



City Research Online

City St George's, University of London

Citation: Zarrin, J., Aguiar, R. L. & Barraca, J. P. (2017). HARD: Hybrid Adaptive Resource Discovery for Jungle Computing. *Journal of Network and Computer Applications*, 90, pp. 42-73. doi: 10.1016/j.jnca.2017.04.014

This is the accepted version of the paper.

This version of the publication may differ from the final published version. To cite this item please consult the publisher's version.

Permanent repository link: <https://openaccess.city.ac.uk/id/eprint/18146/>

Link to published version: <https://doi.org/10.1016/j.jnca.2017.04.014>

Copyright and Reuse: Copyright and Moral Rights remain with the author(s) and/or copyright holders. Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge, unless otherwise indicated, provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way. For full details of reuse please refer to [City Research Online policy](#).

HARD: Hybrid Adaptive Resource Discovery for Jungle Computing

Javad Zarrin, Rui L. Aguiar, João Paulo Barraca

Javad Zarrin {javad@av.it.pt} Instituto de Telecomunicações - Aveiro. Rui L. Aguiar {ruilaa@ua.pt} Universidade de Aveiro, Portugal. João Paulo Barraca{jpbarraca@ua.pt} Universidade de Aveiro, Portugal.

Abstract

In recent years, Jungle Computing has emerged as a distributed computing paradigm based on simultaneous combination of various hierarchical and distributed computing environments which are composed by large number of heterogeneous resources. In such a computing environment, the resources and the underlying computation and communication infrastructures are highly-hierarchical and heterogeneous. This creates a lot of difficulty and complexity for finding the proper resources in a precise way in order to run a particular job on the system efficiently. This paper proposes Hybrid Adaptive Resource Discovery (HARD), a novel efficient and highly scalable resource-discovery approach which is built upon a virtual hierarchical overlay based on self-organization and self-adaptation of processing resources in the system, where the computing resources are organized into distributed hierarchies according to a proposed hierarchical multi-layered resource description model. The proposed approach supports distributed query processing within and across hierarchical layers by deploying various distributed resource discovery services and functionalities in the system which are implemented using different adapted algorithms and mechanisms in each level of hierarchy. The proposed approach addresses the requirements for resource discovery in Jungle Computing environments such as high-hierarchy, high-heterogeneity, high-scalability and dynamicity. Simulation results show significant scalability and efficiency of the proposed approach over highly heterogeneous, hierarchical and dynamic computing environments.

Keywords: distributed operation systems, many-core systems, P2P, resource management, grid computing, DHT

Acronyms

vnode virtual node.
AN aggregate-node.
BRW2 a 2-layered hybrid broad-cast and full random-walk based discovery.
DHT distributed hash table.
DOS distributed operation system.
DPT distributed probability table.
FRW2 a 2-layered hybrid DHT and full random-walk based discovery.
HARD Hybrid Adaptive Resource Discovery.
HARD2 a 2-layered instantiation of HARD.
HARD3 a 3-layered instantiation of HARD.
JCS Jungle Computing System.
LN leaf-node.
PRW2 a 2-layered hybrid DHT, learning-based and partial random-walk based discovery.
QMS Query Management Service.
QREG Query Registry.
QROUT Query Router.
RP Resource Information Provider.
RR Resource Requester.
SN super-node.
SoR Source of Resource.
SQMS Super Query Management Service.

1. Introduction

Large scale distributed computing technologies such as Cloud, Grid, Cluster and High Performance Computing (HPC) super-computers are evolving with the revolutionary emergence of many-core designs (e.g. GPU, CPUs on single die, supercomputers on chip, etc.) and significant advances in networking and interconnect solutions [1]. This has led to increase complexity on the integration of such diverse computing environments, infrastructures, platforms and technologies. Moreover, data distribution, hardware availability, software heterogeneity, and also the sheer size of scientific problems, commonly force scientists to resort to Jungle Computing instead of traditional supercomputers and clusters. Jungle Computing (i.e., leveraging multiple computing platforms simultaneously) is a recent distributed computing paradigm based on concurrent combination of various hierarchical and distributed computing environments with large number of heterogeneous resources [1–8].

In fact, integration of different resources would be necessary specially when no single resource is available that its computation of capacity can meet the computation requirements, or if different parts of the computation have different computational requirements. Furthermore, for large number of users and applications, combination of multiple computing environments with various types of resources leads to achieve high peak performance by potentially accessing a many diverse collection of resources while it is cost efficient. There exist many types of computing landscapes (i.e., isolated computing infrastructures)

that can be efficiently integrated as so-called Compute Jungles at a low cost. However, high heterogeneity (in terms of hardware, resources, connectivities and platforms) and complex hierarchical design of Compute Jungles make them more complex to be efficiently used by scientists.

In a large scale system, where we have a pool of distinct processors, the enabling technology for enhancing the whole throughput of the system is resource sharing. Depending on the computing environment, resource sharing might lead to different issues such as resource allocation, resource provisioning, scheduling, resource description, resource discovery and process migration. Among them, scalable and efficient resource discovery is one of the most challenging issues, particularly for decentralized systems (i.e., resources should be found before we can share them). This is even more critical for future large scale computing environments (e.g., future Cloud, Grid, HPC and Cluster) and also Jungle Computing Systems (JCSs) due to the disparate requirements of these computing environments such as high heterogeneity, high hierarchy and high dynamicity. Resource discovery in JCS can be characterized as a highly adaptive approach (considering the diversity of hardwares, platforms and computing infrastructures) to instantaneously find the most appropriate available set of resources (i.e., computing resources, like as processing cores) for the user applications in the system with minimum cost of communication and computation (i.e., in terms of network latency, network traffic, discovery load, etc.), based on a specific set of computational and communicational requirements for each application or application segments. This has to be achieved in a way that the static and dynamic properties of resources, as well as their interconnected and aggregated characteristics, could be qualified according to the query requirements. Moreover due to the intrinsic complexity of JCS environments, high performance applications then require that resource discovery supports some other important features such as proximity-awareness (i.e., the discovered resources must be close as much as possible) and querying flexibility (in terms of complex querying such as multidimensional querying).

This paper proposes Hybrid Adaptive Resource Discovery (HARD), an efficient and highly scalable resource-discovery approach which deals with the aforementioned resource discovery requirements, applicable to large heterogeneous and highly dynamic distributed environment of JCS. HARD is based on self-configuration and self-adaptation of processing resources in the system, where the computing resources are organized into distributed hierarchies according to a proposed hierarchical resource description model (i.e., multi-layered resource description). Moreover, different algorithms and adapted mechanisms (such as distributed hash tables, distributed probability tables and any-casting) are implemented at different layers in order to efficiently guide queries to proper resources within the specific features of that layer (e.g. leaf-node, aggregate-node and super-node layers).

This work was developed under the framework of the S[o]OS (Service-oriented Operating System) project [9–17], European research project aiming to generate a reference addressing architectures for future very large scale distributed infrastructures. The remainder of the paper is structured as follows. In the next

section, we discuss general requirements and design principles for our work including our hierarchical resource description model. The details of system architecture and HARD are presented in Section III. Section IV presents our algorithms for querying in different layers. Simulation settings and experimental results for the proposed resource discovery are given in Section V. The resource discovery approaches for distributed computing environments in the literature are reviewed and confronted in Section VI and finally Section VII presents our conclusion and future work.

2. Motivation

The motivation behind this work is to propose a resource discovery solution for very large scale distributed systems with respect to the requirements of future large dimensions, many-core enabled, computing systems. Our desired target computing environment can be described as a large scale (ideally, Internet scale), chaotic environment (jungle) of (heterogeneous) processing cores, connected through high speed networks and interconnects and widely distributed across the system. Resource discovery requirements for such future systems are beyond the techniques used in today's Grids, Clouds and HPC clusters. We were inspired by the concept of "Jungle Computing", introduced in the recent years [1–8], and accordingly, we can envision that most of requirements for Jungle can be applied for future computing systems. In fact, two important features of such future environments are: first, heterogeneity of resources; and, second, very large number of resources. With respect to these features, the main objective of this work is to provide a scalable solution.

Current computing systems such as Cloud, HPC and Cluster are generally based on a centralized/hierarchical architecture, leading to the use of centralized resource discovery to respond to user requests. For example, when a request for resources arrives at a HPC cluster-head or a Cloud service provider in the front-end, the resources required can be discovered (allocated or provisioned) by searching in a specified pool of resources or in a back-end data-center where the number and type of resources are known beforehand (this may not be necessarily true in Clouds). For such environments, resource discovery based on a centralized architecture could become an easy task. And, in fact, instead of discovery problem, the resource scheduling issues are dominant.

However, centralized approaches are not scalable and they can not cope with the requirements of future computing systems (future Clouds and HPCs) due to their well-known limitations (bottleneck/congestion issues). This also becomes more critical, as we move toward very large dimension future systems, saturated by a large number of heterogeneous many-core processors, handling a large diversity of tasks. In such a situation, it is not feasible for the control system to have complete and perfect knowledge of the entire system due to the magnitude and diversity of the amount of resources (e.g. processors) and tasks. This is where the role of resource discovery comes into play and becomes very significant to provide on-demand information about the system resources.

Unlike the aforementioned computing systems, decentralized resource discovery approaches are mostly intended/desired for Grids. However, these approaches generally work at task level (due to the Grid nature) in which parallelism of independent tasks is exploited. This makes current Grid discovery methods inadequate to deal with the many-core nature of future computing (in terms of efficient satisfaction of all query constraints) [18, 19], due to the lack of support for thread-level discovery. On the other hand, Operating Systems can perform resource allocation in instruction-level. This provides motivation to go beyond the Grids through the concept of distributed operation systems (DOSs), capable to run on the aforementioned future computing systems. For such DOSs, we used as reference the systems envisaged in the S[o]OS [9–17] project, and propose a thread-level resource discovery approach which works in a fully decentralized, self-determining and autonomous fashion, being able to automatically and efficiently alter any P2P-like underlying computing system to a hierarchically structured overlay. With respect to decentralization and referring to our target computing environment, described above, it is not guaranteed for a processor to have information about all other processors (or even one processor) in the system. In order to address such potential issues, we designed our discovery approach with having this assumption that all processors in the system have equal information initially (and in fact, each processor only knows about itself and its connection gates), leading us to the area of P2P computing.

3. Design Principles

For uniformity of discussion, for JCS environments we will address all kind of control systems as DOSs, although their realization may be quite different (depending on the specific large scale computing technology under discussion).

The general architecture design of a JCS environment can be illustrated as a set of distributed hierarchies like the one shown in the Figure 1. Depending on the level of hierarchies in the architecture and the other designing aspects of a DOS, we may define Control entities in different levels. For instance, as it is discussed in [9], we can describe the entities of Main-Control (i.e., DOS main-kernel), Micro-Control (i.e., DOS micro-kernel) and Nano-Control (i.e., DOS nano-kernel), which are providing either maximal, moderate or minimal amount of capabilities, functionalities and services in the system. These control entities may differ in terms of service types which they can dynamically instantiate on demand. The instances of these entities are positioned in the system in a way to map the structure of the underlying distributed hierarchies (e.g., deploying the main-control instances in the hierarchies head-nodes and the nano-control instances in the leaf-nodes).

3.1. Assumptions and Definitions

HARD is based on methods and techniques to distribute resource information, update and exchange resource data and query and search space exploration, specially when considering

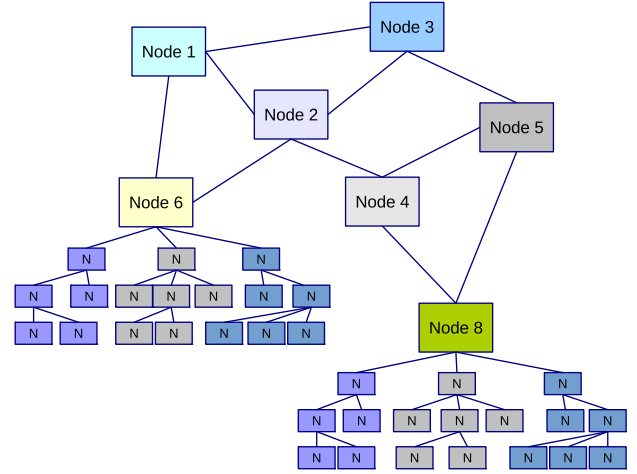


Figure 1: An example of distributed hierarchical architectures.

the particular challenges and requirements of distributed operating systems. In order to create such solution, we impose the following assumptions on the design process:

- We assume that each single resource (i.e., single core) has its own unique ID (e.g., IP, network-ID, Chip-ID, processor-ID) which can specify its address in the entire system. Addressing information can potentially be provided by the operating system or other system component, and is out of scope in this paper.
- We assume that the target computing environment can behave like a P2P distributed environment containing large number of peers (i.e., resource or computing entities) where each peer only knows about itself and its connection gates.

The proposed discovery approach is based on a hybrid virtual overlay network, which will be constructed automatically over the underlying physical network. This virtual overlay contains virtual-nodes that are organized in distributed hierarchies. In this work, we present a 3-layered instantiation of HARD (HARD3), which proposes two levels of resource discovery services (i.e., types of resource discovery components): Resource Requester (RR) and Resource Information Provider (RP). RP services also includes Query Management Service (QMS) and Super Query Management Service (SQMS). We will discuss these services in more detail later in Section 4.1. We will also use the following definitions:

- We define the notion of a “virtual node (*vnode*)” as a group of homogeneous resources, which are not necessarily positioned on a common physical node (e.g., a CPU). Rather they are positioned within a common vicinity that is described by parameters such as number of hops or interconnect latency (i.e., resources are grouped within *vnodes* by proximity and similarity). We use the generic notion of “node” instead of “*vnode*”, which has the same meaning with more emphasizing on the positioning of the “*vnode*” in the underlying self-organized virtual overlay. In addition, in this paper, the term “Source of Resource (SoR)” might

be used instead of *vnode* which also has the same meaning with more emphasizing on the resource-supply-quality aspects such as SoR’s stability or SoR’s strength.

- Every *vnode* is automatically assigned a module-role (i.e., *vnode*-type) in a self-organized and distributed fashion. The module-role defines the specific *vnode*’s role to play in the overall distributed resource discovery operations. We discuss module-roles with more detail in Section 4.1.
- The term “cell” is used to denote a group of *vnodes* which are sharing a common SQMS-ID and the term “mini-cell” is used to refer a group of *vnodes* with a common QMS-ID.
- For better illustration and evaluation of our approach, we use throughout the paper the notion of “resource” to refer to “computational resource” (i.e., physical processing cores) and we discuss resource discovery mostly from the point of view of computational capacity. Nevertheless, HARD is fully generic, and applicable to other types of resources (e.g. storage and networking resources).

HARD3 is implemented as part of the S[o]OS concept [9], which in itself is an example of a DOS. Each DOS kernel, regardless of its level (i.e Main, Micro, Nano, etc.), provides support for a single RR service. Furthermore, the kernels depending on the level which are positioned in the hierarchy provide support for other types of resource discovery services such as QMS and SQMS (e.g., main-kernels or micro-kernels may provide SQMS or QMS services). We must note that it may be possible that a kernel simultaneously provides either one, two or all of these discovery services. For example, the kernels in the top level of hierarchy support all kind of discovery services.

3.2. Resource Description

Resource discovery for the DOS running on the many-core enabled JCS hardware infrastructures requires a scalable and powerful hardware resource description model. In fact, it must deal with two important aspects of resource description: capturing static and dynamic capabilities of hardware resources; and making distribution of hardware information scalable. Considering the aforementioned aspects, we introduce a hierarchical model for dynamic resource description focusing on making the distribution of resource information scalable and able to balance load. It is based in a modular information model that can encapsulate, interface and aggregate the different types of processing hardware information in a hierarchical fashion.

We aim to abstract the characteristics and behavior of underlying hardware infrastructures in a way that both computational and communicational system properties are well represented to provide a close estimation of the real system, while avoiding to describe the hardware at the cycle-accurate level. The description model only focuses on capturing the necessary information that will aid different algorithms along the lines of resource discovery.

The depth of hierarchy (i.e., number of layers in resource description model) and the definition of each layer might range from very high level (e.g. super clusters, clusters) to very low

level (e.g. processing core, ALUs) depending on architecture designing aspects. The layering is performed by partitioning the total number of n peers (i.e., resources) in the system into c disjoint cells of peers (i.e., c is the number of hierarchies in the distributed environment), in the way to ensure that peers in the same layer of each cell are similar to one another in some predefined and inherited attributes. Each layer has two unique related layers: parent layer and child layer. A layer can be either a blank (null) layer with completely no information (definition) or it can be an identifiable layer with a deterministic definition. Furthermore, the properties of the peers in the upper layers are also inherited by the peers in the lower layers (see Figure 2).

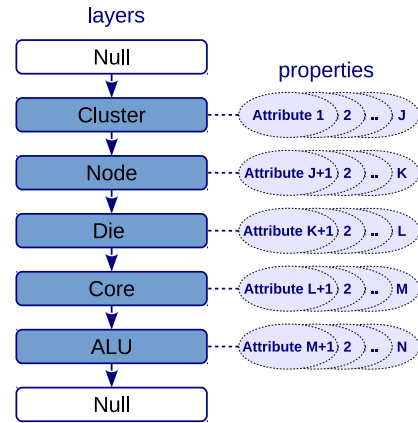


Figure 2: An example of description of resources in hierarchical layers

We must note that the definition of a “resource” (i.e., computing resource) in HARD might be highly variable depending on the levels of the described hierarchy. In fact, HARD refers to a single resource as a representation of any individual peers, belonging to the lowest level of the resource description hierarchy. For example, we can define the ALU layer (i.e., the processing unit or micro-architecture layer) to describe the properties of computing resources in a low level layer with very detail information. The ALU layer also can describe how the ALUs of a processor (CPU or core) gains access to data. In other words it has to decide on the number of the words to be used per cycle, how wide the words should be and whether the word sizes are configurable or not, etc. A HARD resource can represents either a single ALU or a single core if the description model describes either the ALU layer or the Core layer accordingly as the lowest level of the hierarchy. Note that the granularity of the hierarchies will depend on the usage intended for JCS.

According to the proposed resource description we do not need to store locally all the information of a specific resource. The common resource information in each layer, instead of being repeated in all the peers in a local cell, is just maintained in a certain number of peers which are offering resource information services to other nodes in the group or network. Figure 3 depicts the distribution of resource information in two individual cells (group of resources) where due to hierarchy, each cell in each layer stores the common information of the members in the lower layers in certain peers which are providing the information services to the others.

As we need to make the information of the hardware architec-

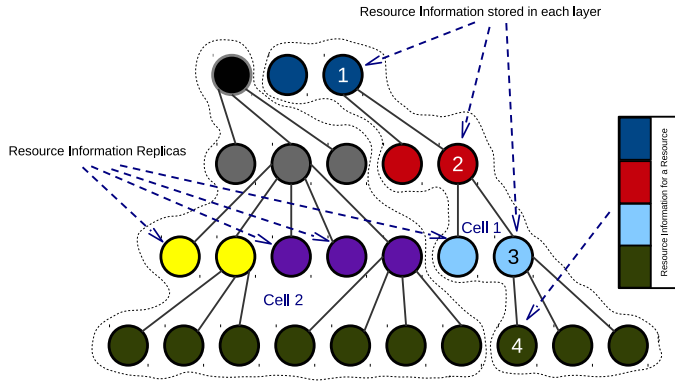


Figure 3: Distribution of resource information in a four-layers-hierarchy with 2 cells

ture and capabilities (i.e., hardware description) available across computing nodes (for efficient code/application distribution) in heterogeneous large-scale dynamic environments, we need to categorize information relevant at the different hierarchies. As the systems are potentially highly dynamic (ephemeral connectivity of mobile devices), capability information might have to be transmitted frequently, imposing the need to minimize overhead of hardware resource description (or capability information) exchange. For this purpose, a hierarchical capability information encoding makes sense, as we can reduce capability information being transmitted to that which is relevant for each specific level in the hierarchy.

We must also note that, one of the primary performance aspect for any hardware description model is related to all forms of communication and data exchange between any two endpoints. It is thereby irrelevant whether the communication takes place between two code instances (threads), residing on two individual single cores, or between a code instance and a storage unit (i.e., between a single core processing unit and a memory unit such as cache or main memory). In all cases, the limiting performance factor is given by the hardware characteristics of the interconnects affected. Furthermore, the degree and types of communication potentially taking place in a system and for an application execution are manifold, ranging from register access over shared data to explicit communication. In all these operations, it is possible that the nodes (individual computing hardwares) in the system, require to exchange their capability information (i.e., resource description). To do this, due to the potential network overhead and traffic, the data size (description size) and the transaction frequency of the description must be reduced as much as possible.

In this paper, for the sake of simplicity in implementation and evaluation of our work, we assume that our description model contains three levels (layers) of the hierarchy, and we refer to this implementation as HARD3. However, depending on the system design concerns and the level of required hardware information details, further layers can also be defined (as described in Figure 2). The following part describes the main lines of information gathered in different layers of our devised resource description model, as implemented in this work:

- Layer 1: Core Layer (Inter-Core Level) describes the individual characteristics of the processing cores as well as cache hierarchies, the connections between processing units, and memory layout/segmentation. Examples of attribute definitions in this layer include cache size, cache associativity, cache latency, memory latency, number of Data-PU channels, number of Instruction-PU channels, vector length, core clock rate (CCR), L1 cache size (L1S), L2 cache size (L2S), number of ALUs (NA), etc.
- Layer 2: Die Layer (Inter-Chip Level) describes the overall properties and behavior of the processing dies (e.g. CPU, GPU, etc.) as well as the interconnection network and topology of the different CPUs to form a many-core machine. Examples of attribute definitions in this layer include BUS frequency, memory bandwidth, memory latency, cache coherence, ISA, micro architecture, interconnection network (INT), size of processor address BUS, processor class, processor type (PT), number of cores (NC), etc.
- Layer 3: Node Layer (Inter-Board Level) describes the overall dynamic and static characteristics of the network nodes containing multiple dies with multiple cores per die. It also describes network topology and inter-nodes communication properties. Examples of attribute definitions in this layer include window size, total number of cores (TNC), memory size (MS), Die count (DC), network bandwidth (NB), network latency, etc.

3.3. Syntaxes and Description Examples

The proposed description model can flexibly describe various "systems" and "queries", ranging from very simple to very complex, while enabling scalable distribution of the resource information across the system. Here, we briefly discuss the features and the syntax of our description language for describing attributes, layers, queries and systems. We store the definitions of layers and attributes in files with extension ".def". We also store the descriptions of queries and systems into files with extension ".des".

3.3.1. Defining Layers and Attributes

Grammar 1 presents the syntax, by Backus-Naur Form (BNF) grammar, for defining layers and attributes using our resource description language. All definitions required for a system design, including layers and attributes, must priorly be declared using the *definition* rule in a .def file. This can be done by using keyword *def*, followed by an *identifier* and sequences of *layer-definitions*. The syntax also supports two other alternative expressions in order to provide capability to define attributes independent of *layer-definitions*. These expressions include either *layer-definitions* as well as definition of other-attributes (i.e., independent attributes) or only definition of independent attributes. In fact, the definition of attributes can be either built-in (i.e., embedded in the layer-definition) or independent. The independent attributes are particularly used to describe queries where the inter-resource and inter-group communications required for a single group or multiple groups of resources need to be clearly

specified in the query. They are specified by using keyword *attributes* at the beginning of a block.

```

<Definition> ::= def <Identifier> <LayerDefinitionList> end
  | def <Identifier> <LayerDefinitionList> <OtherAttributes> end
  | def <Identifier> <OtherAttributes> end
<OtherAttributes> ::= attributes <AttributeDefinitionList> end
<LayerDefinitionList> ::= <LayerDefinition>
  | <LayerDefinition> <LayerDefinitionList>
<LayerDefinition> ::= layer <Identifier> static : <
  AttributeDefinitionList> dynamic : <AttributeDefinitionList>
end | layer <Identifier> <Options> <AttributeDefinitionList>
end | layer <Identifier> <AttributeDefinitionList> end
<AttributeDefinitionList> ::= <AttributeDefinition>
  | <AttributeDefinition> , <AttributeDefinitionList>
<AttributeDefinition> ::= [<AttributeID> <Identifier> <
  AttributeDescription> : <AttributeType>
<Options> ::= dynamic | static
<AttributeType> ::= <Number> | byte | <BitString>
<BitString> ::= bitstring (<Number>, value) { <ValueDefinitionList>
  } | bitstring (<Number>, bit) { <BitDefinitionList> }
<ValueDefinitionList> ::= <ValueDefinition>
  | <ValueDefinition> , <ValueDefinitionList>
<BitDefinitionList> ::= <BitDefinition> | <BitDefinition> , <
  BitDefinitionList>
<ValueDefinition> ::= <Value> = <ValueDescription> | <
  ValueDescription> = <Value>
<BitDefinition> ::= bit<Number> = <BitDescription> | <BitDescription>
  = bit<Number>

```

Grammar 1: The grammar to define layers and attributes in .def files

The keyword *layer* is used to define a layer. A *layer-definition* consists of a list of *attribute-definitions* where each *attribute-definition* individually defines a built-in attribute for its corresponding layer. An *attribute-definition* can be specified by a unique *attribute-id* followed by an identifier, a short description of the attribute and the type of attribute. In addition, we can specify whether the defined attributes are static or dynamic, by using keywords *static* and *dynamic* in the body of the *layer-definition* and before defining each set of attributes (attributes are specified as static by default). Static attributes (e.g., CPU type) represent attributes that do not change at runtime. In contrary, dynamic attributes (e.g., CPU load) will potentially change at runtime. The type of each attribute can be *number* (positive integer), *byte* or *bitstring*. A *bitstring* is defined as a sequence of zero or more bits, by using keyword *bitstring* followed by a number in parentheses, where the number indicates the number of bits required. The type *bitstring* is useful to efficiently represents attributes which their values can be either one of multiple fixed-choices (i.e., *bitstring(number,value)*) or a combination of multiple fixed-choices (i.e., *bitstring(number,bit)*). For example we can specify the type of *AF* attribute (ALU Functionalities) for each core as *bitstring(11,bit)*, where each bit in the binary string (with length=11 bits) indicates whether a specific functionality is supported or not (e.g., "And"=bit0, "OR"=bit1, "Not"=bit2, "XOR"=bit3, "Add"=bit4, "Sub"=bit5, "Mul"=bit6, "Div"=bit7, "ShiftL"=bit8, "ShiftR"=bit9, "Rotate"=bit10). Another example would be the attribute INT which describes the topology of interconnection network for a processor. We can define the type of this attribute as *bitstring(3,value)* where each possible value of the attribute can represent a specific topology as listed as following: "Bus"=0, "Ring"=1, "NOC"=2, "Crossbar"=3, "PointToPoint"=4, "HierarchicalNOC"=5, "Others"=6.

Listing 2 presents a description example, using Grammar

1, which includes definitions of 3 layers: Layer1, Layer2 and Layer3. Each layer consists of definition of 2 to 4 individual built-in attributes. Each attribute definition includes attribute-id, attribute-name, attribute-description and attribute type. The description specifies a set of pre-defined potential values for the bitstring-type attributes (PT and ISA). The code also includes the definition of 4 independent attributes at the end of the *def-block* which define a set of inter-node and inter-group communication attributes to describe queries.

```

def "in-a.def"
  layer Layer1
    [1] CCR "Core Clock Rate by MHz" : number,
    [2] L1S "L1 Cache Size by KB" : number,
    [3] L1L "L1 Latency by ns" : number,
    [4] L2S "L2 Cache Size by KB" : number
  end
  layer Layer2
    [5] PT "Processor Type" : bitstring(2,value) {"CPU"=0, "GPU"=1,
    "FPGA"=2, "Others"=3} ,
    [6] ISA "Instruction Set Architecture" : bitstring(3,value) {"
    X86"=0, "SPARC"=1, "ARM"=2, "XCORE"=3, "RISC"=4, "CISC"=5,
    "Legacy"=6, "Others"=7}
  end
  layer Layer3
    [7] MS "Memory Size by MB" : number,
    [8] DC "Die Count" : number
  end
  attributes
    [9] INB "Inter-node Bandwidth Mb/S" : number,
    [10] INL "Inter-node Latency by ns" : number,
    [11] IGB "Inter-group Bandwidth Mb/S" : number,
    [12] IGL "Inter-group Latency by ns" : number
  end
end

```

Listing 2: An example of definitions for layers and attributes in .def files

3.3.2. Defining Queries

Grammar 3 specifies the syntax for defining queries. A query is defined by using keyword *query*, followed by an *identifier* (the query name) and a *list-of-statements*. The syntax allows using 4 different *statements* including, *import*, *single-node-definition*, *homogeneous-group-definition* and *heterogeneous-group-definition* (statements are separated by using a ";").

The *import* statement is used to import all definitions of layers and attributes, presented in a ".def" file. These definitions (of layers and attributes) are used to specify values for the pre-defined attributes in the *query-block*.

A *single-node* specifies the lowest level of abstraction (for both query and system description) through describing an atomic entity, representing the smallest computing unit in the systems (i.e., we can use the notion of "resource" as an instance of a single-node where each resource can not be divisible into other sub-resources). In fact, we can define a *single-node* depending on the level of abstraction required for querying. For example, a *single-node* can be defined in ALU-level, core-level, CPU-level or node-level. A *single-node* definition contains characteristics required for a single-node by specifying the desired *attribute-value(s)* for a subset of attributes in each pre-defined layer. In other words, a *single-node* definition is a single specification of query requirements for the smallest computing entity in the system. Furthermore, multiple *single-node* definitions (by using different unique identifier for each *single-node*) are allowed in

order to precisely describe all requirements of a query. The language also supports various expressions and operators (such as parentheses, and, or, <, >, <=, >= and =) to specify value(s) for each attribute. We must note that for query description it is not necessary to specify required values for all pre-defined attributes of each layer, rather, depending on the query requirements, conditions for a subset of attributes might need to be specified (non-specified attributes are ignored during query processing). The keyword *all* can be used for layers that do not include any specific attribute conditions.

erally includes keyword *heterogroup*, followed by an identifier (i.e., the *heterogroup* name) and a list of *homogroup-identifiers* (names of *homogroup* members in parentheses and separated by a ”,”). Conditions for communication between *heterogroup* members might be included in the definition of a *heterogroup*. For these conditions, each *homogeneous-group* can be represented by a random member of the group. In other words, the *inter-group-constraints* specify the desired query requirements for communication between each pair of (homogeneous) group-representatives. This can be done by using keyword *igconstraints*, followed by specification of conditions for the required independent attributes.

Listing 4 demonstrates a simple query example that shows a query expression to search for 5 CPU cores with frequency rate of 2000 MHz, L1 cache size of 256 MB, L2 cache size of 512 MB and L1 cache latency of 8 ns. It also indicates that the range of communication latency between the requested resources must be between [20, 130] ns. This example describes a group of homogeneous resources (*HOGroup1*) with indication of inter-resource communication requirements.

```
query Query1
  definition "in-a.def"; // Importing the definition of attributes
  singlenode SingleNode1 // Single resource
    Layer1: CCR=2000, L1S=256, L1L=8, L2S=512;
    Layer2: PT=0; // PT: Processor Type="CPU"
    Layer3: all; // "*" No query constraints for Layer3
  end;
  homogroup HOGroup1(5,SingleNode1) // Homogeneous group of
    resources
    inconstraints: (INL>=20) and (INL<=130); //Inter-node
    communication constraints
  end;
end
```

Listing 4: A query expression example for a homogeneous set of resources in a 3 layer hierarchy

We can also define a heterogeneous group of resources by creating several homogeneous groups (each homogeneous group might have one or several members) and describing the query requirements for communication between those (homogeneous) groups (each homogeneous group can be represented by a random group member). In Listing 5, we have added another homogeneous group (*HOGroup2*) which contains 8 GPU cores with a set of specific attributes in each layer and then in the last part we have created a heterogeneous group by describing the query requirements for communication between groups (*HOGroup1* and *HOGroup2*).

3.3.3. Defining Systems

In our approach, for a DOS, the resource information for every single resources in the system can be extracted from the "system description" by using a component called "resource description provider". These information in turn are encapsulated into layer-stamps (see Section 3.4) and distributed among resources in different levels of hierarchy. Grammar 6 presents the syntax for defining systems. As we can see in the grammar, the syntax used here is identical to the one used for defining queries (see Grammar 3) with the following exceptions:

- The keyword *system* is used to define a system.

```
<QueryDefinition> ::= query <Identifier> <StatementList> end
<StatementList> ::= <Statement> ; <StatementList> | <Statement> ;
<Statement> ::= <Import> | <SingleNodeDefinition> | <
  HomoGroupDefinition> | <HeteroGroupDefinition>
<Import> ::= definition <String>
<SingleNodeDefinition> ::= singlenode <SingleNodeIdentifier> <
  LayerList> end
<LayerList> ::= <LayerExpression> | <LayerExpression> ; <LayerList>
<LayerExpression> ::= <LayerName> : <AttributeList> | <LayerName> :
  all
<AttributeList> ::= <AttributeExpression> | <AttributeExpression> ,
  <AttributeList>
<AttributeExpression> ::= ( <AttributeExpression> )
  | <AttributeExpression> and <AttributeExpression>
  | <AttributeExpression> or <AttributeExpression>
  | <AttributeName> <Operator> <AttributeValue>
<Operator> ::= < > | < > | <= > | >= > | =
<AttributeValue> ::= <Integer> | <Byte> | <Const> | <BitString>
<HomoGroupDefinition> ::= homogroup <HomoGroupExpression>
  inconstraints : <AttributeList> end
  | homogroup <HomoGroupExpression> end
  | homogroups <HomoGroupExpressionList> end
<HomoGroupExpressionList> ::= <HomoGroupExpression> | <
  HomoGroupExpression> , <HomoGroupExpressionList>
<HomoGroupExpression> ::= <HomoGroupIdentifier> (<GroupSize> , <
  SingleNodeIdentifier>)
<HeteroGroupDefinition> ::= heterogroup <Identifier> ( <
  HomoGroupIdentifierList> ) igconstraints : <AttributeList> end
  | heterogroup <Identifier> ( <HomoGroupIdentifierList> ) end
<HomoGroupIdentifierList> ::= <HomoGroupIdentifier> | <
  HomoGroupIdentifier> , <HomoGroupIdentifierList>
```

Grammar 3: The grammar to define queries in .des files

A *homogeneous-group* is defined as a set of *single-node* instances (resources) of a same type (specified by the *single-node-identifier*) with an optional indication of query requirements for communication among single-node instances (resources or group members). In order to define a *homogeneous-group* we can use keyword *homogroup*, followed by an identifier and both *group-size* and *single-node-identifier* (separated by a ”,” and in parentheses). The *group-size* specifies the number of members (resources) in the group. The *single-node-identifier* specifies the type for all members. We can also use keyword *inconstraints* to specify *inter-resource-constraints* if it is required (for a query) to provide conditions for communication between group members. The definition of *inter-resource-constraints* includes the specification of conditions for independent attributes (as discussed in Section 3.3.1). Alternatively, the keyword *homogroups* can be used to define multiple *homogeneous-groups* in a single-block, whenever the specification of inter-resource conditions for those groups are not required by the given query.

Similarly, a *heterogeneous-group* can be defined as a set of *homogeneous-groups* with an optional specification of query conditions for communication among *homogeneous-groups* (inter-group constraints). A definition of a *heterogeneous-group* gen-

```

query Query2
definition "in-b.def";
singlenode SingleNode1 // A single resource
  Layer1: CCR=2000, L1S=256, L1L=8, L2S=512;
  Layer2: PT=0; // PT: Processor Type="CPU"
  Layer3: all; // "*" No query constraints for Layer3
end;
homogroup HOGroup1(5,SingleNode1) // A homogeneous group of
  resources
  inconstraints: (INL>=20) and (INL<=130);
end;
singlenode SingleNode2 // A single resource
  Layer1: CCR=1000, L1S=512, NA=4, TA=0, L2L=15;
  Layer2: PT=1, PC=2, ISA=3; // PT: Processor Type="GPU", ...
  Layer3: WS=90;
end;
homogroup HOGroup2(8,SingleNode2) // A homogeneous group of
  resources
  inconstraints: (INL>=20) and (INL<=50);
end;
heterogroup HEGroup1(HOGroup1,HOGroup2) // A heterogeneous group
  of resources
  igconstraints: (INL>=80) and (INL<=550); //Inter-group
  constraints
end;
end

```

Listing 5: A query expression example for a heterogeneous set of resources in a 3 layer hierarchy (see Table 1 for further details on sample attributes)

- The only supported operator within the *attribute-expression* is the equal sign (“=”), as each attribute of a single-node must provide the exact value for the corresponding attribute.
- In order to precisely describe a system, it is desirable to specify *attribute-values* for most of attributes of each single node in the system. But if this is not feasible, the keyword *none* can be used within a *layer-expression* to highlight that none of attribute-values for the given layer are specified in the description.
- Hierarchies, required for describing different systems, can be built using multiple homogeneous and heterogeneous group definitions. But, unlike query description, the syntax for system description allows heterogeneous groups to be consisted of both homogeneous and heterogeneous groups.

```

<SystemDefinition> ::= system <Identifier> <StatementList> end
...
<LayerExpression> ::= <LayerName> : <AttributeList> | <LayerName> :
  none
<AttributeList> ::= <AttributeExpression> | <AttributeExpression> ,
  <AttributeList>
<AttributeExpression> ::= <AttributeName> = <AttributeValue>
...
<HomoGroupDefinition> ::= homogroup <HomoGroupExpression> end
  | homogroups <HomogroupExpressionList> end
<HomoGroupExpressionList> ::= <HomoGroupExpression> | <
  HomogroupExpression> , <HomogroupExpressionList>
<HomoGroupExpression> ::= <HomoGroupIdentifier> (<GroupSize> , <
  SingleNodeIdentifier>)
<HeteroGroupDefinition> ::= heterogroup <HeteroGroupIdentifier> (<
  GroupIdentifierList>) end
  | heterogroup <HeteroGroupIdentifier> (<GroupIdentifierList>)
<GroupIdentifierList> ::= <HomoGroupIdentifierList> | <
  HeteroGroupIdentifierList>
<HomoGroupIdentifierList> ::= <HomoGroupIdentifier> | <
  HomogroupIdentifier> , <HomogroupIdentifierList>
<HeteroGroupIdentifierList> ::= <HeteroGroupIdentifier>
  | <HeteroGroupIdentifier> , <HeteroGroupIdentifierList>

```

Grammar 6: The grammar to define systems in .des files

Listing 7 presents an example to describe a system containing 5 CPUs with two different types of cores (Core-A and Core-B) (further details of attributes are presented in Table 1). We use this example for discussing the efficiency of our method for information coding later in the next section.

```

system System1
definition "in-d.def";
singlenode Core-A
  Layer1: CCR=2000, L1S=2048, L1L=150, NA=4;
  Layer2: NC=8, PT=0, INT=1, ISA=2;
  Layer3: MS=8192, DC=5, TNC=82, NB=100;
end;
singlenode Core-B
  Layer1: CCR=2500;
  Layer2: none;
  Layer3: none;
end;
homogroups
  CPU0(8,Core-A), CPU1(10,Core-B),
  CPU2(16,Core-A), CPU3(16,Core-B), CPU4(32,Core-B);
end;
heterogroup Config1(CPU0, CPU1, CPU2, CPU3, CPU4);
end

```

Listing 7: A description example for a system containing 5 CPUs with two different types of cores (see Table 1 for further details on Core-A attributes)).

3.4. Coding Efficiency

Referring to the proposed three layers (node-die-core) description model (in Section 3.2), we can describe the relevant hardware capabilities (i.e., the general system attributes) by defining arbitrary number of attributes per each layer. Although the following discussion is very centered in the reference S[o]OS application, its rational can be replicated for other JCS.

All attributes of each layer as well as their values must be represented by a single layer-stamp, which is the concatenation of the hex-decimal values of the attributes (i.e., a fingerprint of all predefined characteristics of a *vnode* in a specific layer). These values are arranged and sorted within the layer-stamp based on the ordering of their attribute identifiers mentioned in the attribute-string (*atr_str*). The attribute-positioning (*atr_pos*) specifies the size of each attribute value correspondent to each attribute identifier in the attribute-string.

The layer-stamp for each layer can be constructed by concatenation of values (binary or hexadecimal values) of its pre-defined attributes with respect to the attribute ordering mentioned in the the corresponding *atr_str*. The resulted layer-stamp can also be decoded by having the values for *atr_str* and *atr_pos*. Depending on the number of attributes in each layer, and the required space to store each attribute-value, the layer-stamp can be longer and consequently need more memory to be stored. Therefore, it is necessary to encode the layer-stamp using a low-cost, efficient encoding mechanism, which can reduce the length of the layer-stamp as much as possible. The expected encoding algorithm must be loss-less and low-cost, with high rate of reduction.

Huffman coding[20] is one of the well-known classic methods to encapsulate data in a way that allows the original data to be perfectly reconstructed from the compressed data (i.e., loss-less data compression)[21–23]. However, the original algorithm might create long-length code-words, which decrease the rate of reduction, and also increase the cost of encoding

Table 1: An example description of general system attributes as well as the capability information for a sample single resource

ID	Attribute	Layer	Unit	Type	Length _{bits}	Value Range	Sample Value	Hex Value	atr_{str}	atr_{pos}	Description
0	CCR	1	MHz	num	16	0-65,535	2000	07D0	0123	FFF7	Core Clock Rate
1	LIS	1	KB	num	16	0-65,535	2048	0800	0123	FFF7	L1 Cache Size
2	LIL	1	NS	num	16	0-65,535	150	0096	0123	FFF7	L1 Latency
3	NA	1	—	num	8	0-255	4	04	0123	FFF7	Number of ALUs
4	NC	2	—	num	16	0-65,535	8	0008	4567	F333	Number of Cores
5	PT	2	—	bit	4	0-15	CPU=0	0	4567	F333	Processor Type (CPU, GPU, FPGA, etc.)
6	INT	2	—	bit	4	0-15	Ring=1	1	4567	F333	Interconnection Network (Bus, Ring, NOC, Crossbar, PointToPoint, HierarchicalNOC, etc.)
7	ISA	2	—	bit	4	0-15	ARM=2	2	4567	F333	ISA (X86, SPARC, ARM, XCORE, RISC, CISC, Legacy, etc.)
8	MS	3	MB	num	16	0-65,535	8192	2000	89AB	F7FF	Memory Size
9	DC	3	—	num	8	0-255	5	05	89AB	F7FF	Die Count
10	TNC	3	—	num	16	0-65,535	82	0052	89AB	F7FF	Total Number of Cores
11	NB	3	MB/s	num	16	0-65,535	100	0064	89AB	F7FF	Network Bandwidth

by larger Huffman tables. We must note that in comparison to Huffman-encoding, other loss-less algorithms such as Run-Length encoding[24], Arithmetic coding[25], Context Tree Weighting[26] and Burrows Wheeler Transform[27] might provide better reduction rate. However, they consume more memory to maintain the required information for decoding, or they require a slower algorithm for data decoding.

In our work, in order to provide an efficient low-cost encoding algorithm, we employ a variation of Length-Limited Huffman (LLH) algorithm [28]. Figure 4 demonstrates the procedures of encoding and decoding hardware resource information in different steps. For each layer, we first, construct the layer-stamps by hexadecimal encoding of the attribute-values, while considering the order of attributes in atr_{str} . In the second step, we encode the layer-stamps using LLH, where the maximum number of symbols is defined as 16 and also the maximum length of each code-word is 16 bits. Using this scheme, we would be able to encode unlimited number of attribute-values, specially for the systems that support large number of attributes per layer.

Figure 5 illustrates the bit-string template that we use to store the LLH coding information (coding-info-string) which consists of symbol, length of the code-word for the symbol and the code-word of the symbol, for the succession of different symbols in the given hexadecimal layer-stamp.

In fact, a LLH-Hex layer-stamp can be decoded using the aforementioned coding-info-string, bearing in mind that according to our defined coding-info-string, the constant length to store each different symbol is 4 bits, and the maximum length of each variable-length code-word is 16 bits. After decoding a LLH-Hex layer-stamp, the resulting hexadecimal layer-stamp can be translated to the attribute-values for each individual attribute in the layer by using the information encapsulated within atr_{str} and atr_{pos} .

In remaining of this section, we show how our resource description model is able to efficiently describe hardware capability information of the individual peers in a system, while it provides an efficient underlying scheme for information transmission and communication between peers during resource discovery. In our example, we assume that we can describe the relevant hardware capabilities, using 4 different attributes per layer. For this purpose, we select and define the general system attributes using Table 1. This table also provides the values of the described attributes for a sample single resource, as described in Listing 7.

As it is shown in Table 1, atr_{str} for the $layer_1$ is 0123, which

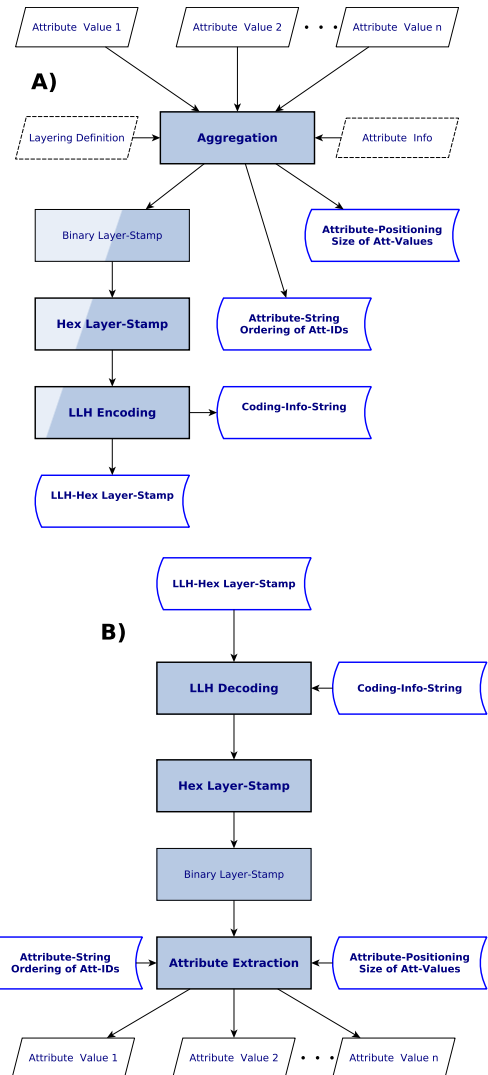


Figure 4: A) Aggregation Procedure, B) Attribute Extraction Procedure

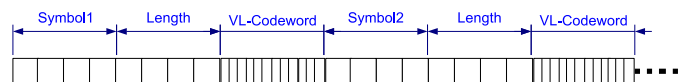


Figure 5: Binary String Template for LLH-Hex Encoding

means that the layer-stamp can be constructed by concatenation of values (binary or hexadecimal values) of attribute 0, 1, 2 and 3. The resulting layer-stamp can be decoded by having

the values of two parameters: attribute-positioning ($FFF7$) and attribute-string (0123). Thus, in order to extract the values of different attributes from the given layer-stamp, the binary stamp, according to atr_{pos} , must be divided to four individual parts with the size of 16, 16, 16 and 8 bits (according to the attribute definitions in Table 1) which are correspondent to the attribute 0, 1, 2 and 3, respectively.

Table 2: Information Encoding

	Layer1	Layer2	Layer3
Hex-smp	07D00800009604	0008012	20000500520064
Bin-smp _{length}	56 bits	28 bits	56 bits
LLH-smp _{length}	30 bits	12 bits	26 bits
LLH-smp _{reduction}	46%	57%	53%
RLH-smp _{length}	26 bits	12 bits	22 bits
RLH-smp _{reduction}	53%	57%	60%

Table 2 provides a comparison between both LLH (our approach) and RLH (i.e., a combination of Huffman and Run-Length algorithms) coding algorithms in terms of reduction of bits in encryption. For this comparison, we use the definition of attributes and their corresponding values (for a sample resource) and also the values of atr_{str} s and atr_{pos} s for each layer, presented in Table 1. The attributes defined are categorized in 3 layers. Hexadecimal-Stamp (Hex-smp) specifies the resulting layer-stamp in hexadecimal for each layer. Bin-smp_{length} shows the length of Binary-Stamp (number of bits) for each layer. LLH-smp_{length} and RLH-smp_{length} are also the lengths of each encoded layer-stamp using LLH and RLH encoding accordingly. Similarly, LLH-smp_{reduction} and RLH-smp_{reduction} demonstrate the rate of reduction for encoded layer-stamps for both methods. The rate of reduction is the ratio of bits-reduction (number of bits for input string minus number of bits for output string) to number of bits for input string. As we can see in Table 2, RLH provides better rate of reduction in the range of [53%, 60%] while the rate of reduction for LLH is [46%, 53%]. However, the rate of reduction is not the only important performance criteria in deciding to use a coding approach, rather, there are other important aspects. Among them the cost of encoding (in terms of memory) is very important. In fact, an encoding process generally creates two type of information in output, including the encoded information and the information which is required for decoding process (e.g., coding-info-string which stores the list of symbols and their corresponding codes). For an encoding process, the cost of encoding specifies the amount of memory which is required to store encoding information (for the purpose of later decoding).

Table 3: Cost of Encoding (Required Space): NOS=Number of Fixed-Length Symbols, BPS=Required Bits per Symbol, BFL=Required Bits for the Fixed-Length, MCL=Maximum Permitted Codeword-Length, MBC=Maximum Required Bits for the Codewords, LID=Length of Sample Input Data by Bits, TCB=Total Cost or Maximum Cost by Bits

	NOS	BPS	BFL	MCL	MBC	LID	TCB
LLH-Hex	$n=16$	4	4	$l=16$	$n * l=256$	any-length	384
RLH-Hex	$n * n=256$	8	4	$l=16$	$n * l=4096$	any-length	7168

In Table 3, we compare RLH to our LLH encoding method in terms of encoding cost. As we can see in the table, LLH creates the maximum cost of 384 bits for each layer encoding while its rate of reduction is more than 30%. It means that using this scheme, we would be able to encode unlimited number of

attribute-values with the maximum memory cost of 384 bits which is very low-cost. Table 3 also shows that despite our LLH algorithm, the maximum memory cost for RLH is too high (7168 bits per each layer encoding). In overall, RLH provides better rate of reduction (compared to LLH), but it significantly suffers from its large encryption cost.

4. HARD Mechanisms

JCS environments are highly-hierarchical and highly-heterogeneous in nature. Accordingly, the HARD architecture deploys various layer-based hybrid adaptive mechanisms (i.e., inter-layers and intra-layers methods) in order to efficiently direct discovery requests to the proper resources across and within layers. This means that, according to properties and characteristics of each layer in the hierarchy, HARD proposes a set of specific adapted methods which have been designed to obtain the maximum discovery efficiency on the target layer while an integrated and coherent approach is used to traverse layers in hierarchy.

4.1. Overall System Architecture

Figure 6 depicts the overall architecture of HARD3, highlighting users, main services, underlying techniques and organization of computing resources in different layers.

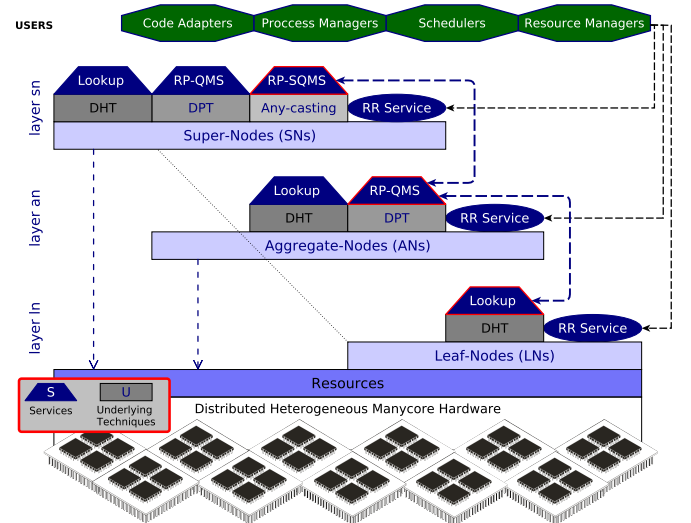


Figure 6: HARD3 Overall System Architecture

We build our system architecture based on a self organized virtual hybrid overlay. In order to create the virtual overlay, at first, the resources in the system are organized within $vnodes$ according to their homogeneity and proximity parameters (i.e., their similarities and locations). In the next step $vnodes$ start to negotiate with each other in a multi-round distributed fashion to seek agreement on the contribution (i.e., module-role or $vnode$ type) of each party in the overlay hierarchy. As negotiations evolve, each $vnode$ shapes its own system-view by improving and consolidating its own knowledge on the entire system. The resulting overlay contains three different types of virtual-nodes: leaf-nodes (LNs), aggregate-nodes (ANs) and

super-nodes (SNs) which take position in $layer_{ln}$, $layer_{an}$ and $layer_{sn}$ of the hierarchy respectively. Depending on the *vnode* type (i.e., module-role), each virtual-node provides different HARD3 services (e.g., QMS and SQMS). *vnodes* in the upper layers are able to provide discovery services specific to their own layer and all the services in the lower layers. *vnodes* respond to the discovery demands based on their module-roles as well as the immediate requirements of the triggered communication events. For example, a *vnode* in $layer_{sn}$ (i.e., a super-node) provides SQMS service. However, depending on the properties of the received communication events, it may also provide QMS or RR services or participate in the overall discovery procedure by playing a role of a leaf-node.

As it is represented in Figure 6, the leaf-nodes in $layer_{ln}$ are organized in distributed hash tables (DHTs) based on the multidimensional fingerprint (layer-stamp) of each participating *vnode*. In fact, the leaf-nodes participate in a core-level specification-based DHT ring where the sibling nodes (i.e., the *vnodes* with similar resources) are linearly organized in linked lists with single entries on the DHT ring. Our proposed DHT ring is a variation of Chord [29] with capability to manage sibling nodes. Similarly, aggregate-nodes (in $layer_{an}$) and super-nodes (in $layer_{sn}$) regardless of their module-role, participate in DHT. For each group of LNs which elect a single AN/SN as their common resource provider (QMS or SQMS), the DHT is composed of all the *vnodes* in the group (containing LN members and either a single AN or a single SN). In other words, all *vnodes*, regardless of their module-role, are able to perform Lookup queries over DHTs (refer to Section 5.1 for a detailed description of the proposed algorithm for DHT lookup).

The reason to use DHT in the core-level ($layer_{ln}$) of our architecture is due to the following aspects: (a) DHT is scalable: this is specially important when considering the potentially large number of cores that can reside on a single die (in future systems). (b) DHT is fast, reliable, fault tolerant and deterministic: resource discovery in the core-level is much more sensitive to speed than querying in the network-level, due to the tightly coupled design of many-core processors. In many-core level, resource discovery might be ineffective if it fails to provide required information in an adequate amount of time (e.g., discovery latency might have a direct impact on the cost of execution migration in a many-core environment). Moreover, in such highly sensitive environments, it is essential for a discovery method to operate reliably and provides deterministic results (undetermined results might have cost by reprocessing the query or exploring an already visited search space). (c) DHT maintenance is low cost (in terms of memory and communication): a very small finger-table is required to be maintained in each *vnode* in $layer_{sn}$. (d) DHT supports attribute-based query description: this makes DHTs more compatible to our attribute-based resource description model. On the other hand, DHTs originally do not support semantic-based querying. To solve this issue, in our DHT variation, we enhanced the original Chord DHT to support a similarity algorithm which makes feasible similar-matching instead of exact-matching (HARD supports both modes of matching through specifying the desired matching mode in the query by the user).

QMS is a service which provides query processing facilities in $layer_{an}$. It uses a probability-based mechanism to guide queries among a group of aggregate-nodes which share a single super-node as the resource provider (i.e., SQMS). During the discovery procedure, distributed probability tables (DPTs) cooperate with each other in a set of dynamic distributed learning processes, which are adapted to the progressive environmental changes. For each AN, its local probability table dynamically collects, aggregates and updates information about the status of the overall resources in the system, gathered from all transacted queries and results through the AN itself. By using this DPT technique, the network that connects ANs becomes increasingly resource-aware, as the number of traversed queries increases across the system (refer to Section 5.2 for detailed description of the proposed methods for DPTs and querying in $layer_{an}$).

We use DPT as a base method in the die-level ($layer_{an}$) due to its scalability, dynamicity, efficiency and also its compact structure. Comparing to DHT, DPT provides probabilistic results instead of deterministic results. But this not a drawback, since DPTs operate in the middle-level of HARD architecture which does not need to provide deterministic results. The reason is that, queries are not going to be concluded in $layer_{an}$. In fact, a query processing starts from the top-level ($layer_{sn}$) (of course, if there exist any query conditions for this layer) and then goes to the middle-level ($layer_{an}$) and finally it could be concluded in the lower-level ($layer_{ln}$). Furthermore, we enhance our DPT approach by introducing a SoR mechanism which can help DPT to provide deterministic results whenever feasible.

The compact design and dynamic nature of DPT provides a facility to efficiently cope with dynamic changes in the environment (e.g., unavailability of resources due to resource failure, resource reservation, etc). Each *vnode* in $layer_{an}$ maintains a small probability table. Depending on the number of predefined attributes in the system, a property table may include multiple records (called resource-type or resource-category records), where each record represents the aggregated probability information for all neighbors with respect to the overall query transaction data (monitoring data), collected and analyzed, for a single attribute over a predefined specific range of values. In fact, each resource-type record includes probability factors for all neighbors as well as a suggestion of a SoR (a *vnode* which deterministically can provide resources, matched with the resource-type definition of the record). We also note that probability tables only cover attributes defined for $layer_{ln}$ and $layer_{an}$. We discuss further details of DPT and SoR mechanisms in Section 5.2.

SQMS is a specific QMS which provides additional capabilities to support query forwarding in $layer_{sn}$. For instance, as we can see in Figure 6, super-nodes (i.e., SQMS providers) are able to concurrently provide multiple services (such as lookup, QMS, SQMS and RR) for the different triggered communication events (refer to Section 4.5). The query forwarding in $layer_{sn}$ uses the specifications of the resources in the node-level to conduct a specification based anycasting method to direct the queries among SNs. It uses the top level layer-stamps to create anycast groups, while nodes in this layer are able to automatically adjust to the anycast group they are interested in based on their specifications in $layer_{sn}$ (refer to Section 5.3). We use anycasting

as a base method for querying in the network-level ($layer_{sn}$) due to its scalability, efficiency and its powerful features which make it more adequate and compatible for resource discovery in computing systems with a large number of network connected nodes (as we discussed in our previous work [30]).

Depending on the specific DOS architecture, HARD3 users could be resource management entities, schedulers, process managers or even code adapters. Each user would be able to perform resource discovery through invocation of a RR entity. RR in turn sends the given query to its local QMS. Due to the type of query and the user's demand (e.g., simple single resource, multiple heterogeneous resources, complex resource graph containing the constraints for inter resource communications, etc.) QMS splits the Main-Query to a set of sub-queries and chooses the appropriate layer that each sub-query must start to process. Finally, the QMS that originated the sub-queries, aggregates the discovery results, and responds the RR with a set of resource matches that optimally satisfy the Main-Query's demand. In the remaining of this section we elaborate more on the mechanisms proposed above.

4.2. Initialization Phase

Initialization procedure is the pre-processing phase of the resource discovery process, where the initialization of the environment variables (e.g. modules configurations) is performed. In fact it is necessary in order to provide the minimum requirements for the execution of the discovery algorithm. The discovery process in the next phase is performed on the basis of these primary settings which consist of the resource grouping and clustering indexes, module roles initialization and allocation of the primary values for the underlying data structures. Some of these settings are directly calculated and stored in the local memory of each node during initialization phase while some others are dynamically updated/recalculated according to different policies and during the discovery procedure. There are also system variables which are required to be set by the system administrator such as grouping policies and grouping thresholds.

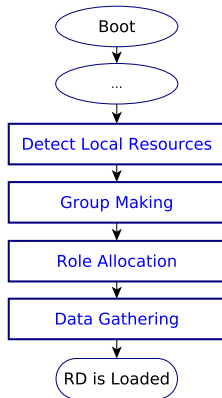


Figure 7: Initialization Phase

In summary, the initialization phase (see Figure 7) of the resource discovery modules consists of the following steps:

1. Self-organization and self-stabilization of the multi-layers communication overlays (zero/auto-configure overlays)

2. Distributed role-allocation (e.g., leaf-node, aggregate-node, super-node)
3. Data gathering and registration (initialization of the successor, probability and neighbor tables)

The details of the algorithm for self-organization and self-configuration of the nodes in hierarchical layers (e.g., $layer_{ln}$, $layer_{an}$ and $layer_{sn}$) have been presented in our previous work in [31].

4.3. Storage and Retrieval

In HARD3 initialization phase, according to a distributed self-configuration mechanism, each single HARD3 instance (running on different $vnodes$) obtains its own instance role (i.e., module-role), which clarifies the future operational behavior of that module instance in terms of discovery. Depending on the module role, each resource discovery instance is responsible for maintaining and updating a set of information in memory, which are the following:

A) The nodes which have the LN role maintain a successor table, a resource state table (i.e., a vector of states for all the resources belonged to a $vnode$), a leaf-stamp, a pointer to the sibling node (i.e., a $vnode$ with similar type of resources in the current DHT-ring) and a QMS-ID. Successor tables in leaf-nodes are created through getting information from the system resource description provider and by leveraging some dynamic algorithms. The leaf-stamp is a key that demonstrates all the characteristics of a computing node (e.g., a processor) in the leaf-node layer and it is generated by extraction and aggregation of the predefined layer's characteristics presented by the system resource description provider. The QMS-ID also specifies the address of a cluster representative which the current leaf-node belongs to (i.e., the address of an aggregate-node in the system which provides QMS service).

B) The nodes which have AN role (i.e., the nodes with QMS functionality) maintain all the information related to the LN layer as well as a probability table, an aggregate-stamp and a SQMS-ID (i.e., Super-Node ID). The probability table will be created and configured by the aggregate-node itself during initialization phase and it would be updated during the resource discovery procedure due to the dynamic behaviors of the HARD3 module instances which are running on the other aggregate-nodes in the system. Furthermore, the aggregate-stamp indicates all the characteristics of a computing node in the AN layer through extraction and aggregation of those properties from the resource description provider.

C) The nodes which have SN role (i.e., the nodes with SQMS functionality) maintain all the information related to both of the LN and AN layers as well as the neighbors table and the node-stamp. The neighbors table provides information about the other super-nodes in vicinity and the node-stamp indicates all the predefined characteristics of a computing node in the SN layer.

It needs to be taken into account that all the instances of the resource discovery modules must have initial states either by using static configuration or performing the initialization

procedure (resulting from a dynamic self-organization of the logical network overlays in the hierarchy). The value for module-role can be LN, AN or SN. We must also note that, each one of the module instances has the capability to act as a leaf-node by default. However they can not play the role of aggregate-node or super-node unless their module-roles clearly are marked as such. Moreover, the role of each module instance can be changed dynamically during the resource discovery procedures, and system self-organization.

4.4. Resource Requester and Resource Information Provider

RRs are the module instances that query RPs on behalf of HARD3 users (i.e., the resource discovery users or the system components such as resource manager or process manager which need to discover resources for purposes like resource allocation) for their needed resource information and the RPs are the module instances that provide information services to other RPs and RRs. As it is shown in Figure 8, RR operations can be summarized according to the following steps:

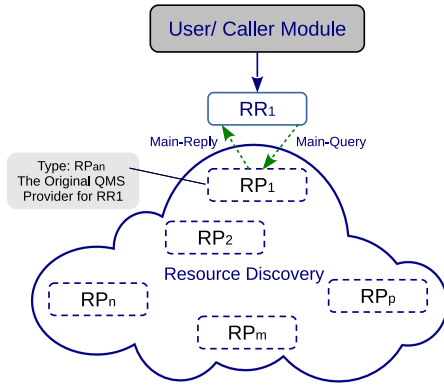


Figure 8: Resource requester general behavior.

(1) RR reads memory to get the description of resources which are demanded by a user (reading and analyzing the user's query description). In accordance with our hierarchical resource description, the description of the required resources can be accurately reflected in a flexible query description. A general query might be described as a single group of heterogeneous resources, which contain several homogeneous group of resources. The description of each homogeneous group represents the group characteristics such as number of desired resources, static and dynamic desired properties of resources in each standard layer and the required inter-resources and inter-groups communication properties.

(2) Using the query description set by the user, RR creates a Main-Query message and sends it to its local QMS. The local QMS is the default self-configured RP for RR. Each local QMS, on behalf of its RR clients, manages all the relevant discovery steps for a Main-Query in the distributed system.

(3) Later, RR receives a main-reply message corresponding to its discovery request. This consists of information on the discovered resources which are pre-reserved for the user (i.e., application segments belonging to the user). RR is able to either release them or reserve these resources for a longer time period. The discovery temporary reservation for each resource

will automatically end after a certain time period if the related RR makes no decision on the reservation policy. Moreover, a RR will release the reserved resources when those resources are not needed any more by the user (i.e., application execution is terminated).

Unlike the irrelevancy of RR behavior to the module's role, the RP algorithms and mechanisms are mostly dependent on the module's role. For this reason, we elaborate on the details of these algorithms in Section 5, explaining the behavior of the HARD3 modules from RPs viewpoint.

4.5. Communication Events

After the initialization phase has been completed, system actions are concentrated along the line of maintaining resource information, handling queries and managing unpredictable changes in the system configuration. In order to handle queries and route discovery requests to the proper resources within and across layers, we have defined several types of events (see Figure 9 and Table 4) where each event specifies the type of communication and also the necessary actions that are required to be done by a receiver node. The receiver node basically acts as an event handler responsible for handling the triggered event. These events are the following:

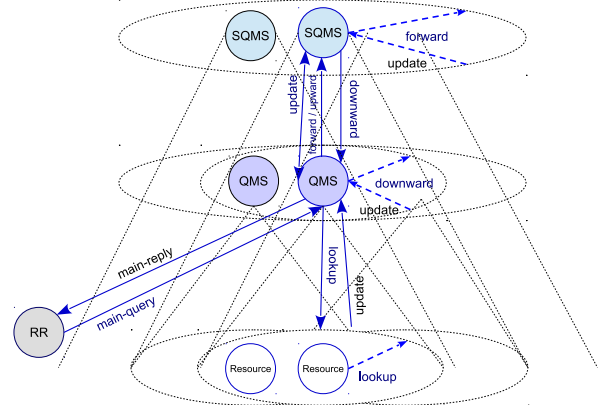


Figure 9: Communication events within layer and across layers.

Lookup event: DHTs have been used to store the highest details of the resource information in the lowest layer of the resource description hierarchy, therefore all the peers in a mini-cell (i.e., a group of *vnodes* with common QMS-ID and SQMS-ID) participate in a ring-based DHT. It is not necessary to have a single or flat DHT, rather in each mini-cell, for each dimension, a flat/hierarchical ring can be used. RPs trigger Lookup events in order to search the entire leaf-nodes in the local mini-cells for the desired resources.

Update event: Once a query is successfully resolved in a *vnode*, an Update event is triggered in order to inform the original resource requester and all the intermediate resource providers on the result of the corresponding query. The response (Update) message follows the reverse path taken by the query message through the overlay network and updates the values for the resource-type-depended variables such as probe factors and preferred-nodes in the probability tables for the next usage.

Table 4: Inter-layer and Intra-layer Communication Events

Event	Source Event Generator			Target Event Handler			Description
	LN	AN	SN	LN	AN	SN	
Lookup	yes	yes	yes	yes	yes	yes	DHT lookup in layer _{ln} , Intra-layer communications in layer _{ln} , layer _{an} → layer _{ln} (inter-layers communications)
Upward	no	yes	no	no	no	yes	Ensuring that the lower layers in the current cell (a group of <i>vnodes</i> with common SQMS-IDs) have already been searched with no results, layer _{an} → layer _{sn} (inter-layers communications)
Forward	no	yes	yes	no	no	yes	There is no assurance that the lower layers of the current cell are explored, Intra-layer communications in layer _{sn} , layer _{an} → layer _{ln} (inter-layers communications)
Downward	no	yes	yes	no	yes	no	Ensuring that the query conditions are met in the upper layer (layer _{sn} but the query is still not fully resolved and further search is required to be carried out in the lower layers., Intra-layer communications in layer _{an} , layer _{sn} → layer _{an} (inter-layers communications)
Downward _{cbk}	no	yes	no	no	yes	no	Ensuring that all the potential children of the current vertex (i.e., an aggregate-node or a <i>vnode</i> with QMS functionality) in the layer ₂ search tree have already been explored. Further search is required by calling back to the parent of the current vertex. The probability table in receiver nodes must be updated. Intra-layer communications in layer _{an}
Update _{qms}	yes	no	no	no	yes	no	The result (i.e., full result, partial result or no result) of a Lookup query is delivered to the caller entity which is the QMS of the event generator (event initializer), layer _{ln} → layer _{an} (inter-layers communications)
Update _{sys}	no	yes	no	no	yes	yes	Ensuring that a query (sub-query) is completed (with partial or full results). The result is delivered to the original QMS requester (i.e., the QMS which has registered the original query). The intermediate entities must be updated. Intra-layer communications in layer _{an} and layer _{sn} , layer _{an} < → layer _{sn} (inter-layers communications)
Update _{vic}	no	yes	no	no	yes	no	The QMS of the aggregate-nodes in the vicinity of the event generator are required to be updated on their knowledge of source of resource. Intra-layer communications in layer _{an}
Update _{null}	no	yes	no	no	yes	yes	Ensuring that a query (sub-query) is unusually completed (with partial or null results). This could happen when all the potential <i>vnodes</i> in the system are explored or the query is expired. The result is delivered to the original QMS requester. Updating is not required for the intermediate entities. Intra-layer communications in layer _{an} and layer _{sn} , layer _{an} < → layer _{sn} (inter-layers communications)

Generally, the leaf-nodes return updates with the whole results, with partial results, or with no results.

Upward event: This is an inter-layer communication event received by a SQMS provider in the super-node layer. It notifies the SQMS that the lower layers in the current cell were already explored with no result, and the discovery process must be continued by directing the query to other SQMS in the system.

Downward event: In order to resolve a query, due to the hierarchical structure of the resource information and query descriptions, each query starts the discovery process from the higher layer and when the resource description in a layer satisfies the query conditions for that layer, the query must be sent downwards to the lower layer which supports adequate detail information. In fact queries are traversing between layers to extract more accurate information of the desired resources. In other words, a Downward event is generated by a RP in an upper layer and it will be send to another RP in the lower layer, so the recipient RP can be sure that the query conditions in the higher layers already have been achieved.

Forward event: This is the major communication event in the super-node layer, which alerts the receiver that the lower layers of the current cell are not explored yet. It is obvious that the lower layer exploration is only required if the query conditions in the super-node layer are met. Forward Event can also be generated by the original QMS of a query in the lower layer and received by the SQMS in the upper layer.

Main-Query event: Once a HARD3 instance, running on a particular processing node, receives a discovery demand from a user, the RR component starts by generating a Main-Query. This happens when the local processing resources are not sufficient for an efficient application execution, and some extra remote resources are required to improve the execution performance. The resource requester will trigger the Main-Query event in a target node in its local mini-cell which provides query

management service (i.e., QMS).

Main-Reply event: Upon completing a Main-Query a Main-Reply will be created by the original QMS of the Main-Query to inform the HARD3 user on the overall result of the discovery request. Depending on the states of the sub-queries, the discovery reply for a Main-Query may contain full results, partial results, or no results.

5. Algorithms

5.1. LN Algorithm Description

RP_{ln}s (i.e., *vnodes* with the module-role of LN) operate as following (see Algorithm 1):

(1) Upon receiving a Lookup request in a receiver node which is denoted as *RP_{ln}*, it checks its local resources in order to find an available match/matches that meets/meet the requester criterias described in the Lookup message. For doing this, the *RP_{ln}* verifies if its leaf-stamp is validated either in terms of equality or similarity (based on the degree of similarity returned by a similarity function) by the lookup-key (i.e., the description of the query conditions for the layer_{ln}) mentioned in the Lookup message. It must be taken into account that the acceptable similarity degree to resolve queries can be a constant value for all the queries, predefined in the entire system, or it can be variable for each query.

(2) If all the resources required by the query are found in the local set of resources (i.e., if the query is fully resolved), the current lookup procedure exits and, consequently the *RP_{ln}* creates and sends an Update_{qms} message to the local QMS (i.e. the aggregate-node, providing the QMS service to *RP_{ln}*, that has initiated the current lookup procedure) containing the information on the matched resources in the current leaf-node. In other cases, if some partial results are achieved or there are

Algorithm 1: Processing a Lookup sub-query by a RP_{ln}

```

Input: sq= The received Lookup sub-query
/* sq, temp:sub-query, ft:finger-table of  $RP_{ln}$  */
/* for a sub-query, nRR is the number of resources requested,
nDR is the number of already discovered resources, DRs is
the list of IDs for already discovered resources */
temp=sq; temp.nRR= sq.nRR- sq.nDR; temp.nDR=0
idx=ft.get-entry-index(temp.cln) // checking the lower bound of the
finger table ft, if idx=NULL ⇒ there is no an entry index
for temp.cln in the finger table, an entry index refers to a
possible range of key values defined in ft, ft specifies a
successor-node-id for each entry index
if  $RP_{ln}$  is qualified due to temp.cln then
    matched-resources=check-local-resources(temp)
    temp.DRs.push(matched-resources); reserve(matched-resources, temp)
    temp.nDR=matched-resources.size(); sibling-node=hasNextSibling()
    if (sibling-node) ∧ (temp.nDR < temp.nRR) then
        temp.nRR=sq.nRR; temp.nDR=temp.nDR+sq.nDR
        send(Lookup, sibling-node, temp)
    else
        temp.nRR=sq.nRR; temp.nDR=temp.nDR+sq.nDR
        temp.preferred-vnode=check-sor-capabilities(temp)
        temp.visited-qms-ids.push(qms-id)
        send(Updateqms, qms-id, temp)
else if ( $RP_{ln}$  is not qualified with respect to temp.cln) ∧ (idx) then
    temp.nDR=0; temp.nRR=sq.nRR; temp.nDR=temp.nDR+sq.nDR
    temp.visited-qms-ids.push(qms-id)
    send(Updateqms, qms-id, temp)
else
    if (vnode-id==maxRank) ∨ ( maxKey < temp.qls) then
        /* checking the upper bound of the finger table ft */
        temp.nDR=0; temp.nRR=sq.nRR; temp.nDR=temp.nDR+sq.nDR
        temp.visited-qms-ids.push(qms-id)
        send(Updateqms, qms-id, temp)
    else
        next-node=ft.get-successor-id(idx)
        forward(Lookup, next-node, sq)
        /* sending the input sub-query to the next-node in the
        DHT without change */
return

```

no resources available in the local set of resources, while the leaf-stamp of the current node is validated by the query's lookup-key, the RP_{ln} checks if it has sibling-node. On the existence of a sibling node, the Lookup message will be redirected to the sibling-node, while the content of message is updated for the partial results achieved in the current node and, if the sibling-node has not exist, similar to what is performed on a full query resolution, an Update_{qms} message containing partial result or no result will be sent to the local QMS. It must be taken into account that each lookup query contains information such as the number of resources required, the number of resources discovered and the properties of the resources desired in layer_{ln}. The receiver of a Lookup message identifies the number of resources, which are currently needed to be discovered due to the information extracted from the message content. This information includes the lookup history, which clarifies on the resources which are already discovered in other SoRs. Sibling-nodes are the *vnode* which provides similar resources. These *vnodes* are not directly participating in the DHT, rather among each group of sibling-nodes only a single member joins the DHT of the leaf-nodes in the current mini-cell. In fact sibling-nodes are the hidden members of the DHT, which are called whenever more SoRs of a particular type of resource are needed. Moreover, each group of sibling-nodes constructs a chain where each node knows only its single sibling-node. The first node of the chain directly gets position in the DHT while the last-node ends the search in the sibling-nodes.

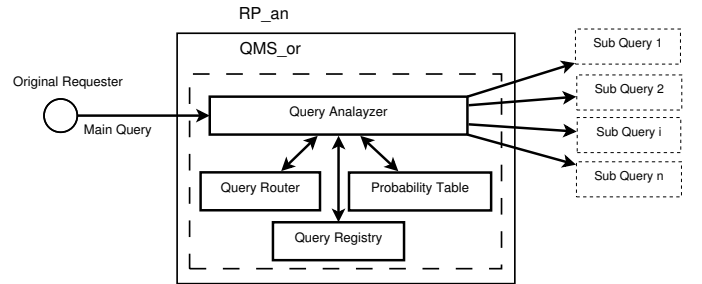
(3) If the lookup-key fails to validate the leaf-stamp of the current node, thus, according to the DHT properties (i.e., the properties of the DHT ring which has been created through participation of a group of leaf-nodes in the LN layer) and the local successor-table of RP_{ln} , the lookup ending conditions would be examined to determine whether they are satisfied or not. Accordingly, there would be two possibilities: (a) If a lookup ending condition is reached, the lookup procedure ends, and the RP_{ln} sends an Update_{qms} message to its QMS address containing information, which ensures the local QMS about the non-existence of available matches for the lookup query constraints in the leaf-node layer, while the search space has efficiently been explored. (b) If no one of the lookup ending conditions is reached, the RP_{ln} redirects the Lookup message to the next RP_{ln} in the DHT by using its successor-table.

5.2. AN Algorithm Description

RP_{ans} (i.e., *vnodes* with the module-role of AN which provide Query Management Service) operate as following:

RPs assign several different event-handlers to manage the communication events. They also receive and inspect each incoming message to identify the event that must be triggered. Depending on the message type in the client-side (source event generator) and the *vnode* type (i.e., node type or module-role) in the server side (target event handler) various operations (i.e., event handler functions) might be performed by receivers. Below we elaborate on the event receivers' behavior (event-handling functions) for different events when the node type of the receiver node is set to AN.

The Lookup event handling in the RP_{ans} is similar to Lookup event handling at the RP_{lns} . The only difference is that the AN receiver nodes play the role of a LN.

Figure 10: Main query processing in QMS_{or}

The RP_{ans} are the major targets for the Main-Query events. Whenever such event occurs in the receiver side, that is supposed to be the local QMS of the requester, several tasks are consequently performed by different sub-components of the QMS (e.g., Query-Analyzer, Query Router (QROUT), Query Registry (QREG), etc) (see Figure 10). At first, the Main-Query is registered in the QREG. This specifies the current QMS provider, as the main responsible entity to collect and manage the overall results of the Main-Query. Afterwards the Query-Analyzer splits the Main-Query in multiple sub-queries (i.e., queries) based on the Main-Query description and the homogeneity of resources in each sub-query. These sub-queries in turn update their query-routing information in the QROUT. Depending on the sub-query

conditions for the different layers, the Query-Analyzer makes a decision for each individual and independent sub-query based on their query-schemes, focused entirely towards conducting the best possible exploration pathway around the system.

For the sake of simplicity, in the rest of this paper, we use a query-scheme to represent sub-query conditions in different layers (the number of homogeneous resources required for each sub-query is a separated query argument which is not shown in the query-scheme). A query-scheme represents all sub-query conditions in 3 layers as $\langle c_{ln} \cdot c_{an} \cdot c_{sn} \rangle$. Here, c_{ln} , c_{an} and c_{sn} denote the existence of query constraints for the layer_{ln}, layer_{an} and layer_{sn} accordingly, while n_{ln} , n_{an} and n_{sn} determine non-existence of any query-conditions on those layers.

The strategy to split a query by Query-Analyzer is related with the description of the query. The Query-Analyzer simply splits the query (a heterogeneous group) to multiple sub-queries (multiple homogeneous groups), as it is originally described by the query description (see Section 3.3.2). For example, for the query description presented in Listing 4, the Query-Analyzer can only create a single sub-query, since the query description only contains one homogeneous group (HOGroup1). The resulting sub-query aims to find 5 similar processing cores with the characteristics described by SingleNode1. As it is shown in Listing 4, the sub-query does not provide any condition in Layer3 ($\langle c_{ln} \cdot c_{an} \cdot n_{sn} \rangle$). Similarly, in other example for the query described in Listing 5, the Query-Analyzer splits the query into two sub-queries (sub-query1 and sub-query2), since the query description contains two homogeneous groups (HOGroup1 and HOGroup2). The sub-query1 is similar to the one discussed for Listing 4. The sub-query2 aims to find a group of similar resources, including 8 processing cores with attributes described by SingleNode2. The query-scheme for sub-query2 can be presented as $\langle c_{ln} \cdot c_{an} \cdot c_{sn} \rangle$, since it provides conditions in all layers.

The conduction of sub-queries by the Query-Analyzer can be done as following: (a) for the sub-queries with the query-scheme $\langle c_{ln} \cdot n_{an} \cdot n_{sn} \rangle$, the RP_{an} initiates a DHT lookup search (i.e., searching in the local mini-cell) by sending a DHT Lookup message to the entry node in the DHT ring. (b) if the query-scheme is $\langle c_{ln} \cdot c_{an} \cdot n_{sn} \rangle$, the RP_{an} first checks if its aggregate-stamp is validated according to the aggregate-key of the query. In other words, it determines whether the layer_{an} information of the local QMS fulfills the sub-query conditions in this layer or not. Thus, if the layer_{an} information of the current node is validated by the c_{an} query conditions, the RP_{an} continues with the DHT lookup, otherwise, the RP_{an} , leveraging the probability table, selects the next QMS (i.e., another RP_{an} in the current layer which represents a mini-cell with the higher probability to find the requested resources) and sends a Downward discovery message towards the QMS address of the next mini-cell. (c) if the query-scheme is $\langle c_{ln} \cdot c_{an} \cdot c_{sn} \rangle$, the RP_{an} generates and sends a Forward discovery message towards the higher layer which offers the SQMS service (the RP_{an} relays the query to its SQMS address in the super-node layer).

Figure 11 demonstrates examples of query processing for sub-queries with different schemes and due to the decisions made by the Query-Analyzer. Upon receiving a request from a HARD

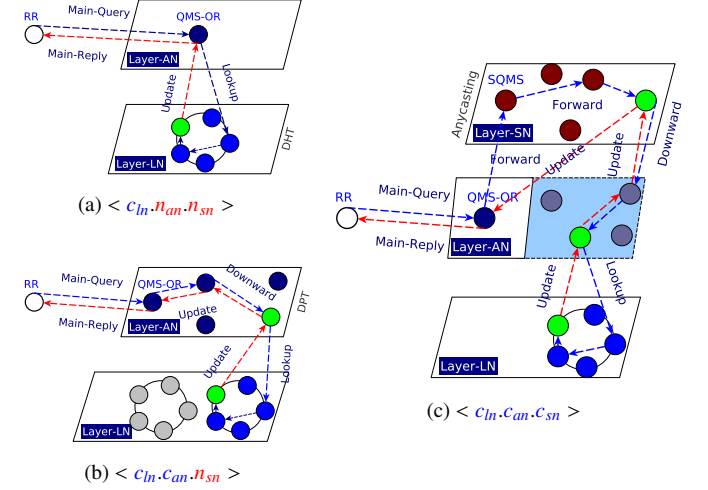


Figure 11: Examples of sequence of the resource discovery message flow for different sub-query schemes (the Main-Query contains a single sub-query).

user, the RR entity starts the discovery procedure by sending a Main-Query to its local pre-assigned QMS provider (QMS_{or}). The Main-Query includes only a single sub-query. In Figure 11-a, the sub-query scheme is $\langle c_{ln} \cdot n_{an} \cdot n_{sn} \rangle$, which means that the query only introduces conditions for the layer_{ln}. Thus, the query is delivered to the lower layer (layer_{ln}) with a Lookup event. This will result in a DHT lookup. The query successfully returned a hit by sending an Update message to the QMS_{or}. The QMS_{or} finally sends the overall query results to the requester by using a Main-Reply message. In Figure 11-a, the sub-query scheme is $\langle c_{ln} \cdot c_{an} \cdot n_{sn} \rangle$. Therefore, the query must find a match for its c_{an} conditions, before going to the lower layer. This can be done by dispatching the query within layer_{an} and visiting potential ANs, by using probability tables and Downward messages. The query finally finds a matched AN and then sends a Lookup message to its lower layer in order to process the rest of the query conditions, similar to the first example. Figure 11-c depicts similar processes for a sub-query with query-scheme $\langle c_{ln} \cdot n_{an} \cdot c_{sn} \rangle$. Due to the c_{sn} constraints, the discovery must be initially started from the upper layer (layer_{sn}). A Forward message is used to transmit the query to the top layer. Subsequently, one or multiple anycasting might be required to find a matched SN. The sub-query then transferred to the layer_{an} for the rest of discovery process in the lower layers.

HARD3 defines four different types of Update events, which are elaborated in Table 4. The Update event handler in a RP_{an} generally performs the following operations in response to the occurrence:

(a) Depending on the content and the type of the incoming Update message, RP_{an} updates the values of the relevant fields in its local probability table. As it is detailed in Algorithm 2, Updating is required only for Update_{sys} and Update_{vic} events. On a Update_{vic} event, the updating process is terminated in this step since the purpose of this event is just to update the resource knowledge of neighboring aggregate-nodes of a potential SoR (the event generator). On Update_{sys} and Update_{null} events, the receiver realizes that the discovery process for the given sub-query is completed, thus, the Update messages follows the

reverse path taken by the query message to the QMSor (i.e., the QMS which has initially registered the query) through the overlay network. The only difference between these two events is that Update_{sys} is required to update the probability table of the intermediate nodes but Update_{null} does not need to do this.

(b) On Update_{qms}, the receiver (which is a RP_{an}) investigates the current status of the query considering the discovery results as provided by the received Update message. Accordingly, three different possibilities would be considered: the query is fully resolved, the query is partially resolved and the query is not resolved. When the query (i.e., the sub query which has already registered in the QMS of the original requester) is fully resolved, a corresponding Update_{sys} event is created and sent back to the QMS_{or} while the probability tables and the QROUTs of the revisited nodes will be updated. In fact, due to the temporal knowledge of the Query-Routers in each node, the Update_{sys} message backtracks its way to the QMS_{or} by traversing the nodes in different layers.

(c) In other case, when a sub query is partially resolved, RP_{an} properly reshapes the sub query as a Downward message including information on the partial discovered resources in order to continue the search to find the rest of the requested resources within the current layer. Probability tables assist RP_{ans} to efficiently decide, choice of the next QMS destination. There are two search ending conditions in layer_{an}: the first one is reached if all the potential QMS providers in the current layer of the current cell are explored, thus, an Upward message including the partial results (if there is any) will be transferred to the address of the SQMS provider in the upper layer. A potential QMS provider is an aggregate-node, representing a group of leaf-nodes in a minicell, which the probability to find the desired resources for a sub-query among its subsidiary resources is more than a specific threshold value. We must note that, even among the potential QMS providers, only qualified QMS providers which fulfills the c_{an} conditions for a given query will be deeply searched in layer_n level; the second layer_{an} search ending condition is also reached if the TTL (i.e time to live in terms of number of hops or expiration time) of the sub-query is expired, therefore, an Update_{null} will be sent to the QMS_{or}.

(d) Upon receiving an Update_{sys} or an Update_{null} message by a QMS_{or} (i.e., the main query registry point), the local QREG of the QMS_{or} will be updated according to the discovered results of the sub query and if all the other sub queries of the main query are also resolved or completed, a final main-reply message including the results of all the sub-queries will be sent to the original requester (the issuer of the given Main-Query) (see Algorithm 3).

As we elaborated in Table 4, there are two types of Downward events: normal Downward and call back Downward. A Downward event is triggered when the query conditions in the upper layer are met. It is the major communication event which is happening in layer_{an}, and notifies the receiver that the required number of resources are not fully discovered yet. The query transmission in layer_{an} is performed by employing DPTs, and exploits the query status information from the content of the received query at each aggregate-node to manage the relying

Algorithm 2: DPT Updating

```

Input: sq=sub-query, up=updater, ut=update-type or type of sub-query
/* sq:sub-query, rt:resource-type, nr:neighbor-record,
  φ():quality fun(), sor:source-of-resource */
sorpf(sq, up, ut, sornew, sorold, pfold, flag)
  // a function to calculate the new pf value and making
  // decision for the new sor
  result.sor=-1 // indicates that the sor value should not change
  δ=random(0,1) // * 0 < δ ≤ 1, λ:latency */
  λ=sq.receiver-time - sq.sender-time // λ is the latency by ms
  // between the sq-sender or updater and the sq-receiver
  if ut==Updatesys then // or if sq.type==Updatesys
    if (sq is fully resolved) ∧ (∃ sornew) then
      result.pf=pfold-δ*(pfold) +  $\frac{\delta}{\lambda}$  // potential increase in pf
      value
      if φ(sornew) > φ(sorold) then
        result.sor=1 // a new sor, suggested by sq, can
        // replace the previous sor in the DPT
      else
        result.pf=δ*(pfold) // potential reduction in pf value
        if (sorold==sornew) ∧ (flag == 0) then
          result.sor=0 // the previous sor in the DPT must
          // be removed by setting the sor value to empty
    else if ut==Updatevic then
      result.pf=pfold-δ*(pfold) +  $\frac{\delta}{\lambda}$ 
    else if ut==Downwardcbk then
      result.pf=δ*(pfold)
    return result

  sornew=sq.preferred-vnode // getting the preferred-vnode/SoR of sq
  foreach rt : rtid ∈ sq.res-type-ids do
    if DPT.find(rt) then // rt already exists in the DPT
      RTrecord=DPT(rt).get(); flag=1
    else // rt does not exist in the DPT
      DPT.add(rt); RTrecord.sor.set-empty(); RTrecord(nb).pf=1.0; flag=0
    foreach nb ∈ List of AN Neighbors do
      if nb==up then
        sorold=RTrecord.sor
        pfold=RTrecord(nb).pf
        res=sorpf(sq, up, ut, sornew, sorold, pfold, flag)
        RTrecord(nb).pf=res.pf
        RTrecord(nb).probability= $\frac{RTrecord(nb).pf}{\sum_{nr} AllNeighbors RTrecord(nr).pf}$ 
        if res.sor==0 then
          RTrecord.sor.set-empty() // SoR sets to empty
        else if res.sor==1 then
          RTrecord.sor=sornew // SoR changes
        else
          RTrecord.sor=sorold // SoR does not change
        DPT(rt).set(RTrecord)
  return

```

process. The query status information contains the fields such as the query-scheme (i.e., the query conditions in each layer), inter resources communication constraints, number of requested resources, number of discovered resources, resource-IDs for the discovered resources, preferred-vnode, source address, destination address, etc. Whenever a Downward query is received/sent from/to an/a aggregate-node/super-node, the query routing information (such as main-query-id, sub-query-id, parent-sender and destination) in the QROUT must be updated. Furthermore, for a given query, the corresponding information in the QROUT of the revisiting nodes automatically will be removed before generating the events such as Update_{sys}, Update_{null}, Upward and Downward_{cbk}. In fact QROUT only traces the mainstream of the queries which contain the events such as Downward, Upward and Forward.

The search in layer_{an} is conducted over a tree graph where the tree's root is the first aggregate-node in the current layer which receives the Downward query from the other layers (i.e., upper or lower layers). The QMS in each aggregate-node makes a decision to relay the query to one of the neighboring aggregate-

nodes if it is required. This is performed depending on several parameters such as the query's resource-type-ids, the query's preferred-vnode and the probability to find the required resource-type in the path which is specified by a neighbor-node as the next QMS destination. The query's resource-type-ids denote the IDs for different resource-types due to query dimensions. Each query dimension specifies a desired value or range of values for a single resource attribute. The query-analyzer (i.e., a QMS component) in the local QMS of each Main-Query is responsible for recognizing the list of resource-type-ids for each created sub-query. On the other side, each QMS in the system constructs, maintains and updates one small probability table containing fields such as neighbor-id and probe-factor(pf) (indicating probability) for each demanded resource-type. Moreover, for each resource-type a preferred source of resource might be assigned or modified during the update procedure, which represents the current QMS's preference to direct the related queries to a SoR that provides quality, in terms of size (i.e., number of available resources in SoR), and distance (i.e., latency or number of hops between current QMS and SoR).

Figure 12 depicts a simple example of DPT in a system with 2 predefined attributes (CCR: Core Clock Rate, L1S: L1 Cache Size) over 4 different ranges of values (resource-types 1-4). We must note that the number of predefined resource-types in the system is an arbitrary design choice. Also the number and definition of resource-types must be uniform among all probability tables in the system. Defining a large number of resource-types in the system may increase the resolution of DPTs (i.e., accuracy of probability tables). However, it might have memory cost, resulting DPTs with larger sizes (HARD limits the number of resource-type definitions for each single attribute in the range of 2 to 4). In our example, a requester (RR) sends a Main-Query to its local QMS provider (vnode-11). The resources desired for the query include two different processing cores (core-A and core-B): 5 cores with CCR=1500 MHz and L1S=512 MB and 8 cores with CCR=2200 MHz and L1S=256 MB. The query is simple and does not introduce any conditions in layer_{an} and layer_{sn} (i.e., $\langle c1_{ln} \cdot n_{an} \cdot n_{sn} \rangle$ and $\langle c2_{ln} \cdot n_{an} \cdot n_{sn} \rangle$).

The Query-Analyzer in the vnode-11 splits the Main-Query into two sub-queries, bearing in mind that each sub-query must represent query conditions for required resources, specified by the query description, which are identical to each other (i.e., resources required by a sub-query are a group of homogeneous resources). In other words, sub-query1 aims to find 5 processing cores of type core-A and sub-query2 is going to find 8 processing cores of type core-B. These two sub-queries finally return their results achieved to the QMS_{OR} in vnode-11. The QMS_{OR} aggregates the results and sends a Main-Reply to the requester (vnode-7). Figure 12 shows that sub-query1, in continuation of its exploration path, arrives in vnode-21. Due to the querying policies for the current layer, if the sub-query is required to be transferred into another QMS provider in the neighborhood, therefore one of the neighboring vnodes must be selected as the next QMS destination. This can be done by using the probability table in vnode-21. This table includes the list of direct neighbors (vnodes-72, 12 and 23) as well as the probe factor for each neighbor for each resource-type-id. The sub-query1 contains its list of desired resource-type-ids (1 and 4) which have been already specified by the Query-Analyzer of the QMS_{OR} of the sub-query. For the sub-query1, the QMS in vnode-21 only analyzes the resource-type records RT-1 and RT-2 which are matched with the resource-type-ids of the sub-query. Accordingly, a neighboring vnode with the absolute maximum of probability in both records (vnode-12) will be selected as the sub-query destination. Each resource-type record also includes a SoR suggestion. The sub-query1 sets its preferred-vnode to 41, since its original preferred-vnode is empty and also both records RT-1 and RT-4 collectively propose an absolute value for SoR. We must note that values for PFs and SoRs in the probability tables are dynamically updated due to the querying data provided by each transacted query (the initial values for PFs are 1, as they are for RT-3, and the SoR value might be empty, as it is for RT-3 and RT-4).

The type of sub-query1 in the above example (Figure 12) is Downward. In fact, upon receiving a Downward query by a QMS, if the query conditions in layer_{an} are met by the aggregate-stamp,

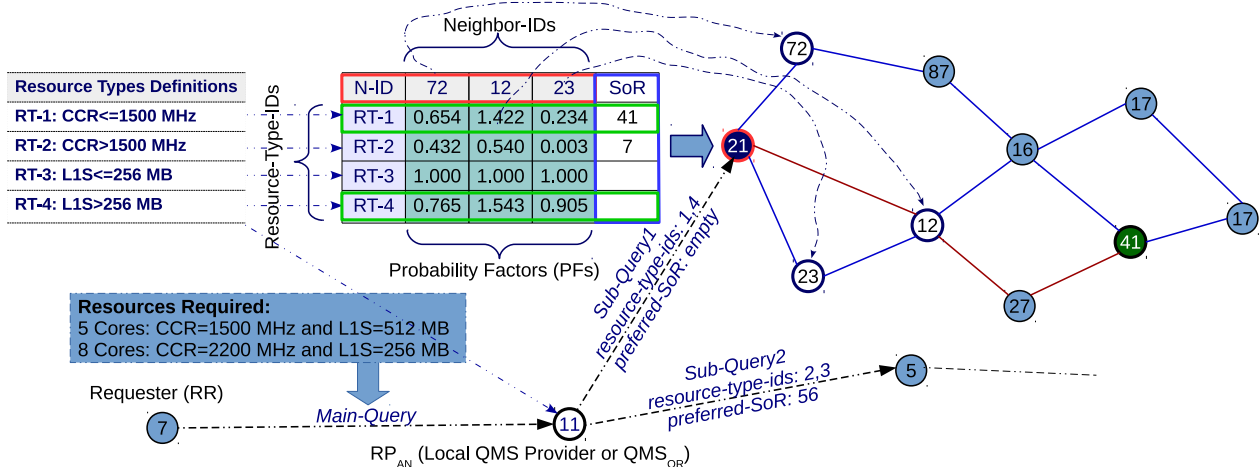


Figure 12: An example of DPT in a system with 2 predefined attributes over 4 different ranges of values (resource-types or resource-categories) (CCR: Core Clock Rate, L1S: L1 Cache Size).

Algorithm 3: Processing an Update_{qms} sub-query by a RP_{an}

```

Input: sq= The received Updateqms message
/* sq, temp:sub-query, qrg:QREG, qrt:QROUT, QMS: the QMS
   provided by RPan */
while sq.TTL is valid do // if sq is valid due to its Time to Live
  if sq is fully resolved then // if all resources required by sq has
  already been discovered
    if sq is registered in QMS.qrg then // if QMS is the QMSor of sq
    qrg.addDiscoveryResults(sq) // adding results of sq to
    overall results collected by other sub-queries of
    the main-query
    if qrg(sq.mainQID).isCompleted then // checking if all
    sub-queries of the main query have already been
    resolved
      send(Main-Reply.qrg(sq.mainQID).sender)
      /* sending the final Main-Reply of the query
      to the original requester */
    else // if QMS is not the QMSor of sq
      send(Updatesys, qrt(sq.subQID).sender); // sending an
      Updatesys to the sender of sq (the last visited RPan
      by sq) through using sub-query tracking
      information recorded in QROUT
      if the current vnode is the preferred-vnode for sq then
      | broadcast(Updatevic, all members of AN-Neighbors)
      // sending an Updatevic message to all neighbors of
      current RPan
      qrt(sq.subQID).remove // removing sq tracking info
      from QROUT
    else // if sq is not resolved or partially resolved
      temp=sq; temp.nRR= sq.nRR- sq.nDR; temp.nDR=0
      // adjusting/reshaping sq (as sub-query temp) based on
      current discovered resources and the remaining
      resources required, the sub-query temp maintains list
      of all previously discovered resources by sq and it is
      identical to sq except for the changes
      temp.type=Downward|Downwardcbk |Upward // QMS makes
      decision for the type of sub-query temp, the initial
      choice is Downward, if it is not possible, then
      Downwardcbk is the choice, and in the case that all ANs
      in the current cell have already been searched, Upward
      is the choice
      qrt(sq.subQID).update
      Continuation of the search with the modified sub-query temp // since
      sq is not fully resolved yet, the search must be
      proceed either in the current layer (layeran using
      Downward or Downwardcbk) or in the upper layer (layersn
      using Upward)
  if sq.TTL is expired then
    if sq is registered in QMS.qrg then
      send(Main-Reply.qrg(sq.mainQID).sender)
      qrg.dreg(sq.mainQID) // removing main query information
      (including sub-queries result info) from QREG
      removeQroutInfo(sq.mainQID) // removing main query
      information (including sub-queries routing info) from
      QROUT
    else
      The sq will be destroyed;
      The sq routing info in the QROUT of all visited nodes (by sq) will be
      removed;
      The QMSor will be notified to issue the final Main-Reply for the query;
  return

```

a Lookup query is conducted to search in the current mini-cell, otherwise the QMS, using its probability table, must select the next Downward destination to continue the search.

In order to select the next QMS provider among the neighboring aggregate-nodes, as it is presented in Algorithm 4, at first, the QMS checks if there exist any absolute SoR preference either by query itself or by QMS (i.e., query based or QMS-based preference). An absolute preference only exists if an absolute maximum number of records (in the probability table) corresponding to the resource-types mentioned in the resource-type-ids of the given query agree to vote to a specific SoR.

HARD3 introduces two different mechanisms for SoR trac-

Algorithm 4: Selection of the next QMS provider

```

Input: sq= sub-query
/* sq:sub-query, rt:resource-type, nb:neighbor, vn:vnode,
   sor:source-of-resource, op:overall-probability */
const: call-back /* indicates that a Downwardcbk must be initiated,
   an Upward is initiated when it is not possible to initiate a
   Downwardcbk */
θ:vector, temp:vector-record // overall probability for the neighbors
A={nb : nb ∈ AN - Neighbors}; B={vn : vn ∈ sq.visited - qms - ids}
C=(A ∩ B') // A is the list of AN neighbors, B is the list of
all already visited ANs by sq and C is the list of neighbors
which have not yet been visited by sq
if |C|=0 then return call-back
if (there is sq-preferred-vnode) ∧ (sq-preferred-vnode ∈ C) then
  return sq-preferred-vnode
foreach rt : rtid ∈ sq.res-type-ids do // calculating the average
probability for each of the C members with respect to
different resource-type-id specified by sq.res-type-ids
  if DPT.find(rt) then
    RTrecord=DPT(rt).get()
    foreach nb ∈ C do
      if θ.find(nb) then
        θ(nb).op=mean(θ(nb).op, RTrecord(nb).probability)
      else
        temp.neighbor=nb
        temp.op=RTrecord(nb).probability
        θ.add(temp)
  if θ.size()==0 then return random(nb : nb ∈ C)
  D={nb : (θ.find(nb) ∧ (θ(nb).op == θ.Maximum - op) ∧ (θ(nb).op > 0))} // D
  is the list of C members with the maximum overall probabilities
  if |D|=0 then return random(nb : nb ∈ C)
  // returning a random member of C as the next QMS destination
  else return random(nb : nb ∈ D)
  // returning a random member of D as the next QMS destination

```

ing: query-based preference and QMS-based preference. In the query-based preference, the query obtains the value for its preferred-vnode (note that each query message includes the information of the preferred-vnode of the query) based on the QMS's SoR preference of the QMS_{or} and it keeps relying on this fixed preference until discovery ends. But in the QMS-based preference, the query in each intermediate QMS dynamically applies to use the SoR preference of the current QMS. On the existence of either the preferred-vnode for the query (in query-based method) or the absolute SoR preference for the current QMS (in QMS-based method), RP_{an} gives priority to a neighboring QMS provider, which is preferred by SoR preference or preferred-vnode as the next query destination. However, if the query doesn't have a target preference, a neighbor, with the highest probability among all neighbors is selected. In the case that the number of neighbors with highest probability is higher than one, a random destination is selected among the highest ranked neighbors. It must also be taken into account that, the overall probability to select a neighbor as destination for a query in a QMS, is measured by averaging the probability values provided by the local probability table, corresponding to the resource-types, identified in the list of resource-type-ids of the query.

Whenever a RP_{an} ensures that all the potential QMS providers in the vicinity of the current node are explored, and since the query is not completed, a Downward_{cbk} message must be sent to the sender of the original query in the upper level of the search tree (i.e., the query parent), using the query tracking information recorded in the QROUT. The query itself also keeps the information about the visited nodes, which helps the query to efficiently explore the tree graph. By triggering the Downward_{cbk}

event in the receiver-side, the normal Downward procedure is performed, exploring other potential branches of the tree, by directing the query to a qualified unvisited neighbor, while the probability table in the Downward_{cbk} receiver is updated due to the query's characteristics and search results, as we elaborated in Algorithm 2. If consecutive down-warding processes (i.e., series of Downward_{cbk} and Downward) fail to complete the given query in different levels, and branches of the search tree and the Downward_{cbk} finally reach the entry QMS provider (i.e., the first QMS provider in the current layer of the current cell which initiated the Downward query within this layer), a RP_{an} sends an Upward message to its SQMS address in the upper layer (i.e., layer_{sn}), informing the SQMS provider that search must be continued in other qualified cells in the distributed environment.

5.3. SN Algorithm Description

In this section we present our specification-based anycasting method to resolve the queries in layer_{sn} . Anycast can be considered as a powerful paradigm for resource discovery in large scale distributed systems. It enables communication between a source node and the nearest (or the best) member of an anycast group. The proximity metric (or the metric for being the best) can be defined in terms of hop count, delay or the minimum amount of load [32].

In our algorithm, each SN creates its own anycast address by hashing the layer-stamp of the SN node which is the representative of all the specifications of resources in layer_{sn} . Using this approach, the anycast address of each SN is a function of the resource specifications in the super-node's layer. The anycast address of a node may change when the node's specifications (e.g. dynamic resource attributes) are modified. SN nodes with uniform specifications advertise the same anycast addresses. The creation and maintenance of the anycast groups are significantly lightweight, since the anycast groups can automatically be created and they can also dynamically be changed without creating any necessity for communication among the group members or group registration. SN nodes advertise their anycast address as well as unicast address to other SN nodes in their neighborhood. The neighborhood can be specified based on proximity metrics such as number of hops or delay. Whenever a RP_{sn} is decided to anycast a discovery request to other potential SNs in the network, it passes the request to its local anycast-resolver which is responsible to determine the unicast address of the best possible destination. Subsequently, the discovery request would be forwarded to the destination by using its unicast address and the regular routing. The anycast-resolver operates as followings: The anycast address of the potential destination for a given discovery request, is extracted from c_{sn} mentioned in the request. Anycast-resolver checks the list of registered SNs in the current node and selects the ones that their anycast address are similar to the anycast address of the given request. In the next step, the unicast address of a SN in the list with minimum number of hops (i.e., minimum delay) is returned to RP_{sn} as the final destination of the request. If the desired anycast destination is not found among the registered SNs, the discovery request is forwarded to the closest SN in the list. If the list is empty, the query is terminated and the proper update (i.e., reply) message

would be sent to the original requester. If c_{sn} conditions are not existed for the given discovery request (i.e., when the creation of the anycast address is not feasible), the discovery request is forwarded to the closest SN in the list.

The RP_{sn} s are the resource providers which provide SQMS services to entities in their local cell, and to other SQMS providers in the system. Depending on the type and status of the event triggered and the characteristics of the given query, SQMS providers might behave as RP_{sn} , RP_{an} or RP_{ln} s. Whenever a RP_{sn} receives a query, a query type retrieval is performed, and different mechanisms and procedures are conducted to resolve or redirect the query. The Forward event is the most regular communication event within layer_{sn} . A Forward receiver (a node which receives a Forward message), first assesses whether the layer stamp (i.e., the super-node-stamp) will be qualified due to the c_{sn} query conditions or not. If it is qualified, a Downward self-event is triggered in the current super-node where the SQMS provider, as the event-receiver acts as a QMS provider, which conducts the Downward query to lower layers. The SQMS provider, later, will receive a response from the lower layers either in the form of Upward (if the query fails or remains uncompleted in the lower layers) or Update messages (if the query is resolved or completed). On the occurrence of a Update event, the RP_{sn} s redirect the incoming messages to the original requesters. Receiving a Upward event is exclusively dedicated to RP_{sn} s (see Table 4 and Figure 9). In fact, on occurrence of an Upward event, the RP_{sn} will know that the requirements of the correspondent query can not be met in the lower layers of the current SN's cell, and the query must be directed to other remote cells in the system.

Algorithm 5: Anycast based Forwarding in RP_{sn}

```

Input: sq= The received sub-query message
/* sq:sub-query, nb:neighbor */
if sq.type==Forward then
    if  $\text{RP}_{sn}$  is qualified according to sq.csn then
        send(sq, Downward,  $\text{RP}_{sn}$ )
        /* generating a Downward self-event by sending a
        self-message to  $\text{RP}_{sn}$ , as a result,  $\text{RP}_{sn}$  behaves as a
         $\text{RP}_{an}$ , receiving a Downward message */
    else
        anycast-address=mapped-ac-address(sq.csn) // generating an
        anycast address by mapping the specification of the
        sub-query sq in the  $\text{layer}_{sn}$  into an anycast address
        anycast(sq.Forward, anycast-address) // anycasting the
        sub-query sq with the type Forward to the extracted
        anycast address
    else if sq.type==Upward then
        if there is sq.csn then
            anycast-address=mapped-ac-address( $\text{RP}_{sn}$ .layer-stampsn)
            // generating an anycast address by mapping the
            specification of the  $\text{RP}_{sn}$  in the  $\text{layer}_{sn}$  into an
            anycast address
            anycast(sq.Forward, anycast-address)
        else
            target=random(nb:(nb ∈ SN-Neighbors)^(nb ∉ sq.visited-qms-ids))
            // selecting a random neighbor from the list of SN
            neighbors which have not yet been visited by sq
            send(Forward, target)
    else if sq.type==Update then
        |  $\text{RP}_{sn}$  performs the corresponding update procedure similar to  $\text{RP}_{an}$ 

```

As depicted in Algorithm 5, if a RP_{sn} receives an Upward query, it means that the receiver already has been qualified for the c_{sn} query conditions, but the query conditions in the lower layers have not been achieved. Accordingly, the RP_{sn} sends a Forward message to an anycast address extracted from c_{sn}

of the given query. The Forward message will be automatically redirected to the nearest SQMS provider in the system, which has the same anycast address. Using the proposed anycast scheme significantly reduces the search space for the given query. It automatically limits the search space to only the SQMS providers in the system that certainly would be able to fulfill the c_{sn} query conditions. However the query conditions in the lower layers must be examined in each target forward-receiver separately.

6. Evaluation and Simulation Results

This section presents the simulation results for evaluating the proposed resource discovery model. For this, we use simulation instead of experiment on the real large scale computing infrastructures (e.g. PlanetLab, TACC, Oak Ridge, BSC, GENCI and public cloud providers like Amazon and Google) due to issues as cost and flexibility of the simulation to design, develop and assess several algorithms as well as providing full control over system behavior and evaluation scenarios. On the other hand, real infrastructures generally provide limitations to explore the design space, particularly for scalable performance and large scale evaluation. Due to these reasons, and also due to space limitations, we consider the evaluation of our discovery approach on the real infrastructures as future work.

6.1. Simulation Approach

To do our evaluation, we developed a simulation platform, based on OMNET++/INET-Framework and Oversim simulation tools (similar to the approach that is presented in [33]), which is able to simulate many-core environments (up to 55000 processing cores in different chips and network-nodes), focusing on communication aspects (i.e., communications between cores, chips and nodes), albeit taking long simulation times (a 3-week simulation run is average).

Objective Modular Network Testbed in C++ (OMNeT++) [34] is a discrete event simulation tool designed to simulate computer networks, multi-processors and other distributed systems associated with a GUI-based simulation library debugging and tracing. A simulation model consists of “nodes” connected by “links”. The nodes represent blocks, entities, modules, etc, while the links represent channels, connections, etc. The structure of how fixed elements (i.e. nodes) in a network are interconnected together is called the topology. OMNeT++ uses the Network Description (NED) language for topology description. The simulator provides a set of built-in components and libraries to instantiate as nodes and as links. We have extended and developed a set of C++ based components to describe details of a manycore system including network and interconnection topologies. This has been done by developing compound modules (derived from the built-in “cComponent” module) to describe different entities and objects such as multicore nodes, core, v-node, QMS and SQMS objects. Each compound module, in turn, includes a set of other compound or simple modules (derived from “cModule” or “cSimpleModule”) as well as the description of interconnection among modules (derived from “cObject”,

“cGate” or “cChannel”). The network topology among simulated network nodes can be also defined by using “cChannel” and through the NED description of the network topology.

Overall, the simulated manycore system includes the simulated network nodes (up to 1000) connected through a network (with bandwidth of 100Mbps) based on a random network topology (a connected graph with $\beta = 62.5$, indicating the ratio of the number of nodes to the number of links in the network graph), where each network node contains manycore processors. The cores of each processor are connected through a three-dimensional mesh/torus interconnect topology (with data rate of 50Gbps) where each core has its own dedicated L1 and L2 cache that are not shared with any other cores. We also do not simulate cache coherency protocols, as we use message-passing for inter-core and inter-chip communication. The routing among cores is performed through a variation of Dimension Order Routing algorithm where packets are routed to the correct position along higher dimensions before being routed along lower dimensions. Furthermore, the processors of each node are connected through a high speed bus with data rate of 20 Gbps.

We conducted and implemented our experiments in different scenarios and settings. Accordingly, we evaluate the scalability and efficiency of HARD3 with respect to the several evaluation criterias (such as discovery latency, discovery load, discovery accuracy and discovery cost) in large scale resource pools (containing 1000 to 55000 simulated processing cores positioned in 100 to 1000 simulated network node) with presence of high dynamicity and high heterogeneity of resources and show the performance achieved with the proposed methods and heuristics.

In order to run our simulation model, we use the inherent capability of OMNET++ to execute multiple simulation runs in parallel [34] and on different processing cores of a single dedicate multicore server (an octa-core with 96 GB of memory). This is different from the distrusted parallel simulation of model partitions across multiple machines/processors in a cluster which might be challenging, as we discussed in our previous work [35].

In the rest of this section, we evaluate scalability (for both synchronous and asynchronous querying) and efficiency (for complex querying in high heterogeneous computing environments) and finally we compare our discovery approach with other different proposals.

6.2. Scalability for Synchronous Querying

Upon starting an iteration in a synchronous querying approach, requesters simultaneously start to propagate their discovery requests (i.e., a single main-query per requester per iteration) in the system. In contrast to asynchronous querying, the reserved resources (by requesters) in synchronous querying would be released after timing synchronization and before starting each successive iteration process. However, the distributed probability information, gathered in the whole system during an iteration will be maintained to be used and updated in the successive iterations. The purpose of our evaluation under these assumptions is to fully understand and precisely study the HARD’s scalability behavior with minimal impact caused by resource reservation.

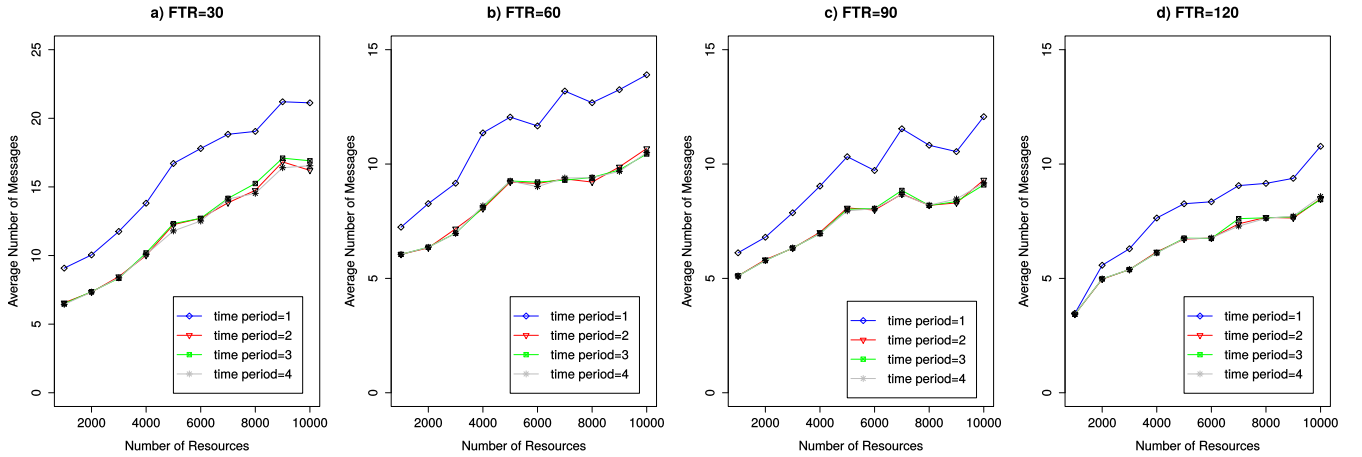


Figure 13: Average number of discovery messages vs system size for basic querying (i.e., single-resource/single-sub-query) in several querying iterations (time periods) with different FTR values. FTR is the overall frequency of the target (desired) resources.

6.2.1. Simulation Setup

We conduct our simulation experiments along two different main scenarios (i.e., synchronous and asynchronous querying scenarios), and for each scenario we conduct several experiments with regard to different simulation parameters.

For evaluating the synchronous querying we simulate a many-core networking environment containing between 10 to 100 types of heterogeneous computing resources which are uniformly distributed within a system, with the size ranging from 1000 to 10000 resources, and where each simulated node represents a computing resource. The simulation starts by performing a self-stabilization of the nodes during the initialization phase, which leads to the establishment of a distributed hierarchy of virtual overlays on top of the simulated network. Upon completion of the initialization phase, nodes in the system would be qualified to start discovery requests.

We schedule 4 synchronous querying iterations (time periods) for all the requesters in the system. A constant fraction of nodes (1%) are randomly selected to initiate discovery requests by specifying their resource requirements (in terms of number of target resources to be discovered, arbitrary level of details of computing characterization factors and communication properties among requested resources) as desired target attributes (in different resource description level). The selected requesters, after a synchronization step, regenerate the similar queries in the next 3 iterations. Moreover, the scheduled queries in all iterations for all the requesters are also uniform with the type of single-resource-single-query. The frequency of the target resources are ranged from 30 to 120. The other simulation parameters for the first scenario are summarized in Table 5.

6.2.2. Simulation Results

In the first experiment we evaluate the impact of changing the system size (in terms of the total number of resources in the system) on the discovery cost, in terms of the number of required messages to perform a discovery request. As it is shown in Figure 13, the discovery cost in all the tests is decreased in the subsequent iterations. In other words, the average number

Table 5: Simulation parameters for scalability evaluation (synchronous querying)

Parameter	Values
Maximum start-up time	3000 ms
Querying iterations per requester	4
Querying interval	2000 ms
Complexity of querying	single-resource/single-query
Percentage of requesters	1%
Frequency of Target Resources (FTR)	30,60,90,120
Physical network size	1000 to 10000 cores
Interconnect topology	mesh/torus
Network topology	random
Interconnect channel data-rate	50Gbps
Routing type	DOR
Network channel	100 Mbps
Simulation runs	10 per system size per scenario

of required messages per discovery request decreases as time progressed which means that the proposed discovery approach is scalable over time. The reason for this is that the query guidance mechanism in the DPTs will be dynamically improved, by enhancing the probability values for successful discovery over larger number of query dimensions, as well as raising the quality of SoR preferences in the QMS providers in the system with respect to the results of both the past queries in the former iterations and the concurrent queries from other requesters in the current iteration.

The discovery cost reduction is specially significant when time progressed from the first iteration to the second iteration, while for the subsequent iterations, the speed of discovery cost reduction is decreased gradually with a lapse of time. This happens because of the lack of querying knowledge in the DPTs for the first iteration. In fact, DPTs should be tuned and warmed up (i.e., build up) through initial queries, before they can efficiently be used in the system.

As we can also see in the similar results of a experiment, for measuring discovery latency in various system sizes (see Figure 14), the initial DPTs warm-up leads to higher latency for the initial queries which are generated in the first iteration. However, depending on several system parameters, such as discovery traffic (in terms of the number of discovery requests, number of

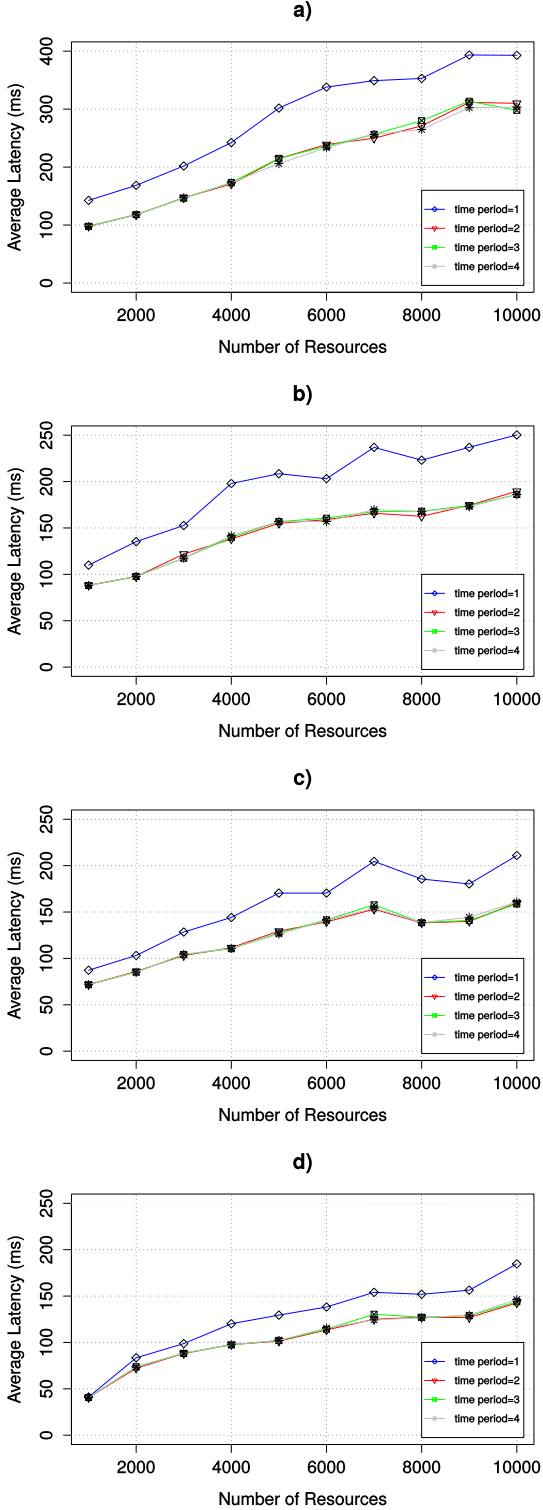


Figure 14: Average discovery latency vs system size for basic querying (i.e., single-resource/single-sub-query) in several querying iterations (time periods) with different FTR values. FTR is the overall frequency of the target (desired) resources. a) FTR=30, b) FTR=60, c) FTR=90, d) FTR=120.

requester, querying intervals, etc.), complexity of the queries, heterogeneity of the resources and DPT's configuration (in terms of predefined resource-types, SoR capabilities, etc), DPTs can quickly be adjusted and updated, in order to predict and recog-

nize the appropriated high quality SoRs in the system for each given discovery request, with higher level of accuracy, and even the DPT warm-up cost might not be visible after several number of querying iterations.

The results in Figure 13 also show that the average number of messages required per discovery request, for different system sizes, increases due to the increased exploring space of the larger systems. However, in all tests, for larger iteration numbers, and specially for the larger systems, the slope (in most places) is very slight or steady, which demonstrates that the proposed discovery approach can tolerate an increase of the system size (in terms of number of resources), while it maintains system efficiency by exploring the larger search space for the discovery requests, with almost constant number of messages. Moreover, the average number of required discovery messages decreases when the Frequency of the Target/Desired Resources (FTR) is getting higher. Considering the similar behavior for the latency tests in Figure 13, it can be seen that HARD3 provides good scalability for the discovery requests in large and very active systems.

The results presented in Figs 13 and 14 also prove that our proposed DPT mechanism is fault tolerant. As depicted in these figures, a large change (improvement) in the discovery performance in the second iteration has happened compared to the discovery performance in the first iteration. This improvements continues in the next iterations. But, as it shown in the results, after a few initial iterations, changes become imperceptible. Indeed, DPTs are empty at the beginning, but after a few iterations, they can rapidly become informative and provide a reasonable stable performance. In other words, altering probability tables in some nodes does not have a visible impact on the overall system behavior. This makes probability tables more powerful for being rapidly recovered from any potential failure.

Figs 15-a and 15-b demonstrate the overall results of all aforementioned iterations including the DPTs warm-up costs to measure the impact of system size on the discovery overhead and discovery latency accordingly. As we can see in these results, the mean discovery overhead and the mean discovery latency, particularly for the reasonable frequency of the target resources (like FTR >60), and for the larger system sizes, remains stable with no significant changes.

The overall results in this section proves the HARD3 scalability at least for the simple querying. In the next section we evaluate the HARD3 scalability under complex querying conditions whereas high resource heterogeneity is emphasized.

6.3. Heterogeneity and Complexity

HARD3 provides flexibility for complex querying in terms of multidimensional querying, resource graph querying, exact/partial and range querying. Multidimensional and resource graph querying are inherent features of HARD3 due to the original attribute-based definition of resources in HARD's hierarchical architecture and mechanisms as well as the query description. In other hand, in addition to exact querying, the partial and range querying are possible by leveraging a similarity function in different layers, where HARD3 algorithms would be able to discover the required resources with a certain amount of approximation.

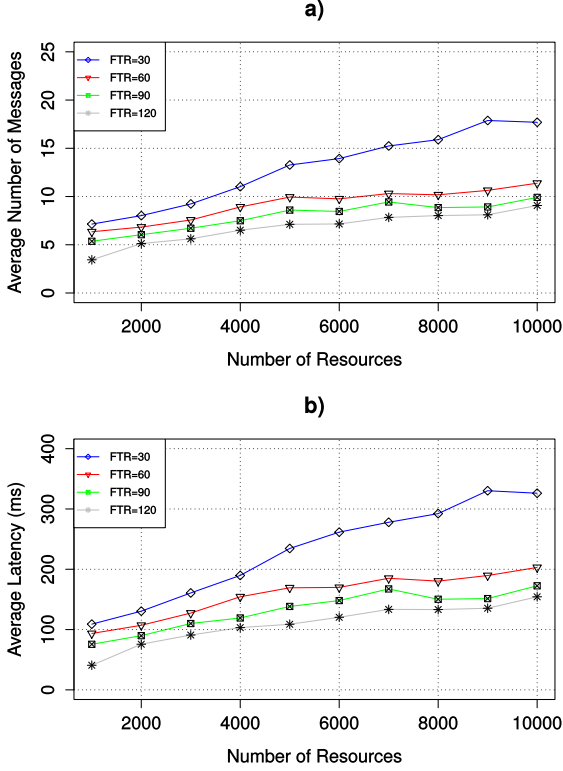


Figure 15: a) Discovery Overhead (in terms of number of required discovery messages) vs system size for different frequencies of the target resources in Synchronous Querying. b) Discovery latency vs system size for different frequencies of the target resources in Synchronous Querying

Indeed, requesters are allowed to send main-queries to their corresponded QMS providers, containing multiple querying conditions in each layer, as well as inter-resource and inter-resource-group communication constraints for the desired resources. A main-query might include the required conditions for several different (i.e heterogeneous) resource groups where each resource group specifies the number of required resources with similar (i.e., homogeneous) characteristics and inter-resource communication properties. QMS providers, in turn, split the main-queries into various number of sub-queries depending on the number of heterogeneous resource groups described in the main-queries. The obtained concurrent sub-queries dynamically and independently choose their own-path across the system in order to find their required number of homogeneous resources and finally returns the discovery results to their origin QMS provider, which is responsible to maintain the overall-state of the main-query and make proper decisions accordingly. In this section, we evaluate the impact of complex querying on the HARD’s scalability by increasing the number of desired heterogeneous resource-groups, as well as the number of required resources per each group in the main-queries of the requesters. We provide evaluations for both uniform and non-uniform distribution of SoRs capabilities (i.e., initial number of potential available resources in each source of resource). In uniform distribution, SoRs in the system initially have equal capability to provide resources to the requesters, while in non-uniform distribution the initial capabilities for each SoR might be different.

6.3.1. Simulation Setup

In order to evaluate the HARD3 performance with respect to high complex querying and high resource heterogeneity, we conduct a simulation scenario based on the previous scenario, but with some added changes. Simulation parameters presented in Table 6.

Table 6: Simulation parameters for HARD3 evaluation under the complex querying conditions and high heterogeneity of resources (synchronous querying)

Parameter	Values
Complexity of querying	multi-resource/multi-query
Frequency of Target Resources (FTR)	300
Physical network size - uniform SoRs	3000,5000,7000 cores
Physical network size - non-uniform SoRs	16500,27500,38500 cores
Homogeneity rate of desired resources	20%, 25%, 33%, 50%, 100%
Uniform SoRs capabilities	10 resources per SoR
Non-uniform SoRs capabilities	normal($\mu = 50, \sigma = 40$)
Desired homogeneous resources/subquery	1 to 6, uniform-SoRs
Desired homogeneous resources/subquery	3 to 18, non-uniform-SoRs
Static subquery dimensions	min=4, max=12
Dynamic subquery dimensions	min=4, max=8
Simulation runs	10 per system size per scenario
Number of resource-type definitions	3 for each single attribute

6.3.2. Simulation Results

Figs 16-a1 and 16-a2 depict the experiment results for average discovery latency, and average required number of discovery messages per query, for complex querying in the system with 3000 resources while the SoR capability has uniformly been applied for all the vnodes in the system. This means that all SoRs in the system initially have similar capability to supply resources. The SoRs capabilities might also be changed over time depending on their resource release or occupation conditions. In these figures we can see that, as the level of querying complexities in terms of number of required resources and the heterogeneity of the desired resources are increased, the mean discovery latency and discovery cost remain scalable. The experiment results presented in Figs 16-bx and 16-cx also show the similar behavior for the larger networks with 5000 and 7000 number of resources. However, for the highest complex queries (like the main-queries with the homogeneity rate less than 50% and the number of required resources per sub-query greater than 4), we see that our resource discovery approach gradually becomes worse when we vary the system size from 3000 in Figs 16-ax to 5000 in Figs 16-bx and 7000 in Figs 16-cx. This happens because of several reasons such as, poor stability of SoRs under sub-query requests with high resource demands, disproportion between weak SoRs capabilities and high rated concurrent queries with large resource demands, inefficiency of the HARD’s SoR preference mechanism in the lack of SoRs with initially non-uniform capabilities which significantly reduces the degree and scope of competitiveness for SoR preference, and finally exceeding resource demands over resource availability.

QMS providers are able to continuously and dynamically detect and recognize the best possible SoRs for each of their registered resource-type-ids in each moment of time. But these SoRs might become weaker (in terms of number of available resources) over time through reservation of their resources by

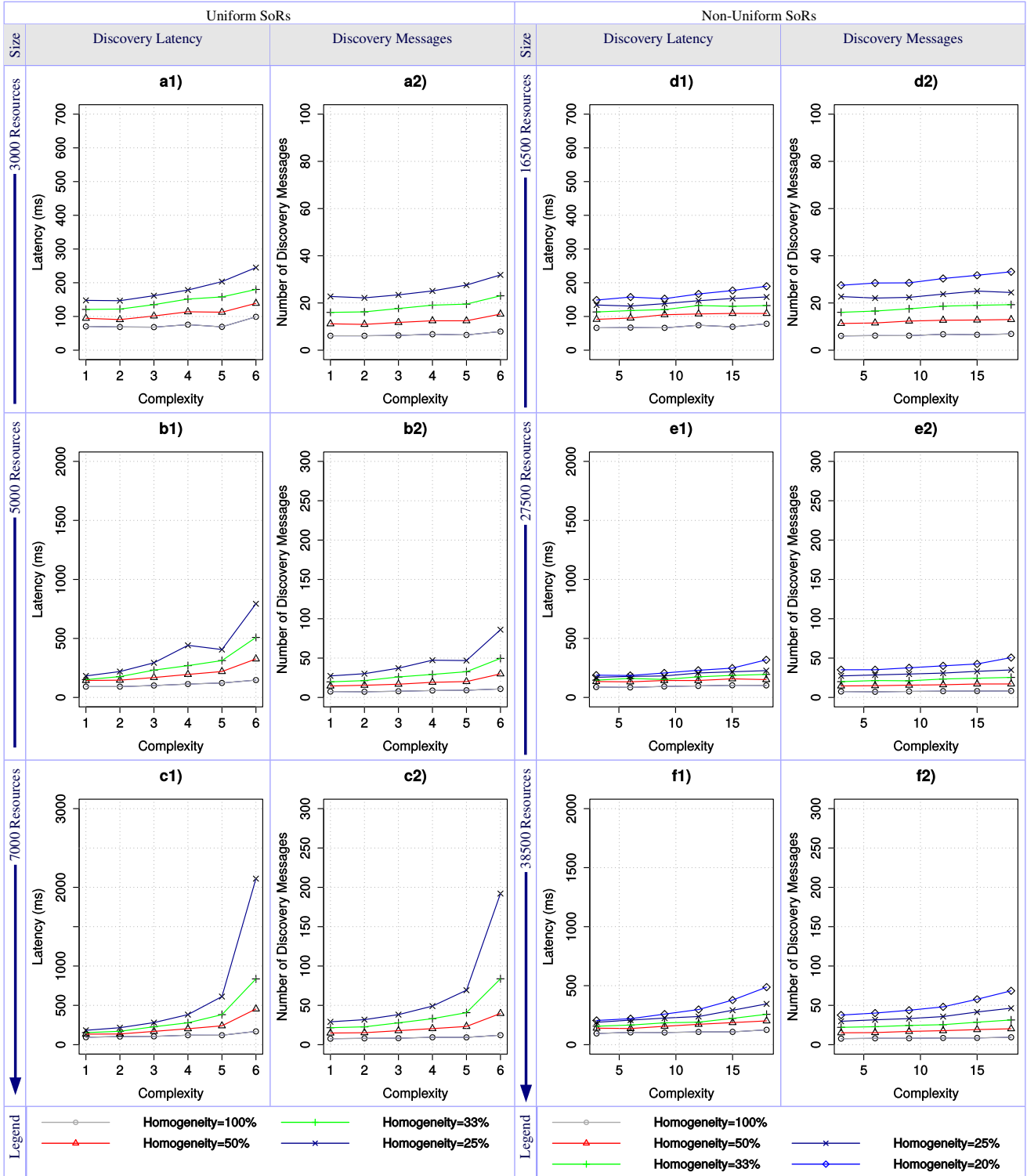


Figure 16: Complexity (in terms of heterogeneity of desired resources and number of desired homogeneous resources per sub-query) vs discovery latency and number of discovery messages for different system size and different distribution of SoRs capabilities: a1 and a2) uniform SoRs with system size=3000, b1 and b2) uniform SoRs with system size=5000, c1 and c2) uniform SoRs with system size=7000, d1 and d2) non-uniform SoRs with system size=16500, e1 and e2) non-uniform SoRs with system size=27500, f1 and f2) non-uniform SoRs with system size=38500.

multiple queries from different requesters. The weak SoRs eventually become unresponsive to the incoming queries, and this will create an extra communication cost to detect the best alternative new SoR as the replacement of the dead SoR. In the experiment results presented in Figs 16-ax, 16-bx and 16-cx, the

initial number of available resources supported by each uniform SoR in the system is equal to 10. This means that a fresh SoR dies after successful handling of 10 successive sub-queries with one desired resource, given our assumption for synchronous querying that the discovered resources in each querying iteration

would be reserved until end of the iteration. As we increase the number of desired resources for each sub-query, the instability rate of SoRs in the system is increased. For instance, for sub-queries with 5 desired homogeneous resources, the target SoRs at best die after only 2 (and even less for higher demands) successful queries handled, which leads to high rate of SoRs instability in the system (see Figs 16-cx). In these experiments, it can be concluded that our resource discovery approach, for complex querying in synchronous manner with uniform SoRs, can remain scalable, while the amount of demands is at least less than half of the target SoRs capabilities.

In the aforementioned experiments we assumed that the SoRs capabilities are uniformly distributed, and all the SoRs initially provide equal number of available resources. However in a real jungle computing environment, the SoRs capabilities are not uniform, and in such systems there exist multiple resources with different capabilities and strengths. Along this line we extend our evaluations for non-uniform distribution of SoRs capabilities. The initial number of available resources for each SoR is obtained using a normal distribution with the given mean of 50 and standard deviation of 40 truncated in the range [10,100]. We also increase the level of complexity for the main-queries as well as the system size (see Table 6). As we can see in the Figs 16-dx, 16-ex and 16-fx, HARD3 provides significant scalability for discovery latency and discovery cost (i.e., number of transacted discovery messages) when we increase the level of complexity (with respect to the heterogeneity and amount of desired resources for each main-query) for even larger system sizes (16500, 27500 and 38500) and more complex discovery requests (3 to 18 for number of desired resources and 100% to 20% homogeneity rate).

6.4. Scalability for Asynchronous Querying

For asynchronous querying in dynamic computing environment, the querying interval for each requester in each iteration is randomly specified by a uniform distribution in the range [2000,6000] ms. Requesters also propagate their successive complex main-queries in the system upon reaching each querying interval. Subsequently, on the successful completion of the resource discovery, the discovered resources would be reserved for the requester in the way that the other concurrent requesters in the system will not be able to discover and get access to the reserved resources until those resources are released by the original requester (i.e., resource occupier). The requesters will release their reserved resources when the execution of the corresponding application is ended. In fact in a dynamic computing environment, resource reservation leads to unexpected unavailability of resources, which can be described as the natural churn. In this section we evaluate the HARD's efficiency and scalability under complex asynchronous querying in dynamic computing environments (i.e., evaluation under natural churn).

6.4.1. Simulation Setup

In order to evaluate the HARD3 performance with respect to complex asynchronous querying in dynamic computing environments, we conduct a simulation scenario based on modifications of the previous scenario, as presented in Table 7.

Table 7: Simulation parameters for asynchronous querying

Parameter	Values
Complexity of querying	multi-resource/multi-query
Frequency of Target Resources (FTR)	1650
Physical network size - non-uniform SoRs	27500 cores
Execution time (i.e., reservation) uniform	[i-2000,i=2000*k],k=1-7
Querying interval by ms	uniform[2000,6000]
Consecutive main-query runs per requester	100
Homogeneity rate of desired resources	33%
Non-uniform SoRs capabilities	normal($\mu = 50, \sigma = 40$)
Simulation runs	1 per system size per scenario
Desired homogeneous resources/subquery	20, non-uniform-SORs

6.4.2. Simulation Results

Figs 17-a1 , 17-a2 , 17-a3 and 17-a4 present the density scatter plots for discovery cost over simulation time (millisecond) for 100 successive main-queries (i.e., discovery requests) per requester in dynamic computing environments with different range of task duration (i.e., application execution) (0,2000] ms, (2000,400] ms, (4000, 6000] ms and (6000, 8000] ms accordingly. Each data point in the graphs represents the result of a single main-query. The darker points in the graphs (i.e., the high density points) represent states that have a higher probability of occurrence in comparison to the lighter points. As presented in these graphs, the majority of the queries results, particularly the high dense data points, fall on or below the regression line. Similar behavior can also be seen in the Figs 17-b1 , 17-b2 , 17-b3 and 17-b4 for discovery latency evaluation over time. This illustrates that HARD3 is highly scalable over time and can efficiently maintain its performance under natural churn, caused by high frequent resource reservations and resource releases in highly dynamic computing environment.

In the aforementioned experiment results, when we vary the distribution range of the task duration from (0,2000] ms in Figs 17-a1 and b1 to (6000, 8000] ms in Figs 17-a4 and b4 while the range for querying intervals is fixed to [2000,6000] ms, as it is expected, the discovery cost and discovery latency are gradually increased. The reason is that for the larger application execution times, it takes longer for the reserved resources to be released by the original requesters and this leads to higher rate of unavailability for the occupied resources in the system. In this regard, the frequent resources (i.e., very common resources) in the system, might become the rare resources over time with higher cost of discovery. Discovering rare resources might be costly due to the potentially larger search space that is needed to be explored. In addition, the rate of resource unavailability (and becoming rare) would be accelerated particularly when the overall task duration (i.e., application execution time) exceeds the overall querying interval.

The dispersion of the data points is bigger for the graphs with the larger application execution times which shows the impact of resource unavailability, rare resources and resource contentions on the HARD3 performance. The mean hit rates (i.e., the success rate of querying) for querying in the experiments presented in Figs 17-a1, 17-b1, 17-a2, 17-b2, 17-a3 and 17-b3 are 100%, which means that all the generated discovery requests by requesters in the system are fully resolved. In Figs 17-a4 and 17-b4, because of the larger tasks duration, the amount of

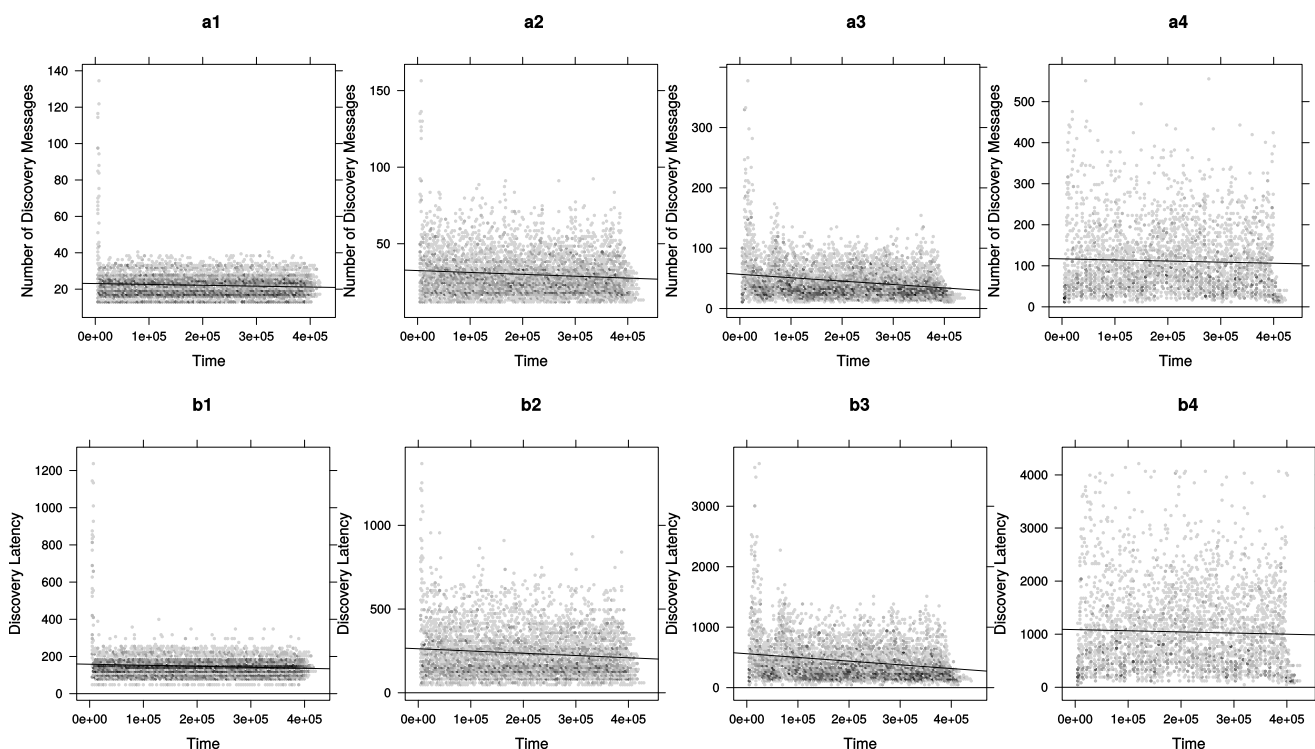


Figure 17: Density scatter plot of discovery cost (i.e., number of transacted discovery messages) and latency for fully resolved discovery requests (i.e., hit rate=100%) over time for various application size (i.e., application execution time) while the querying interval is [2000,6000] ms: a1 & b1) execution time=(0,2000) ms, a2 & b2) execution time=(2000,4000) ms, a3 & b3) execution time=(4000,6000) ms, a4 & b4) execution time=(6000,8000) ms.

available resources is decreased in most of the time-slices, while the amount of requesters in the systems, issuing new queries, is constant. This can lead to resource contention among the requesters, which in turn reduces the overall hit rate for the discovery requests in the system. The density of the data points in Figs 17-a4 and 17-b4 is reduced in comparison with other graphs. It means that the number of fully-resolved main-queries is decreased. Thus, as it is shown in Figure 18, the mean hit rate is expected to be reduced for the task duration=(6000,8000) ms.

Figure 18 illustrates the hit rate for querying in the system with different task duration. It shows that the mean hit rate is decreased while we increase the tasks duration, and the hit rate is equal to 100% when the overall execution time is less than querying interval. In fact HARD3 is able to precisely and successfully discover all the desired resources for the discovery requests without any specific limitations. The hit rate reduction only happens when there are not enough available matched resources for all the concurrent discovery requests in the system in a moment of time, which leads to resource contentions. But the HARD's performance and efficiency in itself is completely isolated from the external conditions such as resource unavailability and resource contention.

6.5. Comparison with Different Proposals

In this section, we present the simulation results which demonstrate the performance of our resource discovery scheme (i.e., HARD) in comparison to other alternative approaches. For comparison, we simulate our discovery scheme, HARD, in conjunction with three generic hybrid distributed approaches:

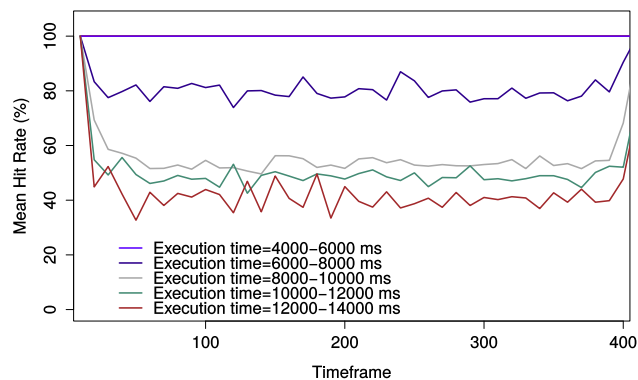


Figure 18: Mean hit rate for queries within the time-frames (each time-frame=10000 ms) for different application execution times in asynchronous querying with system-size=27500 resources.

a 2-layered hybrid DHT, learning-based and partial random-walk based discovery (PRW2), a 2-layered hybrid DHT and full random-walk based discovery (FRW2) and a 2-layered hybrid broad-cast and full random-walk based discovery (BRW2).

We also simulate two versions of HARD, HARD3 and a 2-layered instantiation of HARD (HARD2) for assessing the impact of hierarchy and our proposed layer-based query resolution methods on the HARD's performance. We already discussed and presented the details of HARD3 in the previous sections. HARD2 is a two-layered non-anycast based implementation of

HARD, which is identical to HARD3 except that it doesn't support SN layer and SQMS providers and instead it resolves any SN-dependent queries (e.g., $\langle n_{ln}, n_{an}, c_{sn} \rangle$) by extending the native probability mechanism of HARD3 to support the layer_{sn} resource information within layer_{an}.

Similar to HARD2, PRW2, FRW2 and BRW2 are organized on top of two-layered (i.e., leaf-node layer and aggregate-node layer) distributed hierarchies. PRW2 and FRW2 leverage the same Chord based DHT method which is used in HARD3 in the leaf-node layer while they provide different query forwarding methods in the aggregate-node layer. PRW2 uses both probability and random-walk method to guide queries in layer_{an}. In this approach, distributed probability tables in the system only process the query results with respect to the resource information in layer_{ln} and layer_{an}. PRW2 might behave similar to HARD2 for non-SN-dependent queries (e.g., $\langle c_{ln}, c_{an}, n_{sn} \rangle$), but for SN-dependent queries the selection of the forwarding destination node is partially random, since the probability tables do not actually care about the required resource conditions in the super-node layer for these queries.

PRW2 is comparable to our approach (i.e., HARD3) in the sense that it creates clusters on top of the underlying network. It also provides similarity to some well-known request propagation strategies in the literature such as the shortcut, random walk, learning-based, best-neighbor, learning-based+best-neighbor methods. These methods have been used in many popular resource discovery systems and applications [36–41]. For example in Iamnitchi et al [42] a fully decentralized discovery approach is proposed, which is based on publish/subscription of the resource information on some specific nodes in the virtual organization. Learning-based and also random-walk methods are used to propagate the queries among the server nodes. Our approach (and PRW2) are not based on publish/subscription since it has costs in terms of network traffic, processing, and storage needs for periodical updating and the maintenance of resource information particularly in high dynamic environment. On the other hand, our probability mechanism is comparable to, or even better than, learning-based strategies. In the learning-based method, nodes learn from experience by recording the requests answered by other nodes (i.e., by caching the results of successful queries). A request is forwarded to the node that has answered similar queries previously [43]. This strategy becomes inefficient when the system size, dynamicity and heterogeneity of resources/queries increases due to the larger memory requirements to maintain the query results and unavailability of the pre-discovered resources. But in our proposed probability mechanism, the statistical information about all the transacted queries by each peer are aggregated in the fixed-size DPTs regardless of the successfulness of the queries. In addition, by leveraging techniques such as dynamic best SoR detection, low-resource nodes and resource unavailability detection and various situation-based policies and updating strategies (e.g., shortest path, latency-aware and attribute-based updating) our proposed probability method provides better accuracy and efficiency.

Unlike PRW2, FRW2 employs a fully single random-walk method to guide all type of queries in aggregate-node layer. Random-walk is a common query forwarding method, which

is originally proposed in the literature to alleviate the excessive traffic problem caused by flooding [44], and to deal with the traffic/coverage trade-off. Random-walk is used in many distributed resource discovery applications such as Gnutella [45, 46], Iamnitchi et al [42] and [47–51].

BRW2 or Broad-Walk is a hybrid two layered approach which uses broadcast-based query propagation method [52, 53] in the leaf-node layer and the random-walk forwarding in the aggregate-node layer. In continuation of this section we explain the details of our simulation setup and and we discuss the comparison results.

6.5.1. Simulation Setup

Using our self-organized clustering algorithm we simulate the aforementioned discovery approaches, on top of either two-layered or three-layered distributed hierarchies. Similar to the previous scenarios, we simulate dynamic computing environment containing various number of computing resources, in which a constant number of resources (i.e., requesters) simultaneously issue the discovery requests to the system. The time interval between each pair of consecutive queries issued by a requester is defined by an exponential distribution. We also assume that each requester issues 10 consecutive resource requests to the system over the simulation time. The discovered resources will be reserved for each discovery request. The reserved resources for each process will be released after execution time period which is defined by a Weibull distribution. We execute 10 queries per requester for each system size, each one originating from a uniformly chosen source-node. Each experiment for each system-size is repeated for 100 runs with different topology parameters. All the queries are identical and represent the queries of type $\langle c_{ln}, c_{an}, c_{sn} \rangle$. Each requester is willing to find required resources for a process containing three thread-groups with different resource requirements. Table 8 presents more details of simulation parameters for our evaluation.

Table 8: Simulation Parameters for performance compression

Parameter	Values
Physical network size	5500-55000 cores
Interconnect topology	Mesh/Torus
Network topology	Random
Interconnect channel datarate	50Gbps
Network channel	100 Mbps
Desired Resources for each Request	3x20
Homogeneity rate of desired resources	33%
Frequency of Target Resources (FTR)	1650
Process Duration by sec	Weibull($\lambda=3.58, k=2.40$)
Querying Interval by ms	Exponential($\beta=4000$)
Consecutive Query Runs per Requesters	10
Simulation runs	100 per system size per approach
Rate of Requesters	1%

6.5.2. Simulation Results

In the first test, we perform experiments to measure the average number of required messages, and the average latency (by milliseconds) per discovery request for HARD3, HARD2 and other approaches. The PRW2, FRW2 and BRW2 with the same topology, simulation parameters and conditions are used as alternative reference works. Figs 19-a, 19-b and 19-c show plots of the average discovery messages, and the average discovery latency per query, as a function of the number of computing resources in the system (i.e., system size). Figure 19-b

demonstrates the results presented in the Figure 19-a with better resolution (without BRW2).

In Figure 19-a, we observe that the average required number of discovery messages per query for BRW2 is much larger than the other approaches, while in Figure 19-c the average latency of BRW2 is close to FRW2, PRW2 and HARD2. This means that BRW2 significantly generates more discovery traffic in comparison to others, due to the heavy cost of broadcasting in $layer_{in}$. The queries in BRW2 are guided in the aggregate-node layer by being forwarded to a non-visited single random neighboring aggregate-node. Upon arrival of a query in an aggregate-node, if that node fits the query conditions (i.e., c_{an}) for the current layer (i.e., $layer_{an}$) the query is broadcasted to all the leaf-node members of the current aggregate-node, otherwise it is forwarded further in the network using random-walk. Broadcasting results in increased traffic, but as seen in Figure 19-c, this could provide reasonable response time for queries, since the aggregate-node inquires all of its leaf-node members in parallel. The response time for BRW2 is approximately close to the results for FRW2, PRW2 and even HARD2.

Figs 19-b and 19-c show that our approach, HARD3, provides the highest performance and scalability among others for both discovery traffic (i.e., average number of discovery messages propagated during a search), and latency when varying the number of resources in the system from 5500 to 55000 resources. This is particularly significant for the query's response time (latency) since other approaches, such as HARD2 and PRW2, also provide close results in terms of number of discovery messages. HARD3 efficiently divides the exploring space to the anycast groups in a way that, queries with c_{sn} requirements are only propagated among the SQMS providers whose specifications in $layer_{sn}$ fulfill the c_{sn} conditions of the given query essentially. In comparison to HARD2, this strategy leads to a significant reduction in the response time of HARD3 while its discovery traffic is also slightly decreased. As we already discussed, HARD3 is the enhancement of HARD2 by leveraging our proposed anycast forwarding mechanism in an extra layer which is called $layer_{sn}$. The presented results for HARD2 and HARD3 in Figs 19-b and 19-c also prove that increasing the level of hierarchy along with the implementation of an efficient adaptive corresponding query processing method improves the overall performance of our discovery system. Another factor contributing to HARD3's overall performance is that HARD3 controls the discovery procedure in

a more intelligent way which saves much unnecessary message cost. Moreover due to the anycast nature of HARD3, SQMS providers are able to effectively guide the given queries to the closest qualified SQMS provider in the system which results in a significant reduction in the discovery latency for the queries in the system (see Figure 19-c).

From Figure 19-b, we can also see that, PRW2 generates larger number of discovery messages per query than HARD2 and HARD3 because of its partial random-walk query forwarding mechanism in $layer_{an}$. In fact, PRW2 provides an efficient probability mechanism (similar to HARD2) to guide queries in $layer_{an}$ to the potential matched resources in the system. But this probability mechanism becomes inefficient for processing the queries with c_{sn} requirements because the DPTs in PRW2 do not consider the c_{sn} requirements of the given queries in order to statistically estimate the target aggregate-node for query forwarding. This leads to a sort of partial random-walk for the SN-dependent queries (i.e., for the queries that $c_{sn} < n_{sn}$). But for the other types of queries (e.g. $< c_{in}, c_{an}, n_{sn} >$), which are not considered in our evaluation in this section, PRW2 is expected to behave identically to HARD2.

Figure 19-b illustrates that FRW2 provides a lower performance with respect to discovery overhead compared to PRW2 while they exhibit almost similar behavior for discovery latency as shown in Figure 19-c. This is due to the fact that the mechanism for query resolution in $layer_{an}$ of FRW2 is fully based on random-walk method. This means that the queries in $layer_{an}$ are forwarded to a uniformly random selected neighboring aggregate-node in the system which is not yet visited. Since the next-node selection strategy is completely random-based the number of required traversed discovery messages for resolving a query would get more compared to the approaches benefiting a type of estimation-based strategy.

In the next experiments we analyze the dynamic behavior of the above-discussed approaches for various system size over time. The simulation results for both average generated discovery messages and average discovery latency per request per time-frame (1000 ms) for HARD3, PRW2 and FRW2 are depicted in Figs 20-a1/a2, 20-b1/b2 and 20-c1/c2 respectively. These results lead to the following conclusions:

(a) The overall variation and fluctuation in the query results (in terms of discovery overhead) per time-frame for HARD3 is less than PRW2 and FRW2. This means that HARD3 provides

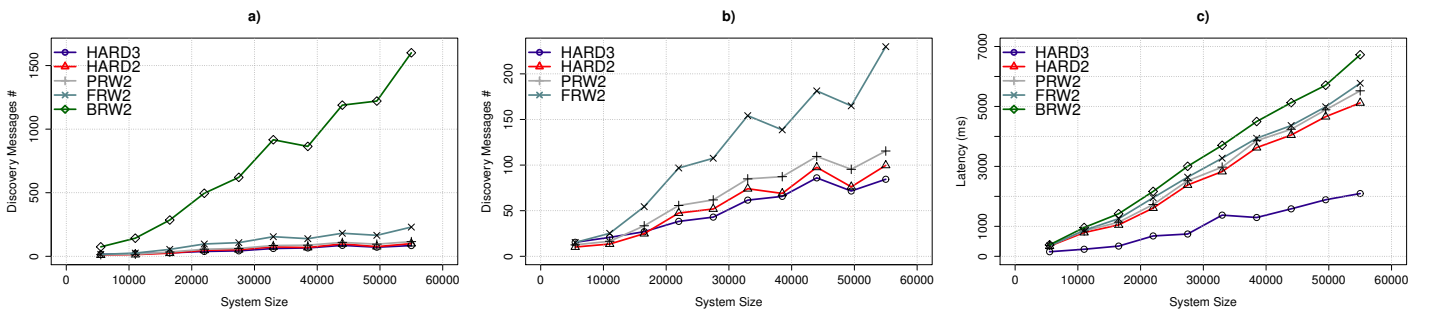


Figure 19: Comparison between HARD3 and other alternative approaches: a, b & c) Average number of required discovery messages and discovery latency per discovery request for different system size and different strategies

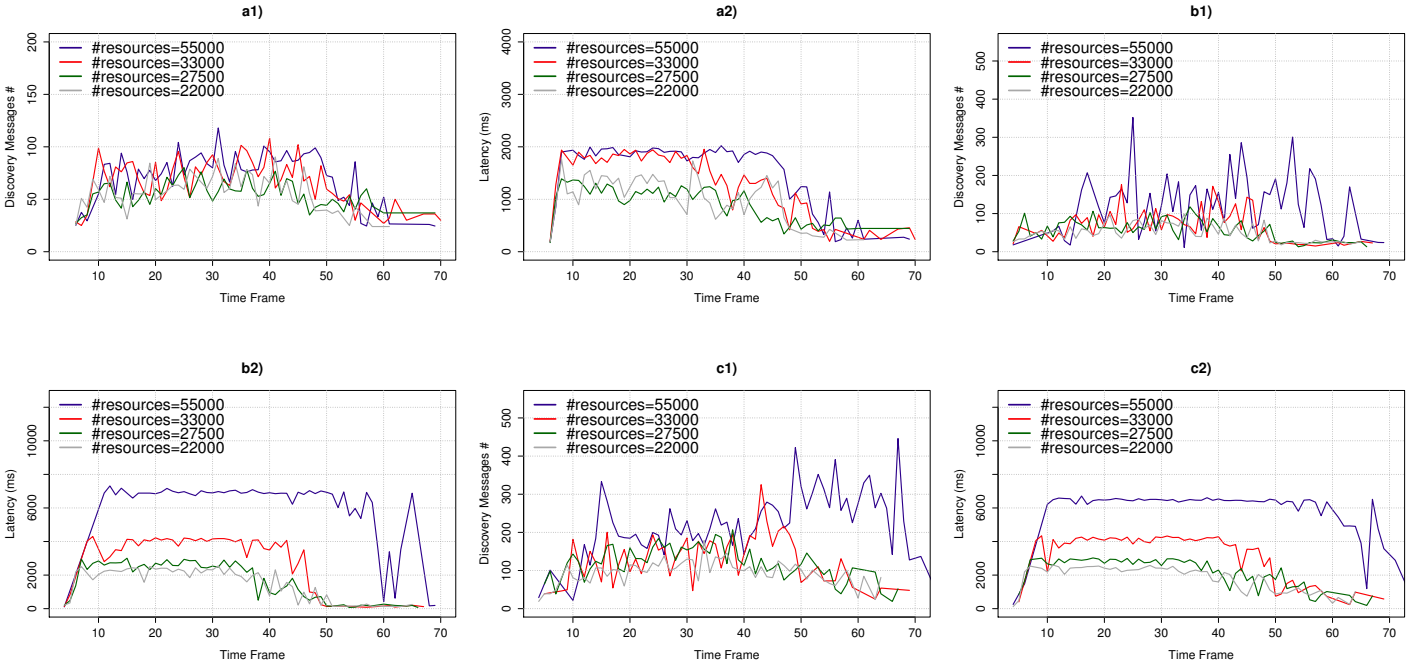


Figure 20: Comparison between HARD3 and other alternative approaches: a1 & a2) Mean number of discovery messages and discovery latency per request per time-frame (1000 ms) over time in different system size for HARD3, b1 & b2) Mean number of discovery messages and discovery latency per request per time-frame (1000 ms) over time in different system size for PRW2, c1 & c2) Mean number of discovery messages and discovery latency per request per time-frame (1000 ms) over time in different system size for FRW2.

better scalability over time with respect to discovery overhead.

(b) The figures that demonstrate the amount of discovery overhead per time-frame for various system size for HARD3 provide relatively better approximation of the overlap than the corresponding charts for PRW2 and FRW2. This means that HARD3 outperforms the other approaches with respect to scalability (in terms of discovery overhead) when varying the number of computing resources in the network from 22000 to 55000. This behavior can also be seen in the figures illustrating the average discovery latency per time-frame for various number of computing resources. Thus, we can conclude that HARD3 provides better scalability (in terms of discovery latency) for different system sizes.

(c) The latency figures in all the approaches approximately provide similar steady behavior except for a portion of time

when the requesters gradually start or stop sending series of discovery requests to the network. This unsteadiness happens because we assumed that at the beginning of the simulation all the computing resources in the system are free (not reserved), and that is how the initial queries for each requester would be able to discover their desired resources in a shorter time. As time proceeds, the total number of reserved resources in the network to execute the waiting processes of the requesters increases which leads to increasing the response time for the new queries. The requesters will release the reserved resources for each of their processes after the execution terminates. This is the reason for the dramatical increment of the response time of the queries at the beginning of the experiments. Similarly, the response time for the queries decreases dramatically at a portion of time at the end of the experiment, when the requesters in the

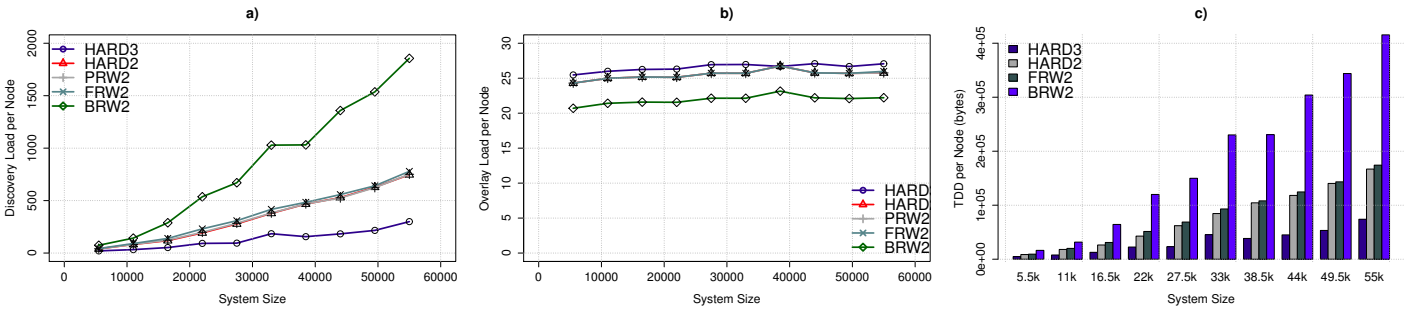


Figure 21: Comparison between HARD3 and other alternative approaches: a & b) Average discovery load (number of transmitted discovery messages) per node (i.e., vnode) and average overlay load (number of transmitted messages for overlay construction) per node during simulation time (60000-80000 ms) for various system sizes, c) Average transited discovery data per node during simulation time (60000-80000 ms) for various system sizes.

system gradually stop sending new queries after issuing fixed number of successive queries. A requester may stop querying earlier or later than other requesters depending on its various querying interval time before generating each new query. With respect to the aforementioned behavior of the latency charts in our experiments we can conclude that all the approaches provide scalability for discovery latency over time. But HARD3 shows better scalability (in terms of latency) than others for different system sizes as elaborated earlier.

In the next simulation, we measure the overall discovery load per node (i.e., $vnode$) during the querying period (60000-80000 ms), which is the duration of time that the requesters propagate a constant number of successive queries across the network in a parallel manner. The querying period ends when the request-initiator corresponding to the last query is replied. The discovery load is the average number of transacted discovery messages by each $vnode$ during the querying period. We also measure the overlay load per node (i.e., the average number of transacted overlay messages per node to create the underlying hierarchical system overlay) and TDD per node (the average amount of transmitted discovery data per node) during the querying period. Figs 21-a, 21-b and 21-c depict the discovery load per node, overlay load per node and TDD per node respectively, as the function of system size for different approaches. As we can see in Figs 21-a and 21-c, HARD3 shows better performance compared to other solutions. However Figure 21-b shows that the overlay cost (in terms of the average transacted overlay messages per node) to establish the underlying overlay for HARD3 is relatively larger than in others. As we already discussed, all approaches employ the same multistage hierarchical overlay algorithm [31] to implement either two-layered or three-layered structure. The overlay cost to create the structures with higher levels of hierarchy, like HARD3-overlay, which has three layers is bigger than the structures with lower levels of hierarchy, such as the underlying overlay of HARD2, PRW2, FRW2 and BRW2. The overlay cost for BRW2 is the least among the other two-layered based approaches because it provides a non-DHT based approach for query processing in $layer_{in}$ unlike HARD2, PRW2 and FRW2. BRW2 does not require the creation of a DHT structure within the $layer_{in}$ which has some extra overlay cost.

6.6. Other Features

Our proposed discovery approach provides a set of important functionalities/features in order to support complex and flexible querying within large scale distributed systems. In this section, we briefly compare our approach with some discovery examples in the literature in terms of querying features. These approaches include SWORD [54], Node-Wiz [55], MDS-4 [56], MatchTree [57], CycloidGrid [58] and OntoSum [59]. Table 9 presents the result of our qualitative assessment. Following, we provide a short description for some of the features, used in the comparison.

Nearest Neighbor Query is a discovery capability to provide a list of discovered resources considering the priority of the closer neighbors. Similar Matching is the ability to find a similar match for a query with respect to query conditions. Resource Graph Discovery is the capability to discover a graph of resources

considering both individual characteristics and interconnecting properties of resources. Multi-Dimensional and Range Querying is the capability of the discovery system to process queries containing multiple attributes either dynamic or static within specific ranges of values. Thread-level discovery specifies the capability of the discovery system to deal with the query conditions in thread-level (i.e., resource requirements for each thread in a process).

Overall, HARD3 seems to present the best solution for JCS, with the most complete set of features. Even of HARD3 does not provide "Nearest Neighbor Query", it is proximity-aware, which copes with this issue. Also, HARD3 is inherently providing "Load-Balancing", due to the probability aspects of the search algorithms.

7. Related Work

7.1. Resource Description

In general, there are two types of approaches for resource description in the current literature: attribute-based and semantic-based schemes.

The attribute based schemes define the resource characterizations through a set of (attribute, value) pairs. Depending on the level of information details and the different storage, retrieval and distribution mechanisms these approaches are able to provide a scalable distribution of resource information. However, the attribute-based description models are facing some challenging issues such as providing appropriate support for dynamic and collective attributes [60]. Dynamic attributes are changing frequently, which results that the above mentioned description models becomes unappropriated to store their values due to expensive updating and maintenance cost. In our description model, we can define dynamic and static attributes for both systems and queries. However, we do not use dynamic attributes in the body of layer-stamps. Instead, dynamic attributes can be defined as independent-attributes, whose their values can be extracted real-time and upon receiving a request from a resource provider entity.

WSDL [61], OASIS UDDI [62] and [63] are some examples of attribute-based resource descriptions which have been used in many resource discovery solutions specially in web-based resource discovery protocols. WSDL is a set of instructions to describe the behavior, characteristics and formats of services, particularly for web service providers. UDDI uses a XML-based repository to provide policies and standards for service discovery which facilitate the process of resource advertisement and service publication. Tutschku et al, a recent work [63], is proposing a resource description method based on the capabilities of on-board Linux tools for describing resource utilisation in cloud networking and NFV infrastructures. It aims to provide a description approach for dynamic attributes such as CPU load and utilization. However, the approach is not general (it is based on linux) and is limited by very abstract description of resources and also the description is restricted to a very few number of predefined general attributes. In addition to the attributed-based description languages, there are a number of

Table 9: An overall comparison between HARD3 and examples of other discovery approaches in terms of querying features.

Functionalities	SWORD	Node-Wiz	MDS-4	MatchTree	CycloidGrid	OntoSum	HARD3
Load Balance		✓		✓	✓		
Fault Tolerance		✓	✓	✓			✓
Self-Organization	✓	✓		✓		✓	✓
Range Query	✓	✓		✓			✓
Similar Matching							✓
Multi-Dimensional Query	✓	✓		✓			✓
Resource Graph Discovery	✓						✓
Nearest Neighbor Query	✓					✓	
Proximity-Awareness					✓		✓
Resource Reservation			✓		✓		✓
Semantic-Awareness						✓	
Thread-Level Discovery							✓

recent attributed-based (resource information) encryption methods in the current literature including [64–69] which are mostly application-oriented. And in fact, they are not really flexible and even efficient to be used for different applications (in this paper we proposed our own encryption method).

Semantic-based description models are alternative approaches which focus on the overall collaborative description of the resources. These approaches are appropriate to describe all system resources where all the possible collaborative system and resource properties and behaviors (e.g., the structure of the processor or memory architectures) are precisely described, but we must take into account that these approaches might not be scalable in terms of distribution of the resource information. RDF [70] is an example of classic semantic-based resource description which provides a type of ontology/knowledge representation approach, which is a primitive language providing a binary relation of classes and properties supporting all kind of range/domain constraints and sub-property/sub-class relationships. However, RDF lacks support of expressive queries which can add useful semantic information to descriptions.

The works in [71–73] are examples of semantic-based schemes which use functional languages to describe hardware resources. The use of higher-order functions allows the composition of arbitrarily complex structures in a clear and concise way. The strong type system of most functional languages also ensures the soundness of the composition of the different hardware components. Functional resource description models focus on capturing the structure as well as numerical properties of hardware resources. Resource descriptions themselves are functions, capturing the fact that behaviors/capabilities relevant for a resource can change under certain circumstances. Additionally, functions can be used to concisely and clearly capture complex, parameterizable collaboratives.

The Lexical Bridge [74] is a recent work which proposes a methodology to translate meaningful information in natural language sources into a standardized, structured knowledge representation for the purposes of semantic normalization, integration, analysis, and reasoning. However, it is a very general work and in fact doesn't provide a resource description language (for the purpose of resource discovery in the distributed system) a complete resource description language, rather, it aims to build "lexical bridges" (LBs) in order to fill the gap between the natural languages and their ontology representations.

The authors in [75] have highlighted that a scalable resource description and information exchange (in terms of distribution

of resource information between Clouds) is an important requirement for sharing heterogeneous Cloud resources among federated Clouds. However, the work presented in this paper, a semantic based resource description model for inter-clouds, does not really provide a scalable solution for distributing all details of resource information in the entire system. It focuses on providing a scalable information exchange method between multiple clouds, where each cloud centrally manage its own resources. In this regard, the concept of scalability is far from scalable inter-resources information exchange in a purely decentralized environment.

Considering the above approaches, an optimum resource description model for resource discovery can be designed in such a way that it takes the concerns of both approaches (attribute-based and semantic-based): capturing individual and collaborative capabilities of resources while still allowing for scalable distribution of this information [75]. In comparison to the current literature, our description model/language is a domain specific language with an embedded encryption method, specifically designed for the purpose of resource discovery in large dimension many-core enabled future computing systems with capability for scalable distribution of encrypted resource information across the system. Furthermore, it is highly expressive and flexible to describe various complex queries/systems and detailed attributes/layers, making feasible to provide almost most of the necessary description requirements (scalability, flexibility, expressiveness and compact design) for resource sharing and discovery in such future systems. Furthermore, the proposed resource description model is computing-oriented, which means, unlike most of the approaches in the literature, we describe all types of resources (Compute, Network and Storage) from the viewpoint of computation. This inherently provides capability for the resource description model for being more adapted to the applications like discovering processors in distributed computing systems.

7.2. Resource Discovery

Besides algorithms were already discussed and compared, it is relevant to highlight that several resource discovery approaches have been proposed to enable grid information services on early examples of JCS [56]. They can be categorized according to their main approach to the problem: centralized and distributed.

The simplest approach to create an information service is the first: employ a centralized directory. As examples for this kind of resource discovery solutions we can find Condor [76],

Condor-G [77] and BOINC [78]. These approaches are only efficient for local area deployments. In large-scale systems, the central point of failure, and poor scalability bottleneck, make these solutions under-perform [79]. The major advantage of these solutions is the simplicity of finding all resource information on the central server, making the resource discovery latency low, and data coherence high. However these approaches suffer from sub-optimal scalability and lower fault tolerance, mostly due to the centralized nature of the directories. Another approach for discovery in grids relies in hierarchically (e.g., MDS[80, 81]) or semantically (e.g., OntoSum [59]) organized servers. In MDS [80, 81], a grid is composed by several resource description providers that are registered to index servers. Resource requesters query directory nodes to discover resource index servers, and to obtain more detailed resource information from their resource description providers. The index servers also follow an hierarchy. The top index server answers requests either directly or by dispatching requests to its child index servers. This approach limits scalability, as requests trickle through the root server, which can easily become a bottleneck and consequently suffer from fault tolerance issues. Indeed, the loss of a node in the higher level of the architecture causes the loss of an entire sub tree. OntoSum [59] organizes a Grid network by a semantically linked overlay, representing the semantic relationships between Grid participants. It improves the discovery efficiency in Grids through propagating the discovery requests only between semantically related nodes.

NodeWiz [55], Mercury[82], SWORD[54] and [83], in particular have investigated the issue of supporting multi-dimensional resource attributes and range-based querying, and improve the existing systems by resorting to DHTs or non-DHT based methods. The main disadvantages of the traditional DHT based discovery approaches (like SWORD) is the lack of support for partial matching (i.e., user needs to give exact keyword to search for information due to the hash table structure). In other hand, The non-DHT based solutions (like NodeWiz) also suffer from inefficiency caused by high response time of the queries due to unstructured techniques such as flooding and blind search.

Distributed discovery approaches have been specially designed to provide a high level of scalability and fault tolerance, which is required in large scale environments. Iamnitchi et al[84] proposes a solution for using the benefit of the distributed Peer-to-Peer (P2P) system for resource discovery in Grids. The combination of P2P and Grid RD models [85, 86] would be desirable to build fault tolerant and large scale distributed systems. There are two kind of approaches in this field which are based on structured and unstructured overlays networks. The first model uses an unstructured overlay network with flooding based query propagation. Relevant solutions are Zorilla [87] and Vishwa[88]. One of the advantages of these approaches is the ability to perform resource discovery with high expressiveness. However, the discovery systems are not exhaustive and efficient. The response time of the queries is high due to flooding and blind search. Also, rarer resource information may be unable to be found. Moreover, one of the common limitations of current grid-based discovery approach is that the queries are in the task-level, resulting in a course-grained discovery. This can reduce

the efficiency of such approaches to cope with the problem of resource sharing/allocation with respect to a high resolution of future many-core systems and also future parallel applications. This is the point that providing a fine-grained adaptive discovery solution becomes important.

The other common model for resource discovery are the discovery systems for P2P networks which offers a significant advantage over their hierarchical counterparts by the way of resistance to failure and traffic congestion. In particular, structured P2P systems based on DHTs such as Pastry [89] and Tapestry [90] are very popular for file-sharing applications but not for sharing resource information. Moreover, typical structured P2P systems such as Chord[91], CAN[92] and Pastry are very sensitive to churning leading to resource unavailability. These systems achieve good performance and scalability characteristics but they are limited to only support exact matching. Moreover their hashing functionalities performs well with static attributes, however they fail in handling dynamic objects appropriately.

Routing Indices (RI) [93] is another form of resource discovery which uses distributed indices in unstructured P2P networks. The advantage of this mechanism relies in the fact that queries are disseminated and forwarded only among the places of the network where resources existed, thus avoiding to flood query requests to the nodes which are not useful. The main drawback of this solution is that this indexing system comes from the presence of cycles in the network graph. A recent work [83] of this type extends RI and proposes a technique to perform resource discovery in grids based on P2P with capability to perform multi-attribute queries and range queries for numerical attributes. It uses an information summarization technique presented in [94] and creates different types of summaries and accordingly presents a metric (called goodness function) needed by RIs to guide the query process. It still suffers from RI drawbacks as well as lack of support for complex querying. A similar proposal [95] presents a task/job-level resource discovery with limited flexibility of querying to handle multi-core machines in desktop grids. This technique handles resource availability based on a few set of numerical parameters, such as CPU speed, number of cores, and memory availability.

For unstructured P2P discovery systems both informed search and blind search can be used. Blind search uses flooding which burdens the communication network heavily, and requires hop limitation and loop limitation mechanisms. One of the best known systems using flooding is probably Gnutella [96], which is also used by CORBA traders[97]. In Gnutella the discovery requests are routed to all neighbor nodes of a given node. This keeps happening until the queries expire or until the matched resources are retrieved. The flooding mechanism creates a large volume of traffic for networks with many nodes, connections and resources.

Further examples of P2P based discovery approaches include MatchTree [57], CycloidGrid [58]. MatchTree proposes a scalable and fault tolerant system by creating a self-organized tree for query distribution and result aggregation with a specific asymptotic latency increase pattern. It reduces the query latency and improves the system fault tolerance through redundant query topologies, sub-region queries, and dynamic timeout policies

and set of dynamic timeout policies. It supports complex queries and guarantees query completeness. CycloidGrid provides a two stages QoS and locality aware discovery algorithm for P2P based volunteer computing systems. In the first stage, it discovers a set of resources based on the required quality of service and the current load of the peers, and in the next stage it selects the closest resource in terms of communication delay (latency) between peers which is calculated using a network model based on queuing theory with considering the background traffic of Internet.

Finally referring to the current state of the art, to the best of authors knowledge, there is no extensive work in the literature, addressing specifically the problem of resource discovery (in thread-level) with respect to the various and complex requirements of future large dimension many-core enabled computing systems (such as high heterogeneity, scalability, dynamicity, efficiency, adaptability, querying flexibility in terms of complex querying, query expressiveness, resource graph discovery, etc).

8. Conclusion

This paper proposes HARD, a novel efficient and highly scalable resource-discovery approach, which deals with the resource discovery requirements in computing environments such as high-hierarchy, high-heterogeneity and high-scalability and dynamicity. The approach is based on self-organization and self-adaptation of processing resources information in the system, where the computing resources are organized into distributed hierarchies according to a proposed hierarchical resource description model (i.e., multi-layered resource description). Our simulation results assure the significant scalability and efficiency of HARD3 (an implementation of HARD) over highly heterogeneous, hierarchical and dynamic computing environments with respect to several scalability and efficiency aspects while supporting flexible and complex queries with guaranteed discovery results accuracy. We further showed that HARD3 outperforms different other potential strategies.

9. Acknowledgment

The authors acknowledge the support of project FP7-ICT-2009.8.1, Grant Agreement No.248465, Service-oriented Operating Systems (2010-2013) [9–17] and of project Cloud Thinking (2013-2015), CENTRO-07-ST24-FEDER-002031 [98].

References

- [1] J. Maassen, N. Drost, H. E. Bal, F. J. Seinstra, Towards jungle computing with ibis/constellation, in: Proceedings of the 2011 Workshop on Dynamic Distributed Data-intensive Applications, Programming Abstractions, and Systems, 3DAPAS '11, ACM, New York, NY, USA, 2011, pp. 7–18.
- [2] F. J. Seinstra, J. Maassen, R. V. Van Nieuwpoort, N. Drost, T. Van Kessel, B. Van Werkhoven, J. Urbani, C. Jacobs, T. Kielmann, H. E. Bal, Jungle computing: Distributed supercomputing beyond clusters, grids, and clouds, in: Grids, Clouds and Virtualization, Springer, 2011, pp. 167–197.
- [3] M. Hajibaba, S. Gorgin, A review on modern distributed computing paradigms: Cloud computing, jungle computing and fog computing, CIT. Journal of Computing and Information Technology 22 (2014) 69–84.
- [4] Y. Wang, T. Uehara, R. Sasaki, Fog computing: Issues and challenges in security and forensics, in: Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual, volume 3, IEEE, pp. 53–59.
- [5] R. S. Segall, N. Gupta, Overview of global supercomputing, Research and Applications in Global Supercomputing (2015) 1.
- [6] E. Jeannot, J. Zilinskas, High-performance Computing on Complex Environments, volume 96, John Wiley & Sons, 2014.
- [7] D. D'Agostino, F. J. Seinstra, A parallel isosurface extraction component for visualization pipelines executing on {GPU} clusters, Journal of Computational and Applied Mathematics 273 (2015) 383–393.
- [8] N. Drost, J. Maassen, M. A. J. van Meersbergen, H. E. Bal, F. I. Pelulessy, S. P. Zwart, M. Kliphuis, H. A. Dijkstra, F. J. Seinstra, High-performance distributed multi-model / multi-kernel simulations: A case-study in jungle computing, in: Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International, pp. 150–162.
- [9] L. Schubert, A. Kipp, Principles of service oriented operating systems, in: P. Vicat-Blanc Primet, T. Kudoh, J. Mambretti (Eds.), Networks for Grid Applications, volume 2 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, Springer Berlin Heidelberg, 2009, pp. 56–69.
- [10] The S[o]OS Consortium , S(o)OS (Service-oriented Operating System): Resource-independent execution support on exa-scale systems, Available at <http://www.soos-project.eu/>, <http://www.soos-project.eu/index.php/publications>, 2010-2013. [Online: accessed 5-September-2014].
- [11] J. Zarrin, R. L. Aguiar, J. P. Barraca, Elcore: Dynamic elastic resource management and discovery for future large-scale manycore enabled distributed systems, Microprocessors and Microsystems (2016) –.
- [12] C. P. R. Baaij, J. Kuper, L. Schubert, SoSiM: Operating System and Programming Language Exploration, in: G. Lipari, T. Cucinotta (Eds.), Proceedings of the 3rd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time System (WATERS), pp. 63–68.
- [13] G. Lipari, E. Bini, A framework for hierarchical scheduling on multi-processors: From application requirements to run-time allocation, in: Real-Time Systems Symposium (RTSS), 2010 IEEE 31st, pp. 249–258.
- [14] T. Cucinotta, Challenges in operating system design for future many-core systems, All Hands Meeting (AHM) 2010, Cardiff, UK, Available at <http://retis.sssup.it/~tommaso/presentations/AHM-2010.pdf>, 2010. [Online: accessed 15-April-2016].
- [15] L. Schubert, Dynamicity requirements in future cloud-like infrastructures, Invited Speaker, EuroCloud CLASS Conference, Available at http://videlectures.net/classconference2012_schubert_infrastructures/, 2012. [Online: accessed 15-April-2016].
- [16] L. Schubert, A. Kipp, S. Wesner, Above the clouds: From grids to service-oriented operating systems., in: Future Internet Assembly, pp. 238–249.
- [17] J. Zarrin, R. L. Aguiar, J. P. Barraca, Dynamic, scalable and flexible resource discovery for large-dimension many-core systems, Future Generation Computer Systems 53 (2015) 119–129.
- [18] K. Sathish, A. RamaMohan Reddy, Workflow scheduling in grid computing environment using a hybrid gaaco approach, Journal of The Institution of Engineers (India): Series B (2016) 1–8.
- [19] H. B. Prajapati, V. A. Shah, Scheduling in grid computing environment, in: 2014 Fourth International Conference on Advanced Computing & Communication Technologies, IEEE, pp. 315–324.
- [20] F. P. Miller, A. F. Vandome, J. McBrewster, Huffman Coding: Computer Science, Algorithm, Lossless Data Compression, Variable- Length Code, David A. Huffman, Doctor of Philosophy, Massachusetts Institute of Technology, Alpha Press, 2009.
- [21] F. P. Miller, A. F. Vandome, J. McBrewster, Lossless Data Compression: Data Compression, Algorithm, Lossy Compression, Bit Rate, ZIP (File Format), Unix, Gzip, Portable Network Graphics, Graphics Interchange Format, Tagged Image File Format, Alpha Press, 2009.
- [22] J.-L. Zhou, Y. Fu, Scientific data lossless compression using fast neural network, in: Proceedings of the Third International Conference on Advances in Neural Networks Volume Part I, ISSN'06, Springer-Verlag, Berlin, Heidelberg, 2006, pp. 1293–1298.
- [23] M.-B. Lin, Y.-Y. Chang, A new architecture of a two-stage lossless data compression and decompression algorithm, IEEE Trans. Very Large Scale Integr. Syst. 17 (2009) 1297–1303.
- [24] B. Alik, N. Lukač, Chain code lossless compression using move-to-front

- transform and adaptive run-length encoding, *Image Commun.* 29 (2014) 96–106.
- [25] A. Moffat, R. M. Neal, I. H. Witten, Arithmetic coding revisited, *ACM Trans. Inf. Syst.* 16 (1998) 256–294.
- [26] F. M. Willems, Y. M. Shtarkov, T. J. Tjalkens, The context-tree weighting method: basic properties, *Information Theory, IEEE Transactions on* 41 (1995) 653–664.
- [27] P. Fenwick, Burrows & Wheeler compression: Principles and reflections, *Theor. Comput. Sci.* 387 (2007) 200–219.
- [28] L. L. Larmore, D. S. Hirschberg, A fast algorithm for optimal length-limited Huffman codes, *Journal of the ACM (JACM)* 37 (1990) 464–473.
- [29] Z. Chen, L. Wu, J. Zhang, X. Hu, Y. Xu, Heuristic resource discovery in P2P network, in: *Proceedings of the 25th international conference on Industrial Engineering and Other Applications of Applied Intelligent Systems: advanced research in applied artificial intelligence, IEA/AIE'12*, Springer-Verlag, Berlin, Heidelberg, 2012, pp. 333–342.
- [30] J. Zarrin, R. L. Aguiar, J. P. Barraca, A specification-based anycast scheme for scalable resource discovery in distributed systems, in: *10th ConfTele 2015 - Conference on Telecommunications*, pp. 13–17.
- [31] J. Zarrin, R. L. Aguiar, J. P. Barraca, A self-organizing and self-configuration algorithm for resource management in service-oriented systems, in: *19th IEEE Symposium on Computers and Communications (IEEE ISCC 2014)*, Madeira, Portugal, pp. 1–7.
- [32] C. Partridge, T. Mendez, W. Milliken, Host Anycasting Service, RFC, IETF, United States, 1993. RFC 1546.
- [33] S. Kumar, T. Cucinotta, G. Lipari, A latency simulator for many-core systems, in: *Proceedings of the 44th Annual Simulation Symposium, ANSS '11*, Society for Computer Simulation International, San Diego, CA, USA, 2011, pp. 151–158.
- [34] Andras Varga, Omnet++ discrete event simulator, Available at <https://omnetpp.org/>, <https://omnetpp.org/pmwiki/index.php?n=Main.SettingUpParallelDistributedSimulations>, 2003-2010. [Online: accessed 14-February-2017].
- [35] J. Zarrin, R. L. Aguiar, J. P. Barraca, Manycore simulation for peta-scale system design: Motivation, tools, challenges and prospects, *Simulation Modelling Practice and Theory* 72 (2017) 168 – 201.
- [36] S. Castano, A. Ferrara, S. Montanelli, D. Zucchelli, Helios: a general framework for ontology-based knowledge sharing and evolution in P2P systems, in: *Database and Expert Systems Applications, 2003. Proceedings. 14th International Workshop on*, pp. 597–603.
- [37] K. Sripanidkulchai, H. Zhang, Content location in peer-to-peer systems: Exploiting locality, in: X. Tang, J. Xu, S. Chanson (Eds.), *Web Content Delivery*, volume 2 of *Web Information Systems Engineering and Internet Technologies Book Series*, Springer US, 2005, pp. 73–97.
- [38] N. Bisnik, A. A. Abouzeid, Optimizing random walk search algorithms in {P2P} networks, *Computer Networks* 51 (2007) 1499–1514.
- [39] J. Li, R. Yahyapour, Learning-based negotiation strategies for grid scheduling, in: *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, volume 1, pp. 8 pp.–583.
- [40] A. Sharma, S. Bawa, Comparative analysis of resource discovery approaches in grid computing, *Journal of Computers* 3 (2008).
- [41] I. Filali, F. Huet, C. Vergoni, A simple cache based mechanism for peer to peer resource discovery in grid environments, in: *Cluster Computing and the Grid, 2008. CCGRID '08. 8th IEEE International Symposium on*, pp. 602–608.
- [42] A. Iamnitchi, I. Foster, D. Nurmi, A peer-to-peer approach to resource location in grid environments, in: *High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings. 11th IEEE International Symposium on*, pp. 419–.
- [43] A. Iamnitchi, I. Foster, A peer-to-peer approach to resource location in grid environments, in: J. Nabrzyski, J. Schopf, Jennifer M. Weglarz (Eds.), *Grid Resource Management*, volume 64 of *International Series on Operations Research and Management Science*, Springer US, 2004, pp. 413–429.
- [44] C. Papadakis, P. Fragopoulou, E. Markatos, E. Athanasopoulos, M. Dikaiakos, A. Labrinidis, A feedback-based approach to reduce duplicate messages in unstructured peer-to-peer networks, in: S. Gorlatch, M. Dane-lutto (Eds.), *Integrated Research in GRID Computing*, Springer US, 2007, pp. 103–118.
- [45] E. Pournaras, G. Exarchakos, N. Antonopoulos, Load-driven neighbour-hood reconfiguration of gnutella overlay, *Computer Communications* 31 (2008) 3030–3039. Special Issue: Self-organization and self-management in communications as applied to autonomic networks.
- [46] A. Furno, E. Zimeo, Self-scaling cooperative discovery of service compositions in unstructured {P2P} networks, *Journal of Parallel and Distributed Computing* 74 (2014) 2994–3025.
- [47] E. Jeanvoine, C. Morin, Rw-ogs: An optimized randomwalk protocol for resource discovery in large scale dynamic grids, in: *Grid Computing, 2008 9th IEEE/ACM International Conference on*, pp. 168–175.
- [48] R. Robinson, J. Indulska, The emergence of order in random walk resource discovery protocols, in: R. Khosla, R. Howlett, L. Jain (Eds.), *Knowledge-Based Intelligent Information and Engineering Systems*, volume 3683 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2005, pp. 827–833.
- [49] V. Bioglio, R. Gaeta, M. Grangetto, M. Sereno, Rateless codes and random walks for P2P resource discovery in grids, *Parallel and Distributed Systems, IEEE Transactions on* 25 (2014) 1014–1023.
- [50] D. Zhou, V. Lo, Cluster computing on the fly: resource discovery in a cycle sharing peer-to-peer system, in: *Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on*, pp. 66–73.
- [51] N. Bisnik, A. Abouzeid, Modeling and analysis of random walk search algorithms in P2P networks, in: *Hot Topics in Peer-to-Peer Systems, 2005. HOT-P2P 2005. Second International Workshop on*, pp. 95–103.
- [52] S. El-Ansary, L. Alima, P. Brand, S. Haridi, Efficient broadcast in structured P2P networks, in: M. Kaashoek, I. Stoica (Eds.), *Peer-to-Peer Systems II*, volume 2735 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2003, pp. 304–314.
- [53] M. Sharmin, S. Ahmed, S. Ahamed, Safe-rd (secure, adaptive, fault tolerant, and efficient resource discovery) in pervasive computing environments, in: *Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on*, volume 2, pp. 271–276 Vol. 2.
- [54] J. Albrecht, et al., Design and implementation trade-offs for wide-area resource discovery, *Acm Transactions on Internet Technology* 8(4) (2008).
- [55] S. Basu, L. Costa, F. Brasileiro, S. Banerjee, P. Sharma, S.-J. Lee, Nodewiz: Fault-tolerant grid information service, *Peer-to-Peer Networking and Applications* 2 (2009) 348–366.
- [56] N. J. Navimipour, A. M. Rahmani, A. H. Navin, M. Hosseinzadeh, Resource discovery mechanisms in grid systems: A survey, *Journal of Network and Computer Applications* 41 (2014) 389–410.
- [57] K. Lee, T. Choi, P. O. Boykin, R. J. Figueiredo, Matchtree: Flexible, scalable, and fault-tolerant wide-area resource discovery with distributed matchmaking and aggregation, *Future Gener. Comput. Syst.* 29 (2013) 1596–1610.
- [58] T. Ghafarian, H. Deldari, B. Javadi, M. H. Yaghmaee, R. Buyya, Cycloidgrid: A proximity-aware P2P-based resource discovery architecture in volunteer computing systems, *Future Gener. Comput. Syst.* 29 (2013) 1583–1595.
- [59] J. Li, Grid resource discovery based on semantically linked virtual organizations, *Future Gener. Comput. Syst.* 26 (2010) 361–373.
- [60] M. Siddiqui, T. Fahringer, Grid Resource Management: On-demand Provisioning, Advance Reservation, and Capacity Planning of Grid Resources, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 157–177.
- [61] E. Newcomer, *Understanding Web Services: XML, WSDL, SOAP, and UDDI*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.
- [62] S. Pastore, The service discovery methods issue: A web services uddi specification framework integrated in a grid environment, *J. Netw. Comput. Appl.* 31 (2008) 93–107.
- [63] K. Tutschku, V. A. Mehri, A. Carlsson, K. V. Chivukula, J. Christenson, On resource description capabilities of on-board tools for resource management in cloud networking and NFv infrastructures, in: *2016 IEEE International Conference on Communications Workshops (ICC)*, pp. 442–447.
- [64] V. Vaikuntanathan, P. Voulgaris, Attribute based encryption using lattices, 2016. US Patent 20,160,156,465.
- [65] S. Fugkeaw, H. Sato, Design and implementation of collaborative ciphertext-policy attribute-role based encryption for data access control in cloud, *Journal of Information Security Research* 6 (2015).
- [66] G. Baranwal, D. P. Vidyarthi, A fair multi-attribute combinatorial double auction model for resource allocation in cloud computing, *Journal of Systems and Software* 108 (2015) 60–76.
- [67] X. Yao, Z. Chen, Y. Tian, A lightweight attribute-based encryption scheme for the internet of things, *Future Generation Computer Systems* 49 (2015)

- 104–112.
- [68] N. S. Kumar, G. R. Lakshmi, B. Balamurugan, Enhanced attribute based encryption for cloud computing, *Procedia Computer Science* 46 (2015) 689–696.
- [69] H. Deng, Q. Wu, B. Qin, J. Domingo-Ferrer, L. Zhang, J. Liu, W. Shi, Ciphertext-policy hierarchical attribute-based encryption with short ciphertexts, *Information Sciences* 275 (2014) 370–384.
- [70] G. Klyne, J. J. Carroll, B. McBride, Resource description framework (rdf): Concepts and abstract syntax, *W3C recommendation* 10 (2004).
- [71] A. Gill, T. Bull, A. Farmer, G. Kimmell, E. Komp, Types and type families for hardware simulation and synthesis: The internals and externals of kansas lava, in: *Proceedings of the 11th International Conference on Trends in Functional Programming, TFP'10*, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 118–133.
- [72] C. Baaij, M. Kooijman, J. Kuper, A. Boeijink, M. Gerards, Clash: Structural descriptions of synchronous hardware using haskell, in: *Digital System Design: Architectures, Methods and Tools (DSD)*, 2010 13th Euromicro Conference on, pp. 714–721.
- [73] J. Kuper, C. Baaij, M. Kooijman, M. Gerards, Exercises in architecture specification using c #x03Bb;ash, in: *Specification Design Languages (FDL 2010)*, 2010 Forum on, pp. 1–6.
- [74] D. K. Bimson, D. R. Hull, D. Nieten, *The Lexical Bridge: A Methodology for Bridging the Semantic Gaps between a Natural Language and an Ontology*, Springer International Publishing, Cham, pp. 137–151.
- [75] B. D. Martino, G. Cretella, A. Esposito, A. Willner, A. Alloush, D. Bernstein, D. Vij, J. Weinman, Towards an ontology-based intercloud resource catalogue – the IEEE p2302 intercloud approach for a semantic resource exchange, in: *Proceedings of the 2015 IEEE International Conference on Cloud Engineering, IC2E '15*, IEEE Computer Society, Washington, DC, USA, 2015, pp. 458–464.
- [76] T. Tannenbaum, M. Litzkow, The condor distributed-processing system, *Dr Dobbs Journal* 20(2) (1995) 40–&.
- [77] J. Frey, T. Tannenbaum, M. Livny, I. Foster, S. Tuecke, Condor-g: A computation management agent for multi-institutional grids, *Cluster Computing* 5 (2002) 237–246.
- [78] D. Anderson, Boinc: A system for public-resource computing and storage, in: *Fifth Ieee/Acm International Workshop on Grid Computing*, pp. 4–10.
- [79] R. Subramaniyan, P. Raman, A. D. George, M. Radlinski, Gems: Gossip-enabled monitoring service for scalable heterogeneous distributed systems, *Cluster Computing* 9 (2006) 101–120.
- [80] X. Zhang, J. Schopf, Performance analysis of the globus toolkit monitoring and discovery service, mds2, in: *the 2004 Ieee International Performance, Computing, and Communications Conference*, pp. 843–849.
- [81] S. Zaniolas, R. Sakellariou, A taxonomy of grid monitoring systems, *Future Generation Computer Systems* 21(1) (2005) 163–188.
- [82] R. Devarakonda, et al., Mercury: reusable metadata management, data discovery and access system, *Earth Science Informatics* 3(1-2) (2010) 87–94.
- [83] A. C. Caminero, A. Robles-Gómez, S. Ros, R. Hernández, L. Tobarra, P2P-based resource discovery in dynamic grids allowing multi-attribute and range queries, *Parallel Computing* 39 (2013) 615–637.
- [84] Iamnitchi, I. T. Foster, On fully decentralized resource discovery in grid environments, in: *the Second International Workshop on Grid Computing (Grid'01)*, Springer-Verlag, London, UK, 2001, pp. 51–62.
- [85] J. A. Torkestani, A distributed resource discovery algorithm for {P2P} grids, *Journal of Network and Computer Applications* 35 (2012) 2028–2036.
- [86] P. Trunfio, D. Talia, H. Papadakis, P. Fragopoulou, M. Mordacchini, M. Pennanen, K. Popov, V. Vlassov, S. Haridi, Peer-to-peer resource discovery in grids: Models and systems, *Future Generation Computer Systems* 23 (2007) 864–878.
- [87] N. Drost, et al., Zorilla: a peer-to-peer middleware for real-world distributed systems, *Concurrency and Computation-Practice & Experience* 23(13) (2011) 1506–1521.
- [88] M. Reddy, et al., Vishwa: A reconfigurable P2P middleware for grid computations, in: *International Conference on Parallel Processing*, pp. 381–388.
- [89] A. I. T. Rowstron, P. Druschel, Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems, in: *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Middleware '01*, Springer-Verlag, London, UK, 2001, pp. 329–350.
- [90] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, J. Kubiawicz, Tapestry: a resilient global-scale overlay for service deployment, *Selected Areas in Communications, IEEE Journal on* 22 (2004) 41–53.
- [91] Y.-C. Wu, C.-M. Liu, J.-H. Wang, Enhancing the performance of locating data in chord-based P2P systems, in: *Parallel and Distributed Systems, 2008. ICPADS '08. 14th IEEE International Conference on*, pp. 841–846.
- [92] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, A scalable content-addressable network, *SIGCOMM Comput. Commun. Rev.* 31 (2001) 161–172.
- [93] A. Crespo, H. Garcia-Molina, Routing indices for peer-to-peer systems, in: *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pp. 23–32.
- [94] R. Brunner, A. C. Caminero, O. F. Rana, F. Freitag, L. Navarro, Network-aware summarisation for resource discovery in P2P-content networks, *Future Generation Computer Systems* 28 (2012) 563–572.
- [95] J. Lee, P. Keleher, A. Sussman, Decentralized multi-attribute range search for resource discovery and load balancing, *The Journal of Supercomputing* 68 (2014) 890–913.
- [96] M. Ripeanu, Peer-to-peer architecture case study: Gnutella network, in: *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*, pp. 99–100.
- [97] Z. Tari, G. Craske, A query propagation approach to improve corba trading service scalability, in: *Distributed Computing Systems, 2000. Proceedings. 20th International Conference on*, pp. 504–511.
- [98] R. Aguiar, D. Gomes, J. Barraca, N. Lau, Cloudthinking as an intelligent infrastructure for mobile robotics, *Wireless Personal Communications* 76 (2014) 231–244.