ones and run with no mutual interference.

Time division multiplexing (TDM) will be applied to those accelerators of the Fog hardware platform that lack spatial parallelism, and over which IoT services can be time-sliced over time (Fig.1.3a). Some accelerators expose spatial parallelism. In this case, an hypervisor would be enforcing segregation of IoT services into isolated and contiguous spatial regions. Alternatively, spatial partitioning could be used to virtualize an array of heterogeneous accelerators, and assign each virtual partition to a different IoT service (Fig.1.3b). In both cases, routing control of the underlying on-chip interconnection network will be of fundamental importance to avoid any kind of mutual interference between traffic flows logically pertaining to different regions. In another scenario, Interestingly, whenever spatial partitioning is inferred, there exists communication flows that break the spatial locality, such as communications from one partition to the memory controller, and vice versa (Fig.1.3c). In this case, time-division multiplexing of the locality-breaking flows could be a solution not to interfere with the spatial partitions crossed by the non-local traffic flow, and to preserve system-wide communication isolation.

### 1.4.2 Elastic allocation of resources

Due to the fact that IoT applications are very sensitive and dependent to the workload, the execution context and the surrounding environment, our system vision assumes elastic computing principles in a multi-dimensional space (resource, quality and cost elasticity).

As already mentioned above, the hardware platform will adopt resource allocation elastic-

ity and will allow services to dynamically get and release resources at a significantly finer granularity than what happens in the Cloud today (i.e., at the level of individual processing elements, rather than of complete virtual machines), and in a more cost-sensitive environment. Furthermore, quality elasticity will be implemented through the use of efficient policies able to dynamically trade-off user experience with the demand for Fog resources. Last but not least, Fog providers will use pricing policies to shape the peak requests for resources from IoT services in the most non-overlapping way possible (cost elasticity).
The elasticity principles should be generalized at each layer of the design hierarchy for the sake of vertical integration and efficient operation, from the service model to the decision making layer, and from the runtime support to the dynamic hardware platform. Along this direction, service models should be extended to explicitly cope with a variable amount of underlying resources.
Overall, elastic IoT services, matched by elastic resource allocation in the hardware platform, will efficiently contribute to bridge the gap between the growing amount of data to be processed in IoTs and the scarcity of compute and memory resources at the infrastructure Edge.

### 1.4.3 Dynamic resource management

Elasticity can only be mastered through a systematic and coherent support for dynamic resource management at several layers. In fact, to enable the dynamic creation of workload/state/environment-adaptive deployment topologies for Cloud IoT services, the system will have to satisfy the following requirements:
- At the Fog node level, it will allow efficient elastic resource allocation through dynamic resource allocation of homogeneous or heterogeneous parallel computing architectures.
- It will make dynamic topology deployment operational at runtime, by moving service components from the Fog to the Cloud or vice versa, depending on workload phase, execution context on the different platforms, on the pricing schemes and on the business model. Along the same direction, the Service Supervisor will consider relevant non-functional requirements such as power budgets or thermal control issues.
The way to fulfill these objectives will be to build on top of three interactive and fully-integrated concepts: local composability of services, distributed service topology management and the employment of a centralized Orchestrator in the Cloud.
At the level of individual Fog nodes, a dynamic resource allocation policy will broker resources among consolidated IoT services. In turn, each service will have a dedicated software component called Service Supervisor, interacting with the Global Orchestrator to map or remap service components all along the Edge-Fog-Cloud continuum. When a service is instantiated or needs to be migrated for dynamic topology deployment, the Service Supervisor will notify the Global Orchestrator about the desired location; this latter will coordinate with the Cloud and Fog devices to execute the command.

## 1.5 Contributions of the thesis

The proposed vision serves as the reference context for the technical contributions of this thesis, that will be the first milestones on the way of a concrete implementation of the vision in industrial hardware and software platforms.

In short, the thesis focuses on emerging public multi-tenant Fog nodes, for which a twofold set of contributions is pursued:

- Hardware support for the partitioning of accelerator resources in space and time. The initial focus will be on an array fabric of homogeneous processor cores for a Fog environment, whose computing and memory resources will be multiplexed in space among IoT services, and whose interconnection network will be multiplexed in space and time depending on the spatial locality of the specific communication flow.

- Hardware support for the dynamic re-partitioning of the manycore architecture. The routing mechanism of the on-chip interconnection network will be demonstrated to be a fundamental enabler for the fast repartitioning of the system, that is, for dynamically changing the shape and size of spatial compute and memory partitions. As a result, the thesis will proposes a lightweight and flexible routing mechanism cutting down on the reconfiguration cost with respect to state-of-the-art routing frameworks.

- Techniques for hardware-dependent resource management of a manycore accelerator. Once hardware support has been provided for dynamic and flexible partitioning, a resource manager will be set up for managing the elasticity of the partitions. Focus will be on the resource management layer closer to the actual computing fabric, where physical resources are assigned, not virtual ones. The proposed resource management framework will then perform online mapping of virtual partitions onto physical ones, and work out the readjustment of the latter whenever a higher-level application manager demands more or less resources for virtual partitions in a placement-agnostic way.

Beyond the focus itself on emerging computing platforms, the key novel aspects of the research reported in this thesis are as follows:

- Time-division multiplexing of communication flows will be targeted to the low-latency and reconfigurability requirements of a Fog node. In practice, once packets are admitted into the network, they will flow uninterrupted all the way to the intended destination in a conflict-free way. At the same time, the proposed architecture relies on the propagation of scheduling commands to the network switches, thus lending itself to the dynamic reconfiguration of the number of time slots over time. The scheduling solution of this thesis is inspired by the Channel Dependency Graph (CDG) of the routing algorithm on top of the considered network-on-chip topology, and as such is original.

- The NoC routing mechanism for space partitioning will extend a lightweight logic-based distributed routing (LBDR) mechanism. The latter is far more efficient than mainstream routing tables in terms of routing computation delay, area and power overhead, and scalability. However, LBDR trades resource-/performance-/area-efficiency with flexibility. The LBDR extension proposed in this thesis enables to reconfigure the partitioning pattern in space of a manycore accelerator without having to reconfigure the routing function. Therefore, the reconfiguration process is extremely fast, since only partition boundaries need to be programmed in the routing mechanism, for the sake of traffic isolation, predictability and composability.

- Traditional dynamic resource management frameworks for manycores focus on the assignment of virtual resources to consolidated applications. The claim of previous work is that determining "how many" resources to assign is far more important than "which" exact resources to assign due to the tight integration of chip-scale manycores. We claim that this assumption does not hold for Fog computing, where the need for isolated execution, predictability and composability puts unique emphasis on the shape and location of space partitions. First, their proximity to the memory controller should be correlated to the latency criticality of the IoT service, so "which resources" we are assigning is important. Second, the matching shrinking virtual partition to a growing one (in terms of resources to be reallocated) might be positioned far apart from each other in the actual computing grid, thus leading to a new optimization problem: which intermediate "neutral" partitions should be repositioned to enable the actual hand-over of resources? For the first time, we model this problem as a mapping problem of poloyominoes, identify objective function and constraints for a Fog environment, and provide optimal solutions within the time frame of a few seconds. When used with a "prior provisioning prompt allocation" strategy of Fog resources, the proposed solver can be used online.

The working methodology will be as follows:

- Design of a multi-plane network-on-chip architecture for emerging multi-tenant Fog computing nodes.

- Design of a time-division multiplexed sub-network with flexibility and low-latency features.

- Design of a space-division multiplexed sub-network with fast dynamic reconfigurability as the main target.

- Design of the hardware-dependent resource management framework mastering the elasticity of the space partitions.

Overall, the thesis has a unique barrier-breaking attitude between abstraction layers and disciplines to gain cross-layer visibility and complete solutions. However, it should be meant only as the first step on the way of implementing the vision presented in this chapter.

## 1.6 Conclusions

In this chapter, we shed light on the emerging trend of the Edge-Fog-Cloud continuum, which consists of a transparent and adaptive hosting environment that fully realizes the "everything as a service" provisioning concept, from centralised Cloud to the Edge, and from network and computing infrastructure up to the application layers.
From several brainstorming sessions we had with academic and industrial partners, the chapter reports a vision with the concepts and components that might shape up this distributed and integrated computing platform. We point out the most vibrant enabling technology for such a vision, which we identify in multi-tenant public Fog nodes. The latter hold promise of bridging the gap between smart objects or IoT sensors and the Cloud infrastructure. However, we observe that building a Fog node with current hw/sw technology cannot span a satisfactory trade-off between computational requirements and limited capacity. In this direction, we identify the promising direction of elastic computing as a way of dynamically reallocating resources from one IoT service to another depending on differentiated service peaks. The thesis places contributions in all three components involved in the materialization of elastic computing, namely the multi-tenant Fog node, the hardware support for dynamic hardware reconfiguration, and the dynamic resource manager. Far from being exhaustive, the thesis represents a stepping stone into further research and development in such a promising yet wide research field.

# Chapter 2

# State of the art

In the previous chapter, we shed light on the growing gap between the requirements of emerging distributed systems and the characteristics of existing computing platforms. Furthermore, we proposed the new concept of elastic computing to bridge this gap, which we pursue for the design and management of Fog node architectures.
This chapter reviews relevant previous work in related disciplines, and for each group of works highlights the novel contribution of this thesis.

## 2.1 A gap in Fog computing

As we have mentioned in the previous chapter, traditional Cloud computing is being challenged by new emerging applications dealing with the proliferation of deployed edge devices, the huge volumes of generated data, the demand for low response latencies and/or the request for augmented privacy and security [149]. To deal with these challenges, Fog computing, an extension of the Cloud at the edge of the network infrastructure, can execute these applications (or part of them) closer to data sources and/or users.

Fog Computing is thus a very appealing component to build a new compute continuum ranging from Edge devices to the Cloud, and serving as an ecosystem for IoT services [33]. The most important issue is arguably that at the moment of this writing there is no public Fog Computing platform widely available on the market. Existing solutions are very specific, offer peculiar services and capabilities, and hit the market with different sales models and prices. On the one hand, Internet giants like Amazon, Microsoft and Google provide Cloud-centric digital products, where management and orchestration of users' resources, models and data are performed at the Cloud side. On the other hand, alternative approaches (e.g., EdgeX Foundry, Nebbiolo) use Cloud-based support only occasionally for demanding operations, and try to push computation as close as possible to the Edge of the network.

In many cases, service providers work around these limitations by deploying and managing their own Fog Computing platforms, typically opting for COTS hardware platforms. These platforms tend to HPC-like performance levels and do not lend themselves to any customization; thus, they are typically overprovisioned and overly expensive for the require-

ments of the application at hand, in addition to large deployment and maintenance costs. Last but not least, this practice gives rise to significant entry barriers for new potential service providers. In all cases, state-of-the-art Fog computing platforms are designed as single-tenant (and often proprietary) infrastructures with mainly static resource management capabilities.

At the same time, there is a lack of any (either de iure or de facto) standard service and/or programming model for Fog Computing applications, despite the efforts to define a conceptual model for Fog Computing [60] and several standardization attempts have emerged, most notably ETSI MEC [125] [34] and the OpenFog Consortium [96]. Hence, service providers have to develop their own service models, with a significant impact in terms of development times and costs, or reuse service models developed for Cloud computing, which are not particularly well suited for Fog and Edge computing. In fact, what characterizes an Edge/Fog Computing environment is the relative resource scarcity and IoT workload dynamicity - demanding computing models with fine-grained adaptivity. Developers of Fog services cannot assume to have enough computational, storage, and communication resources to analyze all the incoming data of several applications concurrently using the full-fledged / sophisticated / fine-grained analytics techniques developed for Cloud environments. Instead, they have to adopt trade-offs: either services discard some data [50] or they have to remodulate big data analytics so that when they run in the Fog they decrease their computational requirements, perhaps switching to coarser grained but less computationally demanding algorithms. In a multi-tenant Fog node, this quality-of-service modulation techniques of IoT services need to be coordinated with the resource manager of the shared Fog node, which can apply prioritization policies.

*Clearly, a new generation of largely parallel computing architectures to be deployed as multi-tenant public Fog nodes would be game changers in the field of IoT, due to the capability to bring more computing power at the edge in a cost-effective way for the delivery of smart IoT services. We believe that the awareness of workload characteristics of Fog applications could help in the design of effective and efficient management and operation of Fog computing platforms. Especially for Fog applications with dynamic workloads, it is necessary for the Fog infrastructure and applications to be deployed in a scalable manner. Meanwhile, Fog management platforms need to incorporate intelligent application placement, dynamic resource allocation mechanisms and automated operation systems to ensure acceptable QoS is guaranteed.*

## 2.2 Elastic Computing

Resource and service management in Edge Computing are compelling topics in the scientific literature [89]. As Fog nodes bring a part of the Cloud computing power closer to users or sensors, the most promising idea for sharing Fog resources among IoT service providers consists of elastically allocating them in an adaptive way over time. Indeed, elasticity represents one essence of cloud computing [140]: when limited resources are offered for potentially

unlimited use, providers must manage them elastically by scaling up and down, as needed. While this notion was originally understood mainly as "resource elasticity" for the Cloud, since 2011 it has been deemed as rather restrictive by the community of distributed systems. In fact, resources' requirements are not determined only by the application using them. If we really treat computation as a service, then we must consider all aspects of a service that might impact the demands on a resource. In 2011, Dustdar [43] presented the Principle of Elastic Processes where he defined cost, quality, and resources as the basic elasticity dimensions that form the foundations of elastic systems. Let's look more closely at these elasticity dimensions:

- Resource elasticity is the traditional dimension of elastic computing, and consists of the capability to allocate/deallocate computing resources on demand, in order to align the state of the execution platform with changing load and business needs.

- Cost elasticity describes the provision of a resource as a result to changes in cost. Service providers apply it when price models are defined for cloud computing systems. In this context, cost elasticity is also referred to as utility computing, in which resources such as computational services provided by virtual machines, data transmission on the network, and storage services provided on different storage hierarchies are charged based on a pay-as-you-go pricing mechanism.

- Quality elasticity measures how responsive quality is to a change in resource usage. The elasticity is a part of the essential nature of cloud applications — that is, to have a precise quality elasticity measurement, the improvement of the service's quality need to be monotonic to the consumption of the resource needed. In other words, the more resources are consumed, the better quality is achieved. The main matter here is to relate a service to a measurable quality and cost function, which computes the resource requirement for a given quality, such as execution speed. In this case, a service gives a deterministic result, but its execution speed is variable based on the required resource. In cloud computing, some computational forms have this desired property. Let's take the example of MapReduce which is a scalable programming framework that lets users process data elastically [39]. It has a desired quality elasticity that asserts that execution speed is scalable to the increase of servers in a distributed file system. The used quality criteria were not restricted to the Response time criterion only, other quality measurements were considered such as the result quality in an approximation-based computing process. This can help provide a new class of cloud algorithms.

Other elastic systems have emerged, such as [123] which is focusing on uniform management of people and computing resources as functional units of the same system. This work introduced the Design by Units principle. The principle defines the Unit as an abstraction over both people and computing resources. Since people have become entangled in bigger heterogeneous systems, [22] have defined the notion of Collective Adaptive Systems. This concept focuses on the societal aspects of systems in which processes, devices, people, and

things, evolve, cooperate, and function as a part of an artificial society.
Thanks to these recent developments we are more familiar and we have better knowledge and understanding on how to manage the increasingly connected and heterogeneous ecosystems of people, computing processes, and things with the help of elastic systems.
While the resource elasticity paradigm is mainstream in Cloud computing, and is being extended to the other dimensions of quality and cost elasticity, it is poorly unexploited in embedded systems since current hardware/software stacks are typically not conceived for dynamic resource management [18, 32, 88, 145]. However, elastic computing could find applicability for Edge and Fog Computing environments [110]. In fact, the use of elastic computing at the Edge has many potential benefits:

- Elastic applications do not need to be constrained by current execution contexts;
- Computation, storage and communication resources need not to be designed for the worst-case;
- IoT services can be synthesized on the fly and not at platform deployment time;
- Opportunistic resource management could improve service quality-of-execution;
- Improved resilience and power management capabilities;
- Capability to preserve real-time constraints in a dynamic and unpredictable execution environment.

*Shaping the hardware/software architecture of Fog nodes even only for resource elasticity (as a first step) is non-trivial and brings Cloud computing elasticity into new ground. Due to their constrained and more cost-sensitive execution environments, it is not possible to deliver the same coarse-grained elastic provisioning of virtual resources as in the Cloud [18], since this would lead to resource over-provisioning (sometimes even tuned to the worst case).*

The resource management problem addressed in this paper is a profound departure from traditional resource management for the Cloud. In that scientific community, bin-packing based algorithms are the mostly used concept to achieve virtual machine placement to physical Cloud servers [72]. At a first approximation, in this thesis we lower the abstraction layer and map such virtual machines (or more suitable virtual entities for the Edge) to an array fabric of homogeneous computing tiles. In this context, physical machine placement matters. Similarly, current research on resource management for Fog computing and for integrated Cloud/Fog systems again abstracts Fog nodes as "containers" of virtual entities with a given capacity [59]. The work of this thesis fully complements these existing frameworks, capable of allocating workloads throughout a network of Fog nodes, or to the Cloud vs. the Fog layer.

## 2.3 Dynamic Resource Management

When using scalable manycore architectures for Fog computing, it is possible to capitalize on previous work on dynamic resource management for them in order to implement the notion of resource elasticity.

Dynamic resource management has been established as an effective technique to improve reliability, efficiency, and performance of computer systems [11]. Managing shared resources during runtime becomes more complex with modern multi- and manycores which support diverse workloads that exhibit varying resource demands, sometimes with conflicting limitations. This dynamic behaviour of workloads that vary across concurrent applications creates significant challenges for homogeneous architectures. The need for a holistic dynamic resource management technique becomes even more vital in heterogeneous parallel processors where heterogeneous compute units are deployed on a single chip, allowing trade-offs between objectives such as maximizing performance and minimizing power [111].
In this context, computer architects use several approaches to perform dynamic resource management. Model-based and rule-based heuristic methods use a model or an encoded algorithm to make decisions during runtime. Optimization methods minimize/maximize an objective while considering certain constraints. Machine learning methods learn the best input values for different observed conditions. Finally, control theoretic techniques use their intrinsic feedback loop to adapt to conditions. A review of these techniques is provided in [66].

Quality of service (QoS) is a primary metric to qualitatively evaluate the system's efficiency in satisfying application's requirements. Applications from different domains have different QoS metrics such as frame rate (multi-media) [62], latency-per-query (web search and financial) [38], throghput (data analytics and streaming) [30], responsiveness (user centric) [77], end-to-end latency and privacy (social media) [100].

Runtime QoS management becomes necessary and challenging with (i) variable workload characteristics, (ii) variable QoS requirements of applications, (iii) identification and translation of QoS metrics into system level parameters for provisioning and (iv) resource contention and arbitration among concurrent applications.

Performance-bound QoS can be guaranteed with compute, memory, network and I/O bandwidth provisioning with dynamic priority identification. In particular, allocating more and/or suitable cores, CPU time slices, exploiting core-level asymmetry to fit application's QoS requirements are common approaches for QoS guarantees [129] [38] [30]. Under workload diversity, smart co-location - scheduling an optimized combination of latency and throughput sensitive applications together, exploits underutilized resources to satisfy QoS of both types of applications [129] [63] [53] [30]. All these techniques feature user/application defined QoS metrics such as latency and throughput bounds or dynamic identification of critical resource contending regions of code [74] and measure QoS in terms of IPC and harmonic speed up for scheduling decision. Monitoring QoS based on IPC and satisfying application requirements through optimized time slice sharing among concurrent applications is proposed in [141].

Combining a set of cores, memory and network bandwidth into a package to priovision isolated resources per application as per their QoS requirements is proposed in [144], to provide infrastructure as a service. All the provisioning techniques prioritize applications based on QoS requirements and dynamically adapt further by monitoring resource utilization upon provisioning.

*None of the above techniques address the hierarchical nature of the dynamic resource management problem, including virtual resource assignment and actual allocation of physical resources. Therefore, the proposed solutions are well-suited for small-scale systems where hardware-dependent effects do not play a fundamental role in determining system performance and/or application quality metrics, and complexity does not need an orthogonalization of concerns to be mastered.*

Fog computing nodes and their management issues are closer to large-scale architectures for which distributed operating systems have been proposed. In fact, with the advent of manycores, researchers had the opportunity to fundamentally restructure operating systems to support a simultaneous mix of interactive, real-time and high-throughput parallel applications. The intuition behind many works is that a much wider variety of performance goals can be met by structuring the operating system around resource distribution, performance isolation, and QoS guarantees.

The foundation of these works is given by virtual machines, exokernels, and multiprocessor runtime systems. Frameworks such as Xen [98] and VMware ESX [5] virtualize machine resources, but not at the partition granularity, whereby CPUs within a partition are scheduled simultaneously. In Exokernel, system services are implemented extensibly at user-level, allowing for instance applications to choose a user-level runtime best suited for the applications. A stronger notion of isolation than Exokernel might enable support for multiple heterogeneous runtimes. In previous work such as LPAR [29] and DLPAR [64] partitioning comes with relevant overhead and lacks of flexibility. In McRT [23], space-time partitioning complements threading runtimes. It envisions a sequestered mode which runs directly on bare-metal resources and acts as a light weight threading system, and is closer to runtimes needed to manage on-chip manycore accelerators. Existing operating systems (e.g., Linux, BSD, or Windows) operate at the granularity of individual CPUs, and therefore use the thread abstraction to make resource allocation and scheduling decisions.

*We claim that spatial partitions provide a more natural abstraction for supporting multiple parallel applications. Further, spatial partitions act as a natural abstraction for implementing resource allocation and accounting frameworks such as resource containers [23] and energy-aware scheduling policies [23].*

CoryOS is a manycore OS that achieves scaling by giving programmers control over the sharing of kernel data structures [114]. However, many more options can be exploited for the sake of scalability, including distributed OS structure [12] and space-time partitioning. The Tessellation OS argues for space-time partitioning as the primary abstraction for resource management on manycore client devices. Like other resource management frameworks for multi- and manycores, it comes with a virtualization layer of physical resources, so that

the high-level assignment of virtual resources can be decoupled by the actual adjustment of physical partitions [36].

*Unfortunately, the mapping problem of virtual assignments to actual resource allocations is typically overlooked or left for future work.*

The gap could be bridged by pioneer works in the field of runtime mapping of multiple applications on NoC-based manycores. This thesis is interested in proactive partition allocation strategies, therefore MapPro is a relevant work in the field [54]. It proactively calculates the propagated impact of spatial availability and dispersion on the network with every new mapped application. This work makes the point for the selection of spatially-contiguous regions and for near-convex shapes to minimize dispersion and external congestion. However, it does not consider the secure-grade isolation of communications, as in Fog computing, and the focus is on internal and external congestion (to the partitions), and on resource utilization. Other key non-functional metrics of Fog computing are not considered, such as proximity of the memory controller for end-to-end latency criticality, and dynamic reconfiguration overhead.

*We observe that no previous work has captured the inherent geometrical nature of the dynamic mapping problem under secure-grade isolation constraints of contiguous regions. The only exception is the work in [102], which models it as a mapping problem of polyominoes. The latter are complex geometric shapes formed by a combination of simple unit square shapes. However, this previous work only focuses on the problem of fragmentation, and is far away from capturing the multiple conflicting quality metrics that come into play when considering the mapping problem in a Fog environment. Above all, the paper exhibits the following fundamental limitations that this thesis addresses:*
*- it imposes a huge limitation of the mapping flexibility in order to make the problem tractable. Limitations include the number of allocated cores, the partition shape and the partition location.*
*- the allocation granularity of cores can become pretty coarse and easily lead to resource underutilization.*
*- there is guarantee on the optimality of the system state after reconfiguration, but not on the transition from one configuration into another.*
Despite these limitations, this work is so relevant for the contribution of this thesis that some details are hereafter reported. The non-interested reader can skip to the next section.

### 2.3.1 Defragmentation framework

The defragmentation framework presented in [102] presents the idea of exponentially separable mapping (ESM), which defines dynamic task mapping constraints on a many-core. In the proposed many core architecture, cores are arranged in a 2D lattice connected together by a mesh NoC.

A multi threaded task on the many-core in general performs more efficiently when the set of cores allocated to it are contiguous (spatially connected by an isolated NoC link) and

compact (the shape formed by the allocated cores has minimum perimeter). Under such an allocation, the communication cost between the task's threads spread over the allocated cores is minimal. The relative benefits from thread co-location would generally increase with the increase in number of spawned threads of a benchmark because of an increase in inter-thread synchronizations. Further, under a contiguous allocation, inter-thread NoC traffic generated by one task remains isolated and does not interfere with another task's NoC traffic. This isolation reduces NoC congestion, enhancing the many-core's multi program performance. But in an open system, neither the arrival nor the departure time of the tasks are known. Therefore, over a span of time, this results in unallocated cores getting scattered all over the many-core, generating fragments in the task mapping. Formation of these fragments leads to the problem of fragmentation. Fragmentation makes it difficult to perform efficient compact contiguous mapping of new incoming tasks.

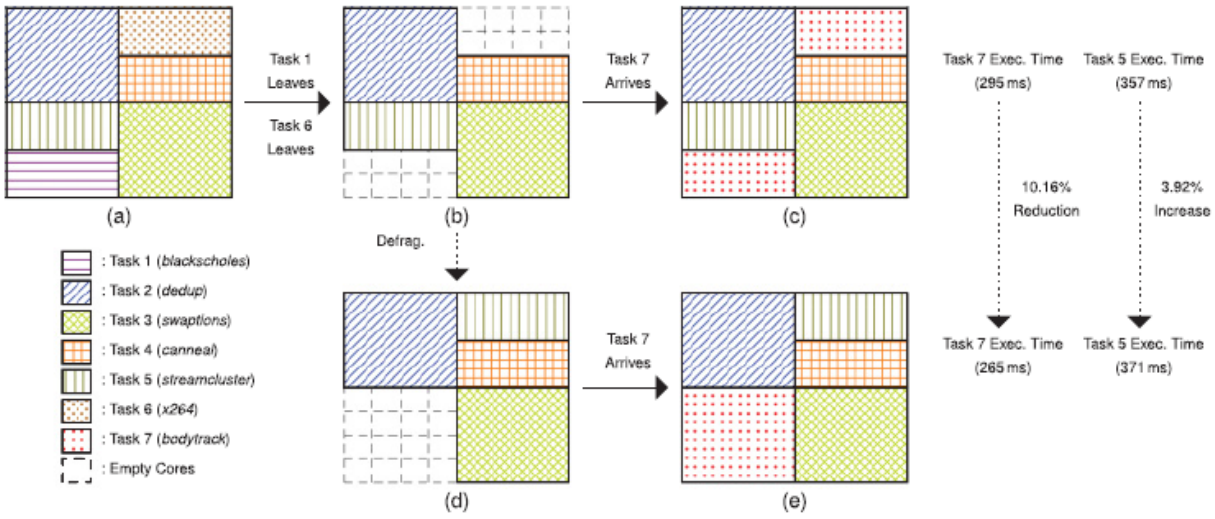



**Figure 2.1:** Figure from [102]. Example showing 10.16% reduction in an arriving task's (Task 7) execution time if mapped after defragmentation against when mapped without defragmentation. An increase of 3.92% in execution time of the task migrated for defragmentation (Task 5) is also observed as an overhead.

Fragmentation can be reduced by using a defragmenter, which consolidates smaller fragments into larger fragments.

Fig.2.1 shows a simple illustration of how fragmentation leads to inefficiency on a 64-core processor. Initially, the processor is executing a total of six tasks as shown in Fig 2.1(a). Tasks 1 and 6 finish and leave the system, changing the many-core state to Fig. 2.1 (b). Task 7 then arrives with a requirement of 16 cores. Fig.2.1 (c) shows the state and corresponding execution time of Task 7 if it is mapped without defragmentation. Fig.2.1 (d) shows the state in an alternate timeline if defragmentation is performed first by migrating Task 5 in the middle of its execution before Task 7 is mapped. Fig.2.1 (e) shows the state and corresponding execution time of Task 7 if it is mapped after defragmentation. Experiments show that the execution time of Task 7 is reduced by 30 ms (10.16%) in the state depicted by Fig.2.1 (e) in comparison to the state in Fig.2.1 (c) because of the optimized

interthread NoC communications. In contrast, the performance penalty of migration on Task 5 for defragmentation is comparatively less at 14ms (3.92%). Task 5 experiences an elongated execution because thread migrations force its threads to wait until the caches on the newly assigned core are refilled from DRAM. Nevertheless, we observe that the net gain in overall performance of the system is positive.
Clearly, Defragmentation would lead to a more responsive open system and a centralized defragmenter is sufficient for a multicore.
However, for a many-core, given the large optimization search space, it would not scale up. Therefore, a distributed defragmenter that distributes its processing across all cores in the many-core and allows multiple fragments to merge in parallel is required.
However, the problem of many-core defragmentation is NP-hard. To overcome these limitations, the main contribution of [102] is twofold: It shows that the exponentially separable mapping (ESM) exhibits properties that allow optimal many-core defragmentation to be performed distributively. At the same time, it also introduces a defragmenter, McD (short for many-core defragmenter), that shows how ESM properties can be exploited for optimal defragmentation of the many-core. McD disburses all of its processing overhead across all unallocated cores in the many-core, allowing it to scale up as the number of cores in the many-cores continue to increase in the future.
This work ends up making the defragmentation problem tractable by overly constraining feasible mappings, especially due to the constraints that ESM puts on the number of cores that can be allocated to a task, the shape of polyominoes that these cores can form, and the physical location of those polyominoes.
Regarding the number of cores constraint, ESM requires that a task must always be allocated a number of cores in an exponentiation series with base 2 (or power of two). If a task comes with a core requirement that is not a power of two, its requirement is buffered up to the next highest (ceiling) power of two. Therefore, tasks can get resources more than their demands which will limit the availability of resources for other tasks. Thus limit the capability to satisfy the demands of incoming and existing tasks in the system. Moreover, this solution is restricted to the grids power of two which will limit feasibility.
As for the shape constraint, they are restricting the number of possible shapes for every demand to one. Which will limit the number of possible solutions. Thus, the feasibility.
Finally, regarding the location constraint, they are limiting the possible positions of the shapes. Therefore, they limit the number of possible solutions.
The defragmentation process gives rise to another issue. In fact, when merging fragments (defragmentation) a number of tasks migrate. In this work, this number of task migrations is not minimized in the objective function, which will affect the system's performance and user experience for offered IoT services.

## 2.4 Copying with state space explosion

It has been demonstrated that resource allocation is one of the most complex problems in large multi-/many-core distributed systems [81, 85].
An accordant search strategy may need to evaluate numerous allocations that are distinct before it finds the optimal solution that meets the systems performance requirements [82, 105].
This presents a big misalignment when it comes to Hard-real-times tasks. In fact, such evaluation most likely will take a long time, thus, it cannot be applied to find the solution rapidly, which is desired within the context of dynamic resource allocation and when real time constraints must be considered. In addition, defining the way that helps to achieve an accurate status of resource allocation during runtime is very difficult and challenging.
This status of resource allocation can be utilization or memory usage of different cores into the system. An imprecise status of resources may lead to an allocation that might not be efficient at runtime. Furthermore, when applications are active simultaneously in the system, we may have various combinations. Thus, it becomes challenging to satisfy performance requirements of each application. Therefore, for each task, since optimal solution cannot be explored at runtime due to limited computation power and evaluation time, it needs to be explored by advanced design-time by Design Space Exploration (DSE) approaches and then to be used at runtime.

### 2.4.1 HAM flow

In order to support real-time applications on a many-core platform, certain performance guarantees such as worst-case execution latency or minimum throughput must be delivered to ensure that the on-line real-time constraints are met.
As a remedy, Hybrid Application Mapping (HAM) is a recent class of mapping methodologies for multi-/many-core systems that has proven its efficient way of delivering exactly such guarantees by combining design-time analysis of application mappings with run-time management.

In Fig.2.2, the flow of HAM is illustrated. First, a Design Space Exploration (DSE) is performed at design time (offline). During this step, multiple mappings with verified real-time properties for the application on the target platform will be generated.
Each mapping requires a certain set of processing and communication resources—to execute the application's tasks and route messages among them and delivers certain qualities in terms of energy dissipation, performance, etc.
For real time applications, the worst-case timing behavior of the mappings is derived within the DSE [136].
Second, the design-time exploration and analysis step is followed by a run-time (online) management step in which the statically-computed mappings are provided to a Resource Manager (RM) that selects and embeds the most fitting mapping for the given set of on-line

system and/or application constraints [71].

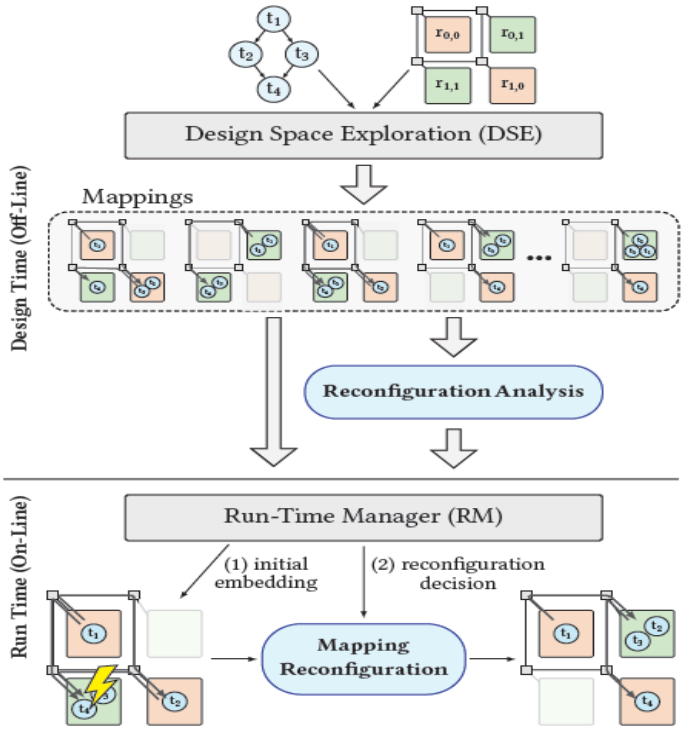



**Figure 2.2:** Hybrid application mapping (HAM) flow

### 2.4.2 Related works to hybrid mapping

Hybrid mapping approaches aim at bridging the gap between predictable application execution and dynamic workload scenarios.

In [9], a hybrid mapping approach is presented. It is designed for heterogeneous MPSoCs. At design time, a single-objective optimization is performed by predefining thresholds for each optimization target (performance, lifetime reliability, energy consumption, and temperature). Multiple optimizations with varied threshold values are conducted, with which a Pareto front is created. This latter is used to determine the best-suited operating points for known applications at runtime.
For applications that their execution is not known before, a dynamic scheduling approach is employed by monitoring the system periodically and occurring when the design parameters diverge from the objectives. In this process, the latency-minimal mapping is chosen.

[135] proposes a scenario-based run-time task mapping algorithm (STM) for dynamic mapping reconfiguration of multiple applications. At design time, mappings optimized for throughput are determined for all scenarios by DSE. At run time, an initial mapping is chosen for all active applications. However, when an application is not compliant with the defined objectives, responsible tasks are remapped considering poor locality and load imbalance. A reconfiguration also occurs upon detecting a new workload.

Works regarding scenario-based (e. g. [130]) and multi-mode (e. g. [137]) embedded system design provide application mapping alternatives which are optimized at design time

for various workload scenarios and execution modes. At run time, it is then possible to choose the application mapping alternative which performs best for the current workload/execution mode. However, all scenarios and execution modes have to be known already at design time. *Optimization for unknown application mixes is not in the focus of these approaches.*

As a solution for unknown applications, some hybrid approaches provide models which give an approximation on how the application performance depends on the availability of resources, (e. g. how execution speedup increases when more resources are available) [70, 138]. However, such functions are not suited for heterogeneous architectures and only approximate the actual performance. Consequently, these approaches are tailored for best-effort execution and not appropriate for execution with real-time constraints.

More accurate anticipation of the impact of resource availability on application performance can be obtained by performing DSE and making the obtained characteristics then available to the run-time resource management [27, 83]. DSE is performed to obtain a set of operating points which are characterized by their resource usage profile and their performance. For handling unknown application mixes, run-time management partitions the available resources between applications. This is done by selecting an operating point for each application such that the overall system objectives are optimized (like maximizing performance/utilization, minimizing energy consumption) under the constraint that the total amount of resources is limited.

In [118], a hybrid strategy based on a formal analysis for throughput calculation and energy consumption is proposed. Although communication is considered in the analysis, it is limited to network interconnects that provide end-to-end connections with fixed latency between tiles. In general, routing of multiple communication over the same communication resources results in non-fixed latencies. The approach therefore requires an interconnect that provides guaranteed service [31, 55, 61]. In this case, the interconnect can only support a limited amount of guaranteed service connections per communication resource.

However, the work in [118] ignores contention on shared resources and instead assumes feasibility solely based on the availability of respective computation resources.
Consequently, ignoring the constrained availability of shared resources may render application mappings infeasible which have been assumed as feasible by [118].

To conclude, hybrid approaches employ a lightweight runtime platform manager to configure the applications efficiently since only selection of the allocation from the storage is required at runtime and most of the computations are done offline. Moreover, the hybrid approach allocates applications more efficiently than on-the-fly heuristics that perform all the computations at runtime.
However, flexibility in these approaches is limited, since all potential applications must be known in their entirety at design time, and analysis results will be applicable only to the analyzed platform. Therefore, design-time analysis needs to be repeated when the application set or platform changes. Furthermore, storing analysis results introduces additional memory overhead.

*Overall, previous work has coped with the proliferation of possible configuration options*

*of shared manycore systems through the combination of online and offline analysis. State-of-the-art Hybrid Mapping Approaches provide tools and methodologies to determine whether the transition to a given configuration meets real-time requirements, both from the viewpoint of the final execution state and of the transient. However, these works explicitly leave the system-level policy that selects the actual target configuration for future work. This thesis tackles the challenge of selecting such state. Above all, it solves the problem to optimality under realistic working conditions for Fog nodes. The presented work is fully complementary with HAM and related works, since once our framework has been identified the shape and size of a partition for the near future, we envision a local manager that can locally manage the resources and map user processes to them with the goal of fulfilling real time requirements.*

## 2.5 Dynamic reconfiguration

Elastic partitioning of a Fog node architecture has not only to do with the computation of the next system state, but also with how to achieve it from an initial condition. Therefore, the work of this thesis addresses in part the domain of dynamic reconfiguration, which goes hand in hand with dynamic resource management, although not necessarily the reconfiguration overhead is accounted for when dynamically readjusting resource allocations.

Many-core systems enable the concurrent execution of dynamic mixes of applications on a shared set of dynamically available resources. Migration of tasks between the cores of a many-core processor allows to dynamically balance the CPU load and hence to improve the temperature distribution of the chip [56]. Furthermore, it increases the resource locality [48] and enhances fault tolerance [115]. It also enables features yet unknown to embedded systems, such as adding new applications at runtime.
Predictability, determinism, and a-priori schedulability guarantees are of paramount importance in safety-critical embedded systems especially with hard real-time (RT) constraints. On many-core processors, task migration has been subject of research with focus on decreasing the migration penalty [26].

Some works also concentrate on soft RT systems where migration is often triggered by missed deadlines [13]. Only little work has been done concerning task migration in hard RT systems.

Authors in [92] devise three agreement protocols that enable a single-threaded application to plan its future execution both temporally and spatially. An application can be executed on the subset of cores on which its instructions are present, encapsulated in a dispatcher. A task migration transfers the context between these dispatchers. The authors present an analytical and experimental evaluation of the worst case delays of the proposed protocols that select the next dispatcher to execute the application.

In [93], authors extend their Limited Migrative Model approach to support inter-application communication on a separate NoC.

Hilbrich and van Kampenhout [57] evaluate methods for dynamic reconfiguration of

avionic embedded systems. They focus on deterministic task migration and leverage mode changes, i. e., task-to-core mappings which are statically derived and can be switched online. Task migration is an important part of a mode change, during which no guarantees are given. At the same time, the application, including its worst-case execution times (WCETs), communication latencies, and external events must be known at design time in order to define the modes.
This work is basically based on flexible partitioning as an approach to improve resource utilization and fault tolerance using dynamic reconfiguration.
Flexible partitioning requires task migration between cores via a shared resource - the NoC - which may endanger the required predictability. Therefore, they analyzed a variety of task transfer mechanisms in many-core processor in order to determine their potential for deterministic reconfiguration during run-time. Moreover, at the NoC level, flexible partitioning and guarantees on Quality of Service (QoS) are fulfilled by using Time Division Multiplexing (TDM).

Katre et al. [69] present a policy to guide task migration decisions on multi-core processors with temporal guarantees.
The worst-case migration delays for the migrating task and the remaining tasks as well as the communication costs are investigated by an offline static timing analysis. This worst case migration overhead is used to determine the set of feasible migrations, i. e., the tasks that can be migrated without disturbing the RT constraints of the system.
The authors present a formula for the weighted migration costs, which allows to select the migration with the least costs among all feasible migrations. Different cache-line transfer policies are considered by a comparative term in the formula.
*The main limitation of those works is that they are limited by static migration options precomputed at design time.*

In contrast to the previously mentioned solutions, the task migration process presented in [91] is not only limited to the set of precomputed options derived at design time, but performs the feasibility test online. Thus, the set of possible destinations for a migrating task is potentially larger and less memory is required to store precomputed results in general.
In this work, the task migration concept is implemented as a service at OS level.
The migration procedure is split in 3 parts that are executed by: the Decision Unit, the Investigation Unit (IU), and the Execution Unit (EU).
First, the Decision Unit (DU) uses global monitoring and profiling information, e. g., temperature and task runtimes, respectively, in order to create new migration requests. They assume that the DU does not issue a second migration request as long as another one is still processed.
Second, the Investigation Unit (IU) receives the migration request and evaluates if the request can be executed without violating the real time constraints. The IU is implemented as a low-priority background task on every tile. On each destination tile, the local IU task checks whether the tile can receive the task.
Finally, on each core, the Execution Unit (EU) is implemented as a low-priority background

task that can act as sender and as receiver and is activated by the DU.
The proposed concept guarantees the application's hard deadlines on a many-core processor during a task migration by a runtime feasibility check of a migration request prior to its execution.
In spite of the benefits of this work, it contradicts the definition of composability which was presented by B.Akesson in [17]. In fact, based on his definition, we can conclude that weighted Round Robin (WRR) arbitration used in this work ensures predictability but not composability, since the starting times of applications suffer from mutual interference.
*Predictability and Composability are two different properties, and one does not imply the other. Predictability means that a useful bound is known on temporal behavior, and composability that the temporal behavior of an application is independent of other applications.*
[17] illustrated the difference by discussing four example systems, shown in Fig.2.3, that cover all combinations of predictability and composability.
The first system, depicted in Fig.2.3(a), consists of two processors, each executing a single application. Assuming that the applications are predictable and that worst-case execution times are known for all tasks. Data is stored in a shared remote SRAM that for simplicity is reached by direct wires. The SRAM has a latency of one clock cycle that is independent of other requestors. The SRAM is shared using TDM arbitration, which is a predictable and composable arbitration scheme, since the latency of a requestor is bounded and independent of other requestors. This makes this system as a whole both predictable and composable.
For the second system in Fig.2.3(b), the TDM arbiter is replaced with a round-robin arbiter (RR). This makes the system predictable, but not composable, since the round-robin arbiter creates a dependence on the presence or absence of other requestors.
In the last two systems private L1 caches are added to the processors in both previous systems. A private cache is composable, since it is not shared between applications. However, it makes the systems unpredictable, since a useful bound cannot be derived on the time to serve a sequence of requests.
The third system, in Fig.2.3(c), is hence composable, but not predictable.
The last system, shown in Fig.2.3(d), is neither predictable, nor composable.

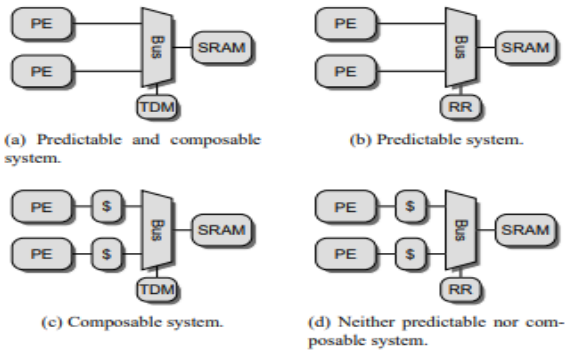



**Figure 2.3:** Four systems demonstrating all combinations of the predictability and composability properties.

*In this thesis, we target the composability property through the spatial partitioning of the underlying hardware platform among the consolidated IoT services. This enables IoT applications to be developed and verified in isolation, and to be later safely composed on the multi-tenant Fog node. We also target the composability through time-multiplexing the global communication flows, for which predictable timing bounds can also be delivered. The target architecture of this thesis allows also the formulation of timing bouds for real-time applications inside each spatial partition, provided suitable arbitration mechanisms are used in the network-on-chip routers and in the core schedulers [107].*

## 2.6 SDM/TDM scheduling of hardware resources

Most existing solutions for resource sharing at the Edge adopt software containerization techniques, that allow to execute service components and run them in a mildly isolated environment.
However, it has been proved that security vulnerabilities can impact this isolation, and thus affecting the overall security [1, 3].
Moreover, software consolidation on manycores gives rise to a number of subtle interference effects that take place in hardware, associated with the partitioning pattern, that may break not only security guarantees, but also predictability and composability. For instance, if communication flows logically pertaining to different partitions/IoT services can slow down each other through resource contention, it is possible to form timing channels to leak out useful information, but also to give rise to hard-to-bound interference for timing analysis and to non-composable performance of individual IoT services. To this end, combining software with complementary hardware techniques is an effective way to deliver the secure-grade isolation, predictability and composability requirements that Fog services demand.

In this section we will address the strong isolation requirement in depth.
Firstly, we will present some emerging fields where strong isolation is a must. Secondly, we will detail the architectural challenges that limit security. Finally, we will take a closer look at the previous works that aimed at fulfilling the strong isolation property.

### 2.6.1 Strong Isolation Requirement in Emerging Fields

Security has always been an area where network related researchers are continuously striving to get through.
In fact, nowadays Chip Multiprocessors (CMPs) are typically designed with a tile-based approach, where a compute tile, typically including distributed L2 memory, is replicated and connected by a NoC.
In high-assurance systems, support for resource partitioning into isolated domains is an emerging requirement. As an example, Integrated Modular Avionics (IMA) architectures are currently mainstream, constituting a logically-centralized and shared computing platform hosting a variety of avionics functions on a single computing platform (multi-function

integration). IMA demands software partitioning [109], which consists of achieving fault containment in software, independently of the underlying hardware platform. However, with the advent of parallel computing architectures, the partitioning concept should be enforced in hardware as well.
The avionics safety standards dictate the enforcement of partitioning in space and time. Spatial partitioning ensures that an application in one partition is unable to change private data or use private devices of another one.
Temporal partitioning guarantees that the timing characteristics of an application, such as worst-case execution time, are not affected by the execution of an application in another partition. While these concepts have been fundamentally driven by fault-tolerance and performance predictability considerations, protection from cyber attacks is getting increasing attention in the design of avionic systems [95]. Systems must be safeguarded not only against DoS but from other vulnerabilities such as timing channels [124]. Similar considerations hold for the automotive domain, where densely populated networks of electronic control units are being replaced by distributed central compute platforms, driven by cost, weight, complexity and security considerations.
The picture becomes even more critical when considering the possible co-integration of safety-critical functions (e.g., the breaking system) with infotainment and even third-party applications. In this domain, numerous works are surveying cars' intercommunication technologies and possible threats [79, 99], and demonstrating different kinds of attacks [126]. Last but not least, computing platforms for space applications are following the same trend, extending the concern from failure cascading avoidance to the interference threats between system components in multicore architectures [46, 134].

### 2.6.2 Architectural challenges: The NoC Sharing Problem

Overall, the key challenge of applying time and space partitioning to CMP platforms lies in the NoC, where an additional layer of possible interaction arises for the system as a whole. Routers and links are shared among domains, and are thus subject to contention and congestion. This renders network performance unpredictable, prevents fault containment and exposes unprecedented security threats.
Several degrees of non-interference can be enforced on the NoC. A first approach to loosening interdependencies among communication flows consists of delivering quality of service (QoS) guarantees. In fact, most QoS techniques aim at limiting flow rates, while restoring nominal rates in the absence of contention. While QoS-augmented NoCs can typically protect from denial-of-service (DoS) and bandwidth depletion attacks between domains, they cannot easily avoid an information leak associated with latency and throughput variations of communication flows as a function of network state. In fact, they can be used as timing channels by an attacker either to infer confidential information from a protected high-security program (side channel attacks) or to have a malicious program deliberately leak information covertly when direct communication channels are protected (covert channel attacks) [142].

When protection against timing channel attacks is required, even cycle-level variations of communication performance should be prevented, a scenario that we hereafter denote as *strong isolation* of domains.
Interestingly, strongly isolated domains simplify handling propagation of faults and the certification process (most system-level interactions are avoided).
In order to deliver strong isolation to networked domains, the NoC must be designed to guarantee the non-interference property in its strictest sense: injection of packets from one domain cannot affect the timing of packet delivery from other domains.
To sum up, a better use of many-core processors or accelerators at the Edge might be by adopting space-division multiplexing (SDM) and time-division multiplexing (TDM) policies. In the next two sections relevant SDM and TDM works for NoCs are presented.

### 2.6.3 TDM-based scheduling

In order to prevent interference between concurrent domains, one straightforward solution is to statically schedule them on the network over time with some form of time-division multiplexing (TDM) [61].
However, this approach typically comes with relevant performance overheads. On the one hand, strict non-interference requires resource allocation decisions being independent from application demands. On the other hand, performance of packets in time-multiplexed NoCs is highly sensitive to the scheduling methodology of time slots.
While existing solutions support a generic global schedule, its reconfiguration at runtime is not obvious. In some cases, this implies solving a slot allocation problem in software or through hardware acceleration, and large programming tables at network interfaces [61] as well. In other cases, invasive hardware modifications are needed to optimally support different system configurations (e.g., number of domains) [10]. Thus, they are mainly suited for design time customizations.

*Prior approaches to TDM-based scheduling in NoCs lose relevance when they are challenged with conflicting requirements of latency optimization, area efficiency and architectural flexibility.*

Numerous designs perform TDM scheduling at the time-slot level [61][14][45] where the scheduling is typically performed offline (and assumes perfect a priori knowledge of applications to be running on the system), and then statically applied to the entire NoC [78]. In these approaches latency overhead can be quite substantial.

AEthereal [61] employs pipelined TDM (at the time-slot level) and circuit-switching to guarantee performance services. Traffic is separated into two main classes: 1) guaranteed service (GS) and 2) best effort (BE). Excess bandwidth not used by GS flows is given to BE flows. Packets on a single connection are always ordered, but ordering cannot be enforced between connections.

Moreira in [94] proposes an online resource manager using AEthereal TDM. It can modify the domain configuration at runtime. However, it can not prevent cycle-level variations of

communication. Both approaches incur a substantial programming overhead of time slots at network interfaces.

The SuperGT NoC [122] is an evolution of AEthereal providing three QoS classes. Aelite[14] simplifies the router architecture by providing only GS and supports multicast traffic and fast virtual-circuit setup.

Argo [45] allows schedules to evolve at the granularity of a single cycle, even when routers have more than one pipeline stage. The hardware cost is low, but the latency overhead can be substantial.

In [143], static space and time network partitioning is used to provide multi-way isolation among supported domains. This multi-way isolation property comes at a high performance cost, which is alleviated by the introduced reversed priority with static limits (RPSL) mechanism. It uses priority-based arbitration and static limits to guarantee one-way isolation between high-security and low-security flows.

A recent architecture, SurfNoC [58], employs optimized TDM scheduling, applied at the VC level, to minimize the latency overhead. However, the required hardware is expensive. Achieving low-cost implementations with SurfNoC would increase the latency overhead of static scheduling.

*The current state-of-the-art in TDM-based scheduling is PhaseNoC [10]. It improves the VC-level scheduling proposed by SurfNoC by pre-configuring the network in order to receive packets from the same domain at all the input ports at every cycle, and performing the arbitration in the next cycle of this incoming domain.*
*Following this approach, PhaseNoC meets the first requirement, by minimizing the latency overhead. However, it lacks flexibility as it requires adding router pipeline stages to support a higher number of domains.*
*PhaseNoC divides the network into $x+y+$ and $x-y-$ to support a higher number of domains. However, following this approach, it can not guarantee the non-interference property any more, for which it would need input speedup similarly to SurfNoC.*
*The work in this thesis presents a more flexible and scalable approach to low-latency TDM communications, and compares directly against PhaseNoC.*

### 2.6.4 SDM-based scheduling

The isolation problem in the SDM NoC can be solved by gaining a tighter control over the routing paths of network packets in hardware. However, the solution should be partition-specific. *In essence, we cannot typically rely on a network-wide global routing algorithm, but we need to custom-tailor a per-partition routing algorithm, which needs to be computed and programmed at runtime as a new partition is set up, or its configuration modified. This thesis aims at overcoming this limitation and overhead.*
In the following paragraphs, we will present the most relevant previous routing mechanism support for spatial isolation.

### Table-Based Routing

Traditional routing mechanisms for NoCs are based on routing tables [139]. The most common solution is to implement them at the network interface of source nodes (source-based routing), and to embed the stored routing path to destination in the packet header.
When implemented in the network interface of source nodes (source-based routing), routing tables need to include a routing path for each possible destination reachable by that node. At each switch, a simple logic reads in the routing path from the packet header and implements the pre-computed routing action for that switch.
Source-based routing comes with overly long packet headers, since the embedded routing path field should be as large as the worst-case number of hops the packet could traverse in the network. Moreover, even though partitions will be used, the network should provide support for the largest possible partition size, which consists of the network as a whole, thus requesting a conservative number of routing table entries for full connectivity.
Routing tables could also be implemented at each switch (distributed table-based routing), but their access delay would end up impairing the network operating frequency. For this reason, this paper will consider source-based routing as the reference approach for comparison.
If tables are implemented at each switch (distributed table-based routing), then packets need only to carry the destination address, which will be looked up in a table to derive the routing decision for that hop. This approach is rarely implemented in on-chip networks, since tables are large and their access delay ends up impairing the switch operating frequency.
Whatever the table location (at network interfaces or switches), *table-based routing exhibits large overhead when reconfiguring partition shapes at runtime, due to the lengthy course of action required.*
By assuming static reconfiguration (that is, the temporary suspension of the application/service under test for the sake of updating its configuration as well as the configuration of the underlying compute and memory partition), and source-based routing (i.e., tables at network interfaces), the following tasks should be performed to reconfigure a partition:

- Draining the interested portion of the network from ongoing packets for deadlock-free reconfiguration, and blocking of traffic injection. This step can be overlapped with the next two ones for an optimized reconfiguration process.
- Computation of an ad-hoc routing algorithm for the new partition shape. This processing can be performed by the host processor or by a dedicated controller.
- Search for all source-to-destination paths for intra-partition minimal-path routing.
- Programming of the routing tables of the network interfaces belonging to the partition through dedicated control packets over a TDM NoC. Only the modified table entries associated with destinations that belong to the partition should be selectively updated.
- Activation of the new routing function, and software-managed traffic resumption.

*This mechanism is illustrated in Fig2.4, where, for every partition belonging to the initial configuration, the routing algorithm is computed, then routing paths are established. Finally they are stored in the routing tables. This procedure gives rise to a huge reconfiguration overhead for physical partitions, including both a hardware (spreading of new configuration information) and a software contribution (search for a routing function and for routing paths). This thesis aims at completely removing the software overhead, and at minimizing the hardware one, at runtime.*

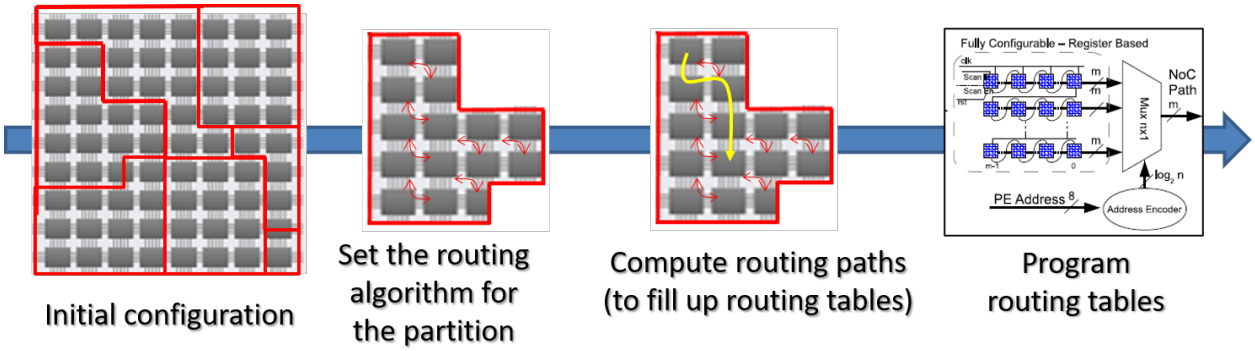



**Figure 2.4:** Table-Based routing mechanism

### Logic-Based Distributed Routing

To prevent the interference between packets belonging to different partitions, some previous works used Logic-Based Distributed Routing (LBDR).

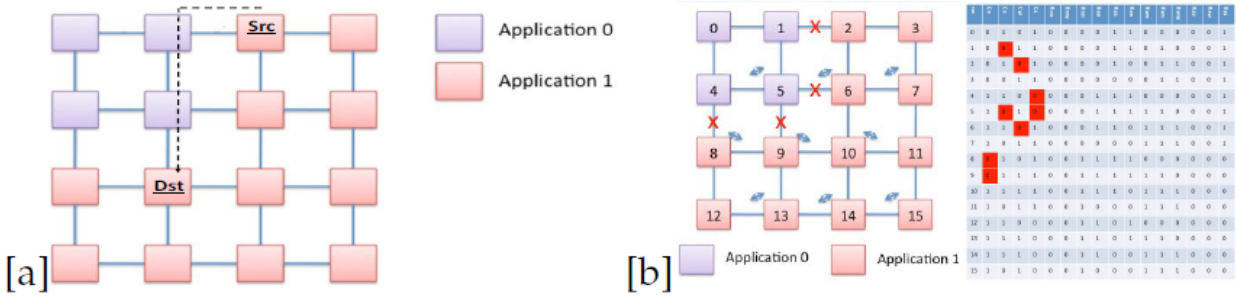



**Figure 2.5:** (a)Interference effects without close control of the NoC routes. (b)Content of the LBDR configuration registers for the switches of a NoC with two partitions.

In Figure 2.5(a) we can see two generic partitions are mapped onto the PMCA, and where the underlying NoC uses xy dimension-order routing. Clearly, some source-to-destination flows in the second partition take routing paths that end up invading the spatial region of the first one. This may result in performance variations, traffic congestion or security threats. In this context many researches have advocated the use of the LBDR [47] not only as a global routing mechanism replacing routing tables for better scalability and lower cost, but also as a simple means of enforcing isolated spatial partitions.
LBDR consists of simple combinational logic at each switch input port processing destination coordinates from packet headers and local switch coordinates. Based on them, it then

computes a productive switch output port for the packets. The decision is taken based on the computation of the destination quadrant, hence its complexity is not a function of the network size, but only of the switch radix. For each target quadrant, two productive directions are identified (e.g., North and East for the North-East quadrant), which are compared against the forbidden turns at the next hop of the routing algorithm. The mechanism finally selects the legal direction that conservatively enables the packet to approach the destination at the next hop.

This logic uses 12-bit configuration register per switch, including 4 connectivity bits $C_i$ and 8 routing restrictions $R_{xy}$.

Forbidden turns at the next hop are encoded in a small configuration register of the LBDR routing logic, the so-called *routing restrictions*, pictorially illustrated as arrows in Fig.2.5(b), where the arrow direction indicates the direction that cannot be taken by packets. Thus, making the routing function easily reprogrammable.

This routing mechanism needs to be aware of whether there is a connected switch in the north/south/east/west direction or not, so to avoid routing a packet to unconnected directions. This is the case of border switches in a 2D mesh topology. This information is coded in the so-called *connectivity bits* of the same LBDR configuration register.

Connectivity bits are not just used to denote the topology boundary, but also the partition boundary.

Figure 2.5(b) illustrates the content of the LBDR configuration registers for each NoC switch. In red, the connectivity bits are set to zero to denote the common boundary of both partitions.

For instance, Fig.2.6 defines the connectivity bits and routing restrictions of switch 5. We can conclude that the switch is connected to the four directions: North, East, West and South. However, it has two forbidden turns that prevent packets coming from the East to turn to the North (vice versa) and packets coming from the South to take the East direction (Vice versa). When compared to the hundreds of bits required by routing tables, the superior area and delay properties of LBDR become apparent, which are however paid with a decrease in flexibility.

In fact, although LBDR has been demonstrated to successfully implement the most common topology-agnostic routing algorithms, it is capable of routing from 30 to 40% of the irregular topologies derived from a 2D-mesh (the *LBDR feasibility domain*) [113]. In fact, the mechanism can easily support only topology irregularities that still enable minimal path routing with respect to the ideal fully-connected 2D-mesh. Even restricing the focus to routable NoCs and partitions, LBDR gives rise to a mismatch between the underlying routing algorithm and specific partition shapes. This mismatch is illustrated in Fig.2.7, where partitions use minimal path routing with respect to the 2D mesh as a whole, however some routes turn out infeasible using LBDR. In fact, nodes belonging to the south-west flank of partition 2 cannot reach nodes belonging to the other flank of the same partition, due to the routing restrictions. Clearly, using distributed routing logic at each switch can minimize the area and delay overhead of routing tables, but such logic is natively routing algorithm- and

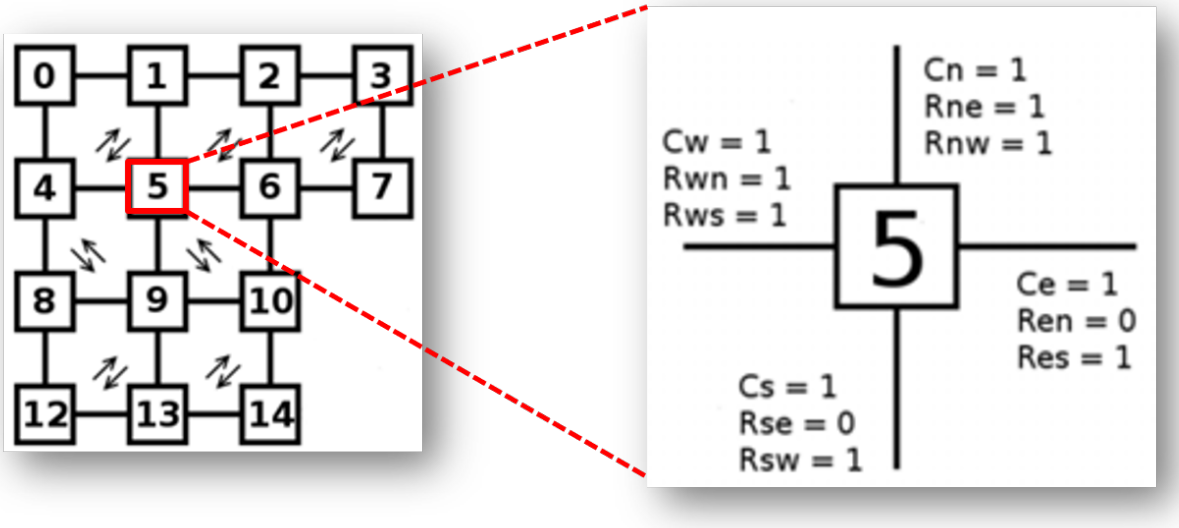



**Figure 2.6:** LBDR logic inside the switch

topology-specific[139].

Most previous work targets an enhanced coverage of faulty links and switches through LBDR without reverting to routing tables.

uLBDR supports non-minimal path routing through the use of special deroute bits [113]. However, 100% coverage of 2D-mesh irregularities is achieved only by means of virtual cut-through switching and by forking packets in some special cases. Moreover, some faults require the re-segmentation of the network, which is compute-intensive and requires deadlock-free dynamic reconfiguration.

$d_2-$LBDR achieves 100% coverage of 1- and 2-link failures with lower overhead than uLBDR [112]. It complements vanilla LBDR with a distance register and a new deroute strategy on each router. This work introduces the interesting notion of selective masking of routing restrictions depending on the communication flow, and aims at avoiding re-segmentation. However, there is no evidence that this fault tolerance can be changed into the support of a set of partition shapes without recomputing the routing function. At the same time, uLBDR and $d_2-$LBDR rely on additional programming bits for the routing logic that are demanding in terms of computation requirements [19, 112]. Hence, these approaches do not lend themselves to fast runtime reconfiguration.

*Overall, LBDR has never been used in the context of a partitioned SDM NoC for the sake of fast and flexible partitioning. This is the primary focus of this thesis, which addresses the problem under the security and flexibility requirements of a Fog computing environment.*

## 2.7 Conclusions

The work of this thesis lies at an unexplored intersection between Cloud computing, computer architecture and resource management at several abstraction layers. As a result, this

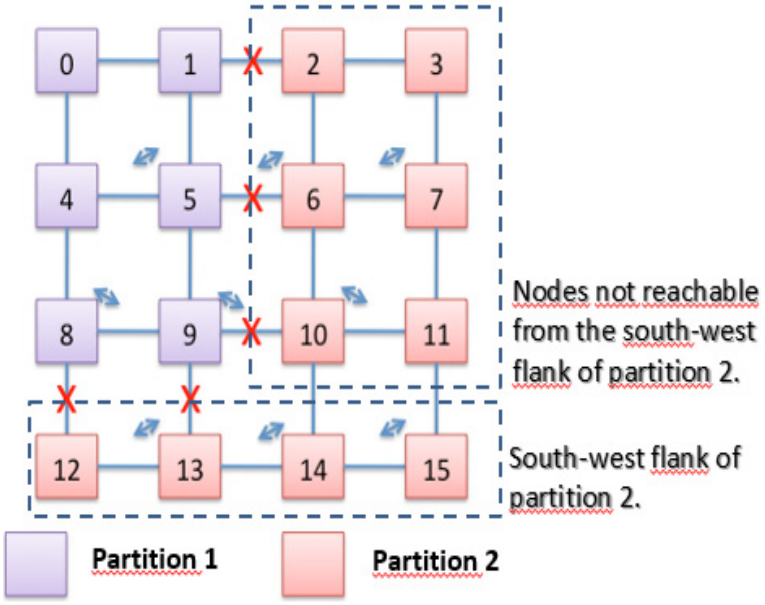



**Figure 2.7:** Representation of a routing algorithm for the NoC as bidirectional routing restrictions. At the same time, a reachability problem is illustrated, which limits routability of partition shapes with vanilla LBDR.

chapter reviews state-of-the-art of the main related disciplines and points out the unique interdisciplinary contribution of this thesis. The latter addresses a twofold extension of baseline disciplines:

- It matches dynamic management of manycores with the non-functional requirements of Fog computing.
- It extends the elastic provisioning of Cloud resources to the Fog through hardware and management support.

# Chapter 3

# Target Architecture

This chapter expands on the proposed target architecture. First, we will describe the Fog node architecture. Second, we will detail the accelerator cluster and the memory architecture. Finally, we will present the on-chip interconnect considered in our work.

## 3.1 Fog node architecture

Our solution targets an heterogeneous Fog node architecture. Figure 3.1 shows the block diagram of the heterogeneous system template targeted in this work.

It consists of a general-purpose multi-core CPU (the *host*) running a separation kernel that partitions platform hardware resources into high-assurance virtual machines with strictly controlled information flows [67]. We assume one IoT service is associated with each virtual machine. General-purpose multi-core CPU are not the most efficient in terms of performance/watt to process compute intensive applications. To address this challenge, the *host* is coupled to a programmable many-core accelerator (PMCA) composed of several tens of simple processing elements (PEs), where critical computation *kernels* can be offloaded by IoT services. The PEs considered here are simple independent RISC cores, perfectly suited to execute both single program, multiple data (SPMD) and multiple program, multiple data (MPMD) types of parallelism. This better copes with the characteristics of data analytics workloads, which alternate data-parallel stages to irregular, task-based parallelism (e.g., graph traversal). [87].

In light of the nature of accelerated applications, we model the PMCA as a massively-parallel processor array that is shared among concurrent IoT services by means of spatial partitioning of its compute and memory resources. Connectivity between PMCA clusters is delivered by means of mainstream network-on-chip (NoC) technology.

The *host* and the PMCA physically share the main memory (DRAM). For improved data and computation locality, they leverage internal memory hierarchies to keep most frequently accessed data in fast, local storage. The host does so by relying on hardware-managed caches, whereas the PMCA leverages scratchpad memory (SPM) and multi-channel, high-bandwidth direct memory access (DMA) engines.

High-end computing platforms are increasingly relying on Input-Output Memory Management Units (IOMMUs) to allow the *host* and the PMCA to exchange virtual shared data pointers.
Without lack of generality, data sharing between the *host* and the PMCA relies on a simplified I/O memory management unit (IOMMU) managed in software, similar to what is proposed in [132]. However, any HW, SW or hybrid technique can be adopted at the system level for *host*-to-PMCA communication.

Access to the main system DRAM is mediated by the IOMMU attached to the NoC. Since the concern of our work is on the PMCA system only, we will not further discuss external memory from now on.

*The main idea is achieving a secure, efficient and flexible PMCA sharing among several concurrent IoT services running on the host and offloading computations. Consequently, in the following we only discuss the internal PMCA architecture design.*

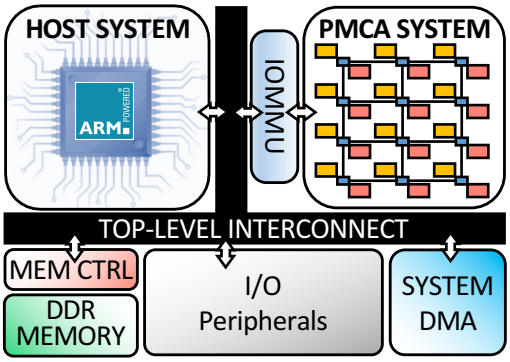



**Figure 3.1:** Target architecture: system-level view.

## 3.2 Accelerator cluster architecture

The block diagram of the PMCA template considered in this work is shown in Fig.3.2. It consists of a hierarchical design, where a top-level communication system (typically a NoC, structured as a 2D mesh) interconnects a number of *clusters* (also known as *tiles*).

Several products are nowadays available that leverage the cluster-based design paradigm, including STHORM[87], KALRAY [2] or Parallela [4]. Modern tiled architectures include several PEs per cluster (typically up to 16).

Cluster cores are typically equipped with private instruction caches, while for data they share a single L1 *tightly-coupled data memory* (TCDM), which is software-controlled.

The TCDM is usually designed as **a multi-banked SPM, explicitly managed by the software via DMA transfers**. This memory can be accessed via a fast local interconnection (e.g., a crossbar, or mesh of trees), which ensures identical, fast read/write latency to every PE (ideally 1 cycle).

The L2 memory is implemented here as a shared software-controlled scratchpad memory (SPM). In addition, we assume the PMCA has one or multiple I/O ports, that can be used

by the system DMA to access internal L2 banks for data/code transfers from/to the main memory. Without lack of generality, we assume a single I/O port located in the bottom-right corner of the 2D mesh.

In our work, we focus on managing the PMCA sharing (i.e., resource multiplexing) among the several *host-based* IoT services offloading computation kernels.

More details about internal PMCA memory hierarchy design, precisely L2 memory organization, will be discussed in the following section.

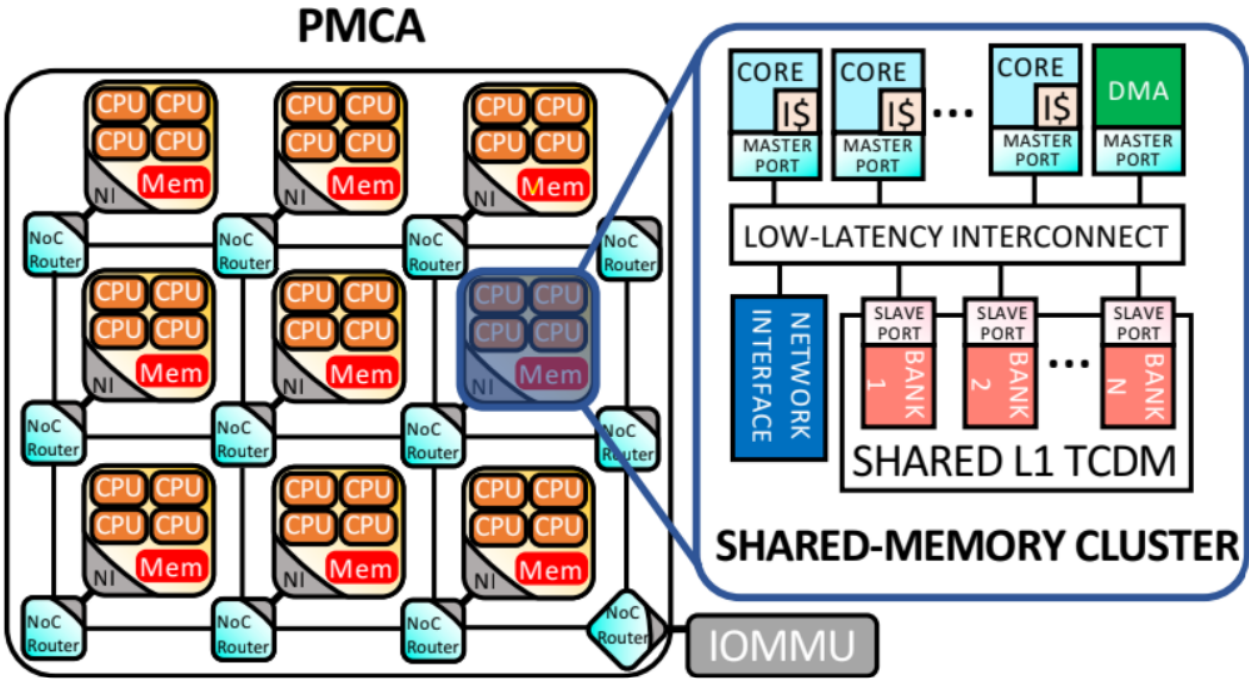



**Figure 3.2:** PMCA architecture, with a zoom into the Cluster architecture. Distributed multi-banked L2 not shown.

## 3.3 Memory Architecture

The PMCA memory system is organized as a partitioned global address space (PGAS), thus each core in the system can explicitly address every memory segment, from local L1 TCDM to the distributed L2 banks. This has demonstrated realizable performance and productivity potential for large parallel computing systems with distributed memory architectures.

The L2 memory is implemented here as a shared SPM. A common design choice is distributed (multi-banked) L2 shared memory, with each NoC router hosting a L2 bank.

## 3.4 Elastic accelerator sharing

If multiple IoT services, from different providers, running on the *host* require simultaneously the use of the PMCA, arbitration policies need to be in place to guarantee resource sharing. *Time-Division Multiplexing* (TDM) is the primary option to share a PMCA. TDM can be

implemented at a fine granularity by executing each offloaded kernel sequentially for a certain *time slot*.

Recent high-end GPGPUs are increasingly supporting context switching in hardware, particularly to implement GPU virtualization. The costs associated with this type of hardware support, however, are very high [132] and it is questionable whether they will ever be affordable in the context of Fog nodes.

Above all, accelerator sharing via fine-grained TDM assumes that the offloaded applications always contain sufficient parallelism to effectively utilize the whole manycore.

This is not necessarily true for IoT applications, where the data sets are usually smaller than in scientific computing. For instance, through the GEM5 simulation of a 9-cluster system with 8 cores per cluser, we empirically verified that under ideal memory and interconnection (each load/store is handled in 1 processor cycle), the *GaussianBlur*, *rBrief*, and *FAST* image processing applications emphasize a plateau in the speedup already beyond 6 clusters, and that only one benchmark (*ROD*) presents a speedup very close to the ideal case.

These results suggest that a better use of many-core accelerators at the Edge might be that of allowing multiple offloaded kernels to co-exist at a given time, namely to adopt *space-division multiplexing* (SDM) policies. We assume each service is associated with a spatial partition of neighboring computing clusters, and that a corresponding partition of L2 memory banks topologically matches the associated compute partition (Figure 3.3(a)).

When we couple this sharing model with a NoC routing mechanism capable of constraining routes within the spatial extension of the compute and memory partitions [128], the outcome is the lack of interference between L2 traffic in nearby partitions, and the delivery of hardware-supported secure-grade isolation.

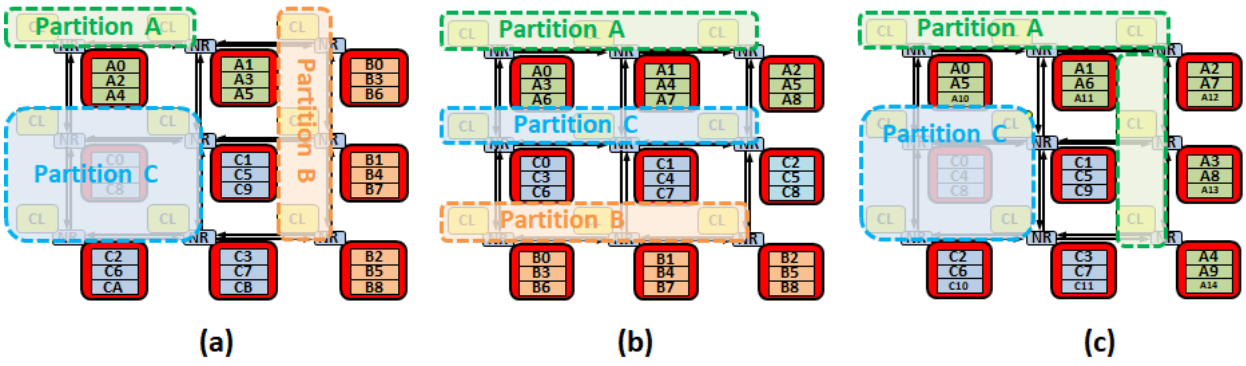



**Figure 3.3:** Distributed multi-banked L2 architecture with partitioning at work in two consecutive partition configurations (a) and (b). (c) Network packet route invading a neighboring partition on the way to destination.

As we explained in previous chapters, our work unlocks the exploitation of the resource elasticity concept at the partition level. In practice, compute and memory partitions are allowed to change in size, shape and location over time, which we refer to as *elastic partitioning*. The latter is required whenever the IoT service executed within the partition asks

for more or less resources to reflect its dynamic workload (e.g., change of execution phase, variability of input data, data-dependent computation). Another scenario is the one where IoT services come and go, since they are migrated to the Cloud, or vice versa, by the Service Supervisor. As an example, Figure 3.3(a) and Figure 3.3(b) represent a possible evolution over time of the partitioning pattern, where a partition includes computing tiles and L2 memory banks. An IoT service is migrated in order to obtain a better allocation or adapting the mapping of the applications to the imposed constraints (e.g., performance requirements, resource availability) [119].
Without lack of generality, in this plot we assume the interleaving of memory locations across the L2 banks of partitions, so to reduce memory congestion.

### 3.4.1 Inter-partition interference

A spatially-multiplexed array fabric of computing tiles is still sensitive to inter-partition interference when irregular partition shapes are allowed, such as in the partitioning pattern in Fig.3.3(c). If the network uses YX routing, packets of the L-shaped $A$ partition would invade the links of partition $C$.

There are three fundamental ways of coping with this problem:

- The irregularity of the partition shape could be limited by construction, for instance enabling only squared or rectangular shapes. In this case, if the NoC uses minimal-path routing, all routing paths would be contained into the spatial extension of the partition, thus yielding communication isolation. This is the assumption of most previous work in the open literature about mapping of workload onto manycore processors. They often target rectangles or squares [68]. In the most flexible case, they target near-convex partition shapes [54]. This design choice has subtle implications. In some cases, this leads to the discretization of resource requests, or to major changes of the partition shape even when small increments or decrements of resources are requested, which negatively affects the reconfiguration overhead. In other cases, the allocation of highly asymmetric rectangular shapes leads to inefficient intra-partition communications, for instance between tasks allocated to cores placed at the opposite extreme of the elongated shape. This effect can be nonetheless controlled by suitable mapping policies of tasks to the cores of a partition.

- Irregularity of partition shapes could be accepted as a way to increase the grid utilization, or to allow fine-grained resource reallocations without major impacts over service continuity (i.e., over the reconfiguration overhead). The challenge for this scenario is associated with the routing mechanism. In fact, routing tables should be reprogrammed at each reconfiguration to constrain the routing paths for intra-partition communications within the partition space, thus avoiding interference between neighboring partitions. This approach is the most flexible one, however it relies on components (i.e., the routing tables) that exhibit significant latency, area and power over-

head, poor scalability and large reconfiguration overhead [113]. Recently, more scalable and lightweight routing mechanisms have been proposed, such as LBDR [113], which however come with flexibility limitations. LBDR can natively route partitions where minimal-path routing is feasible (with respect to the network as a whole). Extensions have been proposed to extend the flexibility to more irregular shapes through the use of pre-programmed deroute information or by even moving from wormhole switching to virtual cut-through switching. Unfortunately, such deroute information cannot be easily computed online for newly established partition shapes. [19]. Whatever the routing mechanism (provided it exhibits some degree of reconfigurability), an unpleasant characteristic of this approach consists of the need to work out differentiated routing functions on a per-partition basis.

- Another way to allow partition irregularity consists of using a global and unmodified routing function for the network as a whole, and of inferring partition boundaries on top of it. This would allow to use more scalable routing mechanisms than routing tables or even than LBDR, such as algorithmic routing [139], due to the removal of the reconfigurability requirement. Unfortunately, there are two major drawbacks of this approach. First, some support for reconfiguration needs to be set up just at the same, since at least the partition boundaries should be reprogrammed into the routing mechanism. Thus, we again need for at least LBDR, or table-based routing. Second, some partition shapes may not be compatible with the underlying routing algorithm, since specific routing restrictions may prevent any kind of communications between neighboring chunks of the partition, thus making the partition as a whole unroutable.

This problem will be addressed more in detail in Chapter 5. However, we anticipate that our approach consists of:

- using a global and unmodified routing function for the network as a whole;
- implementing the routing function through an extension of the scalable and lightweight logic-based distributed routing (LBDR) mechanism.
- turning unroutable partition shapes for the routing algorithm at hand into routable ones through an LBDR extension that selectively and safely ignores redundant routing restrictions for the partition shape at hand.
- accepting a reasonable restriction of the legal partition shapes, which is the trade-off to benefit from the advantages of our approach: fast and scalable routing computation and fast reconfiguration of the partitioning pattern.

## 3.5 Dual NoC architecture

Interference between running partitions needs to be avoided not only between intra-partition traffic flows, but also when accessing memory controller ports. Therefore, we suggest to com-

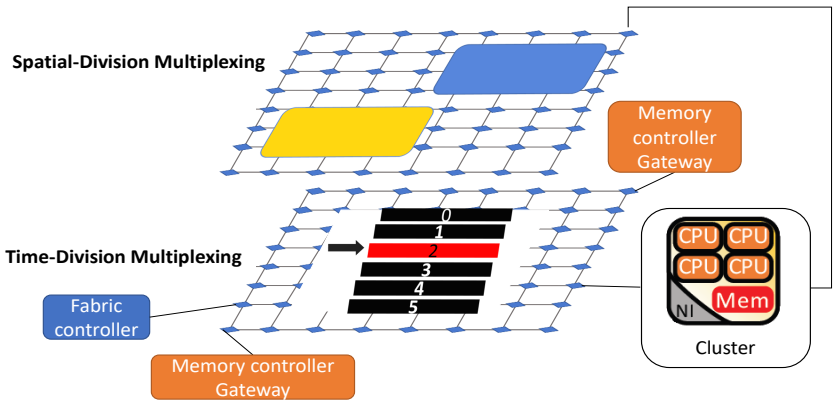



**Figure 3.4:** Dual NoC structure supporting isolation in time and space.

plement spatial-division multiplexing for intra-partition communications with the temporal isolation of non-local traffic to/from the memory controllers, which would otherwise break partition boundaries and invade nearby partitions.

More specifically, our solution envisions a dual NoC structure (Fig.3.4), in which intra-partition traffic goes through an SDM-enabled NoC, and memory controller traffic goes through a low-latency dual TDM NoC. This thesis will contribute a TDM solution that lends itself to the dynamic reconfiguration of the number of time slots based on the number of running partitions at any given point in time. This solution will be highlighted in Chapter 4.

## 3.6 Conclusions

In this chapter, we described the target architecture primarily addressed by this work, and detailed the compute, memory and interconnection components. In particular, the presented architecture targets the distinctive challenges of emerging multi-tenant Fog node platforms. The architecture targets a more ambitious scenario of "resource-elastic multi-tenancy", and comes with dynamic reconfiguration mechanisms supported in the interconnection network capable of flexible and secure partitioning of compute and memory resources among the consolidated IoT services.

# Chapter 4

# A Low-Latency and Flexible TDM NoC

This chapter is organized as follow: We start by an introductory section that presents the motivation of this work and the main idea of our proposal. Then, we detail our approach and its distinctive features. Finally, we analyze and explain the experimental results. This work stems from a joint collaboration with Universidad Politecnica de Valencia (Spain), especially with Miguel Gorgues Alonso and Prof. Josè Flich.

## 4.1 Introduction

In the Edge computing domain, it is often found that executing multiple applications concurrently and dividing hardware threads among them provides greater efficiency rather than time-slicing single applications with large thread counts [49, 148]. As anticipated in the previous chapter, this is also the assumption of this thesis on the usage model of public computing infrastructures offering Cloud-like pay-per-use computing functions to service providers, who end up sharing compute, memory and interconnect resources [97].

This sharing trend has historically motivated research on how to solve contention for access to shared resources, not to limit the execution speedup of concurrent processes [52]. As awareness of security issues keeps growing in the Fog computing domain [20], this contention problem increasingly evolves into a true isolation problem, requiring a trade-off between degree of isolation and performance penalty. In fact, secure-grade decoupling between running applications requires resource allocation decisions to be to some extent independent from application demands.

In the first chapter, we have recalled the importance of strong isolation in multi-tenant Fog node platforms. This requirement is motivated by the need to avoid security threats, to effectively contain faults, and to enable composability and performance predictability. It is in fact of the utmost importance that IoT service providers can develop and verify their applications in isolation, and can then securely run into the system without any kind of functional and non-functional interference from other applications/services. From this viewpoint, multi-tenant Fog nodes share many similarities with security-critical systems, which are typically organized as a set of domains that must be kept separate subject to

certification requirements.

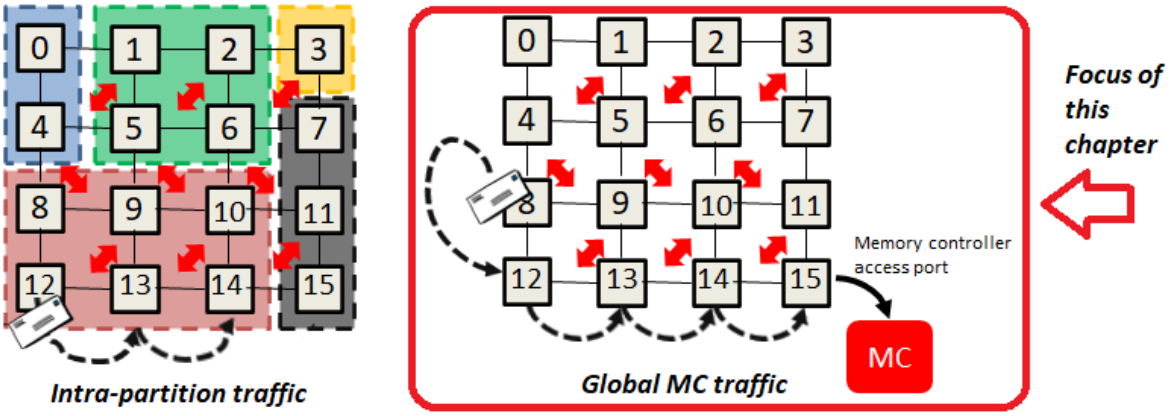



**Figure 4.1:** Focus of this chapter: isolation of global MC traffic.

In the presence of a parallel computing architecture, the on-chip network (NoC) is key to delivering strong domain isolation, since many of its internal resources are shared between packets from different domains [109]. Starting from the dual NoC architecture highlighted in Chapter 3, in this chapter we tackle the challenge of eliminating any form of interference in the NoC serving spatially-non-local communication flows, e.g., from spatial partitions to the memory controller, or vice versa (Fig.4.1). We target interference removal in the strictest sense: injection of packets from one domain/partition cannot affect the timing of packet delivery from other domains/partitions. This way, even timing channel protection is provided. This is also a requirement to enforce the strictest notion of platform composability, which implies not only that worst-case timing bounds can be given for message delivery, but also that the timing of message injection/propagation/delivery should not be affected at all by other domains [17].

### 4.1.1 Goal of this work

One straightforward solution that we have mentioned in chapter 2 is to statically schedule domains on the network over time with some form of time-division multiplexing (TDM). One fundamental issue is that communication performance in time-multiplexed NoCs is highly sensitive to the scheduling methodology of time slots. For use in Fog computing, this schedule should have a twofold characteristic. On the one hand, it should target low-latency communications, since one of the main reasons for IoT service providers for using Fog computing is the promise for low-latency responses. On the other hand, since we consider elasticity as the key management principle for future multi-tenant Fog computing, the dynamic reconfigurability of the schedule (i.e., of the number of domains or of the length of assigned slots) should be enabled from the ground up. Unfortunately, runtime reconfiguration of the generic global schedule is hard to make fast and cost-effective. In most cases, this implies solving a slot allocation problem in software or through hardware acceleration, and dealing with large programming tables at switches or network interfaces [61]. In other cases, invasive hardware

modifications are needed to optimally support different system configurations (e.g., number of domains) [10]. Thus, current TDM solutions are mainly well-suited for design time system configurations, which makes their extension for runtime flexibility problematic.

*The main goal of our proposal is to deliver the strong isolation property for the non-local communications of a runtime-configurable number of domains, while optimizing the latency of TDM communication flows across the entire configuration space.* Our solution provides a flexible architecture from the ground up, and targets high performance throughout the configuration space.

We build on the concept of *Token-based TDM*, which is based on the observation of the Channel Dependency Graph (CDG) defined by the topology and by the routing algorithm on top of it. From the CDG, we derive the generic requirements to have all input ports of all routers serving packets from the same domains at each time slot. This approach allows contention between packets from the same domain, while at the same time delivering secure-grade isolation between domains. Unlike similar works [10, 58], our scheme easily and efficiently generalizes to an arbitrary number of domains by selectively placing propagation stops at specific points in the NoC: this strategy preserves the strong isolation property while delivering more scalable communication performance than previous work.

Finally, we support non-invasive runtime modifications of the system configuration through the distribution of scheduling commands to network switches. In practice, our architecture runs unmodified with high performance regardless both static (switch and link latency) and dynamic (number of running domains) NoC settings.

## 4.2 Architecture instance

Figure 4.2 shows a tile-based multi-core architecture instance compliant with the architectural template of Chapter 3. Each tile includes a core, a private L1, a distributed L2 cache bank and a router. All elements are connected to the router. Routers from different tiles are connected building a 2D mesh topology. Without lack of generality, our baseline router is single cycle. More in detail, link traversal and input buffer storage are performed in one cycle, while VC allocation (VA), switch allocation (SA) and crossbar switching (X) are performed in another cycle. When a packet is stored in the buffer, the head information advances and is used by the routing function to compute the requested output port. Then, the packet uses the selected output port to allocate the output VC through VC allocation logic. Once the packet has acquired an output VC, it tries to gain access to the output port through switch allocation (SA). The winners of the SA (maximum one per output) traverse the crossbar switch and the packets are forwarded through the links. Cores access main memory by sending requests to the memory controllers (MC). Without lack of generality, we assume 4 MCs shared by all tiles.

The architecture is partitioned into disjoint domains, mapped to a subset of contiguous tiles (Figure 4.2). As a result, two traffic types emerge: intra-partition/domain traffic, and

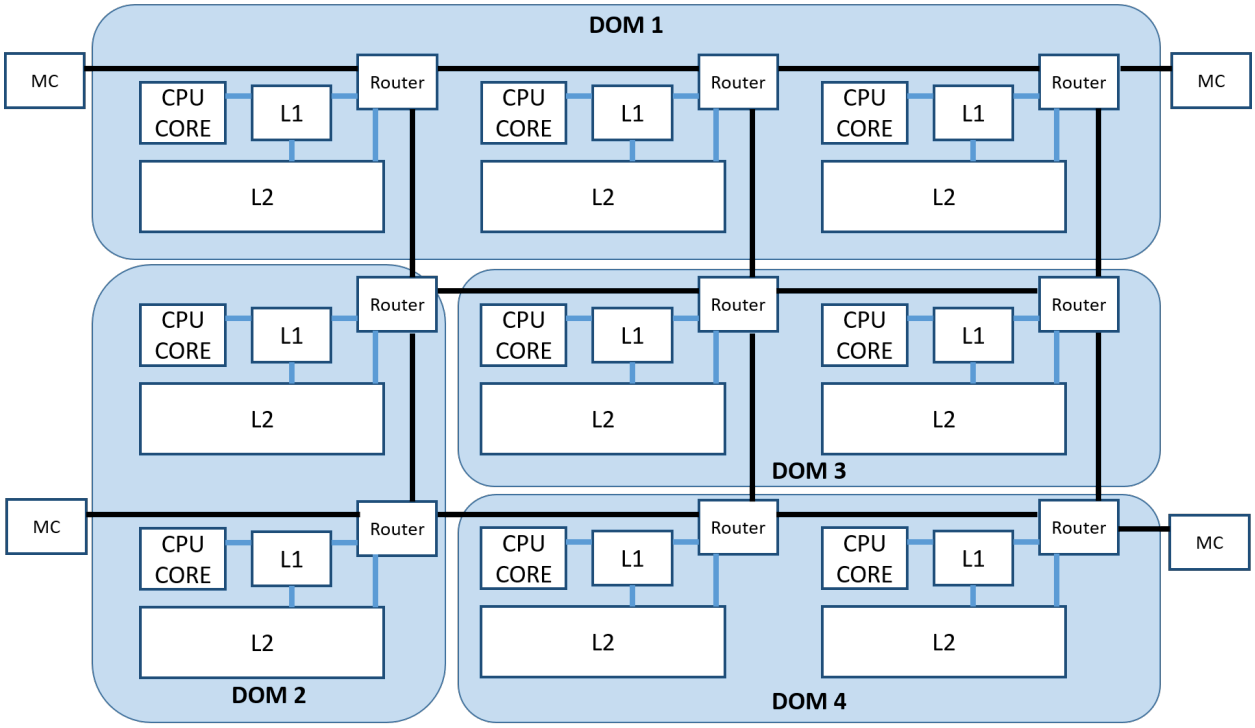



**Figure 4.2:** Multi-core architecture instance with a partitioning pattern into isolated spatial domains.

traffic from/to memory controllers and I/O ports breaking partition boundaries. This work targets the temporal isolation of the latter through a dedicated TDM NoC.

Like most scheduled TDM NoCs, the router provides a set of VCs (Figure 4.3) for each domain, so that flits of unscheduled domains in a given time slot or ungranted flits from the scheduled domain can be stored [10]. There are relevant buffer-optimized exceptions for ultra-low area realizations of TDM NoCs [104], however they trade performance for complexity, and may not be well-suited for the latency-critical Fog computing domain.

Packet routing is performed via logic-based distributed routing (LBDR) [120], which consists of a combinational logic at each input port. LBDR accounts for forbidden turns by the routing algorithm and for topology boundaries, coded in a configuration register. We assume deadlock-free deterministic routing, so that the channel dependency graph (CDG) can be assumed to be acyclic.

## 4.3 Baseline TDM NoCs

The reference solution to avoid any kind of domain interference consists of partitioning the virtual channels among domains **and** time-multiplexing the physical channels and crossbars sequentially among the different domains such that they are only allowed to propagate packets from different domains on consecutive time slots. This global TDM scheme is illustrated in Figure 4.4 (left), where the whole NoC is scheduled for moving packets of one domain at a given time slot.

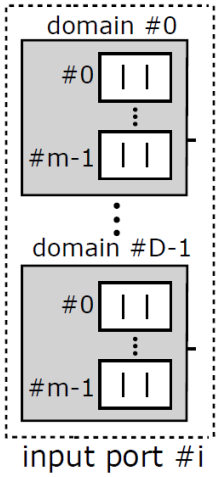



**Figure 4.3:** Partitioning of virtual channel buffers among D domains, with $m$ buffers for each domain, at a generic input port of a NoC switch.

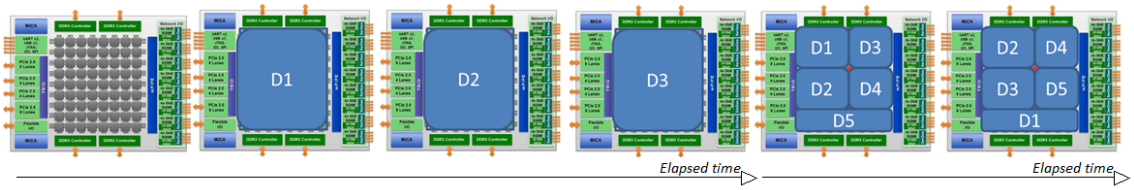



**Figure 4.4:** (left) Global TDM schedule. (right) Local TDM schedules, for which a spatio-temporal correlation is researched in this Chapter.

This schedule ensures that latency and throughput of each domain are completely independent of the other domain's load, and is the baseline schedule used in the experimental results for comparison. However, it is heavily sub-optimal and non-scalable. In fact, packets wait as many time slots as the number of concurrent domains minus one at each hop. The performance penalty grows with the distance of the receiver end node. *To cope with this problem, we revert to local schedules such as those illustrated in Figure 4.4(right), which give rise to waves of propagating domains.* For instance, when a packet moves from one switch to another during the time slot of its domain, it could find again a useful time slot in the receiver switch and keep moving. The problem consists of defining such waves, that is the spatio-temporal correlation of local schedules, in such a way that communication latency is minimized. We tackle this challenge by leveraging the knowledge of the CDG, as later explained.

The proposed TDM architecture could be tailored also to more resource-constrained platforms than the one presented in Chapter 3. In practice, it works also for unified NoC architectures where a single NoC is used to serve both intra-partition and memory controller and I/O traffic. Should this be the case, the awareness of spatial partitions would suggest a straightforward extension to baseline TDM, where intra-partition traffic would be enabled concurrently. For this, a TDM schedule could be designed where all domains/partitions would transmit local traffic during the same time slot, while a different time slot would be available for each domain to send or receive packets to/from memory controllers and I/O. In

fact, while strong isolation of global traffic (to/from MCs and I/O) would be delivered by the TDM mechanism and its optimized schedule, local traffic would be easily kept non-interfering by exploiting its spatial locality. This could be achieved in two ways:

- By defining rectangular or squared partition shapes together with shortest-path routing algorithms, so that packet routes stay within the spatial extension of partitions by construction.
- By using topology-agnostic routing algorithms on top of irregular partition shapes, yet enforcing that packets will never cross their boundaries. This solution, with its pros and cons, has already been discussed in Chapter 3.

In the experimental results, also the unified partition-aware schedule will be used for the sake of comparison with our approach, when used in tandem with regular partition shapes (e.g., see Figure 4.2. Therefore, we will assume the non-interference of intra-partition traffic by construction.

## 4.4 CDG-driven Strong Isolation

Our approach is to associate time slots with partitions/domains, and to change the phase of their oscillations to cut down on network latency based on the observation of the CDG, while preserving the strong isolation property.

### 4.4.1 Router-Level Strong Isolation

Token-Based TDM enforces non-interference between traffic from different partitions to/from the MCs and the I/O ports. Even cycle-level variations are prevented in order to avoid timing channel attacks.

To achieve such property, the network relies on a token propagation scheme. Tokens contain scheduling commands to local router-level domain schedulers. For this purpose, tokens carry a domain identifier (DI), which identifies the domain whose packets can be forwarded from a specific router input upon arrival of the token. *In order to deliver strong isolation between domains, we need to synchronize the timing of token propagation throughout the network in such a way that every router gets homogeneous DIs at each input port on every cycle.* The configuration of tokens at cycle "t" in Fig. 4.6 is an example of such synchronized token dispatch. This means the router will arbitrate and forward messages belonging to the same domain/partition, and messages from different domains will never compete. Tokens can be built with additional switch-to-switch wires, or be carried by special control packets. The former solution is adopted in this work.

### 4.4.2 Synchronized Token Propagation

The basic idea is to inject and propagate as many tokens as running domains, each with its own DI. All tokens are triggered back-to-back in sequence from a single node and propagated through the NoC following the CDG. In our initial implementation, at each NoC cycle a new token is injected and a token for the same domain is injected again after $D$ cycles (where $D$ is the number of domains). Tokens are received in sequence at each switch input port, and instruct the scheduler to serve packets belonging to domain DI from that port.

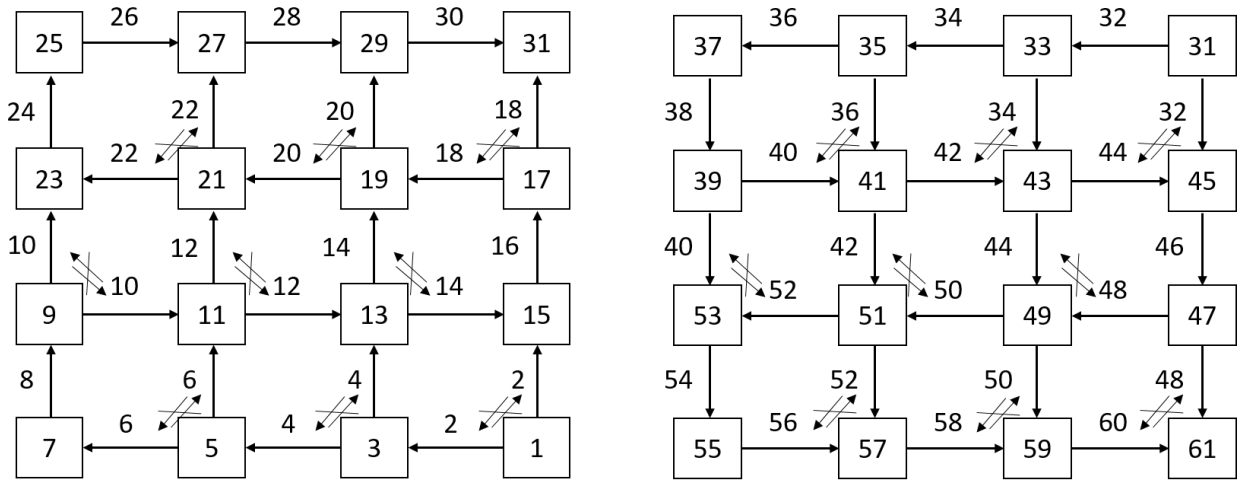



**Figure 4.5:** Network-level token propagation, with annotated latency, in the order of the CDG with periodic SR routing (dictating the position of routing restrictions) and single-cycle routers and links. Routing restrictions represent forbidden turns by the routing algorithm at hand for the sake of deadlock-free routing.

As mentioned, tokens traverse the NoC (router ports and links) following the CDG. Therefore, whether a router receives tokens with the same identifier on all its input ports on the same cycle or not, depends on the *nominal router and link latencies* and on the *dependencies set by the CDG (i.e., the routing algorithm)*. Concerning the former, we assume single-cycle routers and links in a 2D-mesh topology. Concerning the latter, without lack of generality we assume the Segment-based Routing algorithm (SR) from [21]. Following the analysis in [21], we infer segments in the 2D-mesh in such a way that bidirectional routing restrictions (i.e., forbidden turns by the routing algorithm for the sake of deadlock freedom) can be placed with a periodic regularity (see diagonal arrows in Figure 4.5), since this pattern has showed promising performance results. With the above assumptions, tokens would be triggered from the bottom right corner, and would be propagated throughout the CDG as illustrated in Figure 4.5. Numbers in the figure indicate token propagation latencies since initial injection time. Clearly, there are two token propagation phases, a scroll-up one (left) and a scroll-down one (right), which occur one after the other. Their combined effect is the traversal of all router ports and links just once (since the CDG is acyclic), in the order of the CDG.

Focusing on a single router, tokens with the same DI will reach its input ports at different timestamps. Let us define *relative latencies* as the time period elapsed between any two

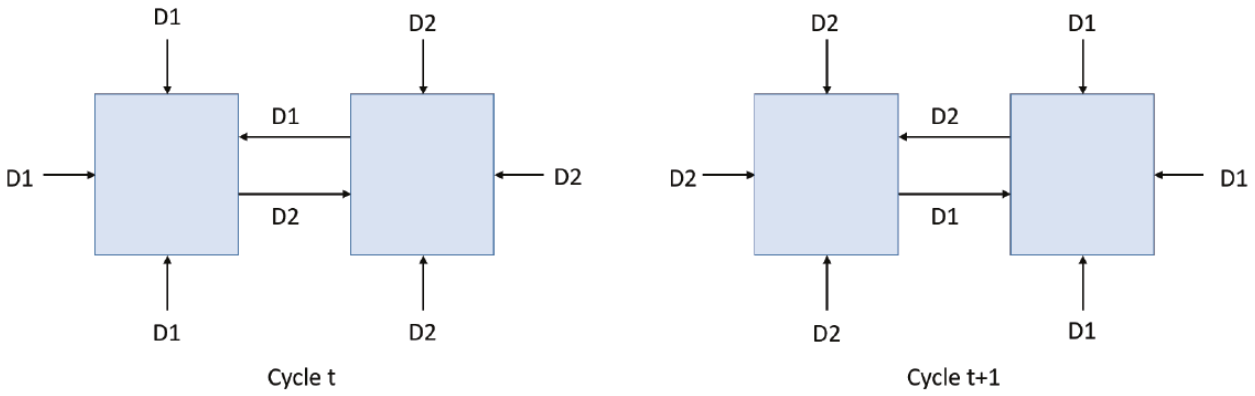



**Figure 4.6:** Perfect scheduling for minimum latency in a 2-domain scenario.

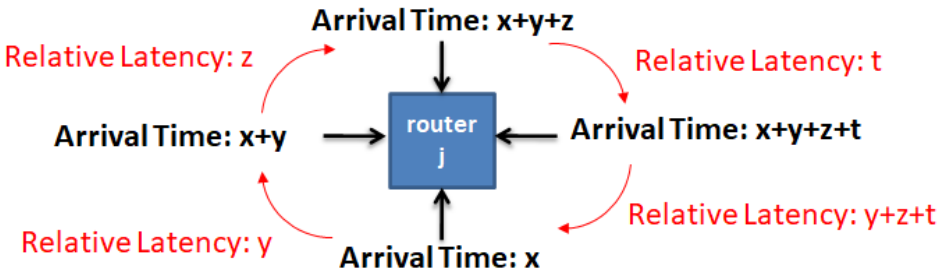



**Figure 4.7:** Generic relative delays for the arrival of tokens at the different input ports of a switch. The arrival order and the relative delays depend on the topology of the channel dependency graph.

consecutive such timestamps (Fig. 4.7). *Router-level operation with strong domain isolation requires that the result of the modulo operation between any of these relative latencies and the number of domains is always 0:*

$\forall(y,x) \in A \rightarrow (ATy - ATx)\ mod\ D = 0$, where $ATy > ATx$.

Where $A$ is the set of router input ports, $D$ the number of running domains, and $ATi$ the arrival time of same-DI tokens at input port $i$. In fact, as we inject the same domain identifier token into the network every $D$ cycles, a token with the same identifier must reach a specified port every $D$ cycles, or multiples thereof. Therefore, if the previous condition is met, *at regime all the router input ports will receive tokens with the same domain identifier exactly at the same time, and can thus work in strong isolation mode.* The number of domains $D_{ideal}$ that enables this operating condition is the Maximum Common Divisor (MCD) of all relative latencies between consecutive pairs of arrival times (in increasing order). For instance, if the three relative latencies of a 4-ported router (excluding the local port) are 4, 32 and 8 cycles, then strong isolation is ideally delivered with 4 domains.

To extend this property to the whole network, we first calculate the maximum number of domains for every router as the MCD illustrated above (if any). If such an MCD can be computed for each router, then the topology, coupled with the target routing algorithm, supports strong isolation of domains. In this case, *the MCD of all router-level computed domains is the ideal number of domains which delivers strong isolation for the network as a whole.* In the 2D mesh example in Fig.4.5, strong isolation is achieved with 4 domains.

As a shortcut, the ideal number of domains for strong isolation can be directly determined by the smallest relative latency inside the network, which corresponds to the latency $D_{ideal} = SCL$ of the smallest cyclic path spanned by a token in the network to reach two different ports of the same router. For instance, $SCL$ in Fig.4.5 amounts to 4 cycles.

Clearly, $SCL$ depends on router and link latencies, and is equal to: $SCL = (R-2) * (P+L)$ where $R$ is the number of switches in the smallest cycle, $P$ is the router latency and $L$ is the link latency.

When applied to 2D-meshes and periodic SR routing, a relevant benefit of this theory is that the composition of strongly-isolated router-level operations at network level through the CDG dependencies gives rise to a *Perfect Schedule*, which consists of the onset of unstopped propagating waves of naturally synchronized same-DI tokens throughout the network (see Fig.4.6), guaranteeing minimum-latency operation of the NoC[1] and matching the latency-sensitive nature of Fog nodes.

Figure 4.8 shows the token propagation flow over the network at regime, which is established once the first token comes back to the injecting router. Despite the 4 running domains, each router works in strong isolation mode, and globally a perfect schedule is established.

Our CDG-based methodology generalizes the constraints for perfect scheduling identified by the designers of PhaseNoC from the observation of local scheduling loops [10]. While their observation is good at efficiently handling specific system configurations, with our generalization it becomes possible to work out optimized solutions for the remaining cases as well, as hereafter illustrated.

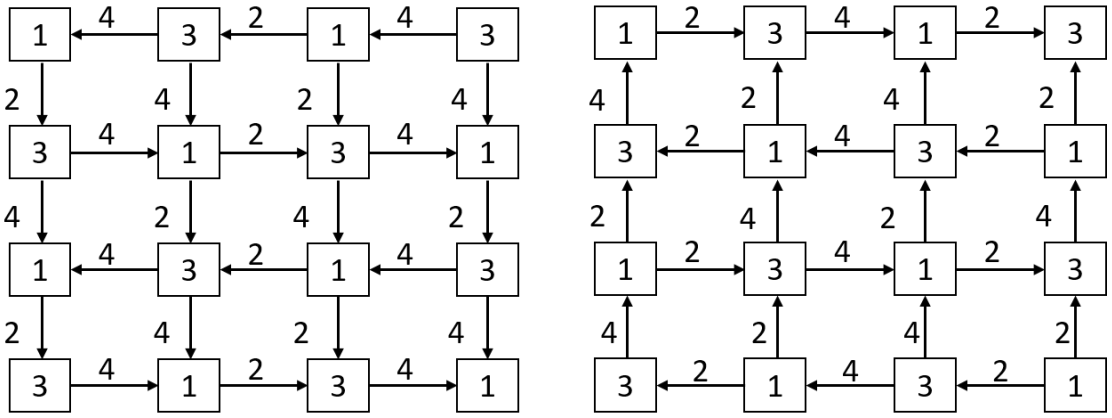
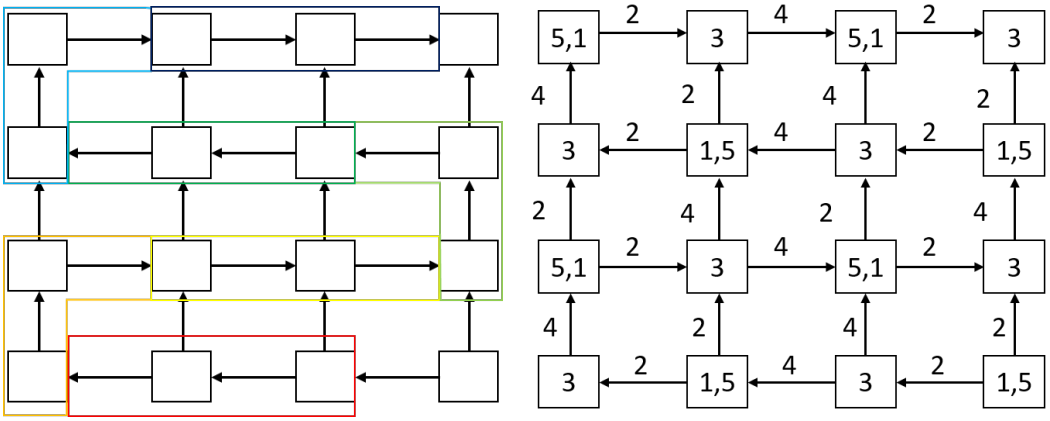


**Figure 4.8:** Token propagation flow at regime in a specific time slot across the scroll-down (left) and scroll-up (right) links. Numbers denote the token DI served on a specific NoC resource at that clock cycle. The assumption is to have 4 running domains.

[1]The property certainly holds for several different routing strategies, but evidence of this is left for future work.

### 4.4.3 Supporting a Higher Number of Domains

In order to handle a larger number of domains than $D_{ideal}$, PhaseNoC requires deep changes in the architecture, *thus proving unsuitable for runtime reconfigurability*. In particular, the pipeline depth of all routers needs to be increased, which would preserve strong isolation and perfect scheduling. As we will see in the experimental results, for some configurations this leads to suboptimal performance. Alternatively, the NoC can be split into communicating subnetworks, similar to SurfNoC. In this case, the strong isolation property is lost (i.e., domains can affect timing of packet propagation in other domains), unless area-expensive input speedup is implemented to avoid VC contention at crossbar inputs. Our CDG-inspired approach leads to more flexible and less invasive solutions, which provide better latency across the configuration space.




(a) Building subnets along the critical path of token propagation through the CDG.

(b) Selective stall placement to support 5 domains with strong isolation. Only scroll-up phase shown.

**Figure 4.9:** Extending the number of domains under strong isolation.

In particular, in a 2D mesh routed through periodic SR, all relative latencies are multiples of $SCL$. Thus, we can split the critical path of the tokens throughout the CDG into subnets of length $SCL$ clock cycles (see Figure 4.9(a)). With the ideal number of domains (which was found to be equal to $SCL$), all inputs to these subnets will be in the same domain at a given time slot.

In order to support a higher number of domains $D$, our intuition is that only $SCL$ domains should be in flight at any given point in time. The remaining $D - SCL$ domains should be stalled. This would enable to preserve the strong isolation property, while breaking the perfect scheduling assumption. As we will show in the experimental results, *preserving perfect scheduling at all costs may not be the best performing solution, as instead pursued by PhaseNoC.*

In order to implement this concept, we place domain propagation stalls selectively within subnets. The constraint to be met for correct operation is to place these stalls at the same positions within subnets (e.g., either in the first router, or in the second one). Figure 4.9(b) shows an example of stall placements to support 5 domains. Stalls allow the network to

synchronously receive the same DI at all router input ports.

Depending on the target number of domains $D$ and the ideal number of domains $SCL$, the number of stalls in each subnet can be set to $D - SCL$, if $D$ is larger or equal than SCL.

### 4.4.4 Supporting a Lower Number of Domains

If a lower number of domains than the ideal one needs to be used, then *strong isolation and perfect scheduling can be preserved provided $D$ is an integer divider of $SCL$.* This way the latency of the shortest cycle spanned by tokens to bridge two consecutive ports of the same router is a multiple of the repetition period of domain identifiers. Therefore, previous conclusions are still valid. As an example, with ideally $SCL = 12$ domains in a perfect schedule with strong isolation guarantees, the same property can be delivered with $D = 2$, 3, 4 and 6 domains.

With a different number of domains (e.g., 5 or 7), the same stall-based methodology can be applied, preserving strong isolation but not perfect scheduling. The following procedure must be used to re-align DI tokens at subnet inputs in the remaining cases where $D < SCL$:

1. Compute an integer $n$ such that $n \times D > SCL$.
2. Compute number of stalls per subnet as $n \times D - SCL$.
3. Enforce number of stalls at the same position within each subnet.

### 4.4.5 Heterogeneous Allocation of Time Slots to Domains

With our methodology, it is possible to allocate a different number of time slots to each domain. For instance, in a 3-domain scenario, the basic schedule $D1, D2, D3$ can be changed into $D1, D1, D2, D3, D3, D3$. Strong isolation is preserved if we configure the network for 6 "equivalent" domains instead of 3. Efficiency analysis and optimization of flexible slot allocation is left for future work.

### 4.4.6 Router Architecture

In a standard router with $M$ ports, $D$ domains and $V$ VCs per domain, there exist $M \times D \times V$ inputs and outputs VCs. However, our proposal only allows one domain to access the virtual channel arbiters at a time, then only $M \times V$ VCs perform virtual channel allocation (VA) at every cycle. Similarly, the $M \times D \times V$ to $M$ switch allocator (SA) in traditional routers is reduced in our implementation to a $M \times V$ to $M$ allocator.

Figure 4.10 shows our Token-based TDM router architecture. For the sake of better understanding, one input and one output port are shown, with one VC per domain. Hence, no VA is needed for domain processing. The showed router has buffering at both input and output ports, although input buffering only is also feasible.

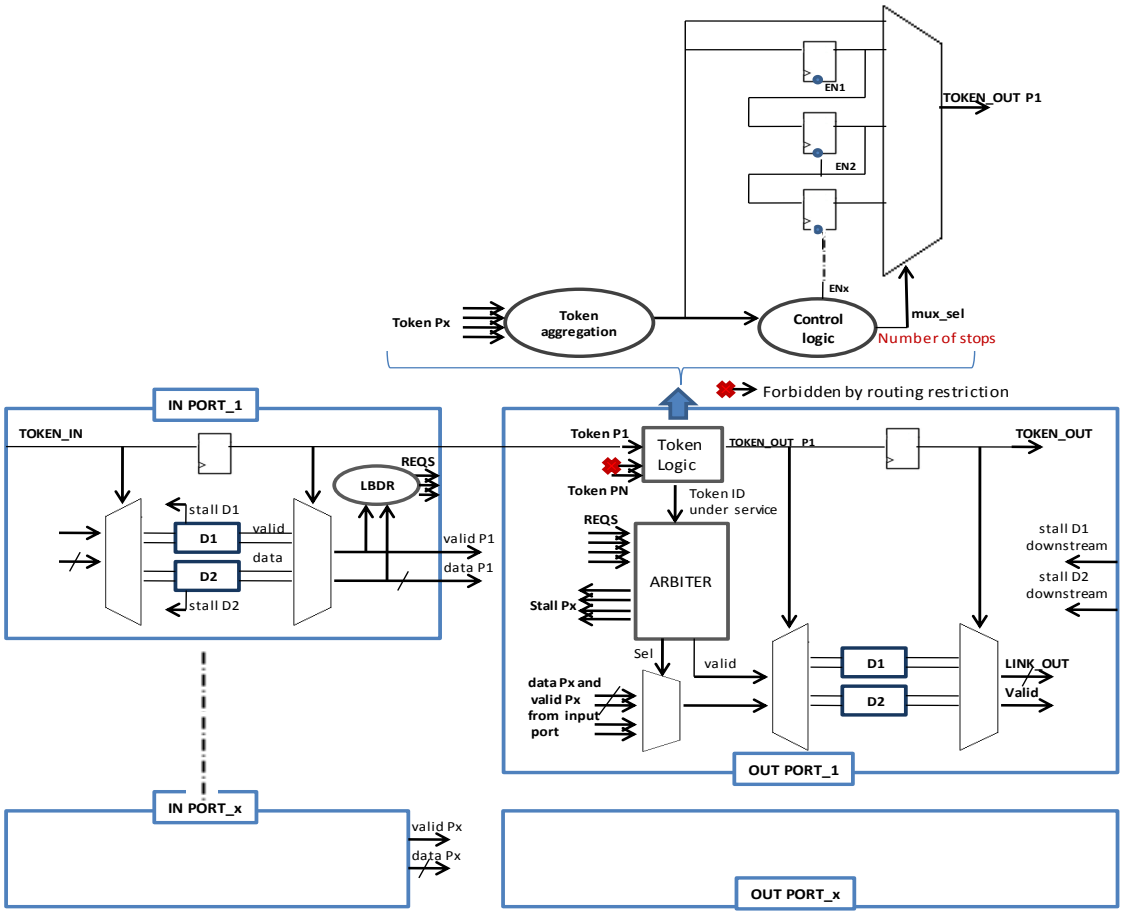



**Figure 4.10:** Proposed router architecture.

The incoming token ($TOKEN\_IN$) is used as the VC selector as it represents the domain buffer (VC) the incoming data flit (if any) will be stored into. The $TOKEN\_IN$ is stored in a retiming register, so that in the next clock cycle it will indicate (and select) the active domain within the router. Indeed, only one domain can access the routing/SA/crossbar resources within the router.

The Token Logic ($TL$) block is implemented for each output port as a modular block. It checks whether the same token DI is available at all input ports with routing dependencies with the associated output port. If so, the logic block forwards the token DI to the SA arbiter, and prepares an ouput token through that port with the same DI for the next cycle ($TOKEN\_OUT$). If not, the router is in transient state, and the logic operates as illustrated in Section 4.4.6.


**Figure 4.11:** Token structure.

### Programmed Stops at Routers

Figure 4.11 shows the token structure. Two control bits are used to provide different information to the router, when set. The $CNT$ bit is used to notify the number of hops before

a stop, while the bit $N$ notifies the number of domains to be configured. At bootstrap or at reconfiguration time, two "programming" tokens are sequentially injected: the first one with the $CNT$ bit set and the second one with the $N$ bit set. If both bits $D$ and $CNT$ are set to zero, the token transports a regular DI. The first programming token (named token $1A$) reports in the field $SET\_DELAY$ a decrementing counting value from the number of routers inside a subnet downto 0. At each hop, $SET\_DELAY$ is first decremented, then analyzed. If its value is equal to 0, then the router may need to implement programmed stops. For this purpose, the second programming token (named $1B$) reuses the field $TOKEN\_ID$ to carry the number of domains $D$ to be set.

When $D = SCL$ (where $SCL$ is pre-programmed into an internal register), no stops are needed. Istead, if $D > SCL$, the number of stops is equal to $D - SCL$, and is enforced by means of a programmable shift register that delays the incoming tokens accordingly (see upper part of Figure 4.10). Similarly, the number of stops is computed when $D < SCL$ based on the algorithm of the previous section.

When the $SET\_DELAY$ field is found equal to zero, the TL restores its value to the number of switches inside a subnet, so that stops can be reprogrammed periodically throughout the topology.

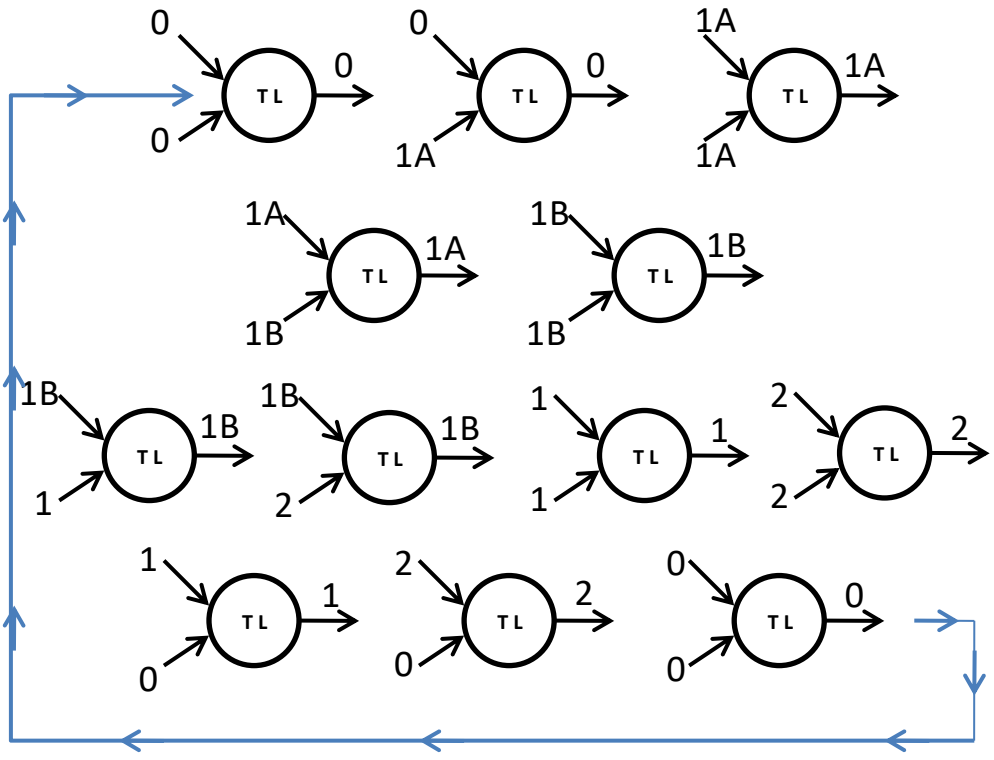



**Figure 4.12:** Token logic (TL) of a single router output port, featuring two input ports with routing dependencies with it.

### Handling the Transient State

Token $1A$ and token $1B$ are notified throughout the network in successive phases with a propagation timing throughout the CDG that is similar to that of the existing OSR protocol

[121]. In practice, an output port receiving one of these tokens keeps it stable till all the inputs with routing dependencies with that port get the same token. Then, the token is fired through the output port (see first row in Figure 4.12). Propagation of token $1A$ is completed when the bottom-right switch that has initiated it receives the very same token, which means the scroll-down phase has finished. Then, token $1B$ is injected, which follows the very same propagation rules (second row in Figure 4.12). When the trigger router receives $1B$ back, the number of stops has been programmed, and it can start injecting "scheduling tokens" (with associated data) with increasing DI (modulo $D$), for istance $D1, D2, D1, D2, ..$ for a 2-domain scenario.

In the third row of Figure 4.12, token $D1$ is received, to enable processing of domain 1. However, it takes some time before it appears on both inputs. Meanwhile, the token is dropped. When all inputs carry the synchronized token DI, then routing/SA/crossbar traversal are performed with packets belonging to that domain, and the token is forwarded as well.

The NoC can be inherently reprogrammed to support any number of running domains since the adaptation is not in the architecture itself, but in the scheduling commands sent through tokens.

In order to reconfigure the number of domains at runtime, in this thesis we assume static reconfiguration. In practice, the network is temporarily stopped by injecting a token DI=0. When all the network is idle (a condition that the trigger router can easily test when the null DI is received back), then programming flits $1A$ and $1B$ for the next configuration are injected. Please notice that strong isolation among domains is retained also during the reconfiguration transients. More performance-efficient dynamic reconfiguration methods will be explored in future work.

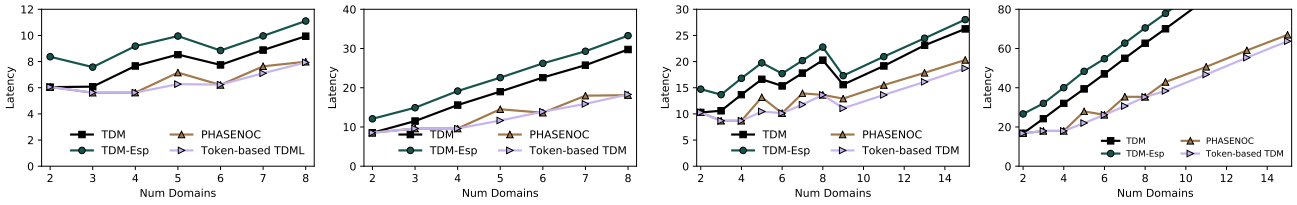
## 4.5 Experimental Results

We use the SystemC-based VirtualSoC virtual platform [37] to model our architecture with RTL-equivalent accuracy. Modelled topologies include $4 \times 4$ and $8 \times 8$ 2D-mesh.

We experimentally assess the most workload-intensive case, that is, the case where the TDM NoC is used both for intra-partition and for memory controller traffic (i.e., a unified NoC approach). We take it for granted that if the NoC works fine in this scenario, it can be applied to the dual NoC scenario of Chapter 3 as well, where it has to serve only MC traffic, thus giving it more bandwidth and lower latency.

We inject a mix of read/write transactions from the tiles to the distributed L2 banks of a partition (intra-domain traffic). Inter-domain traffic is obtained by injecting a mix of read/write transactions from domain tiles to memory controller nodes (MC traffic).

Several configurations of the 16-tile network are used, where it is split into a different number of domains. We evaluate four mechanisms across a different number of running domains, so to test the flexibility/scalability claim:




(a) Local traffic (b) MC traffic (c) Local traffic (d) MC traffic

**Figure 4.13:** Zero-load traffic for 4x4 2D-mesh (a-b) and for the 8x8 2D-mesh (c-d).

**1) Baseline TDM**. All domains are served in consecutive order. At each time slot, both intra- and inter-domain traffic for the associated partition is served.

**2) Partition-aware TDM (TDM-esp)**, described in Section 4.3, where a dedicated time slot is concurrently used for intra-partition traffic of all domains. Then, every partition has one additional slot to transmit or receive its own MC traffic.

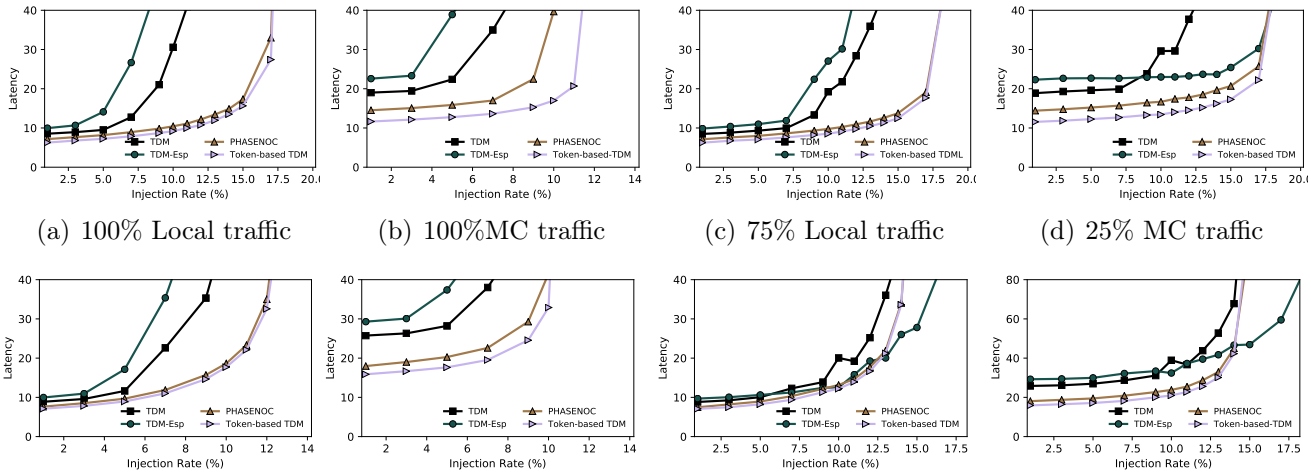
**3) PhaseNoC** [10]. For each configuration, we tune the PhaseNoC router with the proper number of pipeline stages for perfect scheduling and strong isolation (although this would be problematic to apply at runtime). However, in some cases PhaseNoC ends up in a suboptimal configuration. For instance, PhaseNoC with two pipeline stages per router can support 6 fully-isolated domains in perfect schedule, but if the required number of domains is 5, one domain would go unused. For a fair comparison, we allocate such an unused time slot to the active domains in a round robin fashion. With PhaseNoC, both intra-domain traffic and MC traffic are served when a domain is active. PhaseNoC also recommends network splitting into subdomains to handle critical cases. However, we verified that in this case the strong isolation property is lost, unless input speedup is implemented. We do not consider this case, since it would amplify the complexity gap between PhaseNoC and our approach.

**4) Token-based TDM**. Our approach, which serves both inter- and intra-domain traffic when a domain is active.

### 4.5.1 Zero-Load Latency

Figure 4.13(a) shows the zero-load latency results for local intra-domain uniform random traffic. The x-axis represents the number of domains and the y-axis the zero-load latency. PhaseNoC and Token-based TDM improve upon the baseline TDM variants. However, our proposal improves upon PhaseNoC in scenarios where PhaseNoC's pipeline has to be oversized for strong isolation (5 and 7 domains). The improvement on the network latency is 13% and 9% for 5 and 7 domains, respectively. In these scenarios, we claim "generalized perfect scheduling" through selective placement of stalls. In scenarios where PhaseNoC achieves perfect scheduling by increasing the switch pipeline depth (with 4, 6 and 8 domains), our proposal provides matched performance without changing the architecture. Note that the higher the number of domains the smaller the partition size. However, latency tends to increase because there is a higher waiting time to pay in the network interfaces to wait for the suitable injection time slot.

Figure 4.13(b) plots the zero-load latency results for MC traffic. The performance reached and the considerations are similar to the local traffic case. Token-based TDM improves network latency by 20% for 5 domains and by 12% for a 7-domain configuration.




(a) 100% Local traffic (b) 100%MC traffic (c) 75% Local traffic (d) 25% MC traffic

(e) 100% Local traffic (f) 100%MC traffic (g) 50% Local traffic (h) 50% MC traffic

**Figure 4.14:** Uniform traffic for 5 Domains (a-b-c-d) and 7 Domains (e-f-g-h).

Figure 4.13(c) shows the zero-load latency results for local intra-domain traffic in a 64-tile scenario. As in the 16-tile network, Token-based TDM improves upon PhaseNoC in scenarios where the pipeline is oversized. Even the reallocation of the unused slot is not able to compensate for this inefficiency. The improvement of Token-based TDM oscillates between 20% (5 domains) and 9% (15 domains). For MC traffic (Figure 4.13(d)), Token-Based TDM reaches the best performance improving up to 30% the PhaseNoC network latency. For a number of domains higher than 8, the cases where PhaseNoC gets perfect scheduling are not shown because in those cases Token-based TDM performs the same, as we have shown previously. Notice that, the higher the number of domains the smaller the benefit achieved by Token-based TDM as PhaseNoC decreases its bandwidth underutilization.

### 4.5.2 Load Curves under Perfect Scheduling

We computed the network saturation curves for local and MC traffic under uniform random traffic with four domains.

PhaseNoC and Token-Based TDM should deliver perfect scheduling in this context. In fact, we found that Token-based TDM exactly matches PhaseNoC performance, validating the following claim: Token-based TDM *matches the best case for PhaseNoC, while delivering extended flexibility and communication performance scalability, as proven next.*

### 4.5.3 Network Performance in Challenging Configurations

Next we analyze the network load curves in those cases where perfect scheduling does not hold. PhaseNoC increases the pipeline depth but needs to reallocate a redundant slot, while Token-based TDM inserts stops selectively throughout the network.

We analyze four scenarios in a 16-node NoC: one where the whole traffic is local, another with only MC traffic, and two scenarios with mixed traffic, one with 50% of each type of traffic and another one with 75% of local traffic and 25% of MC traffic.

Figure 4.14(a) shows the results for local traffic for the 5-domain scenario. PhaseNoC and Token-based TDM improve network performance of TDM. In addition, Token-based TDM reduces network latency of PhaseNoC by 10% along the complete injection range, reaching saturation at similar injection rates. Figure 4.14(b) plots the results for MC traffic. Similarly, Token-based TDM outperforms PhaseNoC by up to 20% before reaching saturation. Moreover, our approach increases network capacity. With a mixed-flow scenario, Token-based TDM reaches also the best performance. Figures 4.14(c) and 4.14(d) show results for local and MC traffic with 75% of local traffic and 25% of MC traffic. Token-based TDM reduces PhaseNoC latency by roughly 10% and 20% for local and MC traffic, respectively. Moreover, the customized TDM scheme (TDM-esp) has higher network latency compared with baseline TDM. However, for MC traffic TDM-esp matches accepted traffic before reaching saturation. This behavior occurs because for a given domain with mixed flows, TDM-esp provides higher bandwidth than baseline TDM, as each domain has two time slots, one for intra-domain traffic and one for inter-domain traffic (for access to memory).

Similarly, for the 7-domain scenario (Figure 4.14(e-h)), Token-based TDM reduces network latency by 8% in intra-partition traffic scenario and 15% in MC traffic scenario. For 50% mixed-flow scenario (Figures 4.14(g) and 4.14(h)), Token-based TDM gets the minimum network latency, improving PhaseNoC by 7% on local traffic and 12% on MC traffic. However, as seen above, TDM-esp achieves higher throughput.

Finally, we found that results obtained for the network of 16 nodes can be directly extrapolated to the network of 64 nodes.

The fundamental explanation for the performance speedups documented above is that while Token-based TDM introduces a configurable number of stops for DI tokens at specific routers, PhaseNoC spreads the latency overhead everywhere.

## 4.6 Wrap-up

In this chapter we explained how Token-based TDM offers a flexible TDM NoC architecture that can support a variable number of domains beyond design time configuration, while preserving low-latency communications. Our philosophy is not to update the switch architecture to the execution scenario, but rather to change the scheduling commands sent to the NoC via a token-based notification mechanism. Through extensive performance benchmarking against state-of-the-art PhaseNoC, we documented the capability to match PhaseNoC when perfect scheduling can be applied, while consistently reporting better performance in the remaining case (generalized perfect scheduling) by exploiting the properties of the CDG. Therefore, our approach enables smooth reconfigurability while combining it with better performance scalability. In future work, implementation of the routers under test into industrial

technology libraries will enable timing, area and power analysis.

## 4.7 Deployment for Fog Computing

For the sake of deployment within a Fog hardware platform, we observe that a unified NoC approach would end up penalizing the intra-partition traffic too much. In fact, most of the time slots would be devoted to MC traffic, and the bandwidth reserved for local communications would be largely insufficient. Similarly, NoC sharing among several kinds of messages would lead to a latency degradation for MC traffic, which is at odds with the latency-critical requirements of a Fog computing environment. As a result, we anticipate the use of our Token-Based TDM NoC as the Memory Controller dedicated NoC, as stated in Chapter 3, in the context of a dual-NoC architecture. The companion NoC would be instead devoted to spatial partitioning, and is addressed in the next Chapter.

# Chapter 5

# A Routing Mechanism for Flexible Space Partitioning

This chapter motivates the use of spatial division multiplexing for the isolated execution of consolidated IoT services. Then, a flexible and dynamically reconfigurable routing framework is proposed for the on-chip interconnection network, whose effectiveness for the fast reconfiguration of the partitioning pattern is demonstrated in the final experimental results. This work stems from a joint collaboration with Universidad Politecnica de Valencia, and with prof. Josè Flich in particular. We also acknowledge the seminal work of former student Marco Balboni at UNIFE, and prof. Andrea Marongiu for the useful discussions.

## 5.1 Introduction

Previously, we demonstrated that Fog platforms are challenged by new design requirements, encompassing low-overhead sharing mechanisms among IoT services, the strong isolation of their execution environments and adaptive resource allocation to them, matching the inherent dynamic nature of most IoT applications.

In the previous chapter, we have explained how our Token-based TDM approach satisfies the strong isolation requirement for MC traffic, revolving around a low-latency schedule and the capability of its dynamic reconfigurability.

In this Chapter we face the problem of spatially isolating the multiple IoT services that will run concurrently in the PMCA. In fact, in Chapter 1 we anticipated that assigning all parallel compute and memory resources to applications is not always reflected into congruent execution speedups in Edge computing. Most applications exhibit a performance saturation as the level of hardware parallelism is increased. As an example, Figure 5.2 shows the scalability of several image processing kernels offloaded to a PMCA consisting of 9 clusters of 8 cores each [24]. The Y-axis shows speedup versus kernel execution on a single cluster (8 cores); the X-axis shows the number of clusters used. For this experiment, ideal memory and interconnection are considered (i.e., each load/store is handled in 1 processor cycle), since the only interest is in observing the scalability of the kernels themselves. As depicted

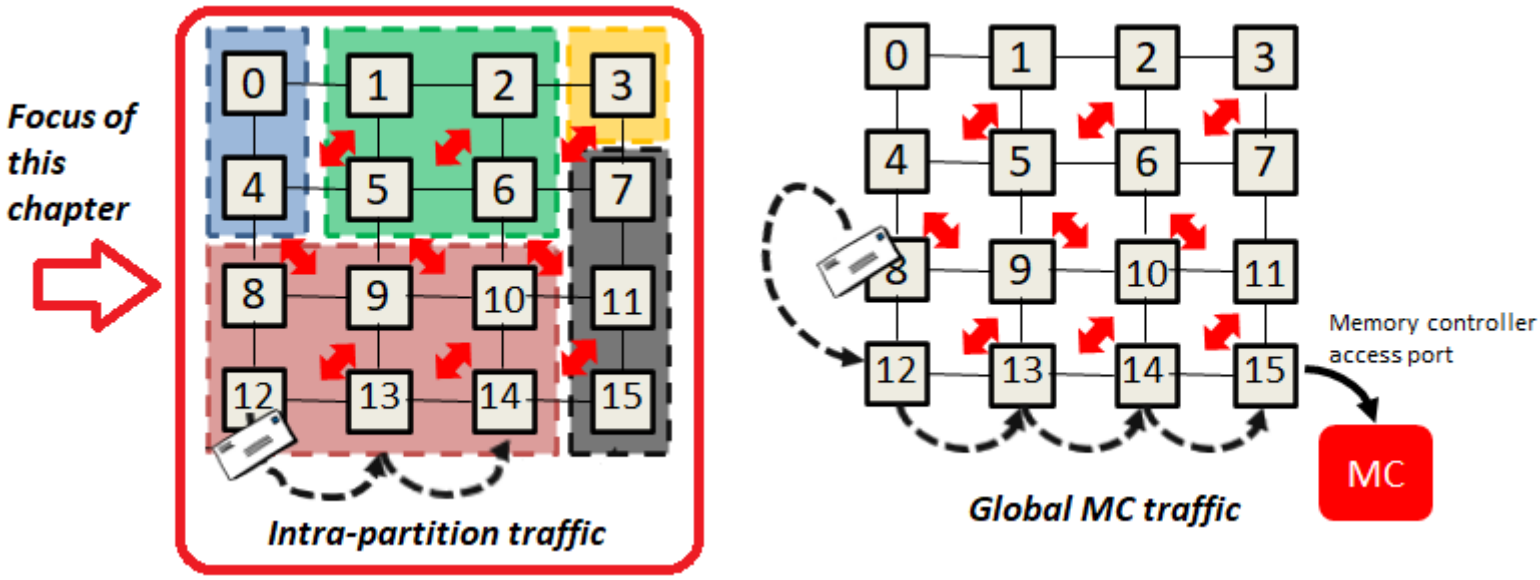



**Figure 5.1:** Focus of this Chapter: spatial isolation of compute and memory partitions.

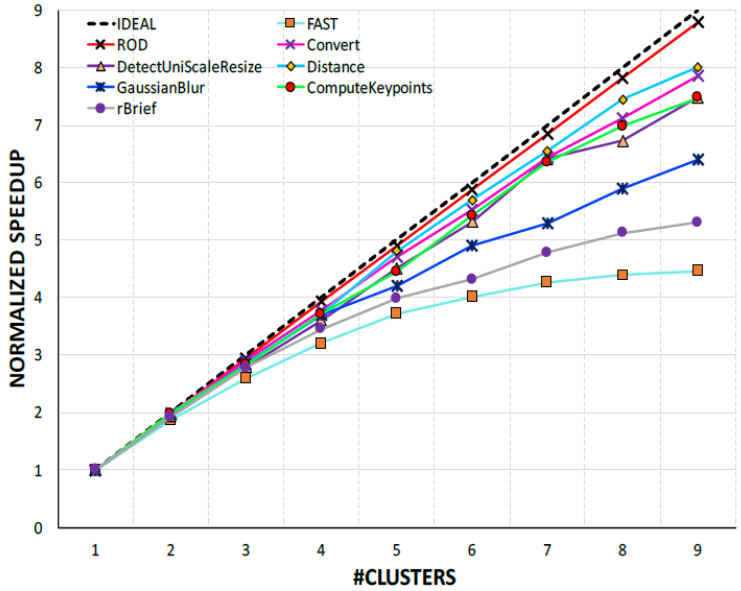



**Figure 5.2:** Performance saturation with increased hardware parallelism.

in the graph, there is one benchmark (*ROD*) that presents a speedup very close to the ideal case (it contains abundant *embarassing parallelism*). Four benchmarks (*Distance*, *Convert*, *ComputeKeypoints*, *DetectUniScaleResize*) have an almost ideal speedup for a low numbers of clusters but it starts to saturate for high cluster (core) counts. Three benchmarks (*GaussianBlur*, *rBrief*, *FAST*) emphasize a plateau in the speedup beyond 6 clusters.

When we combine these results with the observation that using the whole mesh would result in a very high average and worst-case latency for each L2 access, the outcome is that in the context of Fog computing a better use of manycore accelerators might be that of allowing multiple offloaded kernels to co-exist in the PMCA at a given time, namely to adopt *space-division multiplexing* (SDM) policies.

Topology-overlapped compute/memory partitions should be spatially isolated from one another for the sake of composability, predictability and security. In particular, the on-chip

interconnection network plays a pivotal role to enforce such spatial isolation, in that all possible mutual interferences can be removed through proper control of the routing paths.

The specific focus of this chapter is thus on the on-chip interconnection network of the PMCA, which will provide hardware support to split the PMCA into strongly-isolated compute and memory partitions, and for safely and efficiently reconfiguring their shapes, size and location at runtime.

### 5.1.1 Which NoC routing mechanism for isolation?

The key intuition behind this work consists of identifying the fundamental role that the routing mechanism of the NoC plays with respect to fulfilling the efficient, flexible and secure sharing requirements of the PMCA among running services in the target Fog node. This intuition stems from a twofold observation:

- A naive approach to spatial-division multiplexing (SDM) consists of the controlled mapping of those tasks that belong to the same service onto nearby cores, thus forming a partition. However, the lack of hardware-supported isolation prevents communication packets logically pertaining to one partition from interfering with those from nearby partitions, as already illustrated in Chapter 3 (Fig. 3.3(c)), thus raising vulnerability concerns (e.g., denial of service attacks, formation of timing channels) and breaking composability. This effect becomes unavoidable as partition shapes become irregular to maximally exploit available resources and to flexibly reassign them with fine granularity at runtime.

- Matching the context-sensitive computation requirements of services implies the runtime evolution of the size and shape of the associated compute and memory partitions. State-of-the-art platforms have been designed for more static scenarios, hence they are not very efficient at the dynamic reconfiguration of resource assignments. Current routing technology for the NoC significantly contributes to this inefficiency. In fact, a global and statically-defined routing algorithm for the network as a whole is not compatible with a reconfigurable partitioning pattern at runtime, since the latter is equivalent to frequent topology changes (i.e., each isolated partition can be logically viewed as a self-contained smaller-scale topology to be routed). The most straightforward solution of recomputing the routing function for each partition as its configuration is changed leads to unacceptable runtime overhead.

### 5.1.2 Goal of this work

In order to tackle these challenges, our solution revolves around an extended logic-based distributed routing framework capable of inferring isolated compute and memory partitions in a cost-effective way, and of dynamically reconfiguring the shape and size of such isolated partitions with overly limited overhead. For this purpose, a new routing logic is proposed that

autonomously finds suitable paths to destinations without requiring the costly modification of the programmed routing function as partition configuration is evolved.

We named our framework *Partition-Enabling Logic-Based Distributed Routing (pLBDR)*, since this low-overhead routing mechanism builds upon past work on logic-based distributed routing [120], but exhibits a distinguishing key feature to support flexible and isolated partitioning: it guarantees partition routability leveraging a unique, unaffected global routing function. Thus, as the partition configuration needs to be modified at runtime, only the new partition boundaries need to be reprogrammed in the routing mechanism (limited to just a single 4-bit register per switch), while avoiding any routing path management overhead in software and in hardware. For the sake of fast reconfigurability, the proposed mechanism poses some restrictions to the irregularity of legal partition shapes. Nonetheless, this is a significant step forward with respect to state-of-the-art approaches that, in the lack of hardware-assisted isolation, prevent partition interference by restricting feasible shapes to squares and rectangles only. Moreover, many of the infeasible partition shapes would hardly be considered by system designers anyway, due to the network congestion they cause on some specific links or to the large intra-partition communication latency that might affect mapped tasks.

In the following section, we will detail the implementation of our routing mechanism. Then, we will bolster its robustness using a mathematical proof. Finally, we will provide a comparative study with state-of-the-art table-based routing, coupled with a topology-agnostic routing algorithm to derive a suitable routing function for each partition shape at hand.

## 5.2 Our Approach: pLBDR

In this section, we present the fundamental feasibility limitations of the LBDR routing mechanism, for which background was provided in Chapter 2. Aware of the inherent benefits of LBDR over routing tables, such limitations were an incentive to augment vanilla LBDR with more flexibility. The improved routing mechanism, pLBDR, will be then elucidated and demonstrated by a mathematical proof.

### 5.2.1 Enforcing Partition Boundaries

This work builds on vanilla LBDR, and thus accepts its fundamental feasibility domain: the *legal partition shapes* (i.e., routable with LBDR) are those where minimal path routing with respect to an ideal fully-connected network including the partition is feasible (see Fig.5.3). Based on this definition, U-shaped partitions and their derivatives are not legal, and so are partitions with holes. The reason is because they would end up routing packets from sources to destinations located at opposite flanks of "U"-shaped regions by taking routing paths contained within the spatial boundaries of the irregular partition for interference-free communication. Unfortunately, such routing paths would be "minimal" (i.e., requiring the

minimum number of network hops) for the partition, but not for the 2D-mesh as a whole. As a result, vanilla LBDR can not route packets across such globally non-minimal routing paths.
In contrast, T-shapes, L-shapes and their variants are all legal ones since the number of hops stays minimal, both locally and globally, for each routing path within the partition boundary.
The above restriction is reasonable since it is nonetheless a significant flexibility extension with respect to traditional approaches relying on squares and rectangles for interference freedom (e.g., [68]).
In any case, system designers would hardly pick up such illegal shapes, since communicating tasks placed at the opposite extremes of the partitions would end up incurring large communication latency. At least, he would have to deal with more restrictive task mapping constraints on the partition at hand.

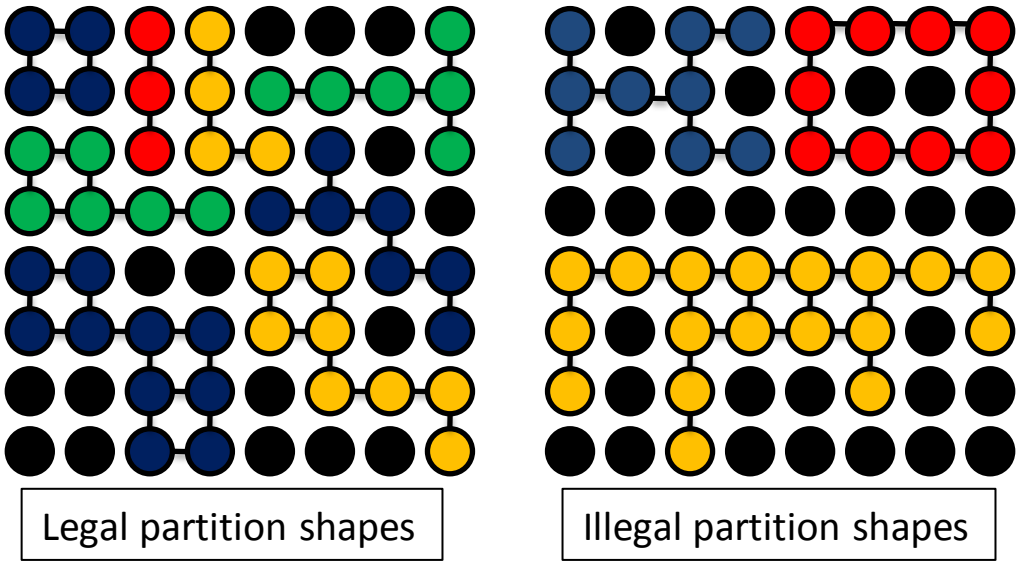



**Figure 5.3:** Legal partition shapes are those where minimal-path routing is feasible.

Despite its flexibility limitations, vanilla LBDR can effectively constrain routing paths within partitions: for this, connectivity bits can be programmed in a configuration register per switch to model partition boundaries. If an output direction is specified as virtually non-connected by such register, the routing logic would never select it as a productive output port for packets.

This solution provides relevant disruptive advantages for spatial partitioning:

- First, *partitioning can be implemented on top of a NoC without changing its global routing algorithm, or without instantiating differentiated routing functions for each partition. In fact, the algorithm is modified only when the routing restrictions are modified, not the connectivity bits*.

- Second, no additional mechanism is needed to provide freedom from deadlocks, since

this is delivered by the unmodified routing algorithm. Especially, no virtual channel is needed, thus keeping the network extremely simple.

- Finally, connectivity bits can be reconfigured at runtime by a fabric controller or by the host processor by means of the dual TDM NoC illustrated in Chapter 4.

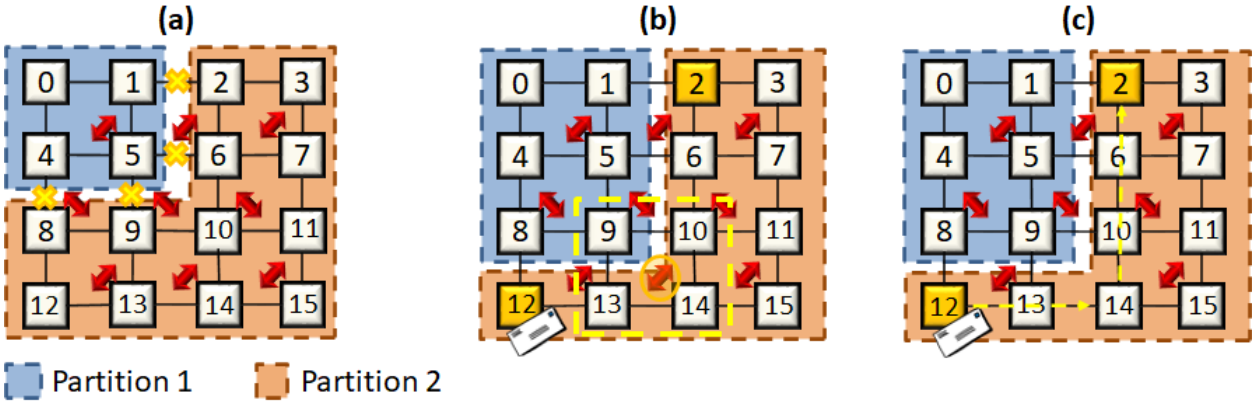



**Figure 5.4:** (a) Global routing function. (b) Selective restriction masking. (c) pLBDR to flexibly partition Manycores.

In Fig. 5.4 (a), 2 partitions are inferred by setting the connectivity bits of the border switches either to 1 if the router is connected to a direction still within the partition, or to 0 if it is connected to a partition boundary. For example, the connectivity bits of router 5 to the north, east, south and west are 1, 0, 0 and 1 respectively.

*Unfortunately, LBDR incurs a fundamental limitation to support flexible partitioning of many-core devices: some partition shapes may turn out not to be routable although they belong to the feasibility domain.* The reason lies in a mismatch between the unmodified global routing function and the partition shape to be inferred on top of it, as illustrated in Fig.5.4 (b). Because of the routing restrictions at switches 14 and 15, no packet from the south-west flank of partition 2 can reach the nodes to the north. For example, no packet coming from router 12 can reach router 2. The opposite is also true.

The problem could be naively solved in one of the following ways:

- By forbidding partitions like this (although they belong to the feasibility domain of LBDR), thus further restricting flexibility.
- By modifying the global routing function on-the-fly. However, previously-feasible partitions may turn out not to be feasible any more.
- A new routing algorithm could be recomputed for each partition, which would waste the LBDR's advantage over table-based routing.

### 5.2.2 pLBDR

The fundamental idea of this approach is to preserve a global and unique routing function for the network as a whole and to infer partitions on top of it, while fully exploiting the LBDR feasibility domain and avoiding the invalidation of running partitions. This global routing function is presented in Fig. 5.4(a).

The basic intuition is that some routing restrictions make sense for the network as a whole, but they uselessly limit routability at the partition-level. In particular, once partition boundaries are set, some routing restrictions end up preventing the formation of cycles in the channel dependency graph (i.e., deadlock) that include links and/or switches of different partitions, or unused inter-partition links (see the cycle formed by switches 9, 10, 13 and 14 in Fig.5.4(b)). In practice, such cycles will never occur, since the cyclic dependency is already broken by the inter-partition connectivity bits. As a result, in Fig.5.4(b)) the routing restriction at switch 14 is useless for routing of Partition 2, while the one at switch 15 should be enforced, since it prevents routing cycles inside the partition (e.g., switches 10, 11, 14 and 15). By masking the former, a legal and safe routing path would connect the west and north partition flanks through switch 14. Thus, the packet will be routable from switch 12 to switch 2, and vice versa (Fig.5.4(c)).

Our proposal is thus to augment LBDR logic with the capability to recognize the useless routing restrictions that can be ignored while preserving deadlock-free intra-partition routing. We express such an extension with the following behavioral rule that we enforce at each switch:

*IF direction x is a partition boundary THEN*
*at next hop in direction y ignore forbidden turn to direction x*
*AND*
*at next hop in direction z ignore forbidden turn to direction x*

where
$x, y, z \in \{North, South, East, West\}$
and
$(y, z) = (East, West)$ if $x =$ *North or South;*
or
$(y, z) = (North, South)$ if $x =$ *West or East;*

### 5.2.3 Routing Strategy

We select segment-based routing (SR) [86] as the routing algorithm of choice. When used with pLBDR, it serves as the global routing function for the network as a whole [86]. At the end of this Chapter, it will be used with table-based routing as well to set up a state-of-the-art solution for partition reconfiguration, and for comparison against pLBDR. In that case, routing functions need to be custom-tailored to each partition shape individually, thus SR

serves as a topology-agnostic routing framework, capable of providing suitable routing paths for arbitrary partition shapes (see Chapter 2).

With SR, the network is split into disjoint sets of interconnected switches and links called segments. All network links and switches belong to one and only one routing segment. In general, three kinds of segments can be defined in a generic topology:

- Initial segments are those that start and end on the same switch, thus forming a cycle (see $s1$ in Fig.5.5).
- Regular segments instead start with a link, contain at least a switch, and end with a link (see $s2$, $s3$, and $s4$ in Fig.5.5).
- Unitary segments consist of only one link.

Every routing segment, except for the initial segment, starts and ends on a switch already part of a computed segment (see $s2$, $s3$, and $s4$ in Fig.5.5).

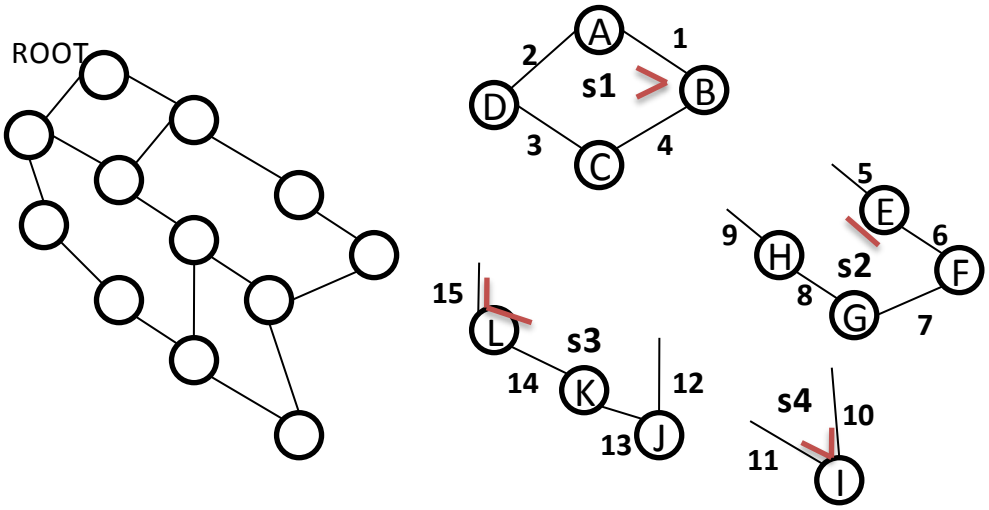



**Figure 5.5:** Segmentation of an irregular topology, and placement of routing restrictions inside segments for deadlock-free routing.

The next step consists of placing routing restrictions locally inside each segment. Since segments are independent, then routing restrictions can be placed inside segments independently from one another. Routing restrictions break potential dependency cycles within the initial segment, and through all the successive segments added to it [86].

With SR on top of traditional table-based routing, implementing the baseline methodology in Chapter 2 for partition reconfiguration implies the following steps: (i) the runtime computation of segments for the new partition shape, (ii) the assignment of routing restrictions to the segmented partition, (iii) the search for source-to-destination paths and (iv) the derivation of updated routing tables, in addition to their reprogramming.

Since pLBDR uses a unique global routing function and infers partition boundaries on top of it, it searches for segments on a complete 2D-mesh topology only once at design

time. Thus, it has no runtime software overhead for path management. An additional advantage is that *it can exploit knowledge of the baseline topology and of its properties for optimized search*. Following the analysis in [86], we infer segments in the complete 2D-mesh by using only initial and regular segments, and opt for a partitioning pattern that places bidirectional routing restrictions with a periodic regularity, and that has showed promising performance results. In particular, this pattern places routing restrictions always between any two contiguous directions of a switch, and never between two opposite ones (e.g., from south to north or from east to west). Instead, table-based routing searches for segments inside each partition, thus it has to undergo the compute-intensive segmentation methodology in [86] each time a partition shape changes during the system lifetime.

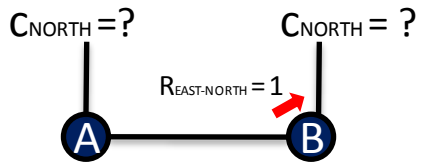



**Figure 5.6:** Scenario considered for the deadlock-freedom and connectivity proof of pLBDR.

### 5.2.4 Deadlock freedom of pLBDR

In partitions where minimal path routing is feasible (with respect to the ideal 2D mesh), pLBDR will always find a deadlock-free routing solution to connect any source-to-destination pair within the partition.

To prove the deadlock freedom and connectivity properties, we focus on two connected switches A and B, and assume that switch B has a routing restriction to direction $y$, which is seen by switch A as $R_{xy} = 1$, where $x, y \in \{North, South, East, West\}$. This is pictorially illustrated in Fig. 5.6 for $x = East$ and $y = North$, but spanning $x$ and $y$ gives all the possible situations in a real-life topology, given our optimized segmentation pattern in a 2D mesh. Next, we wonder whether switch A can safely ignore $R_{xy}$ or not for routing computation. We can distinguish three cases:

**CASE 1:** In this case we assume that connectivity bit of switch A in direction $y$ is equal to zero ($C_{y_A} = 0$), while the connectivity bit of switch B in the same direction does not matter. In this situation we need at least a switch outside the partition to close a cycle (at least the one connected to A in direction $y$), which will never happen, since no partition can invade the others. Thus, the routing restriction can be safely ignored, and this is exactly the situation that pLBDR captures. It is also possible to form larger cycles when the partition wraps the unconnected node in direction $y$ of A, but we explicitly omit partitions with holes from the ground up. Therefore, this routing restriction should be ignored because it unnecessarily restricts routing options to prevent a deadlock condition that will never occur. pLBDR routing logic exactly captures this case. As a result, overwriting this routing restriction by p-LBDR will not affect the routing since the cycle is created outside the partition.

**CASE 2:** In this case we assume that $C_{y_A} = 1$ in switch A and $C_{y_B} = 1$ in switch B. In this situation, the switches connected in direction $y$ of A and B belong to the partition, and so will be their interconnecting links. As a result, the routing restriction prevents a possible cycle inside the partition and should not be ignored. In fact, pLBDR is triggered only by $C_{y_A} = 0$, and correctly accounts for this restriction.
**CASE 3:** In this case we assume $C_{y_A} = 1$ in switch A and $C_{y_B} = 0$ in switch B. The routing restriction would prevent partition routing only in case there were destinations in the $y - x$ quadrant, which would imply a U-shaped partition or its derivatives. Hence, this can never happen with the stated assumptions. pLBDR accounts for this restriction since $C_{y_A} = 1$, but routability is not jeopardized.

## 5.3 Experimental Results

The first part of our experiments assesses the hardware and software overhead for establishing configurations, while the second part performs analysis of individual legal partitions of a 2D-mesh topology to analyze the execution efficiency of pLBDR. All results are compared to the Table-based routing baseline solution, using SR as the partition-specific routing function and going through the baseline reconfiguration methodology highlighted in Chapter 2.

### 5.3.1 Hardware Reconfiguration Cost

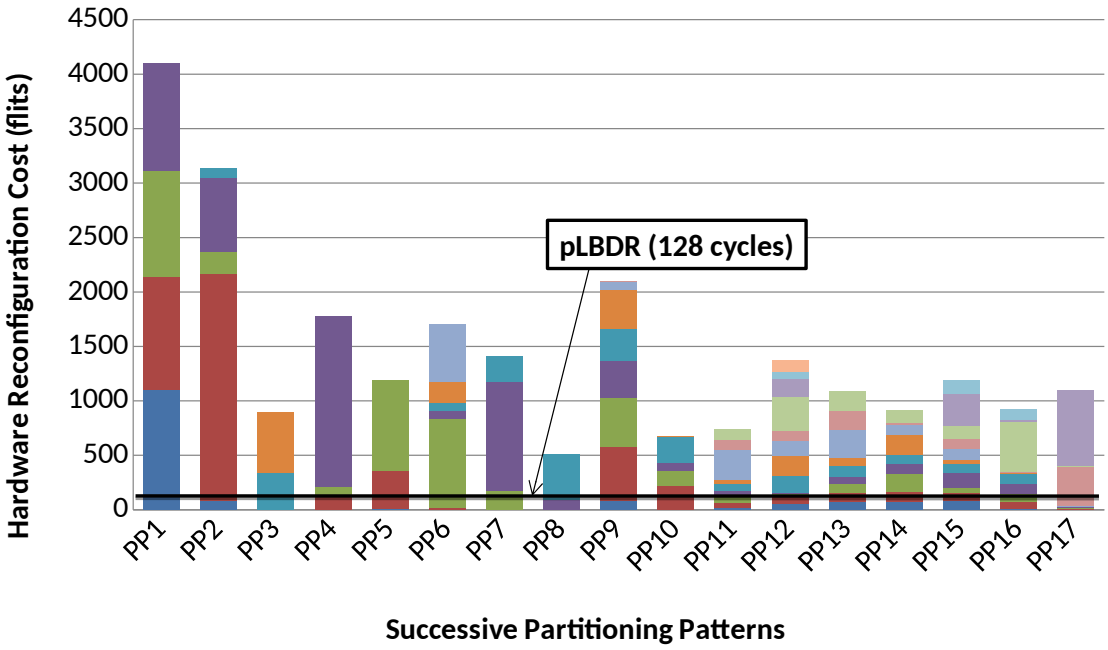



**Figure 5.7:** Hardware reconfiguration overhead for 17 partitioning patterns, with the partition-level breakdown for each pattern.

We consider 17 partitioning patterns of an 8x8 2D-mesh topology. Each pattern is derived from the previous one by randomly reshaping existing partitions, and/or activating or

deactivating new/old partitions. Scenarios are manually hand-crafted to target high context sensitivity and fine-grained resource remodulation in future shared Fog nodes (e.g., the number of partitions ranges from 2 to 11). The hardware configuration cost is reported in Fig.5.7, in terms of the number of flits[1] it takes to reprogram routing tables or pLBDR configuration registers through the TDM NoC (cumulative and breakdown for each partition) when transitioning from one scenario to the next. The overhead for packet headers is considered. For table-based routing, in a partition of $N$ clusters only $N-1$ entries are updated in each routing table, and only in case those entries differ from the old configuration by at least one bit. As a result, programming packets for each network interface have variable lengths. For pLBDR, connectivity bits are always re-programmed, hence 1 programming packet is sent to each switch.

The hardware overhead for table-based reconfiguration ranges from 500 to 4000 flits depending on the partitioning pattern, with an average value of 1476 flits. For comparison, pLBDR requires 128 flits only, which is 74% lower in the worst case, and 96.8% lower in the best case.

### 5.3.2 Algorithm Recomputation Overhead

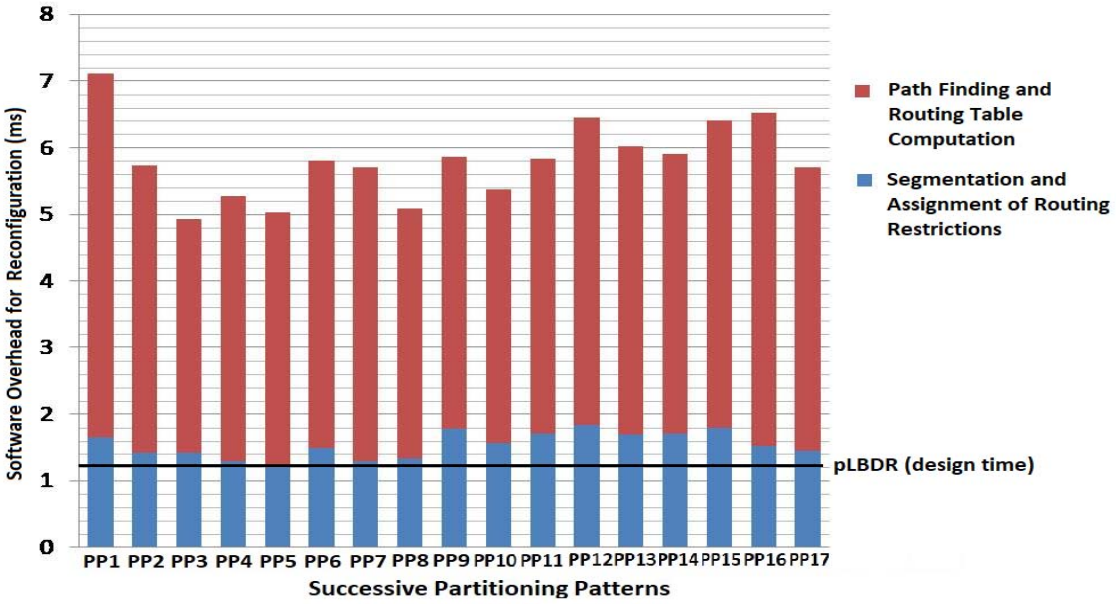



**Figure 5.8:** Path management overhead (in software) for table-based routing. For comparison, the design-time software overhead of pLBDR is reported (no runtime overhead though).

Table-based routing needs to recompute segments, routing restrictions and source-to-destination paths for each partition that changes. This computation time was quantified on a 2-core Intel(R) Core(TM) i7-7500U CPU clocked at 2.70GHz and 2.90 GHz, respectively.

Fig.5.8 reports the cumulative software overhead at runtime for table-based reconfiguration across scenarios. It ranges from 5 to 7ms, with an average of 5.8ms. Segmentation

[1]Each configuration packet is split into 32-bit words, called flits, which are individually transmitted. Hence, the number of flits dictates packet length.

and routing restriction assignment account on average only for 26.5% of the total software overhead, while most of the time is spent for the computation of routing paths, and to derive the new routing table configuration. pLBDR as no runtime overhead in software, but only a 1-shot design-time overhead of 1.2ms to find segments and routing restrictions on the network as a whole. This value is much lower than the runtime overhead for table-based routing, since the procedure is invoked only once, and skips the most expensive tasks (i.e., path search, table update).

### 5.3.3 Execution Efficiency

We now want to verify whether pLBDR leads to sub-optimal routing paths. Thus, we consider a 4x4 2D-Mesh topology, and all the "legal" partition shapes with a number of clusters $N$ ranging from 1 to 12. On the clusters of each of the 20 partition shapes under test, we run the FAST image processing benchmark by means of the cycle-accurate VirtualSoC simulation framework [37]. The benchmark has been parallelized into a scalable number of clusters. With pLBDR, routing restrictions (i.e., the routing algorithm) are those of the global network, while with table-based routing, the routing function is computed for each partition shape individually. To search for paths to fill up routing tables, we test two approaches: a baseline one that randomly selects from available shortest paths to destination, and an optimized one that selects the path that minimizes link congestion.

Results are illustrated in Fig.5.9. The execution time with table-based routing is slightly higher than our approach in some cases. The overhead is erased by the optimized path search strategy. Overall, the key take-away is that the routing paths selected by pLBDR do not affect execution efficiency.

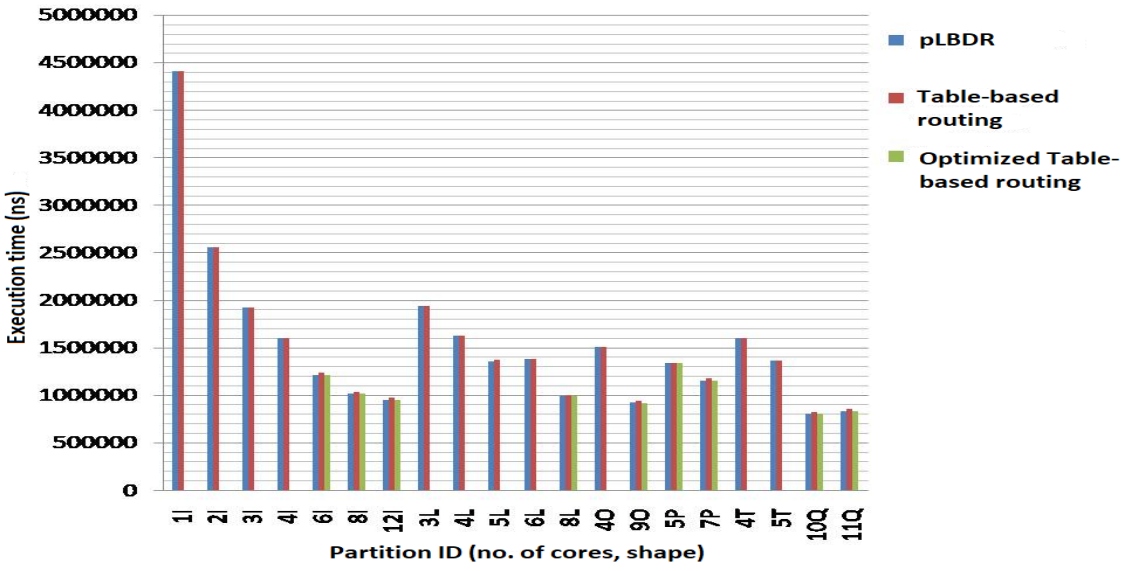



**Figure 5.9:** Execution efficiency of FAST.

## 5.4 Conclusions

In this chapter, we presented an interconnect-centric approach to flexible partitioning and isolation of many-core accelerators for future Fog computing nodes. The new requirements of strong isolation, efficient sharing and dynamic reconfiguration raise a routing control challenge in hardware that our work tackles through pLBDR. For a synthetic scenario of flexible repartitioning, the hardware reconfiguration overhead is from 74% to 97% lower, while no software overhead is incurred at runtime. In future work, pLBDR extensions will be targeted to alleviate partition shape limitations and find out whether such relaxation truly makes sense from the viewpoint of some non-functional system metric (e.g., resource utilization, fragmentation, etc.).

# Chapter 6

# A Partition Manager for Elastic Fog Nodes

In the wake of the hardware support for secure and flexible partitioning of a multi-tenant Fog node, a resource manager will be in charge of the lining up of the elastic partitions. The implementation of this resource manager will be the focus of this chapter. This work is a joint collaboration with two other instructors at university of Ferrara: Prof. Nonato Maddalena and Prof. Mauro Tortonesi.

## 6.1 Introduction

Previous Chapters have developed hardware support for the elastic partitioning of parallel Fog node architectures, revolving around isolation capabilities in the on-chip interconnection network in time and space, and around the efficient dynamic reconfigurability of the partitioning pattern. The support for isolated execution and resource elasticity well matches the public Fog scenario and the fluctuating workload of emerging IoT applications for it [15], including mobility-as-a-service [35], recognition and perception-based mobile applications (e.g., image recognition) [42], multiplayer online gaming [146], or edge network content caching for mobile streaming [80]. *By combining awareness of dynamic application workloads with resource allocation elasticity, the same (limited) pool of Fog resources could be dynamically reassigned to running services to match their differentiated usage peaks or business models [88].*

However, the developed hardware support needs to be complemented by a suitable resource management framework capable of mastering the tuning knobs of the "dynamic" hardware. Unfortunately, dynamic management of elastically-allocated Fog resources is a challenging problem due to the large space of system states and available control actions, and to the need for fast decision-making and small memory overhead to store the control policies [41]. To some extent, Fog managers can take advantage of prior art on the consolidation of diverse workloads with mixed-criticalities and/or varying resource demands on scalable chip-scale manycore architectures [66, 71].

There is a vast literature on dynamically adapting application mapping (i.e., partitions configuration) to optimize the system state in the presence of applications that frequently come and go or with fluctuating workloads [66, 119]. However, existing approaches to resource elasticity are not entirely suitable for Fog computing for one or more of the following reasons.
First, most state-of-the-art mapping frameworks target the optimality of the resulting system state following a reconfiguration event, or trade such optimality for online computation time, while not devoting the same effort to optimizing the reconfiguration cost (e.g., the number of needed application migrations) [102].
In a Fog computing environment, this would cause a lack of control over the service suspension time for hosted users.
Second, a tight control over the geometric shape of the space partition is not enforced, hence potentially giving rise to subtle interference effects on the NoC that break fundamental design and operational properties of multi-tenant Fog nodes such as composability, predictability [16] and secure-grade isolation [51].
Third, fundamental features of future multi-tenant public Fog architectures are not modelled when formulating the resource management problem, including:

- *The performance-sensitivity of partition placement on the computation grid.* Response latency or real-time requirements of IoT services impose a service-level agreement with the public Fog provider bounding the maximum average distance of partition tiles from the memory controller(s), or associating temporary exceptions with downgrade adjustments of the pricing scheme.

- *The strict prioritization of consolidated IoT services.* Fog nodes are capacity-constrained and cost-sensitive equipment; hence service priorities should be defined for partition management not only to define the next system state accordingly (e.g., to correlate memory controller proximity to latency criticality), but also to distribute the overhead for system reconfiguration (e.g., it is important to determine not only how many applications to remap, but also which ones, depending on their attitude to migrate).

- *The "everything-as-a-service" provisioning concept of Fog computing.* Optimizing the system state and its evolution over time does not consist of maximizing an aggregate performance metric of consolidated applications, but should rather aim at preserving the correlation of costs and benefits to the priorities of affected services. For instance, lightweight applications should be more frequently selected for migration, while applications with millisecond-range target response latencies should constantly be placed close to the memory controller.

### 6.1.1 Goal of this work

This thesis tackles the challenge of mastering resource-elastic spatial partitions at runtime on a scalable many-core architecture for Fog computing, while taking the distinctive features

of emerging public Fog environments into account.

The proposed partition manager fulfils the virtual resource reassignments performed by high-level implementation-agnostic service managers by rearranging the physical partition shapes on the actual computation grid in a non-overlapping way.
Our approach strives to pursue the correlation of partition proximity to the memory controller(s) with the performance criticality of the associated IoT service, while minimizing the perturbation of the system state, which is inherently related with the preservation of user experience.

This work contributes to break existing barriers in the open literature between research domains that should collaborate towards a complete, hierarchical resource management framework for the Edge-Fog-Cloud continuum. On the one hand, we complement existing resource management frameworks for networks of Fog nodes or for integrated Fog-Cloud systems [59] with another level of hierarchy locally managing the multi-tenant nature of individual public Fog nodes. On the other hand, we bring dynamic resource management of chip-scale many-core systems into the new ground of Fog computing, with its own features, constraints and goals.

This last milestone of the thesis revolves around a well-known *prior provisioning and prompt allocation* scheme of Fog resources [84]. The latter are reserved for future IoT use based on accurate traffic predictions, so that when actual IoT requests arrive, it is assumed the resources are available and immediate processing takes place. This paradigm already finds wide applicability in some application domains where the workload can be easily predicted (e.g., online gaming [146] or smart cities [103]), but is gaining further momentum as learning algorithms which are increasingly applied to learn complex prediction models [75].

This chapter is organized as follow: First we give a global view of the holistic (hardware and software) Management Framework for IoT devices. Then, we expound the different parts of our innovative optimization problem for elastic partitioning of Fog platforms under dynamic operational behavior. Finally, we prove the efficiency of the proposed resource management framework in managing elastic partitions while taking into account their requirements and specifications.

## 6.2 Analysis of real workload characteristics

Designing resource management frameworks for the Fog has to face a difficult challenge: we need to understand the workload variability rate of Fog applications, and to be aware of their QoS requirements. This is non-trivial, since no large-scale general-purpose Fog computing platform is currently available. As a result, few developers will spend significant time building applications which exploit the capabilities of Fog computing platforms unless these platforms already actually exist.

A recent report on the taxonomy and the requirements of Fog computing applications has delivered a fundamental milestone to break this vicious cycle [15]. The report carefully

selected 30 actual or proposed applications that cover a wide range of usage types for future Fog computing platforms. As expected, it turns out that support for dynamic workloads is one of the specific features that Fog platforms should have to support specific categories of applications. More in detail, the workload produced by the reference applications can be categorized into two general categories and few sub-categories based on the characteristics of such workloads:

- **Stable:** the workload is almost static.
- **Dynamic:** the workload varies according to some criterion:
  - Location: the workload varies according to the location of the node.
  - Time: the workload varies as a function of time.
  - User: the workload varies according to the load generated by users.

**Table 6.1:** Workload characteristics of emerging Fog applications.

| Category | Sub category | Applications | Total |
|---|---|---|---|
| Stable | | APP01, APP02, APP03, APP04, APP06, APP07, APP08, APP10, APP12, APP14, APP15, APP16, APP17, APP18, APP19, APP20, APP21, APP22, APP23, APP24, APP25 APP26, APP28 | 23 |
| Dynamic | Location | APP05, APP09, APP11 | 3 |
| | Time | APP13 | 1 |
| | User | APP27, APP29, APP30 | 3 |

Table 6.1 classifies emerging Fog applications, that cover a wide range of usage types for future Fog computing platforms, according to their type of workload. We see that the workload for many fog applications is currently stable, due to the static resource management nature of current hardware/software architectures they are designed for. In these scenarios, sensors collect data periodically and send them to fog application for processing. For example, APP04 [6] uses surveillance cameras to take pictures at periodic intervals and sends them to the Fog for analysis.
These types of applications are present in different sectors such as transportation, health, entertainment, smart cities, smart factories, smart buildings, and smart grid.
This does not mean that the workload of such applications will continue to be stable once dynamic reconfiguration capabilities of the underlying hardware platforms will be exposed to software developers. For instance, the scientific literature is already extensively reporting several IoT frameworks where in-sensor intelligence can pre-process data so that the most relevant ones are sent to the next computing layer (e.g., a Fog layer) for further processing.

Simply, the applications analyzed in [15] mostly reflect state-of-the-art COTS platforms, and have been (or are being) developed for them.

In light of this, it is quite promising that a significant fraction of applications already exposes a dynamic workload to platform managers. In fact, seven out of the surveyed applications have dynamic workloads, among which three are dynamic with respect to location, one is dynamic with respect to time and the rest three are dynamic with respect to users. In App09 the number of self-adaptive added stations may change [8]. In App11 the number of surrounding fog nodes with which the fog node communicates may changes [7]. Finally, App13 collects real-time transport demand in a mobile way which makes the workload change [35].
We believe that the awareness of workload characteristics of fog applications helps in the design of effective and efficient management and operation of fog computing platforms. Especially for fog applications with dynamic workloads, it is necessary for the fog infrastructures and applications to be designed and deployed in a scalable manner. Meanwhile, fog management platforms need to incorporate intelligent application placement, dynamic resource allocation mechanisms and automated operation systems to ensure acceptable QoS is guaranteed.

In this thesis, we focus on user-centric applications such as APP27 [42], APP29 [146], APP30 [80], deploying their server part in Fog platforms. Interestingly, from [15] we can deduce that applications with dynamic workload as a function of the load generated by users are all latency-sensitive: the primary reason for them to use Fog computing is the expected low response latency.

For these reasons, in designing our partition management framework we targeted emerging user-centric Fog applications with latency criticalities. We are aware that data velocity requirements are quite diverse for such applications, ranging from $<$ 10msec, 10-100msec, 100msec-1sec, 1-10 sec, or even more [15]. As a result, we consider the strict prioritization of latency requirements, depending on their magnitude.

Predictive models of user load for the selected class of applications reveal that workload variability takes place at a time scale of hours (see for instance [146]). Therefore, we find that the online elastic partition management problem can even be solved to optimality given the dynamic behavior of these realistic applications, as long as a computing budget of a few seconds is not exceeded.

## 6.3 Global Management Framework for the Fog

Fog managers can take advantage of prior art on the consolidation of diverse workloads with mixed-criticalities and/or varying resource demands on scalable chip-scale manycore architectures [66, 71]. The most complete approaches share the same basic intuition of decomposing the dynamic resource management problem into two hierarchical sub-problems

[36, 44, 73]. This separation of concerns allows to adopt the divide and conquer approach to resource management, and develop independently solutions at the software and at the hardware management layers.

At the top layer, the allocation problem of platform resources to applications is formulated by leveraging an abstract view of both application requests (e.g., service levels [44]) and physical resources along both spatial and temporal dimensions (e.g., virtual private machines [73], cells [36], virtual processors [44]). This way, application and machine models are implementation-independent and have non-functional characteristics (e.g., performance, power, reliability) that are easier to reason about. We hereafter denote this management problem as *the application-level manager. At this level, the emphasis is typically on how much of a resource to allocate, rather than on which exact resource [76]. There is extensive literature on this topic, which can be reused for our target elastic Fog node as well [119].*

*At a lower abstraction layer, virtual resource assignments should be enforced by securely multiplexing or distributing the concrete hardware resources among services.* This mapping step of virtual resources onto physical ones gives rise to an additional dimension of the resource management problem in both time and space, which we hereafter denote as the *partition manager*. In time, a common approach consists of the proportional-fair temporal scheduling of one or more selected processing elements [151]. In space, it is generally recognized that applications perform better when the set of tiles allocated to it are contiguous [102], that is, when they form a *spatial partition of adjacent tiles* connected by reserved links and routers of the backbone on-chip interconnection network (NoC). As anticipated in the introduction, the partition management problem has never been specialized for a Fog platform, which comes with its own distinctive challenges: secure-grade isolation (i.e., partitions should be contiguous and non-overlapped), performance-sensitive placement of applications on the computation grid, strict prioritization of requests, delivering resource elasticity over time, and prioritized distribution of reconfiguration overheads. For this reason, it is the focus of this Chapter.

The context for the work of this chapter is thus illustrated in Fig. 6.1. A hierarchical Global Management Framework (GMF) attempts to harmonize the conflicting requests of IoT services for the limited pool of Fog resources, while exploiting workload dynamism for opportunistic reassignment of resources among consolidated services over time. This framework can leverage the multi-dimensional properties of the elasticity concept, including resource, cost and quality elasticity, although the framework as a whole goes outside the scope of this thesis. For instance, services could be able to adapt their quality-of-service to their assigned level of resources.

Following the analysis in previous section, GMF leverages the number of users accessing each IoT service at any given point in time as a theoretical foundation to realize fair resource assignment: the more the users the more the resources to be devoted to each IoT service on the Fog node.

In particular, we consider a common resource allocation policy for Fog computing where predictive models of the workload are used by the top-level "Application Manager" to proac-

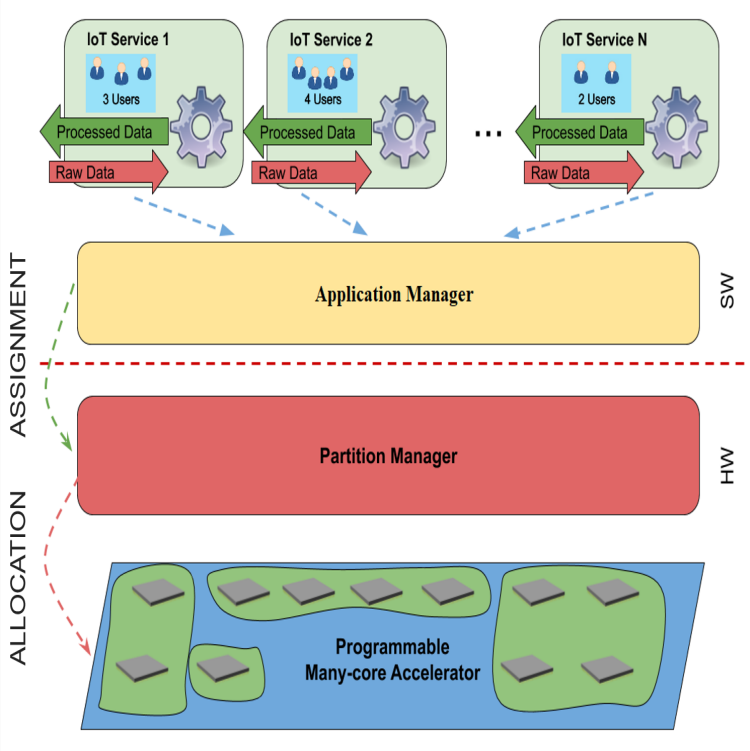



**Figure 6.1:** Global Management Framework

tively re-negotiate resources among IoT services for the next monitoring period. The latter has a typical duration of a few hours for the emerging user-centric dynamic-workload applications in [15]. When the actual user requests will arrive, resources will have been already allocated, and the deployment is almost immediate. Workload projections are translated into an actual request for more or less resources by the Application Manager, which solves conflicts based on some prioritization criteria, while fulfilling the service-level agreements and using an adaptive pricing scheme. During this process, preassigned virtual resources can be revoked when the system is overloaded. What matters to us is that this virtual resource assignment is completely placement-agnostic, since the Application Manager does not have to deal with the concrete physical resources but only with their abstract view. At this level, variations in resource assignment have to compensate each other, that is, a pool of services can be assigned more resources provided such resources are currently idle or revoked to some other services.

Upon receiving the up/down-scale requests, an underlying "Partition Manager" computes the optimal partition map for the given assignment. In Fog nodes, this problem has an inherent geometrical structure, in that a partitioning pattern of the system has to be changed into a new one, where existing partitions grow, shrink or remain neutral, and where new partitions are set up or old partitions are removed (see Fig. 6.2).

There are a number of subtle effects that should be considered during this *conversion of a resource assignment into an actual resource allocation:*

- shrinking partitions could hand over resources to growing partitions that are placed

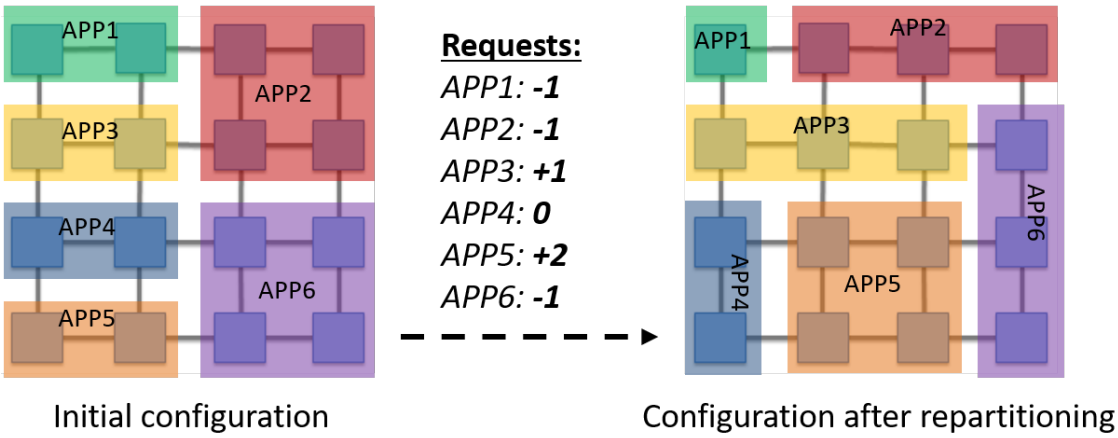



**Figure 6.2:** Evolution of the physical partitioning pattern to fulfil reassignments of virtual resources.

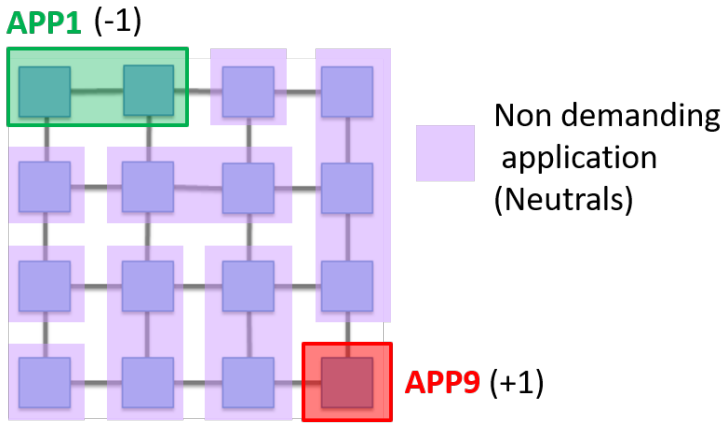



**Figure 6.3:** Problems in translating virtual resource assignments into actual resource allocations.

far apart on the computation grid, thus raising the need to shift neutral partitions in-between (Fig. 6.3). This can have unpleasant chain repercussions on such neutral partitions, which will undergo unwanted suspensions/transformations to shift and/or adapt their shape and position.

- enabling the actual hand-over of resources implies the choice of the neutral partitions to be shifted. This requires that its users will undergo application migration thus affecting quality of experience. Clearly, the amount of system perturbation during this reconfiguration stages is an unmistakable optimization target.

- During the adjustments of the partitioning pattern, the tiles assigned to a partition should keep an average distance from the memory controller that should be kept correlated to the latency criticality of the application. Since communication to the memory controller can take place with predictable performance through the TDM NoC, controlling the position of the partition on the grid enables to fulfil the latency criticality of the application at hand. Typically, a service level agreement will define the maximum distance that each application can afford.

- The effort of preserving the correlation of grid positioning to latency criticality through-

out the dynamic reconfigurations of the system depends on the pattern of increment and decrement requests from the Application Manager, and on how they map to the grid. Therefore, we expect a trade-off between the number of user migrations and the preservation of such correlation.

In order to find the optimal hardware allocation, our Partition Manager implements and solves an ILP optimization model, which will be illustrated in the following section. The optimization problem aims at a trade-off between the applications proximity to the memory controller depending on their latency criticality and the minimization of the migration overhead which is stimulated by the number of tiles that are migrated in order to satisfy the demands of running or new admitted services.

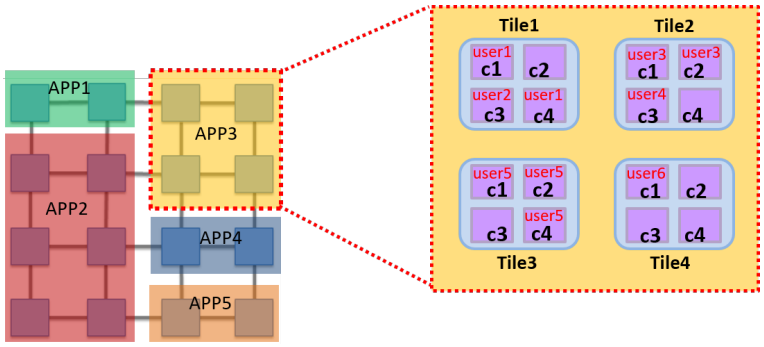



**Figure 6.4:** Mapping of user processes to the cores of the tiles of a reserved partition.

## 6.4 System Architecture Refinement

In light of the realistic Fog applications considered, and of the Partition Management constraints, we add details to the Global Architecture of Chapter 3, which are requested by the specification of the management framework.

First, the PMCA is spatially partitioned among IoT services. In each computing tile of a partition, users of the partition-specific IoT service are mapped to one or more cores (core reservation scheme). In principle, they can own a different number of cores (see Fig. 6.4). The only constraint we pose is that each user process is allocated to one tile, that is, its applications cannot be distributed throughout several tiles. This way, only loose inter-dependencies can be modelled among tiles in the partition management problem. There are exceptions to this, which however do not violate the assumption. In fact, several MMOG applications instantiate virtual machines and do not correspond to users, but to virtual regions of the game [40]. User load is then mapped to the different virtual machines. Such virtual machines can sporadically communicate with each other. Again, we find that in the partition management problem it is enough to preserve partition contiguity, rather than modeling application-specific inter-tile dependencies.

Second, our approach enables hierarchical resource management, which is a mainstream approach to tackle the management complexity of large manycores. In practice, we assume that in each partition a local manager is set up by the IoT service provider in order to

allocate user processes to tiles and cores inside them. From this viewpoint, such local managers can take advantage of the most advanced research findings in the open literature for the dynamic mapping of applications to multi-/many-core architectures. For instance, the partition could be multiplexed in time, like in [76], or the mapping of user processes to tiles could be dynamically modified as a function of the service workload, or to evacuate thermal hotspots, even in the presence of real-time constraints [108]. This local manager is outside the scope of this paper.

Finally, we observe that in this chapter we address the management problem of spatial partitions. We do not address the related management problem of time partitions in the TDM NoC, under the simplifying assumption that each IoT service is getting the same bandwidth on the TDM NoC. The co-design of the two management frameworks is left for future work, and will have to deal with new degrees of freedom for the optimization problem. In particular, the allowable distance of a partition from the memory controller will depend also on the allocated bandwidth on the TDM NoC for MC traffic.

## 6.5 A Shape-Oriented Model For Optimal Partitioning

The dynamic reconfiguration process should allow to update the partition shape, size and location on the PMCA so that the required resource remodulation is accomplished, i.e., the demand is satisfied. The new configuration should comply with a few quality criteria, based on the above discussion.

A first issue concerns how much the transition to the new configuration will affect the computation. This depends on the migration overhead that user processes of each IoT service experience during reconfiguration. Some services can be more sensitive than others, depending on the amount of state to transfer from the source node to the destination node for user process reactivation. This criterion is referred to as *continuity*. In principle, optimizing the system dynamism for continuity means to minimize the number of user processes that are migrated during reconfiguration events. This would mean to make the optimizer aware of how many users are running at any given point in time on each tile, so that the least loaded tiles could be preferentially migrated. While this is certainly possible with the proposed framework through the insertion of an ad-hoc term to the objective function, in the first stage of development we do not discriminate tiles based on their current loads, i.e., we assume all tiles of a partition can be selected for migration with the same probability as long as this leads to better aggregate system state and/or to a lower number of migrations overall. This corresponds to scenarios where the local manager keeps the tiles of its partition as uniformly loaded as possible.

A second issue concerns the location of partitions on the PMCA grid. Since the distance of a tile to its memory controller port affects communication latency, the closest the tiles of a partition are to it, the better the performance will be for the IoT service; again, different services may have different latency criticalities and be affected in a different manner by a

change in proximity to the memory controller access port. This second criterion is thus referred to as *priority*, and boils down to preserving the correlation between positioning on the grid and latency criticality.

A reconfiguration is spurred by a resource remodulation that changes the number of tiles assigned to one or more partitions. This, in turn, induces changes in the partition shapes and in their allocation on the PMCA.
Finally, partitions mutually compete for the best positions on the PMCA. All these decisions are so deeply intertwined that, if taken sequentially in a greedy way, may prevent not only solution quality but also feasibility, due to geometrical constraints. An alternative is to code the problem, i.e. decisions, constraints, and quality criteria, into a mathematical model and solve it to (near) optimality.

An instance is defined by the current configuration, i.e., the set of the partitions and their location on the grid, and the partition size increments and decrements yielded by the Application Manager's reassignments (the *requests*). We assume that: (i) requests are zero sum (they balance each other), while (ii) the number of growing partitions may be different from the number of shrinking ones. In both cases, we refer to these partitions as the *demanding IoT services* as opposed to *neutral* ones, that do no ask for any change in resource assignment.

### 6.5.1 Model assumptions

The optimization framework relies on the following assumptions:

- Demands are always feasible (zero sum).
- The memory controller access port is located at the bottom-right corner of the 2D-mesh topology. The model can be easily extended with other memory controller ports.
- The main focus is on trading the optimality of the execution state after reconfiguration with the reconfiguration overhead.
- The optimality of the execution state is defined as the preservation of a correlation between average Manhattan distance of partition tiles to the memory controller with the performance criticality level of the IoT service mapped to that partition.
- The model bounds the management overhead of neutral partitions affected by reconfiguration rounds by allowing them only to shift and not to be resized.
- We consider a congested scenario where several IoT services at a time compete for scarce resources. Even in case the system is not overloaded, our GMF tends to allocate all resources to partitions anyway as a backup solution in case the workload of IoT services has been incorrectly predicted for the next monitoring period.

- Without lack of generality, only squared and rectangular partition shapes are considered in this chapter. In principle, all legal partition shapes enabled by pLBDR can be modelled and managed through the proposed Partition Manager, however the focus of this work is primarily on how to trade-off reconfiguration overhead with system state optimality (i.e., on the formulation of the optimization problem itself), rather than on the trade-offs between regular vs. irregular partition shapes (i.e., not on the comparative efficiency of several partitioning strategies). Given the outcome of this chapter, the proposed Partition Manager is indeed ready for the latter research goal as well, which is however left for future work.

From now on, an IoT service component will be more conveniently denoted as an application and the 2D-mesh topology as a grid. We propose an Integer Linear Programming (ILP) shape-based model which builds on a predefined set of partition shapes, i.e., all rectangular partitions that are feasible in an $n \times n$ 2D-mesh, up to a certain size specified by the system designer. We focus on a single iteration where the partition map $P$ of the current configuration is reconfigured into $P'$ to accomplish size variations while optimizing the aforementioned quality criteria.

### 6.5.2 Model inputs

The model receives in input i) a formal description of the grid and its geometry, ii) the current configuration, iii) the list of the application priorities and requests. Formally:
- The square grid $G$, made of $n^2$ tiles identified by their position $(ij)$ in the grid, $1 \leq i, j \leq n$.
- The ordered list of predefined shapes $s \in S(c)$, made of all rectangles $s = [l, w]$ of cardinality $c = lw$, for all admitted sizes $c$. $l$ and $w$ are respectively the number of columns and the number of rows of the rectangle, which allows to distinguish orientation when $l \neq w$. A shape's placement on the grid is described as the coordinates of its *handle*, i.e., the top-left corner of the shape. Therefore, $G_{sc}$, the set of potential locations of a shape's handle is known for each $s \in S(c)$. Likewise, for each $(i, j) \in G_{sc}$, the set $G_{sc}^{ij}$ of the grid points occupied by $s$ when its handle is at $(i, j)$ is also known.
- The current partition map $P$ in terms of the status of the tiles: either $(ij)$ is free or it has been assigned to an application $t = \mathcal{P}(i, j)$. Based on these tile level information, $P$ may also be described at the application level, for each $t \in T$, in terms of i) $G^P(t)$ the partition assigned to $t$ in $P$, ii) its cardinality $c^P(t) = |G^P(t)|$, iii) its shape $s \in S(c^P(t))$, and iv) the handle's coordinates $(h, k)$ so that $G_{sc}^{ij} = G^P(t)$.
- The demand $d^t$ for each $t \in T$, intended as the variation with respect to $c^P(t)$. If $d^t = 0$ we speak of a *neutral* application, $t \in T^0$. Otherwise ($d^t > 0$ or $d^t < 0$ ), $t$ is a *requester* and $t \in T^D = T \setminus T^0$.
- The application priorities $p^t \in \{1, \cdots, max_p\}$ with respect to the VoI of the application of which application $t$ is implementing a service component, as well as $\tilde{p}p^t \in \{1, \cdots, max_{\tilde{p}}\}$, related to the migration overhead, had the application to be suspended.

### 6.5.3 Decision variables

The new partition map $P'$ after reconfiguration is described in terms of the following (binary) variables.

- $x_{ij}^t \in \{0,1\}, \forall t \in T, (ij) \in G$ : is equal to 1 if tile $(ij)$ is assigned to application $t$.
- $z_{sc}^t \in \{0,1\}, \forall t \in T, s \in S(c), c = d^t + c^P(t)$: is equal to 1 if shape $s \in S(c)$ is assigned to application $t$.
- $w_{sc}^{ij,t} \in \{0,1\}, \forall t \in T, (ij) \in G_{sc}$: is equal to 1 if the handle of shape $s \in S(c)$ is located at $(i,j)$ and $s$ is assigned to $t$.

For a given $c$, the number of shapes $|S(c)|$ is linear in $c$, and the possible handles are bounded by $n^2$. Therefore, the number of variables at each iteration is polynomial with respect to $n$. For sake of readability, it will be assumed $c = d^t + c^P(t)$ whenever speaking of $z_{sc}^t$ and $w_{sc}^{ij,t}$, without further specification.

### 6.5.4 Constraints

The following constraints filter out unfeasible tile allocations.

Partition constraints (6.1) ensure that each tile is assigned to at most one application. Constraints (6.2) ensures that after the reconfiguration $t$ will be assigned exactly one shape $s \in S(c)$. Constraints (6.3) ensure that each assigned shape $s$ has been properly located on the grid. Constraints (6.4) state that tile $(i,j)$ is assigned to application $t$ if and only if the handle's location $(h,k)$ of the shape assigned to $t$ is such that $(i,j) \in G_{sc}^{hk}$. Reversely, constraints (6.5) state that if the handle of shape $s \in S(c)$ is located in $(i,j)$ and $s$ is assigned to $t$ then each tile in $G_{sc}^{ij}$ is assigned to $t$. Constraints (6.6) establish the relation between the number of tiles $t$ requires and the size of the assigned shape. Finally, constraints (6.7-6.9) enforce integrality for tile assignment, shape, and handle variables.

$$\sum_{t \in T} x_{ij}^t \leq 1 \qquad \forall i,j \in G \tag{6.1}$$

$$\sum_{s \in S(c)} z_{sc}^t = 1 \qquad \forall t \in T \tag{6.2}$$

$$z_{sc}^t = \sum_{(i,j) \in G_{sc}} w_{sc}^{ij,t} \qquad \forall t \in T, s \in S(c) \tag{6.3}$$

$$x_{ij}^t = \sum_{s \in S(c)} \sum_{(hk):(ij) \in G_{sc}^{hk}} w_{sc}^{hk,t} \qquad \forall t \in T, (ij) \in G \tag{6.4}$$

$$w_{sc}^{hk,t} \leq x_{ij}^t \qquad \forall t \in T, s \in S(c), (hk) \in G_{sc}, (ij) \in G_{sc}^{hk} \tag{6.5}$$

$$\sum_{(i,j)\in G} x_{ij}^t = c \sum_{s\in S(c)} z_{sc}^t \qquad \forall t \in T \tag{6.6}$$

$$x_{ij}^t \in \{0,1\} \qquad \forall t \in T,\ \forall (i,j) \in G \tag{6.7}$$

$$z_{sc}^t \in \{0,1\} \qquad \forall t \in T, s \in S(c) \tag{6.8}$$

$$w_{sc}^{ij,t} \in \{0,1\} \qquad \forall t \in T, (ij) \in G(s,c) \tag{6.9}$$

### 6.5.5 Objective function

The objective function provides an analytical formulation of *priority*, i.e., how much partition placement is aligned with applications latency criticality, as well as *continuity*, i.e., the migration overhead due to reconfiguration.

Priority is formalized in terms of the minimization of the sum over all tiles $(i,j) \in G^P(t)$ of the Manhattan Distance $MD^{ij}$ from $(ij)$ to the MC location, which we assumed in $(n,n)$ for all applications, weighted by the application priority $p^t$, for all $t \in T$.

Regarding continuity, note that reconfiguration might affect any application, including neutral ones, and not just requesters. Consider the borderline example of a requester asking for additional tiles to host incoming demand, which is assigned a larger partition which includes all its previous tiles. If additional demand is allocated to the new tiles, then none of the past processes needs to be interrupted. On the opposite, a neutral application allocated next to the requester in the current configuration may have to be shifted somewhere else on the grid, to free the additional tiles required by the requester. It follows that continuity must take into account the migration overhead of any tile that will be assigned to a different application after reconfiguration, weighted by the application priority $\tilde{p}^t$ (application priorities may be different in the two components). As continuity and priority are potentially conflicting, we consider a weighted sum, and solve:

$$min \qquad \alpha \sum_{t\in T} \left( p^t \sum_{(i,j)} MD^{ij} x_{ij}^t \ + \ \gamma\, \tilde{p}^t \sum_{(i,j)\notin G^P(t)} x_{ij}^t \right) \tag{6.10}$$

subject to (6.1 - 6.9). Taking $\alpha = 1$, the weight parameter of the second term $\gamma$ will be experimentally calibrated in the next section. Note that the first component sums over all tiles and models a feature of the new configuration. Therefore, this is a term without memory. On the contrary, the second term contains a reference to the old configuration, as the summation concerns only tiles that in the new configuration will be assigned to a different application, therefore impacting the transition by penalizing the shift of tiles of high priority applications.

Beyond continuity and priority: While here we focus on priority and continuity, many other criteria could be formalized. Let us mention the most relevant ones.

The migration overhead could specifically differentiate among the tiles, by penalizing the reallocation of those that are processing critical contents. Suppose that $(ij)$ is such a tile

and $\alpha_{ij}^t$ is a measure of its migration criticality. In such a case $\alpha_{ij}^t \left(1 - x_{ij}^t\right)$ adds the penalty term $\alpha_{ij}^t$ if and only if the content of tile $(ij)$ will be migrated.
Moreover, communication overhead between processes running on different tiles in the same partition could be taken into account. In the worst case, the time depends on the distance between the two furthest apart tiles, which in turn is correlated to the elongation of the selected partition shape. For each $s \in S(c)$, its elongation can be expressed as a function of its sizes $l$ and $w$, and multiplied by $z_{sc}^t$. By adding to the objective function a term which penalizes the choice of elongated shapes (or more elongated than the current one) we can model this criterion.
Finally, in case of particular applications with very high latency criticality, the required real time throughput could be enforced by simply banning all those locations and shapes that do not comply, due to the distance from the MC or a too elongated shape. This feature can be checked in real time, and the variables corresponding to infeasible choices will be set to 0.

The ILP model (6.1 - 6.10) is efficiently solved by state of the art solvers for small $n$ and beyond, as discussed in Section 6.6.4. To efficiently scale further up with the grid size to encompass many tile grids, it can be tightened by adding various types of cuts, such as those exploiting conflicts based on shape geometry. Indeed, conflicts arise between decisions causing overlapping, either detectable at the shape level or when locating handles on the grid. Ad hoc separation algorithms can be devised to dynamically generate violated cuts and speed up convergence within a Branch and Cut approach, which is left for future work.

## 6.6 Experimental results

The experimental part aims to calibrate the $\gamma$ parameter and to assess the effectiveness of the objective function. Then, the approach is tested against a simulated real case where the dynamic workload spurs real time demand reconfiguration at the Edge's devices. Finally, the approach scalability is tested, in terms of the running time required to solve the model to proven optimality on larger size instances.

### 6.6.1 Preserving ideal configurations

Starting from an ideal application placement i.e., such that distance from the MC access port is inversely correlated to application priority, it will be verified that reconfigurations of the partitioning pattern preserve this status despite the needed adjustments to fulfil the assignment variations of placement-agnostic Application-level Managers. We consider a 4x4 2D-mesh topology with no idle tile and a set $T$ of 5 applications with different priority, ranging from $(p^{t_1} = 1)$ to $(p^{t_5} = 10)$, $p^t < p^{t+1}$, and $p^t = \tilde{p}^t \forall t \in T$.
The initial configuration is obtained by solving the model with respect to the null configuration (empty grid) and demand equal to the application size. In such a way, the objective function is driven only by priority.
To avoid initial configuration biases, we consider 50 scenarios differing for $T^D$, i.e., the re-

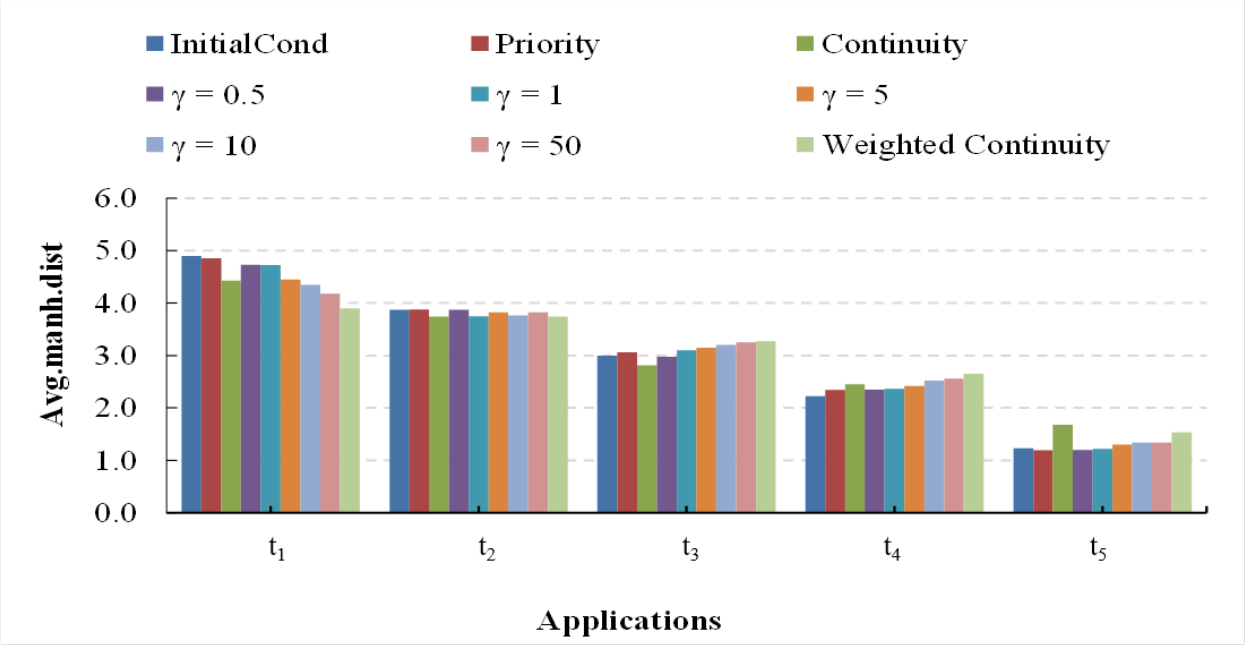



**Figure 6.5:** Average Manhattan distance after reconfiguration when starting from application placements correlated with their priority.

questers in $T$, and their demand, to get 50 different ideal initial configurations.
For each of them, a random, 0 sum, demand is computed and the model is solved with respect to 8 different objective function variants, say $f^1, .., f^8$, obtained by varying $\alpha$, $\gamma$ and $\tilde{p}^t$ as follows:

- $f^1$=($\alpha = 1$, $\gamma = 0$). (*Priority*)
- $f^2$=($\alpha = 0$, $\gamma = 1$, $\tilde{p}^t = 1 \forall t$). (*Continuity*)
- From $f^3$ to $f^7$, $f^i$=($\alpha = 1$, $\tilde{p}^t = p^t \forall t$, $\gamma \in \{0.5, 1, 5, 10, 50\}$).
- $f^8$=($\alpha = 0$, $\gamma = 1$, $\tilde{p}^t = p^t \forall t$). (*Weighted Continuity*).

Note that coefficients $p^t$ do not vary.
In short, $f^1$ is only based on priority, then it is not influenced by the current configuration state.
On the contrary, both $f^2$ and $f^8$ are driven by the minimum alteration of the current state, but $f^8$ differentiates based on the application priority.
Finally, $f^3, .., f^7$ are weighted sums of $f^1$ and $f^8$ with an increasing weight on the latter.
First, let us analyze how the different functions affect the ideal initial configuration, in terms of preserving the inverse correlation between application priority and the Manhattan distance to the MC access port. Fig. 6.5 reports such distance for every application, averaged over the 50 scenarios.
Applications are depicted in descending priority. For each application, the first column refers to the initial configuration and the next ones are associated to the configurations obtained

by $f^1$ onward.
It can be noticed that $f^1$ the most preserves the ideal allocation, i.e., a slight improvement can be noted for $t^1$ and $t^5$, their average Manhattan distance reached 4.85 and 1.19, respectively. Slight differences are due to changes in partition sizes, as the distance is averaged over the tiles of each application.
$f^2$ upgrades the status of the lowest priority application while downgrading the highest priority one, i.e., $t^1$ distance drops from 4.9 to 4.43 ($p^1 = 1$) while $t^5$'s one raises from 1.23 to 1.68 ($p^{10} = 10$).
Minor changes are observed for the other applications.
We argue that reconfiguration may often have entailed swaps between requesters' partitions, thus leaving neutral ones mostly untouched, rather than operating local adjustments such as expanding or contracting requesters partitions, which would have greatly affected neutral applications in a sort of a wave effect. When high priority applications are swapped with low priority ones the existing distance-priority correlation is affected.
Concerning the other objective functions, we observe that for each application, whatever is the change w.r.t. to ideal state, it gets emphasized along the growth of $\gamma$.
Indeed, one may think of $f^8$ as the extreme one gets when $\gamma >> \alpha$.

$f^4$ ($\gamma = 1$) shows the best average Manhattan distance, as the results are very close to $f^1$ where applications priority matters, while keeping at bay the number of migrated tiles.

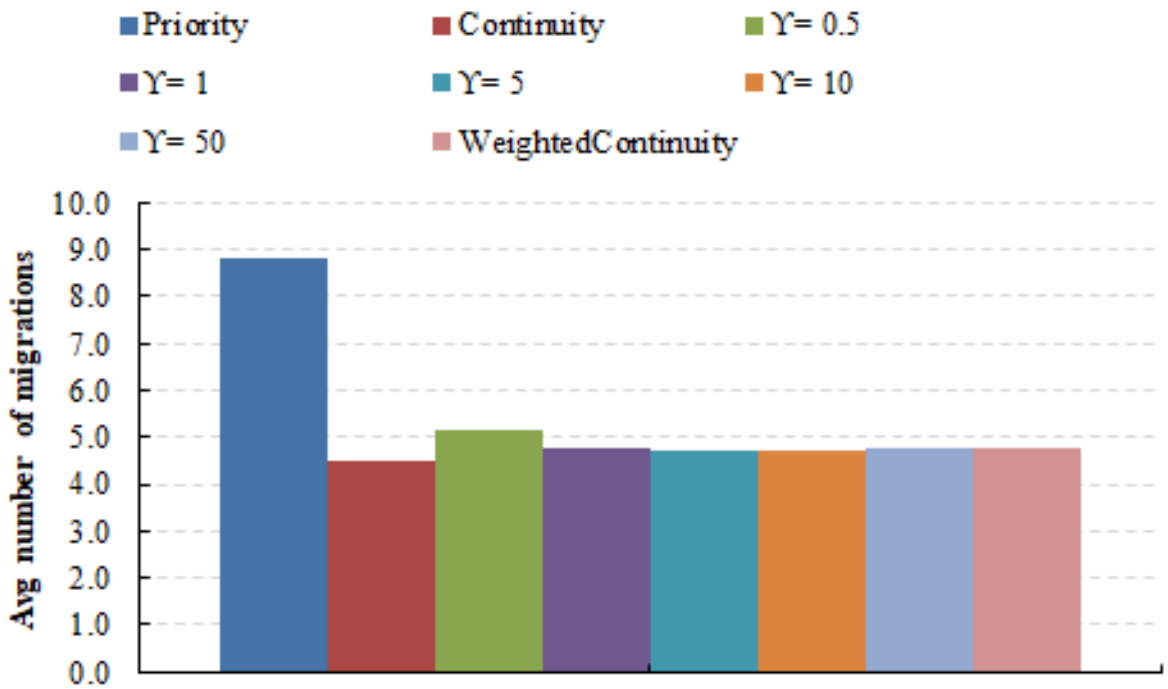



**Figure 6.6:** Average migration overhead

For the same experimental setting, Fig. 6.6 outlines the average total number of migrated tiles over the 50 scenarios.

The simulations are run for each of the previously discussed variants. The lowest average is equal to 4.48, which is reported in the case $f^2$, while in $f^1$ we note the highest average of migrations, i.e. 8.84. In the case $f^4$, when $\gamma = 1$, the average is 4.78, which confirms that for this setting the tool provides the best trade off between priority and continuity. For the sake of comparison, we also report the normalized total number of migrated tiles for each application over the 50 scenarios, results are shown in Fig. 6.7. The results obtained in $f^2$ are normalized to 1 and other variants outcome is normalized accordingly.
Planned comparisons demonstrate two major things. First, comparing to $f^1$, the results obtained using $f^8$ are substantially better in term of high priority applications migrations. In fact, the normalized total number of tiles that migrate for $t_4$ and $t_5$ has decreased by 14% and 26%, respectively. Second, the results confirm the choice of the value of $\gamma$. For this setting ($f^4$), the normalized total number of tiles that migrate for $t_4$ and $t_5$, comparing to $f^1$, has decreased by 6% and 12%, respectively. While for low priority applications it has increased by 9%, 34% and 12% for $t_1$, $t_2$ and $t_3$ respectively.
All together with the previous results, we conclude that using this value of $\gamma$, the tool hits a very good trade-off between the average Manhattan distance from the MC access port and the total number of tile migrations. In the next experiments, we will only consider this value of $\gamma$.

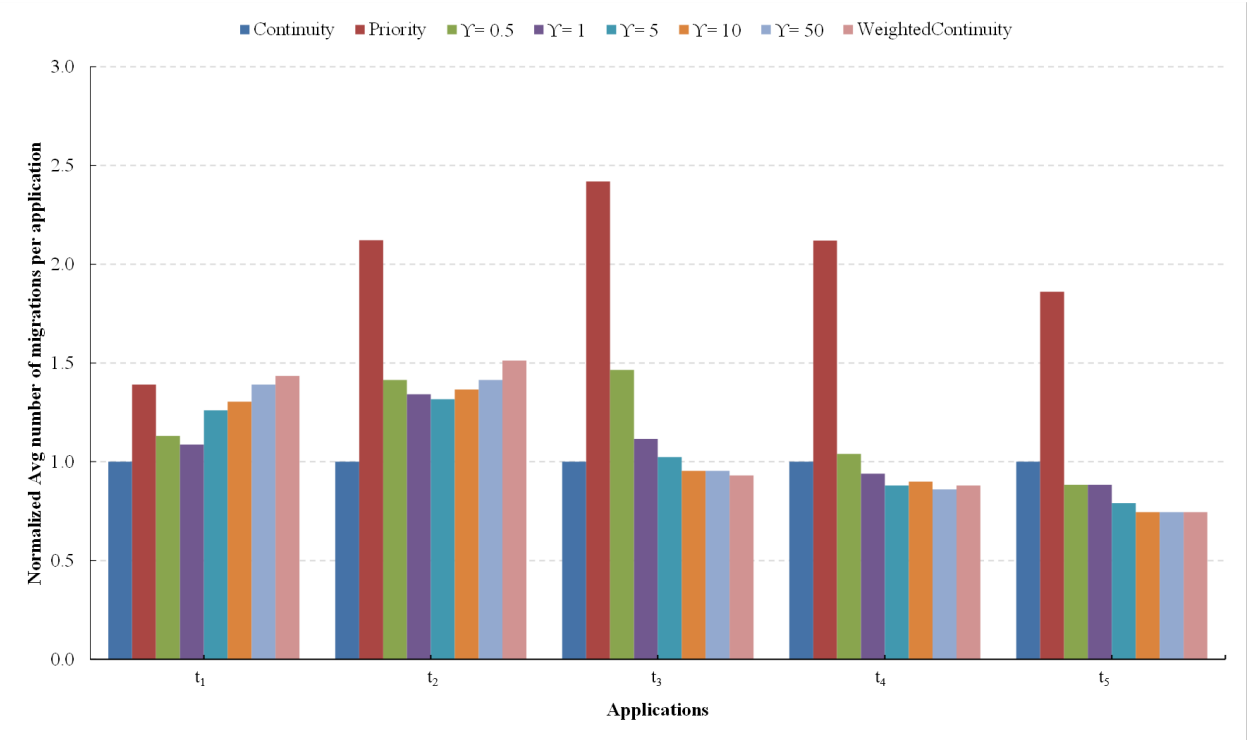



**Figure 6.7:** Normalized total number of migrated tiles

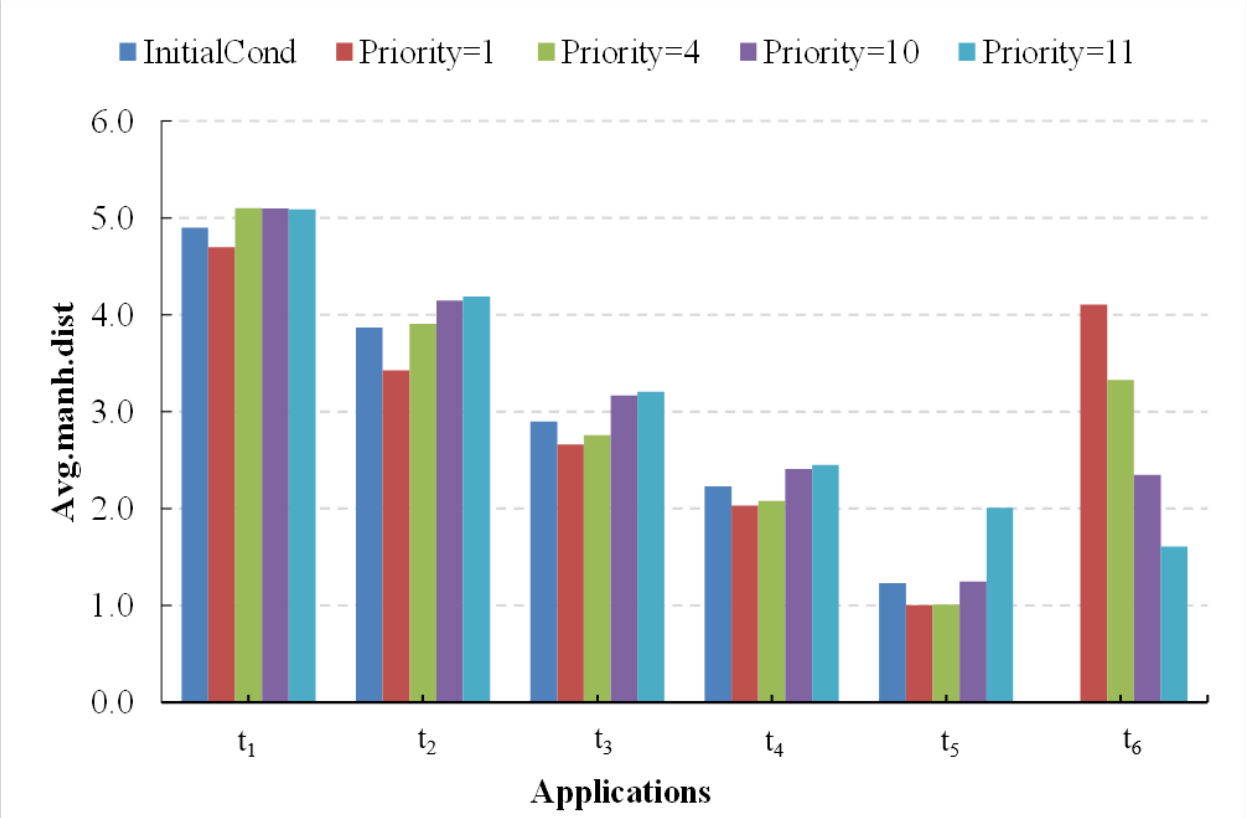



**Figure 6.8:** Average Manhattan distance after reconfiguration when a new application is admitted

### 6.6.2 Model's capability to optimally place new admitted applications

Next, we demonstrate the tool capability to optimally place new admitted applications when starting from an ideal placement. We re-used the same 50 initial conditions of the previous experiment. For every configuration among the 50 initial scenarios, we admit a new application $t_6$ and we run 20 instances where requesters and demands are randomly generated. This experiment is repeated using different priorities of $t_6$, where $p_{t_6} \in \{1, 4, 10, 11\}$. Fig. 6.8 outlines the average Manhattan distance of every application, the existing ones and the new admitted ($t_6$), after reconfiguration as a function of the new application's priority. When $p_{t_6} = 1$ ($p_{t_1} \leq p_{t_6} \leq p_{t_2}$), the average Manhattan distance of $t_6$ is equal to 4.11 which is bounded by the average Manhattan distance of $t_1$ (4.70) and $t_2$ (3.43). At the same time, the other applications preserved their ideal placement. Using $p_{t_6} = 4$ ($p_{t_2} \leq p_{t_6} \leq p_{t_3}$), $t_6$ is ideally placed between $t_2$ and $t_3$. In contrast, when $p_{t_6} = p_{t_5} = 10$, both applications are striving against one another to gain the ideal placement. However, as we have seen in the last experiment, the second part of the objective function (*Weighted continuity*) prevent from a high migration overhead by minimizing the tiles that change for the high priority applications. For this reason, $t_6$ cannot achieve a similar positioning as $t_5$. To correct this behavior, we increased $p_{t_6}$ to 11, in this case $t_6$ got the ideal placement. Using the same experiments, we highlighted the average total number of changed tiles shown in Fig. 6.9. We

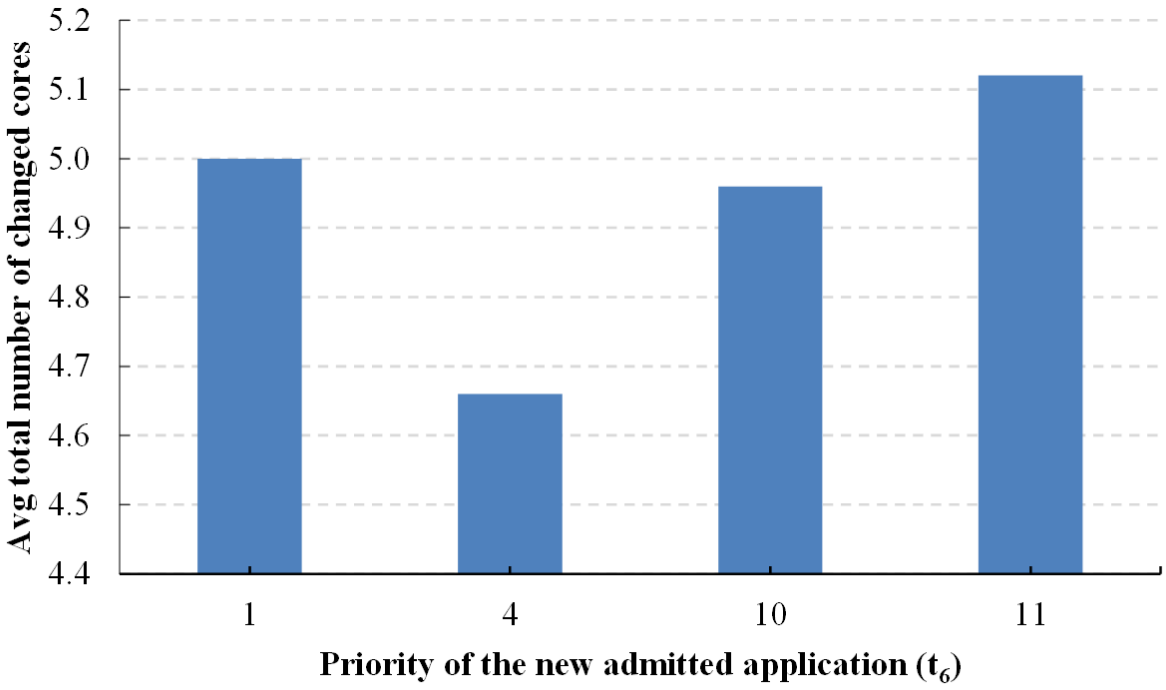



**Figure 6.9:** Average total number of changed tiles as a function of the new admitted application's priority

notice a high average equal to 5, 4.96 and 5.12 when $p_{t_6}$ is equal to 1, 10 and 11, respectively, while, when $p_{t_6} = 4$ it produces the migration of the lowest number of tiles in order to place the new application. This is an important finding in the understanding of the behavior of the optimizer. On the one hand, admitting a new application in the grid's borders, especially when requesters are placed far away from each other, requires pushing many tiles in order to free a space for the new application. On the other hand, placing the new application in any position in the middle of the grid does not oblige many tiles to migrate.

### 6.6.3 Evaluating Simulated Traces

With the goal of experimenting the adoption of our ILP model on a realistic use case, we leverage Phileas [106], a discrete event simulator for Edge and Fog computing environments. More specifically, to reenact the behavior of image processing applications operating in a Smart City environment, we extended an Edge Computing scenario set in Washington DC area [106] to include 8 *applications* representing image processing algorithms, allocated on 4 edge devices, and 4 different user groups interacting with these applications.
In the experiments, we simulated a dynamically varying service demand, either increasing or decreasing workload to trigger the re-configuration of resources on edge devices.
More specifically, the scenario considers a simulation time of 4 days, with workload changes triggering resource reconfigurations every 15 minutes, for a total of 384 reconfiguration events. We used the total number of users in each service during the simulation time as

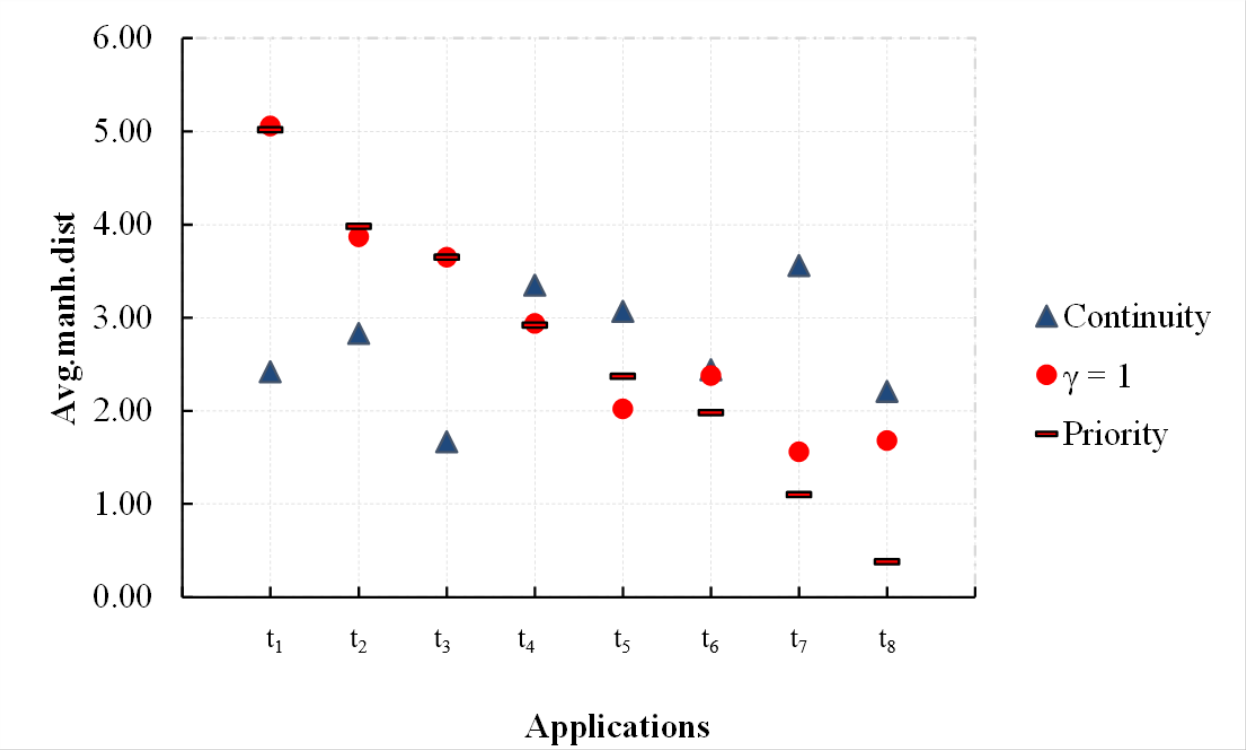



**Figure 6.10:** Average Manhattan distance for dynamically generated traces

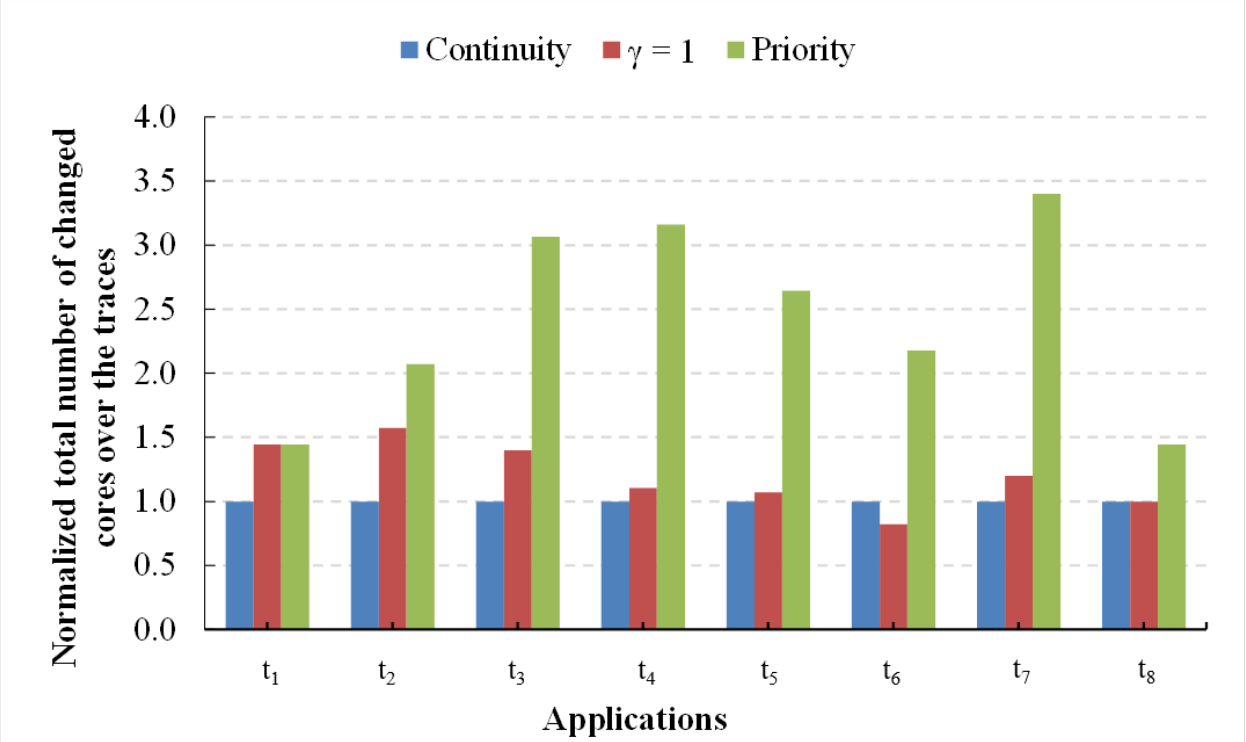



**Figure 6.11:** Normalized total number of migrations over traces

criterion to assign the application priorities, i.e, the application that produced lower number of users will be assigned with lower priority following the notation where $t_1$ has the lowest priority ($p_{t_1} = 1$) and $t_8$ has the highest one ($p_{t_8} = 10^{10}$).

Results are illustrated in Fig. 6.10. With $f^2$, the average Manhattan distance is oscillating and the solver does not perform well when it comes to high priority applications ($t_7$'s and $t_8$'s average are equal to 3.56 and 2.21, respectively). With $f^1$, the results show a descending trend line. In particular, the average Manhattan distance for $t_1$ is 5.02, while for $t_8$ is 0.38. This demonstrates that applications were placed proportionally to their priorities. Using $\gamma = 1$, the trend line is substantially comparable to the previous one except for $t_6$, $t_7$ and $t_8$ where the average is higher than in the preceding results. This is on account of the intention of the solver to minimize the migration of tiles belonging to big partitions and to high priority applications. Another significant outcome of this experiment was that the normalized total number of migrated tiles, presented in Fig. 6.11. The average total number of migrations is normalized for the sake of comparison. Comparing to $f^2$, $f^1$ reaches the highest number of changed tiles which is equal to 3.4 for $t_7$. Using $\gamma = 1$, we observe a decreasing trend except for $t_7$. This outcome provides evidence to the part of the objective function that minimizes migrations of tiles belonging to high priority applications. Together, the present findings confirm that the value of $\gamma = 1$ is the one that obtained the most robust results in term of establishing the balance between $f^2$ and $f^1$.

### 6.6.4 Coping with large grid size instances and with large number of applications and requesters

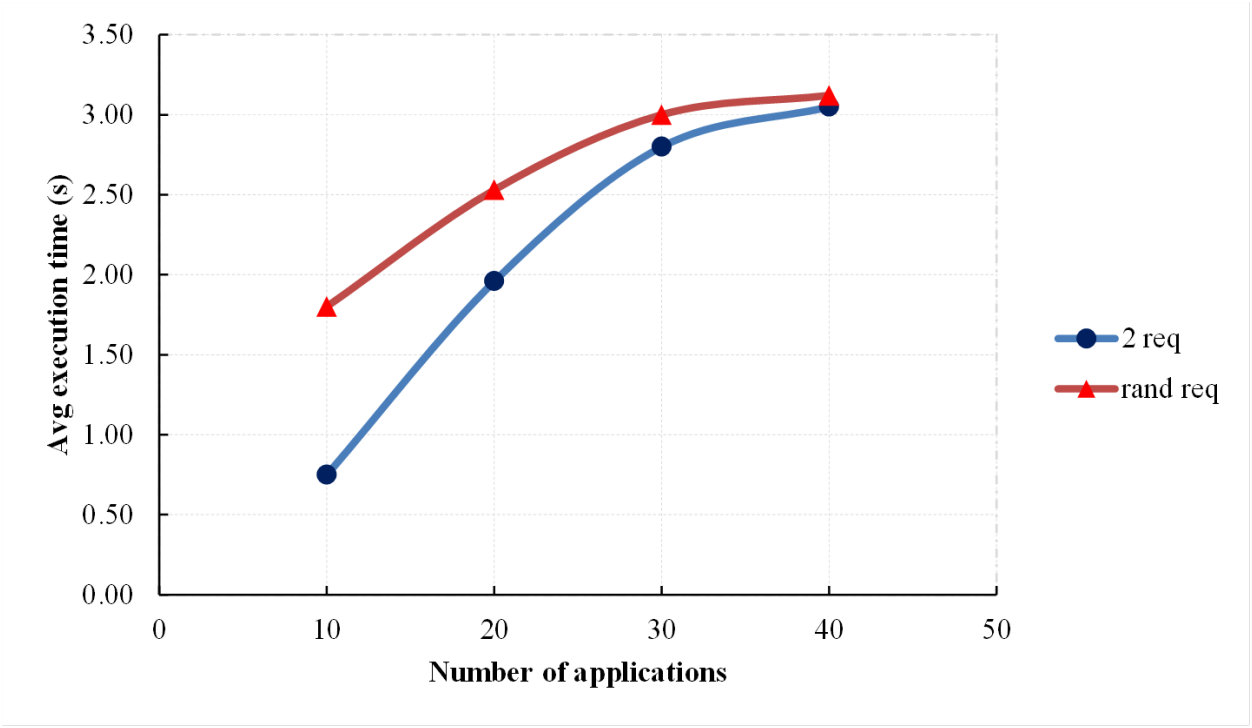



**Figure 6.12:** Execution time as a function of the number of applications and requesters

Finally we challenged the ILP model ability to cope with large size instances and with large number of applications and requesters. The size of an instance is characterized by the

grid size $n$ ($n^2$ tiles), the number of applications $n_T \leq n^2$ and the number of requesters $n_D \leq n_T$. For every grid size $n$, with $n \in \{4, 8, 10, 12\}$, we run 100 scenarios with $n_T = 8$. The grid is always congested (no free tiles). The demands and the number of requesters $n_D$ are randomly computed in every scenario, where $n_D$ is ranging from 2 to $n_T$. Table 6.2

**Table 6.2:** The impact of large grid size instances on the model's running time

| Grid size (n) | 4 | 8 | 10 | 12 |
|---|---|---|---|---|
| Execution time (s) | 0.03 | 0.57 | 1.85 | 2.81 |
| Standard deviation | 0.01 | 0.54 | 2.81 | 4.77 |

shows the impact of increasing grid's size on the running time of the model. On small grids convergence to optimality was reached, on average, after 0.03 ($n = 4$) and 0.57 ($n = 8$). On the contrary, on large grids optimality is attained after 1.85 and 2.81 seconds for $n = 10$ and $n = 12$ respectively. Interestingly, the standard deviation also has the same upward trend as the execution time. The observed increase in the running time is attributed to the growth of the number of binary variables (linearly with n) and the feasible solutions (exponentially with n). For the sake of examining the impact of the number of applications and requesters on the model's scalability, we did the following experiment: for a fixed grid size ($n = 10$) and for every 100 configurations, we identify a number of applications $n_T$, where $n_T \in 10, 20, 30, 40$. As for the number of requesters $n_D$ we used 2 different variants for every $n_T$. In the first (*2 req*), we considers a fixed number of requesters $n_D = 2$. As for the second variant (*rand req*), we randomly generated $n_D$, where $n_D \in [2, n_T]$.

Fig. 6.12 outlines the average execution time over the 100 instances while increasing $n_T$ and using the two aforementioned variants to identify $n_D$. In both cases (*2 req* and *rand req*) the average execution time trend is linear with respect to the number of applications, it ranges from 0.75 for $n_T = 10$ to 3.05 for $n_T = 40$, respectively (When using the *2 req*'s variant). Using *rand req*, the optimizer needs more time than in *2 req* to converge to optimality. In this case, the average execution time is equal to 1.80 and 3.12 for $n_T = 10$ and $n_T = 40$, respectively. In conclusion, the present results revealed that the model's scalability does not depend only on the grid size, but also on the number of applications and requesters.

Overall, measured execution times are typically lower than a few seconds, which is compatible with the online utilization of this solver with a "prior provisioning and prompt allocation" scheme for those realistic Fog computing applications whose workload changes at the time scale of hours.

## 6.7 Conclusions

In this work, we proposed a Partition Manager for the optimal elastic partitioning of a many-core programmable accelerator in high-end shared nodes for Edge computing. We strike a good balance between memory controller proximity correlation to performance criticality and system state perturbation. The parameters of our objective function lend themselves to

online adaptivity to execution state and for dynamic goal management.
This effort represents a first glace toward embodying the elastic computing concept deeper into the hardware/software hierarchies of Edge computing platforms.
Last but not least, the presented Partition Manager can be used to master the hardware support for elastic spatial partitions elaborated in Chapter 5, and can be extended to master the time partitions of Chapter 4 as well.

# Chapter 7

# Conclusions and Future Works

As Fog nodes are gaining momentum as a fundamental component of the Edge-Fog-Cloud continuum, their multi-tenancy is becoming a daunting design and management challenge. Elastic computing is a concept that has been originally conceived for the Cloud, and consists of multi-dimensional properties of resource, quality and cost elasticity. The current challenge consists of extending these properties to Fog platforms as well, as an effective way of consolidating multiple IoT services onto shared public Fog nodes. Unfortunately, current Fog nodes leverage on COTS devices that are not even optimized for the simplest of these properties, that is, resource elasticity.

This thesis has translated the general resource-elastic computing principle into concrete architectural design requirements for advanced Fog hardware platforms, and into concrete requirements for the hardware-dependent resource management layer as well. Then, vertically-integrated research contributions have been placed in both domains, leveraging a barrier-breaking working methodology across related disciplines.

In particular, we believe that future Fog nodes should be designed as parallel computing architectures enforcing isolated and composable compute environments, with the capability to dynamically and elastically reassign resources among consolidated IoT services with minimum runtime overhead. At this level, the fundamental intuition has been to identify the crucial role of the routing mechanism of chip-scale interconnection networks to fulfil these requirements. Therefore, in this thesis a novel dual-layer network-on-chip architecture has been proposed for Fog computing, and support for efficient, flexible and secure partitioning in time and space has been designed. The time-division multiplexed NoC has been designed for low-latency propagation of packets, leveraging the observation of the channel dependency graph, and for schedule reconfigurability through an innovative propagation scheme of scheduling commands. The space-division multiplexed NoC has been designed for communication isolation through the tight control of the partition boundaries. Above all, fast reconfigurability of the partitioning pattern has been achieved through a global unmodified routing algorithm for the network as a whole, and through a lightweight routing logic that selectively processes the routing restrictions depending on the shape of the partition at hand.

Finally, a software Partition Manager has been designed to master the elasticity of spatial

partitions during reconfiguration events. While traditional Application-level Managers for workload-changing systems perform the reassignment of virtual resources based on abstract application and system metrics, the proposed Manager changes virtual assignments into concrete resource allocations. This gives rise to a new optimization problem where the the partitioning pattern is formulated as a mapping of polyominoes, under a strict trade-off between the performance-critical positioning of partitions on the computation grid and the migration overhead of user processes from one tile into another. IoT services are strictly prioritized, and both performance benefits and management overheads are kept correlated (directly or inversely) to such priorities. Overall, this is a highly interdisciplinary piece of work that paves the way for elastic Fog computing, and for a dynamically-orchestrated Edge-Fog-Cloud continuum serving as seamless hosting environment for the next generation of smart IoT services.

Future directions to improve this work include:

- Extension of the pLBDR routing mechanism to non-minimal partition shapes.
- Optimization of the TDM NoC for less buffering resources, trading latency for area and power.
- Definition of a suitable programming model for an elastic Fog node, including awareness of virtual resource allocation.
- Augmenting the resource management framework with a monitoring infrastructure coupled with reinforcement or imitation learning.
- Extend the resource manager to applications with different kinds of QoS requirements (e.g., throughput guarantees vs. low latency responses).
- Co-optimizing the management of spatial partitions with that of time partitions.
- Assessing the trade-offs for using regular vs. irregular partitions from the viewpoint of application and system quality metrics.
- Making the resource management framework aware of technology-related issues, such as power consumption, reliability or thermal control.
- Integration of the Fog node with Edge nodes and with the Cloud for a complete continnum with dynamic topology deployment.

# Bibliography


[1] Code injecton common vulnerability and exposures (cve). `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-14271`. Accessed: 2020-02-19.

[2] Kalray MPPA Manycore 256. http://www.kalrayinc.com/portfolio/processors. Accessed: 2019.

[3] Kuberneets runc common vulnerability and exposures (cve). `https://kubernetes.io/blog/2019/02/11/runc-and-cve-2019-5736/`. Accessed: 2019-06-08.

[4] Parallela reference manual. http://www.parallella.org/docs/parallella_manual.pdf. Accessed: 2019.

[5] Vmware esx.

[6] Visual security and surveillance scenario (3.2). 2017.

[7] Autonomous driving. 2018.

[8] Real-time subsurface imaging. 2018.

[9] A. Abdi et al. Hystery. A hybrid scheduling and mapping approach to optimize temperature, energy consumption and lifetime reliability of heterogeneous multiprocessor systems. *The Journal of Supercomputing*, page 2213–2238, 2018.

[10] A. Psarras et al. Phasenoc: Tdm scheduling at the virtual-channel level for efficient network traffic isolation. In *In Proceedings of DATE*, pages 1090–1095, 2015.

[11] A. Rahmani et al. The dark side of silicon. In *1st ed. Springer*, 2016.

[12] A. Schupbach et al. Embracing diversity in the barrelfish manycore operating system. In *In Proc. of the Workshop on Managed Many-Core Systems (MMCS)*, 2008.

[13] A. Acquaviva, A. Alimonda, S. Carta, and M. Pittau. Assessing task migration impact on embedded soft real-time streaming multimedia applications. In *EURASIP JES*, page 9:1–9:15, 2008.

[14] A.Hansson et al. Phasenoc: Tdm scheduling at the virtual-channel level for efficient network traffic isolation. In *In Proceedings of DATE*, pages 250–255, 2009.

[15] A. Ahmed, H. Arkian, D. Battulga, A. Fahs, M. Farhadi, D. Giouroukis, A. Gougeon, F. Gutierrez, G. Pierre, P. Souza, M. Tamiru, and L. Wu. Fog Computing Applications: Taxonomy and Requirements. In *arXiv e-prints*, page 1907.11621, 2019.

[16] B. Akesson, A. Molnos, A. Hansson, J. Ambrose, and K. Goossens. *Composability and Predictability for Independent Application Development, Verification, and Execution.* Book chapter in "Multiprocessor System-on-Chip", 2010.

[17] K. Akesson. *Predictable and Composable System-on-Chip Memory Controllers*. 2010.

[18] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle. Elasticity in cloud computing: State of the art and research challenges. *IEEE Transactions on Services Computing*, 11(2):430–447, 2018. cited By 73.

[19] G. Alberto, L. Daniele, F. Triviño, A. Strano, J. Flich, J. L. Sánchez, F. Alfaro, M. Favalli, and D. Bertozzi. A complete self-testing and self-configuring noc infrastructure for cost-effective mpsocs. *ACM Trans. Embed. Comput. Syst.*, 12(4):106:1–106:29, July 2013.

[20] A. Aljumah and T. A. Ahanger. Fog computing and security issues: A review. In *2018 7th International Conference on Computers Communications and Control (ICCCC)*, pages 237–239, 2018.

[21] A.Mejia et al. On the potentials of segment-based routing for nocs. In *In Proceedings of ICPP*, pages 594–603, 2008.

[22] S. Anderson, N. Bredeche, A. Eiben, G. Kampis, and M. van Steen. Adaptive collective systems herding black sheep.

[23] B. Saha et al. Enabling scalability and performance in a large scale cmp environment. In *In Proc. of the ACM SIGOPS European Conference on Computer Systems (EuroSys)*, 2007.

[24] M. Balboni. *NoC-Centric Partitionin and Reconfiguration Technology for the Efficient Sharing of General-Purose Prorammable Many-core Accelerators*. PhD thesis, University of Ferrara, 2016.

[25] P. Bellavista, J. Berrocal, A. Corradi, S. K. Das, L. Foschini, and A. Zanni. A survey on fog computing for the internet of things. *Pervasive and Mobile Computing*, 52:71 – 99, 2019.

[26] S. Bertozzi, A. Acquaviva, D. Bertozzi, and A. Poggiali. Supporting task migration in multi-processor systems-on-chip: A feasibility study. In *in Proc. of Date*, page 15–20, 2006.

[27] E. Bini and G. Buttazzo et al. Resource management on multicore systems: The actors approach. In *Micro, IEEE*, pages 72–81, 2011.

[28] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA, 2012. ACM.

[29] T. L. Borden and J. P. Hennesy et al. Multiple operating systems on one processor complex. In *IBM Systems Journal*, 1989.

[30] C. Delimitrou et al. Paragon: Qos-aware scheduling for heterogeneous datacenters. In *in ASPLOS*, 2013.

[31] E. Carara and G. Almeida et al. Achieving composability in noc-based mpsocs through qos management at software level. In *In Proceedings of Design, Automation Test in Europe Conference Exhibition (DATE)*, 2011.

[32] Y. Chen, X. Chen, W. Liu, Y. Zhou, A. Zomaya, R. Ranjan, and S. Hu. Stochastic scheduling for variation-aware virtual machine placement in a cloud computing cps. *Future Generation Computer Systems*, 105:779–788, 2020. cited By 3.

[33] M. Chiang and T. Zhang. Fog and iot: An overview of research opportunities. *IEEE Internet of Things Journal*, 3(6):854–864, Dec 2016.

[34] C. Cicconetti, M. Conti, and A.Passarella. Uncoordinated access to serverless computing in mec systems for iot. In *Computer Networks*, 2020.

[35] Cisco. Enabling MaaS Through a Distributed IoT Data Fabric, Fog Computing and Network Protocols.

[36] J. Colmenares, S. Bird, H. Cook, P. Pearce, D. Zhu, J. Shalf, S. Hofmeyr, K. Asanovic, and J. Kubiatowicz. Resource management in the tessellation manycore os. In *2nd USENIX Workshop on Hot Topics in Parallelism*, 2010.

[37] D. Bortolotti et al. Virtualsoc: A research tool for modern mpsocs. In *In ACM Trans. Embed. Comput. Sys.*, 2016.

[38] D. Lo et al. Towards energy proportionality for large-scale latencycritical workloads. In *in ISCA*, 2014.

[39] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Comm. ACM*, 51(1):107–113, 2008.

[40] E. Dhib, K. Boussetta, N. Zangar, and N. Tabbane. Modeling cloud gaming experience for massively multiplayer online games. In *2016 13th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pages 381–386, 2016.

[41] J. Doppa, J. Rosca, and P. Bogdan. Autonomous Design Space Exploration of Computing Systems for Sustainability: Opportunities and Challenges. In *In IEEE Design and Test*, number 5, pages 35–43. IEEE, 2019.

[42] U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan. Cachier: Edge-Caching for Recognition Applications. In *in Proc. ICDCS*, pages 276–286, 2017.

[43] S. Dustdar, Y. Guo, B. Satzger, and H. Truong. Principles of elastic processes. *IEEE Internet Computing*, 15(5):66–71, Sep. 2011.

[44] E. Bini et al. Resource management on multicore systems: The actors approach. In *in IEEE Micro*, pages 72–81, 2011.

[45] E. Kasapaki et al. Argo: A real-time network-on-chip architecture with an efficient gals implementation. In *In IEEE Transactions on VLSI Systems*, pages 479–492, 2016.

[46] E. Ong et al. System f6: Progress to date. In *In Small Satellite Constellations: Strength in Numbers.*

[47] J. Flich and J. Duato. Logic-Based Distributed Routing for NoCs. *Computer Architecture Letters*, 7(1):13–16, 2008.

[48] W. Fu, T. Chen, C. Wang, and L. Liu. Optimizing memory access traffic via runtime thread migration for on-chip distributed memory systems. In *Supercomputing*, page 1491–1516, 2014.

[49] G. Georgakoudis et al. Scalo: Scalability-aware parallelism orchestration for multi-threaded workloads. In *In ACM Trans. Archit. Code Optim.*, pages 54:1–54:25, 2017.

[50] M. Giordani, A. Zanella, T. Higuchi, O. Altintas, and M. Zorzi. Investigating value of information in future vehicular communications. In *2019 IEEE 2nd Connected and Automated Vehicles Symposium (CAVS)*, pages 1–5, 2019.

[51] M. Gorgues Alonso, J. Flich, M. Turki, and D. Bertozzi. A low-latency and flexible tdm noc for strong isolation in security-critical systems. In *IEEE 13th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, 2019.

[52] B. Grot, S. Keckler, and O. Mutlu. Preemptive virtual clock: a flexible, efficient and cost-effective qos scheme for networks-on-chip. In *In 42th Int. Symposium on Microarchitecture*, pages 269–279, 2009.

[53] H. Yang et al. Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers. In *in ISCA*, 2013.

[54] M.-H. Haghbayan, A. Kanduri, A.-M. Rahmani, P. Liljeberg, A. Jantsch, and H. Tenhunen. Mappro: Proactive runtime mapping for dynamic workloads by quantifying

ripple effect of applications on networks-on-chip. New York, NY, USA, 2015. Association for Computing Machinery.

[55] J. Heisswolf and R. Konig et al. Providing multiple hard latency and throughput guarantees for packet switching networks on chip. In *Computers and Electrical Engineering*, 2013.

[56] S. Heo, K. Barr, and K. Asanovi'c. Reducing power density through activity migration. In *Proc. of ISLPED*, page 217–222, 2003.

[57] R. Hilbrich and J. van Kampenhout. Partitioning and task transfer on noc-based many-core processors in the avionics domain. In *Softwaretechniktrends*, page 1–6, 2011.

[58] H.M.G. Wassel et al. Surfnoc: A low latency and provably non-interfering approach to secure networks-on-chip. In *In Proceedings of ISCA*, pages 583–594, 2013.

[59] C.-H. Hong and B. Varghese. Resource management in fog/edge computing: A survey on architectures, infrastructure, and algorithms. In *Infrastructure, and Algorithms," ACM Comput. Surv.*, 2019.

[60] M. Iorga, L. Feldman, R.Barton, M. Martin, N. Goren, and C. Mahmoudi. Fog computing conceptual model. In *NIST Special Publication.*, 2018.

[61] K. G. J. Dielissen et al. Aethereal network on chip: Concepts, architectures, and implementations. In *IEEE Design and Test of Computers*, 2005.

[62] J. Hamers et al. Scenario-based resource prediction for qos-aware media processing. In *Computer*, 2010.

[63] J. Mars et al. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *in MICRO*, 2011.

[64] J. Jann and L. M. Browning, et al. Dynamic reconfiguration: Basic building blocks for autonomic computing on ibm pseries servers. In *IBM Systems Journal*, 2003.

[65] A. Jantsch, N. Dutt, and A. M. Rahmani. Self-awareness in systems on chip? a survey. *IEEE Design Test*, 34(6):8–26, Dec 2017.

[66] K. Moazzemi et al. Trends in On-chip Dynamic Resource Management. In *21st Euromicro Conference on Digital System Design (DSD)*, pages 62–69, 2018.

[67] R. Kaiser and S. Wagner. Evolution of the pikeos microkernel. 02 2007.

[68] N. Kapadia and S. Pasricha. Varsha: Variation and reliability-aware application scheduling with adaptive parallelism in the dark-silicon era. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1060–1065, 2015.

[69] K. Katre, H. Ramaprasad, A. Sarkar, and F. Mueller. Policies for migration of real-time tasks in embedded multi-core systems. In *in Work-in-Progress Proc. of RTSS*, page 17–21, 2009.

[70] S. Kobbe and L. Bauer et al. Distrm: Distributed resource management for on-chip many-core systems. In *In Proceedings of the International Conference on Hardware/-Software Codesign and System Synthesis (CODES+ISSS)*, pages 119–128, 2011.

[71] A. Kumar, S. Piotr, D. Hashan, R. Mendis, and L. S. Indrusiak. A survey and comparative study of hard and soft real-time dynamic resource allocation strategies for multi/many-core systems. In *ACM Computing Survey*, 2017.

[72] S. Kumaraswamy and M. Nair. Bin packing algorithms for virtual machine placement in cloud computing: a review. In *International Journal of Electrical and Computer Engineering (IJECE)*, volume 9, pages 512–524, 2019.

[73] J. Kyle, M. Nesbit, J. F. A. Cazorla, M. Valero, and J. Smith. Multicore resource management. In *IEEE Micro*, volume 3, pages 6–16, 2008.

[74] L. Tang et al. Compiling for niceness: Mitigating contention for qos in warehouse scale computers. In *in CGO*, 2012.

[75] T. Le Duc, R. Leiva, P. Casari, and P.-O. Oestberg. Machine learning methods for reliable resource provisioning in edge-cloud computing: A survey. In *ACM Comput. Surv.*, 2019.

[76] R. Liu, K. Klues, S. Bird, S. Hofmeyr, K. Asanovic, and J. Kubiatowicz. Tessellation: space-time partitioning in a manycore client os. In *In Proceedings of the First USENIX conference on Hot topics in parallelism.*

[77] M. P. Papazoglou et al. Introduction: Service-oriented computing. In *Commun. ACM*, 2003.

[78] M. Schoeberl et al. A statically scheduled time-division-multiplexed network-on-chip for real-time systems. In *In Proceedings of IEEE/ACM NoCs*, pages 152–160, 2012.

[79] M. Wolf et al. Security in automotive bus systems.

[80] G. Ma, Z. Wang, M. Zhang, J. Ye, M. Chen, and W. Zhu. Understanding Performance of Edge Content Caching for Mobile Video Streaming. In *IEEE Journal on Selected Areas in Communications*, number 5, pages 1076–1089. IEEE, 2017.

[81] R. Marculescu, U. Ogras, N. Li-Shiuan Peh, E. Jerger, and Y. Hoskote. Outstanding research problems in noc design: System, microarchitecture, and circuit perspectives. *IEEE Trans. Comput.-Aid. Des.*, pages 3–21, 2009.

[82] G. Mariani, P. Avasare, G. Vanmeerbeeck, C. Ykman-Couvreur, G. Palermo, C. Silvano, and V. Zaccaria. An industrial design space exploration framework for supporting run-time resource management on multi-core systems. In *In Proceedings of IEEE Conference on Design, Automation and Test in Europe (DATE)*, DATE, page 196–201, 2010.

[83] G. Mariani and V. Sima et al. Using multi-objective design space exploration to enable run-time resource management for reconfigurable architectures. In *In Proceedings of Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1379–1384, 2012.

[84] I. Martinez, A. Hafid, and A. Jarray. Design, resource management and evaluation of fog computing systems: A survey. In *in IEEE Internet of Things Journal*, 2020.

[85] P. Marwedel, J. Teich, G. Kouveli, I. Bacivarov, L. Thiele, S. Ha, C. Lee, Q. Xu, and L. Huang. Mapping of applications to mpsocs. *Conference on Hardware/Software Codesign and System Synthesis*, pages 109–118, 2011.

[86] A. Mejia, J. Flich, J. Duato, S. . Reinemo, and T. Skeie. Segment-based routing: an efficient fault-tolerant routing algorithm for meshes and tori. In *Proceedings 20th IEEE International Parallel Distributed Processing Symposium*, pages 10 pp.–, April 2006.

[87] D. Melpignano, L. Benini, E. Flamand, B. Jego, T. Lepley, G. Haugou, F. Clermidy, and D. Dutoit. Platform 2012, a many-core computing accelerator for embedded socs: Performance evaluation of visual analytics applications. In *DAC Design Automation Conference 2012*, pages 1137–1142, June 2012.

[88] D. Moldovan, G. Copil, and S. Dustdar. Elastic systems: Towards cyber-physical ecosystems of people, processes, and things. *Computer Standards and Interfaces*, 57:76 – 82, 2018.

[89] C. Mouradian, D. Naboulsi, S. Yangui, R. Glitho, M. Morrow, and P. Polakos. A comprehensive survey on fog computing: State-of-the-art and research challenges. *IEEE Communications Surveys and Tutorials*, 20(1):416–464, 2018. cited By 209.

[90] M. Mukherjee, L. Shu, and D. Wang. Survey of fog computing: Fundamental, network applications, and research challenges. *IEEE Communications Surveys Tutorials*, 20(3):1826–1857, thirdquarter 2018.

[91] P. Munk, B. Saballus, J. Richling, and H. U. Heiss. Position paper: Real-time task migration on many-core processors. In *ARCS 2015 - The 28th International Conference on Architecture of Computing Systems*, 2015.

[92] B. Nikoli'c and S. Petters. Towards network-on-chip agreement protocols. In *in Proc. of EMSOFT*, page 207–216, 2012.

[93] B. Nikoli'c, P. Yomsi, and S. Petters. Worst-case communication delay analysis for many-cores using a limited migrative model. In *in Proce. of RTCSA*, page 1–10, 2014.

[94] O. Moreira et al. Online resource management in a multiprocessor with a network-on-chip. In *In Proceedings SAC*, pages 1557–1564, 2007.

[95] O. Stan et al. Protecting military avionics platforms from attacks on mil-std-1553 communication bus. 2017.

[96] OpenFog Consortium Architecture Working Group. Openfog reference architecture for fog computing. In *OpenFog Consortium.*, 2008.

[97] O.Sander et al. A research perspective on fog computing. In *In Int. Conf. on Service-Oriented Computing*, pages 198–210, 2017.

[98] P. Barham et al. Xen and the art of virtualization. In *In Proc. of the ACM Symp. on Operating Systems Principles (SOSP)*, 2003.

[99] P. Kleberger et al. Security aspects of the invehicle network in the connected car. In *In Proceedings of IEEE IV*, pages 528–533, 2011.

[100] P. Ranganathan et al. Enterprise it trends and implications for architecture research. In *in HPCA*, 2005.

[101] P. Patel, M. Intizar Ali, and A. Sheth. On using the intelligent edge for iot analytics. *IEEE Intelligent Systems*, 32(5):64–69, Sep. 2017.

[102] A. Pathania, V. Venkataramani, M. Shafique, T. Mitra, and J. Henkel. Defragmentation of tasks in many-core architecture. *ACM Transactions on Architecture and Code Optimization*, 2017.

[103] J. Perez, A. Gutierrez-Torre, J. Berral, and D. Carrera. A resilient and distributed near real-time traffic forecasting application for fog computing environments. In *Future Generation Computer Systems*, pages 198–212, 2018.

[104] T. Picornell, J. Flich, C. Hernández, and J. Duato. Dcfnoc: A delayed conflict-free time division multiplexing network on chip. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2019.

[105] R. Piscitelli and A. D. Pimentel. Design space pruning through hybrid analysis in system-level design space exploration. In *In Proceedings of IEEE Conference on Design, Automation and Test in Europe (DATE)*, DATE, page 781–786, 2012.

[106] F. Poltronieri, C. Stefanelli, N. Suri, and M. Tortonesi. Phileas: A simulation-based approach for the evaluation of value-based fog services. volume 2018-September, 2018.

[107] B. Pourmohseni, F. Smirnov, S.Wildermann, and J. Teich. Isolation-aware timing analysis and design space exploration for predictable and composable many-core systems. In *In the proceedings of the 31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*, 2019.

[108] B. Pourmohseni, F. Smirnov, S. Wildermann, and J. Teich. Real-Time Task Migration for Dynamic Resource Management in Many-Core Systems. In *Workshop on Next Generation Real-Time Embedded Systems (NG-RES 2020)*, volume 77 of *OpenAccess Series in Informatics (OASIcs)*, pages 5:1–5:14, 2020.

[109] R. Hilbrich et al. Partitioning and task transfer on noc-based many-core processors in the avionics domain. In *In Software technik-Trends*, 2011.

[110] F.-J. Rafael, F.-C. Santiago, J. Segura Garcia, P.-A. Adolfo, and L.-B. Jesus. Elastic computing in the fog on internet of things to improve the performance of low cost nodes. *Electronics*, 8:1489, 12 2019.

[111] A. M. Rahmani, A. Jantsch, and N. Dutt. Hdgm: Hierarchical dynamic goal management for many-core resource allocation. *IEEE Embedded Systems Letters*, 10(3):61–64, 2018.

[112] B. Rimpy, L. Vijay, G. M. Singh, and F. José. D2-lbdr: Distance-driven routing to handle permanent failures in 2d mesh nocs. In *Proceedings of the 2015 Design, Automation and Test in Europe Conference*, DATE '15, pages 800–805, San Jose, CA, USA, 2015. EDA Consortium.

[113] S. Rodrigo, J. Flich, A. Roca, S. Medardoni, D. Bertozzi, J. Camacho, F. Silla, and J. Duato. Cost-efficient on-chip routing implementations for cmp and mpsoc systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(4):534–547, April 2011.

[114] S. Boyd-Wickizer et al. Corey: an operating system for many cores. In *In Proc. of the ACM Symp. on Operating Systems Design and Implementation (OSDI)*, 2008.

[115] P. Saraswat, P. Pop, and J. Madsen. Task migration for fault tolerance in mixed-criticality embedded systems. In *SIGBED Rev*, page 6:1–6:5, 2009.

[116] O. B. Sezer, E. Dogdu, and A. M. Ozbayoglu. Context-aware computing, learning, and big data in internet of things: A survey. *IEEE Internet of Things Journal*, 5(1):1–27, Feb 2018.

[117] S. Shahhosseini, I. Azimi, A. Anzanpour, A. Jantsch, P. Liljeberg, N. Dutt, and A. M. Rahmani. Dynamic computation migration at the edge: Is there an optimal choice? In *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, GLSVLSI '19, pages 519–524, New York, NY, USA, 2019. ACM.

[118] A. Singh and A. Kumar et al. Accelerating throughput-aware runtime mapping for heterogeneous mpsocs. In *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2013.

[119] A. Singh, M. Shafique, A. Kumar, and J. Henkel. Mapping on multi/many-core systems: survey of current and emerging trends. In *In Proceedings of the 50th Annual Design Automation Conference*, pages 1–10, 2013.

[120] S.Rodrigo et al. Cost-efficient on chip routing implementations for cmp and mpsoc systems. In *In IEEE TCAD*, 2011.

[121] A. Strano, D. Bertozzi, F. Trivino, J. Sanchez, F. Alfaro, and J. Flich. Osr-lite: Fast and deadlock-free noc reconfiguration framework. In *SAMOS*, pages 86–95, 2012.

[122] T. Marescaux et al. Introducing the supergt network-on chip. In *In Proceedings of IEEE/ACM DATE*, pages 116–121, 2007.

[123] S. Tai, P. Leitner, and S. Dustdar. Design by units: abstractions for human and compute resources for elastic systems. *Internet Comput.*, 16(4):84–88, 2012.

[124] T.D.Nguyen et al. Towards mil-std-1553b covert channel analysis. 2015.

[125] The Multi-access Edge Computing (MEC) ETSI Industry Specification Group (ISG) and represents the views of those members who participated in this ISG. Multi-access edge computing (mec): Framework and reference architecture. In *ETSI GS MEC 003 V2.1.1*, 2019.

[126] T.Hoppe et al. Security threats to automotive can networks-practical examples and selected short-term counter measures. In *In Proceedings of SAFECOMP*, pages 11–25, 2008.

[127] M. Tortonesi, M. Govoni, A. Morelli, G. Riberto, C. Stefanelli, and N. Suri. Taming the iot data deluge: An innovative information-centric service model for fog computing applications. *Future Generation Comp. Syst.*, 93:888–902, 2019.

[128] M. Turki and D. Bertozzi. An interconnect-centric approach to the flexible partitioning and isolation of many-core accelerators for fog computing. In *XXXIV Conference on Design of Circuits and Integrated Systems (DCIS)*, 2019.

[129] V. Petrucci et al. Octopus-man: Qos-driven task management for heterogeneous multicores in warehouse-scale computers. In *in HPCA*, 2015.

[130] P. van Stralen and A. D. Pimentel. Scenario-based design space exploration of mpsocs. In *In Proceedings of Conference on Computer Design (ICCD)*, page 305–312, 2010.

[131] M. Villari, M. Fazio, S. Dustdar, O. Rana, and R. Ranjan. Osmotic computing: A new paradigm for edge/cloud integration. *IEEE Cloud Computing*, 3(6):76–83, Nov 2016.

[132] P. Vogel, A. Marongiu, and L. Benini. Lightweight virtual memory support for zero-copy sharing of pointer-rich data structures in heterogeneous embedded socs. *IEEE Transactions on Parallel and Distributed Systems*, 28(7):1947–1959, July 2017.

[133] M. Vögler, J. M. Schleicher, C. Inzinger, and S. Dustdar. Optimizing elastic iot application deployments. *IEEE Transactions on Services Computing*, 11(5):879–892, Sep. 2018.

[134] W. Otte et al. Partitioning and task transfer on noc-based many-core processors in the avionics domain.f6com: A component model for resource-constrained and dynamic space-based computing environments. In *In Proceedings of ISORC*, pages 19–21, 2013.

[135] W. Quan et al. A hybrid task mapping algorithm for heterogeneous mpsocs. *TECS'15*, 2015.

[136] A. Weichslgartner, D. Gangadharan, S. Wildermann, M. Glaß, and J. Teich. DAARM: Design-time application analysis and runtime mapping for predictable execution in many-core systems. In *In Proceedings of the International Conference on Hardware/-Software Codesign and System Synthesis*, 2014.

[137] S. Wildermann and F. Reimann et al. Symbolic design space exploration for multi-mode reconfigurable systems. In *In Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, pages 129–138, 2011.

[138] S. Wildermann and T. Ziermann et al. Game-theoretic analysis of decentralized core allocation schemes on many-core systems. In *In Proceedings of Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1498–1503, 2013.

[139] D. William and T. Brian. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[140] J. Wu, B. Zhou, D. Qian, M. Xie, and W. Chen. Elastic resource allocation in the cloud. In *2013 IEEE 16th International Conference on Computational Science and Engineering*, pages 1338–1342, 2013.

[141] Y. Ding et al. Qos aware dynamic time-slice tuning. In *in IISWC*, 2014.

[142] Y. Wang et al. Efficient timing channel protection for on-chip networks. In *In Proceedings of IEEE/ACM NoCs*, pages 142–151, 2012.

[143] Y. Wang et al. Efficient timing channel protection for on-chip networks. In *In Proceedings of IEEE/ACM NoCs*, pages 142–151, 2012.

[144] Y. Zhou et al. Cash: Supporting iaas customers with a sub-core configurable architecture. In *in ISCA*, 2016.

[145] Y. Yang, M. Interlandi, P. Grover, S. Kar, S. Amizadeh, and M. Weimer. Coded elastic computing, 2018.

[146] Y.Lin and H. Shen. CloudFog: Leveraging Fog to Extend Cloud Gaming for Thin-Client MMOG with High Quality of Service. In *IEEE TPDS*, pages 431–445. IEEE, 2017.

[147] A. Yousefpour, C. Fung, T. Nguyan, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. Jue. All one needs to know about fog computing and related edge computing paradigms: A complete survey. In *Journal of Systems Architecture*, 2019.

[148] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang. A survey on the edge computing for the internet of things. *IEEE Access*, 6:6900–6919, 2018.

[149] B. Zhang, N. Mor, J. Kolb, D. Chan, N. Goyal, K. Lutz, E. Allman, J. Wawrzynek, E. Lee, and J. Kubiatowicz. The cloud is not enough: Saving iot from the cloud. In *7th UNENIX Workshop on Hot Topics in Cloud Computing*, 2016.

[150] X. Zhang, J. Schiffman, S. Gibbs, A. Kunjithapatham, and S. Jeong. Securing elastic applications on mobile devices for cloud computing. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, CCSW '09, pages 127–134, New York, NY, USA, 2009. ACM.

[151] S. Zhuravlev, J. Saez, S. Blagodurov, A. Fedorova, and M. Prieto. Survey of scheduling techniques for addressing shared resources in multicore processors. In *ACM Comput. Surv*, page 28 pages, 2012.