



City Research Online

City, University of London Institutional Repository

Citation: Zarrin, J., Aguiar, R. L. & Barraca, J. P. (2016). ElCore: Dynamic elastic resource management and discovery for future large-scale manycore enabled distributed systems. *Microprocessors and Microsystems*, 46(B), pp. 221-239. doi: 10.1016/j.micpro.2016.06.007

This is the accepted version of the paper.

This version of the publication may differ from the final published version.

Permanent repository link: <https://openaccess.city.ac.uk/id/eprint/18148/>

Link to published version: <https://doi.org/10.1016/j.micpro.2016.06.007>

Copyright: City Research Online aims to make research outputs of City, University of London available to a wider audience. Copyright and Moral Rights remain with the author(s) and/or copyright holders. URLs from City Research Online may be freely distributed and linked to.

Reuse: Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge. 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.

City Research Online:

<http://openaccess.city.ac.uk/>

publications@city.ac.uk

ElCore: Dynamic Elastic Resource Management and Discovery for Future Large-Scale Manycore Enabled Distributed Systems

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

Large-scale computing environments (such as HPC Clusters, Grids and Clouds) provide a vast number of heterogeneous resources (such as computing, storage, data and network resources) for the users/machines with various types of accessibility (in terms of resource, data, service and application). Resource management is one of the most significant underlying challenges for efficient resource sharing and utilization in such computing environments. Designing a resource management model which can be applied and adjusted to the requirements of these different future complex computing environments is an extra challenge. This paper will address the problem of resource management for the future large-scale manycore enabled computing environments by focusing on resource allocation issues. It provides a fully decentralized generic resource management architecture which can be applied to such distributed environments. Simulation results prove that our resource management scheme is highly scalable and provides a high level of accuracy for resource allocation which has a significant impact on the overall system performance.

Keywords: Resource Allocation, Manycore, Many-Chip, HPC, Cluster, Grid, Cloud Computing, Scheduling, Resource Discovery, Resource Utilization

1. Introduction

Modern large-scale distributed computing systems are undergoing with the rapid evolution of processor and network architectures. And they have made possible: i) the integration of more and more cores into one single chip; ii) many-chips being interconnected into a single machine; iii) more and more machines getting connected with highly increasing bandwidth. This leads to the emergence of the next generation of manycore enabled large-scale computing systems which rely on thousands of billions of heterogeneous processing cores connected to form a single computing unit.

Current large-scale computing environments such as HPC clusters (e.g., Infiniband-based distributed memory machines, Bewolf clusters), Grids and Clouds are common scenarios when discussing enhancements to overall computing/system performance and resource/data/service/application accessibility through efficient sharing and utilization of the integrated infrastructures and hardware resources (such as computing, storage, data and network resources) in large-scale systems with high heterogeneity (in terms of resources, applications, platforms, users, virtual organization, administration policies, etc.) and high dynamicity. In such large-scale computing environments, resource management is one of the most challenging, and complex issues for efficient resource sharing and utilization, particularly as we move toward Future ManyCore Systems (FMCS).

There are various types of techniques and methods to control and manage the infrastructure for each one of the aforementioned computing environments, which differ based on their main focus, embedded technologies and system architectures. And in

fact, designing a resource management architecture which can be applied and adjusted to the requirements of these different computing environments is an extra challenge.

In this paper, by “manycore enabled computing systems”, we mean future computing systems that support thousands of chips per compute node and thousands of heterogeneous cores per chip (as predicted in [1]). We address the problem of resource management for this future large-scale manycore enabled computing environment. In such large-scale systems (e.g., future multi-tenant Clouds, petascale manycores, and heterogeneous clusters) it is not feasible for the control system to centrally and statically have a complete and perfect knowledge of the entire system due to the magnitude and diversity of the amount of cores/processors and other resources. For uniformity of discussion, we will address all these control systems as distributed operation systems (DOS), although their realization may be quite different, depending on the specific large-scale computing technology under discussion. Furthermore, the variety of new emergent applications and the heterogeneous resources distributed across the network owned by different organizations and administrative domains, demand that these DOSs support an efficient mechanism for both process and resource management. Designing a DOS resource management system that meets the requirements of such large-scale environments is challenging due to several issues: a) Supporting adaptability, scalability and extensibility, b) Incorporating computing resources within different virtual organizations (VOs) while preserving the site autonomy (i.e., the distributed ownership of the resources), c) Co-allocating resources, d) Supporting low-cost computation, e) Supporting

Quality of Service (QoS) guarantees.

The main contribution of this paper is to propose EICore, a new elastic, scalable, accurate and dynamic resource management architecture for distributed systems in future large-scale manycore enabled computing environments, which supports high quality resource mapping and allocation (i.e., resource allocation accuracy) in a fully decentralized manner. In our work, we mainly focus on scalability and accuracy issues, since in our target computing environments (future large-scale manycore system), one of the most important requirements for resource management is the ability to deal with very large number of heterogeneous resources.

Using EICore, applications/processes can be distributed among a set of automatically self-organized and self-configured Resource Management Entities/Components (RMs or RMCs) in the system. RMCs can individually control and monitor their own collection of resources while resources can dynamically be traded (transacted) among RMCs on demand basis. This is achieved by the usage of an embedded dynamic resource discovery method which is able to efficiently discover the most accurate and appropriate resources with the lowest possible latency (i.e., nearest matching resources) among many available resources belonging to different RM entities. EICore provides scalability since its decentralized discovery mechanism allows scalable communications and resource trading between RMC instances. EICore discovery mechanism also supports a highly sophisticated query and resource description model (from the very low level of processors and processes to the level of computer clusters with respect to inter-resource and inter-process communication constraints) which enables the requesters to accurately discover the resources based on the required specifications and constraints for each given query. These features make EICore flexible to be efficiently adapted to the requirements of future large-scale manycore enabled systems, be the future HPC clusters, Grids or Clouds.

EICore also supports elasticity (roughly defined as the ability to address variable run-time changes on resources and workloads) with respect to the following two aspects. First, in our approach, we employ several resource management entities which are distributed across the system in a fully decentralized fashion. Each RMC instance can flexibly control and modify its own pool of resources according to on-demand basis. This gives elasticity to each instance of RMC to dynamically meet various, arbitrary scale, users' resource requirements, ranged from small applications to large computing-intensive tasks (we discuss EICore policy to trade resources among different instances of RMC in Section 4.2). Second, similar to the requirements of the current Cloud computing technology [2], we can expect that one of the most important requirement for application execution in future large-scale manycore enabled systems would be the ability of the system to deal with dynamic applications which have high variability fluctuations in their workloads. As an example of this type of applications we can mention malleable applications [3] where malleability is defined as the capability of the application to dynamically modify its data size and also number of computational entities (threads). The dynamicity of applications (in term of dynamic workload) affects the performance of resource

management systems, since resource mapping is based on initial QoS requirements of applications, when workload changes (during run-time), the application execution might fails to meet its initial QoS requirements. EICore provides elasticity to dynamically detect overloaded resources and reallocate additional resources, allowing migration of processes (like new threads) from overloaded to newly assigned resources (see Section 5).

Furthermore, the modularity and Service-Oriented Architecture (SOA-based) design of EICore allow it to be coherently integrated and used within DOSs. In fact, we developed this work under the framework of the S[o]OS project, as an example of DOSs [4–10]. S[o]OS (Service-oriented Operating System) was a European project aiming to develop reference architecture concepts for future very large-scale, distributed infrastructures, considering current system trends.

In this paper, for better illustration and evaluation of our approach, we use the term “resource” to refer to “computational resource” and we discuss resource management mostly from the point of view of computational capacity. Nevertheless, our resource management model is fully generic, and applicable to other types of resources (e.g., storage and networking resources).

The remainder of paper is organized as follows: In Section II we describe the related works. Section III elaborates on some important requirements for designing a resource management model for future large-scale manycore systems. In Section IV we explain our proposed resource management architecture (DOS level discussion). In Section V, we discuss our approach with respect to FMCS. Section VI provides detail on our manycore simulation methods and presents the simulation results. Finally, Section VII presents our conclusion.

2. Related Work

We investigate resource management issues for DOS in petascale computing environments such as HPC, Grid and Cloud in the matter of future manycore architectures. High Performance Computing (HPC) is generally recognized as a role model for manycore systems due to the essential similarity of their architectures and hence the means of program development for both of them (HPC and manycore systems). At the same time, the manycore movement leads to a sudden boost of scale in high performance computing without essentially increasing the requirements or costs. Many HPC cluster systems employ a centralized resource allocation and management, where the jobs are being controlled centrally and statically by a single resource manager that has complete knowledge of the system. In fact, HPC clusters are generally based on a batch scheduler and resource management system which allows users to submit their jobs to a batch system where a central scheduler and resource manager entity that is hosted on the cluster-head makes decision (in terms of timing aspects and the physical location of resources) for the resource allocation to run a given job. A centralized resource manager generally manages a pool of resources and runs the jobs based on a prioritization policy. However, it may distribute the jobs among a set of other resource manager entities in a hierarchical fashion. The resources might be organized in single or multiple queues (aka partitions) based

on some common characteristics of resources. Slurm [11, 12], TORQUE [13], LSF [14], PBS [15], Load Leveler [16], CCS [17] and Univa's Grid Engine (formerly Sun Grid Engine) [18] are some of the popular centralized schedulers and resource managers for the HPC clusters. HTCondor [19, 20] is another example of centralized resource management in High Throughput Computing (HTC) environments which is able to utilize large heterogeneous clusters where large numbers of relatively small-sized and short-live jobs are processed.

The future manycore enabled HPC clusters can not rely on the centralized and static resource management approaches. Resource management for large-scale manycore systems is inherently a NP-complete problem. The large number of heterogeneous cores and applications with various requirements causes scalability issues for centrally acting heuristics, where a centralized component must maintain a global view of the entire system. Resource management itself can become a bottleneck due to high computational requests and communication latencies. Furthermore, the amount of the detailed information on the resource characteristics which is maintained by the central resource managers would be limited to a very abstract level and this leads to reducing the accuracy of resource mapping and allocation (i.e., the quality of resource mapping) for the applications. On the other hand, managing dynamic resources (when resources join, leave or fail) with dynamic attributes and behaviors becomes more complex for the central resource managers since the solutions such as periodical or triggered updating, for the large number of resources and state changes, impose significant communication overheads on the system.

In decentralized resource management, multiple resource management entities cooperate to keep the workload for all computing resources in different pools or clusters balanced and satisfy the user requirements. REXEC [21], Tycoon [22] and Cluster-on-demand [23] are some examples of decentralized resource managers for HPC clusters.

In Grid computing where multiple geographically distributed and heterogeneous clusters are combined into a single system, according to the type of Grid, there exists different types of resource management system particularly designed for computing Grids such as Nimrod/G [24, 25], 2K [26] (on-demand hierarchical), Bond [27] (on-demand flat), Condor [28] (computational flat), Darwin [29] (multimedia hierarchical), Globus [30] (various hierarchical cell), Legion [31] and Ninf [32] (computational hierarchical).

Similar to HPC clusters, it is also common to use centralized resource management in Grid computing where users could submit their jobs to a single resource management or scheduling entity that makes decision on which cluster and nodes run the job. Grids also deploy decentralized approaches for managing resources in the system.

However, there are some major drawbacks with the current decentralized resource management schemes in HPC clusters and Grids. Most of these approaches do not consider the dynamicity of the applications (e.g., the application load might change during the run-time) and the resources (e.g., resources might join, leave or fail) in computing environments and they statically pre-assign a group of resources to each resource manager

(RM) entity in order to provide a load balanced solution to distribute jobs among different queues or RM entities in the system. This limits the dynamicity of resource management, since the resources are not allowed to be traded among different RM entities in the system (i.e., each resource is managed by a fixed RM entity). In other words, when a job is submitted to a RM, if the resources in RM resource-pool do not satisfy the query conditions the RM is not able to dynamically borrow resources from other RMs in the system and this results to either failing the query or forwarding the query to other resource manager entities in the system. Efficient decision making to forward queries among the distributed RMs leads to a resource discovery problem. Some of the decentralized RM approaches employ resource discovery techniques to manage query-forwarding between RMs, but these discovery mechanisms generally are not really dynamic and decentralized. For example Nimrod/G [24, 25] employs MDS resource discovery [33] which is based on publish/subscribe method to maintain resources in a set of hierarchical LDAP directories. MDS hierarchical discovery approach still suffers from single point failure mentioned for centralized methods because at each level there is a central data base server responsible for resource update requests. Moreover, MDS does not scale well in computing environments with frequent updates and large numbers of application requests.

In Cloud computing systems, resource provisioning using on-demand virtualization and VM migration technologies enable the data centers to consolidate their computing services and use minimal number of physical servers. In fact, the virtualization allows workloads to be deployed and scaled-out quickly through the rapid provisioning of Virtual Machines (VMs) on physical machines. But similar to HPC cluster and Grid, Cloud computing systems still need to deal with resource allocation and resource discovery problem to efficiently map the virtual machines to the the real physical resources.

An efficient resource management (in IaaS Cloud service model) can potentially provide significant number of benefits specifically in terms of quality of service, scalability, throughput and utility optimization, cost efficiency and overhead/latency reduction. However, for future diverse, large-scale systems, there are issues such as resource modeling (description), estimation (identification), provisioning, discovery, brokering, mapping, allocation and adaptation that it should yet deal with.

Implicitly, the scale in Cloud systems must be regarded as horizontal where instances of data and code are replicated to increase availability. Whilst this is particularly useful for data and service hosts (eBay, Amazon, etc.), it does have little or no impact on scalable systems acting on vertical scale (i.e., where the actual application is split into multiple processes or threads that jointly contribute to the application's functionality). Recently there have been more and more references to the so-called HPC Cloud [34]. However these either focus on making small parallel machines (for instance, in the order of 8 cores in Amazon EC), or provide access to thousands of cores in an almost unconnected fashion, similar to Compute Grids, (like Plura Processing). In the current literature there are several types of resource management systems for Cloud computing. For instance we can mention OpenNebula [35], Eucalyptus [36],

In-VIGO [37] and Cluster-on-Demand [38].

We can summarize that most of the existing resource management systems do not support fully decentralized, dynamical, scaleable, elastic and high quality resource mapping and allocation particularly for future large-scale manycore enabled computing systems with respect to multiple aspects such as changes in resources and the application behaviors. Workloads (i.e., the loads for applications) might be changed over time, thus the resource allocation for a process/application might need to grow or shrink in size. And also a process migration mechanism may required to tolerate failing/overloading resources. These are paradigms that are beyond most of the current resource management approaches in large-scale distributed computing.

3. Requirements

This section provides some key aspects required to design an efficient resource management model with respect to the requirements of future large-scale manycore systems. It also must be taken into account that our proposed approach does not address all the shortcomings of prior works. Nevertheless, we address the following key aspects for future DOSs:

System Size: In the future, billions of devices may be connected to form a single computing system. This means that, in order to still be able to efficiently use such computing environments, we need to provide scalable approaches which can deal with so very large-scale systems. Thus, scalability becomes one of the most important requirements for resource management for future computing systems [39, 40]. Along this line, decentralized models are generally the approaches that lead to maximum scalability [41], since centralized models commonly suffer from several inherent drawbacks (such as single point of failure and communication bottlenecks). On the other hand, implementing purely decentralized resource management models might hugely increase the level of system complexity (and in some cases it may not be feasible). Distributed hierarchies (which used for our work) can be considered as a realistic alternative decentralized model for resource management, since it can provide scalability while avoiding to overly increase the system complexity.

Heterogeneity: For future massively parallel, distributed and heterogeneous systems, capability of resource management approaches to efficiently handle high heterogeneity of resources is an essential requirement [42]. High heterogeneity of resources (i.e., core diversity) results in increasing the complexity of organizing and managing the resources. For example, different resources, positioned in the same chip or node, may provide different types and levels of capabilities which makes it difficult to apply a common resource management policy to efficiently explore all those heterogeneous resources. In order to solve this problem, we introduce the concept of virtual node (vnode) where each group of homogeneous resources, positioned in a common vicinity, participates to form a single vnode based on given vnode clustering parameters (we discuss this further in Section 5). In addition, we must note that using a shared memory programming or communication model is not feasible for heterogeneous cores, as well as affecting the scalability of the

system. Thus, we assume message passing for communication between cores and chips.

Mapping: For future scenarios, with very large numbers of diverse applications and resources, high quality process-resource mapping can be identified as one of the most urgent problems to be solved [43–45]. The quality of mapping is directly relevant to the accuracy of both resource identification (i.e., precise identification of required resources for each given process) and resource allocation (precise process-resource mapping to meet process requirements). In our work, assuming the former can be dynamically performed by process managers (we discuss this further in Section 6), we focus on the later. In order to achieve high resource allocation accuracy, we provide a matching strategy which uses a decentralized strategy (based on distributed hierarchies) to find and map resources. Our mapping strategy supports querying for both design-time (static workload) and run-time (dynamic workload) application requirements through enabling dynamic resource discovery and facilitating process migrations (for overloaded resources). We further discuss our matching strategy in Section 5.

Elasticity: The elasticity can be defined as the capability of the resource management system to flexibly and sensitively behave on demand basis where the demand comes from resource manager users (requests for resources). Elasticity is one of the desired requirements for resource management in current Cloud computing systems [46–50]. It must also be considered as an important resource management requirement for future manycore systems, due to the trend of moving Cloud computing concepts to manycore systems. As we mentioned earlier, ElCore provides elasticity with respect to the amount of resources controlled by each resource management entity and the amount of resources assigned to each process.

4. The Proposed Architecture

We propose a novel generic resource management architecture for DOSs which is able to efficiently manage the allocation of resources in the large-scale computing environments with high rate of dynamicity and heterogeneity of resources. We assume a reliable (no message loss, duplication or corruption) and asynchronous message passing model, for inter-core and inter-chip communication in such environments. Our system covers all levels of resources, in all different levels of clustering.

4.1. Architecture Functional Description

We organize the distributed resources in the computing environment (containing a large number of interconnected resources in different levels) in a set of distributed hierarchies through leveraging a dynamic self-organization and self-configuration approach [51]. We define the entities of Main-Control (i.e., DOS main-kernel) and Nano-Control (i.e., DOS nano-kernel) which are providing maximal and minimal numbers of capabilities, functionalities and services in the system through running each kernel on multiple cores [4, 5, 10]. These control entities 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 (i.e., deploying the main-control instances in the hierarchies head-nodes and the nano-control instances in the leaf-nodes). Similar to Multitenant Clouds architecture, each (main/nano) control instance provides a dedicated share of the instance including a set of dynamic on demanded services to client machines/components in their region in the hierarchy. In fact, this approach could potentially enhance the multitenancy model for the resource management to work in the hierarchies and support dynamic service instantiation (although this paper does not highlight this aspect due to space constraints). Moreover, we classify control services as atomic-services (e.g., for the resource management service where every island of distributed resources in the system is managed by a single instance) and non-atomic services. For the non-atomic-services, for instance, we can consider the scheduling service: a local scheduler instance per tile / processor may be responsible for scheduling all code segments within that domain, but will itself be subject to a higher-level scheduler which triggers the respective code block as a whole. The atomic-services are the ones that should not be subdivided in terms of functionalities and workloads.

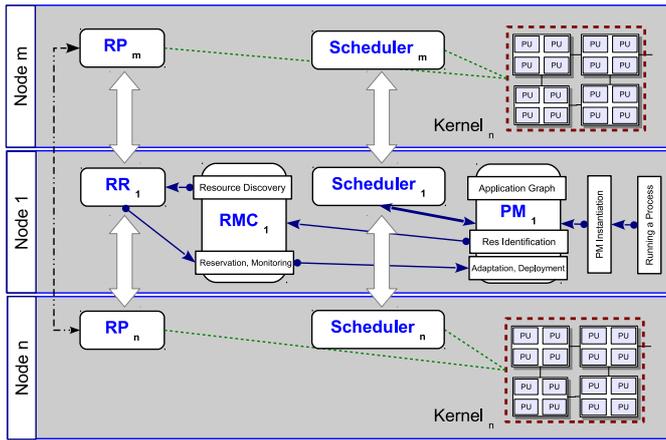


Figure 1: ElCore resource management architecture

Figure 1 highlights the key services associated to control decisions for resource management (aspects such as deployment instantiation on process movement are important but are not involved in the resource management decisions). These services include:

Resource Requester (RR): is a client-side discovery component which is responsible for generating, starting and disseminating a query in the system which finally returns information about the requested resources (e.g., available qualified processing cores and their respective capabilities). It is an atomic service that can be instantiated by any control entities. Each potential resource client in the system can also have a single RR instance.

Resource Provider (RP): or Resource Information Provider is a server-side discovery component which directly offers information services to Resource Requesters and communicates with other Resource Providers in the distributed system. RPs collaborate with each other to resolve the received queries from RRs in a completely decentralized and distributed fashion. It is an

atomic service which can only be instantiated by main-control instances. However, depending on the specific level of hierarchy, it can be configured as a non-atomic service considering a wider searching space. The RR-RP model is based on self-organizing processing resources in the system where the computing resources are organized into distributed hierarchies according to a hierarchical resource description (i.e., multi-layered resource description). Moreover, in each layer different algorithms and adapted mechanisms (such as distributed hash tables, distributed probability tables and any-casting) are implemented in order to redirect the discovery requests to the proper requested resources within layers (e.g., leaf-node, aggregate-node and super-node layers) in the system. More details of our RR-RP resource discovery model have been presented in [51] and [52].

Resource Management Component (RMC): is an atomic service which can only be instantiated by the main-control entities. Each RMC instance provides a global view of the available resources (i.e., existing capabilities) in the system for its clients while it offers accessibility mechanisms to access those resources and capabilities.

4.2. Resource Management Component

Since the resource management is the major focus of this paper, we discuss the RMC mechanisms in more detail. RMC manages, monitors and controls the resources in the distributed system. It keeps, monitors and modifies the status of the resources which are used by the local process managers. It invokes resource discovery components (i.e., RRs and RPs) to find the list of required resources according to the incoming request (wishlist) from Process Managers (PMs) or any other DOS components. It is also able to reserve a single resource or set of resources for a specific process/thread/threads, in order for the result of resource discovery (originated by other remote instances of RRs in the distributed system) not to contain the occupied resources. The reserved resources would be released upon receiving the notification (i.e., notification of process execution completion) from PMs. Additionally, RMC provides a set of capabilities to manage and monitor resources for the other DOS components like the Process Manager. In order to allocate the proper available resources to the application the Process Manager needs to query the Resource Manager.

RMC provides support for distributed management and control of resources. Every pool of resources is managed by one instance of RMC and each instance can provide a global view of all the potential resources in the network. It receives requests from many PM components while each request will specify a set of resource requirements (see Figure 2). For example a PM may ask for a minimum and maximum number of processors per each kind, for a certain fraction of the network bandwidth. RMC checks if the required amount of resources is available, and in case of positive answer it returns a set of resource descriptors that the PM caller module can use to run its tasks. The interaction between the PM and the RMC can be very simple (one simple request that can be accepted or rejected) or very complex (based on a negotiation protocol), depending on the specific DOS configuration.

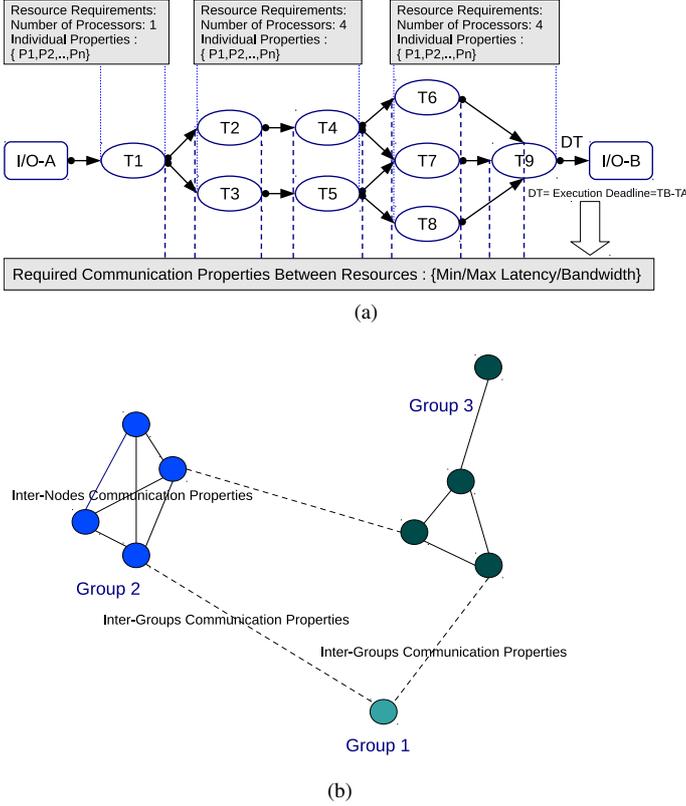


Figure 2: a) Extraction of the resource requirements from a given application graph, b) The resource graph of the qualified resources for a given query

Algorithm 1 illustrates the RMC resource requesting mechanism when it is invoked by a PM. Upon receiving a PM request, RMC tries to conduct all the requirements of the given request using all the resources it has under control. When it fails to handle a request using the available resources in its own administrative domain (i.e., the collection of distributed resources which are managed by an instance of RMC) RMC starts to extend its global view of resources by invoking a RR component (see Figure 3). Consequently RR negotiates with other remote RP instances in order to get information about the required resources which are in the free-ownership-list (i.e., a list of free resources managed by an RMC instance whose ownership is allowed to be transferred on request) of other RMC instances in the system. If those resources can be found in the free-ownership-list of other remote

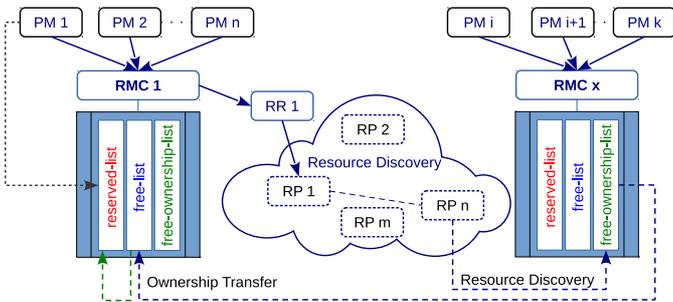


Figure 3: RMC mechanism

RMC domains, the local RMC would be able to add them to its own administrative domain.

We must note that the resources which are controlled by a single instance of RMC are not necessarily positioned in the local node. In fact there, each RMC instance has three main resource collections which include the reserved-list (or allocated resources), the free-list and the free-ownership-list. The reserved-list maintains the status of the resources which are allocated to the processes. The free-list keeps the information of the resources which are available for the upcoming processes. However, the other remote RMC instances do not have access to these resources. After releasing a resource, it will be moved from the reserved-list to the free-list. Generally, the resources in the free-list will be moved to the free-ownership-list after a certain period of time. Nevertheless, depending on the type of processes and their QoS requirements, RMC can decide to keep some particular resources in the free-list for a longer time. This could be an important feature particularly for deadline-sensitive applications. Examples for this situation would be real-time applications (such as online gaming, voice over IP), where processes must guarantee response within specified timeframes (i.e., deadlines). This means that for such processes, correctness of an operation depends not only upon its logical correctness, but also upon the time in which it is performed [53] (i.e., a process may fail if has not met its specific deadline). RMC policy flexibility may allow longer maintenance of resources, released by deadline-sensitive processes, in the free-list, for two reasons: first, RMC potentially gets similar requests from PMs for upcoming processes, so that, RMC would be able to respond immediately to those requests without a need to perform a global resource discovery. Second, by keeping a resource in the free-list, instead of free-ownership-list, the scope for resource contention for the upcoming similar, deadline-sensitive, requests would be limited only to local RMC processes, and not to all processes around the system. This increases the probability of getting immediate response for those requesters.

The free-ownership-list contains the resources that RMC will share with other remote RMC instances. Upon receiving a request for a resource, presented in the free-ownership-list, RMC transfers the authority to the requester (i.e., the remote RMC instance) to manage that resource. In fact, despite the free-list, the free-ownership-list contains the only available (free) resources which are allowed to be accessed globally in the system and their ownership can be transferred to the requesters on demand.

4.3. Resource Management System Operation

In the continuation of this section we elaborate on the inter-operations between the modules/services (which have been previously described in this section) in a general DOS environment and specifically for the discussed under the terminology for the project Service Oriented Operating System (S[o]OS) [4]. Figure 4 demonstrates general steps in configuration/inter-operation (i.e., in terms of modules, resources or threads) which contain all the major activities needed to be performed for resource management and allocation (i.e., before the actual execution of the code's processes). These configuration/inter-operation steps include the following: a) Analysis of the program and extracting

Algorithm 1: Pseudo code of mechanism for resource requesting in RMC when PM asks RMC to provide the resource descriptors for a set of required resources

```

Input: ProcId app, ResourceRequestList rl; /* process identifier and the
description of the required resources */
Output: ResIDs; /* res-IDs for the matched qualified available
resources */
ResID-Collection reserved-resources; ResId i;
ResID-Collection free-remote-resources; Resource res;
ResourceRequestList rr = rl;
ResourceRequestList request-remote-resources;
ResourceType rt;
foreach rt  $\in$  rr do
    while rt.requested-number > 0 do
        i = find-free-resource(rt.descriptor); /* searching in the local
free-list and free-ownership-list */
        if i == null then
            ; /* if the resources cant be found in the local
set, the RMC will ask RR to find them in the
system */
            request-remote-resources.add(rt);
            break ;
        else
            dep = analyze-dep-constraints(i, app, rl);
            if dep is not achieved then
                request-remote-resources.add(rt);
                break;
            else
                make-busy(app, i);
                reserved-resources.add(i);
                rt.requested-number--;
        endwhile
    endfor
    free-remote-resources = RR.FindResources(request-remote-resources);
    foreach res  $\in$  free-remote-resources do
        free-resources.add(res);
        make-busy(app, res);
        reserved-resources.add(res);
    endfor
    return reserved-resources;

```

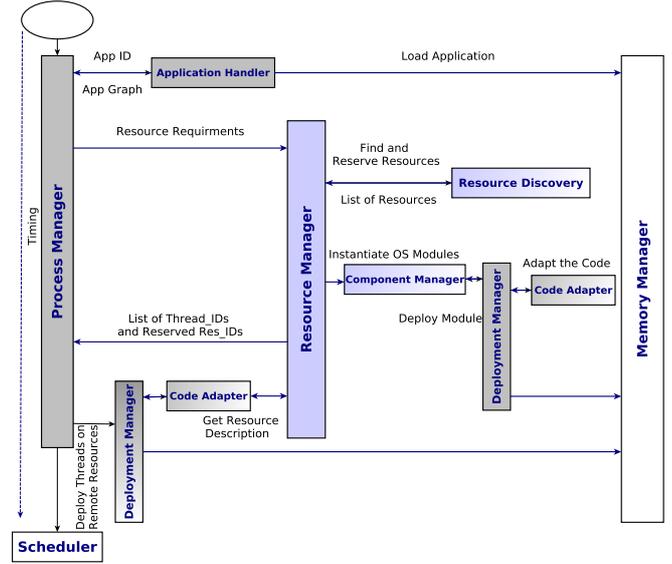


Figure 4: Interaction diagram among modules

the application's graph. b) Identifying the resources required for a given process (i.e., identifying the resources which maximize the matching between resource capabilities and process characteristics). c) Resource discovery, reservation, monitoring and management to find and reserve a matching graph of resources for a given application graph. d) Loading, configuring and instantiating all the necessary modules. e) Deploying the application segments in the reserved resources. f) Distributing code segments and their co-related reserved resources between the assigned sub-schedulers and starting to schedule the allocated threads on the reserved resources.

As we see in Figure 4, the PM employs the application handler to load the requested application. The application handler invokes memory manager (or stream manager) in order to load the application in memory and then it returns the application graph to the PM. In fact the code is analyzed with regard to its main dependencies and the resulting application graph can be created with respect to the specific hardware features that would suit the execution best. Due to the application graph, the PM subsequently creates the list of resource requirements, which consists of all the individual characteristics of the potential resource candidate for each vertex combined with the aggregated characteristics and the required communication behavior of all the edges in the application graph. The PM will also pass the minimum and maximum resource requirements (i.e., the individual characteristics of the required resource in terms of processor architecture, type of processor, speed, etc.) for each thread as

well as the minimum and maximum communication properties among threads in terms of latency and bandwidth (i.e., the inter threads communication requirements) to RMC. This means that RMC, regardless of the specific requirements of different applications, is able to discover and assign the required set of resources within given ranges (in terms of different computation or communication properties/attributes). In other words, RMC can be queried for resources satisfying arbitrary range conditions on different attributes since it benefits from a range-query enabled resource discovery mechanism. For example a PM may send a query to RMC with conditions such as the following: the required number of resources is 5, core clock rate for each resource must be greater than 1.5 Ghz, L1 cache size for each resource must be in the range of [128Kb, 1Mb] and also maximum communication latency between each pair of resources must be less than 10ms.

In current Grid and Cloud computing technologies, it is very common to specify the minimum and maximum computation requirements (i.e., desired computing attributes) for the resources requested by each query. Similarly, for many real applications, it is very important that the resource management and querying system can perform process-resource mapping with respect to minimum and maximum inter-resource/inter-process communication requirements. For example, for real-time applications, in order to meet the deadlines, we need to ensure that the maximum communication latency among the allocated resources does not exceed a given threshold.

RMC specifies the required resource graph and then it triggers the resource discovery module. Resource Discovery starts to locate resources based on the required computing and communicating conditions. It analyzes all the possible selection of resources and finally in order to fulfill all the requirements of the input query, it chooses the most efficient and appropriate graph of resources by considering the behavior of the communication routes between resources which follows the communication

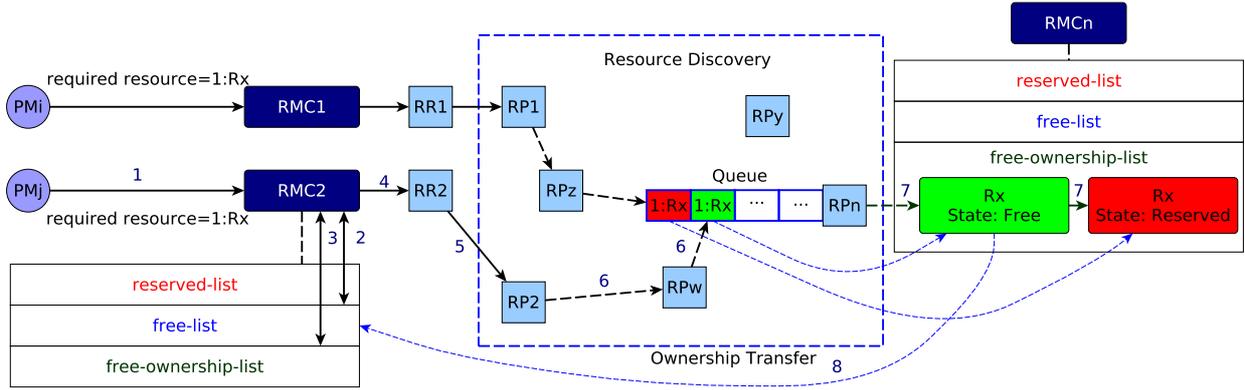


Figure 5: Example of resource contention occurs when two (or multiple) different RMC instances (like RMC1 and RMC2 on behalf of PMi and PMj) simultaneously attempt to obtain the same or overlapping set of rare resources (like Rx).

dependencies (i.e., data source and sink) as identified in the application graph during code analysis. We must note that during the discovery procedure, every discovered resource (within the free-ownership-list of a remote RMC) will initially be marked as “reserved” for a relevant thread in the application graph. This is the process which happens before transferring the ownership of the discovered resources to the local RMC (requester) and it results in a more immediate prevention of other remote instances of discovery module to find the resources that are locally marked as “reserved” (i.e., the discovered resources for a query will immediately be hidden from other queries in the system and if not used in a given time window will be again released). It also must be taken into account that, in our approach, the discovery results (set of discovered resources for a request/process) are deterministic. This means that RMC does not need to either reevaluate the mapping conditions for the discovered resources or select the best matches among a set of candidates (discovery results), rather it simply assigns the discovered resources for the corresponding process.

Figure 5 shows an example of resource contention between two RMC instances which require the same resource and initiated at the same time. As we can see in the figure, the resource discovery requests from both RMC1 and RMC2 suddenly and simultaneously arrive at the same destination resource provider (RPN). RPN orders the incoming requests in its own FIFO queue based on the time of arrival (requests that arrive at the same time slot are organized in a generally random order). RPN processes requests one by one. Accordingly, the request from RMC2 will be successful in discovering its desired resource (resource Rx). The state of resource Rx will immediately be changed to “reserved”, thus the next request in the queue (from RMC1) fails to discover the resource Rx. Afterwards, the ownership of Rx from RMCn will be transferred to RMC2.

In fact, once the destination resources for application threads are known, they will be reserved and prepared. All necessary DOS modules for maintaining communication, loading code, executing rendering, etc., will be loaded along with the actual code. For doing this, RMC will trigger the component manager in order to instantiate and setup all the necessary DOS modules.

The component manager consequently employs the deployment manager to deploy the respective modules. The deployment manager has the resource description of the target hardware platform (remote resource) and it can invoke code adapter to translate the code and finally it can write the respective adapted code in the target resource memory by invoking memory manager. In the next step, PM triggers deployment manager in order to deploy each of the thread codes in the allocated remote resources. In turn, deployment manager calls the local RMC to get the full resource description for each resource and afterwards it starts deploying each thread on the relevant remote resource platform. PM will finally pass a set of threads on a set of reserved resources to the scheduler component in order to begin the actual execution.

5. Resource Management for FMCS

We can envision that, in future, very large dimension many-core systems will be constructed by connecting a large number of computing nodes, with many chips and several thousands of heterogeneous cores per chip, using very high-speed network and interconnect technologies. In such FMCS, mapping applications to cores and adapting each application to the allocated cores plays a key role for an efficient utilization of the computation resources. Efficient resource mapping and allocation in systems with thousands of cores integrated per chip spans a large solution space and leads to the problem of optimal mapping of parallel applications to the cores which is known to be NP-complete [54].

Current approaches for resource management in manycore systems can be categorized to off-line, mixed and on-line approaches. The off-line approaches such as [55–57] and [58] are based on priori (static) knowledge about the whole system states and resources, thus they can not resolve the unpredicted dynamic situations particularly when the application workload dynamically varies at run-time. The mixed approaches such as [59, 60] and [61] are based on the pre-calculation of all potential application mapping and selection of the best mapping at run-time.

This also requires that all input applications and all possible combinations of mapping (for those applications) must be known at design-time. These approaches generally employ a central entity to select the best mapping amongst the pre-calculated solutions at run-time. But the application behavior and sequence of execution might be changed at run-time. Dynamic workload is not supported by mixed approaches. In online mapping (e.g., [62–64] and [65]), instead of pre-determination, the application mapping is dynamically decided at run-time. However we must note that online mapping trades off the resource allocation accuracy (i.e., the quality of resource mapping) versus computational complexity and scalability.

5.1. RM Strategy

Our proposed resource management scheme can be leveraged for FMCS to provide a highly decentralized, distributed, scalable and online application mapping with highly resource allocation accuracy. The system is initially configured as follows:

I1) In the first step, a virtual hierarchical overlay is created on top of the manycore system by dynamic self-organization of the vnodes (virtual nodes) over the system. Each vnode is defined as a group of homogeneous resources (i.e., processing cores) which are not necessarily positioned on a common physical chip (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 virtual-nodes by proximity and similarity). We must note that the resources within a vnode can be also bound to a single common chip. Furthermore, the number of resources within a vnode is bound by a pre-configured threshold value. Therefore, the vnodes with similar resources (i.e., sibling-nodes) might coexist within a single common chip. All vnodes in the system are connected to each other through an overlay topology which is a connected graph created based on the underlying network and interconnect topology during the self-organization phase.

We define two main parameters to construct every vnode which are η and λ . η is the maximum number of resources (cores) in the vnode and λ is the maximum distance between each pair of resources. The maximum distance can be also defined in terms of latency or number of hops.

Depending on the values of the aforementioned parameters, the vnode clustering can be obtained through various approaches. It can be simply achieved by specifying each single core ($\eta=1$, $\lambda=0$), or all the cores of a homogeneous processor (η =number of cores in the processor, λ =maximum latency among cores in the processor, $\lambda < \delta$ where δ is minimum latency for inter-chip communication) as a vnode. Another way is to simply specify every group of homogeneous cores, positioned in different processors (i.e., chip or die) of a network node, as a single vnode (η =total number of cores in the node, λ =maximum latency among cores in the node, $\lambda < \Delta$ where Δ is minimum latency for communication between cores of different nodes across the network). In fact a vnode is limited to include cores inside a single chip when $\lambda < \delta$ and it is extended to multiple chips when $\lambda \geq \delta$. Apart from that, various dynamic approaches can be also leveraged to perform vnode clustering. However, since vnode

clustering is out of scope for this paper we briefly describe a sample approach for dynamic vnode clustering.

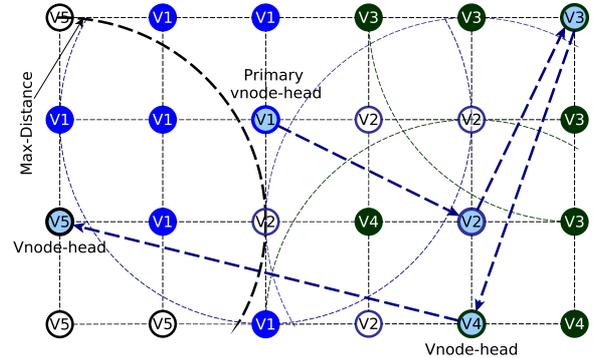


Figure 6: Example of vnode clustering for a processor containing 24 heterogeneous cores, connected through 2-D mesh topology

In this approach, for each multiprocessor node in the system, a random single resource (core) is triggered to initiate the clustering process. This resource, which is called primary vnode-head, would be responsible to organize the initial vnode for each physical node (multiprocessor node). The primary vnode-head broadcasts a vnode-clustering-request to all the cores in all the processors positioned in the current physical node and it will receive the description of each resource as well as the latency information from the cores. The primary vnode-head chooses a subset from the responding resources with respect to their similarity and vnode clustering parameters (i.e., η and λ). In this way, a candidate resource of c is selected based upon three conditions: firstly the resource description of c must be matched to the primary vnode-head. Secondly the latency between the primary vnode-head and c must be equal or less than λ and lastly by selecting c , the number of members for the primary vnode-head, should not exceed η . Upon establishing the first vnode, the primary vnode-head specifies the next vnode-head by randomly selecting a resource among the list of unsuccessful candidates. The next vnode-head will be triggered by receiving a message from the primary vnode-head, containing the list of unsuccessful candidates. The operation of the next vnode-head is similar to the primary vnode-head, but what makes it different is that instead of broadcasting to all cores, it uses multicast to send its vnode-clustering-request only to the resources which were unsuccessful to join the previous vnode-head. The overall clustering process will continue until the list of unsuccessful candidates for the current vnode-head becomes empty. Figure 6 demonstrates an example of using this approach for vnode-clustering in a single chip containing 4 different types of resources. It results in establishing 5 vnodes.

I2) In the next step, each vnode acquires its role (LN:leaf-node, AN:aggregate-node, SN:super-node) to play in the computing environment using a dynamic distributed election mechanism. ANs and SNs are the resource providers (RPs) where ANs provide Query Management Service (QMS) and SNs provide Super Query Management Service (SQMS). These services work as the system services interleaved with the applications on

the cores of each vnode with the AN or SN role. Alternatively, they can be also executed on the dedicated cores that are not available to the applications in each target vnode.

The vnodes which host QMS (i.e., RP-QMSs) are responsible to manage querying within a group of LN members and themselves. Similarly, the vnodes which host SQMS (i.e., RP-SQMSs) manage the querying within a group of AN members as well as their local resources. This mechanism divides the querying space to multiple layers such as $layer_{ln}$, $layer_{an}$ and $layer_{sn}$ where in each layer different adaptive query resolution and forwarding methods (such as DHT-based, Probability-based and Anycast-based approaches) are applied to direct and resolve the queries within or across layers. We must note that a single LN/AN is not allowed to participate simultaneously in more than one LN/AN group. Moreover, each compute node at least hosts a single RP-SQMS while every chip on a compute node hosts a single or multiple RP-QMS. Depending on the system configuration, the resources within a vnode or multiple vnodes initially managed by a single instance of RMC which cooperate with a RR entity. In other words, each RMC instance initially assigns a set of resources to its local free-list.

5.2. Application and Resource Description Aspects

We describe resources (i.e., computing resources) in the system based on their specific attributes (i.e., computation and communication characteristics) in a multi-layer hierarchy. The depth of the 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 the architecture designing aspects. In this paper, we assume that our description model contains three levels (layers) of the hierarchy: Core Layer or Inter-Core Level (i.e., $layer_{ln}$ which has attributes such as Data-PU channels, number of Instruction-PU channels, vector length, core clock rate, etc.), Die Layer or Inter-Chip Level (i.e., $layer_{an}$ which has attributes such as cache coherence, instruction set architecture (ISA), micro architecture, interconnection network, etc.) and Node Layer or Inter-Board Level (i.e., $layer_{sn}$ which has attributes such as window size, total number of cores, memory size, Die count, etc.).

The model is conducted by gathering and combining the individual attributes (ranging from more abstract information in the higher layers to more detailed characteristics in the lower layers) in each layer, augmented with information aggregation and summarization techniques, in order to create the layer-stamps. In fact, all specifications (i.e., attributes) of each layer as well as their values are represented by a single layer-stamp. We also use a similar description model to specify the desired resources by each single query as $\langle c|n_{ln}.c|n_{an}.c|n_{sn} \rangle$. This query-scheme identifies the status of the query conditions for each individual layer in the hierarchy. 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.

Application description (i.e., application performance modeling) is necessary to accurately specify the desired resources

as well as the minimum/maximum required inter-resource communication capacities for the efficient application execution. By using an appropriate application description model, PMs would be able to dynamically query for the right set of cores for their corresponding processes. Since the application description is out of scope for this paper, we assume that the PMs are independently able to initially extract the number and characteristics of required resources as well as the required inter-resource communication capacities for their corresponding processes at run-time and in an explicit manner. For example (see Figure 2), a PM for a process containing 9 threads with different resource and inter-thread communication requirements might ask RMC to allocate 3 different groups of homogeneous cores to fulfil both the inter-group and intra-group constraints for communication between allocated cores. We also assume applications/processes to consist of tasks/threads that are executed on single cores and that may freely be re-allocated to different cores.

5.3. Matching Strategy

When a RMC entity receives a resource request from a PM, if the demanded resources are not in the local free-list and free-ownership-list, RMC invokes a RR component to perform a resource discovery. RR in turn sends the given query to a RP-QMS where the RR host (i.e., a vnode which hosts the RR service) is a member of its LN group. Due to the type of query and query demand (e.g., simple single resource, multiple heterogeneous resources, complex resource graph containing constraints for inter resource communications) RP-QMS splits the main-query to a set of sub-queries and chooses individually the appropriate layer for each sub-query to start being processed. The query processing in $layer_{an}$ is based on distributed probability tables that facilitate dynamic distributed learning processes which are adapted to the progressive environmental changes. In $layer_{ln}$, LN members participate in a specification-based DHT ring where the sibling nodes are linearly organized in linked lists with single entries on the DHT ring and the vnodes are positioned in DHT based on their core-level specification stamp. The query processing and forwarding in $layer_{sn}$ also leverages the specification of the resources in the node-level to conduct a specification based anycasting method to direct the queries among SNs.

The matching strategy of the resource discovery mechanism is based on matching the given query descriptions for each layer (desired attribute conditions for each layer) to their corresponding layer-stamps, at the time of visiting every individual vnode in different hierarchies. As we mentioned earlier, each hierarchy consists of three layers and every vnode depending on its role in the hierarchy (LN, AN or SN) provides layer-stamps. The exploration process for every sub-query (i.e., a sub-request, aiming to discover a set of homogeneous resources within a common vicinity) starts from the nearest, potentially qualified and available, resources to the far resources with respect to giving matching priority from the highest layer to the lowest layer ($layer_{sn}$ to $layer_{ln}$). This way guaranties that the nearest resources (to the origin requester) are discovered in advance. Each sub-query may include a number of required resources with similar specifications as well as inter-resource communication requirements (e.g., maximum communication latency among desired resources). For

each sub-query, in order to match the given communication requirements to the inter-resource communication links among the matched resources, we use the following approach:

1-Sub-query proceeds to explore vnodes/resources (from nearest resources to far resources), regardless of its communication conditions, focusing on computation requirements (query requirements for different layers).

2-Upon discovering the first match, sub-query sets it as the initial pivot (reference) for measuring inter-resource communications.

3-Sub-query continues to discover the rest of required resources and when it found the next match, it makes a decision, by first evaluating lower bound, and then evaluating upper bound of inter-resource communication requirements, to perform one of the actions listed: ignore the current matched resource, perform pivot switching or continue the normal search. Sub-query ignores the current matched resource and continues the discovery process, if the current matched resource fails to meet the lower bound requirement (minimum latency). If this doesn't happen, sub-query must next evaluate the upper bound condition (maximum latency). Accordingly, pivot switching might be required. Pivot switching is the operation for changing the pivot of a sub-query to the current matched resource (see Figure 7). By doing a pivot switch, sub-query ignores all previously matched resources and adds the current matched resource, as the first match, to its list of matched resources. Afterwards it proceeds to discover the rest of required resources. Pivot switching happens when the current matched resource fails to meet the upper bound requirement (maximum latency). If pivot switching doesn't happen, the discovery process proceeds normally, until all required resources are detected.

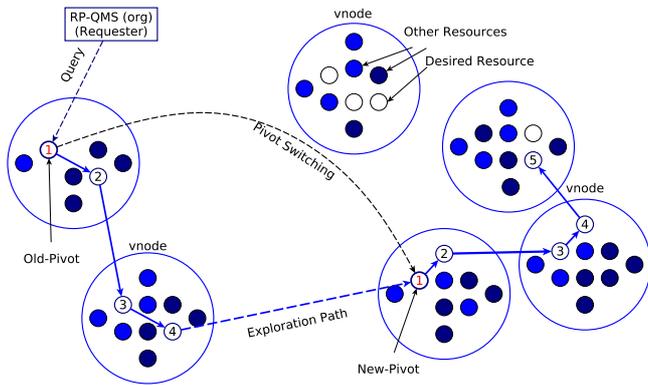


Figure 7: Example of matching strategy for inter-resource communication requirements using Pivot Switching mechanism. A sub-query, for 5 homogeneous resources with given maximum inter-resource latency, initiated from $RP-QMS_{org}$ and explores vnodes (from the closest vnodes to the most far one). The sub-query switches the pivot, when the recent matched resource fails to meet the upper bound communication requirement (maximum latency).

Finally, the RP-QMS that originated the sub-queries, aggregates the discovery results and replies the RR with a set of resource matches that optimally satisfy the main-query demand. After acquiring the discovery result from RR, RMC starts to perform a resource trading operation. It begins to negotiate

with other related RMCs in the system which maintain the discovered resources in their free-ownership-list and consequently transforms the ownership of those resources to its local free-list.

Each RMC instance periodically monitor the status of resources in its own resource pool through a HotSpot-detector component which is responsible to detect the overloading resources in the resource-pool. Whenever a HotSpot is detected, the RMC re-evaluate the resource requirements for the running application. Accordingly RMC allocates set of new resources for that application and start to migrate processes to the new resources. This mechanism provides elasticity for EICore to be able to tolerate the application with dynamic behaviors (aka malleable applications).

6. Simulation Results

In this section, we evaluate the performance of our proposed resource management scheme with respect to different evaluation criteria. 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 low cost and flexibility of the simulation to design, development and evaluation of the new algorithms as well as providing full control over system behavior and evaluation scenarios. Furthermore, the real infrastructures generally provide limitations to explore the design space particularly for scalable performance and large-scale evaluation.

In general, one of the traditional evaluation criteria, in order to evaluate resource management system is resource utilization which demonstrates how the resources in terms of computation and communication capabilities (e.g., CPU usage, Memory usage, Network Bandwidth, etc..) have been efficiently utilized in order to distribute and manage the workload. Resource utilization as an evaluation criteria has been used for many resource management systems specifically for virtualization based resource management systems or Cloud platforms available today.

In this paper, we decided to use the Resource Allocation Accuracy (RAA) as the main criteria to evaluate our work instead of Resource Utilization. RAA indicates how much a set of allocated resources for a process fulfills the original resource requirements emphasized by the PM requester.

In our resource management scheme we assumed that PM would dynamically be able to precisely identify the resource requirements for each process and consequently in the next level RMC would be responsible for finding and reserving the optimum set of qualified resources which will be allocated to the relevant threads by PM where the scheduler will be invoked to run the process accordingly.

We note that our assumption of dynamic identification of resource requirements for given applications/workloads (by PMs), may not be fully practical in current manycore systems (or implemented in current operating systems), but it is becoming a practice in nowadays Cloud computing technologies [66–77]. We argue that current Cloud computing systems are a potential ancestor of the future manycore systems (Intel 48-Core “Single-Chip Cloud Computer” [78] is an example of this trend).

Since the target computing environments of EICore is the future manycore systems, our assumption is reasonable; this is further supported by the concepts developed in the context of S[o]OS (which is an example of distributed operation systems, concerning the requirements of future manycore systems) [6]. In other words, S[o]OS is not for today, but we can see the ancestors of S[o]OS on current cloud systems, where the service demand (or the resources needed to run a job) is estimated prior to VM allocation and job execution [66–70].

Due to the above assumption, and the fact that the efficient mapping between the RMC-offered resources and the PM resource requests has significant impact on the whole system performance, we consider RAA as one of the most important performance criteria for managing resources in future large-scale manycore environments (with presence of high diversity of resources and applications) since it fully depends on the quality of resource management component, while resource utilization very much depends on the specific dynamicity of the requests being processed as well.

As it is shown in Equation 1, RAA for each query can be measured by calculating the ratio of the fully satisfied conditions to the total number of query conditions (in terms of the required computing attributes for each single resource and the desired inter-resource communication properties).

$$RAA = \frac{\gamma + \sum_i^g \left(\sum_j^{m_i} (\phi_j) + \tau_i \right)}{L + \sum_i^g (n_i \rho_i + l_i)} \times 100 \quad (1)$$

Here, ρ_i is the number of desired (computing) attributes for each requested resource in a sub-query for a group of homogeneous resources (i is the group index); n_i and m_i are respectively the number of requested resources and discovered resources for each group i ; l_i and τ_i are respectively the number of dependency links and qualified links for each group; L and γ are respectively the number of inter-group links (communication conditions) and qualified links for all groups; g is the number of resource-groups (number of sub-queries for the query) and ϕ_j is the number of qualified attributes for a discovered resource (j is the resource-index).

6.1. Simulation Setup

To do our evaluation, we developed a simulation platform, based on OMNET++ (similar to the way that is presented in [79]), which is able to simulate manycore environments (up to 55000 cores in different chips and nodes), focusing on communication aspects (i.e., communications between cores, chips and nodes). Using our simulator, we have simulated a manycore networked environment containing 2000 computing resources (i.e., processing cores) where there are 6 different types of processing resources. Each of these types have their own specific computation properties in terms of core clock rate, cache size, cach line, ALU properties and functionalities, memory bandwidth, etc. For instance, the processor frequency for resource type A, B, C, D, E and F are 2.53 GHz, 3.6 GHz, 1.6 GHz, 2.8 GHz, 1.2 GHz and 2.4 GHz accordingly. We conduct our simulation in a way to produce 20 percent of the resources as distributed homogeneous resources of a specific type-A which later will be required for several different processing scenarios. The reason for adjusting the amount of a certain resource-type in the system (as the target resources for the queries) is to facilitate measuring the value of Maximum Reachable Accuracy (MRA) in different settings of the experiment, but in fact it does not impact the generality of our evaluation because the resource management procedure is completely independent from the frequency of the target resources.

The type of resources, as well as their distribution in the system, is random. To do this, in the first step, we randomly order all processors (of different nodes) in a queue. We create an array of pairs (tx, ty) which contains all different two-combinations from the given set of types $S = \{A, B, \dots, F\}$. We iterate through the array, and for each iteration, we dequeue a processor from the queue and proceed to randomly specify the type of tx or ty for each of its cores. The iteration is repeated and the process continues until the queue becomes empty. As a result, each processor in the system, will consist of maximum two different types of cores (heterogeneous cores). In the next step, we control and regulate the desired amount of cores of a given specific type (type-A) through modifying the type of cores from type-A to ty

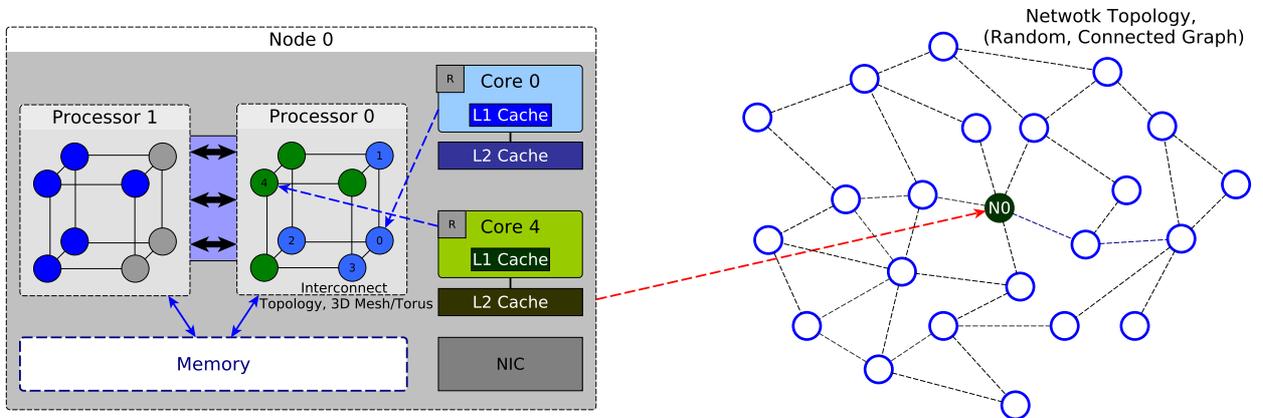


Figure 8: Manycore Simulation - System Architecture

Table 1: Simulation Parameters for Accuracy Evaluation

Parameter	Values	Description
Physical network size	2000 cores	Total number of resources in the system (i.e., total number of cores)
Number of network nodes	125 nodes	Total number of connected network nodes in the system
Number of tiles (CPUs) per node	2 processors	Number of processors per network node
Number of cores per CPU	8 cores	Each processor contains maximum two different types of cores
Interconnect topology	3-Dimensional Mesh/Torus	Topology among cores of each processor
Network topology	Random (connected graph, $\beta=62.5\%$)	Topology among network nodes, β indicates ratio of #nodes to #links in the network graph
Interconnect channel datarate	50Gbps	Inter-core communication
Routing type	DOR	Dimension Order Routing algorithm for routing among cores
Bus speed	20 Gbps	Inter-processor communication
Network channel	100 Mbps	Inter-node communication
Vnode clustering	$\eta=8, \lambda=100ns$	Clustering is limited to include cores inside multiple chips of a single common network node
Desired inter-resource latency	(0,150 μs], maximum 150 μs	The maximum inter-resource latency required for each query
Process Duration by sec	Weibull($\lambda=3.58, k=2.40$)	Execution time for each process through Weibull distribution, λ is the scale parameter and k is the shape parameter.

or from t_y to type-A, in the processors that contain cores with type-A ($t_x=type-A$).

Figure 8 shows the overall architecture of the simulated manycore system that we use for the evaluation in this paper. The cores of each processor are connected through a three-dimensional mesh interconnect topology (with datarate of 50Gbps) which, in our simulation scenario, is the same as three-dimensional torus. Each core has its own dedicated L1 and L2 cache that are not shared with any other cores. We 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 [80] where packets are routed to the correct position along higher dimensions (x or y) before being routed along lower dimensions (y or z). Furthermore, the processors of each node are connected through a high speed bus with datarate of 20 Gbps and we construct a network (with bandwidth of 100Mbps) based on a random network topology, a connected graph with $\beta = 62.5\%$ that indicates the ratio of the number of nodes to the number of links in the network graph.

We define 4 settings for the processing scenarios. Every setting is evaluated in 10 different run numbers. In each run the network topology parameters will be randomly changed according to our predefined simulation parameters (see Table 1).

In the first setting we assume that PMs will generate 1000 resource requests in a fixed interval (4000 ms) for different processes of the same type (i.e., in each interval, only a single request is generated in the entire system). The predefined process in this setting includes 2 threads that need 2 resources of type-A with specific communication requirements between those resources (maximum 150 μs for each dependency link). The other settings are similar to the first setting except for the type of processes. However, for the sake of simplicity in the evaluation, the type of requested resources in all of these settings are homogeneous (type-A). The differences between the processes in the aforementioned settings are the number of requested resources and also the level of dependency between the threads of the process in each setting. We also assume that the required resource for each thread is a single processing resource (i.e., a core). The required number of resources for each process in settings 2, 3 and 4 are 3, 5 and 9 while the threads dependency constraints are applied for 2, 4 and 8 communication links accordingly (see

Figure 9).

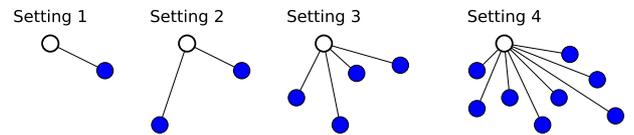


Figure 9: Dependency graph for different settings

In each run, after a query is finished, the matched resources will be allocated to the corresponding process. The allocated resources will be released after a period of execution time (termination of the process, see Table 1). In our simulation we assume that the released resources will be immediately transferred from the free-list to the free-ownership-list. For each resolved query, the simulator records information including query description, query results and the current status of the network (system) graph as a trace. Upon completion of simulation, the simulator individually analyses the generated traces for each run in order to calculate values of RAA and Maximum Reachable Accuracy (MRA) for each query. MRA is measured, similar to RAA (as it is presented in Equation 1). But, unlike RAA, MRA uses the parameters of the ideal query results, instead of the real query results achieved by our proposed resource management approach. To do this, the simulator statically evaluates the query conditions for each potential set of results in the network graph (provided by a trace) and concludes with the most optimal results attainable for the given query.

6.2. Resource Allocation Accuracy (Mapping Quality)

Figure 10 demonstrates the simulation results for each of the defined settings for different run numbers. The linear plots above the bar charts in each graph represent the variation of the value for MRA which is dependent on network topology and random distribution of resources.

As we mentioned earlier, MRA can be calculated by analyzing the states of resources and the network topology in each run number. In fact, it is possible that request requirements for a set of resources not be fully obtained in the system. This might happen when the requested set is not available (reserved), or never existed in the system. In such a case our approach tries

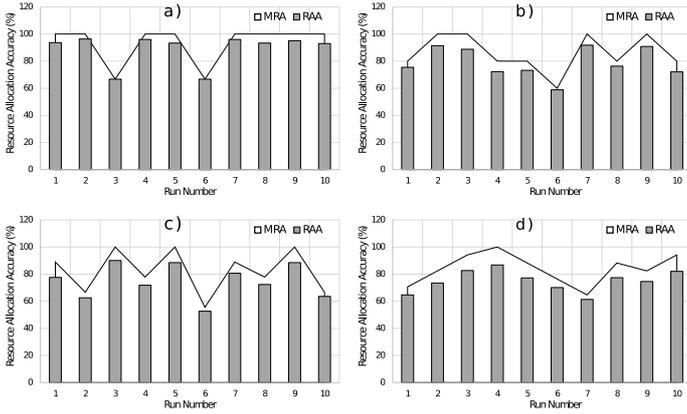


Figure 10: RAA ratio for different settings and network topology parameters: a) Setting 1: the processes with 2 threads b) Setting 2: the processes with 3 threads c) Setting 3: the processes with 5 threads d) Setting 4: the processes with 9 threads (MRA is the Maximum Reachable Accuracy, each run number indicates different random network topology parameters)

to offer a similar set which can fulfill most of the requirements. MRA specifies the border line to offer the most optimal set of resources for a specific process due to the realistic conditions of the resources in the system. The bar charts represent the average percentage of RAA with different settings and run numbers. As we can see in the results, the RAA in all of the tests are almost close to the value of the maximum reachable accuracy which presents a resalable level of accuracy for the resource allocation in the proposed resource management approach. The result shows that, the RAA in the settings for the small processes (i.e., the process with small number of threads) (Figure 10.a and Figure 10.b) in comparison to the larger processes (Figure 10.c and Figure 10.d) is closer to the maximum in overall.

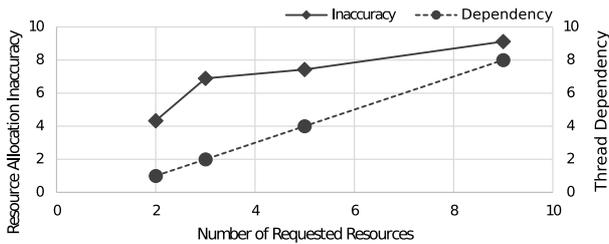


Figure 11: Resource Allocation Inaccuracy ratio for the processes with different number of threads and threads' dependency

Figure 11 shows the average inaccuracy ratio for different types of processes with various levels of dependency between their threads. It can be seen that inaccuracy ratio grows for the larger processes with higher level of threads dependency. However the inaccuracy slope is still less than the dependency slope. The reason for this behavior is that the larger processes with higher dependencies have more requirements for their required set of resources which can't be potentially fulfilled in the system in comparison to the smaller processes with lower dependencies. When the system is not able to provide the qualified set of resources considering all the process requirements, it returns the

optimal results that partially meet the requirements. And this increases the inaccuracy ratio. Our approach tries to provide the most qualified set of resources for allocation. However if it is not feasible to obtain, (for reasons such as unavailability of the resources and system limitations) it offers the best possible set of resources.

6.3. Scalability and Performance

In continuation of this section, we evaluate the performance of our resource management model with respect to scalability. To do this we conduct a simulation scenario based on the simulation parameters mentioned in Table 1 with the following changes, presented in Table 2.

Table 2: Simulation Parameters for Scalability Evaluation

Parameter	Values
Requested Resources for each Process	20
Frequency of Target Resources (FTR)	1650
Physical network size	27500 cores
Process Duration by sec	Weibull($\lambda=3.58, k=2.40$)
Querying Interval by ms	Exponential($\beta=4000$)
Consecutive Query Runs per RMC	1000
Number of Active RMCs	275

In fact, we simulate a distributed dynamic computing environment containing 27500 computing resources in which a constant number of RMCs (i.e., active RMCs) simultaneously handle the incoming resource requests from their PM clients. The time interval between each pair of consecutive queries issued by PMs (i.e., the arrival rate of incoming resource requests from PMs) is defined by Exponential distribution. We also assume that each active RMC only receives 1000 consecutive resource requests from PMs over the simulation time. RMCs assign and reserve the proper resources for each PM request. The reserved resources for each process be released after execution time period which is defined by Weibull distribution. We measure the system overhead caused by resource management components (i.e., number of transaction messages between RMCs, RRs and RPS to handle a resource request issued by a PM) as well as the RMC latency for requests over time.

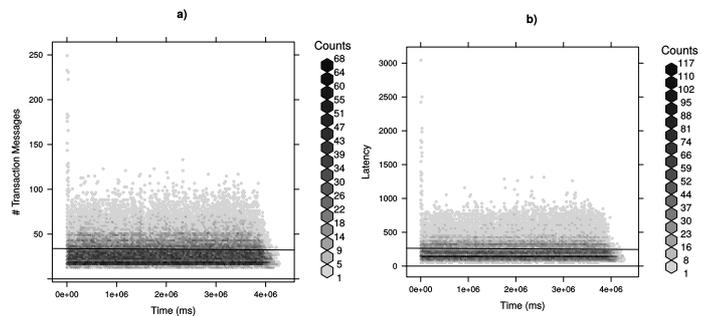


Figure 12: Hexbin plots for the number of transaction messages and the latency for each resource request over time.

Figure 12-a and Figure 12-b present the Hexbin plots for resource management overhead and latency over simulation time (by millisecond). Each data point in the graphs represents the

result of a single PM resource request. 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 we can see in these graphs, the majority of the queries results, particularly the high density data points, fall on or below the regression line. This illustrates that our resource management scheme 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.

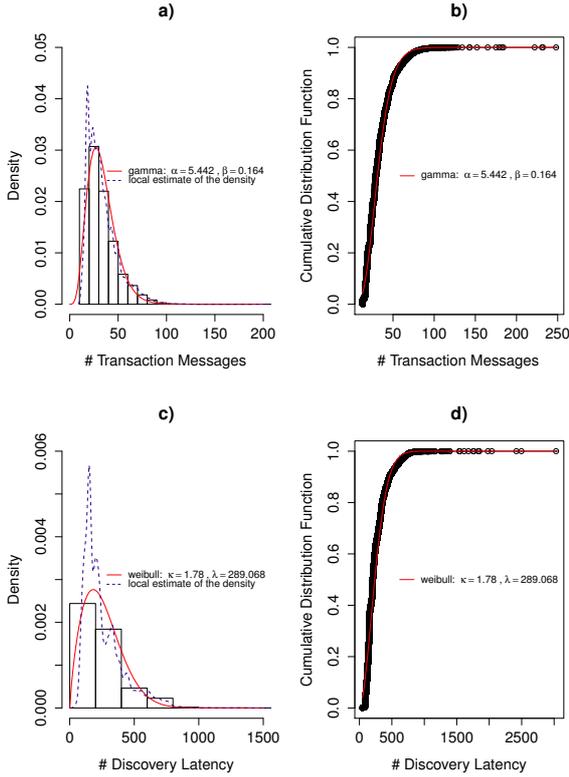


Figure 13: a and b) Histogram, density plot and CDF plot for the number of transaction messages between resource management components over simulation time, c and d) Histogram, density plot and CDF plot for the querying latency for the PM requesters over simulation time

Figure 13-a and Figure 13-b illustrate that at least 80% of the resource requests are managed by their correspondent RMCs with less than 50 transaction messages which shows a reasonable overhead. But for less than 20% of the resource requests, the resource management overhead is a value between 50 to 250 messages. This happens mainly due to resource unavailability, rare resources and resource contentions. In fact, for larger process 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 in the system, might become rare resources over time with higher cost of discovery. In addition, the rate of resource unavailability and the speed with which the resources become rare will accelerated particularly when the overall task duration (i.e., process execution time) exceeds the overall querying interval. Figure 13-c and Figure

13-d also present the similar results for the average querying latency to process the requests that were initiated by different PM components in the system over simulation time.

6.4. Scalability and Resource Discovery

In continuation of this section, we present the simulation results which demonstrate the performance of our resource management scheme with respect to scalability in comparison to other alternative approaches. The scalability of our resource management model is critically dependent on the deployment of a proper resource discovery mechanism that can discover resources managed by different independent distributed RMC components around the network. For comparisons, we simulate our model, EICore, in conjunction with three other approaches which benefit from different generic hybrid distributed methods for resource discovery. These approaches are PRW, FRW and Broad-Walk. We also simulated another version of EICore which employs a non-anycast based method for resource discovery. This model is identical to EICore except that it doesn't support SN layer and SQMS providers for resource discovery and instead it resolves any SN-dependent queries by extending the native probability mechanism of EICore to support the layer_{sn} resource information within layer_{an}.

In order to perform resource discovery, PRW (or partial random-walk), FRW (or full random-walk) and Broad-Walk are organized on top of two-layered (i.e., leaf-node layer and aggregate-node layer) distributed hierarchies. PRW and FRW leverage the same Chord based DHT method which is used in EICore in the leaf-node layer, however, they provide different query-forwarding methods in the aggregate-node layer. PRW uses both probability and random-walk methods 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_m and layer_{an}. PRW might behave similar to EICore-nAC for non-SN-dependent queries (e.g., $\langle C_{In}, C_{an}, n_{sn} \rangle$) 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 the given queries.

The PRW is comparable to our approach (EICore) in the sense that it creates clusters on top of the unstructured network. It also provides similarity with 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 [81–86]. For example in Iamnitchi et al [87] 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 also PRW are not based on publish/subscription since it is costly in terms of network traffic, processing, and storage needs for periodical updating and the maintenance of resource information particularly in high dynamic environments. On the other hand, our probability mechanism is comparative to or even better than learning-based strategy. 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 query previously [88]. This strategy becomes inefficient when the system size, dynamicity and heterogeneity of resources/-queries increase 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 PRW, FRW employs a fully single random-walk method to guide all types 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 [89] and to deal with the traffic/coverage trade-off. Random-walk is used in many distributed resource discovery applications such as Gnutella [90, 91], Iamnitchi et al [87] and [92–96].

Broad-Walk is a hybrid two layered approach which uses broadcast-based query propagation method [97, 98] 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.

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 environments containing various numbers 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, originating from a uniformly chosen source-node. Each experiment for each system-size is repeated for 100 run numbers 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 3 presents more details of simulation parameters for our evaluation.

In the first test, we perform experiments to measure average number of required messages and average latency (by milliseconds) per discovery request for EICore, EICore-nAC and other approaches. The PRW, FRW and Broad-Walk with the same topology, simulation parameters and conditions are used as al-

Table 3: 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
Rate of Requesters	1%

ternative reference works. Figs 14-a, b and 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 14-b demonstrates the results presented in the Figure 14-a with better resolution (without Broad-Walk).

In Figure 14-a, we observe that the average required number of discovery messages per query for Broad-Walk is much larger than the other approaches while in Figure 14-c the average latency of Broad-Walk is almost close to FRW, PRW and EICore-nAC. This means that Broad-Walk significantly generates more discovery traffic in comparison to others due to the heavy cost of broadcasting in $layer_{ln}$. The queries in Broad-Walk 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 it could also provide reasonable response time for queries (as seen in Figure 14-c) since the aggregate-node inquires all of its leaf-node members in parallel. The response time for Broad-Walk is approximately close to the results for FRW, PRW and even EICore-nAC.

Figs 14-b and 14-c show that our approach, EICore, 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 response time (latency) since the other approaches such as EICore-nAC and PRW also provide close results in terms of the number of discovery messages. EICore efficiently divides the exploring space to 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 the EICore-nAC, this strategy leads to a significant reduction in the response time of EICore while its discovery traffic slightly decreases. As we already discussed, EICore is the enhancement of EICore-nAC by leveraging our proposed anycast-forwarding mechanism in an extra layer which is called $layer_{sn}$. The presented results for EICore-nAC and EICore in Figs 14-b and 14-c also prove that increasing the level of hi-

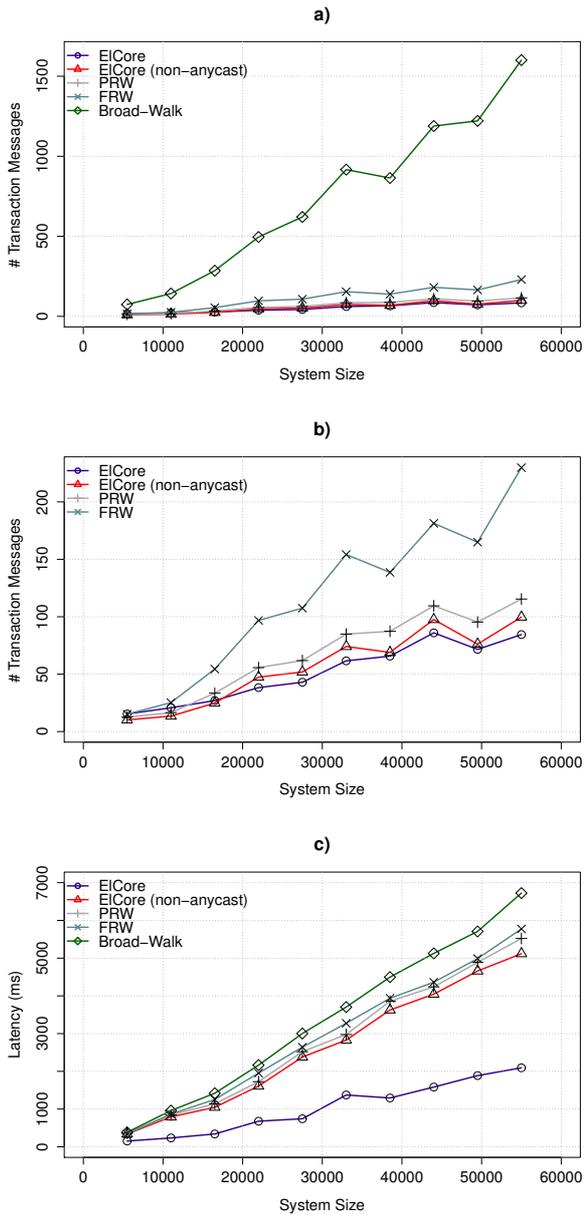


Figure 14: Comparison between EICore and other alternative approaches: Average number of transacted messages and latency per query for different system size and different approaches

erarchy along with the implementation of an efficient adaptive corresponding query-processing method improves the overall performance of our discovery system. Another factor contributing to EICore's overall performance is that EICore controls the discovery procedure in a more intelligent way which saves much unnecessary message cost. Moreover due to the anycast nature of EICore, 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 14-c).

From Figure 14-b, we can also see that, PRW generates larger number of discovery messages per query than EICore-nAC and EICore because of its partial random-walk query-forwarding

mechanism in $layer_{an}$. In fact, PRW provides an efficient probability mechanism (similar to EICore-nAC) 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 PRW 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 $c_{sn} \ll n_{sn}$). But for the other types of queries (e.g., $\langle C_{ln}, C_{an}, n_{sn} \rangle$) which are not considered in our evaluation in this section, PRW is expected to behave identically to EICore-nAC.

Figure 14-b illustrates that FRW provides a lower performance with respect to discovery overhead compared to PRW while they exhibit almost similar behavior for discovery latency as shown in Figure 14-c. This is due to the fact that the mechanism for query resolution in $layer_{an}$ of FRW 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 gets more compared to the approaches benefiting a type of estimation-based strategy.

In the next experiment, we measure the overall discovery load per node (i.e., $vnode$) and the average bandwidth consumption per node 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 (or control overhead per node) is the average number of transacted discovery messages by each $vnode$ during the querying period. Figs 15-a and 15-b depict the discovery load per node as the function of system size for different approaches. Figure 15-b demonstrates the results presented in Figure 15-a with better resolution (without Broad-Walk). As we can see in these figures, EICore shows better performance compared to other solutions by generating minimal traffic transmission control overhead per node. This also proves that EICore provides better scalability than others (in terms of discovery load per node for querying) as the system size increases.

Figure 15-c compares the average bandwidth consumption per node (during querying period) for different approaches and various system sizes. Bandwidth consumption per node indicates the amount of control traffic that each node generates during simulation time. The traffic load that we measured contains control, query and description messages that are transmitted between nodes. As we can expect, communication among RMC instances, to resolve various parallel resource requests, inevitably imposes unwanted traffic into the network. But, as we can see in Figure 15-c, the bandwidth consumption per node for our approach is significantly lower, compared to other approaches for all different system sizes. It can be also observed that, when we vary the system size from 5500 to 55000 resources, the bandwidth consumption for EICore remains reasonable (with minimum changes), resulting in better scalability, compared to

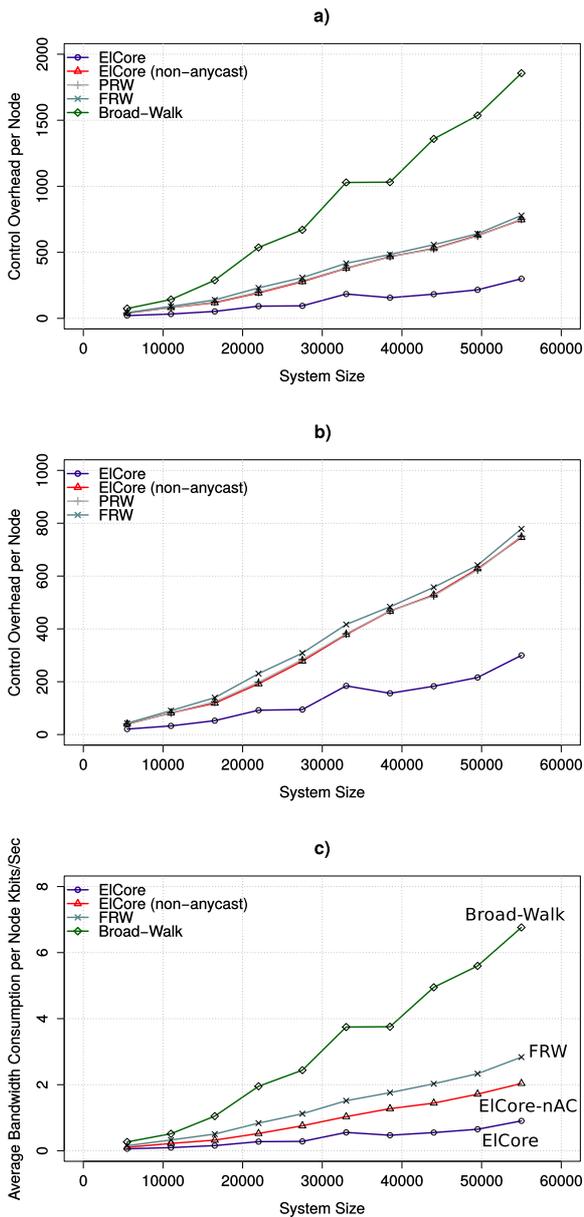


Figure 15: Comparison between EICore and other alternative approaches: a,b) Average control overhead per node (i.e., average number of transmitted messages per vnode) and c) Average bandwidth consumption per node, during simulation time (60000-80000 ms), for various system sizes.

other approaches.

7. Conclusion

We have provided a novel resource management architecture for large-scale distributed computing environments. The proposed architecture contains a set of modules which will dynamically be instantiated on the nodes in the distributed system on demand. Our approach is flexible to allocate the required set of resources for various types of processes/applications. It is a generic resource management approach which can be applied to different large-scale computing architectures and specifically it

can be explored in Cloud systems. We must note that we have not specifically designed our approach for Cloud computing only. Instead, the proposed resource management architecture provides a generic solution (considering the general requirements of large-scale computing environments) that can bring a set of interesting features (such as auto-scaling, multitenancy, multi-dimensional mapping, etc.,) which facilitate its easy adaptation to any distributed technology (such as SOA, Grid and HPC manycore). Our resource management scheme can support auto-scale (the ability to provide budget to more-or-less expensive tasks) in two different aspects: dynamic resource allocation and dynamic module instantiation. The simulation results show that, using our approach, the mapping between processes and resources can be done with high level of accuracy which will potentially lead to a significant enhancement in the overall system performance. System module instances will be automatically created when they are needed. The specificities of application of our model to a given architecture start here. Each module instance would be dependent on the type of application (e.g., real-time, HPC), the computing resource (where the application starts to be executed), heterogeneity of resources, positioning of the resources in the system, network topology and interconnection between resources. Moreover, leveraging discovery components (RR-RPs) enables our resource management platform to dynamically find and allocate available resources that guarantee the QoS parameters on demand.

8. Acknowledgment

The authors acknowledge the support of project FP7-ICT-2009.8.1, Grant Agreement No.248465, Service-oriented Operating Systems (2010-2013) [4–10] and of project Cloud Thinking (2013-2015), CENTRO-07-ST24-FEDER-002031 [99].

References

- [1] W. M. Arden, The international technology roadmap for semiconductors—perspectives and challenges for the next 15 years, *Current Opinion in Solid State and Materials Science* 6 (2002) 371–377.
- [2] I. K. Kim, J. Steele, Y. Qi, M. Humphrey, Comprehensive elastic resource management to ensure predictable performance for scientific applications on public iaas clouds, in: *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, UCC '14*, IEEE Computer Society, Washington, DC, USA, 2014, pp. 355–362.
- [3] T. Desell, K. E. Maghraoui, C. A. Varela, Malleable applications for scalable high performance computing, *Cluster Computing* 10 (2007) 323–337.
- [4] 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.
- [5] 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].
- [6] 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].

- [7] C. P. R. Baaij, J. Kuper, L. Schubert, SoOSIM: 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.
- [8] 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.
- [9] 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].
- [10] L. Schubert, A. Kipp, S. Wesner, Above the clouds: From grids to service-oriented operating systems., in: Future Internet Assembly, pp. 238–249.
- [11] A. B. Yoo, M. A. Jette, M. Grondona, Slurm: Simple linux utility for resource management, in: Job Scheduling Strategies for Parallel Processing, Springer, pp. 44–60.
- [12] A. B. Yoo, M. A. Jette, M. Grondona, Slurm: Simple linux utility for resource management, in: Job Scheduling Strategies for Parallel Processing, Springer, pp. 44–60.
- [13] G. Staples, Torque resource manager, in: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, SC '06, ACM, New York, NY, USA, 2006, p. 8.
- [14] S. Zhou, X. Zheng, J. Wang, P. Delisle, Utopia: A load sharing facility for large, heterogeneous distributed computer systems, *Softw. Pract. Exper.* 23 (1993) 1305–1336.
- [15] R. L. Henderson, Job scheduling under the portable batch system, in: Job scheduling strategies for parallel processing, Springer, pp. 279–294.
- [16] S. Kannan, M. Roberts, P. Mayes, D. Brelsford, J. F. Skovira, Workload management with loadleveler, IBM Redbooks 2 (2001) 2.
- [17] A. Keller, A. Reinefeld, Ccs resource management in networked hpc systems, in: Proceedings of the Seventh Heterogeneous Computing Workshop, HCW '98, IEEE Computer Society, Washington, DC, USA, 1998, pp. 44–.
- [18] W. G. S. Microsystems), Sun grid engine: Towards creating a compute power grid, in: Proceedings of the 1st International Symposium on Cluster Computing and the Grid, CCGRID '01, IEEE Computer Society, Washington, DC, USA, 2001, pp. 35–.
- [19] M. J. Litzkow, M. Livny, M. W. Mutka, Condor-a hunter of idle workstations, in: Distributed Computing Systems, 1988., 8th International Conference on, IEEE, pp. 104–111.
- [20] J. Basney, M. Livny, T. Tannenbaum, Deploying a high throughput computing cluster, *High performance cluster computing* 1 (1999) 356–361.
- [21] B. N. Chun, D. E. Culler, Market-based Proportional Resource Sharing for Clusters, Technical Report, University of California at Berkeley, Computer Science Division, Berkeley, CA, USA, 2000.
- [22] K. Lai, L. Rasmusson, E. Adar, L. Zhang, B. A. Huberman, Tycoon: An implementation of a distributed, market-based resource allocation system, *Multiagent Grid Syst.* 1 (2005) 169–182.
- [23] J. S. Chase, D. E. Irwin, L. E. Grit, J. D. Moore, S. E. Sprenkle, Dynamic virtual clusters in a grid site manager, in: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing, HPDC '03, IEEE Computer Society, Washington, DC, USA, 2003, pp. 90–.
- [24] D. Abramson, J. Giddy, L. Kotler, High performance parametric modeling with nimrod/g: Killer application for the global grid?, in: *ipdps*, IEEE, p. 520.
- [25] R. Buyya, D. Abramson, J. Giddy, Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid, in: High Performance Computing in the Asia-Pacific Region, 2000. Proceedings. The Fourth International Conference/Exhibition on, volume 1, IEEE, pp. 283–289.
- [26] F. Kon, R. H. Campbell, M. D. Mickunas, K. Nahrstedt, 2K: A Distributed Operating System for Dynamic Heterogeneous Environments, Technical Report, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 1999.
- [27] L. Bölöni, K. Jun, K. Palacz, R. Sion, D. C. Marinescu, The bond agent system and applications, in: Proceedings of the Second International Symposium on Agent Systems and Applications and Fourth International Symposium on Mobile Agents, ASA/MA 2000, Springer-Verlag, London, UK, UK, 2000, pp. 99–112.
- [28] M. Litzkow, M. Livny, M. Mutka, Condor-a hunter of idle workstations, in: Distributed Computing Systems, 1988., 8th International Conference on, pp. 104–111.
- [29] P. Chandra, Y. H. Chu, A. Fisher, J. Gao, C. Kosak, T. S. Ng, P. Steenkiste, E. Takahashi, H. Zhang, Darwin: Customizable resource management for value-added network services, *Netwrk. Mag. of Global Internetwkg.* 15 (2001) 22–35.
- [30] M. Ripeanu, M. Bowman, J. S. Chase, I. Foster, M. Milenkovic, Globus and planetlab resource management solutions compared, in: Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing, HPDC '04, IEEE Computer Society, Washington, DC, USA, 2004, pp. 246–255.
- [31] S. J. Chapin, D. Katramatos, J. F. Karpovich, A. S. Grimshaw, The legion resource management system, in: Proceedings of the Job Scheduling Strategies for Parallel Processing, IPPS/SPDP '99/JSSPP '99, Springer-Verlag, London, UK, UK, 1999, pp. 162–178.
- [32] H. Nakada, M. Sato, S. Sekiguchi, Design and implementations of ninf: Towards a global computing infrastructure, *Future Gener. Comput. Syst.* 15 (1999) 649–658.
- [33] A. Bradley, K. Curran, G. Parr, Discovering resources in computational grid environments, *The Journal of Supercomputing* 35 (2006) 27–49.
- [34] T. S. Somasundaram, K. Govindarajan, Cloudb: A framework for scheduling and managing high-performance computing (hpc) applications in science cloud, *Future Generation Computer Systems* 34 (2014) 47 – 65. Special Section: Distributed Solutions for Ubiquitous Computing and Ambient Intelligence.
- [35] J. A. Wickboldt, R. P. Esteves, M. B. de Carvalho, L. Z. Granville, Resource management in iaas cloud platforms made flexible through programmability, *Computer Networks* 68 (2014) 54 – 70. Communications and Networking in the Cloud.
- [36] D. Nurmi, R. Wolski, C. Grzegorzczak, G. Obertelli, S. Soman, L. Youseff, D. Zagorodnov, The eucalyptus open-source cloud-computing system, in: Cluster Computing and the Grid, 2009. CCGRID '09. 9th IEEE/ACM International Symposium on, pp. 124–131.
- [37] S. Adabala, V. Chadha, P. Chawla, R. Figueiredo, J. Fortes, I. Krsul, A. Matsunaga, M. Tsugawa, J. Zhang, M. Zhao, L. Zhu, X. Zhu, From virtualized resources to virtual computing grids: the in-vigo system, *Future Generation Computer Systems* 21 (2005) 896 – 909.
- [38] J. Chase, D. Irwin, L. Grit, J. Moore, S. Sprenkle, Dynamic virtual clusters in a grid site manager, in: High Performance Distributed Computing, 2003. Proceedings. 12th IEEE International Symposium on, pp. 90–100.
- [39] S. Kobbe, L. Bauer, D. Lohmann, W. Schröder-Preikschat, J. Henkel, Distrm: distributed resource management for on-chip many-core systems, in: Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, ACM, pp. 119–128.
- [40] B. Saha, A.-R. Adl-Tabatabai, A. Ghuloum, M. Rajagopalan, R. L. Hudson, L. Petersen, V. Menon, B. Murphy, T. Shpeisman, E. Sprangle, A. Rohillah, D. Carnean, J. Fang, Enabling scalability and performance in a large scale cmp environment, *SIGOPS Oper. Syst. Rev.* 41 (2007) 73–86.
- [41] J. Henkel, A. Herkersdorf, L. Bauer, T. Wild, M. Hübner, R. K. Pujari, A. Grudnitsky, J. Heisswolf, A. Zaib, B. Vogel, et al., Invasive manycore architectures., in: ASP-DAC, pp. 193–200.
- [42] A. Schüpbach, S. Peter, A. Baumann, T. Roscoe, P. Barham, T. Harris, R. Isaacs, Embracing diversity in the barrellish manycore operating system, in: Proceedings of the Workshop on Managed Many-Core Systems, Association for Computing Machinery, Inc., 2008, p. 27.
- [43] R. Marculescu, U. Y. Ogras, L. S. Peh, N. E. Jerger, Y. Hoskote, Outstanding research problems in noc design: System, microarchitecture, and circuit perspectives, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 28 (2009) 3–21.
- [44] P. Marwedel, J. Teich, G. Kouveli, I. Bacivarov, L. Thiele, S. Ha, C. Lee, Q. Xu, L. Huang, Mapping of applications to mpocs, in: Proceedings of the Seventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS '11, ACM, New York, NY, USA, 2011, pp. 109–118.
- [45] A. K. Singh, M. Shafique, A. Kumar, J. Henkel, Mapping on multi/many-core systems: Survey of current and emerging trends, in: Proceedings of the 50th Annual Design Automation Conference, DAC '13, ACM, New York, NY, USA, 2013, pp. 1:1–1:10.
- [46] H.-S. Wu, C.-J. Wang, J.-Y. Xie, Terascaler elb-an algorithm of prediction-based elastic load balancing resource management in cloud comput-

- ing, in: *Advanced Information Networking and Applications Workshops (WAINA)*, 2013 27th International Conference on, IEEE, pp. 649–654.
- [47] U. Sharma, Elastic resource management in cloud computing platforms, Ph.D. thesis, University of Massachusetts, Amherst, 2013.
- [48] M. Kesavan, A. Gavrilovska, K. Schwan, Elastic resource allocation in datacenters: Gremlins in the management plane, *ELASTIC 1* (2012) 8.
- [49] T. Erl, R. Puttini, Z. Mahmood, *Cloud computing: concepts, technology, & architecture*, Pearson Education, 2013.
- [50] I. K. Kim, J. Steele, Y. Qi, M. Humphrey, Comprehensive elastic resource management to ensure predictable performance for scientific applications on public iaas clouds, in: *Utility and Cloud Computing (UCC)*, 2014 IEEE/ACM 7th International Conference on, IEEE, pp. 355–362.
- [51] 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.
- [52] J. Zarrin, R. L. Aguiar, J. P. Barraca, A self-organizing and self-configuration algorithm for resource management in service-oriented systems, in: *Computers and Communication (ISCC)*, 2014 IEEE Symposium on, pp. 1–7.
- [53] K. G. Shin, P. Ramanathan, Real-time computing: a new discipline of computer science and engineering, *Proceedings of the IEEE* 82 (1994) 6–24.
- [54] E. G. C. Jr., M. R. Garey, D. S. Johnson, An application of bin-packing to multiprocessor scheduling., *SIAM J. Comput.* 7 (1978) 1–17.
- [55] Y. Jiang, X. Shen, J. Chen, R. Tripathi, Analysis and approximation of optimal co-scheduling on chip multiprocessors, in: *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, PACT '08*, ACM, New York, NY, USA, 2008, pp. 220–229.
- [56] D.-W. Kim, K.-H. Kim, W. Jang, F. F. Chen, Unrelated parallel machine scheduling with setup times using simulated annealing, *Robotics and Computer-Integrated Manufacturing* 18 (2002) 223 – 231. 11th International Conference on Flexible Automation and Intelligent Manufacturing.
- [57] J. E. Boillat, P. G. Kropf, A fast distributed mapping algorithm, in: *Proceedings of the Joint International Conference on Vector and Parallel Processing, CONPAR 90/VAPP IV*, Springer-Verlag, London, UK, UK, 1990, pp. 405–416.
- [58] C. Marcon, E. Moreno, N. Calazans, F. Moraes, Evaluation of algorithms for low energy mapping onto nocs, in: *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, pp. 389–392.
- [59] H. Hoffmann, M. Maggio, M. D. Santambrogio, A. Leva, A. Agarwal, SEEC: A framework for self-aware management of multicore resources, Technical Report, Massachusetts Institute of Technology, 2011.
- [60] H. Shojaei, A.-H. Ghamarian, T. Basten, M. Geilen, S. Stuijk, R. Hoes, A parameterized compositional multi-dimensional multiple-choice knapsack heuristic for cmp run-time management, in: *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, pp. 917–922.
- [61] L. Schor, I. Bacivarov, D. Rai, H. Yang, S.-H. Kang, L. Thiele, Scenario-based design flow for mapping streaming applications onto on-chip many-core systems, in: *Proceedings of the 2012 International Conference on Compilers, Architectures and Synthesis for Embedded Systems, CASES '12*, ACM, New York, NY, USA, 2012, pp. 71–80.
- [62] B. Yang, L. Guang, T. Säntti, J. Plosila, Mapping multiple applications with unbounded and bounded number of cores on many-core networks-on-chip, *Microprocessors and Microsystems* 37 (2013) 460 – 471.
- [63] I. Anagnostopoulos, V. Tsoutsouras, A. Bartzas, D. Soudris, Distributed run-time resource management for malleable applications on many-core platforms, in: *Proceedings of the 50th Annual Design Automation Conference, DAC '13*, ACM, New York, NY, USA, 2013, pp. 168:1–168:6.
- [64] F. Dong, S. G. Akl, Scheduling algorithms for grid computing: State of the art and open problems, Technical Report, Technical report, 2006.
- [65] G. Sabin, M. Lang, P. Sadayappan, Moldable parallel job scheduling using job efficiency: An iterative approach, in: E. Frachtenberg, U. Schwiegelshohn (Eds.), *Job Scheduling Strategies for Parallel Processing*, volume 4376 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2007, pp. 94–114.
- [66] J. F. Perez, G. Casale, S. Pacheco-Sanchez, Estimating computational requirements in multi-threaded applications, *Software Engineering, IEEE Transactions on* 41 (2015) 264–278.
- [67] W. Wang, G. Casale, Bayesian service demand estimation using gibbs sampling, in: *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2013 IEEE 21st International Symposium on, IEEE, pp. 567–576.
- [68] A. Kalbasi, D. Krishnamurthy, J. Rolia, S. Dawson, Dec: Service demand estimation with confidence, *Software Engineering, IEEE Transactions on* 38 (2012) 561–578.
- [69] D. Ardagna, G. Casale, M. Ciavotta, J. F. Pérez, W. Wang, Quality-of-service in cloud computing: modeling techniques and their applications, *Journal of Internet Services and Applications* 5 (2014) 1–17.
- [70] S. Spinner, G. Casale, F. Brosig, S. Kounev, Evaluating approaches to resource demand estimation, *Performance Evaluation* 92 (2015) 51–71.
- [71] W. Iqbal, Minimalistic adaptive resource management for multi-tier applications hosted on clouds, in: *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW)*, 2012 IEEE 26th International, IEEE, pp. 2546–2549.
- [72] J. Ortigoza, F. López-Pires, B. Barán, Workload trace generation for dynamic environments in cloud computing, *arXiv preprint arXiv:1507.00090* (2015).
- [73] A. Quiroz, H. Kim, M. Parashar, N. Gnanasambandam, N. Sharma, Towards autonomic workload provisioning for enterprise grids and clouds, in: *Grid Computing, 2009 10th IEEE/ACM International Conference on*, IEEE, pp. 50–57.
- [74] A. Khan, X. Yan, S. Tao, N. Anerousis, Workload characterization and prediction in the cloud: A multiple time series approach, in: *Network Operations and Management Symposium (NOMS)*, 2012 IEEE, IEEE, pp. 1287–1294.
- [75] G. Galante, L. C. E. de Bona, A survey on cloud computing elasticity, in: *Utility and Cloud Computing (UCC)*, 2012 IEEE Fifth International Conference on, IEEE, pp. 263–270.
- [76] J. Rao, Y. Wei, J. Gong, C.-Z. Xu, Qos guarantees and service differentiation for dynamic cloud applications, *Network and Service Management, IEEE Transactions on* 10 (2013) 43–55.
- [77] M. Andreolini, S. Casolari, M. Colajanni, M. Messori, Dynamic load management of virtual machines in cloud architectures, in: *Cloud Computing*, Springer, 2009, pp. 201–214.
- [78] S. C. C. Computing, Platform overview, Intel White paper 7 (2011).
- [79] 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.
- [80] J. M. Montanana, M. Koibuchi, H. Matsutani, H. Amano, Balanced dimension-order routing for k-ary n-cubes, in: *Parallel Processing Workshops, 2009. ICPPW '09. International Conference on*, pp. 499–506.
- [81] 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.
- [82] 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.
- [83] N. Bisnik, A. A. Abouzeid, Optimizing random walk search algorithms in {P2P} networks, *Computer Networks* 51 (2007) 1499 – 1514.
- [84] 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.
- [85] A. Sharma, S. Bawa, Comparative analysis of resource discovery approaches in grid computing, *Journal of Computers* 3 (2008).
- [86] 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.
- [87] 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–.
- [88] 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.
- [89] 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. Gorbach, M. Dane-

- lutto (Eds.), *Integrated Research in GRID Computing*, Springer US, 2007, pp. 103–118.
- [90] E. Pournaras, G. Exarchakos, N. Antonopoulos, Load-driven neighbourhood reconfiguration of gnutella overlay, *Computer Communications* 31 (2008) 3030 – 3039. Special Issue: Self-organization and self-management in communications as applied to autonomic networks.
- [91] 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.
- [92] 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.
- [93] 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.
- [94] 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.
- [95] 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.
- [96] 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.
- [97] 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.
- [98] 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.
- [99] R. Aguiar, D. Gomes, J. Barraca, N. Lau, Cloudthinking as an intelligent infrastructure for mobile robotics, *Wireless Personal Communications* 76 (2014) 231–244.