



SIMPRO

Computational Resource Management Systems

Authors: Janne Keränen, Juha Kortelainen, Marko Antila

Confidentiality: Public

Report's title		
Computational Resource Management Systems		
Customer, contact person, address		Order reference
Tekes, Matti Säynätjoki Kyllikinportti 2, P.O. Box 69, FI-00101 Helsinki, FINLAND		Tekes: 40204/12
Project name		Project number/Short name
Computational methods in mechanical engineering product development		78634/SIMPRO
Authors		Pages
Janne Keränen, Juha Kortelainen, Marko Antila		60
Keywords		Report identification code
high-performance computing, resource management system, computing, simulation		VTT-R-01975-15
Summary		
<p>Industrial processes require sufficient computational resources and their high availability, along with right tools and practices. Often, the amount of computational resources is not the problem, but the easy and efficient utilisation of the resources. To tackle this challenge, a class of information systems, computational distributed resource management systems (DRMSs), has been developed. These systems aim to optimise the computational load balancing and resource availability for the users letting the user to focus on the actual challenge in hands. The trend in this sector is from managing the resources of a single system, such as a computational server, towards distributed large computational resources in form of computational grids and cloud computing. This report discusses different software tools to improve the usability and utilisation of computational resources.</p> <p>There are different tools to utilise office laptop and desktop computers idle resources, dedicated computer clusters, grids of heterogeneous computational hardware, and cloud computing resources. Some DRMSs can deal with several use scenarios, but none of them can handle all the scenarios well, and thus there is multitude of different workable systems. In this report, most of the major computational DRMSs are discussed, and use experiences are covered for the selected tools: Grid Engine, SLURM, HTCondor, and Techila. Grid Engine and SLURM are designed for cluster computation, i.e. for systems from small to vast collections of dedicated computers connected with a special network. HTCondor's and Techila's strengths are in workstation grids, i.e. utilisation of idle computational resources of the workstations and laptops in a heterogeneous computer network or cloud computing resources.</p>		
Confidentiality	Public	
Tampere 14/8/2015		
Written by	Reviewed by	Accepted by
Janne Keränen, Research Scientist	Kari Tammi, Research Professor	Johannes Hyrynen, Head of Research Area
VTT's contact address		
VTT Technical Research Centre of Finland, P.O. Box 1000, FI-02044 VTT, FINLAND		
Distribution (customer and VTT)		
Tekes/Matti Säynätjoki, 1 copy VTT/archive, 1 copy		
<p><i>The use of the name of the VTT Technical Research Centre of Finland (VTT) in advertising or publication in part of this report is only permissible with written authorisation from the VTT Technical Research Centre of Finland.</i></p>		

Contents

1	Pre-study of computational management systems.....	4
1.1	Introduction.....	4
1.1.1	Terminology.....	5
1.1.2	Principles.....	6
1.2	Requirements for resource management systems.....	6
1.2.1	Resource management in computational clusters.....	8
1.2.2	Managing workstation, desktop, and server computer resources.....	10
1.2.3	Grid computing and resource management.....	11
1.2.4	Distributed resource management application API.....	12
1.3	Open source solutions for resource management.....	12
1.3.1	Resource management systems.....	12
1.3.1.1	Grid Engine.....	12
1.3.1.2	HTCondor.....	15
1.3.1.3	SLURM.....	16
1.3.1.4	TORQUE.....	17
1.3.2	Grid management systems.....	17
1.3.2.1	Globus Toolkit.....	17
1.3.2.2	GridWay.....	18
1.3.2.3	Bosco.....	18
1.4	Other solutions for utilising remote computational resources.....	19
1.4.1	Techila.....	19
1.4.2	Azure cloud service together with Techila.....	20
2	Experiences with resource management systems.....	22
2.1	Grid Engine in a cluster system.....	22
2.1.1	Description of the computational environment “the Doctor”.....	22
2.1.2	Resource management in the cluster.....	22
2.1.2.1	Operation of SGE in cluster environment.....	23
2.1.2.2	SGE configuration in “the Doctor” cluster.....	24
2.1.3	Practical examples and how the systems works.....	25
2.1.3.1	Using OpenFOAM in the SGE cluster.....	25
2.1.3.2	Using Abaqus in the SGE cluster.....	28
2.1.3.3	Elmer in SGE cluster.....	30
2.1.4	Lessons learned with the Grid Engine system.....	31
2.1.5	Future steps for system development in “the Doctor”.....	32
2.2	SLURM in a cluster system.....	33
2.2.1	Description of the computational environment Sisu.....	33
2.2.2	Resource management in Sisu.....	33
2.2.3	Practical examples with Elmer.....	33
2.2.4	Lessons learned with the SLURM system.....	34
2.3	HTCondor in a workstation and PC network.....	34
2.3.1	Description of the computational environment.....	34
2.3.1.1	Configuration of the computational pool.....	35
2.3.1.2	Using the computational pool.....	38

2.3.1.3	How HTCondor treats job files.....	39
2.3.2	Practical examples and how the systems works	39
2.3.2.1	Example 1: Simple single simulation run in the local computer	39
2.3.2.2	Example 2: Single simulation run in the computational pool with concurrency limits.....	40
2.3.2.3	Example 3: Running HTCondor jobs from DAKOTA.....	41
2.3.2.4	Example 4: Running DAKOTA as an HTCondor job	42
2.3.2.5	Example 5: Local grid computing approach, a “fire-and-forget” scenario	43
2.3.3	Lessons learned with the HTCondor system	44
2.3.4	Future steps for local HTCondor system development.....	44
2.4	Techila in a workstation and Azure network	45
2.4.1	Overview.....	45
2.4.2	Launch of the grid computing	46
2.4.3	Techila test cases overview	46
2.4.4	Test case A.....	46
2.4.5	Test case A results	47
2.4.6	Test case B.....	49
2.4.7	Conclusions and summary.....	51
3	Acknowledgements.....	51
	References.....	52
	APPENDIX A: Running HTCondor jobs from DAKOTA	56
3.1	Details for example 3: Running HTCondor jobs from DAKOTA.....	56
3.2	Details for example 4: Running DAKOTA as a HTCondor job	57
3.3	Details for example 5: Local grid computing approach, a “fire-and-forget” scenario	58

1 Pre-study of computational management systems

1.1 Introduction

The use of computational methods, e.g. computer-based modelling and simulations, has become a standard approach in engineering. The design of complex products, such as passenger cars, heavy-duty vehicles, or aeroplanes, is practically impossible without the use of computational tools. In research, computational science has turned into the third pillar of scientific inquiry, together with theory and experimentation [1-3]. Successful application of computational methods requires mastering the physics that is involved in the case, understanding the mathematics behind the models of the physics, and understanding the numerical and software implementation of the mathematical model. All these steps affect the end results and success of the computation.

When computational methods and tools are applied in an engineering process, the efficiency of the process becomes crucial. Being able to perform the computation is not enough, but the results have to be produced at the right time and they have to be in the right form. Managing computationally heavy methods, such as structural analysis and multi-physics using Finite Element Methods (FEM) or Computational Fluid Dynamics (CFD), in industrial processes requires sufficient computational resources and high availability of these resources. For that, application of right tools and practices is important. The step to utilise computational tools often begin as running single analyses manually in the local computing environment, such as a workstation. While the requirements for the results of the computational studies increase, the level of automation in the process also increases. The tools for the automation are scripting and utilising design analyses and optimisation. This, on the other hand, increases the need for computational resources. As a result, the local computing facilities, such as the user's own workstation, are not enough anymore.

The traditional alternative for more computational resources has been to use computational servers or supercomputers. The uniting factor in them is that they both are *shared memory* systems, i.e. the memory of the system is shared by all CPU resources.¹ Presently, the price of PC hardware have dropped such a way that all the modern computation resources are built on multitude of PC-type hardware connected via fast communication channel, such as local network. Thus, the CPU's do not share the same memory or, in other words, the systems have *distributed memory*. If dedicated hardware for computation is used, including a fast dedicated network between the separate computers, the system is called a *computation cluster* and the computers are called *nodes* [5]. Sometimes the system combined from separate but connected computation clusters is called a *computation grid*. If normal desktop or laptop computers' idle resources are utilised, the system can be called a *workstation grid*. Nowadays, computation power is sold as service in pay-per-use idea and the connection between the resources and used hardware is not explicit; this is called *cloud computing*.

This report discusses the management of computation in the shared-memory, distributed memory, computation cluster, grid, and cloud systems. The management is handled by dedicated software tools called *distributed resource management systems* (DRMS)² or *meta-schedulers*. In addition, the requirements they set to computational engineering are discussed. Some of the available open source DRMS and meta-scheduling systems are introduced, and experiences of the selected systems are examined.

¹ Distributed memory approach have been utilised also in supercomputer from 1980s [4].

² Synonyms for DRMS are e.g. job scheduler, batch system, distributed resource manager, and workload automation.

1.1.1 Terminology

The field of computational resource management is wide and the terminology is varying. Still, similar concepts are present across the field, and the objectives are more or less comparable for all the systems. Above all, there are two levels of computational resource management:

A distributed resource management system (DRMS) control the usage of hardware resources, such as CPU cycles, memory, disk space and network bandwidth, in high-performance parallel computing systems or e.g. workstation networks. Users request resources by submitting jobs, which can be sequential or parallel. The goal of a DRMS is to achieve the best utilisation of resources and to maximise system throughput by orchestrating the process of assigning the hardware resources to users' jobs. [6]

A meta-scheduler is a software system that provides a virtual layer on top of heterogeneous computational grid middleware, DRMSs, and resources. It is used, for example, in grid computing, i.e. in computing that supports workload execution on computing resources that are shared across a set of collaborative organisations. Meta-schedulers typically enable end-users and applications to compete over distributed shared resources through the use of one or more instances of the same meta-scheduler, in a centralised or distributed manner, respectively. [7]

Thus, a DRMS is used to manage local computational resources, typically only one resource pool. In contrast, meta-schedulers are used for scheduling in one level upper: meta-schedulers manage groups of resource pools, which themselves are managed by DRMSs. A group of resource pools can consist of several computational clusters. The above division is somewhat artificial, as meta-schedulers can also be seen as a subgroup of DRMSs.

From the hardware and security point of view, the field of resource management can be organised as follows:

Cluster and workstation network management: The remarkable feature is that the resources are known by the users and the management and maintenance is done by local administrators. The network can be assumed, at least in most cases, to be trusted. This is a typical use of DRMSs.

Management of the local resource pools: This is similar to the previous category; except that the users do not necessarily know the resources anymore and they may not have direct access to the resources.

Grid management: In this architecture, the resources are managed and maintained by local administrators of several organisations. The computational resources are only partially available for external users. In this approach, meta-scheduling is typically utilised.

Cloud computing and management: Computational resources are provided as a service, and depending on the conditions of the service, the user may be responsible for installing and maintaining all the low-level computational facilities, such as the operating system and all the necessary software applications.

These divisions are not ordered and are not exclusive, but may help the reader to understand, on the one hand, the common features and concepts of the different systems, and, on the other hand, the differences and needs for different IT systems and approaches.

Further, let us define some terminology common for computational resource management systems:

Worker, computation node: Computational cluster or grid include one (or few) frontend computers (e.g. master nodes, servers), which handle the resource management. Addi-

tionally, the system also includes workers or computational nodes which are the computers doing the actual computations.

Shared memory: A multiprocessing design where several processors can access the same memory, i.e. the memory of the computer system is shared by all CPU resources.

Checkpoint: Checkpoint is a snapshot of the application's state, which can be used to restart the application from that state. Restart can be performed e.g. in another computer in case of a failure in the first computer. The resource management systems typically ask for a checkpoint from the application, i.e. start the checkpoint procedure.

Embarrassingly parallel: A problem type in parallel computation, for which little or no effort is required to separate the problem into a number of parallel tasks, is called embarrassingly parallel. In other words, it means there are little or no connections between computation processes during the computation, so the processes are almost independent and do not share any data or use common memory allocation, or the sharing and common memory portion is minor.

1.1.2 Principles

Taking computational resource management systems into use requires planning and deciding the use scenarios, architecture, supporting systems, access control and policies, among other things. From the end user point of view, these are often unessential points compared to the computational challenges in hand. Thus, some guidelines may be useful when planning to setup computational resource management. The following principles are proposed for building new resource management strategies:

“Keep it simple”: This applies both for the end user point of view as well as for the administration point of view. The system should be easy and intuitive to use for the end users, otherwise the advantage will not be fully utilised. For the administration, systems with clean architecture and straightforward configuration are simpler to keep safe and running.

“Keep it flexible”: The needs and requirements in computational engineering are changing rapidly. New software applications are taken into use and new versions of old software are utilised. Thus, new features are introduced, and to utilise the existing resources efficiently, the infrastructure has to follow the changes. In addition, requirements for short term special cases, such as a project that has high demands on computational resources, may address needs for temporary changes in the system.

“Make it modular”: Simple low-level architectures are easier to maintain and well-designed configurations can be easily copied to new installations. This improves the efficiency of the administration and makes the overall system more reliable and flexible.

“Make it extensible”: As the users learn to utilise the resources through the resource management systems, the request for larger resources will usually follow. Well-designed systems and modules are easy to copy and extend with existing maintenance resources.

1.2 Requirements for resource management systems

There are several definitions for distributed resource management systems. Even the concept is defined with several names, such as *resource management system* and *workload management system*. Throughout this report, the concept is called *distributed resource management system* (DRMS) and it is used for a system that provides the following functionality:

- job submission system;
- queuing system;

- load balancing functionality; and
- job progress monitoring and execution control functionality.

The main feature of DRMS is to improve the utilisation of the computational resources in the organisation and to free the users from taking care of their computational jobs in detail. The computational resources include the cluster systems, computational servers, workstations, and even desktop computers and laptops. Even though the cost for floating-point operations per second (FLOPS) in computing is decreasing, the need for more computational resources is increasing faster and all the available resources should be utilised efficiently. Still, the working conditions at a workstation should be guaranteed for the main user of that system. Further, parallelisation of computational software and hardware has made the management of computational resources more complex than in past.

In Figure 1, the general structure of the computational resources for VTT's local organisation is illustrated from the computational resource management point of view. In the figure, the computational pools are local computational resources, such as research areas at VTT (an organisational unit of 100–150 people). Inside a computational pool, the computational resources may include desktop computers, workstations, computational servers, and cluster systems. A cluster system is in fact a sub-pool in the computational pool, because it has its own computational resource management system. Due to its special role (e.g. users are not supposed to log in to one computational node in a cluster system and thus use its resources), cluster systems are treated as special type of computers in this study. The local computational resources are typically managed using one resource management system, such as Grid Engine, Techila, HTCondor, or SLURM. Figure 2 presents the needed software and system components for a local resource pool using HTCondor system. In each system component, a set of server and client processes are running to fulfil the different roles needed by the overall system.

When local resource pools or management sets—whatever the locally managed resources are called—are connected, a computational grid is formed. The technical difference between locally managed resources and grids is fuzzy, and often the distinct feature of a grid is that it connects the computational resources of several organisations or organisation units, and only some of the resources are opened for the other users of the grid. In addition, the maintenance of the local resources is done by the local administrators.

A *computer cluster* consists of a set of connected computers that work together so that in many respects they can be viewed as a single system. Each computer in the cluster is called a *node*, running its own operating system and, likely, suitable server software processes. Typically, a computer cluster is used through a single interface computer and a user does not necessarily know which nodes of the cluster they are utilising.

The essence of a cluster system is to provide a single system image (SSI) for all the cluster nodes, providing an illusory feeling to the user that the cluster is one single computer system with all the power of the constituent nodes as one powerful resource. SSI can be offered by [8]:

- hardware layer;
- operating system kernel; and
- software applications.

A hardware solution allows user to view the system as shared memory system, i.e. a virtual shared memory among cluster nodes is utilised by means of intermodal space mapping [8]. Alternatively, scheduling and load balancing can be included in the operating system kernel, like in SCO UnixWare, Sun Solaris MC, and GLUnix. The application solutions can be founded on system management tools, runtime systems (e.g. parallel file system), or middle-

ware modules (e.g. resource management and scheduling software). In this study, we concentrate on the middleware application solution.

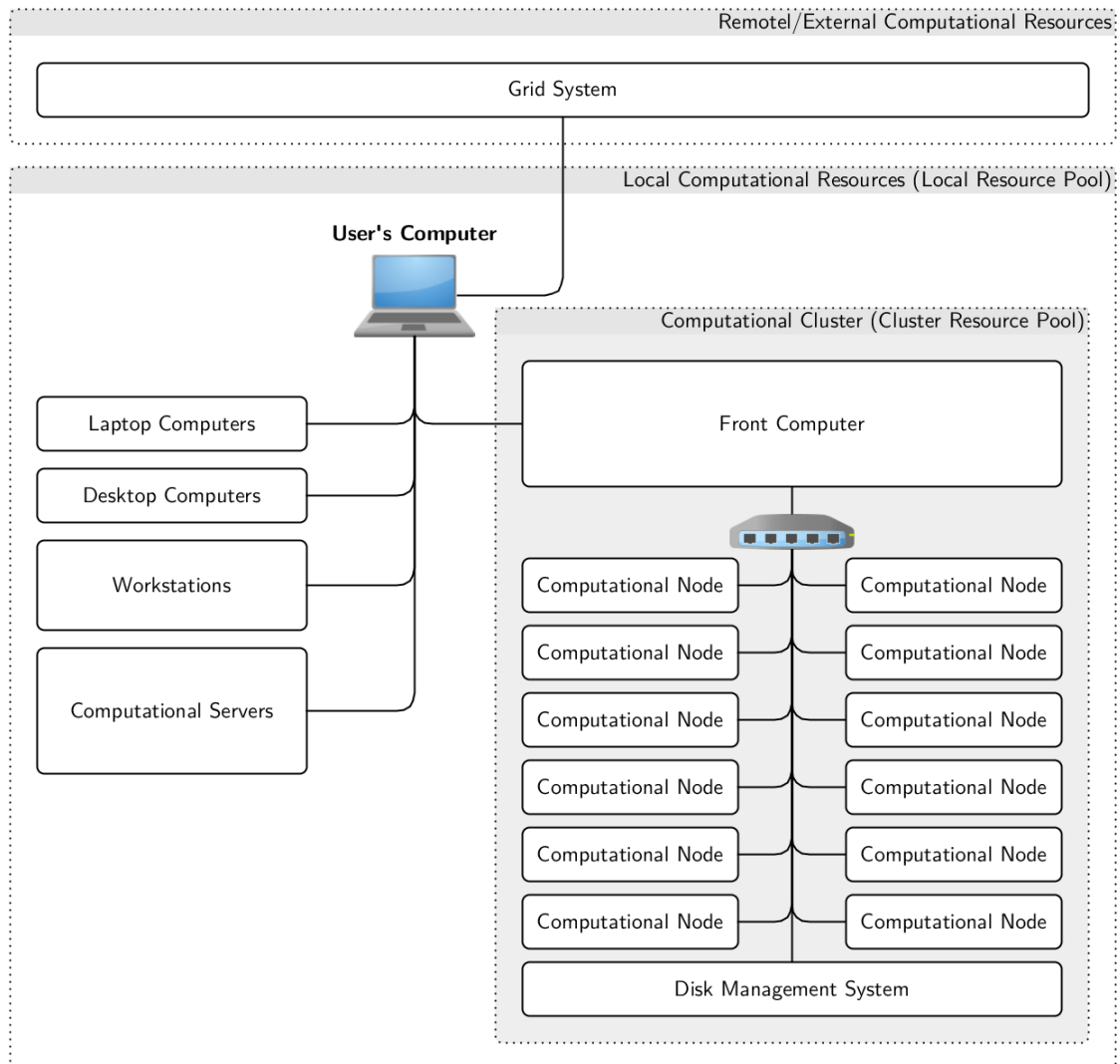


Figure 1: Illustration of the computational resource offering in an organisation.

1.2.1 Resource management in computational clusters

Typically, the activities of the computation cluster nodes are handled by so-called *clustering middleware*, a software layer above the operating system layer and below the user interface layer [8]. The middleware sits atop the computation nodes and allows the users to treat the cluster as one large computing unit—i.e. from the user's point-of-view, all the computers' resources (CPU, memory, and storage) seem to work together as one single computer. The middleware should allow the user to utilise the cluster easily and effectively, without the knowledge of the details of the underlying system architecture [8]. This middleware includes a batch scheduling, that is above called a distributed resource management system (DRMS), and also a parallel programming environment to support parallel computation. In traditional solutions, a computer cluster DRMS has centralised management and it works in a client-server fashion. It is distinct from other approaches such as grid computing DRMSs, which also uses many nodes, but with a far more distributed nature. [9]

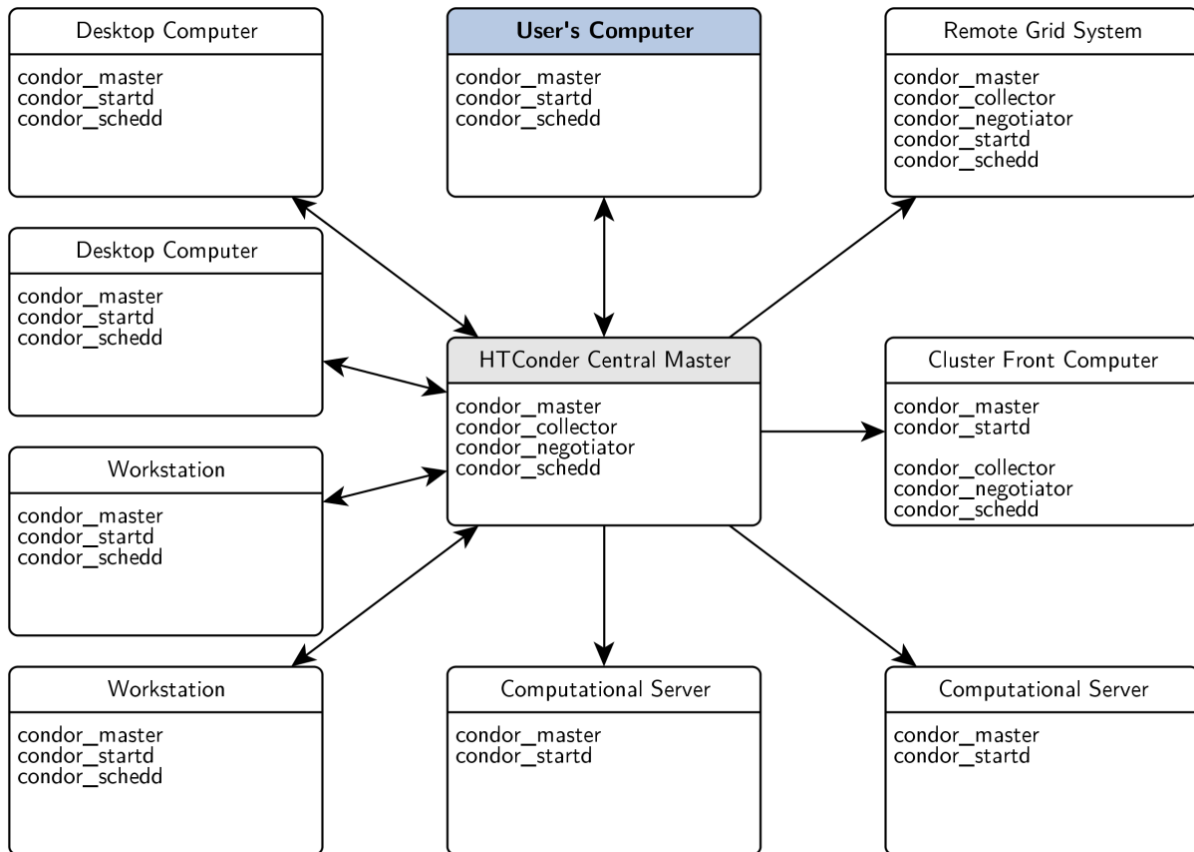


Figure 2: Illustration of the architecture of an HTCondor local resource pool.

The extendibility of cluster systems is presently prominent. Although a cluster may consist of just a few personal computers connected by a simple network, the cluster architecture may also be used to achieve high performance. The list of the fastest supercomputers often includes many clusters, e.g. the world's fastest computer system in June 2014, Tianhe 2 [10], has a distributed memory and cluster architecture. From the conventional point-of-view, clusters are not supercomputers, because they do not have shared memory, but presently the difference has diminished, as many supercomputer architectures have also abandoned shared memory.

In addition to scientific computations, computer clusters are used, for example, in web server clusters and high-availability clusters; the latter operates by using redundant nodes in reserve for a component failure. Here we concentrate on high-performance computing (HPC) clusters, i.e. the computation-intensive purposes rather than handling IO-oriented or failover operations.

As discussed above, operation of a computer cluster relies on two separate software tools (so-called clustering middleware): message passing interfaces and the cluster management tools (i.e. DRMSs). Here, message passing is a form of communication used in parallel computing between the computing nodes. As the number of nodes in a cluster has rapidly increased, the complexity of the communication subsystem has exploded. Furthermore, specifically the extent of the inter-node communication differentiates cluster computation from grid computation, where the communication is severely more limited.

In last decades, the two most common solutions for communication between cluster nodes have been the Parallel Virtual Machine (PVM) and the Message Passing Interface (MPI). However, MPI has now emerged as the de facto standard for message passing on computer clusters. PVM is free software enabling distributed processing and grid computing by a set of

software libraries, whereas MPI is a specification rather than a specific set of software libraries. Presently, MPI is the most common communication model enabling parallel programs to be written e.g. in C, C++, Fortran, Java, and Python [8]. The most popular general MPI implementations are Open MPI³ and MPICH. Open MPI aims to use the best ideas and technologies from other older MPI projects and to create one world-class open source MPI implementation that excels in all areas [11,12]. Most of the software compiler and development tool vendors, such as Intel and Microsoft, as well as complete computer system vendors, such as Oracle and IBM, provide their own optimised MPI implementations.

Presently, the most popular operating system for the HPC computer clusters is Linux, and all the main DRMSs support at least it. Further, many other UNIX and UNIX-based operating systems are supported. For Windows, HTCCondor [13] supports natively (but limitedly) all contemporary Windows versions, Grid Engine [14] supports Windows through different Unix subsystem under Windows, and TORQUE [15] works within Cygwin environment. Oppositely, SLURM [16] do not support Windows platform at all.

1.2.2 Managing workstation, desktop, and server computer resources

Workstations and desktop computers are not the primary target systems of distributed resource management systems, but they can form cluster-like groups, or *workstation or desktop grids*. There are many challenges in applying resource management systems for workstation and desktop networks. One of those is slower network connection between computational nodes compared to dedicated cluster systems. Further, the workstations (and also computational servers) are also used interactively from the desktop, and this hinders the cluster usage. In addition, workstation and desktop system management may be heterogeneous and e.g. availability and configuration of software applications in the workstation network can cause additional challenges. On the other hand, locally inside a multi-user workstation or computation server, DRMSs could be used for queuing, but the usefulness of this is questionable. The utilisation of DRMS is understandable only if there is surplus of users and some prioritisation is necessary or there are e.g. less software application licenses than there are computing resources (processors or processor cores) and computational cases.

Workstation or desktop grids are computer grids that use the idle CPU resources of the workstations [17]. The use of desktop or laptop computer instruction cycles that would otherwise be wasted, e.g. at night or during breaks, is called e.g. *cycle-scavenging*, CPU scavenging, cycle stealing, or shared computing. This is natural usage for a network of workstations for HPC usage. In practice, participating computers also donate some supporting amount of disk storage space, RAM, and network bandwidth, in addition to raw CPU power. Workstation grid DRMSs are able to detect that a machine is no longer available (i.e. to detect that the main user of the workstation starts to use the computer from the desktop, e.g. a key press or mouse movement is detected), and are able to transparently produce a checkpoint⁴ and migrate a job to a different machine which would otherwise be idle. Further, these workstation grid middleware tools can transfer the job's data files on behalf of the user and no shared file system across machines is necessarily needed. Examples of tools that embody, apart from basic DRMS features, above workstation grid specialities contains HTCCondor [13], Techila [18], BOINC [19], and XtremWeb⁵ [22]. Some of these workstation grid tools are compatible with volunteer computing, in which private people can give their personal computer resources to scientific research.

³ Open MPI should not be confused with OpenMP, which is an implementation of multithreading, i.e. interface for developing parallel applications for shared memory multiprocessing. In contrary, MPI is a solution for distributed memory parallelisation.

⁴ Checkpoint is a snapshot of the application's state, which can be used to restart the application from that state e.g. in another computer in case of a failure in the first computer.

⁵ Along with its derivatives XtremWeb-CH [20] and XtremWeb-HEP [21].

Heterogeneous workstation grids with different operating systems create some challenges to the operation of the grid. For example, HTCondor supports pools with workstations of different operating systems, but one job gets executed in all nodes only if there is a compatible executable for each operating system. Further, with Techila, there is a possibility to compile the executable separately in each Worker node, which enables efficient use of heterogeneous grids. There exists also research on combining cycle-scavenging and virtualisation, i.e., system where the job in Worker is run inside a virtual machine. This allows usage of different operating systems by sandboxing the job executives from the main operating system, and reduces compatibility issues between the job executable and node operating system.

The node-to-node network connection speed in the workstation grids is typically much lower than with dedicated cluster systems. Further, this connection speed is normally too slow for efficient MPI-computations. Hence, workstation grids are normally used for *embarrassingly parallel* computation, i.e., to computation for which little or no effort is required to separate the problem into a number of parallel tasks. Thus, there is none or little communication between grid nodes.

1.2.3 Grid computing and resource management

By grid computing, we mean a collection of computer resources from multiple locations used in computationally heavy computation. Computational grid is often compared to electric grid that makes electric power or, in case of computational grid, computational power easily available to the users [23]. The main difference to computer clusters is that grids tend to be more loosely coupled, heterogeneous, and computational resources may be located in and managed by several organisations. Workstation grids are a special type grid, but here we concentrate on grids built from dedicated computers. Such a grid comprises from separate computer clusters, computation servers, etc., likely having their own DRMS. Grids are often constructed with general purpose grid middleware software that is built on top of the cluster managements systems. Thus, grid resource management systems are typically above-mentioned meta-schedulers.

Informally, a computer grid is something between a computer cluster and workstation grid. The main technical differentiating issue is the connection speed between the nodes, which is slower in grids than in clusters but typically faster than in workstation grids. Thus, grid computing is favourable for parallel computing with little communication between grid nodes. This behaviour varies from grid to grid, depending on the connection speed between the nodes.

One disadvantage of computer grids is that the grid computers might not be entirely trustworthy. This can be overcome by above-mentioned virtualisation and sandboxing idea. Also, due to the lack of central control over the hardware, there is no way to guarantee that nodes will not drop out of the network at random times. The unreliability of network connection can be partly overcome by reducing need of continuous network connectivity and reassigning work units when a given node fails to report its results in expected time. Further, a computer grid does not necessarily share disk storage, but the data must be sent to the grid nodes as a data package.

Globus Toolkit can be seen as one kind of a standard of grid computing middleware. In addition to working as a front end for job schedulers, it includes also tools for e.g. security, data management, communication, and fault detection [24]. Implementations of Globus Toolkit are based on various standards and it is compatible with various DRMSs and other computing related tools, for example HTCondor, Grid Engine, SLURM, and TORQUE. For another, GridWay [25] is a Globus Alliance project aiming for a meta-scheduler allowing user to submit, monitor, synchronise, and control jobs by means of a DRMS-like command line interface.

1.2.4 Distributed resource management application API

The *Distributed Resource Management Application API* (DRMAA) [26] is a set of specifications for submission and control of jobs to cluster, grid, and cloud systems, developed by the Open Grid Forum [27]. DRMS vendors can provide a standardised access to their product through a DRMAA implementation. The API standardisation is focused on job submission, job control, reservation management, and retrieval of job and monitoring information. It is meant for meta-scheduler architects and end users for a unified access to execution resources. Thus, with DRMAA, it is possible to submit jobs to a grid with a meta-scheduler, assuming each grid is running a local compatible DRMS. Presently, Grid Engine, HTCondor, TORQUE, GridWay, and SLURM support DRMAA.

1.3 Open source solutions for resource management

1.3.1 Resource management systems

1.3.1.1 Grid Engine

Grid Engine [14] is a resource management system originally developed by Genias Software, Gridware, and Sun Microsystems. Grid Engine was available as open source software from year 2001 until Oracle Corporation acquired Sun Microsystems in 2010. After 2010, the new versions were licensed under a commercial license. In October 2013, Oracle sold the source code, copyrights, and trademarks associated with Grid Engine to Univa and cancelled the Oracle Grid Engine product. Because of its meandering history, the software is also known as CODINE, GDR, Sun Grid Engine (SGE), Oracle Grid Engine (OGE), and Univa Grid Engine (UGE), but we choose to call it SGE and Grid Engine. After Oracle Corporation announced that the new versions of SGE will be available under commercial license, several forks of the open source version were formed. At the moment, the most active Grid Engine versions are:

- Son of Grid Engine [28];
- Open Grid Scheduler [29]; and
- Univa Grid Engine Core [30].

Of the above projects, Univa Grid Engine (UGE) is the only official commercial product. The present situation, i.e. there are several forks⁶ of the open source project, has led to a situation, in which none of the projects is dominating and the community seems to be passivising.

Grid Engine includes features, such as:

- multiple scheduling algorithms enabling policy-based resource allocation;
- queuing of jobs in a cluster;
- job and scheduler fault tolerance, including job checkpoint procedure;
- MPI, PVM, and OpenMP support for parallel jobs;
- DRMAA support; and
- resource reservation.

Based on the long history of Grid Engine and its vast usage, users value SGE—above all—to be reliable. As being one of the most popular DRMS, it has been thoroughly tested during last decades in various sizes of clusters. This also tells that SGE is quite scalable; it works well in small to very large clusters. Further, it is quite flexible and easily modifiable, but because of the above-mentioned ramification of its development, development of new features is not any

⁶ A software project fork stands for existence of several parallel development branches of originally same software. Apart from parallel development branches, the developer community is commonly split. An open source software can be forked without violating licences or copyrights.

more straightforward. Further, its own graphical user interface `qmon` is quite out-of-date and supposedly not developed any more. However, the DRMAA-support makes it possible to use third-party software to submit and control jobs, along with monitoring the cluster.

The present open source versions Son of Grid Engine and Open Grid Scheduler are distributed by source code and Linux binaries, and their compatibility with other than Linux operating systems is not clear. However, the commercial version of Grid Engine from Univa (and previously from Oracle) is supporting the following platforms:

- Solaris (SPARK, x86, x64);
- Linux (x86, x64, ia64);
- Microsoft Windows (2000, Server 2003, Server 2008, XP, Vista Ultimate);
- Mac OS X;
- AIX; and
- HP-UX.

There are small differences in which computer architectures are supported by Univa and were supported by Oracle, but mainly the support is identical.

History of Grid Engine

The origin of Grid Engine (SGE) is in the Distributed Queuing System (DQS) developed by Florida State University (FSU) in early 1990s [31,32]. DQS was designed by Thomas Green as a management tool to aid in computational resource distribution across a network. Genias Software in Neutraubling, Germany, got into agreement with FSU and started to distribute and develop a commercial version of DQS from 1992–1993, named CODINE. From the beginning, Fritz Ferstl was a member of the CODINE development team in Genias.

In parallel with CODINE development in Genias, from year 1996, Global Resource Director (GRD) policy module was developed in collaboration between Genias, E-Systems (later part of Raytheon) and Instrumental Inc. Because the ownerships of the GRD module was shared, GRD (i.e. CODINE and GRD module) and the plain CODINE were co-marketed as separate products until year 2000. Raytheon marketed GRD to government customers and Genias marketed both tools to commercial accounts. Genias' business began to grow rapidly and in 1999 they merged with California-based Chord Systems and renamed the company GridWare [31].

In year 2000 Sun Microsystems acquired GridWare. As part of acquisition, Sun Microsystems acquired all rights to GRD for compensation to Raytheon. Sun Microsystems renamed the combination of CODINE and GDR-module as Sun Grid Engine (SGE) in 2000. Raytheon retained a right to sell Grid Engine into own accounts. In early 2001, Sun Microsystems released a free version of SGE for Solaris and Linux, and in June 2001 released the source code of Grid Engine. Sun Microsystems hosted a `gridengine.sunsource.net` site which included the source code, documentation, how-to documents and a very active mailing list. The mailing list became the default support channel for many SGE users and administrators. In this time, Sun Microsystems did the major part of the implementation and distributed everything as open source under SISSL (Sun Industry Standards Source License). Sun Microsystems provided free qualified binaries and offered superior community support to organisations that adopted Grid Engine. During this open source period, Grid Engine gained a large adoption and become the most popular DRMS.

Sun Microsystems marketed commercial version which occasionally included some small modifications from the free version that were later also included in the open source version. The main benefits from the commercial SGE were mainly support, training, localisation, and

extra testing. In the middle of 2000s, the commercial version of SGE was called N1 Grid Engine (N1GE).

Oracle acquired Sun Microsystems in January 2010. By the end of 2010, Oracle announced that Grid Engine would no longer be freely available as an open source product, closed the open source community project, increased the license fees, and essentially eliminated Sun Microsystems' HPC business. The last open source version of SGE was 6.2u5. After that Oracle published two updates to the non-free proprietary Grid Engine, now named Oracle Grid Engine. However, the last update dates back to December 2010. Thus, practically Oracle stopped the development of Grid Engine already in late 2010.

During the announcement of open source project closure, Oracle officially announced [33] that pass on the torch for maintaining the open source code base to the Open Grid Scheduler (OGS) project. The main developers of OGS are Ron Chen and Rayson Ho. At the beginning of the project, Ron Che wanted to wait for Oracle to officially sunset project before adding new changes to the OGS, but then Dave Love (from the University of Liverpool) wanted to add changes and started his own project, Son of Grid Engine (SoGE). The both projects survived as open source versions of Grid Engine and both are forks of Sun Microsystems' free Grid Engine version 6.2u5.

In January of 2011, Univa announced that it had hired the core Oracle/Sun Grid Engine development team, who had worked on Grid Engine for several years, including Fritz Ferstl. Univa started its own repository for the third open source grid engine fork. However, Univa did not publish all its Grid Engine modifications as open source, but only some core patches. Thus, Univa is taking an open core approach, where the commercial version of Grid Engine (called Univa Grid Engine, UGE) will be built from the open core with additional features added by Univa. Univa tried to get the other two open source forks (i.e. OGS and SoGE) to join Univa core project, but by others this was seen as an attempt to remove the rivals. When the possibility of merging the project became even more difficult, Univa started a slandering campaign against the open source rivals and Oracle [34,35]. Further, Univa hinted to its customers that OGS and SoGE are distributing Sun Microsystems' proprietary material and giving an image that Univa is the only official continuer of SGE development, even if it had not bought the rights from Oracle yet then.

Starting from October 22, 2013, Oracle handed over the support of Oracle Grid Engine customers to Univa for the remaining term of their existing Oracle Grid Engine support contracts. Univa acquired the source code, copyrights, and trademarks associated with the Oracle Grid Engine. Thus Oracle Grid Engine do not exists anymore, which makes Univa the only commercial provider of Grid Engine.

It seems like the situation with the three separate forks of SGE will continue in near future; the Univa's acquisition of SGE source code, copyrights, and trademarks did not change the situation. Univa claims to be the official supplier of commercial Grid Engine and support for it, whereas OGS sees itself as the official open source version. There is no conflict between OGS and SoGE developers. The two versions can be seen as two close patch packages to the original 6.2u5 SGE version. Apart from Univa, also other companies, like Scalable Logic [36], sell support for SGE. Scalable Logic is presently the main developer of OGS, but their proprietary development delays the development of open source version OGS. They cannot sell the SGE core as it is SILSS licenced and Univa owns the rights to Grid Engine code base. Thus, in commercial perspective, support and development of Grid Engine is guaranteed by two rival companies, but the situation of open source versions is frail. Checking in October 2014, no updates to OGS have been published in 2014 and the last update for SoGE was in January 2014.

Differences of Grid Engine versions

The differences of the different present versions of Grid Engine, i.e. UGE, OGS, and SoGE, are not clear, as there is much accusations, disinformation, and self-praise between the organisations behind the versions. However, for all three, almost all features are common and developed by Sun Microsystems and its predecessors. Most of the updates from the Univa and open source projects are bug fixes or other technical improvements, and new features are quite rare. Further, the different parties are copying code enhancement from each other, or re-implementing the same features from scratch. Basically both the open source projects—OGS and, particularly, SoGE—are only collections of different patches to the original 6.2u5 SGE version. Their patch collection is quite identical and, thus, their differences are slight. However, Univa has (or at least advertises to have) multiple new features unique to their version only and they are not publishing that part of the code as open source.

One major feature that Open Grid Scheduler project implemented first was the hwloc topology binding, i.e. user can specify the processor cores used for the job with the hwloc package, instead of Sun Microsystems' deprecated PLPA implementation. Hwloc helps applications to gather information about computing hardware so as to exploit it accordingly and efficiently. Hwloc supports newer hardware architectures and more operating systems than old PLPA implementation. However, Univa re-implemented this and went further by using topology masks to allow users to reserve some abstract computer power instead of specific cores. This allows equally utilising different hardware in the same cluster, as UGE translates the abstract computer power binding to the number of cores on the specific node [37].

1.3.1.2 HTCondor

High Throughput Condor (HTCondor) is a resource management system developed at the University of Madison-Wisconsin, USA. The website of the HTCondor [13] defines the software as “a specialised workload management system for compute-intensive jobs”. HTCondor provides a job queuing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. Users submit their serial or parallel jobs to HTCondor, HTCondor places them into a queue, chooses when and where in the computational environment (called computational pool) to run the jobs based upon a policy, monitors the progress of the jobs, and informs the user when the jobs are completed. The use and administration of HTCondor is described in the extensive manual [38].

What differentiates HTCondor from most other DRMSs, such as SGE and PBS, is that HTCondor also works with workstation grid middleware, i.e. it supports cycle-scavenging, transparent process checkpoint procedure and migration of a job to a different machine, and transferring the job data. Thus, unlike traditional DRMSs, HTCondor can effectively utilise non-dedicated machines to run jobs, i.e. workstations that are currently not being used (no keyboard activity, low load average, etc.). HTCondor also supports workstation grids with different operating systems, but a job gets executed in all nodes only if there is a compatible executable for each operating system. Another difference from most other DRMSs is that the operation of HTCondor does not rely on named computational queues. When submitting a job into the HTCondor pool, the user does not have to think about what computational queue (named resource) to utilise, but instead just submits the job into the general queue and the system selects matching resources from the pool. This simplifies especially the utilisation of a heterogeneous workstation computational pool, in which individual resources may or may not be available for the user.

HTCondor can be used to build grid-style computing environments that span many administrative domains by allowing multiple HTCondor compute installations to work together. This mechanism is called flocking and is, in practise, utilising the HTCondor-C mechanism (the condor grid type in HTCondor) that allows a job to be moved from one pool's job queue to the

remote pool's job queue. Further, HTCondor-G (the `gt2` and `gt5` grid type in HTCondor) allows the same mechanism to utilise remote Globus resources and, with this, HTCondor is fully interoperable with resources managed by Globus. The HTCondors manual describes the operation as follows: *“HTCondor-C is highly resistant to network disconnections and machine failures on both the submission and remote sides. An expected usage sets up Personal HTCondor on a laptop, submits some jobs that are sent to an HTCondor pool, waits until the jobs are staged on the pool, then turns off the laptop. When the laptop reconnects at a later time, any results can be pulled back.”* *“It may appear that HTCondor-G is a simple replacement for the Globus Toolkit's `globusrun` command. However, HTCondor-G does much more. It allows the submission of many jobs at once, along with the monitoring of those jobs with a convenient interface. There is notification when jobs complete or fail and maintenance of Globus credentials that may expire while a job is running. On top of this, HTCondor-G is a fault-tolerant system; if a machine crashes, all of these functions are again available as the machine returns.”*

HTCondor works either with shared file systems (e.g. Network File System, NFS, like typical cluster DRMS) and without such (like workstation grid DRMS). The latter one utilises HTCondor's built-in File Transfer Mechanism, which transfers files needed by a job from the submit computer to the execute computer and transfers all the result files from the execute computer back to the submit computer. Further, it is possible to download and upload input and output files from and to a specified URL. In addition, a continuous transfer during the computation is possible [38].

HTCondor supports running jobs in virtual machines on execute machines. Presently, VMware, Xen, and KVM virtual machine software are supported [38]. This feature enables utilising cloud computing environments with HTCondor. HTCondor also supports the DRMAA job API (version 1.0). This allows DRMAA compliant clients to submit and monitor HTCondor jobs.

1.3.1.3 SLURM

The Simple Linux Utility for Resource Management (SLURM) [16] is an open source software system for highly scalable cluster management and job scheduling for large and small Linux clusters. It is developed and maintained by a consortium of organisations that—at beginning—included Lawrence Livermore National Laboratory, SchedMD, Linux NetworX, Hewlett-Packard, and Groupe Bull. Presently it is used and developed by many of the organisations with large computation resources. SLURM is available under GNU General Public License version 2 (GPL 2). While SLURM was originally written for Linux, the latest version supports many other UNIX-like operating systems: AIX, different BSD variants, Linux, Mac OS X, and Solaris.

SLURM's design is very modular with many optional plugins. Optional plugins can be used for accounting, advanced reservation, time sharing for parallel jobs, backfill scheduling (ability to start a not-high-priority job before highest priority jobs, if it causes no delay to the high priority job), topology optimised resource selection, resource limits by user account, and sophisticated multifactor job prioritisation algorithm. Like many other DRMSs, SLURM has a centralised management server and the computational nodes work as clients. The manager runs a `slurmctld` daemon and the nodes a `slurmd` daemon, which can be compared to a remote shell: it waits for work, executes that work, returns status, and waits for more work. The administration is handled by `sacctmgr` database, which identify e.g. the clusters and valid users. [16]

1.3.1.4 TORQUE

TORQUE [15] (*Terascale Open-Source Resource and QUEue Manager*) is an open source⁷ DRMS based on *Portable Batch System* (PBS) developed for NASA beginning from year 1991. The main developers were NASA Ames Research Center, Lawrence Livermore National Laboratory, and Veridian Information Solutions Inc. The latter one acquired the original developer, MRJ Technology Solutions, in the late 1990s. MRJ released PBS as open source in 1998 under the name OpenPBS [39], but this version is not developed anymore. TORQUE is a fork of OpenPBS maintained by Adaptive Computing Enterprises, Inc. (formerly Cluster Resources, Inc.). In 2003, Altair Engineering acquired all the PBS technology rights and intellectual property from Veridian. Presently, Altair has a commercial product on basis of PBS, called PBS Professional (PBS Pro), and it employs the original development team from NASA. The development of TORQUE is not directly connected to the PBS Pro, even though they both are based on the same original PBS code base.

TORQUE is a typical DRMS with a centralised management server and computational nodes working as clients. It has typical features: control over batch jobs and distributed compute nodes. However, its default scheduler does not have advanced scheduling options, like back-fill or tying to license servers. For that reason, TORQUE can integrate with the open source Maui Cluster Scheduler or its commercial version, Moab Workload Manager (from the same company Adaptive Computing than TORQUE), to improve the utilisation, scheduling and administration of a cluster. [40]

TORQUE supports different UNIX-like operating systems, but it is not officially delivered in binary form for any operating system, but only in source code. The source code can be compiled on different Linux distributions, UNIX systems, Mac OS X, and Windows on the Cygwin environment.

1.3.2 Grid management systems

In the following, we discuss grid management solutions the Globus Toolkit and GridWay that both are projects of the Globus Alliance [41]. The Globus Alliance is an international collaboration that conducts research and development to create fundamental Grid technologies. The alliance includes Argonne National Laboratory, the Information Sciences Institute in the University of Southern California, the University of Chicago, the University of Edinburgh, the Swedish Center for Parallel Computers, and the National Center for Supercomputing Applications (NCSA).

1.3.2.1 Globus Toolkit

The Globus Toolkit [42] is a middleware solution for computational resource sharing and computing in a heterogeneous network environment. The toolkit collects a set of useful technologies into a one combined set and simplifies the management of the technologies in organisations. Strictly speaking, The Globus Toolkit is not a DRMS, but a toolkit for building computing grids that utilises external utilities e.g. for DRMS style operations.

The Globus Toolkit has been called the de facto standard of grid computing. It includes tools for security, information infrastructure, resource management, data management, communication, fault detection, and portability [24]. It includes components for building systems that follow the Open Grid Services Architecture (OGSA) framework defined by the Open Grid Forum (OGF) [27].

The Globus Toolkit includes the *Globus resource allocation manager* (GRAM) that handles jobs on grid computing resources. GRAM is a not a meta-scheduler or other type of DRMS,

⁷ TORQUE is described by its developers as open source software, using the OpenPBS version 2.3 license. However, Debian Free Software Guidelines sees it as non-free software owing to issues with the license.

as GRAM does not provide job scheduler functionality and is just a front end to the functionality provided by an external scheduler. GRAM's idea is to simplify the use of remote systems by providing a single standard interface for requesting and using remote system resources for the execution of jobs. The Globus Toolkit is compatible—through GRAM or otherwise—with various DRMSs and other computing related tools, for example HTCondor, Grid Engine, SLURM, and TORQUE. It is mainly developed by the Globus Alliance.

1.3.2.2 GridWay

GridWay [25] provides a unified interface to large, grid computing resources inside organisations and between organisations. In the GridWay documentation, this solution is illustrated to operate on top of the local resource management and the Globus Toolkit layers, providing unified interface to access and monitor resources in large resource pools. GridWay aim to reduce the gap between grid middleware and users (including application developers) by providing a consistent and familiar working environment to access all cluster, grid and cloud resources in the organisation, i.e. it decouples the usage from the underlying local resource management systems.

GridWay is a Globus Alliance project aiming for a meta-scheduler that allows a user to submit, monitor, synchronise and control jobs by means of a DRMS-like command line interface. GridWay performs job submission and scheduling transparently to the end user and provides fault recovery mechanisms, load and resource based dynamic scheduling, and resubmission of the job when poor performance is detected. It enables sharing of all types of computing resources: clusters, supercomputers, and stand-alone servers. GridWay supports most of the existing grid middleware systems, can be used on grid infrastructures, and can access cloud resources. GridWay interfaces to remote resources through Globus GRAM. Thus, it supports all Globus compatible remote platforms and resource managers, e.g. SGE, PBS, and HTCondor.

For operating system support, GridWay has been tested on Linux and Solaris [25].

1.3.2.3 Bosco

Bosco [43] is a meta-scheduler for simplifying the job scheduling in a large and heterogeneous computing environment. The system enables submitting computational jobs from the user's local computer to the network of resource management systems, such as the intranet of a large organisation. The network of resource management systems may consist of a different kind of local cluster and resource management systems. At the moment, the following cluster and resource management systems are supported:

- PBS-based TORQUE and PBS Professional;
- HTCondor (starting from version 7.6);
- Platform Load Sharing Facility, LSF (IBM's proprietary workload management platform);
- SGE; and
- SLURM.

Bosco uses Secure Shell (ssh) for submitting jobs from the user's local computer. The job scheduler is based in HTCondor and the jobs are submitted using HTCondor job description mechanism and syntax.

The advantage of using a meta-scheduler, such as Bosco, is that the users need to learn only one scheduler system and its job submission, monitoring, and management procedures, and can utilise the resources of a large and heterogeneous computational environment. The challenges of the approach are the differences of the resource management systems. The load balancing and submission mechanisms are different and e.g. the syntax of submission descrip-

tions differs from each other. This may cause some detailed features of the different resource management systems not to be available for the users of the meta-scheduler systems.

1.4 Other solutions for utilising remote computational resources

1.4.1 Techila

Techila is a commercial grid computation management environment for embarrassingly parallel computations [18]. As described before, embarrassingly parallel means there are little or no connections between computation processes during the computation, so the processes are independent and do not share any data or use common memory allocation. Techila runs in background in the computational nodes (i.e. computer that belong to the Techical grid), and is only activated when no other processes are loading the processor. Techila data flow is illustrated in Figure 3. There are three layers which may be separated in practise: Server, Workers, and End User. Techila environment consists of Techila Server (for distributing the work load) and Techila Workers which do the actual computations. They create together the gridified section, which internal operations are hidden from the end user.

The end user may monitor the operations and launch new projects and bundles with Techila User Interface. Typically, the launching of new projects and bundles is done by some 3rd party software, such as MATLAB. In addition to MATLAB, Python, R, Perl, Java, C/C++, .NET and FORTRAN programming languages are supported. The source code created with supported environment must be compiled with Techila software development kit (SDK). The compilation may be done either at End User part (local) or in the Techila grid (remote). Both approaches have their advantages and disadvantages. For local compilation, you must know the architecture of the Workers to cross-compile directly to the correct run-time format. On the other hand, if compiled remotely there must be suitable compilers and their control structures installed in the remote system.

The monitoring function is fulfilled with a web-based view to the Techila server. There the general, Worker and project status of the grid and computation may be monitored.

The Workers may be computers in the local computer network or they may be any computer resources which may be bundled with the Techila Server system, such as Microsoft Azure cloud computing resource. One of the original ideas of Techila has been to utilise the local computer network computers as Workers while they are idling (for example in night times).

Ideally, when using the local network capacity, Techila is very sensitive to a normal computer usage, so it quickly transfers the computations elsewhere if the user needs his or her computer. Undocking of the laptops etc. may also be done as usually. Techila dynamically searches for the best computation capacity, and takes the best idle computer into use. The more idle computers there are, the better the system works.

Unfortunately, this vision has not been fulfilled especially due to the more widespread usage of laptop computers, which are often not connected to the network when not in use, and which also have less computational resources (processing power, cores and memory) than desktop computers.

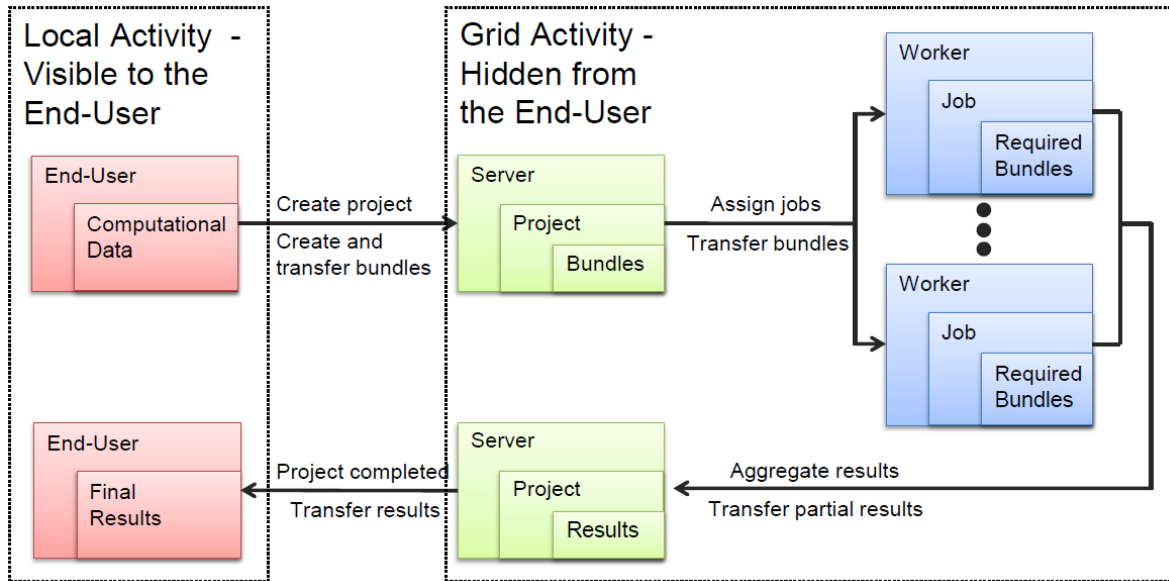


Figure 3: Techila data flows. Courtesy of Techila presentation material 2011.

When using independent software vendor (ISV) applications, there are three ways to integrate them to the Techila environment: 1) plug-in integration, 2) seamless integration, and 3) black box integration [44]. The plug-in integration is possible, if the software supports the plug-in Techila architecture. The seamless integration may be utilised, if the source code is available, and suitable computationally intensive parts may be separately compiled with Techila SDK. The black box integration is possible, if there is suitable command line interface (CLI) available to control the program execution. This integration scheme is shown in Figure 4.

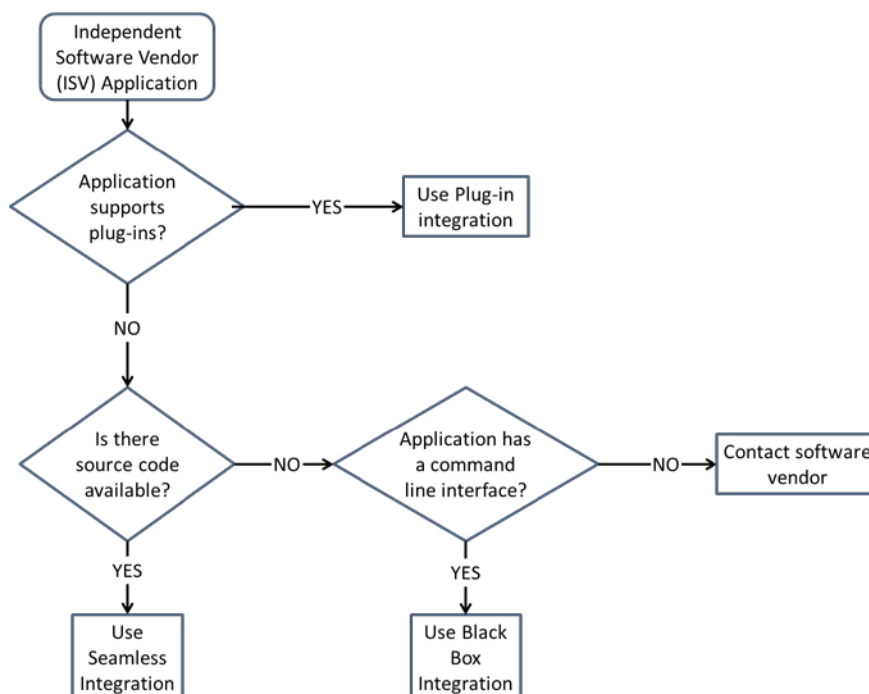


Figure 4: ISV integration schemes to Techila [44].

1.4.2 Azure cloud service together with Techila

The Microsoft Azure cloud service was tested with Techila environment. The Azure cloud service with Techila consists of *Windows Azure Virtual Machines* (WAVM) and Techila

front-end [45]. WAVM provides *infrastructure as a service* (IaaS). Figure 5 shows its components.

Techila supports several cloud services, including Google GCE, Amazon EC2 and Microsoft Azure. Furthermore, local virtualisations such as VMware ESXi are supported. Virtual machines may be created using either the Windows Azure Management Portal or the REST-based Windows Azure Service Management API. When used together with Techila, Techila front-end with suitable API is used. Creating a new VM requires choosing a virtual hard disk (VHD) for the VM's image. These VHDs are stored in Windows Azure Blobs. Azure Blob storage is a service for storing large amounts of unstructured data.

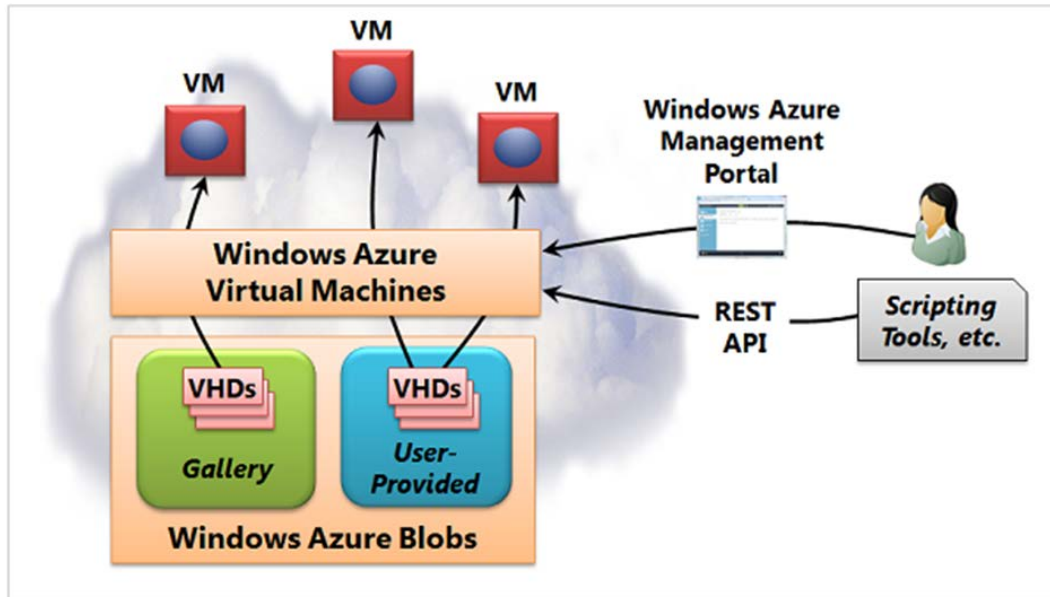


Figure 5: Windows Azure Virtual Machines provides Infrastructure as a Service. Courtesy and copyright of Microsoft Corporation, 2014.

2 Experiences with resource management systems

2.1 Grid Engine in a cluster system

2.1.1 Description of the computational environment “the Doctor”

VTT has a computer cluster called “the Doctor”. Its configuration has presently (early 2015) 1456 cores in 69 nodes including a master host, a backup master host (working as normal execution host) and 67 execution hosts. 61 of the cluster nodes are Dell PowerEdge M620s with 16 or 20 cores total in two processors. In addition, there are three R820 and five R630 machines with 32 cores total in four processors and one R920 with 60 cores total in four processors. The processors for M620s are Intel Xeon E5-2680 at 2.70GHz for the first 16 nodes and E5-2680v2 at 2.8GHz for the newer 45 nodes. The R820s have E5-4650 processors at 2.70GHz, R630s E5-2698 v3 at 2.30GHz, and the R920 E7-4880 v2 at 2.50GHz. All the execution hosts are connected together with an InfiniBand network (the 40 Gbps version) for MPI data and 10 Gbps Ethernet for Network File System (NFS) traffic (disc usage) and non-MPI communication between nodes. Most of the nodes have 2 SSD discs for swapping and system image, but the computation results are stored in two NetApp systems having totally 130TB capacity. Several tests have shown that the M620s (older ones with E5-2680) are somewhat more efficient than the R820s, even if the clock rate and architecture of the processor are the same. The R820s have even more L1 cache (512 kB vs. 256 kB). The efficiency difference is supposedly related to the identical memory bandwidth, which in R820s is shared with 32 cores and in M620s with 16 cores. The M620/R630 nodes have 128–256 GB of memory, whereas the R820s have 512GB memory and the one R920 has 1536 GB of memory.

All the nodes have a Rocks Cluster Distribution 6.1.1 (CentOS release 6.5) as their operating system intended for high-performance computing clusters. The operating system includes tools for mass installation onto many computers. Further, Rocks Cluster includes many tools (such as MPI) which are not part of CentOS but are vital to make a group of computers work as a computational cluster. The operating system includes several alternative DRMSs, chosen install-time by using so-called Rolls, along with useful cluster tools like Ganglia [46].

The cluster is, for the time being, meant for all VTT staff, more than 2000 people, but there are only few hundred users who utilise modelling and simulation; even less who need HPC environment for their simulations. However, in reality there are less than 20 users in the cluster, and even then the cluster is close to 100 % usage most of the year. Thus, there is real need for resource management, i.e. scheduling and automatic load balancing. Further, the variety of usage is quite wide, from embarrassingly parallel computation to barely parallel FEM simulations. An example of embarrassingly parallel computation is bioinformatics algorithm and tool BLAST, for comparing primary biological sequence information, which could utilise even several thousands of cores in parallel. For the other end, commercial FEM tools ACTRAN and ABAQUS, especially in modal analysis, can effectively utilise only 6–10 cores due to strongly connection of the parallel sub problems, causing heavy core-to-core (MPI) data traffic causing slowdowns even inside one computation node.

2.1.2 Resource management in the cluster

The (Sun) Grid Engine (SGE) was selected as the computation resource manager, or distributed resource management systems (DRMS), on the cluster. The reason for the choice was the fact that SGE is still the most popular clusters DRMS, even if SLURM is raising its popularity in new clusters. Most of the tests are done with SGE version 6.2u5, which is the last open source version released by Oracle and the last unified version before the division of the open source development of SGE. Thus, all the results should be applicable for all SGE versions,

but possibly some newer versions include features that are still missing from version 6.2u5. Presently, after the update from Rocks 6.0 to 6.1.1, SGE was replaced by OGS 2011.11p1. The change did not affect the usability of the DRMS in the cluster.

2.1.2.1 Operation of SGE in cluster environment

A SGE cluster is composed of execution machines, a master machine, and possible shadow master machines. The execution hosts run the Grid Engine execution daemon and master host runs the SGE `qmaster` daemon. In a case that the master machine fails (e.g. crashes), the shadow daemon on one of the shadow master machines will take over the role of the master host. The `qmaster` daemon is the heart of the cluster, and it handles the submitting and scheduling of all the jobs. The running of the jobs is done in the execution daemons, i.e. the execution daemons are the work horses of the cluster [47]. However, like in the VTT cluster, the master host can (partially) work as execution host and the shadow masters work in normal conditions as normal execution hosts. In “the Doctor” cluster, the master host has 16 cores, but only 12 of them are employed for SGE execution and the rest are saved for SGE master usage and non-SGE usage.

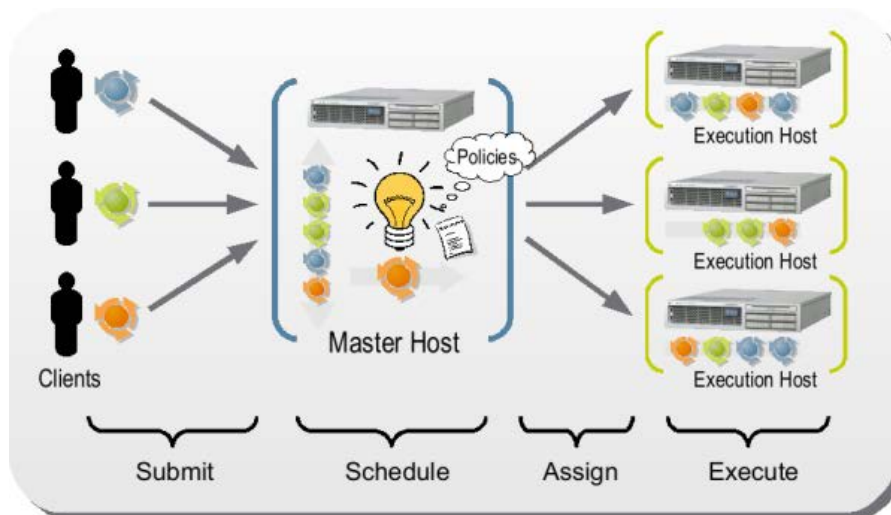


Figure 6: Illustration of Grid Engine usage and different hosts. [47]

All the usage of the SGE cluster is handled through the master host. The user logs with SSH to the master host and uses SGE commands to manage his/her computation. For example, the job submission is done in command line by `qsub` or graphically with `qmon` tool, but the command line tools are by far more popular and handy. In the job submission command, the user includes all of the important information about the job, like what it should actually run, what kind of execution machine it needs; possibly also upper limits to how much memory it will consume, how long it will run, etc. All of that information is then used by the `qmaster` to schedule and manage the job as it goes from pending to running and finally to finish.

First, a submitted job always enters the pending state. On the next scheduling run, the `qmaster` will rank the job in importance compared to the other pending jobs. The configured scheduling policies largely determine the relative importance. After ranking, the most important jobs will be scheduled to available job slots. Typically, a CPU core in an execution host represents a job slot; i.e. the number of slots is equal to the total number of CPU cores in the cluster execution hosts. Every available slot is filled with a pending job. If a job requires a resource or a slot on a certain type of machine that isn't currently available, that job will be skipped over during that scheduling run. For example, a typical job for HPC usage is a parallel job, which is distributed across multiple nodes and ran in parallel. Thus, the job needs as many free slots

as there is parallel jobs; if there is not enough free slots, job will be skipped and waits for slots to be released.

After the job has been scheduled, it is sent to the execution daemon on the selected machine. The execution daemon executes the job, i.e. the job enters the running state. The job runs until it completes, fails, is terminated, or is re-queued. Further, the job may be suspended, resumed, or performed a checkpoint any number of times. SGE does not handle checkpoint procedure itself, but triggers whatever checkpoint mechanism is available and configured for the job. After a job has completed or failed, the execution daemon cleans up and notifies the `qmaster`. The `qmaster` saves the job's information in the logs and removes the job from the list of active jobs.

SGE organises the computation resources by different queues. The user selects, which queue the job is sent to, based on the requirements of the case in hand. The queue is a collection of slots, i.e. a certain group to computers, treated as one computation resource. If the user sends the job to a certain queue, the scheduling is done inside these resources. However, to be precise, the connection between queues and computational hardware is not so simple, as one execution host can be included into several queues. Thus, queues can be configured to stand for some resource or for some special type of job: e.g. short, medium length, and long jobs; interactive jobs; high memory jobs; and queues with different priorities. Of course, there could be correlation between different types of use and needed hardware, e.g. a high memory queue would use execution hosts with more memory than average hosts. In many organisations, there are policies restricting who can use certain resources or limiting the resource usage of certain users. For example, a cluster can include nodes owned by a small unit of the organisation, and the members of the unit have higher priority on that resource. Further, the duration limited queues (e.g. short, medium, and long queues) can have stiff limitations on the job execution duration, meaning that the job is terminated or suspended if the limits are exceeded.

As mentioned in section 1.2.1, there exist several message passing solutions for communication between cluster nodes, e.g. MPI and PVM. In SGE, these solutions are selected by choosing a so-called parallel environment. In detail, the parallel environments are parallel programming and runtime environments allowing for the execution of shared memory or distributed memory parallelised applications. Presently, different MPI implementations are the most used ones. For Open MPI, the parallel environment is called `orte`, and for MPICH simply `mpich`. There are also shared memory parallel environments, like `smp` and `openmp`, and some software specific environments.

2.1.2.2 SGE configuration in “the Doctor” cluster

A Grid Engine based cluster have, at the time of writing (end of 2014), been two years in test use in VTT. The following sections are based on the experiences during the test period.

In the present configuration of “the Doctor” cluster in VTT, the main queue, `a11.q`, is the most used one and there are no stiff limitations on the job duration or memory usage. However, there are separate queues for certain unit's own hardware, where the access to the queues is restricted to a certain group of users. However, these kinds of queues are planned to be used by others e.g. via a Techila grid, when the hosts of these queues are in idle. Presently, there are only four queues: two for the above-mentioned unit-restricted nodes, the main queue `a11.q`, and one for the 12 (of 16) cores of the master host. The last mentioned queue is intended for interactive work, such as post-processing, optimisation control, and small MATLAB jobs. Several overlapping queues were tested in the beginning of the test period (separate queues for ABAQUS and all the other use), but this was found confusing, as the users had to check the state of a host from two separate queues.

One of characteristics of the default SGE configuration was that the execution hosts were allowed to be overloaded, i.e. a 16 core host was allowed to have load⁸ close to 32. Thus, the default SGE configuration allows heavy overloading of the hardware. This was not effective in typical FEM and CFD simulation jobs, as the overloading only caused lower average efficiency of the computation. By default, the loading of a host is monitored by the five minute average load of the machine divided by core number, denoted `np_load_avg`. The configuration was changed such a way that jobs are not scheduled if `np_load_avg > 1.05`, i.e. if the host has full load, no new jobs are assigned to it. This change solved the above-mentioned problem of harmful overloading.

Also other types of load sensors can be configured to affect the scheduling, e.g. the memory usage in an execution host. In some large memory usage situation, all the memory of a host can be used but most of the cores are free. In that case, with a load sensor based only on CPU load, SGE can assign new jobs to the host resulting an out-of-memory situation and one of the jobs will then be killed. In addition, so-called complex values of SGE can be used for reserving the node memory of a host to one job. For example, `virtual_free` complex value can be used for reserving the host memory and scheduling a job to a host that has enough free (virtual) memory. Also, complex value `exclusive` can be helpful. If it is enabled for a host, a user can exclusively reserve the host for her/his job, e.g. in a situation when the user knows that the job will use all the memory from the node.

2.1.3 Practical examples and how the systems works

2.1.3.1 Using OpenFOAM in the SGE cluster

Submitting computation

This section describes how to send a job of a computational fluid dynamics (CFD) analysis to the computational nodes from the master host (the front node) of “the Doctor” cluster; in the following text, the name of the front node is `vttcal1c001`. Notice that in the examples, the command prompt of the command shell is included and is “%>”. After the SSH connection has been established, the environmental variables have to be changed to proper ones for the case in hand. In this case, the computation is performed with the OpenFOAM-2.1.x software application. The environmental variables can be changed e.g. by modifying the `.bashrc` file in the `/home/user` folder. In this case the `.bashrc` file is changed to (the additions are in bold face):

```
# .bashrc
# Source global definitions
if [ -f /etc/bashrc ]; then
. /etc/bashrc
fi
# User specific aliases and functions
# OpenFOAM-2.1.x
source /share/apps/OpenFOAM/OpenFOAM-2.1.x/etc/bashrc
```

In the beginning of a session, the `.bashrc` script is run e.g. with the `source` command:

```
%> source .bashrc
```

The computations are not to be executed in the front node, but instead in the computational nodes. The computations can be sent to the computational nodes with the `qsub` command:

⁸ Load is a number to measure the amount of computational work that a computer performs. For 16 core machine, load value 16 equals that all the cores are fully utilised. Value 32 means that there is about double of amount of computational load compared the real computation power of the hardware.

```
%> qsub submit
```

and for a specific computational (e.g. `all.q@compute-0-5.local`) node with the `-q` option:

```
%> qsub -q all.q@compute-0-5 submit
```

where the string after the `-q` option defines the computational queue (in this case `all.q`) and the computational node (in this case `compute-0-5`) and the `submit` file includes all the information needed to execute the computation as demonstrated below for the OpenFOAM's `simpleFoam` solver:

```
#!/bin/sh
# Used command-line interpreter
#$ -S /bin/bash
# The name of the job
#$ -N test
# Use this folder as a working folder
#$ -cwd
# Write one output file and give its name
#$ -j y
#$ -o output
# Set the Open MPI parallel environment and the number of processors
#$ -pe orte 16
# Pass the enviromental variables
#$ -V
# Reserve resources for the parallel job
#$ -R y
#OpenFOAM specific command
decomposePar
mpirun -np $NSLOTS simpleFoam -parallel
#OpenFOAM specific command
reconstructPar
```

The queues and the state of the computations and other information, such as the job ID, can be viewed with the `qstat` command, e.g.:

```
%> qstat -f -u <user>
```

where `<user>` is your or some other user's user name. You can view jobs from all users by replacing `<user>` with `"*"` in the above example. The computation can be deleted from the computational nodes with the `qdel` command:

```
%> qdel <job-ID>
```

where `<job-ID>` is the ID number of the job to be deleted.

Scaling results

One of the most important aspects of parallel computation in clusters is the scaling of the computation, i.e. how the real computation time behaves as the function of the number of computation cores. Here we present the scaling results of the OpenFOAM software application for a 2.5 million cell case with double precision, SCOTCH decomposition, and 500 iterations, computed in "the Doctor" cluster in its 16 core nodes. The scaling results are presented in Table 1, Figure 7, Figure 8, and Figure 9.

Table 1: Scaling results of the OpenFOAM test case.

# of cores	Real time	Efficiency	Speed-up
80	110 s	0.907	72.56
64	126 s	0.990	63.36

48	169 s	0.984	47.232
32	250 s	0.998	31.94
16	499 s	1.0	16

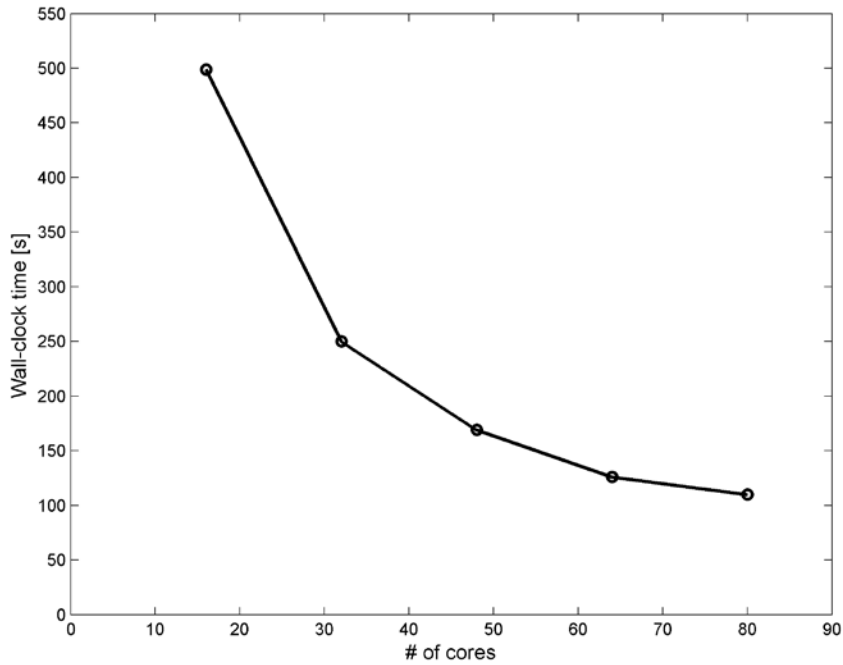


Figure 7: Wall-clock times for solutions with different number of cores.

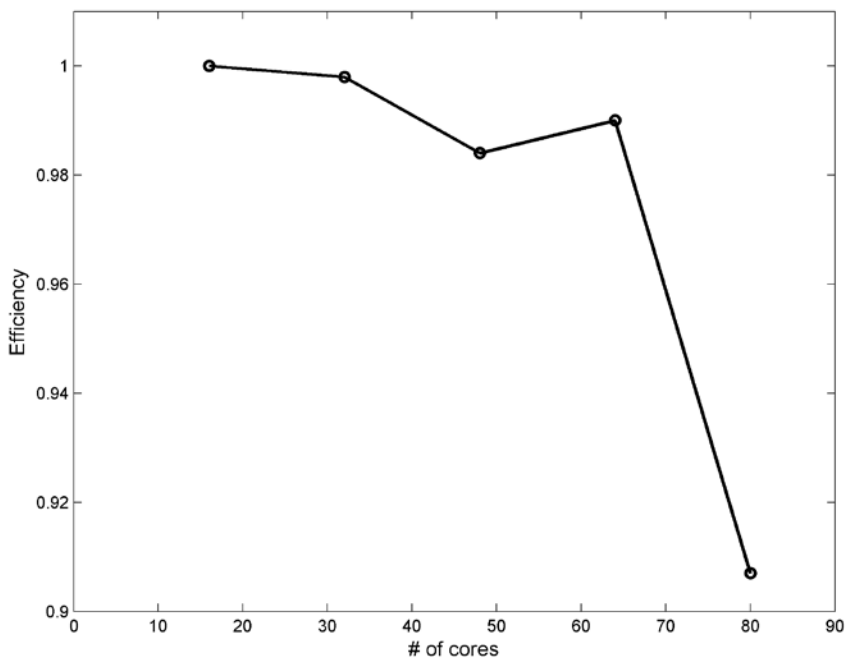


Figure 8: Efficiency for solutions with different number of cores. Efficiency means the capability to utilise the cores, value one means full utilisation.

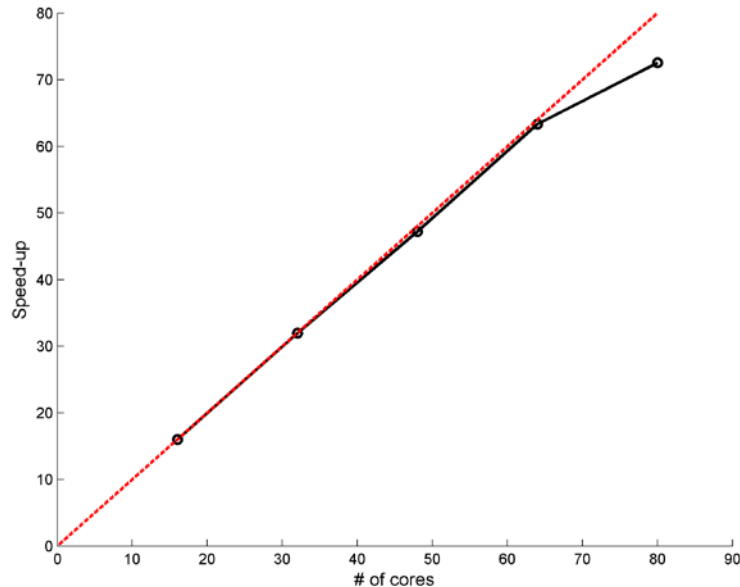


Figure 9: Speed-up for solutions with different number of cores. Speed-up is the ratio of computation time compared to the case computed with one core, here 16 core case is the reference level. The red dashed line represents the optimal performance (linear scaling).

The results show that OpenFOAM scales quite ideally up to 64 cores in the cluster system. Above that, the scalability decreases. Hence, runs using more than 64 cores should be avoided in OpenFOAM simulations. However, this result is valid only for just this hardware setup and this version of the OpenFOAM software—in other types of OpenFOAM cases the scaling may be different.

2.1.3.2 Using Abaqus in the SGE cluster

Abaqus FEA is a FEM software suite from SIMULIA, owned by Dassault Systèmes, originally released in 1978. It is among the most popular FEM tools along with Ansys.

For Abaqus usage in the VTT computational cluster environment, user must log in to the cluster front-end machine `vttcalc001.ad.vtt.fi`. All software applications installed to the cluster are located in the `/share/apps` folder. Different versions of the Abaqus software package can be found from the `/share/apps/Simulia` folder. General `abaqus` command starts the latest Abaqus version; an exact individual Abaqus version can be started by using a version labelled format, for example `abq6123`. The Abaqus environment file (`abaqus_v6.env`) can be found from each Abaqus version's site directory (for example, `/share/apps/Simulia/6.12-3/SMA/site`). If needed, that can be copied to one's own home directory for necessary modifications. A `scratch` directory definition is not needed, because that is handled automatically by the job invoking procedures.

All cluster jobs have to be directed to the `a11.q` queue. An Abaqus job is started by using e.g. the following command:

```
%> abaqus job=jobname cpus=8 queue=a11.q
```

This command reserves 8 CPU cores for the job. If the user wants to direct the job to a certain computation node, the following modification is needed:

```
%> abaqus job=jobname cpus=8 queue=a11.q@compute-0-2
```

where `compute-0-2` is the name of a node. The number of Abaqus license tokens needed for a job with certain number of cores (CPUs) can be determined by using the following command:

```
%> abaqus job=test cpus=30 queue=tokens
```

Abaqus supports distributed memory parallelising (DMP) and an analysis job can be extended beyond a single computation node. In these cases, the number of necessary analysis tokens should be determined and compared against available tokens. Licence usage (using the license manager set in the file `abaqus_v6.env`) can be checked by using the following command:

```
%> abaqus licensing ru
```

If it is required that a job will be run only in a single computation node, it can be forced by using the memory type option `mt[dmp;smp]` in the queue definition, for example:

```
%> abaqus job=test cpus=8 queue=all.q:mtsmp      (single node)
%> abaqus job=test cpus=8 queue=all.q:mtmpi      (several nodes when needed,
                                                    default)
```

In the case of extensive disk usage during an analysis, instead of local computation node disk space, front-end machine disk resources have to be used for storing the analysis data. The analysis data storage option can be enforced by adding file system definition `fs[local;shared]` to the queue definition, for example:

```
%> abaqus job=test cpus=8 queue=all.q:fsshared    (data files at front-end
                                                    directory)
%> abaqus job=test cpus=8 queue=all.q:fslocal     (data files at
                                                    computation node local directory, default)
```

All analyses have to be run through SGE queuing system invoked from the front-end machine. The detailed cluster queue status can be monitored by using the following command:

```
%> qstat -f -u "*" -q all.q
```

During an Abaqus analysis, there is no other information available in the front node start directory than a log file that includes the name of the primal execution node and working directory in that host. After the analysis has completed, the run files are copied to the front node automatically. The work directories are created automatically to the `/tmp/username` directory, where `username` is the user's user name.

List of one's own submitted runs can be viewed simply by using the following command:

```
%> qstat
```

If the submitted run has to be terminated before its normal completion, run termination is invoked using the following command:

```
%> qdel <job_ID>
```

Here `<job_ID>` refers to the run ID number. When using Abaqus, if the analysis was run with the local mode in the computation node, restoring analysis data to front-end directory can take few moments.

In order to get the user subroutines working, few settings have to be added to the user files. When using bash or bourne shell as the command shell, following lines should be added to the `.bashrc` file (in the `/home/user` directory):

```
source
/share/apps/intel/composer_xe_2013.0.079/composer_xe_2013.0.079/bin/compilervars.sh intel64

export PATH=/share/apps/Simulia/Commands:${PATH}
```

In the case of `csch` or `tcsh` command shell, the following lines should be added to the `.cschrc` or `.tcshrc` file (in the `/home/user` directory):

```
source
/share/apps/intel/composer_xe_2013.0.079/composer_xe_2013.0.079/bin/compilervars.csh intel64
setenv PATH "/share/apps/Simulia/Commands:/home/user/scripts:${PATH}"
```

2.1.3.3 Elmer in SGE cluster

Elmer is an open source finite element software that is developed by CSC – IT Center for Science Ltd. (later in this document CSC) and it is intended for multi-physics simulations. Elmer has been recently developed towards low-frequency electromagnetic simulations, applications being especially transient electrical machines. For a long time, Elmer’s strengths have been in parallel computing and the parallel computation has been partly developed in the SIMPRO project. Especially a vast number of parallel performance tests have been run in large computational environments utilising the SGE and SLURM systems.

The test for this study was performed in VTT’s “the Doctor” cluster system. As an open source software application, parallel performance of Elmer is vastly based on the used compilation options. After the update to Rocks Linux version 6.1.1 (CentOS 6.5), the performance of Elmer was slowed down considerably, if the system’s default `gcc` compiler and Open MPI were used in the compilation of Elmer. Utilisation of locally compiled `gcc` version 4.9.0 and Open MPI version 1.8.1, together with Open MPI flags “`--with-sge --disable-mca-dso`” accelerated the test runs three-fold. Further, Elmer was compiled with optimisation options “`-g -O2 -march=sandybridge`” to improve its performance.

To run Elmer, one has to set some additional environmental variables:

```
export MPIHOME=/share/apps/local
export ELMER_HOME="/share/apps/elmer/Elmer"
export ELMER_LIB="$ELMER_HOME/share/elmersolver/lib/"
export ELMER_POST_HOME="$ELMER_HOME/share/elmerpost"
export LD_LIBRARY_PATH="$MPIHOME/lib64:$MPIHOME/lib:$LD_LIBRARY_PATH"
export PATH="$MPIHOME/bin:$PATH:$ELMER_HOME/bin:$HOME/bin"
```

In the following examples, these are included in the user’s `.bashrc` file (in the user’s home directory) and are automatically taken into account. Without using SGE, Elmer is run in parallel with the following command:

```
%> mpirun -np 20 ElmerSolver_mpi
```

where `-np 20` defines the number of cores used in the host where the command is run. In a multiuser system, the computation can be queued and divided into several nodes by using SGE. Here is an example script of an Elmer run for SGE:

```
#!/bin/bash
# Used command-line interpreter
#$ -S /bin/bash
# The name of the job
#$ -N Elmer_scaling
# Use this folder as a working folder
#$ -cwd
# Write one output file and give its name
#$ -j y
#$ -o output
# Pass the enviromental variables
#$ -V
# Reserve resources for the parallel job
#$ -R y
```



```
source ~/.bashrc
mpirun -np $1 ElmerSolver_mpi case.sif > logfile.log 2>&1
```

The number of cores utilised is given as a command line input argument for the script. This script, for example saved as a file named `run_elmer_case.sh`, is run as SGE job with the following command:

```
%> qsub -pe orte 100 -q all.q -l excl=true run_elmer_case.sh 100
```

where 100 is the number of cores, given as input through the Elmer script to mpi and to the SGE parallel environment orte. The queue `all.q` is utilised exclusively by “`-l excl=true`”, which means that no other job can share the nodes with this job. This is very important for reliable parallel scaling tests results, i.e. reliable computation times with variable number of cores.

For parallel tests, multiple runs for the same problem with different number of cores were utilised. Here is an example script to run such a series of jobs:

```
#!/bin/sh
cases="160 140 120 100 80 60 40 20 10"
mkdir scaling_tests
cd scaling_tests
for m in $cases; do
  echo "case $m"
  mkdir $m; cd $m ;
  cp ../../case.sif .
  cp ../../lorentz* .
  cp ../../mesh.names .
  cp -r ../../mesh .
  cp ../../ELMERSOLVER_STARTINFO .
  ElmerGrid 2 2 mesh -partdual -metis $m 3
  qsub -pe orte $m -q all.q -l excl=true ../../run_elmer_case.sh $m
  cd ..
done
cd ..
```

This creates hierarchy of folders for different cases, copies the input files to the folders and runs Elmer with proper number of cores in each folder. This demonstrates how one can benefit from SGE; there is no need for the user to know in which nodes the computation is done. SGE can decide in which order the different runs are performed in such a way that the utilisation rate of the machines is the highest possible. The built-in feature of SGE for overloading the nodes by several jobs—which can be a wanted feature, if the jobs do not utilise the CPU all the time by 100 %—can be avoided here by the “`-l excl=true`” flag. To use this feature, this must be enabled to the nodes by `qconf -me` command in every node (which opens the proper configuration file in the default editor), and by editing the following:

```
complex_values      slots=20
```

where 20 is the number of physical cores in the node.

2.1.4 Lessons learned with the Grid Engine system

Taking SGE into use in VTT’s new computation cluster “the Doctor” was a big improvement to the situation, where no computation management system was utilised in the previous smaller clusters. With SGE, real queueing and load balancing was possible. However, in the beginning there were some challenges and many SGE features were not taken into use due to lack of user friendliness or challenges in the usage. For example, SGE could restrict the running time and usage of resources of a user, but the rules are quite strict for organisation like

VTT and as an example, jobs exceeding the maximum time would be killed. This is not acceptable for long simulations of seldom users, whose intermediate results of the simulation are possibly not saved at all. Thus, some of these strict SGE limits were replaced by user rules, i.e. a kind of gentlemen's agreement.

One of the lessons learned from SGE was that SGE inheritable tries to utilise the resources as effectively as possible by overbooking the resources. There is a threshold for how high the load in a node can raise until no further jobs are submitted to that node. The default value was 1.75 for the five minute average of load.⁹ This means, the computation nodes could be almost doubly booked with jobs, slowing down the simulation times and sometimes even doubling the solution time. This could be advantageous for some software tools, where the CPU usage is not constant for all solution phases. For example, some FEM tools have a long phase during the overall case solving, when large amount of memory is used but the CPU usage is low. In these cases, running two simulations together and thus overbooking the resources may be justified. However, tools like OpenFOAM and Elmer can use 100 % of the reserved CPU for the whole simulation time and in these cases overloading is highly disadvantageous. For this reasons, to avoid overbooking, the five minute load average threshold was lowered to 1.05. In some queues, also the above-mentioned exclusive reservation of nodes was found to be practical.

One problem with SGE is that many grand users of SGE have been switching to use other DRMSs within the last 5 years. For example, CSC's old clusters (sometimes also called supercomputers) have utilised SGE, but the newest one called Sisu is using SLURM. Thus, the practical knowledge of SGE and its new features is declining and help for administrating SGE is increasing hard to find. It is very easy to get SGE to work tolerable, but fine-tuning of its settings is time-consuming and experimenting with settings are unwise in a production system. However, by basic settings with minor modifications (i.e. tuning of the load threshold and enabling exclusive reservations) along with above-mentioned gentlemen's agreement have turn out to be a satisfactory solution for time being.

2.1.5 Future steps for system development in "the Doctor"

The present settings of the SGE system in "the Doctor" cluster are somewhat satisfactory, but enabling some features could be beneficial for some users. In addition, new users and new tools sometimes give challenges to present settings of the system. One previously occurred problem has been software tools that use relatively more memory than CPU. In the basic setting, a user can only reserve CPU from computation nodes, not memory. For example, certain software applications use the full 512GB memory of the 32-core nodes, but utilise only 1–4 cores. In these cases, SGE—even with the 1.05 load threshold—sees only load of $1/32$ – $4/32$, and submits new jobs until load exceeds 1.05. Then, the new jobs have no free memory and all the jobs are slowed down due to vast usage of virtual memory. There is possibility to add a setting to reserve memory from nodes, but then every scheduled job must include the data for maximum needed memory in the submission phase. This seems quite cumbersome from the users' point of view. Thus, exclusive reserving of the node is a better solution for a high memory case, as long as node's all memory is needed and there are nodes with free high-memory available.

In the future, there will be further discussion whether or not changing to some other DRMS, probably SLURM, would be more beneficial than the troubles caused by the change.

⁹ Load is a measure of the amount of computational work that a computer performs. The load average is the time average of the system load over a certain time period. In SGE, load maximum is one, not depending on the number of cores, i.e. SGE load is the UNIX load divided by number of cores.

2.2 SLURM in a cluster system

2.2.1 Description of the computational environment Sisu

Sisu is a supercomputer owned and managed by the CSC. The computer is manufactured by Cray Inc., belonging to the XC30 family, and it is the fastest supercomputer in Finland. It was originally taken into use in autumn 2012 and updated in September 2014. Structurally it equals to computer clusters, differentiating from VTT's cluster especially by its size and the connecting network between the nodes. Sisu has Cray's Aries proprietary interconnect fabric network. The topology of the network is called dragonfly, which is an "n-dimensional" torus. After September 2014 update, Sisu is composed of 422 compute blades, each of which hosts 4 computing nodes, i.e. 1688 computation nodes in total, each with 2 12-core Intel Xeon E5-2690v3 processors. In the whole system there are 40512 cores, with 64 GB memory per compute node. The theoretical peak performance of the system is 1688 TFLOPS, placing the system among the 25 most efficient computers in the world. The compute nodes run a light-weight Linux kernel provided by Cray, called Compute Node Linux (CNL).

2.2.2 Resource management in Sisu

Instead of SGE, which was used in CSC's old supercomputers, SLURM is used in Sisu. The setup in Sisu is not a typical SLURM setup. The parallel commands are launched using ALPS (Application Level Placement Scheduler) resource manager, produced by Cray. It is a rudimentary scheduling software system dependent upon external scheduler for workload management. In Sisu, the external scheduler is SLURM. For launching a batch job, the ALPS command `aprun` is used instead of the `srun` which is normally used in SLURM.

In Sisu, it is recommended to use full computation nodes for running jobs. This is done by using SLURM option `-N` with the number of nodes to be reserved.

2.2.3 Practical examples with Elmer

Comparative Elmer electromagnetic electrical machine simulations were performed in Sisu, to compare the scalability of Elmer in VTT computation cluster and in Sisu. Thus, similar series of runs with variable number of cores were performed also in Sisu (see section *Elmer in SGE cluster*).

Here is an example script to run an Elmer job in Sisu:

```
#!/bin/bash -f
#SBATCH -t 06:00:00
#SBATCH -J ElmerJob
#SBATCH -o ElmerJob.%j
#SBATCH -e ElmerJob.%j
#SBATCH -p small
#SBATCH -N 4
(( ncores = SLURM_NNODES * 24 ))
aprun -n $ncores /path/to/Elmer/bin/ElmerSolver_mpi > logfile.log 2>&1
```

The second line defines the maximum execution time of the job, in this case six hours. The following lines in the script describe the name of the batch job, along with the file names for the standard output and standard error (both including the SLURM batch job number after dot). The partition called "small" is used; a partition is similar to a queue in SGE. With option `-N 4` full nodes are reserved for this job, and the job gets 4×24 cores to be used in total. The batch job is submitted to the SLURM system with the following command:

```
%> sbatch file_name.sh
```

The command `squeue` can be used to jobs submitted to the scheduler. If the user wants to see only his or her own jobs, the following command is useful:

```
%> squeue -l -u $USER
```

A job can be cancelled with the `scancel` command, followed by the job ID given by `squeue` command. The usage of different partitions and the number of free nodes can be checked with the command `sinfo`.

2.2.4 Lessons learned with the SLURM system

From the user point of view, Sisu's SLURM system is easier to use than the Doctor's SGE system. The reservation of full nodes seemed to be more straightforward than SGE's approach with overbooking resources. SLURM seems to fit better with the present highly scalable codes than SGE. Actually, Sisu is mainly intended for massively parallel codes, not for example for commercial FEM packages, whose parallel performance is still quite limited. In addition, Sisu is such a large system that reservation by full nodes is much more straightforward solution than reserving the resources by slots, i.e. basically by cores. Hence, SLURM fits better to larger clusters than SGE, and for those cases, in which resource overloading is not acceptable.

2.3 HTCondor in a workstation and PC network

2.3.1 Description of the computational environment

The common practice nowadays in research and development is that product development engineers and researchers have a personal computer, i.e. a desktop computer, laptop, or a workstation, as their primary tool for modelling and simulation. In addition, more powerful computational resources, such as computation servers and cluster systems, may be available for large and resource intensive computational cases. Depending on the company policy on computational hardware investments and user's personal preferences, the local computational infrastructure can be heterogeneous. Computers may have different amount of main memory and disk capacity, different number of processors and processor cores, different operating systems, and varying set of software applications. In addition, the utilisation rate of individual computers in the local office network may vary between users and in time. This sets difficult constraints for efficiently utilising the computational resources of the office network. Still, a company design office can have remarkable overall computational resources when utilised properly (Figure 10).

Utilisation of the distributed computational resources in an office network is demanding for the end users. The heterogeneous hardware and software environment sets challenges for selecting appropriate computing resources and managing computational load balancing. In addition, when utilising personal computers, the primary user must have the highest priority in using his or her local computing resources. This, on the other hand, means that the availability of PC resources may vary in time, which makes the utilisation of these resources even more difficult. Flexible distributed resource management system may provide a solution for this. HTCondor (see Section 1.3.1.2), a distributed resource management system, was tested for the use in a heterogeneous office network. The test environment for the test contained a workstation and a laptop computer (Figure 11). Technical details of the test environment computers are given in Table 2. The workstation computer was the HTCondor computational pool Central Manager and the laptop computer was a Submit client. Both computers were connected to a local area network (office network) that provided centralised domain name service (DNS) and dynamic host configuration protocol (DHCP) service. The virtual machine (in which the Submit client was installed) used the host computer's network in bridged mode by

replicating the physical network connection state. The virtual machine was not in DNS, which meant that the HTCCondor master (the workstation system) could not communicate with the client computer using only hostnames.

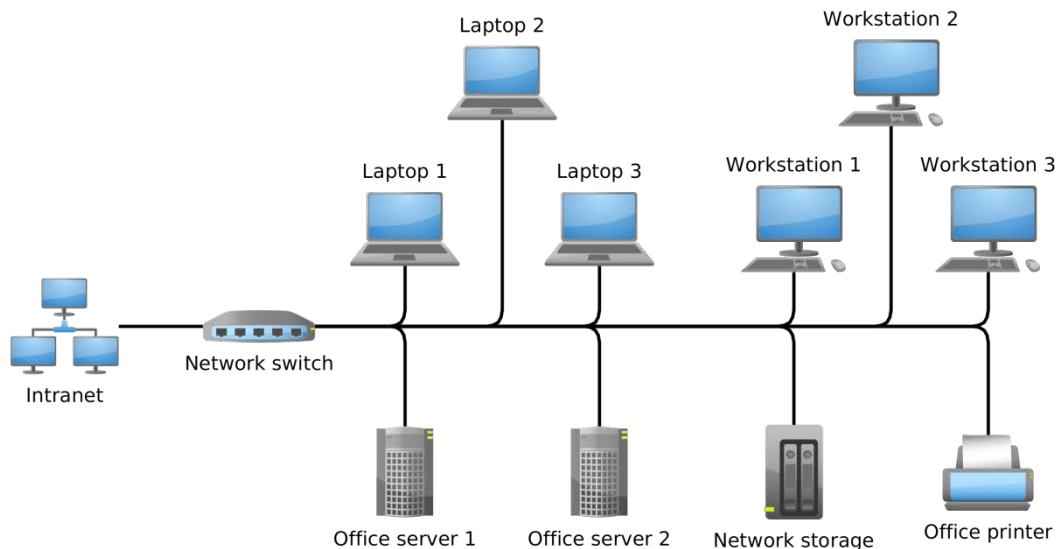


Figure 10: A schematic picture of an office computational infrastructure. The network can be heterogeneous both from computational hardware and software point of view.

Table 2: Technical details of the computers in the test environment.

Feature	Workstation	Laptop computer
Manufacturer and model	Dell WS7500	Dell Latitude E6520
CPU	2 × Intel Xeon X5690, 3.46 GHz	Intel Core i7-2760QM, 2.4 GHz
# of computational cores	12 (12 threads)	4 (8 threads)
Main memory	96 GB	8 GB
Disk space	3 TB	500 GB
Operating system	Ubuntu Linux version 12.04 LTS, 64 bit	Host: Microsoft Windows 7 Enterprise, 64 bit Virtual machine: Ubuntu Linux version 12.04 LTS, 64 bit
HTCondor version	8.0.4 x86_64 Ubuntu12	8.0.4 x86_64 Ubuntu12
HTCondor roles	Central Manager, Execute node, Submit client, Checkpoint Server	Submit client

2.3.1.1 Configuration of the computational pool

The challenge of utilising computational resources in a heterogeneous office network has been discussed in Section 1.2.2 *Managing workstation, desktop, and server computer resources*. The variation of computer hardware and software in the local network computers, and the availability of the computational resources of personal computers make it very chal-

lenging to utilise these resources using a computational queue based approach. The operating principle of HTCondor makes all this relatively easy for the user.

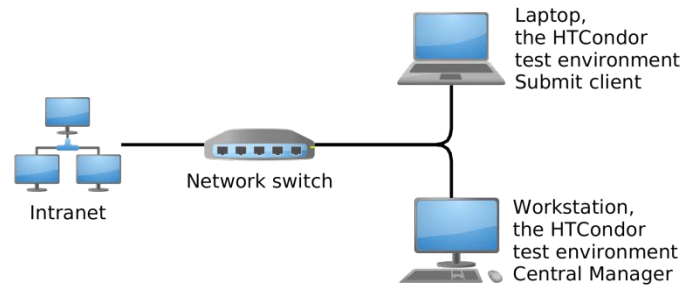


Figure 11: The HTCondor test environment.

The workstation in the HTCondor test environment was defined to be the computational pool Central Manager, i.e. the workstation took care of collecting information from the computational nodes in the computational pool, resolving computational requirements for the submitted jobs and balancing the computational load in the pool. In addition, the workstation was defined to be the Execute node of the computational pool and a Submit client in the system. The laptop had only one role (submit): computational jobs could be submitted and the pool could be monitored and managed from it. An example printing of the `condor_status` utility of the HTCondor test environment, executed from the computational pool client:

Name	OpSys	Arch	State	Activity	LoadAv	Mem	ActvtyTime
slot10@esvvm50019.	LINUX	X86_64	Unclaimed	Idle	0.000	8056	3+00:25:47
slot11@esvvm50019.	LINUX	X86_64	Unclaimed	Idle	0.000	8056	3+00:25:48
slot12@esvvm50019.	LINUX	X86_64	Unclaimed	Idle	0.000	8056	3+00:25:49
slot1@esvvm50019.a	LINUX	X86_64	Unclaimed	Idle	0.000	8056	0+00:49:34
slot2@esvvm50019.a	LINUX	X86_64	Unclaimed	Idle	0.000	8056	2+20:20:41
slot3@esvvm50019.a	LINUX	X86_64	Unclaimed	Idle	0.000	8056	2+20:20:42
slot4@esvvm50019.a	LINUX	X86_64	Unclaimed	Idle	0.000	8056	2+20:20:15
slot5@esvvm50019.a	LINUX	X86_64	Unclaimed	Idle	0.000	8056	2+20:20:44
slot6@esvvm50019.a	LINUX	X86_64	Unclaimed	Idle	0.010	8056	3+00:25:51
slot7@esvvm50019.a	LINUX	X86_64	Unclaimed	Idle	0.000	8056	3+00:25:52
slot8@esvvm50019.a	LINUX	X86_64	Unclaimed	Idle	0.000	8056	3+00:25:45
slot9@esvvm50019.a	LINUX	X86_64	Unclaimed	Idle	0.000	8056	3+00:25:46
Total Owner Claimed Unclaimed Matched Preempting Backfill							
X86_64/LINUX	12	0	0	12	0	0	0
Total	12	0	0	12	0	0	0

The printing shows that the HTCondor test environment has totally 12 slots (processor cores), which are running the Linux X86-64 operating system, and all the cores are in idle state (at the time of printing).

The properties of the computer's HTCondor installation are defined in the HTCondor configuration files. In a Linux system, these files are located in the directory `/etc/condor`. The main configuration files are:

- `condor_config`: The main configuration file that contains all the relevant configuration variables. The file is divided into four sections, based on the importance likeliness of the variables to be needed to change. The changed variables are presented in Table 4.
- `condor_config.local1`: The system's local configuration file that contains only the main variables for the computer. The defined variables for the Central Manager computer are presented in Table 4 and for the Submit client computer in Table 5.

Table 3: The changed variables in the condor_config configuration file for the Central Manager and Submit client computer.

Variable and value	Notes
UID_DOMAIN = ad.vtt.fi	The common Internet domain for the pool
ALLOW_WRITE = *.ad.vtt.fi	Allow all the computers in the domain to have write access to the pool (required for e.g. submitting jobs to the pool)
DEFAULT_DOMAIN_NAME = ad.vtt.fi	Define the default domain name for the computers in the pool; this allows alias names to be used even though they are not defined in NIS or in the /etc/hosts file
NO_DNS = TRUE	Enable computers that are not in DNS to use the pool (this was the case with the HTCCondor test environment for the client computer)
TRUST_UID_DOMAIN = TRUE	Define the HTCCondor to trust that the client given UID_DOMAIN is correct; this setting is not secure!

Table 4: The variables set in the condor_config.Local configuration file for the Central Manager computer.

Variable and value	Notes
CONDOR_HOST = ESPVM50019.ad.vtt.fi	The Central Manager computer
COLLECTOR_NAME = VTT/TK3047 workstation - \$(FULL_HOSTNAME)	Description for the collector system (this is visible in the pool status for the collector)
START = TRUE	Start the local HTCCondor system daemons on system start-up
SUSPEND = FALSE	Do not allow job suspend
PREEMPT = FALSE	Do not allow killing nicely jobs
KILL = FALSE	Do not allow killing instantly jobs
DAEMON_LIST = COLLECTOR, MASTER, NEGOTIATOR, SCHEDD, STARTD	Local system daemons to be started
ADAMS_SOLVER_LIMIT = 2	This variable was used for defining the maximum number of parallel MSC Adams processes in the pool to be two

Table 5: The variables set in the condor_config.Local configuration file for the Submit client computer.

Variable and value	Notes
CONDOR_HOST = ESPVM50019.ad.vtt.fi	The Central Manager computer
COLLECTOR_NAME = Personal Condor at \$(FULL_HOSTNAME)	Description for the collector system (this is visible in the pool status for the collector)
START = TRUE	Start the local HTCCondor system daemons on system start-up
SUSPEND = FALSE	Do not allow job suspend

PREEMPT = FALSE	Do not allow killing nicely jobs
KILL = FALSE	Do not allow killing instantly jobs
DAEMON_LIST = MASTER, SCHEDD	Local system daemons to be started
ADAMS_SOLVER_LIMIT = 2	This variable was used for defining the maximum number of parallel MSC Adams processes in the pool to be two

HTCondor recognises the computers in the computational pool based on the computers host-name and IP address. For the Central Manager computer to be able to communicate with the other computers in the pool by using hostnames, a domain name service (DNS) is needed or the hostnames of the hosts in the pool must be otherwise defined, e.g. in the operating system's `hosts` file. If no DNS is available, the HTCondor system has to be defined accordingly (variable `NO_DNS = TRUE` in the `condor_config` file). If there are problems in configuring HTCondor, the log files should be studied and especially the hostnames and IP addresses checked carefully.

The default configuration of the HTCondor system for Ubuntu Linux systems is targeted for a local single computer computational pool. In the main configuration file (`condor_config`), the pool write variable (`ALLOW_WRITE`) is set by default to point to the local computer. To enable other computers to submit jobs to the pool, the variable needs to be defined to include all the allowed submitting computers in the pool (e.g. `ALLOW_WRITE = *.ad.vtt.fi`).

2.3.1.2 Using the computational pool

Submitting computational jobs to the HTCondor computational pool is described in detail in the HTCondor Manual, User's Manual section [38]. A Job is defined with a *job description file* and it is submitted from command line with the `condor_submit` command. The job description file defines in minimum:

- The command or program to be executed (the `executable` keyword)
- The queuing command (the `queue` keyword)

If no other keywords are defined, the job will be submitted to the vanilla HTCondor universe and standard out and standard error channels are not recorded. A universe defines the execution environment for the HTCondor jobs. Different universes have different features and e.g. the standard universe enables checkpoint procedure (a mechanism to pause a job and move the execution to another computational node) to be used which is not available for the jobs run in e.g. the vanilla universe. In this document, only the vanilla universe has been used. The details of the HTCondor job description file are given in the HTCondor Manual. An example of more typical simple job description file (`example_01.job`) is:

```
executable = test.sh
universe   = vanilla
output     = test.out
log        = test.log
queue
```

This job description file defines that a shell script `test.sh` is executed in the vanilla universe, and the standard output is recorded into the `test.out` file. The HTCondor job log is recorded into the `test.log` file. The job is submitted from the command line with the command:

```
%> condor_submit example_01.job
```

In this example case, successful job submission into the computational pool returns:


```
Submitting job(s).
1 job(s) submitted to cluster 126.
```

This tells that the job has been submitted, there are one jobs defined in the job description file and the job ID for this case is 126; the job ID can be used for getting more information of the status of this particular job or e.g. removing the job from the computational pool. For more information, see the HTCondor documentation.

2.3.1.3 How HTCondor treats job files

HTCondor runs the computational jobs in the execution nodes in a dedicated temporary execution directory. To enable the execution, HTCondor copies by default the defined executable file from the submit computer to the execution node's execution directory. In addition, the explicitly defined input files are copied to the same directory. The copying of the executable file can be prevented in the job description file with the option:

```
transfer_executable = false
```

Another way to execute a program is to set HTCondor to use the execution node's local software installation. This is done by defining the job with a *driver script*. An example of such a driver script (a bash shell script) for running the MSC Adams Solver is presented below:

```
#!/bin/bash
adams2013_2 -c ru-s i case.acf exit
exit
```

In this case, HTCondor copies the driver script from the submit computer to the execution node and executes it. To explicitly define the input files to be transferred to the HTCondor execution node, the HTCondor *File Transfer mechanism* can be used. In the job description file, the File Transfer mechanism needs to be defined to be on, HTCondor has to be told when the files are transferred, and the files to be transferred need to be listed, e.g.:

```
should_transfer_files = yes
when_to_transfer_output = on_exit
transfer_input_files = file1, file2
```

2.3.2 Practical examples and how the systems works

2.3.2.1 Example 1: Simple single simulation run in the local computer

MSC Adams [48] is a multibody system simulation software package for simulation and analysis of mechanical systems, such as vehicle dynamics and handling. The simulation cases with the software are typically relatively fast, taking from couple of tens of seconds to half an hour to complete.

In this example, a simple MSC Adams simulation case is run in the local computer using the HTCondor local universe. This makes HTCondor to execute the computational job immediately in the job submitting directory. The MSC Adams software is a large software package, so it is convenient to run the simulation using the execution node local installation of the software instead of transferring the whole software package together with the computational job. In this case, an executable script is used for running the software application. The simulation case execution standard output is recorded into the `run_case.out` file and the HTCondor log is recorded into the `run_case.log` file. The whole job description file for the example is:

```
executable = run_case.sh
universe   = local
output     = run_case.out
log        = run_case.log
queue
```

Notice that when local universe is used for executing a case, no file transfer is done and no actual pooling is done for the job. This means also that the HTCondor's concurrency mechanism does not apply (see example 2 and HTCondor manual for more details about the concurrency mechanism). This case can be seen as a simple test for case execution.

2.3.2.2 Example 2: Single simulation run in the computational pool with concurrency limits

In this example, more advanced features of the HTCondor system than in example 1 are used for running a single computational case. The example introduces how transferred files are defined in HTCondor and how concurrent execution of processes can be controlled.

One common analysis type with the MSC Adams simulation software is to run parameter studies to gather more information of the simulated system behaviour and to e.g. optimise the simulated system's performance. Large parameter studies with the software are so-called embarrassingly parallel problems, i.e. they do not require any additional effort for case parallelisation and they are very suitable for execution in an HTCondor computational pool.

When the analysis is run in the HTCondor vanilla universe (the HTCondor execution environment for closed, black box software applications and scripts), all the necessary files have to be provided for the system. HTCondor will transfer the files to the execution node's execution directory and run the defined execute file. In the case of MSC Adams, the minimum is to provide the solver model file (.adm) and in most cases a separate solver command file (.acf). If MSC Adams is used for vehicle simulation, an additional tyre model component is needed, which requires additional data files for defining tyre properties and the road. In this case, the HTCondor file transfer mechanism is used for transferring the necessary input files for the simulation case to the execution node. For this, the keywords `should_transfer_files`, `when_to_transfer_output`, and `transfer_input_files` are defined, and the related parameters are given (the `simulation_case.acf` file as the MSC Adams solver command file, the `simulation_model.adm` file as the simulation model description for the solver, the `roaddata.rdf` file as the road definition file, and the `tyredata.tpf` as the tyre property file; see the job description file below).

MSC Adams is a commercial software package that has a licensing mechanism to control how many concurrent software application features are used. For this, the job description file, presented in example 1, needs only modest modifications. The HTCondor computational universe is changed to vanilla and an additional concurrency limit is set for the MSC Adams solver. The concurrency limit is a simple limiting mechanism that needs modifications to the HTCondor local configuration file, e.g. `condor_config.local`. The configuration and use of concurrency limits are described in the HTCondor documentation. In this case, a concurrency limit for MSC Adams solver is defined in the `condor_config.local` file as:

```
ADAMS_SOLVER_LIMIT = 2
```

This limits then maximum number of concurrent processes with this limit to be two. To take the concurrency limit into use for the job, it has to be defined in the job description file as:

```
concurrency_limits = ADAMS_SOLVER
```

This defines that one unit of the named limit is taken for this job. The HTCondor job description file for this case is:

```
executable          = run_case.sh
universe            = vanilla
concurrency_limits  = ADAMS_SOLVER
should_transfer_files = yes
when_to_transfer_output = on_exit
transfer_input_files = simulation_case.acf, \
                    simulation_model.adm, \
```

```
output      roaddata.rdf, \  
log         tyredata.tpf  
queue      = run_case.stdout  
           = run_case.log
```

The concurrency limit mechanism seems to require the use of other HTCondor computational universes than local (e.g. vanilla universe). It may be possible to change this behaviour by modifying the configuration parameters of the HTCondor installation.

This use scenario can be useful if several simulation cases needs to be run and there is only limited number of software licenses available. With this approach, the user can submit the cases into the local computational pool and the HTCondor system takes care that the software licenses are utilised optimally.

2.3.2.3 Example 3: Running HTCondor jobs from DAKOTA

DAKOTA [49] is a software package for large computational analyses, such as parameter studies (e.g. *design of experiment*, DOE, and *design and analysis of computer experiments*, DACE) and computational optimisation. These analyses typically require large number of single computer simulations or analyses. By default, DAKOTA runs the analyses in local computer system. The study can be defined as parallel and the number of parallel processes can be explicitly defined. The DAKOTA package is a general computing package and it has a general, file-based interface for the actual computational tools as well as software application-specific interfaces to some software applications (e.g. Abaqus, MATLAB, and Python).

The single computations in DAKOTA parameter studies are independent of each other (i.e. they are embarrassingly parallel) and can be run in parallel. In case when e.g. software licenses are not limiting the number of parallel processes, the bottleneck is usually the computational resource available in the computer running the DAKOTA analysis. When using the general interface between DAKOTA and the computational software application, the system can be configured so that DAKOTA submits the single simulations to a computational queue, such as the HTCondor computational pool, which then takes care of distributing the computations and balancing the computational load. This strategy is illustrated in Figure 12.

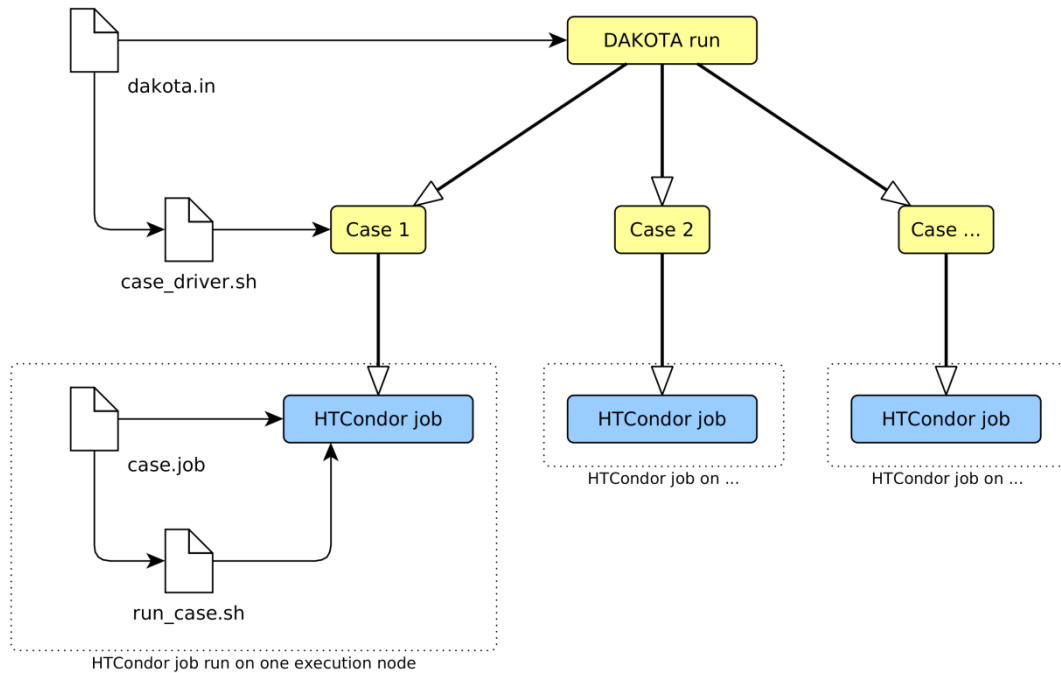


Figure 12: Illustration of the strategy of running single DAKOTA computations using HTCondor.

In the strategy presented in Figure 12, the procedure of running a DAKOTA analysis using HTCondor differs only slightly from the example 2 presented above, i.e. the case is quite similar from the HTCondor point of view, but what the case does differs quite much. Instead of executing the software application from the DAKOTA `case_driver.sh` script, the driver script submits an HTCondor job by a simple call:

```
condor_submit case.job
```

The `case.job` file defines the software application to be run and the files to be transferred between the submit client and the computational nodes. The details and necessary input files are presented in Appendix A.

This strategy of running DAKOTA analysis utilising HTCondor is well suited for computations that are not limited by the number of parallel processes. The DAKOTA can be defined so that large number of parallel processes is allowed and HTCondor takes care of efficiently utilising the computational infrastructure.

2.3.2.4 Example 4: Running DAKOTA as an HTCondor job

When DAKOTA is used for running software applications that have e.g. license limitations for the number of parallel processes, the strategy presented in the previous section does not necessarily provide any added value but has small additional execution latency due to the delays in submitting single jobs through HTCondor. If it is preferable to utilise existing computational resources (others than the local computer), a better strategy can be to run the DAKOTA analysis through one HTCondor job. In this approach, all the DAKOTA cases are run in the same computer and new cases are initiated immediately after the previous process has ended, but still taking care of the maximum number of parallel processes (for this, DAKOTA's asynchronous evaluation concurrency mechanism can be used; for more information, see the DAKOTA documentation). This strategy is illustrated in Figure 13. The input files for this example are listed in Appendix A.

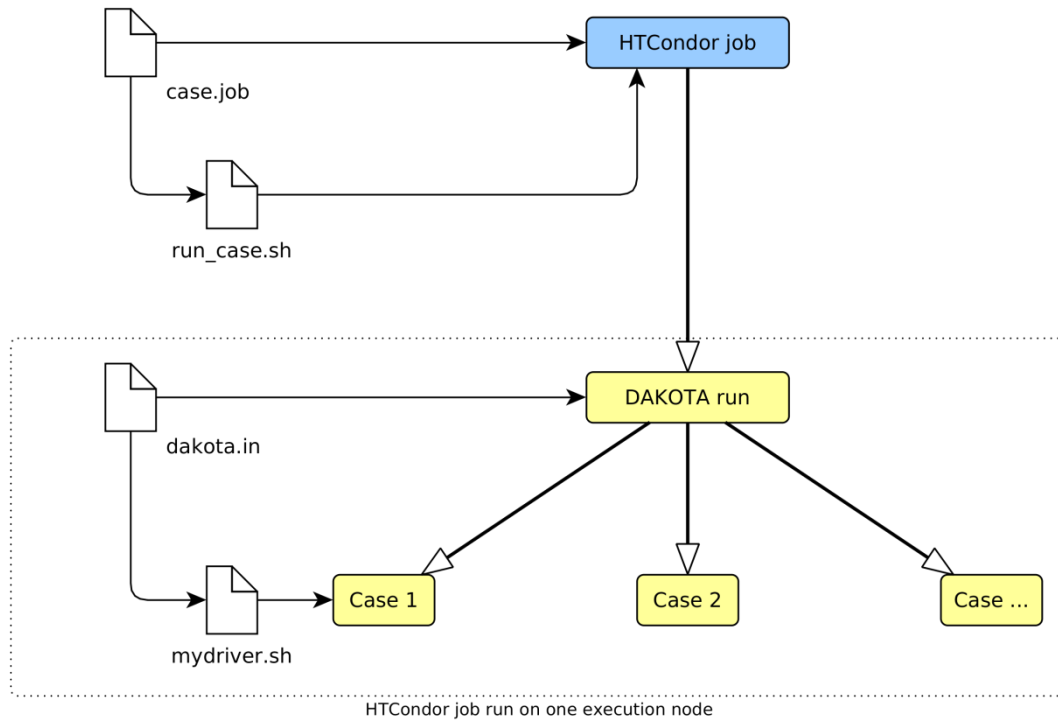


Figure 13: Illustration of the strategy of running a DAKOTA analysis using one HTCondor job.

2.3.2.5 Example 5: Local grid computing approach, a “fire-and-forget” scenario

The function of an HTCondor pool requires that the members of the pool that are related to the particular job execution are available and the local HTCondor daemon processes of the members are running. In the case where HTCondor is used from a laptop computer, it is likely that this computer is not available all the time, which would prevent this scenario to work. To enable flexible utilisation of the computational pool resources, another mechanism, *HTCondor-C grid*, is available. In this approach, jobs are first submitted to the Submit computer's internal local pool. With additional parameters, the internal local Central Master submits the job further into the grid pool (i.e. the workstation network local pool) for actual execution. This approach is illustrated in Figure 14.

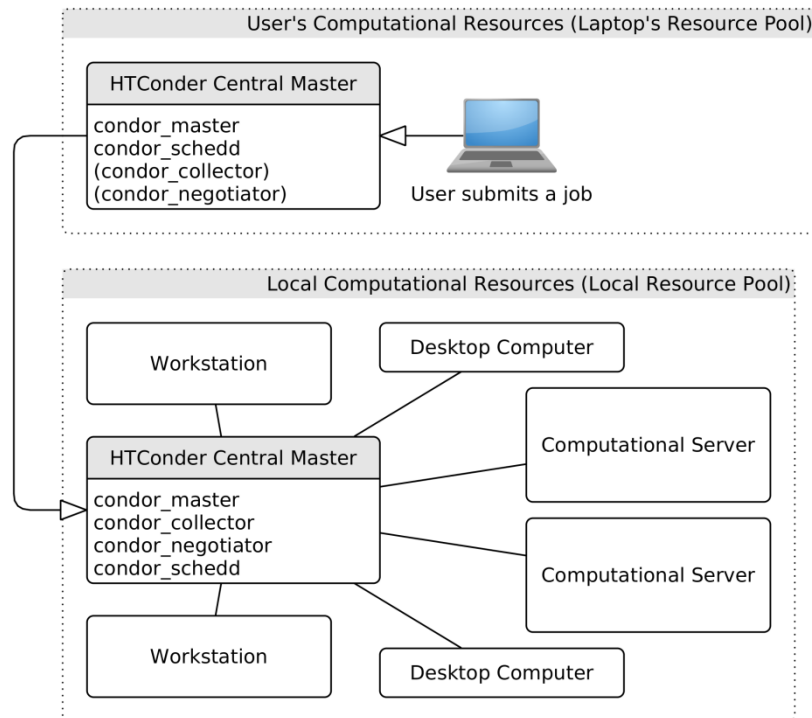


Figure 14: Utilising HTCondor grid computing features, HTCondor-C, for flexible computing.

This approach enables a “fire-and-forget” scenario for computing. After the job is transferred into the grid pool (i.e. the workstation network local pool), the Submit client can be excluded from the local pool. The local pool takes care of the job and the results will be waiting for the Submit client to be available again in the grid pool. When this happens, the job results are transferred back to the Submit client.

The grid computing approach with HTCondor-C grid mechanism requires additional definitions both in the Submit client's internal local pool as well as in the HTCondor workstation network local pool. This is described in the *Grid Computing* chapter of the HTCondor manual [38]. In addition, the job description needs to have additional definitions for grid computing. An example of this approach is presented in Appendix A, where the above described example 4 is executed in a grid.

2.3.3 Lessons learned with the HTCondor system

The HTCondor configuration is sensitive to the network configurations. The daemon log files, both in the local computer and the computational pool Central Master, are a good source for debugging possible configuration problems. Connections via virtual private network (VPN) to the computational pool can cause problems, especially when the local system is running in a virtual machine, such as Oracle VirtualBox or VMware Player. Virtual machines can utilise network connection of the host system using several different configurations, such as network address translation (NAT) or bridged networking, latter with or without replicated host network interface. Depending on the HTCondor configuration, the computational pool may or may not accept connections.

2.3.4 Future steps for local HTCondor system development

HTCondor provides a flexible and scalable solution for utilising heterogeneous computational resources that are available in office networks. The fundamental approach in HTCondor, i.e. each computational job is “buying services from the computational pool based on the individual requirements of the job” is well-suited for ever-changing computational environment that still offers good computational performance. One of the challenges in HTCondor is the lack of

graphical tools for all the operations. A simple graphical user interface would be especially useful for job and queue monitoring. The learning curve for the command line user interface can be high for those who have not been using it for e.g. scripting.

2.4 Techila in a workstation and Azure network

2.4.1 Overview

The VTT Techila environment consists of a *Techila Server* (for taking care of load balancing and distributing the work load), *Techila Workers* that do the actual computations, and *Techila User Interface*, where the users can monitor the operations. In Figure 15, there is a screenshot of the VTT Techila Server Status view including the status of the Workers.

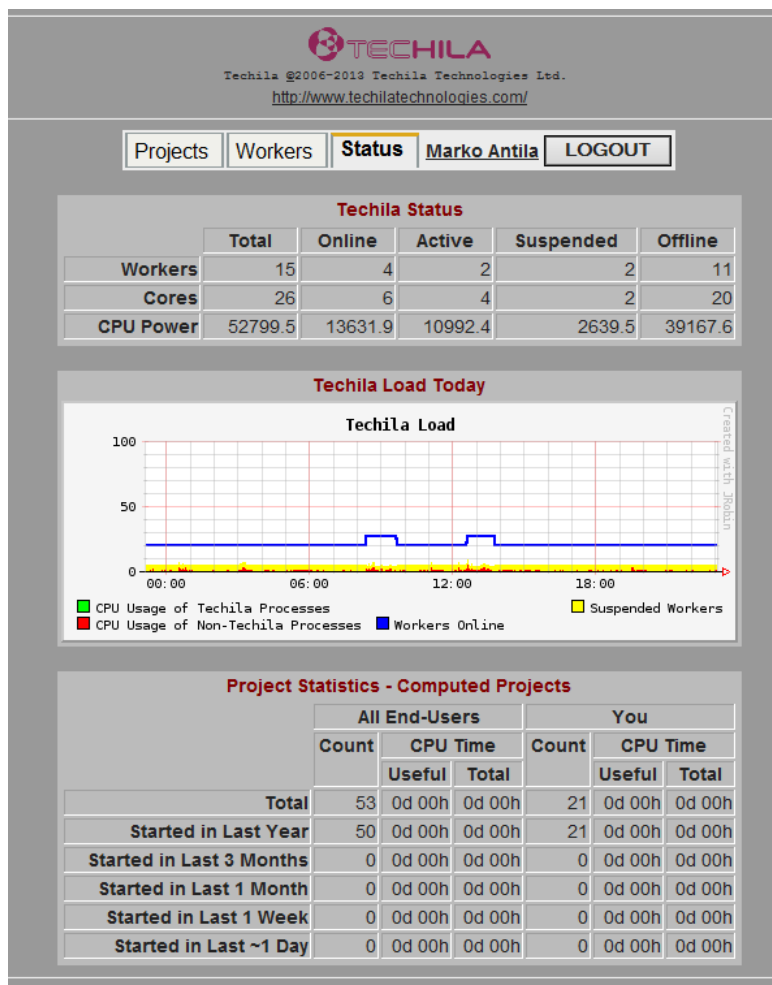
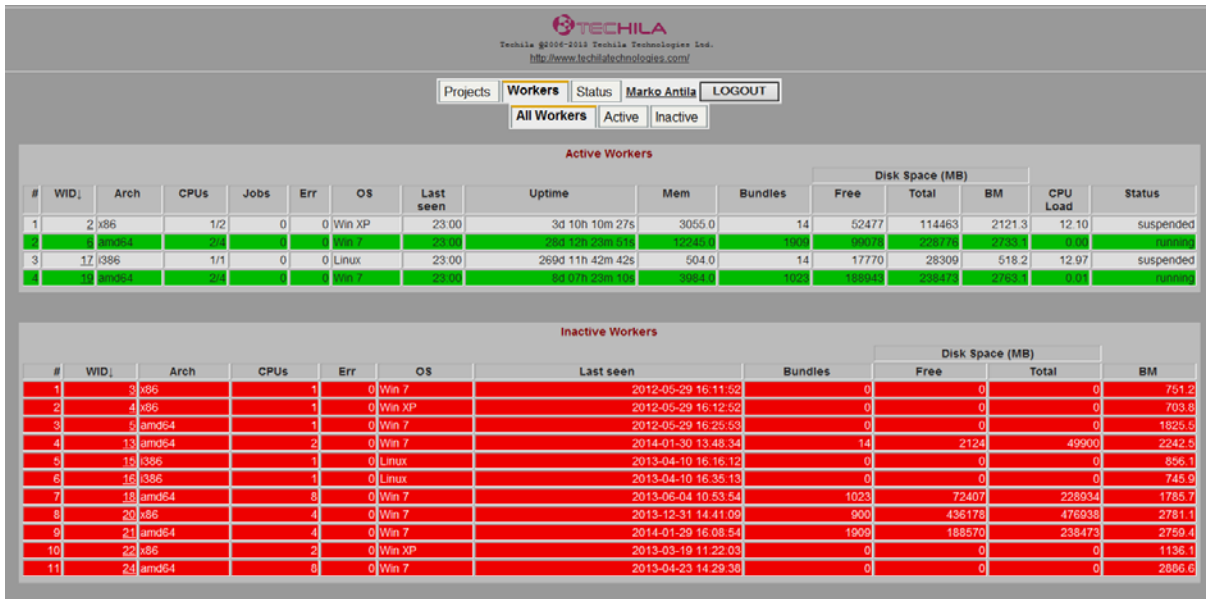


Figure 15: VTT Techila Server Status view.

Currently there are only few VTT's own Techila Workers as can be seen in Figure 15. Additionally, VTT has also tested Microsoft's Azure cloud service, with its own Server and User Interface. VTT has currently access to Azure cloud service and demanding computations can access it as needed. Techila tasks can be directed to that service instead of VTT's own Techila Workers, and this may also be more cost-effective and reliable solution.



The screenshot shows the VTT Techila Workers interface. At the top, there is a navigation bar with 'Projects', 'Workers', 'Status', 'Marko Antila', and 'LOGOUT'. Below this, there are buttons for 'All Workers', 'Active', and 'Inactive'. The main content is divided into two sections: 'Active Workers' and 'Inactive Workers'.

Active Workers Table:

#	WID	Arch	CPUs	Jobs	Err	OS	Last seen	Uptime	Mem	Bundles	Disk Space (MB)			CPU Load	Status
											Free	Total	BM		
1	2	x86	1/2	0	0	Win XP	23.00	3d 10h 10m 27s	3055.0	14	52477	114463	2121.3	12.10	suspended
2	17	amd64	1/1	0	0	Win 7	23.00	26d 12h 23m 51s	12245.0	1909	99078	228276	2733.1	0.00	running
3	17	x86	1/1	0	0	Linux	23.00	269d 11h 42m 42s	504.0	14	17770	28309	518.2	12.97	suspended
4	17	amd64	1/1	0	0	Win 7	23.00	8d 07h 23m 10s	3964.0	102	189843	238473	2763	0.01	running

Inactive Workers Table:

#	WID	Arch	CPUs	Err	OS	Last seen	Bundles	Disk Space (MB)			BM
								Free	Total	BM	
1	3	x86	1	0	Win 7	2012-05-29 16:11:52	0	0	0	751.2	
2	4	x86	1	0	Win XP	2012-05-29 16:12:52	0	0	0	703.8	
3	5	amd64	1	0	Win 7	2012-05-29 16:25:53	0	0	0	1825.5	
4	13	amd64	2	0	Win 7	2014-01-30 13:45:34	14	2124	49900	2242.5	
5	15	x86	1	0	Linux	2013-04-10 16:16:12	0	0	0	856.1	
6	16	x86	1	0	Linux	2013-04-10 16:35:13	0	0	0	745.9	
7	18	amd64	8	0	Win 7	2013-06-04 10:53:54	1023	72407	228934	1785.7	
8	20	x86	4	0	Win 7	2013-12-31 14:41:09	900	436178	476938	2781.1	
9	21	amd64	4	0	Win 7	2014-01-29 16:08:54	1909	188570	238473	2759.4	
10	22	x86	2	0	Win XP	2013-03-19 11:22:03	0	0	0	1136.1	
11	24	amd64	8	0	Win 7	2013-04-23 14:29:38	0	0	0	2866.6	

Figure 16: VTT Techila Workers (situation on 30 Jan 2014). Only four active Workers, with only one Worker with adequate amount of memory (Worker #2).

2.4.2 Launch of the grid computing

The grid computations can be launched in several ways, but three methods were the most suitable for the purposes of this study:

1. Direct launch from the MATLAB using MATLAB functions;
2. Using MATLAB as a pre-compiled binaries launching platform; and
3. Using CLI (Command Line Interface) to run pre-compiled binaries.

In practise, the first method was used in comparison of the grid performance, meaning that Techila Workers were launched only inside MATLAB. The most important function for this was GridFor, which was later changed to more contemporary CloudFor. From the functional point of view these two are equal. CloudFor is a MATLAB function, which automatically divides the parallel computations to the Techila Workers that have been set up for that system. Techila is in such a way more integrated in certain environments than many other resource managers.

2.4.3 Techila test cases overview

Two test cases were initially planned: test cases A and B. Test case A was an engineering test case using specific MATLAB functions. The test case A was run on Azure cloud due to the limited availability of VTT-specific Workers. The test case B was a more challenging but yet interesting case to generate Techila-compatible code directly from SIMULINK graphical model. The test case A was successfully run but there were practical difficulties with test case B. Finally, only test case A was successfully run.

2.4.4 Test case A

The plan was to create an engineering test case using MATLAB functions Peach or GridFor. Test case A final realization was an engineering test case using MATLAB function CloudFor (initially GridFor), and also for a comparison the MATLAB native ParFor function. The test case was realised using the Shotgun Optimiser core code [50]. Shotgun Optimizer applies multi-level random search optimisation to find the global optimum of the target function within the given parameter space. The realisation principle of the Shotgun Optimiser is to compute 3-round random selection of the parameters within a selected parameter space. Each

successive round is based on the results of the previous round, and finally a global optimum is found. Shotgun Optimiser is an example of resource-hungry embarrassingly parallel application, and thus inherently suitable for this evaluation. The code for such an optimiser is available in MATLAB and it was a good solver for this test case.

The system, subject to optimisation, will be a feedback compensation filter for a time-invariant linear system. The system response is first obtained from a model or from a measurement, or a pre-existing response is used. The response may be derived from a mechanical or acoustic system. Then, a relatively complex compensation filter is used for feedback compensation. Finally, the parameter space, boundary conditions and the optimisation resolution are set.

There were three different types of test runs:

1. A local solution using only the local computer resources and one core. Conventional for loops and similar structures were used.
2. A local solution using local computer resources and several cores. MATLAB-based parallelisation was utilised (ParFor).
3. A Techila Grid solution, in which CloudFor loops were utilised.

2.4.5 Test case A results

The local solution (test run type 1) gave a baseline for the results. The local ParFor solution (test run type 2) did not speed up the computations. The acceleration factor of the Techila runs is an internal Techila indicator and it is a ratio between non-parallelised and parallelised computations.

With Azure environment it was possible to carry out relatively large tests using CloudFor (test run type 3). The use of CloudFor was only compared to regular for loops, because the ParFor loop did not speed up the computation in any relevant amount in a 2-core test computer. One important finding with the Azure cloud was its sensitivity to 32- or 64-bit environments. If the launching end user computer was 32-bit, the system did not use Workers effectively, but it was necessary to separately configure it to utilise 32-bit Workers. This issue is becoming now less important due to the transfer to 64-bit systems in general.

Additional tests were carried out to study the behaviour of the Techila Azure cloud. Results are presented in Table 6, Table 7, and Table 8. For Table 6, simple functions were carried out as a pre-test. The acceleration factor is clear, and there is a saturation point for the acceleration factor above 100 000 iterations. For Table 7, more demanding computations were involved. For such a case, the acceleration factor was even higher up to one million iterations. For Table 8, a test case resembling a real Shotgun Optimizer core code was run, with almost 8000 jobs and 10 Workers. Such a demanding computation was accomplished in about 13 minutes wall clock time. Based on this, it seems that especially Monte Carlo type of simulations and optimisation cases that utilise algorithmically simple solvers benefit most of the Techila gridification (cloudification).

Table 6: Results of an Azure pre-test.

Description	Iterations	Jobs	Workers participated	CPU time used (s)	Wall clock time used (s)	Acceleration factor	Comments
Simple one iteration loop	1	1	1	2	192	0.01	At initialisation the overhead may be very large
More	10	1	1	2	8	0.25	Wall clock time varies a lot with

iterations								small projects (unpredictable overhead)
Repeat of the previous test	10	1	1	2	4	0.50		
Second repeat of the previous test	10	1	1	2	4	0.50		
More iterations...	100	1	1	2	3	0.67		No difference in CPU time
And more	1000	1	1	2	4	0.5		Still very light load
...more	10 000	1	1	2	4	0.5		
... more	100 000	6	2	13	5	2.6		Positive acceleration
... more	1 000 000	64	4	161	56	2.88		

Table 7: Results of an Azure test with a harder-to-calculate function and larger memory need.

Description	Iterations	Jobs	Workers participated	CPU time used (s)	Wall clock time used (s)	Acceleration factor	Comments
More iterations	10	1	1	2	14	0.14	Wall clock time varies a lot with small projects (unpredictable overhead)
More iterations...	100	2	1	4	4	1.00	No difference in CPU time
And more	1000	1	1	2	4	0.5	Still very light load
...more	10 000	1	1	2	4	0.5	
... more	100 000	6	2	13	4	3.25	Positive acceleration
... more	1 000 000	64	10	159	14	11.36	
... too much	10 000 000	-	-	-	-	-	Memory limit exceeded, might have been possible to circumvent with <code>%cf:force:largedata</code>
... too much	2 000 000	-	-	-	-	-	Memory limit exceeded, might have been possible to circumvent with <code>%cf:force:largedata</code>
Down to 1 000 000 again	1 000 000	81	10	200	21	9.52	

Table 8: Results of a large-scale Azure test, using the core loop of a Shotgun Optimizer code.

Description	Iterations	Jobs	Workers participated	CPU time used (s)	Wall clock time used (s)	Acceleration factor	Comments
Shotgun	1 000 000	7693	10	24 817	881	28.17	

optimizer
core code

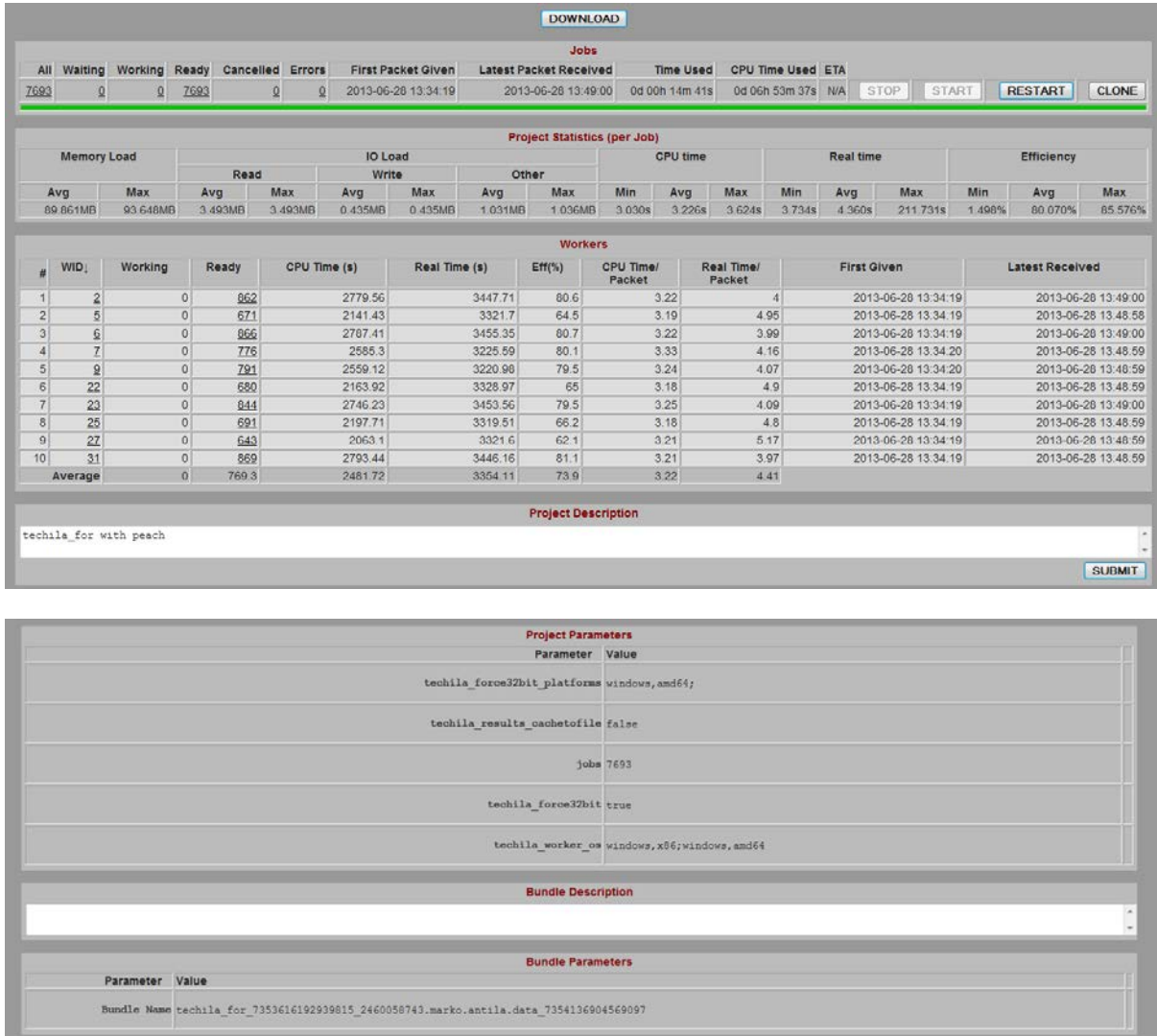


Figure 17: Azure Techila Workers in a Shotgun Optimizer core code test.

2.4.6 Test case B

Test case B was planned to be an engineering test case using MATLAB as a pre-compiled binary launching platform. The first idea was to use the VTT’s “The Doctor” cluster. However, the cluster was in full usage and the plans were changed. The updated plan used Simulink-generated code and it was supposed to run on Techila. However, despite of the instructions received from Techila and the example model (my_model) the compilation was not successful. The code to run the executable for Techila was:

```
function run_test()
% Local Control Code. Creates and computes a simulink model on the Workers.
% Simulink example used in the example can be found here:
%
% http://www.mathworks.se/support/solutions/en/data/1-6M3F5F/index.html?pro
% duct=CO&solution=1-6M3F5F
%
% To run the example, use command
%
% run_test()
```

```

jobs=5 % Defines the number of Jobs.

peach('Simulink Example', {'%P(jobidx)'},{ 'my_model.exe', 'newmain.ctf', 'mymodel_param.mat'},
1:jobs,...
  'Binaries', {{'newmain.exe', 'Windows', 'x86'}},... % Windows 32-bit executable binary
  'Executable', 'true',... % Specify that the funcname refers
  % to a precompiled binary.
  'OutputFiles', {'simulink_output.mat'},... % Define the name of the output file
  % which will be transferred from the
  % Workers
  'StreamResults', 'true',... % Stream results
  'ReturnResults', 'true',...
  'CallbackMethod', @cbfunc,... % Name of the callback function
  'MatlabRequired', 'true') % Matlab Runtime components required
end

function cbfunc(f) % Callback function
[pathstr, name, ext, versn] = fileparts(f); % Retrieve information on result file
% Copy the result file
copyfile(f, ['result_' num2str(bitand(str2num(name), 2^32-1)) '.mat'])
end

```

The Techila project is a complete package of executable, control and other codes which run on Techila system. The procedure to generate the Techila project from a Simulink model (`my_model`) was:

1. The Simulink model is compiled. If the MATLAB CTF file is not properly generated, it may be generated with:

```
mcc -m -C newmain.m
```

2. The resulting files should be:

```

- my_model.exe
- newmain.exe
- newmain.ctf
- mymodel_param.mat

```

3. The file `Run_test.m` is copied to working directory; and
4. A new project is created with the function call `run_test()`.

The project runs the `newmain.exe` programme with various parameters. The streaming of the results is used. Results should be in `.mat` files. In practise, this did not work properly. The project was sent to Techila server, but no real operation was done:

```

>> run_test

jobs =

    5

Grid initialized in 10.940 seconds.
Executable CRC sum computed in 0.084 seconds.
Creating 3 databundle(s)...
Datafile bundle 1 created in 1.167 seconds.
Executor bundle created in 0.418 seconds.
Parameter bundle created in 0.256 seconds.
Project ID 82 created in 0.347 seconds.
Streaming results....

Project completed and results streamed in 297.249 seconds.
Project removed in 0.036 seconds.

Total time used 0 d 0 h 5 m 10 s.

Project Statistics:
  0 nodes participated
Avg efficiency per job:    0.00%
CPU Time per job:        0.000s (min) 0.000s (avg) 0.000s (max)

```

Memory used per job:	0.000MB (avg)	0.000MB (max)
I/O read per job:	0.000MB (avg)	0.000MB (max)
I/O write per job:	0.000MB (avg)	0.000MB (max)
Average total I/O per job:	0.000MB	(NaNMB/s)

The reason for this remained unclear, but due to these difficulties the test case B was dropped and only test case A was used as a real test case.

2.4.7 Conclusions and summary

A Techila environment offers interesting possibilities to launch grid and cloud computations also with relatively little experience on distributed computing. In this context, Techila was launched within MATLAB and with relatively small solver computational complexity (processing power and memory requirements) but with a large scale of concurrency. In addition, the interface between the Techila environment and SIMULINK was tested, but it seems to require more attention to make it work properly.

Currently, VTT's own Techila environment is not suitable for large computational jobs. More Workers are needed for those. In addition, the setup of Workers should be similar to the earlier VTT setup of the Techila environment.

The Azure environment could be used for high-performance computations, as a pay-per-use platform. However, the memory limitations with that service could cause difficulties.

3 Acknowledgements

The authors sincerely acknowledge the help and contribution from Eero Kokkonen (VTT), Juha Virtanen (VTT) and Juho Peltola (VTT). Eero Kokkonen made the tests in SGE system with OpenFOAM, and Juho Peltola the scalability tests for it. Juha Virtanen performed the Abaqus tests with SGE and has been developing and improving the SGE submission scripts.

References

- [1] Benioff M, Lazowska E. Computational Science: Ensuring America's Competitiveness. 2005.
- [2] Glotzer S, Kim S, Cummings P, Deshmukh A, Head-Gordon M, Karniadakis G, Petzold L, Sagui C, Shinozuka M. International Assessment of Research and Development in Simulation-Based Engineering and Science. 2009.
- [3] Oden J, Belytschko T, Fish J, Hughes T, Johnson C, Keyes D, Laub A, Petzold L, Srolovitz D, Yip S, Bass J. Simulation-Based Engineering Science. 2005.
- [4] Wikipedia. Supercomputer architecture — Wikipedia, The Free Encyclopedia. Available at: http://en.wikipedia.org/wiki/Supercomputer_architecture. Accessed Oct 22, 2014.
- [5] Wikipedia. Computer cluster — Wikipedia, The Free Encyclopedia. Available at: http://en.wikipedia.org/wiki/Computer_cluster. Accessed Oct 22, 2014.
- [6] Yan M, Chapman M. Comparative study of distributed resource management systems- SGE, LSF, PBS Pro, and LoadLeveler. 2008.
- [7] Yanbin Liu, Masoud Sadjadi S, Liana Fong, Rodero I, Villegas D, Kalayci S, Bobroff N, Martinez JC. Enabling Autonomic Meta-Scheduling in Grid Environments. International Conference on Autonomic Computing (ICAC '08) 2008:199-200.
- [8] Prabhu CSR. Grid And Cluster Computing. Delhi: Prentice-Hall Of India Pvt. Limited; 2008.
- [9] Tritrakan K, Kanchana P, Muangsin V. Jamjuree Cluster: A Peer-to-Peer Cluster Computing System. In: Enokido T, Barolli L, Takizawa M, editors. Network-Based Information Systems: Springer Berlin Heidelberg; 2007. p. 375-384.
- [10] Top 500 supercomputer sites, June 2014. Available at: <http://www.top500.org/list/2014/06/>. Accessed Oct 28, 2014.
- [11] Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. Proceedings, 11th European PVM/MPI Users' Group Meeting; September; Budapest, Hungary; 2004.
- [12] Open MPI: Open Source High Performance Computing. Available at: <http://www.openmpi.org/>. Accessed Feb 7, 2013.
- [13] HTCondor distributed computing software. Available at: <http://research.cs.wisc.edu/htcondor/>. Accessed Feb 8, 2013.
- [14] Oracle Grid Engine. Available at: <http://www.oracle.com/technetwork/oem/grid-engine-166852.html>. Accessed Feb 13, 2013.
- [15] Torque Resource manager. Available at: <http://www.adaptivecomputing.com/products/open-source/torque/>. Accessed Feb 8, 2013.

- [16] Slurm Workload Manager. Available at: <http://slurm.net/>. Accessed Feb 8, 2013.
- [17] Cérin C, Fedak G. Desktop Grid Computing. 1st ed. Boca Raton: Chapman and Hall/CRC; 2012.
- [18] TECHILA TECHNOLOGIES LTD. Available at: <http://www.techila.fi/>. Accessed Feb 8, 2013.
- [19] BOINC - Open-source software for volunteer computing and grid computing. Available at: <http://boinc.berkeley.edu/>. Accessed Feb 8, 2013.
- [20] XtremWeb-CH - The Volunteer Computing Middleware. Available at: <http://www.xtremwebch.net/>. Accessed Feb 8, 2013.
- [21] XtremWeb-HEP. Available at: <http://www.xtremweb-hep.org/>. Accessed Feb 8, 2013.
- [22] XtremWeb: the Open Source Platform for Desktop Grids. Available at: <http://www.xtremweb.net/>. Accessed Feb 8, 2013.
- [23] Foster I, Kesselman C. Computational grids. Cern European Organization for Nuclear Research-Reports-Cern 1998:87-114.
- [24] About the Globus Toolkit. Available at: <http://www.globus.org/toolkit/about.html>. Accessed Feb 12, 2013.
- [25] GridWay Metascheduler. Available at: <http://www.gridway.org>. Accessed Feb 12, 2013.
- [26] DRMAA - Distributed Resource Management Application API. Available at: <http://www.drmaa.org/>. Accessed Feb 13, 2013.
- [27] OGF - The Open Grid Forum. Available at: <http://www.ogf.org/>. Accessed Feb 13, 2013.
- [28] Son of Grid Engine. Available at: <https://arc.liv.ac.uk/trac/SGE>. Accessed Feb 13, 2013.
- [29] Open Grid Scheduler. Available at: <http://gridscheduler.sourceforge.net>. Accessed Feb 13, 2013.
- [30] Univa Grid Engine Core. Available at: <https://github.com/gridengine/gridengine>. Accessed Feb 13, 2013.
- [31] Ferstl F. History of Sun Grid Engine, From Genias to Univa. Available at: <http://www.wherengridengine lives.com/content/history-sun-grid-engine>. Accessed March 22, 2013.
- [32] Florida State University. Distributed Queuing System (DQS). Available at: <http://www.research.fsu.edu/techtransfer/showcase/dqs.html>. Accessed March 22, 2013.
- [33] Templeton D. Oracle Grid Engine: Changes for a Bright Future at Oracle. Available at: https://blogs.oracle.com/templdef/entry/oracle_grid_engine_changes_for. Accessed March 22, 2013.

[34] Univa. Look at Grid Engine Now! Available at: <http://www.univa.com/products/grid-engine-comparison.php>. Accessed March 22, 2013.

[35] Ferstl F. Free isn't Free. Available at: <http://www.wherengridengine.com/content/free-isnt-free-0>. Accessed March 22, 2013.

[36] Scalable Logic - The Open Source Grid Engine Maintainer. Available at: <http://www.scalablelogic.com/>. Accessed March 22, 2013.

[37] Fun With Grid Engine Topology Masks -or- How to Bind Compute Jobs to Different Amount of Cores Depending where the Jobs Run. Available at: <http://www.gridengine.eu/grid-engine-internals/147-fun-with-grid-engine-topology-masks-or-how-to-bind-compute-jobs-to-different-amount-of-cores-depending-where-the-jobs-run>. Accessed March 22, 2013.

[38] HTCondor Version 8.0.4 Online Manual. Available at: <http://research.cs.wisc.edu/htcondor/manual/v8.0.4/>. Accessed March 27, 2014.

[39] OpenPBS Public Home. Available at: <http://www.mcs.anl.gov/research/projects/openpbs/>. Accessed Feb 20, 2013.

[40] The Research IT Services group at Cincinnati Children's Research Foundation and University of Cincinnati College of Medicine. Torque / Moab. Available at: <http://bmi.cchmc.org/resources/software/torque-moab>. Accessed Feb 19, 2013.

[41] The Globus Alliance. Available at: <http://www.globus.org/alliance/>. Accessed Feb 21, 2013.

[42] Alliance TG. Globus Toolkit 5.2.3 Release Manuals.

[43] BOSCO. Available at: <http://bosco.opensciencegrid.org/>. Accessed Oct 22, 2014.

[44] TECHILA INTEGRATION GUIDE. 2012; Available at: <http://www.techilatechnologies.com/wp-content/uploads/2012/05/Techila-Integration-Guide.pdf>. Accessed Oct 22, 2014.

[45] Windows Azure, Azure Execution Models. Available at: <http://www.windowsazure.com/en-us/documentation/articles/fundamentals-application-models/>. Accessed Oct 15, 2014.

[46] Ganglia Monitoring System. Available at: <http://ganglia.sourceforge.net/>. Accessed Oct 20, 2014.

[47] Templeton D. Sun Grid Engine for Dummies. Available at: https://blogs.oracle.com/templedf/entry/sun_grid_engine_for_dummies. Accessed Nov 29, 2013.

[48] MSC Adams version 2014.1. Available at: <http://www.mscsoftware.com/product/adams>. Accessed May 1, 2015.

[49] DAKOTA version 5.4 Online Manual. Available at: <https://dakota.sandia.gov>. Accessed May 1, 2015.

[50] Optimisation of digitally adjustable analogue biquad filters in feedback active control. Proceedings of Forum Acusticum 2005; 2005.

APPENDIX A: Running HTCondor jobs from DAKOTA

3.1 Details for example 3: Running HTCondor jobs from DAKOTA

Below are presented the key input files for the HTCondor example case discussed in section 2.3.2.3 *Example 3: Running HTCondor jobs from DAKOTA*. In the example, DAKOTA launches the case computations into the HTCondor computational pool. The DAKOTA run is controlled with an input file; in this case it is called `dakota.in`. The file is listed below:

```

strategy
  tabular_graphics_data
    tabular_graphics_file 'dakota.dat'
  single_method
method
  multidim_parameter_study
    partitions = 3 3
model
  single
variables
  continuous_design = 2
  lower_bounds -10.0 -10.0
  upper_bounds 10.0 10.0
  descriptors 'x' 'y'
interface
  analysis_drivers 'case_driver.sh'
  fork
    parameters_file 'params.in'
    results_file 'results.out'
    file_save
    work_directory
      named 'case'
      directory_tag
      directory_save
      template_directory 'case_template'
  asynchronous
responses
  descriptors 'z'
  objective_functions 1
  no_gradients
  no_hessians

```

In the listing above, a multi-dimensional parameter study is defined with totally $(3 + 1) \times (3 + 1) = 16$ cases and the `driver_simulate.sh` script has been defined for the analysis driver. In addition, a template directory (`case_template`) is specified, which includes only the executable (in this case `sombrero.py`) and the HTCondor job description file `run_case.job`. All other required files, such as pre- and post-processing utilities `pyprepro.py` and `pypostpro.py` are in the case main directory. The DAKOTA study is executed with the following command:

```
%> dakota -i dakota.in -o dakota.out
```

The analysis drivers are listed below, first the `case_driver.sh`:

```

#!/bin/bash
# DAKOTA function evaluation pre-processing:
../pyprepro.py -i params.in -t ../template.dat -o input.dat
# DAKOTA function evaluation:
condor_submit run_case.job
# DAKOTA function evaluation post-processing:
condor_wait run_case.log

```

```
../pypostpro.py -i output.dat -o results.out  
exit
```

In the above `case_driver.sh` script, first a pre-processing phase is done to define the input values for the DAKOTA function evaluation (a run of one DAKOTA case). Then, the DAKOTA function evaluation is submitted to the computational pool with `condor_submit` command. In the third, post-processing phase, `condor_wait` utility is used. This utility makes the script to wait for the HTCondor job to finish before continuing the execution of the script. In the actual post-processing phase, the result of the DAKOTA function evaluation is processed for DAKOTA. The HTCondor job description file (`run_case.job`) for the case is:

```
executable          = run_case.sh  
should_transfer_files = yes  
when_to_transfer_output = on_exit  
transfer_input_files = input.dat, sombrero.py  
universe            = vanilla  
output              = run_case.out  
log                 = run_case.log  
  
queue
```

The `run_case.sh` file referred in the job description file is:

```
#!/bin/bash  
./sombrero.py -i input.dat -o output.dat  
exit
```

More details about how to use DAKOTA software can be found from the DAKOTA project website (<http://dakota.sandia.gov/>) and from the software documentation.

3.2 Details for example 4: Running DAKOTA as a HTCondor job

Below are presented the key input files for the HTCondor example case discussed in section 2.3.2.4 Example 4: Running DAKOTA as an HTCondor job. This case is otherwise relatively straightforward, except for file transferring. Below is the job description file (`case.job`) for the example:

```
getenv              = true  
executable          = dakota_case.sh  
universe            = vanilla  
log                 = dakota_case.log  
should_transfer_files = yes  
when_to_transfer_output = on_exit  
transfer_input_files = dakota.in, case_template  
transfer_output_files = \  
    case.1, case.2, case.3, case.4, case.5, case.6, case.7, \  
    case.8, case.9, case.10, case.11, case.12, case.13, case.14, \  
    case.15, case.16, case.17, case.18, case.19, case.20, case.21, \  
    case.22, case.23, case.24, case.25, case.26, case.27, case.28, \  
    case.29, case.30, case.31, case.32, case.33, case.34, case.35, \  
    case.36, case.37, case.38, case.39, case.40, \  
    dakota.dat, dakota.out  
queue
```

Execution script, `dakota_case.sh`, is used in this case:

```
#!/bin/sh  
dakota -i dakota.in -o dakota.out  
exit
```

The DAKOTA input file (dakota.in) for this example case is:

```

strategy
  tabular_graphics_data
    tabular_graphics_file 'dakota.dat'
  single_method
method
  vector_parameter_study
    final_point 10.0 -6.0
    num_steps 39
model
  single
variables
  continuous_design 2
    initial_point -10.0 8.0
    descriptors 'x' 'y'
interface
  analysis_drivers 'mydriver.sh'
  fork
    parameters_file 'params.in'
    results_file 'results.out'
    file_save
    work_directory
      named 'case'
      directory_tag
      template_directory 'case_template'
      copy
      directory_save
  asynchronous
    evaluation_concurrency 12
responses
  descriptors 'z'
  objective_functions 1
  no_gradients
  no_hessians
  
```

The rest of the files needed for this example, including mydriver.sh script, are located in the DAKOTA case template directory case_template. The content of the mydriver.sh script is:

```

#!/bin/bash
# DAKOTA pre-processing phase:
./pyprepro.py -i params.in -t ./template.dat -o input.dat
# DAKOTA function evaluation phase:
./sombbrero.py -i input.dat -o output.dat
# DAKOTA post-processing phase:
./pypostpro.py -i output.dat -o results.out
exit
  
```

3.3 Details for example 5: Local grid computing approach, a “fire-and-forget” scenario

This case is otherwise similar to example 4 presented in Section 2.3.2.4, except that the HTCondor environment is defined for grid computing and the job description file contains additional definitions for the case. The approach requires that the local computer, e.g. a laptop, has its own HTCondor computing pool defined. To enable grid computing with a local pool, the following additional settings need to be defined in the HTCondor configuration files (e.g. in the condor_config.local file of the Submit client (e.g. a submit laptop):

```

CONDOR_GAHP = $(SBIN)/condor_c-gahp
C_GAHP_LOG = /tmp/CGAHPLog.$(USERNAME)
  
```

```
C_GAHP_WORKER_THREAD_LOG = /tmp/CGAHPWorkerLog.$(USERNAME)
C_GAHP_WORKER_THREAD_LOCK = /tmp/CGAHPWorkerLock.$(USERNAME)
SEC_DEFAULT_NEGOTIATION = OPTIONAL
SEC_DEFAULT_AUTHENTICATION_METHODS = CLAIMTOBE
```

Notice that in this case authentication methods `claimtobe` is defined. This authentication method does not do any actual authentication and should only be used for testing or in trusted environments. The Submit client has to have at least the following daemon processes running (for more details about the daemon processes, see the HTCondor Administrator's Manual):

- master;
- collector;
- negotiator; and
- schedd.

The submit client does not need to provide execute features. In the local computational pool, only security settings are required, e.g.:

```
SEC_DEFAULT_NEGOTIATION = OPTIONAL
SEC_DEFAULT_AUTHENTICATION_METHODS = CLAIMTOBE
```

The job description file has to contain definitions for both the submit client internal local computational pool as well as for the local grid computational pool:

```
getenv          = true
universe        = grid
executable      = dakota_case.sh
log             = dakota_case.log
should_transfer_files = yes
when_to_transfer_output = on_exit
transfer_input_files = dakota.in, dakota_case.sh, case_template
transfer_output_files = dakota.dat, dakota.out
grid_resource   = condor \
                 worker-2.cornet.fi \
                 worker-2.cornet.fi

remote_universe = vanilla
+remote_requirements = true
+remote_should_transfer_files = "yes"
+remote_when_to_transfer_output = "on_exit"
+remote_transfer_input_files = "dakota.in, \
                               dakota_case.sh, \
                               case_template"
+remote_transfer_output_files = "dakota.dat, dakota.out"
queue
```

In this case, only the DAKOTA final results files are transferred from execute nodes back to submit client. Notice that the file transfer settings are needed both for the internal local computational pool as well as for the grid computational pool.