Co-funded by the European Commission within the Seventh Framework Programme

Project no. 318521

HARNESS

Specific Targeted Research Project
HARDWARE- AND NETWORK-ENHANCED SOFTWARE SYSTEMS FOR CLOUD COMPUTING

http://www.harness-project.eu/

Final Publishable Summary Report

Due date: 30 November 2015
Submission date: 22 February 2016
Resubmission date: N/A

Start date of project: 1 October 2012

Document type: Report
Activity: MGT
Work package: WP1

Coordinator: Prof. Alexander Wolf
Imperial College London
+44 207 594 8211
+44 207 594 8932
a.wolf@imperial.ac.uk

Contributing partners: all

Reviewers: Joint Steering Committee

Dissemination Level

PU	Public	√
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Revision history:

Version	Date	Authors	Institution	Description
0.1	2015/09/22	Alexander Wolf	IMP	Initial set up and overall content
1.0	2015/12/14	Alexander Wolf	IMP	Final edits

Tasks related to this deliverable:

Task No.	Task description	Partners involved°
T1.2	Interact with EC	IMP*

°This task list may not be equivalent to the list of partners contributing as authors to the deliverable
*Task leader

Executive Summary

Modern cloud computing technologies can vastly improve the flexibility and ease-of-use of application systems while simultaneously simplifying administration, reducing downtimes and maintenance costs. However, they do not meet the stringent requirements of several application domains, including scientific computing, business analytics and on-line machine learning.

Today, the dominant approach to constructing data centres is based on the assembly of large numbers of relatively inexpensive personal computers, interconnected by standard IP routers and supported by stock disk drives. This is consistent with the current business model for cloud computing, which leverages commodity computation, communication, and storage to provide low-cost application hosting.

To better serve workloads that are currently not supported by modern-day data centres, we created *Hardware- and Network-Enhanced Software Systems for Cloud Computing* (HARNESS), an enhanced cloud platform stack that fully embraces heterogeneity, allowing the side-by-side deployment of commodity and specialised resources, such as *field-programmable gate array*s (FPGAs), *general-purpose graphics processing unit*s (GPGPUs), network middleboxes, hybrid switches and *solid-state drive*s (SSDs), in data centre infrastructures.

With HARNESS, a flexible application may be deployed in many different ways over different types and numbers of resources, each option having its own cost, performance, and usage characteristics. In this context, applications express their needs to the HARNESS platform, as well as the price they are prepared to pay for various levels of service. This expression of needs and constraints builds upon what can be expressed through today?s simple counts of virtual machines or amounts of storage, to encompass the specific characteristic of specialised technologies.

These specialised technologies are virtualised into resources that can be managed and accessed at the platform level. The idea is to provide flexibility to the platform as to which, when, and how many resources are used, and to separate that concern from the low-level management of the concrete technology elements. Associated with the virtualised resources are policies that govern how allocation and optimisation decisions are made, as well as facilities to track their capacity, usage, and general availability.

Contents

Acronyms

AM *Application Manager*. 17–19, 25, 27, 29, 31, 33

API *application programming interface*. 9, 12, 29, 33

BSD *Berkeley software distribution*. 27

ConPaaS *Contrail platform-as-a-service*. 27

CPU *central processing unit*. 3, 7, 8, 10, 11, 17–19

CRS *Cross-Resource Scheduler*. 5, 25, 31, 32

DevOps *Development and Operations*. 28

DFE *dataflow engine*. 7, 8, 10, 11, 17–20, 22, 32, 36

DFG *data-flow graph*. 16

DIR *directory service*. 32, 33

DOI *Document Object Identifier*. 27

EPL *École Polytechnique Fédérale de Lausanne*. 37

FPGA *field-programmable gate array*. i, 1, 3, 5, 8, 10–12, 16, 31–33, 36

GPGPU *general-purpose graphics processing unit*. i, 1, 3–5, 8, 15, 31–33

HARNESS *Hardware- and Network-Enhanced Software Systems for Cloud Computing*. i, 2–10, 12, 14–19, 22, 25, 27–31, 33, 35–37

HDD *hard disk drive*. 17

HPC *high-performance computing*. 17

IaaS *infrastructure-as-a-service*. 1, 14, 36

IMDB *in-memory database*. 15

IMP *Imperial College London*. 7, 17, 37

IOPS *input/output operations per second*. 33

IRM *Infrastructure Resource Manager*. 31

MAX *Maxeler Techologies*. 15, 36, 37

MRC *metadata and replica catalog*. 32, 33

OLAP *on-line analytics processing*. 15

OSD *object storage device*. 32, 33

PaaS *platform-as-a-service*. 1, 4, 13, 36

PCM *phase-change memory*. 1

PDP *platform deployment package*. 13

QoS *quality of service*. 10, 27

RAM *random-access memory*. 18

RTM *reverse time migration*. 15, 17–20, 36

SAP *SAP AG*. 36, 37

SHEPARD *Scheduling for Heterogeneous Platforms using Application Resource Demands*. 7, 36

SLO *service-level objective*. 6, 18, 22, 25, 29, 31

SSD *solid-state drive*. i, 1, 7, 17

UR1 *Université de Rennes 1*. 27, 37

VM *virtual machine*. 25, 31–33

ZIB *Konrad-Zuse-Zentrum für Informationstechnik Berlin*. 27, 37

1 The Challenge

Cloud computing is reshaping IT, with increasingly large numbers of businesses, governments, and scientists seeking to offload mission-critical applications to third-party data centre providers. Today, the dominant approach to constructing data centres is based on the assembly of large numbers of relatively inexpensive personal computers, interconnected by standard IP routers and supported by stock disk drives. This is consistent with the current business model for cloud computing, which leverages commodity computation, communication, and storage to provide low-cost application hosting.

Taken together, the resources offered by cloud providers constitute a platform upon which to execute applications, commonly divided into a resource layer, *infrastructure-as-a-service* (IaaS), and an application layer, *platform-as-a-service* (PaaS). The efficacy of this platform relies upon the provider's ability to satisfy a broad range of application needs, while at the same time capitalising on infrastructure investments by making maximal use of the platform's resources. This *allocation and optimisation problem* exists whether the cloud data centre is operated for public use, as done by Amazon, Microsoft, and many smaller players, or for private use, as done by financial and government institutions. In trying to solve this problem, providers are naturally led to a platform design that abstracts and homogenises, which explains today's ubiquitous platform elements: virtualisation, simple key/value stores and distributed file systems, map/reduce dataflow computational structures, service-oriented architectures, and the like. The programming tools and middleware layers provided to application developers reflect and enforce these abstractions.

Despite continuing efforts to refine and improve this basic foundation through numerous research and innovation programmes, cloud data centre providers are realising that ***a fundamental limit has been reached in the ability of traditional application scaling alone to adequately achieve specific performance, availability, and cost requirements*** for many other, equally important applications. This limit is threatening the broader adoption of cloud computing, and thereby threatening the market for cloud services. Data centre providers must therefore rethink and reformulate the foundations of their computing platform, and look to radical new ways of satisfying application requirements.

The seed for a new approach can be found in the emergence of innovative ***hardware and network technologies***. These technologies span from computational devices, such as *field-programmable gate array*s (FPGAs) and *general-purpose graphics processing unit*s (GPGPUs), to new communication fabrics, such as programmable routers and switches, to new storage devices and approaches, such as *solid-state drive*s (SSDs) and *phase-change memory*s (PCMs). These technologies are already playing an essential role in substantially improving the performance, security, and cost profiles of many non-cloud-hosted, mission-critical applications.

Advanced hardware and network technologies for improving computation, communication, and storage have yet to be integrated into cloud computing platforms. The simple reason is that they break the abstract, homogeneous, and commodity nature of today's IaaS/PaaS computing model. This substantially increases the cost and complexity of application development, shifts some amount of responsibility for resource management from the cloud provider back onto the application operator, and significantly reduces the flexibility of the cloud provider to optimally manage multiple tenants.

We are thus led to the key research question at the core of the *Hardware- and Network-Enhanced Software Systems for Cloud Computing* (HARNESS) project: ***How can innovative hardware and network technologies be incorporated seamlessly into data centre clouds?*** The HARNESS project answered this challenging question through a carefully constructed and managed programme of research, validated by commercial and open source case studies.

2 The Project's Proposition

The goal of the HARNESS project was to advance the state of the art in cloud data centre design so that:

1. cloud providers can profitably offer and manage the tenancy of specialised hardware and network technologies, much as they do today's commodity resources; and

2. software developers can seamlessly, flexibly, and cost-effectively integrate specialised hardware and network technologies into the design and execution of their cloud-hosted applications.

To realise this goal, we enhanced the cloud software stack to not only incorporate a wide variety of specialised technologies, but to embrace the heterogeneity (of performance and of cost) that those technologies embody. In fact it is precisely their heterogeneity that makes them both appealing and challenging, since they offer a much richer, but also more complex, context in which to make price/performance trade offs, bringing wholly new degrees of freedom to the cloud resource allocation and optimisation problem. Further, those trade offs are inherently time dependent and time critical, with the dynamic goals and behaviour of the application, the user, the other cloud tenants, and the cloud provider all intersecting and sometimes conflicting through time.

In our vision, depicted in Figure 2.1, a demanding cloud application consists of a number of components, some of which (individually or as an assembly) can have multiple, alternative implementations that exhibit different resource demands, performance characteristics, and cost. Those components encapsulate the performance-critical parts of an application, such as a computational kernel for a scientific application, a filtering predicate for reducing communication traffic in a distributed overlay, or a database execution engine specialised for certain types of queries. Applications express their computing needs to the cloud platform, as well as the price they are prepared to pay for various levels of service. This expression of needs and constraints builds upon what can be expressed through today's simple counts of virtual machines or amounts of storage, to encompass the specific and varied new factors characteristic of specialised hardware and network technologies.

The HARNESS cloud platform has access to a variety of resources to which it can map the components of an application. Resources include standard virtual machines, commodity *central processing unit*s (CPUs), and stock disk stores, of course, but also specialised devices such as various types of high-end FPGAs and GPGPUs, network appliances, and advanced storage media. A flexible application may potentially be deployed in many different ways over these resources, each option having its own cost/performance/usage characteristics.

Our approach hypothesises three levels of concern. At the lowest level are the specialised technologies themselves. These are virtualised into the computation, communication, and storage resources available at the next level. Finally, the platform level provides a management and programming abstraction to applications. The idea is to provide flexibility to the platform as to which and how many resources within a category are used and when, and to separate that concern from the low-level deployment and monitoring of the concrete elements. Associated with the virtualised resources are policies that govern

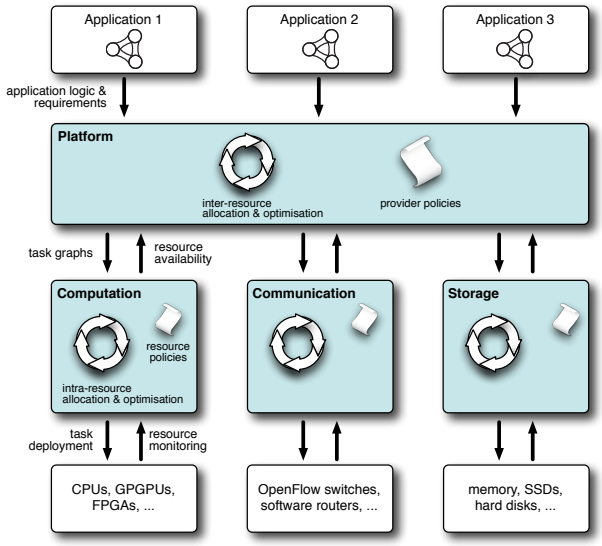


Figure 2.1: Overview of the HARNESS approach.

how allocation and optimisation decisions are made within each category. Also associated with these resources are facilities to track their capacity, usage, and general availability.

In general, the decisions at the platform level will result from choices that cross the conceptual boundary between the categories. In other words, a given deployment option will typically involve trading off different categories of resources, since the choice to map a particular component to a particular category of resource may imply a reduced set of choices for the placement of the other components with which it interacts. Moreover, in the context of multi-tenancy, those choices may also affect the choices available to other applications, which is precisely the allocation and optimisation problem faced by the cloud provider. Therefore, we assume that the provider also expresses their needs and constraints in terms of a set of policies.

Overall, the platform is charged with making informed decisions as to the optimal deployment that can simultaneously satisfy tenant applications (e.g., performance and price) and the cloud provider (e.g., profitability or energy efficiency). We see these deployment decisions taking place upon application start up or at well-defined points during execution (i.e., as a dynamic re-deployment or reconfiguration). The optimal deployment may thus change at run time based on dynamic criteria such as variations in the workload, changing performance expectations, and the spot price and availability of specialised computing devices.

Clearly, some of these very same concepts and problems have been explored since the beginning of the cloud revolution. But what makes the HARNESS project unique is the first-ever focus on accommodating and promoting the use of heterogeneous, specialised hardware and network technologies within a flexible and unified PaaS framework. We argue that instead of exposing heterogeneity at the bare device level, as is done today, for example, by Amazon with its GPGPU facility, tenants should be able to describe needs and constraints for their applications, including an indication by when they will require results to be delivered or how much they are willing to pay, so that the provider, based on the current state of the data centre, can decide how and when to allocate the appropriate mix of commodity and specialised

resources. We see this as a fundamental paradigm shift from the traditional *resource oriented* view of cloud computing to one that we characterise as *results oriented*. This shift has broad implications and far-reaching consequences for the design, development, and use of cloud data centres.

To realise the project goal, we set out to make progress on several difficult scientific and technical objectives:

1. **Define models for heterogeneous resources and application characterisation.** The first objective was a precise characterisation of the heterogeneous hardware resources being studied in HARNESS. These include FPGAs and GPGPUs, network switches and routers, and storage systems. The goal was to extract the key properties of these resources and build accurate performance models to be used during resource allocation and optimisation processes. Besides the resources themselves, there must also be appropriate mechanisms to express application requirements and to model application performance. Resource and application performance models have been subject to continuous refinement while putting together the HARNESS platform. For example, application performance modelling was sped up through resource monitoring and feedback and storage device modelling was extended by taking into account the fill-level of devices.
2. **Develop a management platform for heterogeneous clouds.** A key objective of HARNESS was to provide a software infrastructure to manage heterogeneous resources. This platform operates at both the level of individual resources and across resources. At the individual resource level, the HARNESS platform is responsible for executing applications (and tasks therein) on appropriately virtualised heterogeneous resources. At the cross-resource level, the platform manages all resources and mediates between different applications and resources.
3. **Design new programming interfaces and abstractions.** One of the main concrete benefits that we expect from HARNESS is to simplify the efficient use of heterogeneous resources in the cloud from the perspectives of cloud tenants and software developers. HARNESS shifts away from the traditional, resource-oriented cloud programming models in favour of results-oriented paradigms, in which tenants specify their application logic and requirements in high-level languages. One such language is the HARNESS *manifest* specification language Cloud providers satisfy these requirements by mapping applications to specific heterogeneous resources of the platform. This requires flexible and efficient programming interfaces and abstractions. On the one hand, they should be expressive enough to allow applications to achieve high performance. On the other, they must be flexible enough to permit cloud providers the opportunity to decide at run time which and how many resources to dedicate to a given application, potentially reallocating resources when operating conditions (or prices) change.
4. **Support run-time monitoring and fine-grained resource accounting.** To make admission decisions about the best mapping of application components to heterogeneous resources, the HARNESS platform monitors resource availability and observed application performance. Resource monitoring ensures that resources that deviate from their performance targets are identified early and replaced by others through task reallocation. Run-time monitoring also provides fine-grained accounting statistics on resource usage that can inform the pricing and billing models of the cloud provider. Resource monitoring and accounting is closely related to the management infrastructure for computation, communication, and storage resources; their interactions with the *Cross-Resource Scheduler* (CRS) forms a feedback loop that guides our scheduling algorithms toward optimal resource allocation decisions.

5. **Formulate algorithms for heterogeneous resource allocation and optimisation.** At the core of the HARNESS approach are algorithms that receive as input application requirements and characteristics, the current state of the heterogeneous resources annotated with their performance models, and the provider's internal policies and *service-level objective*s (SLOs), and then output the set of resources that should be assigned to an application.
6. **Create technologies for resource virtualisation and sharing.** Closely related to the objective of resource allocation and optimisation is the question of how to share heterogeneous resources efficiently. Resource sharing can be seen as a form of *intra-resource* allocation, as opposed to the *inter-resource* allocation described above. This objective involved the development of new approaches for the efficient and secure virtualisation of heterogeneous resources, including hardware accelerators, programmable network devices, and storage technologies. HARNESS addressed this objective along two dimensions: *scalability* and *performance isolation*.

3 Highlights of Achievements

We can summarise the main achievements of the project as the following:

1. The new-generation HARNESS cloud platform architecture is documented and demonstrated, including a prototype implementation capable of managing resources such as CPUs, *dataflow engine*s (DFEs), network middleboxes, and SSDs. The architecture is based on a novel layered management approach, designed to make cloud platform systems resilient to new forms of heterogeneity such that introducing new types of resources does not result in having to redesign the entire system. We have evaluated specific features of the HARNESS integrated platform using our validation use cases on both the *Imperial College London* (IMP) heterogeneous cluster and Grid'5000.

2. The HARNESS platform automates the process of selecting resources to satisfy application-specific goals (e.g., low completion time and/or low monetary cost), as specified by the cloud tenants. To do so, it performs automatic application profiling to build performance models. The platform takes advantage of the fact that application performance characteristics may be observed using smaller inputs, so employs extrapolated application profiling techniques based on feedback information provided by resource managers to reduce the time and cost of profiling.

3. The HARNESS infrastructure is managed by a collection of resource managers, each designed for a specific type of device incorporated into the HARNESS cloud. These managers deal with provisioning and making effective use of reserved resources on behalf of a cloud tenant. Management technologies include: (a) MaxelerOS Orchestrator for networked DFE reservations; (b) *Scheduling for Heterogeneous Platforms using Application Resource Demands* (SHEPARD) for hardware accelerator reservations; and (c) XtreemFS for heterogeneous storage reservations. A cross-resource scheduler enfolds all these resource-specific managers to optimise multi-tenant reservation requests alongside (optional) network placement constraints.

4. To maximise resource utilisation, new virtualisation technologies allow physical devices, some of them originally designed for only single tenants, to be effectively shared in a multi-tenant environment. Our current virtualisation technologies include: (a) a virtualised DFE, backed up by a group of physical DFEs that can grow or shrink depending on the workload; (b) an XtreemFS virtualised storage object that enables multiple reservations with different requirements; and (c) network virtualisation giving tenants the ability to deploy general network functions (e.g., Layer-2 switching, Layer-3 forwarding) and application-specific network functions (e.g., an HTTP load balancer).

5. Two new software engineering methodologies are introduced that allow application developers to make effective use of the HARNESS cloud. The first one uses aspect-oriented programming techniques to seamlessly enrich application code with expert programming tips and configurations for specific hardware architectures. The second focuses on stream-processing applications. Here, the idea is to sidestep the problem of "when" and "what" to offload a computational task on an accelerator by introducing a hybrid processing model for heterogeneous architectures, where tasks are executed opportunistically using all available heterogeneous processors.

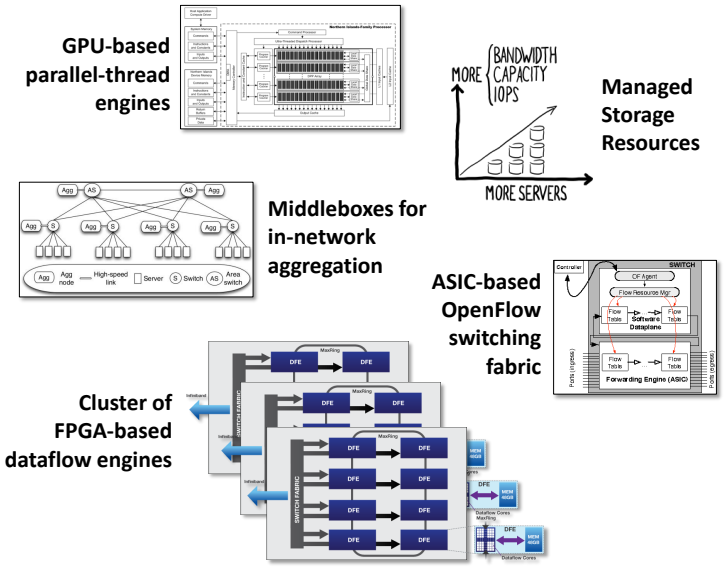


Figure 3.1: The HARNESS resource set consists of a group of compute, communication and storage resources. Some of these resources include (clockwise from top): (a) GPGPUs, (b) managed storage volumes with performance guarantees, (c) hybrid switches that can handle millions of access control rules, (c) a cluster of DFEs that can be accessed via network, and (d) general-purpose middleboxes that allow application-specific network functions to be performed along network paths.

3.1 Managing Heterogeneity

One of the problems addressed by the HARNESS project was how to seamlessly incorporate and manage different types of heterogeneous resources in the context of a cloud computing platform. This includes not only supporting resources targeted by the HARNESS consortium (Figure 3.1), but also generalising this approach beyond the HARNESS resource set. This means designing a cloud computing platform that is resilient to heterogeneity, such that supporting new types of resources does not require a complete redesign of the system.

There are two main challenges in handling heterogeneity in the context of a cloud computing platform. First, in contrast with commodity CPU-based servers, specialised resources are largely invisible to operating systems, while cloud infrastructure management software such as OpenStack provide very limited support. Heterogeneous compute resources such as GPGPUs and FPGAs are often accessed by applications as I/O devices via library-call and/or run-time interfaces. These devices must be directly managed by the application programmer, including not just execution of code but also in many cases tasks that are traditionally supported by both hardware and software, such as allocation, deallocation, load balancing, context switching and virtualisation. The second challenge relates to creating a system that does not require redesigning the architecture or rewriting the allocation algorithms when introducing new types of resources, considering that each type of resource exposes different control mechanisms, as well as interfaces for acquiring feedback about availability and monitoring. More importantly, each type of resource exhibits different semantics for expressing capacity and allocation requests.

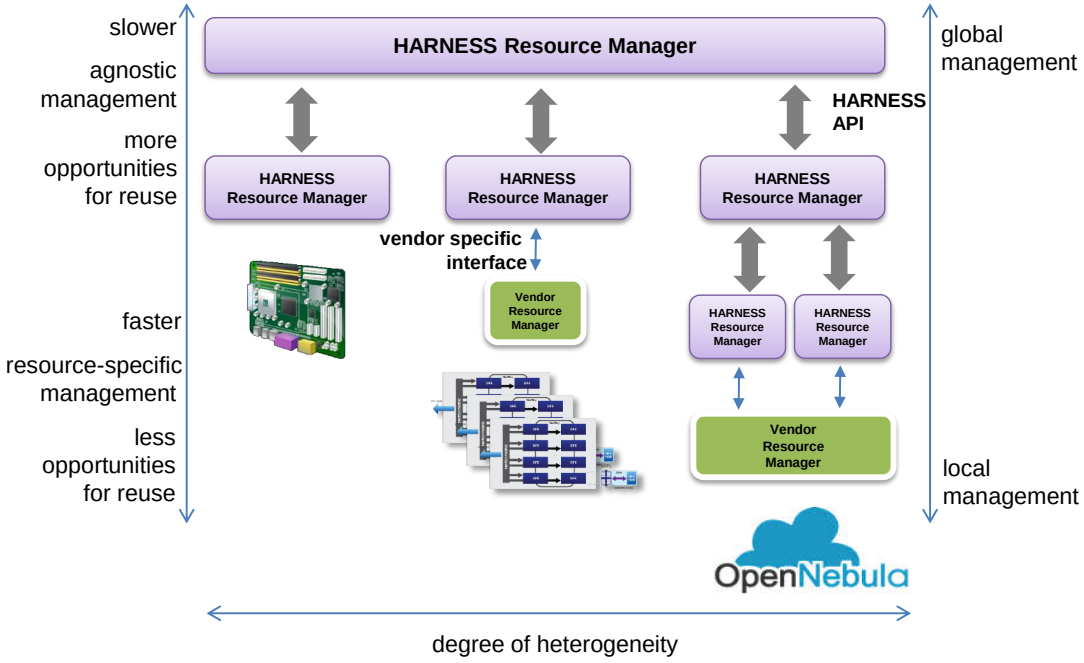


Figure 3.2: Heterogeneous cloud computing platforms, supporting various types of resources, can be built by composing resource managers that implement the HARNESS API in an hierarchical system.

3.2 Hierarchical resource management

To mitigate these problems, we have designed a cloud computing architecture where run-time resource managers can be combined hierarchically, as illustrated in Figure 3.2. In this organisation, the top levels of the hierarchy are the HARNESS resource managers, which service requests using the HARNESS API. At the lower levels of the hierarchy we find vendor resource managers that handle specific devices. One of the responsibilities of the HARNESS resource managers is to translate agnostic requests defined by the HARNESS *application programming interface* (API) into vendor-specific requests.

Supporting multiple hierarchical levels allows a system integrator to design an architecture with a separation of concerns, such that each manager can deal with specific types of resources. In particular, a typical HARNESS platform deployment has a top-level manager that has complete knowledge about all the resources that are available in the data centre, but very little understanding of how to deal with any of these resources specifically. Any management request (such as reservation or monitoring feedback) are delegated to child managers that have a more direct understanding of the resource(s) in question. A child manager can handle a resource type directly or can delegate the request further down to more specific resource managers. What we see is that at the top level of the hierarchy we have a more agnostic management that can handle a wide range of resource types, and thus have a globalised view of resources available in the data centre. As we go lower in the hierarchy, we have more localised and resource-specific management.

3.3 Agnostic resource scheduling

The scheduling research community has for some time been aware of, and interested in, the problem of allocating resources and scheduling tasks on large-scale parallel distributed systems with some degree of heterogeneity. While work in this area generally recognises the underlying variations in capabilities between classes of resources, and much has been made of the differences in how time- and space-shared resources (e.g., CPUs vs. memory) are partitioned between users and tasks, they usually assume some uniformity in semantics presented by the interfaces used to allocate these resources. For example, Stillwell et al. [21] assume that a multi-resource allocation can be mapped to a normalised euclidean vector, and that any combination of resource allocation requests can be reasonably serviced by a compute node as long as the vector sum of the allocations assigned to a node does not exceed the vector representing that node's capacity in any dimension.

What we observe, however, is that heterogeneous resources expose far more complicated semantics that cannot be captured by a simple single- or multi-dimensional availability metric, and cannot be normalised such that they can be directly understood by a central scheduling algorithm. For some devices, it may be appropriate to allow resources to present different levels of abstraction, or even to represent complex aggregations. Examples include: a (not necessarily homogeneous) collection of machines capable of running some mix of programs or virtual machines (e.g., a cluster); a distributed file service capable of guaranteeing some level of performance or reliability; a virtual network with embedded processing elements; a cluster of FPGA-based dataflow engines with a specific interconnect topology; or a set of network links with enforced bandwidth and latency *quality of service* (QoS) requirements.

In HARNESS we developed an approach for describing the problem of scheduling malleable jobs on heterogeneous virtualised resources with non-uniform semantics, under the assumption that the central scheduler (employed by the top-level HARNESS manager) has no knowledge or understanding of how underlying resources function internally, thus allowing its scheduling algorithms to be independent from the type of resources they manage.

To support a central scheduling process that has no specific understanding about the type of resources it manages, we use *child resource managers*, as shown in Figure 3.3 that are in charge of translating an agnostic management request from the top-level manager to a specific request for a particular type of resource. Each child resource manager is responsible for reporting available resources and their respective capacities, performing allocation and deallocation requests, as well as reporting monitoring information (e.g., resource utilisation).

One challenge to supporting resource-agnostic scheduling, is that the top-level manager cannot simply look at the descriptions of available resources and reason about their nominal and residual capacities, including understanding whether a sequence of allocation requests can "fit" a resource. For instance, consider the MPC-X cluster shown in Figure 3.3, where an allocation request can specify not only the number of DFEs, but also their topology (for instance, whether DFEs must be interconnected via a ring topology, or be independent in a group topology). Such requests require knowledge about the internal state of a resource in order to interpret its capacity.

Therefore, while a given resource x has some finite capability or *capacity* (as resources with infinite capacities can be safely ignored by definition), it may not be possible to fully represent this capacity as a singleton or vector of numerical values. Rather than modelling finite resource capacities as integers or vectors, we assume a more abstract representation, wherein the top-level manager has access, through

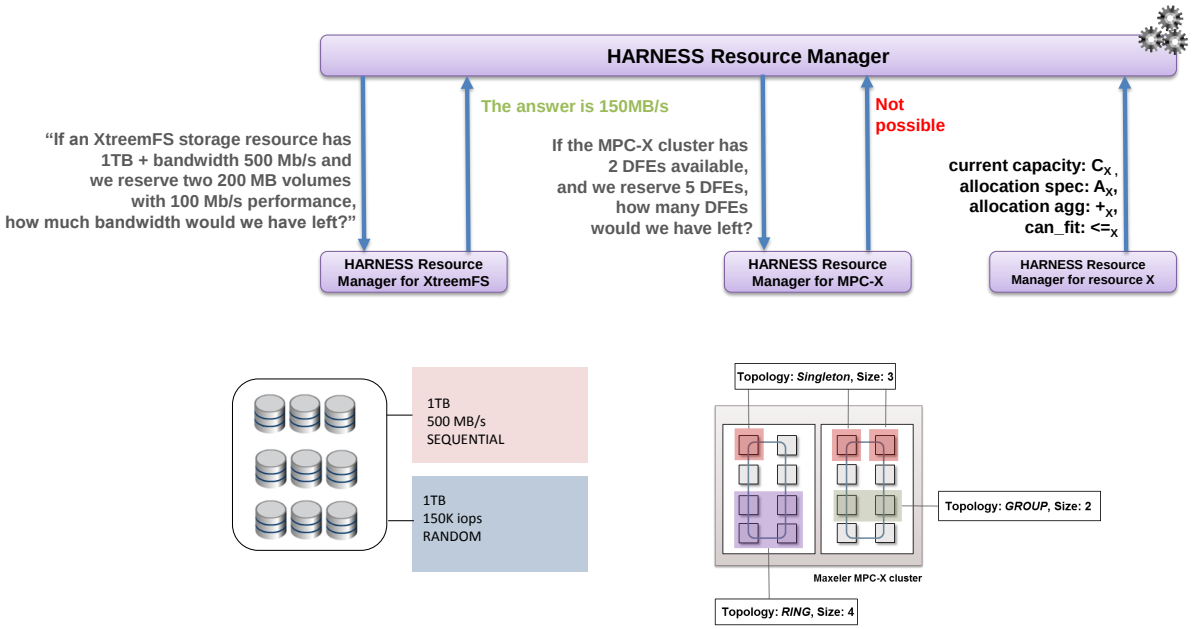


Figure 3.3: The top-level manager can schedule abstract resources with the support of child resource managers. To support higher-level management, each child resource manager handling a resource x exposes abstract operators ($+_x$, $\leq_x$), the allocation specification (A_x) which defines the space of all valid allocation requests, and the capacity of the resource (C_x). XtreemFS is a cloud file system that manages heterogeneous storage devices. The MPC-X cluster provides a pool of Maxeler DFEs. A DFE is a physical compute resource which contains an FPGA as the computation fabric and RAM for bulk storage, and can be accessed by one or more CPU-based machines.

the child manager that manages resource x, to an operation of composition ($+_x$) that groups allocation requests, a partial-ordering relation ($\leq_x$) to reason about resource capacity, the allocation specification (A_x) that defines the space for all valid allocation requests, and the capacity of x, denoted C_x. Then, for any ordered sequence of requests to acquire or release allocations on a resource x, it is possible to compute deterministically whether or not that set of requests could be serviced by x.

In practice, this means that the top-level resource manager can use these abstract operators during the scheduling phase to derive a set of resources that can best service an allocation request without having to directly interpret the capacity of each type of resource. In Figure 3.3 we illustrate the interactions between the top-level and child resource managers during the scheduling process.

3.4 The HARNESS API

The HARNESS API is an interface implemented by all HARNESS resource managers to handle a specific type of resource. It mostly follows the RESTful style, where interactions between components are handled through HTTP requests. Resource managers supporting this API can be combined in an hierarchical fashion to support a heterogeneous cloud computing platform that can handle various types of resources, from physical clusters of FPGAs to conceptual resources such as virtual links.

While the implementation of the API is specific based on the nature of the resource, the API makes a few assumptions about its nature. First, all resources have a capacity that can be finite or infinite. Second, a resource operates on a specific allocation space. Third, there is a function that can compute (or estimate) changes in capacity based on an allocation request. Fourth, we can create and destroy instances of a resource. Fifth, resources can be monitored with feedback provided on specific metrics.

We have grouped the HARNESS API functions into four categories (Figure 3.4):

1. **Management.** Allows resource managers to connect to other HARNESS resource managers in a hierarchical organisation.

2. **Resources.** Provides information about the state of available resources, and meta-information about allocation requests and resource capacity.

3. **Reservations.** Allows resource instances to be created and destroyed.

4. **Metrics and Logging.** Provides monitoring information about resources and their instances, and also logging information.

When designing the HARNESS API, we considered existing cloud standards. One key difference between the cloud and its predecessor, the grid, is that cloud standards are often de facto and based on simple interfaces and running code, rather than being meticulously specified in advance by committees representing the interests of various stakeholders (as was the case for the grid). In the early days of the cloud, open-source projects would either attempt to mimic the not-fully-documented interfaces of commercial implementations [6] or simply create their own APIs (e.g., OpenStack Nova). While this approach has allowed for the rapid evolution of cloud service offerings, it has also resulted in instability and the proliferation of a widespread set of similarly functioned, but incompatible interfaces [2].

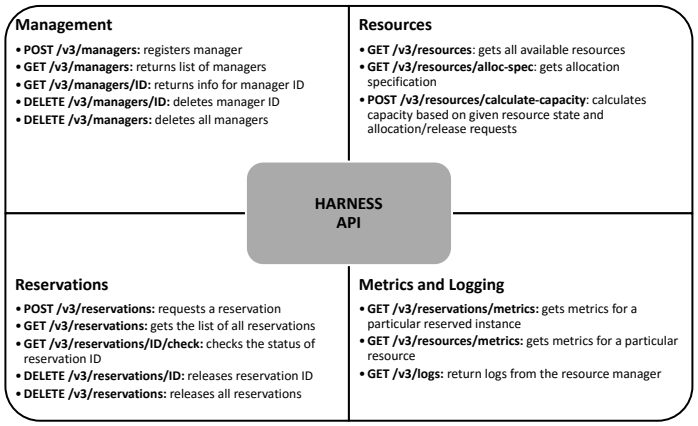


Figure 3.4: The HARNESS API provides a REST-based interface for HARNESS resource managers to handle heterogeneous resources.

Relation to related work. Attempts to standardise on cloud interfaces have been spearheaded by a number of organisations, most notably OASIS, DMTF, OGF and SNIA. While cloud vendors have been slow to fully embrace these projects, third party projects such as Apache Libcloud [4] and Deltacloud [3], which provide thin translation layers between incompatible APIs, have more or less made it possible for end users to escape vendor lock-in. The current state of the major available open cloud interface standards is summarised in Table 3.1, and described below:

- The **TOSCA** [19] standard from OASIS describes how to specify the topology of applications, their components, and the relationships and processes that manage them. This allows the cloud consumer to provide a detailed, but abstract, description of how their application or service functions, while leaving implementation details up to the cloud provider. With TOSCA, cloud users can not only describe how their distributed application should be deployed (in terms of which services need to communicate with each other), but also how management actions should be implemented (e.g., start the database before the web service, but after the file server).
- The **CAMP** [18] standard from OASIS targets PaaS cloud providers. This standard defines how applications can be bundled into a *platform deployment package* (PDP), and in turn how these PDPs can be deployed and managed through a REST-over-HTTP interface. It includes descriptions of *platforms*, *platform components* (individual services offered by a platform), application components (individual

Specification	Standards Org.	Version	Focus
TOSCA	OASIS	1.0	orchestration
CAMP	OASIS	1.1	deployment
CIMI	DMTF	2.0	infrastructure
OCCI	OGF	1.1	infrastructure
CDMI	SNIA	1.1.1	storage infrastructure

Table 3.1: Current open cloud interface standards.

parts an application that run in isolation) and assemblies (collections of possibly communicating application components). Users can specify how application components interact with each other and components of the platform, but infrastructural details are left abstract. For example, a user would not know if their components were executing within shared services, containers, individual virtual machines, and the like.

- The **CIMI** [5] standard was developed by the CMWG of DMTF. As with most cloud standards it specifies REST over HTTP. While XML is the preferred data transfer format due to the ability to easily verify that data conforms to the appropriate schema, JSON is also allowed. CIMI has predefined resource types in the standard, notably networks, volumes, machines, and systems (collections of networks, volumes, and machines), and the actions that may be performed upon a resource are constrained by its type (e.g., machines can be created, destroyed, started, stopped, or rebooted).
- The **CDMI** [22] international standard specifies "the interface to access cloud storage and the data stored therein". CDMI was developed by SNIA, an industry group with an obvious interest in promoting standard interfaces for cloud storage. It defines *containers*, which represent abstract storage space in which files can be stored, and *data objects*, which roughly correspond to files. The standard implementation uses REST over HTTP. It provides similar functionality to Amazon's S3 interface [23].
- The **OCCI** [17] standard comes from OGF, an organisation that was previously very active in defining grid computing standards. It is divided into three sections: core, infrastructure and HTTP rendering. The core specification defines standard data types for describing different types of resources, their capabilities, and how they may be linked together, while the infrastructure specification explicitly defines resource types for compute, network, and storage. The HTTP rendering section defines how to implement OCCI using REST over HTTP. Taken together, these three sections allow OCCI to present similar functionality as found in other IaaS-layer interfaces, such as Amazon EC2, OpenStack Nova, or CIMI. However, OCCI is designed to be flexible, and the core specification can be used with extension standards to define new resource types.

The HARNESS API sits slightly above the pure IaaS layer. It does not attempt to describe application orchestration, like TOSCA, or deployment, like CAMP, leaving these problems to the AM. Instead, like OCCI and CIMI, it is more concerned with allocating and linking infrastructure resources. However, unlike these standards, which come with built-in models of different resource "types", such as physical machines, virtual machines, networks, and storage devices, the HARNESS API considers all abstract resources to be of the same type.

This allows for a more flexible model that can accommodate a wider array of cloud-enabled devices, as well as supporting cross-cutting services such as pricing and monitoring, which do not have to conform to multiple resource models. Once resources have been allocated in HARNESS, the deployment phase allows each provisioned resource to be handled using specific APIs, tools and services to make full use of their capabilities.

4 Pilot Applications

The HARNESS project results have been measured against three representative validation use cases ("pilot applications").

4.1 Reverse Time Migration

In the domain of computational seismography, *reverse time migration* (RTM) is an algorithm for creating a 3D-model of underground geological structures. It is a computationally intensive technique employed by project partner *Maxeler Techologies* (MAX) with wide applicability in oil and gas exploration. However, its use is limited by the cost of computation and by constraints of network bandwidth. Typical running times are on the order of several weeks to months, with input data on the order of tens of terabytes, and final output data of a few gigabytes. Several different RTM algorithms exist that trade computation for storage cost. Depending on the relative performance of computation and storage resources, it is important to make the right choice among the algorithms.

The RTM application deployed in HARNESS is *malleable*: it can adapt to different resource configurations, flexible in the number and type of resources provisioned by the cloud platform even during execution. Application malleability provides more flexibility, both in terms of completion time and monetary cost than a static application configuration. HARNESS also offers performance isolation guarantees between two or more RTM applications (submitted by different tenants) that run at the same time.

4.2 In-Memory Data Analytics in SAP HANA

The SAP HANA Cloud uses an *in-memory database* (IMDB) system offering real-time performance for large data volumes (on the order of terabytes) with a current focus on *on-line analytics processing* (OLAP). The column store underlying the SAP HANA database uses dictionary encoding and other compression algorithms to ensure that all relevant data are accessible in main memory. This provides two major advantages over row-based tables, the first being sequential and contiguous allocation of data in main memory for each column. For analytic queries, where individual columns may be processed at a time, this contiguous storage can speed up processing of the data due to reduced cache misses associated with random access and the ability to vectorise processing of the data. The second advantage comes in the form of table compression through dictionary encoding. As the data are column oriented, each unique data item in the column can be recorded in a dictionary and replaced by a single index reference to the position the data item resides in the dictionary. Using this scheme results in high levels of compression for columns with few unique values and is especially useful for strings.

Analytic queries are compute intensive and can place significant load on a HANA system. With HARNESS, multiple users can benefit from acceleration using special-purpose processors. Some of these queries, for example, can be accelerated using GPGPU plugins. HARNESS manages a provisioned set of

heterogeneous processing resources across many simultaneous user requests (i.e., application tasks) to a HANA system. Load balancing tasks across computational resources reduces demand on the host system, thereby indirectly improving the performance of the database as a whole.

4.3 Machine Learning: AdPredictor

AdPredictor represents a class of industrial applications commonly known as *recommender systems* implemented using open-source map/reduce frameworks. AdPredictor is used to match advertisements ("ads") with a user query posed to a Web search engine (e.g., Google or Bing) such that the displayed ads will have a high probability of being clicked by that user.

The computation is expressed using a *data-flow graph* (DFG), where vertices are data-parallel tasks and edges their I/O relationships. AdPredictor's Bayesian training model results in a complex DFG. With gigabytes of user click logs recorded daily, the input data must be partitioned and distributed on disks and memory accordingly. The application is also compute intensive, having to calculate updates for a Gaussian mixture model that consists of millions of parameters. Put together, data movements and data parallel computations must be carefully orchestrated to ensure high quality (i.e., accurate) predictions.

HARNESS enhances DFG processing by taking into account the diverse capabilities of heterogeneous resources: it uses different storage technologies based on the read/write patterns and placement of tasks to reduce storage costs; it uses FPGA accelerators to speed-up computations; it aggregates model updates within the network, taking advantage of the computation capabilities of modern switches and routers, to reduce the network traffic and data movement costs; and, it enables programmers to explore different configurations that trade accuracy against deployment costs.

5 Results

In this section we report on two case studies selected from the several that we conducted over the course of the project:

- **Executing HPC Applications in the Cloud.** This case study (Section 5.1) demonstrates how scientific computing applications such as RTM can exploit hardware accelerators and managed storage volumes as cloud resources using HARNESS. These experiments were performed in the IMP cluster testbed.
- **Resource Scheduling with Network Constraints.** This case study (Section 5.2) demonstrates the benefits of managed networking when deploying distributed applications, such as Hadoop MapReduce, in a cloud platform. In this scenario, cloud tenants are able to reserve bandwidth, which affects where jobs are deployed. This work was conducted in Grid'5000, with physical nodes located in the French cities of Rennes and Nantes.

5.1 Executing HPC Applications in the Cloud

In this section we demonstrate the flexible deployment of an *high-performance computing* (HPC) application on the HARNESS cloud platform, using RTM as the case study subject. RTM represents a class of computationally intensive scientific applications used to process large amounts of data. Some of the most computationally intensive geoscience algorithms involve simulating wave propagation through the earth. The objective is typically to create an image of the subsurface from acoustic measurements performed at the surface. To create this image, a low-frequency acoustic source is activated and the reflected sound waves are recorded, typically by tens of thousands of receivers. We term this process a *shot*, and it is repeated many thousands of times while the source and/or receivers are moved to illuminate different areas of the subsurface. The resulting dataset is dozens or hundreds of terabytes in size. The problem of transforming this dataset into an image is computationally intensive and can be approached using a variety of techniques.

The experiments were conducted on the IMP cluster testbed, which includes three CPU machines, a cluster of 24 DFEs, and standard *hard disk drive* (HDD) and SSD storage drives. The dataset used in our experiments are based on the Sandia/SEG Salt Model 45 shot subset.[1]

5.1.1 Deploying RTM on a heterogeneous cloud platform

In this experiment, we demonstrate how RTM is deployed and executed in the HARNESS cloud platform. The RTM binary, along with its deployment and initialisation scripts, are compressed into a tarball. This tarball is submitted to the HARNESS cloud platform frontend, along with an *application manifest*. The application manifest describes the resource configuration space, which allows the *Application Manager* (AM) to derive a valid resource configuration to run the submitted version of RTM.

[1] `http://wiki.seg.org/wiki/SEG_C3_45_shot`

When the application is submitted, the HARNESS platform creates an instance of the AM to oversee the life cycle of the application. The AM operates in two modes. In the *profiling* mode, the AM creates a performance model by running the application on multiple resource configurations and capturing the execution time of each configuration. Associated with the performance model, we use a cost model that, given a resource configuration, provides the cost to provision the configuration per unit of time (Euros/second). With the performance and cost models, the AM can translate cost and performance objectives specified in an SLO (for example, "execute the fastest configuration") into a resource configuration that can best achieve these objectives.

For this experiment, the AM deployed RTM on different resource configurations, varying the number of CPU cores (from 1 to 8), *random-access memory* (RAM) sizes (1024MB and 2048MB), number of DFEs (from 1 to 4) and storage performance (10MB/s and 250MB/s). The cost model is as follows:

$$cost(c) = c.\mathsf{num\_dfes} \times \mathsf{9e\text{-}1} + c.\mathsf{cpu\_cores} \times \mathsf{5e\text{-}4} + c.\mathsf{mem\_size} \times \mathsf{3e\text{-}5} + c.\mathsf{storage\_perf} \times \mathsf{1e\text{-}05}$$

where for a given configuration c, num_dfes represents the number of DFEs, cpu_cores corresponds to the number of CPU cores, mem_size corresponds to the size of RAM, and storage_perf the storage performance. The resulting cost is given in Euros/second.

Figure 5.1 shows the performance model generated by the AM using the cost model described above. The AM automatically selected and profiled 28 configurations, with five of these configurations part of the Pareto front. The number of profiled configurations is dependent on the profiling algorithm used by the AM [14]. For each configuration, the AM reserves the corresponding resources, and deploys the application. The initialisation script supplied with the application automatically detects the configuration attributes, and adapts the application to it. For instance, if the configuration specifies 8 CPU cores, then the initialisation script configures the `OMP_NUM_THREADS` environment variable to that number, to allow the application to fully utilise provisioned CPU resources.

As shown in Figure 5.1, there are four configurations highlighted at the top-right quadrant, which are the most expensive and slowest configurations, and thus the least desirable configurations. This is due to the use of slow storage (9MB/s) which dominates the performance of the job despite the use of DFEs. At the bottom-right quadrant, we highlight five configurations that are relatively inexpensive. However, they perform slowly, since they do not use DFEs. Finally, in the centre of the figure, we highlight three configurations that use a limited number of CPU cores and DFEs, but they do not provide the best trade-off between price and execution time. Instead, the five configurations that provide the best trade-offs are those in the Pareto front, as listed here:

#ID	#DFEs	#CPU Cores	RAM	Storage Speed	Execution Time	Price
A	0	4	1024MB	250MB/s	84s	0.03
B	1	8	2048MB	250MB/s	17s	0.17
C	2	1	1024MB	250MB/s	12s	0.21
D	3	3	2048MB	250MB/s	10s	0.27
E	4	3	2048MB	250MB/s	8s	0.31

With these configurations and the corresponding pricing, the AM can service SLO-based requests. For instance, if the user requests the fastest configuration under 0.25, the AM selects configuration C, while the cheapest configuration is A. The performance model in our example works only to a specific problem size and configuration, with other problem sizes requiring additional profiling information.

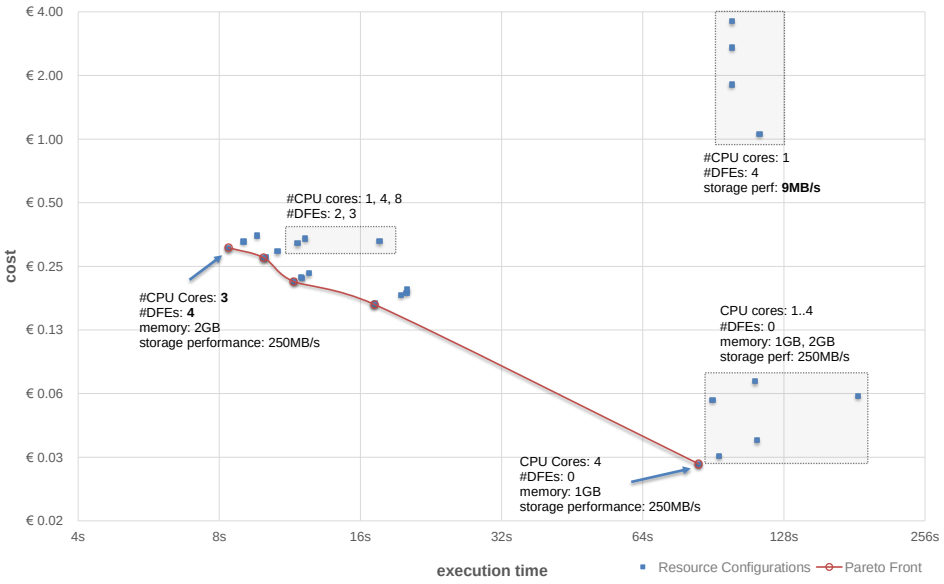


Figure 5.1: Performance model for the RTM demonstrator, automatically generated by the AM.

5.1.2 Exploiting different DFE topology reservations

As seen in the previous experiment, the RTM job deployed in HARNESS is *moldable* [7]) because it can adapt to different resource configurations, thus being flexible in the number and type of resources provisioned. While a moldable job can only adjust to available resources at the *start*, a *malleable* job can be reconfigured at run time, during execution. Both types of jobs provide more flexibility than a *rigid* job that can only execute on a single resource configuration. In this experiment, we explore how RTM behaves when we exploit its moldable and malleable properties.

Our current implementation of the HARNESS platform supports the *DFE cluster resource*, as shown in Figure 5.2. A DFE cluster is composed of an arbitrary number of MPC-X boxes. An MPC-X box contains a set of DFEs that are connected together in a ring topology. This allows each DFE to connect directly to the CPU host, as well as with adjacent DFEs. The CPU host connects to the DFE cluster via Infiniband.

With the HARNESS platform, cloud tenants can reserve an arbitrary number of DFEs (subject to availability) and its topology using the application manifest. In our current implementation, we support three types of topologies: *SINGLETON*, *GROUP* and *RING*. More specifically, when reserving a *SINGLETON* with N DFEs, the platform (through MaxelerOS Orchestrator) returns DFEs from any MPC-X box. Reserving a *GROUP* with N DFEs returns resources within the same MPC-X box. Finally, reserving N DFEs with a *RING* topology results in N DFEs that are interconnected via ring interconnect. The difference between a SINGLETON and a GROUP reservation is that the latter allows a group of physical DFEs to be operated as a single virtual DFE [8].

Figure 5.3 shows the performance of a single RTM shot using different problem dimensions and numbers of DFEs. Multiple DFEs are connected via RING. The characterisation of these jobs is summarised in Table 5.1. We can see that the number of DFEs makes little impact on smaller jobs, such

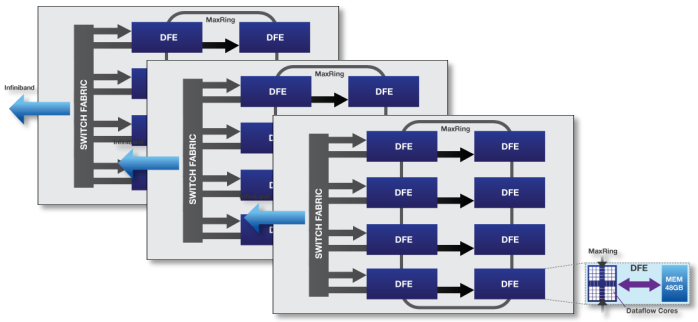


Figure 5.2: A DFE cluster is composed of an arbitrary number of MPC-X boxes. Here, a DFECluster manages three MPC-X boxes with eight DFEs each. The DFEs in each MPC-X are connected physically via a RING interconnect.

design	configuration	dimension	#iterations
S1	1×DFE	200 × 200 × 200	2000
S2	2×DFEs (ring)	200 × 200 × 200	2000
S3	3×DFEs (ring)	200 × 200 × 200	2000
M1	1×DFEs	400 × 400 × 400	4000
M2	2×DFEs (ring)	400 × 400 × 400	4000
M3	3×DFEs (ring)	400 × 400 × 400	4000
L1	1×DFEs	600 × 600 × 600	6000
L2	2×DFEs (ring)	600 × 600 × 600	6000
L3	3×DFEs (ring)	600 × 600 × 600	6000

Table 5.1: Three classes of RTM jobs using different numbers of DFEs.

as $S1$, $S2$ and $S3$. This is due to the fact that smaller workloads will not be able to fully utilise multiple DFEs. Larger jobs, on the other hand, scale better and are able to exploit larger numbers of DFEs. For instance, $S3$ is only 0.7 times faster than $S1$, while $L3$ is 2.6 times faster than $L1$.

We next focus on the impact of the DFE topology on completing a multi-shot RTM job. Figure 5.4 shows the results of completing an RTM job with a varying number of shots. Each shot has a dimension of $600 \times 600 \times 600$, running in 6000 iterations. For each number of shots, we compare the performance of running three DFEs independently as GROUP against three DFEs interconnected as a RING. The three independent DFEs can process three shots in parallel, while the three interconnected DFEs, working as a single compute resource, can only process shots sequentially. Each independent DFE is capable of computing a shot in 2352.7s, while the interconnected DFEs process a shot in 881.99s. It can be seen from the figure that depending on the total number of shots, one of the topologies is more efficient than the other. For seven shots, the independent DFEs run faster, while for 11 shots the interconnected DFEs run faster. Since RTM jobs are moldable, it can be adapted to run with one topology or another, depending on the required number of shots.

We can further speed up the computation by making the RTM job *malleable*, such that it adapts at run time. As seen in Figure 5.4, depending on the number of shots, we can combine both types of topologies to reduce the execution. For instance, for 11 shots, we can execute 9 shots in three sequences

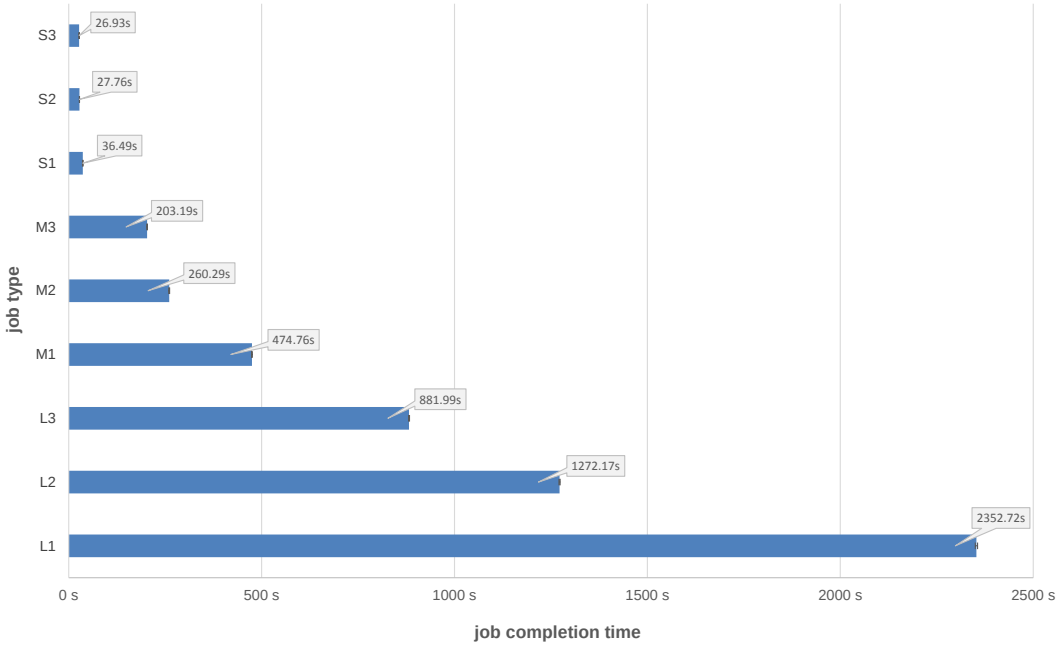
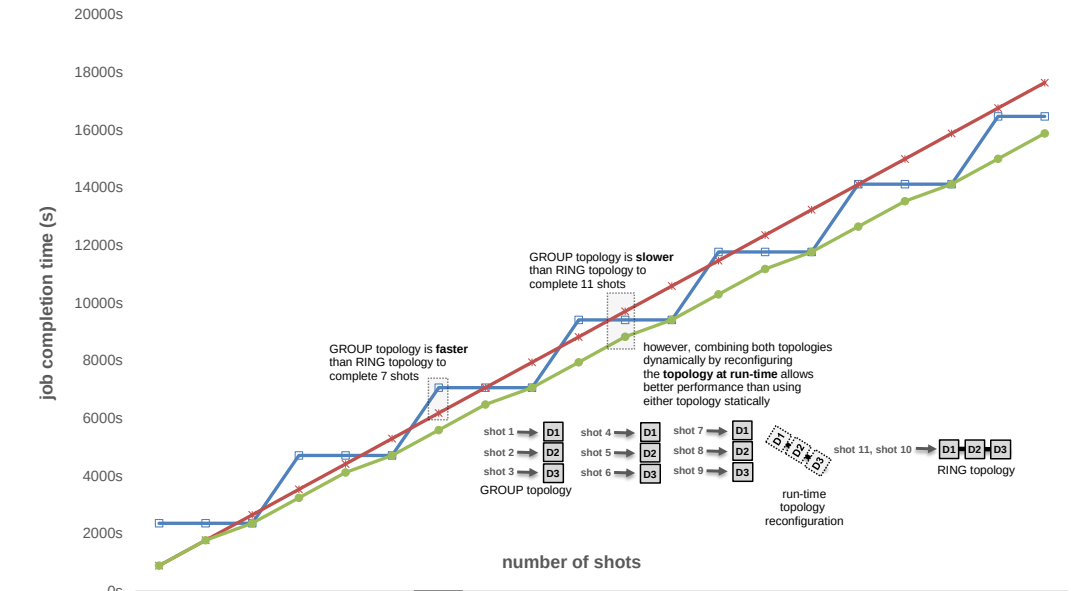


Figure 5.3: Performance of a single RTM shot using different problem dimensions (S, M and L) and numbers of DFEs (1, 2 and 3).



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
GROUP	2352.7	2352.7	2352.7	4705.4	4705.4	4705.4	7058.2	7058.2	7058.2	9410.9	9410.9	9410.9	11764	11764	11764	14116	14116	14116	16469	16469
RING	881.99	1764	2646	3528	4410	5291.9	6173.9	7055.9	7937.9	8819.9	9701.9	10584	11466	12348	13230	14112	14994	15876	16758	17640
Reconfiguration	881.99	1764	2352.7	3234.7	4116.7	4705.4	5587.4	6469.4	7058.2	7940.2	8822.1	9410.9	10293	11175	11764	12646	13528	14116	14998	15880

Figure 5.4: The effect of different topologies on RTM performance.

in parallel followed by two shots done with the three DFEs interconnected. This combination yields the best performance (8821.1s), better than if we had selected a static configuration (9410.9s for the parallel configuration and 9701.9s for the interconnected configuration). We expect larger problems to greatly benefit from this flexibility.

5.2 Resource Scheduling with Network Constraints

The purpose of the second case study is to demonstrate the benefits of managed networking when deploying a distributed application. We use the Hadoop-based AdPredictor on the large-scale testbed Grid'5000 as our subject. AdPredictor represents a class of modern industrial-scale applications, commonly known as *recommender systems*, that target either open-source or proprietary on-line services. For example, one such service is Mendeley [16], a free reference organiser and academic social network that recommends related research articles based on user interests. Another such service is Bing, a commercial on-line search engine that recommends commercial products based on user queries. In general, items are matched with users and, due to the modern "data deluge", these computations are usually run in large-scale data centres. The ACM 2012 KDD Cup *track2* dataset [1] is used in the evaluation.

Grid'5000 is a large-scale multi-site French public research testbed designed to support parallel and distributed computing experiments. The backbone network infrastructure is provided by the French National Telecommunication Network for Technology, Education and Research RENATER, which offers 11,900 km of 120 optic fiber links and 72 points of presence.

Figure 5.5 reports the throughput of one MapReduce worker over time, where the steady throughput consumption peaks at approximately 200 Mbit/sec, excluding any bursts during shuffling. On the other hand, Figure 5.6 presents the results of an AdPredictor job using the same configuration by varying the available bandwidth between the worker compute hosts. It can be seen that the application exhibits degradation below 200 Mbit/sec, which is consistent with our measurements in Figure 5.5.

Consequently, a tenant deploying an AdPredictor application on a cloud platform that does not provide any guarantees could have their resources reserved in a low-bandwidth environment, hindering the performance of the application. HARNESS addresses these issues by exposing network bandwidth and latency as resources and constraints, respectively. Applications may define and submit their desired network performance guarantees and the underlying infrastructure will provision them, if possible.

Next, we report three scenarios conducted in Grid'5000: (a) scheduling without network constraints; (b) scheduling using bandwidth reservation requests; and (c) scheduling with SLOs. The testbed used consists of eight physical nodes, across two different sites: *Rennes* (four nodes) and *Nantes* (four nodes). While both sites offer high-speed gigabit connectivity of 1500 Mbit/sec, we have emulated heavy network congestion in Nantes, so that the throughput of any network flow in Nantes is limited to 500 Mbit/sec.

5.2.1 Scheduling without network constraints

In the first experiment, we request one *master* and three *worker* instances without specifying network constraints. Figure 5.7 presents all the possible configurations that may result from this allocation request, as each worker may be placed either in Rennes or in Nantes. If the tenant specifies no network constraints, any one of these placements may be possible and, therefore, the end user will experience a considerable variability in the application's performance over multiple deployments on the same cloud platform. It is

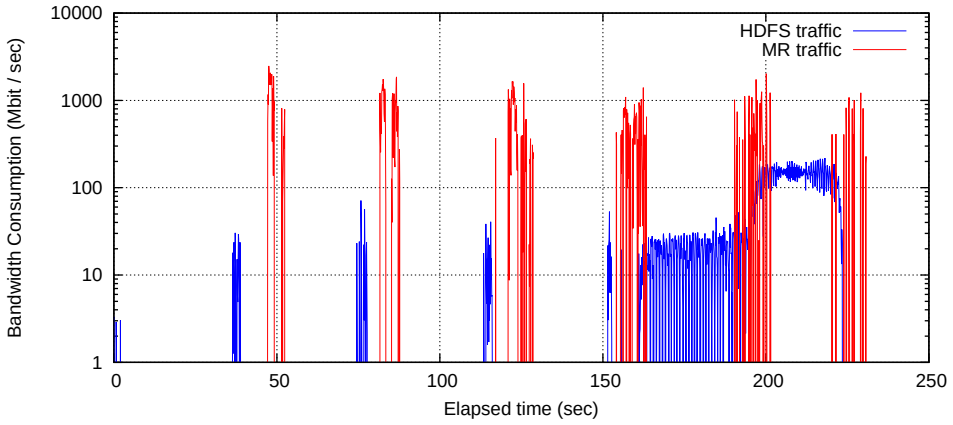


Figure 5.5: AdPredictor throughput over time.

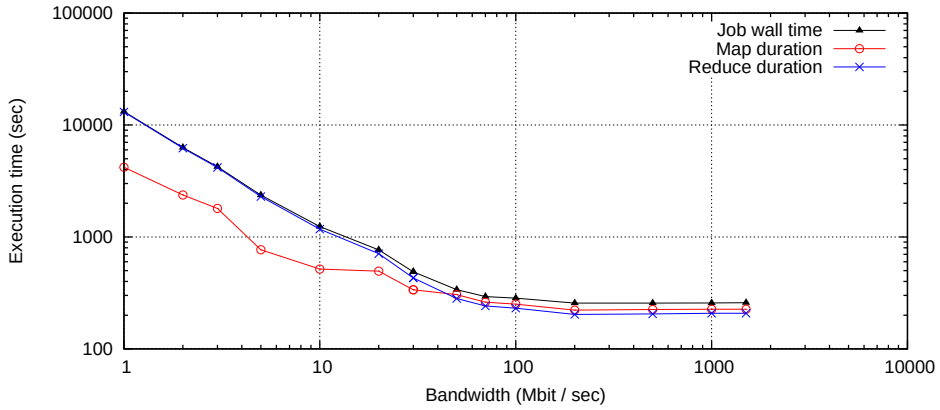


Figure 5.6: AdPredictor performance vs. bandwidth.

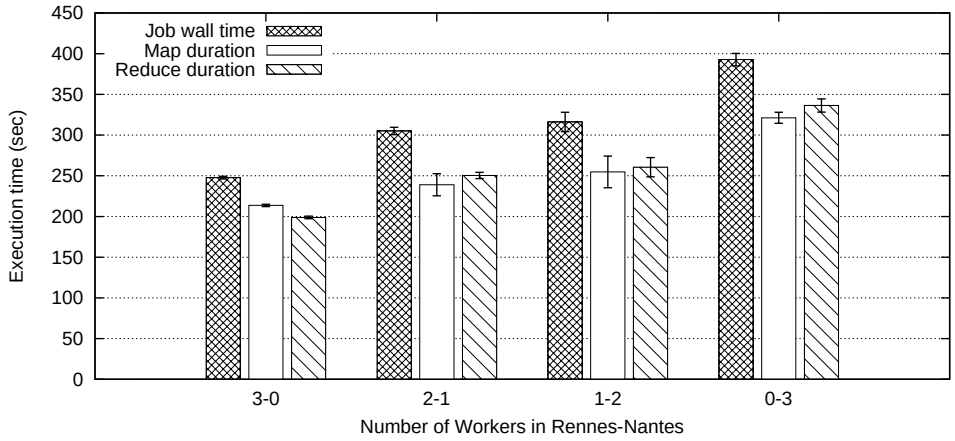


Figure 5.7: AdPredictor performance vs. resource placement.

evident that the application's execution time is dependent on whether the workers are deployed in the high-bandwidth Rennes cluster, in the congested Nantes cluster, or across both sites. Nevertheless, in this scenario the tenant has no control over the final placement.

5.2.2 Scheduling using bandwidth reservation requests

In order to eliminate the suboptimal placements presented in Figure 5.7, labelled as "2-1", "1-2" and "0-3", which involve at least one of the workers being placed in the congested cluster of Nantes, the tenant can specify the following allocation request in the application manifest:

```
{
    "Master": {
        "Type": "Machine",
        "Cores": 4,
        "Memory": 4096
    },
    "Worker1": {
        "Type": "Machine",
        "Cores": 12,
        "Memory": 16384
    },
    "Worker2": {
        "Type": "Machine",
        "Cores": 12,
        "Memory": 16384
    },
    "Worker3": {
        "Type": "Machine",
        "Cores": 12,
        "Memory": 16384
    },
    "Link1": {
        "Type": "Link",
        "Source": "Worker1",
        "Target": "Worker2",
        "Bandwidth": 300
    },
    "Link2": {
        "Type": "Link",
        "Source": "Worker2",
        "Target": "Worker3",
        "Bandwidth": 300
    },
    "Link3": {
        "Type": "Link",
        "Source": "Worker1",
        "Target": "Worker3",
        "Bandwidth": 300
    }
}
```

This request demands a reservation of 300 Mbit/sec between workers `Worker1` and `Worker2`, `Worker1` and `Worker3`, and `Worker2` and `Worker3`. Consequently, the CRS takes into account this bandwidth reservation request when allocating *virtual machine*s (VMs) (containers) for the workers, therefore eliminating the suboptimal placements and deploying all workers in Rennes under the conditions presented in this experiment.

5.2.3 Scheduling with service-level objectives

In the previous experiment, we requested bandwidth reservation in the application manifest to deploy a Hadoop-based AdPredictor job. Typically, the cloud tenant is more concerned with job completion time and the cost of reserving resources. In HARNESS, such objectives are specified as an SLO. In order for the HARNESS platform to derive a resource configuration that can meet performance or cost objectives, it needs to have a performance/cost model. This can be automatically generated by the AM or provided by the user. For this experiment, we have generated a performance model based on profiling AdPredictor with different bandwidth reservation requests.

The cost model is as follows:

$$cost(c) = c.cpu\_cores \times 5e-4 + c.mem\_size \times 3e-5 + c.bandwidth \times 2e-1$$

The bandwidth unit is Mb/s. With this cost model, the price of 1Gb/s bandwidth is roughly equal to one container with four cores and 8GB RAM.

Figure 5.8 presents the performance/cost model of AdPredictor running on Grid'5000. It contains 14 points, where we vary the bandwidth requirements from 1Mb/s to 1.5Gb/s, while maintaining the same compute and storage configuration. It can be seen that different bandwidth reservations have an impact in both pricing and performance. Not all configurations provide a good trade-off between price and execution time, and they are discarded. The remaining configurations, seven in total, are part of the Pareto front. These configurations are then selected to satisfy objectives in terms of pricing (the cheapest configuration costs 2.93 but requires 39 minutes to complete) or in terms of completion time (the fastest configuration completes the job in 256s but costs 12.69).

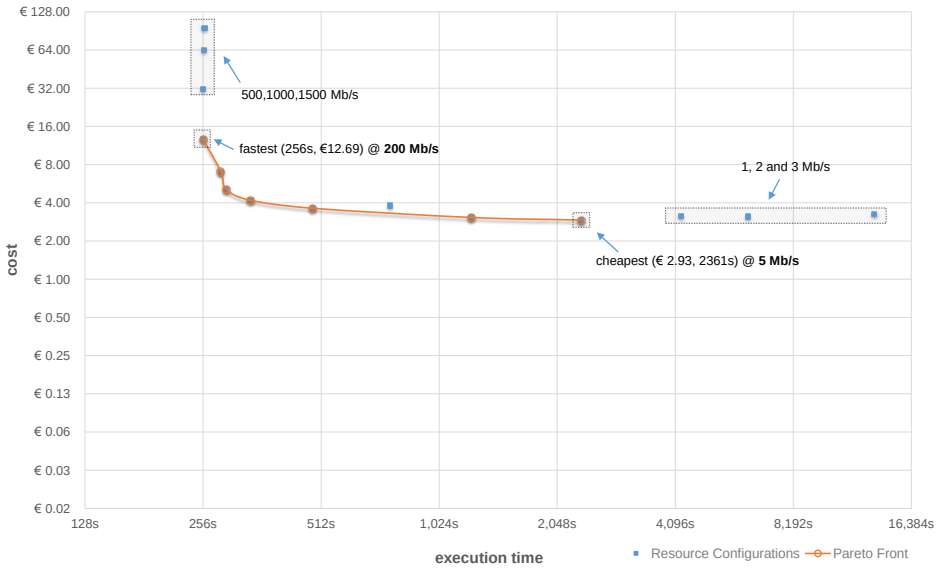


Figure 5.8: Performance/cost model generated for AdPredictor running on Grid'5000.

6 Availability of Results

The HARNESS consortium has made available to the general public the key outcomes of its joint collaboration, including up-to-date source code and documentation:

`http://github.com/harnesscloud/`

The model for software licensing within the project is open source, primarily using a *Berkeley software distribution* (BSD) license in order to maximise the potential for reuse. However, there are some exceptions to this approach, particularly when development resulted in extension of existing software projects, or the open source model would interfere with plans for commercialisation. To improve the referencability and long-term availability of these software projects, final release versions were permanently archived and given unique *Document Object Identifier*s (DOIs) through Zenodo.[1]

Table 6.1 presents the HARNESS components currently available for download in GitHub, along with the partners that are anticipated to maintain them beyond the end of the project.

6.1 ConPaaS and XtreemFS

Two partners in HARNESS, *Université de Rennes 1* (UR1) and *Konrad-Zuse-Zentrum für Informationstechnik Berlin* (ZIB), are supporting and maintaining two large open-source projects, *Contrail platform-as-a-service* (ConPaaS), a run-time environment for hosting applications in the cloud, and XtreemFS, a distributed file system designed for the cloud environment, both of which are currently being employed by a number of users and researchers in various collaboration initiatives. Expertise in these fields can lead to a successful commercialisation of the basic research and technology that were developed during the project, possibly resulting in start-up companies or direct exploitation of the outcomes through consulting services. One start-up, Quobyte,[2] has already been built on XtreemFS technology, which would benefit from the QoS virtualisation extensions developed by ZIB as part of HARNESS.

XtreemFS releases new versions about twice per year. In the March 2015 release,[3] it integrated the Hadoop improvements developed for AdPredictor and quota support. The October 2015 contains further technologies developed in HARNESS [10].

ConPaaS also releases new versions about twice per year. In the latest release,[4] it integrated the Generic service developed originally for the specific needs of the HARNESS project. The next release is planned for the end of 2015. It will contain further technologies stemming from HARNESS, most notably a significant structural reorganisation to integrate AMs in ConPaaS. This reorganisation is of sufficient importance to justify the transition from release numbers such as 1.5.1 to 2.0.x.

[1] `http://zenodo.org`
[2] `http://www.quobyte.com/`
[3] `http://xtreemfs.blogspot.de/2015/03/xtreemfs-151-released.html`
[4] `http://www.conpaas.eu/conpaas-releases-version-1-5-0/`

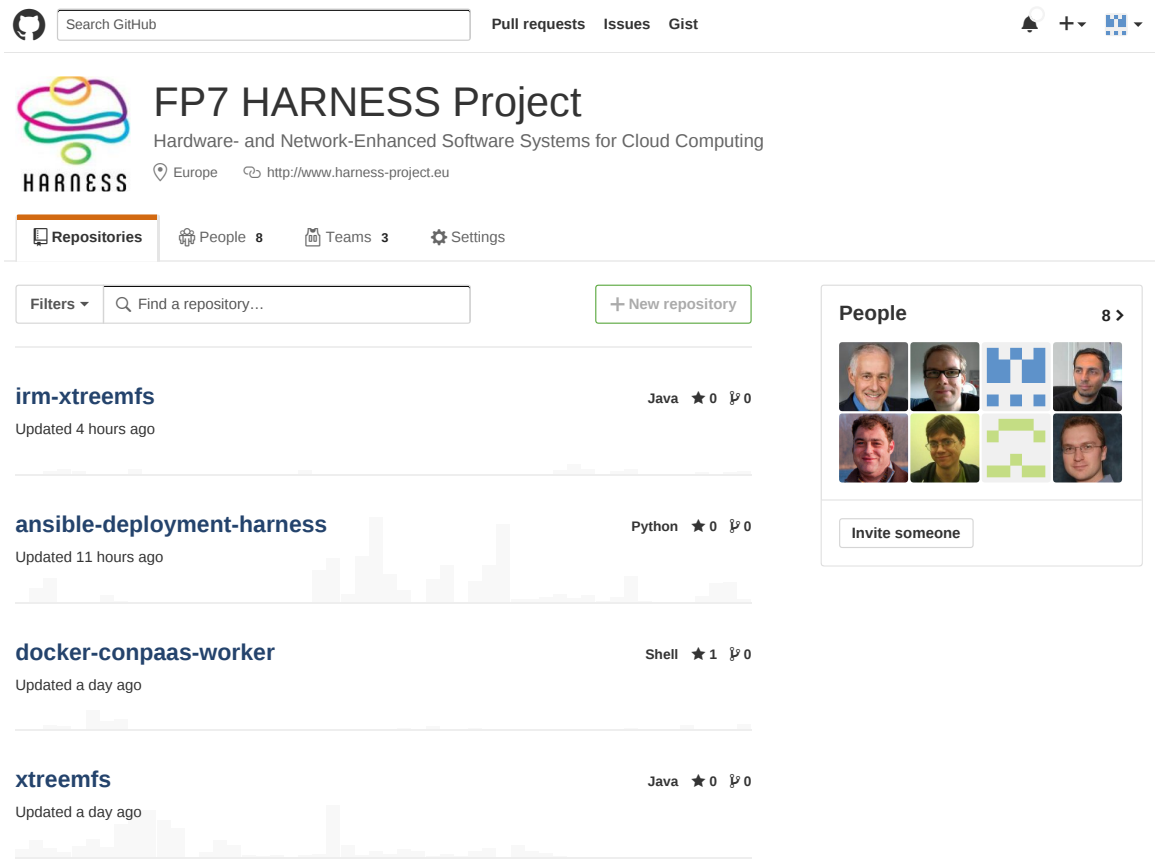


Figure 6.1: The HARNESS GitHub Repository.

6.2 Ansible HARNESS Deployments

A key challenge of the HARNESS project is that it requires bringing together specialists from a number of geographically distributed partner institutions in academia and industry. Individual components addressing different classes of heterogeneous resources or requirements can be developed independently, but there is a need to coordinate effort and ensure that components can be deployed in a way that provides a coherent distributed computing infrastructure. Furthermore, while responsibility for the quality of the individual components is shared among a number of lead institutions, there is a need to provide a way to test and evaluate the fully integrated deployment as well.

To address these challenges, we have created the **Ansible**[5] **HARNESS deployment project**, available on our GitHub repository. The project defines a *Development and Operations* (DevOps) workflow that allows reproducible and automated HARNESS deployment on heterogeneous and large-scale testbeds.

[5] `http://ansible.com`

Software Project	Work Packages	Partners	License	On GitHub?
IRM-NOVA	WP3	SAP	BSD	Y
IRM-SHEPARD	WP3	IMP	BSD	Y
MaxelerOS Orchestrator	WP3	MAX	Commercial	N
SHEPARD Framework	WP3	SAP	Commercial	N
IRM-NET	WP4	EPL, IMP	BSD	Y
IRM-NEUTRON	WP4	SAP	BSD	Y
IRM-XtreemFS	WP5	ZIB	BSD	Y
XtreemFS	WP5	ZIB	BSD	Y
ConPaaS	WP6	UR1	BSD	Y
CRS	WP6	UR1, IMP	BSD	Y
logging messaging system	WP7	SAP	BSD	Y
monitoring system	WP7	SAP	BSD	Y
OpenStack Ansible	WP7	IMP	GPLv2	Y
HARNESS deployment	WP7	IMP	GPLv2	Y

Table 6.1: Software developed or extended in HARNESS.

For example, for our experiments, this project was used to configure the Grid'5000 testbed (see Chapter 5). The project also contains a Vagrant configuration file[6] so that a complete HARNESS platform can easily be created in a virtual machine running on an individual laptop or workstation.

6.3 The HARNESS Prototype

We now describe the HARNESS prototype, whose architecture is shown in Figure 6.2. The components of this architecture adhere to the HARNESS API (see Section 3.4).

The HARNESS prototype has been implemented to validate our approach, and has been deployed in two testbeds: Grid'5000, which allows us to explore a large-scale research infrastructure to support parallel and distributed computing experiments, and the Imperial testbed, which supports clusters of hardware accelerators and heterogeneous storage.

The HARNESS architecture is split into three layers: (1) a *Platform layer* that manages applications, (2) an *Infrastructure layer* that is responsible for managing resources, and (3) a *Virtual Execution layer* where the applications actually run. Next, we explain each layer in more detail.

6.3.1 The Platform Layer

The Platform layer includes the *Frontend*, the *Director* and the AM components:

- **The Frontend** is a Web server that provides a graphical interface where users can manage their applications. To deploy an application, a user must upload an application manifest and an SLO document [12]. The application manifest describes the structure of an application and the resources it needs, while the SLO specifies the functional and non-functional parameters of one particular execution

[6] `http://docs.vagrantup.com/`

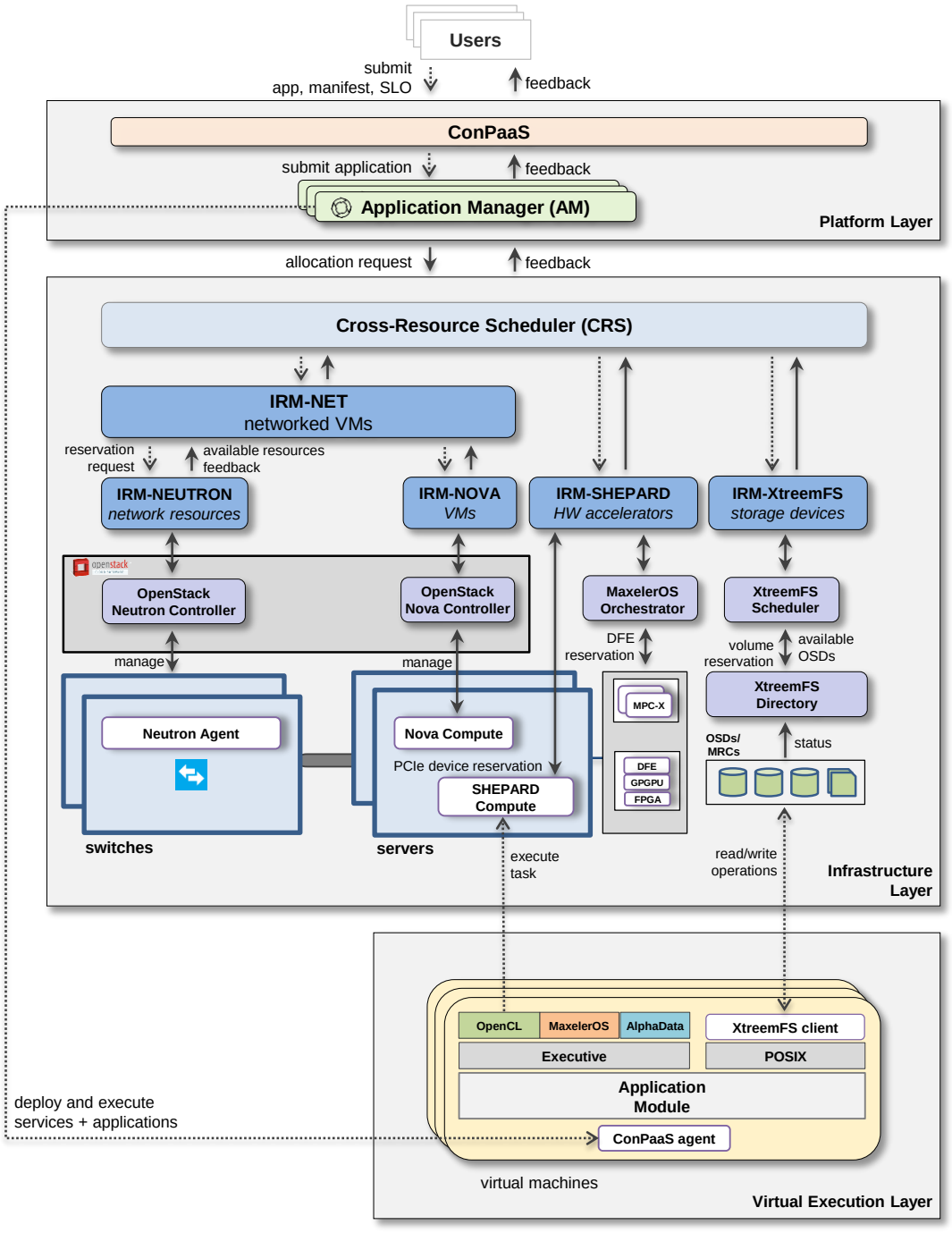


Figure 6.2: The HARNESS architecture is composed of three layers: a Platform layer that is responsible for managing applications; an Infrastructure layer where cloud resources are managed; and a Virtual Execution layer where applications are deployed and executed.

of the application. The purpose of the Frontend is to provide an intuitive graphical user interface for the end users, and translate all user actions into HTTPS/JSON requests which are sent to the Director.

- **The Director** is in charge of authenticating users and instantiating one AM for each application instance submitted by a user. Once it has created an AM instance, it forwards all subsequent management requests about this application to the AM in charge of the application.
- **The Application Manager** is in charge of controlling the execution of one particular application [13]. It is a generic and application-agnostic component, and thus a new AM does not need to be developed for every new application. The AM operates in a virtual machine that is provisioned using the HARNESS cloud resources. This virtual machine contains a specific program in charge of interpreting application manifests and SLOs, building performance models for arbitrary applications, choosing the type and number of resources that an application needs to execute within its SLO, provisioning these resources, deploying the application's code and data in the provisioned resources, and finally collecting application-level feedback during and after execution.

Whenever the Director or an AM needs to provision resources on the HARNESS platform (either to create a new application manager or to run an actual application), it sends a resource provisioning request to the Infrastructure layer.

6.3.2 The Infrastructure Layer

The Infrastructure layer is in charge of managing all cloud resources and making them available on demand. Its key components are the CRS and the *Infrastructure Resource Manager*s (IRMs). These correspond respectively to the top-level HARNESS resource manager, and the child HARNESS resource managers described in Section 3.2. In particular, the CRS handles high-level requests involving multiple resources, while the IRMs are responsible for translating agnostic management requests into resource-specific requests.

The currently implemented IRMs are: *IRM-NET*, *IRM-NOVA*, *IRM-NEUTRON*, *IRM-SHEPARD* and *IRM-XtreemFS*. Beneath the IRMs, we have components that manage specific resources, which include OpenStack, SHEPARD, MaxelerOS orchestrator and the XtreemFS scheduler.

- **The Cross-Resource Scheduler** is in charge of handling resource provisioning requests [13]. In particular, it processes single-resource requests and requests for a *group* of heterogeneous resources, with optional placement constraints between resources. For example, an application may request one virtual machine and one FPGA such that the two devices are located close to each other. It uses the network proximity maps provided by IRM-NET, and decides which set of physical resources should be chosen to accommodate each request. Once this selection has been made, it delegates the actual provisioning of the resources to corresponding IRMs. Each IRM is in charge of managing some specific type of heterogeneous resource, including VMs, GPGPUs, FPGAs, storage and network devices.
- **The Nova Resource Manager** (IRM-NOVA) is in charge of managing virtual machines running on traditional server machines. It is a thin layer that converts resource-agnostic provisioning requests issued by the CRS into OpenStack Nova requests. Managing the creation of virtual machines over a pool of physical machines is a well-studied problem, so we rely on the popular and well-supported OpenStack Nova[7] to supply this functionality.

[7] `https://www.openstack.org/`

- **The SHEPARD Resource Manager** (IRM-SHEPARD) is in charge of managing hardware accelerator resources: GPGPUs, FPGAs and DFEs. It therefore translates resource-agnostic provisioning requests issued by the CRS into requests for locally installed accelerators and networked DFEs via the MaxelerOS Orchestrator.
- **The Networked Resource Manager** (IRM-NET) provides the CRS with up-to-date maps of the physical resources that are part of the cloud. In particular, this map contains network proximity measurements realised pairwise between the physical resources, such as latency and available bandwidth. This information allows the CRS to service allocation requests with placement constraints, such as allocating two VMs with a particular latency requirement. This component also handles bandwidth reservations, allowing virtual links to be allocated. Finally, IRM-NET supports subnet and public IP allocations by delegating these requests through IRM-NOVA and IRM-NEUTRON. In particular, users can request one or more subnets and assign VMs to them, and also assign public IPs to individual VMs.
- **The Neutron Resource Manager** (IRM-NEUTRON) is in charge of managing subnet and public IP resources. It is a thin layer that converts resource-agnostic provisioning requests issued by the CRS into OpenStack Neutron requests.
- **The XtreemFS Resource Manager** (IRM-XtreemFS) is in charge of managing data storage device reservations. It therefore translates resource-agnostic provisioning requests issued by the CRS into concrete requests that can be processed by the XtreemFS Scheduler.

The following components are part of the infrastructure and handle specific types of resources:

- **OpenStack Nova Controller** is the interface by which IRM-NOVA can request the creation/deletion of virtual machines on specific physical machines. It is a standard OpenStack component that we adapted to accept the placement decisions made by the CRS. The OpenStack Nova controller operates by issuing requests to the OpenStack Nova Compute components installed on each physical host.
- **OpenStack Nova Compute** is a daemon running on each physical host of the system, and which controls the management of virtual machines within the local physical machine. It is a standard OpenStack component.
- **The MaxelerOS Orchestrator** supports the allocation of networked DFEs located in MPC-X chassis. The MaxelerOS Orchestrator provides a way to reserve DFEs for IRM-SHEPARD. These accelerators will then be available to applications over the local network.
- **SHEPARD Compute** is a daemon running in each physical host, which provides information to IRM-SHEPARD about available hardware accelerators installed in the physical host, and performs reservation of those resources.
- **OpenStack Neutron Controller** is the interface by which IRM-NEUTRON can request the creation/deletion of subnet and public IP resources. The OpenStack Neutron Controller operates by issuing requests to the OpenStack Neutron Agent components installed on each physical host.
- **OpenStack Neutron Agent** is a daemon running on each physical host of the system, and controls the management of several services including DHCP and L3 networking. It is a standard OpenStack component.
- **XtreemFS** is fault-tolerant distributed file system that provides three kinds of services: a *directory service* (DIR), *metadata and replica catalogues* (MRCs), and *object storage device*s (OSDs) [15, 20].

The DIR tracks status information about the OSDs, MRCs, and volumes. The volume metadata is managed by one MRC. File contents are spread over an arbitrary subset of OSDs.

- **The XtreemFS Scheduler** handles the reservation/release of data volumes to be used by the HARNESS application [11]. Data volumes are characterised by their size, the type of accesses for which it is optimised (random vs. sequential), and the number of provisioned *input/output operations per second* (IOPS). The XtreemFS Reservation Scheduler is in charge of ensuring data volume creation such that these requested properties are respected. The actual creation/deletion of data volumes is handled using the XtreemFS DIR, while the actual data and meta-data are respectively stored in the OSDs and MRC.

6.3.3 The Virtual Execution Layer

The Virtual Execution layer is composed of reserved VMs where the application is deployed and executed. In addition to the application itself, the VMs contain components (APIs and services) that support the deployment process, and also allow the application to interact with (reserved) resource instances. These components include:

- **The ConPaaS Agent**, which performs management actions on behalf of the AM, configuring its VM, installing code/data resources such as GPGPUs, FPGAs and XtreemFS volumes, configuring access to heterogeneous resources, starting the application, and finally collecting application-level feedback during execution.
- **The Executive** is a process that, given a set of provisioned heterogeneous compute resources, selects the most appropriate resource for a given application task [9].
- **The XtreemFS client** is in charge of mounting XtreemFS volumes in the VMs and making them available as regular local directories [11].

7 Potential Impact of the Results

7.1 Who can benefit from HARNESS

The HARNESS approach provides a number of significant benefits to both cloud tenants and cloud service providers.

1. It allows cloud tenants to take advantage of different types of hardware. This has the potential to attract a broader set of users to the cloud, such as automated trading or large-scale scientific applications, thus further expanding the reach of the cloud market.

2. It is beneficial for existing and new applications because it enables them to improve their performance significantly. For example, platforms for data analytics processing, such as Apache Hadoop MapReduce[1], are able to exploit our services (e.g. in-network processing) to increase application throughput. This increase in speed permits new applications that are too demanding for today's cloud platforms. Moreover, there is the added benefit of reducing the equipment footprint as well as power and energy consumption from using appropriate hardware acceleration technologies.

3. We expect the new opportunities and business models enabled by HARNESS to be particularly attractive for SMEs offering cloud-based services. They will have the opportunity to differentiate themselves, potentially expanding their market shares against big players such as Amazon or Microsoft. HARNESS technologies will allow different cloud providers to create differentiated technical offers, e.g., by specialising for certain kinds of applications, while retaining standard cloud interfaces to avoid customer lock-in. Having created a common software substrate, tenants will be free to move seamlessly from one cloud provider to another, thus creating a more open market and making it easier for new companies to enter the cloud business.

4. HARNESS has the potential to fundamentally change the relationship between providers and tenants with a positive impact for both. In our model, tenants are relieved from the burden of providing a custom implementation and a low-level specification for the heterogeneous resources needed. This will also make it possible for those without expertise in accelerator technologies to benefit from such resources in the cloud. At the same time, providers are likely to gain the flexibility dynamically decide on the amount of resources—in one technology or in multiple technologies—for a given tenant's request based on current allocation state and user constraints.

5. The HARNESS approach will enable resource allocation to be decided by the cloud provider rather than by the tenants. This arrangement allows the cloud provider to offer resource reservations at fine time granularities, giving more flexibility and control to tenants. This capability is especial for cloud platforms that include resources for hardware acceleration because it enables allocating of expensive resources for short-lived application sessions in a cost-effective manner.

[1] `http://hadoop.apache.org`

7.2 Impact on open standards

We expect the HARNESS project to contribute significantly to open standards relating to cloud computing platforms and technologies. Standardisation efforts in cloud computing are highly prominent at the moment and experience broad commercial and academic support. They promise to address issues regarding cloud interoperability and platform independence, which are seen as major obstacles to a wider adoption of cloud technologies by industry. While cloud technology adhering to an IaaS model has seen mature open standards governing virtualisation and management interfaces, we believe that the HARNESS project has a unique opportunity to contribute to early standardisation efforts in PaaS clouds.

7.3 Impact on industry

Industrial partners have specific interests in the outcomes of HARNESS.

7.3.1 Maxeler Technologies

For MAX, outcomes from HARNESS provide an opportunity to: (i) enhance their run-time system to improve resource allocation and sharing mechanisms, as well as to improve the performance and cost/performance of MAX applications; (ii) commercialise accelerated versions of RTM applications to oil companies and to provide a cut-down version to academic research; (iii) inform future product plans, particularly with regard to opportunities for heterogeneous resources in cloud environments; and (iv) grow its new ecosystem, called *AppGallery*, that helps members of MAX-UP to share their DFE designs and to rapidly develop new DFE applications.

7.3.2 SAP

The HARNESS project can make key contributions to *SAP AG* (SAP)'s cloud offerings, including leveraging heterogeneous computation, communication and storage, and managing computations across different compute platforms. In addition, SAP believes that HARNESS can make key contributions in proof-of-concepts and prototypes that could be of interest to SAP and its partners. Specifically SAP intends to:

1. leverage the SHEPARD framework to extend current PAL libraries to offer accelerated implementations for customers. Initially this will be a limited offering for choice customers but based on demand could be extended to a full cloud offering.

2. leverage the SHEPARD framework to manage task execution for the text analytics offering within the SAP HANA Preprocessor that targets Intel PHI technology.

3. integrate components of the SHEPARD framework within SAP HANA as a second class job executor. (This work depends on roadmap features being introduced in 2016).

4. extend the current Delta Merge FPGA implementation for execution on new technology from Intel with integrated FPGA technology.

8 Partners

The HARNESS consortium was a lean, well-balanced, high-profile consortium consisting of world-class, highly experienced academic and industrial research groups, and a dynamic, fast-growing, and financially successful SME. Furthermore, the makeup of the consortium represented a broad demographic of the key contributors to furthering the European innovation agenda. There were three university partners (IMP, *École Polytechnique Fédérale de Lausanne* (EPL), and UR1), one research institute (ZIB), one large enterprise (SAP), and one small enterprise (MAX). They came from four countries: The United Kingdom, France, Germany, and Switzerland. Table 8.1 provides a brief overview of the specific expertise of each partner.

From the industrial perspective, the consortium had partners deeply involved in the challenges that drove the HARNESS research. SAP owns and operates several extremely large and costly data centres, providing cutting-edge, high performance multi-tenancy services to both internal SAP business units and high-value external customers. SAP is also a customer of other data centre providers. Taken together, SAP brought experience and perspective from all sides of the problem space. The SAP group engaged in HARNESS were responsible in particular for investigating the use of specialised hardware and network devices to accelerator the performance of mission-critical computing tasks. Our SME partner, MAX, is also a prominent provider of data centre services through its MaxCloud offering. Furthermore, MAX is a leading developer of reconfigurable systems, including not only the hardware devices, but also the software tools used to program and manage them.

Table 8.2 gives provides key contacts for the project. The primary HARNESS contact is:

`info@harness-project.eu`

Partner	Expertise	Contribution
IMP	Software engineering, distributed middleware systems, industrial-strength data centre networking, field-programmable logic, reconfigurable systems, disruptive business innovation	Project coordination, management abstractions for customisable hardware, network virtualisation and resource characterisation, flexible application architectures, AdPredictor validation case
EPL	Programmable routers and networks, open source network frameworks	Programmable network management, network design to support large-scale data centre computations
UR1	Management of large-scale computing infrastructures, resource provisioning, scalable application hosting	Performance prediction, run-time application control, resource allocation decisions
ZIB	Distributed data management, novel storage devices, scalable services, autonomous computing	Storage management infrastructure, storage virtualization
MAX	Heterogeneous compute technologies, FPGAs, high-performance scientific applications	Industrial requirements, customisable hardware virtualisation, dissemination and exploitation management, RTM validation case
SAP	In-memory database technologies, grid and cloud computing, customisable hardware and networks, business intelligence applications	Industrial requirements, computation resource characterisation, data centre storage and networks, integration and demonstration, OLTP acceleration validation case

Table 8.1: Partners and their expertise.

Name	Partner	Area	E-mail
All	*All*	*All*	info@harness-project.eu
Alexander Wolf	IMP	Project Coordinator	a.wolf@imperial.ac.uk
J. Gabriel F. Coutinho	IMP	Project Architect; computational resources	gabriel.figueiredo @imperial.ac.uk
Katerina Argyraki	EPL	Networking resources	katerina.argyraki @epfl.ch
Thorsten Schuett	ZIB	Storage resources	schuett@zib.de
Guillaume Pierre	UR1	Platform	guillaume.pierre @irisa.fr

Table 8.2: Key contacts.

Bibliography

[1] ACM SIGKDD. Predict the click-through rate of ads given the query and user information. Available at `http://www.kddcup2012.org/c/kddcup2012-track2/`.

[2] Apache Software Foundation. Apache Deltacloud drivers. Available at `https://deltacloud.apache.org/drivers.html`.

[3] Apache Software Foundation. Apache Deltacloud Project. Available at `http://deltacloud.apache.org`.

[4] Apache Software Foundation. Apache Libcloud Project. Available at `http://libcloud.apache.org`.

[5] Cloud Management Working Group (CMWG). Cloud Infrastructure Management Interface (CIMI) Specification. Available at `http://www.dmtf.org/standards/cmwg`.

[6] Eucalyptus Project. Available at `http://www.eucalyptus.com/`.

[7] D. G. Feitelson and L. Rudolph. Towards convergence in job schedulers for parallel supercomputers. In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, IPPS '96, pages 1–26. Springer-Verlag, 1996.

[8] FP7 HARNESS Consortium. Monitoring and virtualisation report. Project Deliverable D3.2, 2014.

[9] FP7 HARNESS Consortium. Experimental Hardware Prototype and Report (initial). Project Deliverable D3.4.1, 2013.

[10] FP7 HARNESS Consortium. Final implementation evaluation report and release of data storage component. Project Deliverable D5.3, 2015.

[11] FP7 HARNESS Consortium. Data storage component (initial). Project Deliverable D5.4.1, 2013.

[12] FP7 HARNESS Consortium. Application characterisation report. Project Deliverable D6.1, 2013.

[13] FP7 HARNESS Consortium. Heterogeneous platform implementation (initial). Project Deliverable D6.3.1, 2013.

[14] FP7 HARNESS Consortium. Heterogeneous platform implementation (updated). Project Deliverable D6.3.3, 2015.

[15] F. Hupfeld, T. Cortes, B. Kolbeck, J. Stender, E. Focht, M. Hess, J. Malo, J. Marti, and E. Cesario. The XtreemFS architecture—A case for object-based file systems in Grids. *Concurrency and Computation: Practice and Experience*, 20(17):2049–2060, 2008.

[16] Mendeley. Available at http://www.mendeley.com/.

[17] Open Grid Forum (OGF). Open Cloud Computing Interface (OCCI) Specification. Available at http://occi-wg.org.

[18] Organization for the Advancement of Structured Information Standards (OASIS). Cloud Application Management for Platforms (CAMP) v1.1. Available at http://docs.oasis-open.org/camp/camp-spec/v1.1/camp-spec-v1.1.html.

[19] Organization for the Advancement of Structured Information Standards (OASIS). Topology and Orchestration Specification for Cloud Applications (TOSCA) v1.0. Available at http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.html.

[20] J. Stender, M. Berlin, and A. Reinefeld. XtreemFS – a file system for the cloud. In D. Kyriazis, A. Voulodimos, S. V. Gogouvitis, and T. Varvarigou, editors, *Data Intensive Storage Services for Cloud Environments*. IGI Global, 2013.

[21] M. Stillwell, F. Vivien, and H. Casanova. Virtual machine resource allocation for service hosting on heterogeneous distributed platforms. In *Proceedings of the 26th International Parallel and Distributed Processing Symposium*, May 2012.

[22] Storage Networking Industry Association (SNIA). Cloud Data Management Interface (CDMI) v1.1.1. Available at http://www.snia.org/sites/default/files/CDMI_Spec_v1.1.1.pdf.

[23] Storage Networking Industry Association (SNIA). S3 and CDMI. Available at http://www.snia.org/sites/default/files/S3-like_CDMI_v1.0.pdf.