Conceptual Design of an ALICE Tier-2 Centre

Integrated into a Multi-Purpose Computing Facility

Dissertation

zur Erlangung des Doktorgrades

der Naturwissenschaften

vorgelegt beim Fachbereich Informatik und Mathematik

der Johann Wolfgang Goethe - Universität

in Frankfurt am Main

von

Mykhaylo Zynovyev

aus

Kiew, Ukraine

Frankfurt (2012)

(D 30)

vom Fachbereich Informatik und Mathematik der

Johann Wolfgang Goethe - Universität als Dissertation angenommen.

Dekan:	Prof. Dr. Tobias Weth
Gutachter:	Prof. Dr. Volker Lindenstruth
	Prof. Dr. Udo Kebschull
Datum der Disputation:	29.06.2012

Abstract

This thesis discusses the issues and challenges associated with the design and operation of a data analysis facility for a high-energy physics experiment at a multi-purpose computing centre. At the spotlight is a Tier-2 centre of the distributed computing model of the ALICE experiment at the Large Hadron Collider at CERN in Geneva, Switzerland. The design steps, examined in the thesis, include analysis and optimization of the I/O access patterns of the user workload, integration of the storage resources, and development of the techniques for effective system administration and operation of the facility in a shared computing environment. A number of I/O access performance issues on multiple levels of the I/O subsystem, introduced by utilization of hard disks for data storage, have been addressed by the means of exhaustive benchmarking and thorough analysis of the I/O of the user applications in the ALICE software framework. Defining the set of requirements to the storage system, describing the potential performance bottlenecks and single points of failure and examining possible ways to avoid them allows one to develop guidelines for selecting the way how to integrate the storage resources. The solution, how to preserve a specific software stack for the experiment in a shared environment, is presented along with its effects on the user workload performance. The proposal for a flexible model to deploy and operate the ALICE Tier-2 infrastructure and applications in a virtual environment through adoption of the cloud computing technology and the 'Infrastructure as Code' concept completes the thesis. Scientific software applications can be efficiently computed in a virtual environment, and there is an urgent need to adapt the infrastructure for effective usage of cloud resources.

Table of Contents

0 Introduction

While there are many things to take care of when designing and operating a data analysis facility, the key point is to ensure that all present resources are fully utilized in an efficient way, without sacrificing system manageability for the sake of performance optimization. A failure to optimize both performance and manageability of the facility can lead to unjustified money expenses, misuse of manpower, and most important, loss of valuable time. To prevent this, a thorough analysis of user workload and requirements, hardware and software capabilities, and administration and operation techniques is needed. Extensive testing and prototyping is needed to guarantee fruitful results. The shared nature of a multi-purpose computing facility and abundance of available remote computing resources puts additional stress on the flexibility and portability characteristics of the infrastructure.

This thesis has several research objectives:

- evaluation and improvement of the I/O (Input/Output) access performance of the ALICE (A Large Ion Collider Experiment) [ALICE] analysis jobs,
- efficient integration of the ALICE Tier-2 storage resources,
- migration of scientific computing applications to a virtual environment,
- efficient management of the scientific computing infrastructure in a cloud.

The main results of the research include the techniques for I/O access optimization of the ALICE analysis jobs, the guidelines for the ALICE storage resources integration, and an application of the ‘Infrastructure as Code’ approach to the deployment and operation of scientific software, which allows to migrate the workload between the computing resources of different administrative domains. The latter technique takes the full advantage of the cloud computing technology, which is essential to keep the infrastructure flexible, manageable, and up-to-date, and comes from the commercial world of Web services. Transference of proven successful system administration techniques from the commercial computing to scientific computing environment is the key feature of the last part of the thesis. The following practical results have been obtained:

- the improvement of the I/O access performance of the ALICE analysis jobs, which has a direct effect on the time spent on analysis and an indirect effect on the provisioning costs;
- the technique to easily deploy and dispose virtual clusters per user on demand, which allows the user to preserve his custom software environment;
- a virtual grid site on the prototype private cloud which processes jobs for the ALICE experiment;
- the radiation simulation data for the “Safety and Radiation Protection” department at FAIR (Facility for Antiproton and Ion Research) [FAIR] on the private cloud testbed and on the community cloud of Frankfurt Cloud Initiative [FCI];
- a testbed for a PROOF analysis cluster of a record scale;
- a testbed for transition to a virtual environment for the HPC (High-Performance Computing) department at GSI Helmholtz Centre for Heavy Ion Research [GSI].

The scientific software applications can be efficiently computed in a virtual environment, and there is an urgent need to adapt the infrastructure for effective usage of cloud resources.

The research has been carried out at the Scientific Computing department of the GSI, situated in Darmstadt, Germany. GSI is home to a large, in many aspects worldwide unique accelerator facility for heavy-ion beams and its main activities include plasma physics, atomic physics, nuclear physics, material research, biophysics, and cancer therapy.

Currently, while preparing and regularly enhancing its computing infrastructure for the FAIR project, GSI operates a high-end multi-purpose computing farm. It is shared by multiple physics working groups and most extensively contributes to the ALICE experiment of the LHC (Large Hadron Collider) project [LHC], as an ALICE Tier-2 centre.

The LHC is the world’s largest and highest-energy particle accelerator operated by CERN, the European Organization for Nuclear Research [CERN], in Geneva, Switzerland. ALICE is one of the four large experiments at the LHC, where particle beams are collided, and is set to study the physics of strongly interacting matter at extreme energy densities. The ALICE collaboration numbers more than 1000 members from 115 institutes from 33 countries. GSI, which is among these institutes, plays a significant role in this collaboration, both in detector construction, and analysis of the experimental data.

The data generated by ALICE describe either real events produced by the experiment or events simulated with Monte Carlo (MC) methods [Kalos]. In both cases, the data are stored in ROOT [Brun] files of various sizes with an internal structure of a tree, which holds various relevant physics information for each event. An event is a snapshot of the particle interactions recorded with the detectors during a beam collision. ROOT is an object-oriented software framework developed at CERN, implemented in C++, and used in a major share of the tasks related to processing the physics data within the High-Energy Physics (HEP) community. Thus, data consist of persistent ROOT objects and may be manipulated exclusively with ROOT based libraries. This is a reason why the I/O issues during data processing with ROOT are at the focus of this thesis.

Petabytes of data are produced by ALICE every year. They are distributed for processing or storage among the computing centres which form together the ALICE grid [ALICE Computing]. There are four tiers in the ALICE grid. The main tasks of the single Tier-0 centre, which is located at CERN, include storage of a master copy of raw data and event reconstruction. The Tier-1 centres keep copies of raw data, share the task of event reconstruction with the Tier-0 centre, and perform scheduled data analysis. The Tier-2 centres provide an infrastructure for running Monte Carlo simulation of p+p (proton) and Pb+Pb (lead ion) collisions and end-user analysis of simulated and real experiment data. These tasks can run as jobs submitted to the grid, local batch jobs, or interactive computing processes. The capacity of a Tier-2 varies from one centre to the other. Currently, the Tier-2 centre in GSI provides the global ALICE community with 5700 HEP-SPEC [Merino] of CPU (Central Processing Unit) power and 440 TB of storage [WLCG]. The resources, which are used by the national ALICE physicists and are not accessible over the grid to the global community, constitute the additional tier, the Tier-3.

Thus, the computing tasks performed at a Tier-2 are mostly of 2 kinds: simulation, and analysis. The simulation tasks are characterized by being CPU bound and having a large memory footprint. Therefore, their performance depends mostly on the CPU and memory capabilities of the computer hardware.

The analysis tasks are calculations performed on the sets of input data, which are the ESD (Event Summary Data) files [Offline Bible]. These ESD files are usually chained together, providing a continuous list of events. The analysis code is designed in the following way: after a data file is opened, and a tree structure of the contents is initialized, each event entry is read and analyzed in turns. Event by event data are read, calculations are carried out, and results are accumulated in a way that, for example, a histogram is filled. After all event entries have been processed, the output AOD (Analysis Object Data) file is generated. In case separate analysis tasks have to be performed on the same set of data, these tasks are chained in what is called an “analysis train” and processed together after each event entry is read, so that data are read only once. This trick helps to shift the analysis performance from being I/O bound in the direction of being CPU bound, although the problem of the data transfer bottleneck still has to be addressed.

The causes leading to an insufficient data transfer rate can be divided into network related and disk related. The network related causes are those which originate in network misconfiguration and end up overloading network interfaces and increasing network latency. The disk related causes may originate from a non-optimal disk configuration and bad access patterns in the system where every I/O request counts.

Basically, the access to data can be organized at a data analysis facility in two ways. Computing resources can retrieve data from storage resources over the network or from direct-attached storage. The former is considered the easier one to set up and maintain, because computing and storage resources are decoupled. Performance of such a system would depend on how well the capabilities of each of the three elements: CPU, network, and storage, match each other. The latter option is generally regarded as the one that could provide the best possible performance, because of at least a principal absence of the network-related issues between CPUs and storage.

Other interesting topics, beyond the scope of the thesis, include the integration of a Tier-2 centre into the global ALICE grid, distributed architecture of its AliEn grid middleware [Saiz], wide area network connectivity of an ALICE Tier-2 centre, and the migration of data within the ALICE tier computing model.

This thesis provides the answers to questions which are of specific interest to those who provide the resources for an ALICE Tier-2 centre, and those who use them for data analysis. But at the same time, some of the topics discussed below, such as cloud computing and infrastructure management, may be relevant also to those who provide any computing infrastructure, especially in a shared environment, and those users, who would like to run their trivial parallel HPC applications in an efficient way on any shared resources available.

1 Data Analysis on Direct-Attached Storage

In order to have an efficient data analysis process, data to be processed must be efficiently read from a storage system, where it must be efficiently stored. Efficiently means that performance should meet users' requirements, preferably be optimal for installed hardware, and not jeopardize system's reliability and availability. The amount of data to store and the rate of requests to process it dictate that currently hard disk storage is the only suitable and affordable storage technology for an ALICE Tier-2 centre.

Two approaches for organizing storage mentioned in the introduction are examined. In the case, when the part of the system serving data does not itself run the calculations, it can be optimized specifically for storage. This implies that the storage system would be comprised of the file servers, designed for fast parallel access by multiple clients. In the case when computing nodes access data from local disks, the hardware should be capable of both performing intensive calculations and sustaining the data transfer.

1.1 Benchmarking Issues

While benchmarking the hard disk storage, one has to keep in mind the following issues. The performance parameters of the CPUs, memory, and all other computer components, which are involved in running a benchmarking test affect the overall performance of the disk storage system [Kozierok]. To be able to compare two sets of hardware the tests have to run multiple times under identical conditions. The results of the tests need to be averaged over multiple runs to provide greater precision. In some cases the worst performance results have to be analyzed and compared to understand the worst-case scenarios. Certainly, the benchmarking tests which are based on user applications are the ones that will show the performance of the system in practical use, but a comparison with the synthetic tests' results is often necessary too [Traeger]. It helps to understand whether the limitations in performance come from the hardware, the application, or the data files.

The amount of free space on the disks can also affect positioning and transfer performance. *Positioning* is the process of setting the hard disk head to the place on the hard disk where the data are to be written or read. *Transfer* is the process of transporting data to or from the hard disk platter after the head is positioned and starting to write or read. When the disks are almost full, the number of positioning operations may increase due to a number of data blocks, and possible data fragmentation. Besides, the transfer performance may suffer because of greater utilization of the inner tracks of the disks with fewer sectors than the outer ones and so sustain a lower transfer rate. So, to ensure fair comparison between different disks, the data for the benchmarking tests has to be put on the empty disks.

For the data analysis facility, where large amounts of data are orders of magnitude more often read than written, read performance is the main target of the benchmarking tests. Each read operation on a disk storage system goes through several stages. First, the data are retrieved from the tracks on the hard disk platters. The part of the data, which has been recently accessed, is retrieved from the cache residing on the disk's logic board. Then, the data are transferred to the RAID (Redundant Array of Inexpensive/Independent Disks) controller, and a part of it is stored in turn in its cache. Finally, the data are communicated via main memory (RAM) to CPU. Main memory can in turn have a considerable page cache configured, which would make another place where the recently accessed data can be stored. During reads, all of these caches pursue the same goal of improving performance by intercepting repeated requests for the same data, and thus saving the time needed to access the hardware on the lower level. This caching can seriously affect the results of repeated benchmarking tests which involve reading same data. While for the first test run data would be read from a disk, for the second one it would be read already from a cache. Additionally, if for example the data set has been copied in the OS (Operating System) to a particular destination right before a test on this data set is started, there is a good chance that the data are in cache. To assure that the data to be read is not stored in cache, one can clear it, or fill it with any other data before each test run.

1.2 A Storage System Based on RAID

Many factors are affecting a storage system's performance, reliability, and data availability. To begin with, there are hardware factors. De facto standard for large-scale hard disk storage is the RAID technology. The idea behind it is to store data on multiple hard disks running in parallel. Various RAID configurations, called 'levels', differ in performance, reliability, storage efficiency, and costs characteristics. Improvement in any of them goes at the expense of all other. So, requirements for all of these characteristics have to be considered in order to choose the most suitable level for a particular storage system. The I/O access patterns of the applications which are going to use RAID storage have to be taken into account too. Reliability in case of a RAID usually implies fault-tolerance, the ability to withstand the loss of at least one disk.

If both performance and reliability are equally important, and there is no place or money for too many redundant disks, one of the most suitable RAID levels is RAID 5. The disks in a RAID 5 are striped, which means they are split into units (stripe units) of a certain size, called *stripe size*, which can be specified during installation. Striping improves performance because the I/O requests for data in stripe units on different disks can be processed in parallel. More disks used in parallel result in a better transfer performance. However, for a single read request spanning data on more than one disk positioning has to be done on each of the disks, although in parallel. For the small read requests the positioning takes the larger part of the total time of the read operation than the transfer. For numerous random small read requests positioning drags overall read performance down significantly. Thus, to optimize performance, stripe size should be defined empirically with regard to the application’s I/O access patterns. Generally, the optimal stripe size should be larger than the system’s average request size, so that the requested data can be found inside a single stripe unit and be processed by one disk [Watts]. The average request size can be calculated from the values of the average requests per second and the bytes per second.

A RAID 5 is tolerant to the loss of one disk. For every write operation to the RAID 5 the controller calculates parity information which is distributed among all participating disks. In the event when one disk is faulty this information can then be used for rebuilding the failed drive while it is either repaired or substituted. Certainly, parity calculation badly affects write performance, but for the system where data are written once, and read multiple times, this drawback is not that significant. The array capacity equals to (Size of the Smallest Disk) * (Number of Disks – 1). For full disk capacity utilization, all disks have to be of the same size. Hot sparing of disks and an automatic rebuild allow good data availability. The Dual parity of a RAID 6 gives time to rebuild the array without the data being at risk if a single additional drive fails before the rebuild is complete.

In the case when performance is much more important than storage reliability a RAID 0 and a variation of the JBOD (“Just a Bunch Of Disks”) configuration may be considered. When a RAID 0 is used, a file is striped across the disks, but no parity information is calculated and recorded. The

I/O performance is at its best, but the failure of one disk results in the loss of data in the whole array. The JBOD configuration is usually understood as just a concatenation of the hard disks into a single array without any striping or replication mechanisms. A variation of JBOD is when each hard disk is mounted in an OS independently. In this case it is the responsibility of the software to manage the I/O requests that way, as to ensure parallel utilization of the available disks. The benefit of the improved performance only comes when disks are accessed simultaneously. Data availability is slightly better for such JBOD configuration than it is for a RAID 0 because with a loss of one disk the data on remaining disks can still be accessed.

Another decision that has to be taken in addition to choosing a RAID level is where to store the OS. One possibility is to store it on the same array as the data. In this case the I/O requests by the OS or service applications for various modules or libraries to load can interfere with the I/O requests for the data for processing. To separate these I/O requests the OS can be installed either on a single disk which is not a part of the RAID, or on another RAID to ensure fault-tolerance. To install an OS on a RAID 1 requires a minimum of two disks, with one of them serving as a mirror. Obviously this separation comes at the expense of the storage efficiency, since disks would be exclusively used for the OS, and would not store user data.

In addition to specifying the stripe size for a RAID there are other possibilities to tune the I/O subsystem (Figure 1.1) [Ciliendo] [Domingo]. The *read-ahead* property determines how many bytes adjacent to the ones requested by the application the RAID controller will transfer into its cache. In the case of the sequential reads increasing the read-ahead value improves the read performance. However, for random read requests, this does not help because a cache hit is unlikely. Whether read-ahead is switched off automatically in case of a non-sequential access has to be clarified. The *queue depth* property indicates the maximum number of the outstanding I/O requests a controller can schedule for processing. The outstanding requests in the queue can be reorganized according to which disk is responding, and combined before being issued. This improves throughput but also increases latency. The read-ahead and queue depth properties can be specified on different layers of the I/O subsystem.

The way the I/O requests are scheduled depends on the type of scheduler used by the operating system. Linux supports CFQ (Complete Fair Queuing), NOOP (No Operation Performed), Anticipatory, and Deadline algorithms. One can tune aforementioned properties on the level of the OS kernel.

Furthermore, one can tune the file system [Jones] parameters. *Block size* is the minimal number of bytes the file system can write or read from a disk for a single request. Theoretically, it can be expected that a system handling many small requests gains performance from a smaller block size, a system handling larger requests – from a larger block size. The maximum file system block size is the page size of the kernel. Default value for the block size in Linux is 4096 bytes. Block and stripe sizes should not be confused.

The file systems like ext3 and XFS can be aligned in accordance to the RAID stripe size setting. For ext3 the needed parameter is *stride*, for XFS – *sunit* and *swidth*.

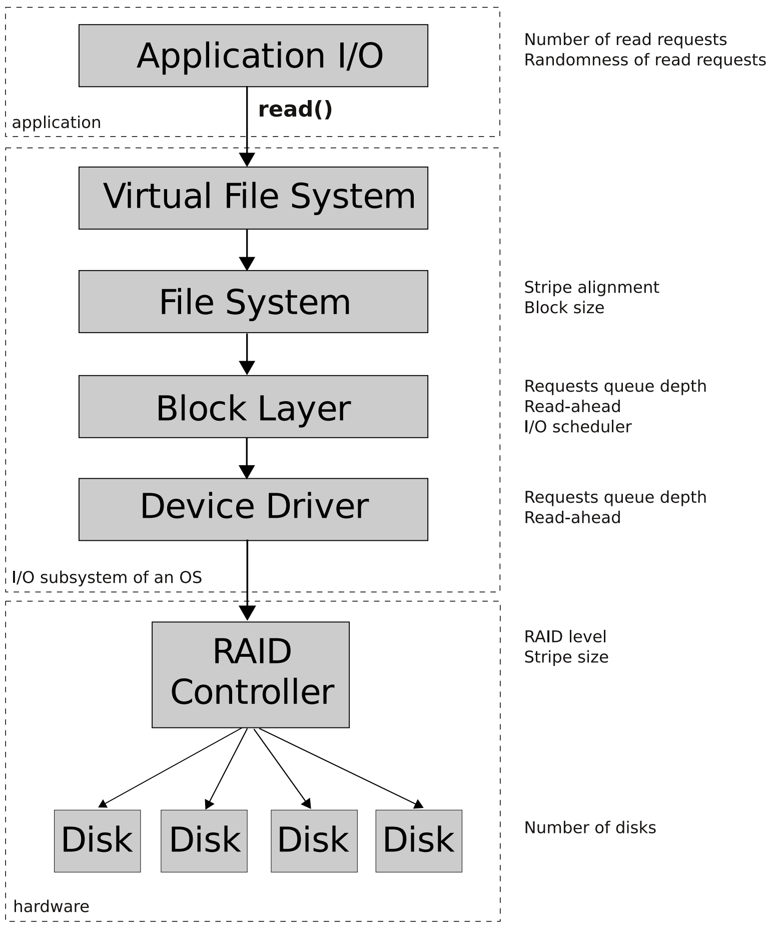

Figure 1.1 There is a multitude of characteristics on each layer of the I/O subsystem which might affect the performance.

1.3 I/O Access of the ALICE Analysis Tasks

The overall performance of the storage system is greatly influenced by the I/O access patterns. Sequential I/O access of the large data files performs well with one setting of system’s configuration parameters, random I/O access of small data blocks with another. Therefore, in order to configure the system properly the I/O access patterns of the user applications have to be examined [Wang]. Random I/O access will in any way lose to sequential I/O access in performance due to additional positioning time involved. So, it should be worth considering to modify the I/O access patterns where possible to avoid or reduce random access.

1.3.1 The ALICE Data Files

The analysis tasks run by the physicists of the ALICE experiment in GSI are using AliRoot, an ALICE-specific framework based on ROOT, for processing their ESD files [Offline Bible]. Those files are created after raw data, either recorded by the experiment, or simulated, is passed through the stage of event reconstruction.

An average data set contains around 100 000 of the ESD files, each of them holding information about 100 events and being in a span from 2 to 8 MB in size. An ESD file contains a list of events in the format of a ROOT tree. The logical format of a ROOT tree is made up of *branches* [ROOT]. Each branch constitutes a homogeneous list of objects describing a particular property of each event. A branch can be split into sub-branches for each data member in the object. A typical ESD tree for the ALICE analysis tasks is comprised on average of more than 400 branches.

The nature of the analysis tasks implies that all events in an ESD file and all the files in a data set need to be processed. Therefore, branches are processed entirely, the only thing that can vary for different tasks is the number of branches processed, since not every analysis task requires all information regarding the event. In order to access a particular property of every event, an application does not have to read all event information and can read only the corresponding branch. Splitting the branch increases granularity of information, which helps to pin-point the particular event properties. The user himself is able to activate and deactivate specific branches for reading.

On disk a tree is laid out as a list of *baskets*. Basket is a memory buffer that serves as a unit of the I/O requests for a ROOT tree. It is filled with the objects of a single particular tree branch. In case of the ALICE ESD files a basket holds the objects that constitute event property information from a specific branch.

The number of baskets used to represent a particular branch on disk depends on the chosen basket size and the size of all the objects in the branch. Although the event information objects within a particular branch are homogeneous, for each event they can substantially differ in size. This leads to the fact that the same-sized baskets can store a varied number of the events' objects. The default basket size is 32000 bytes. If the basket size is not large enough to hold objects of one event, the tree will expand the basket to be able to hold that one event. It is done to avoid excess memory allocation operations necessary when one event is spread over several baskets. The baskets are filled until one of the following three situations happens:

- there is not enough space (predictable in case of the objects of a fixed size),
- the space remaining is less than the size of the last event filled,
- the last event has overfilled the requested basket size.

Baskets then can be compressed according to the chosen compression level before being written to the disk. It has to be mentioned that for a tree in a particular file some branches, and thus their baskets, can be stored in a different file making use of the *Friends* feature of the tree [ROOT user's guide].

1.4 Analysis Tasks and Trains

The tasks which are used to analyze ESD files, are based on the AliRoot Analysis framework [Offline Bible]. Written in C++ with the use of the AliRoot and ROOT libraries a task consists of at least the following 5 functional blocks:

- Basic constructor. In this block the types of the input and of the output of the task are specified (e.g. data files, and histogram respectively). It is invoked once at the start of the analysis process.
- ConnectInputData. The input objects (e.g. a tree, event object) are initialized in this function. Invoked once per analysis process.
- CreateOutputObjects. The output objects that will be written in the file (e.g. histogram) are created in this function. Invoked once per analysis process.

- Exec. The analysis code which is run for every event is implemented here. Invoked once per processed event.
- Terminate. This function is invoked at the end of the analysis process and is used for processing the output (e.g. draw histograms).

The input of an analysis task, usually, is a *chain* of ESD files. A chain is a list of ROOT files containing the same tree. In order to identify the branch structure of the tree in the chain, a special event handler is invoked, which sets the correct branch addresses. It can be used to deactivate certain branches of the tree, so that unnecessary branches are not read. For some analysis tasks there are several additional files which have to be accessed.

The tasks developed to run on the same data set can be combined into a single process of an analysis *train*. The AliAnalysisManager object collects the tasks, defines the input containers, and ensures execution of each task. In this case, for every event, an Exec block of each task controlled by the manager is executed sequentially before the next event can be processed. Thus, instead of multiple tasks issuing duplicate read requests for the same data separately, the manager of the analysis train issues a single request servicing all the tasks.

Users, the ALICE physicists, are free to fill each block of the task with their code. The train does not protect from any superfluous actions of the user code inside the tasks.

1.5 Evaluating Read Performance on the Application Level

To understand the layout of the trees on a disk, and to be able to analyze the factors affecting analysis read performance, a ROOT code has been developed. It produces a list of baskets of every branch in a tree in the order of their appearance on the disk with the following information:

- their size before and after compression,
- the number of events stored,
- compression factor,
- the name of the branch they belong to.

This information (Table 1.1) helps to understand which parts of a ROOT file a process is targeting for each analysis scenario. It may be useful in order to understand whether seeking on disk is imminent. Particularly, it leads to an assumption that data read because of a read-ahead operation is unlikely to be requested for analysis, where only few of a total number of branches are required, and baskets of those branches are non-adjacent on disk. However, when a train of tasks is reading all event information from every branch, and thus the whole file is needed, increasing read-ahead should improve the read performance. In the same case, it also makes sense to prefetch the data file with a tree into memory before the tree is accessed by an analysis process. Thus, multiple requests for branch baskets will be issued to the RAM rather than to the disk, which should result in a speed-up. Access to disk, in case of prefetching, will be sequential, in contrast to access from ROOT, which is unlikely to be sequential for the ALICE ESD files, where object data for a particular event is scattered across the file.

Address	Size (B) on disk	Compression factor	Branch name	Basket Number / Total	Events	User content size (B)	Basket size (B) in memory
917428	21200	1.37	Tracks.fIp	#0 / 7	16	28866	29018
1725394	21555	1.37	Tracks.fIp	#1 / 7	11	29312	29444
2736920	23958	1.37	Tracks.fIp	#2 / 7	15	32566	32714
3245382	19589	1.36	Tracks.fIp	#3 / 7	6	26612	26724
4074341	20799	1.37	Tracks.fIp	#4 / 7	12	28306	28442
5424231	28846	1.37	Tracks.fIp	#5 / 7	19	39400	39564
6803933	20922	1.37	Tracks.fIp	#6 / 7	21	28578	28750
220	2624	187.36	AliESDFMD.fMultiplicity.fData	#0 / 100	1	491521	491632
1318938	4596	106.97	AliESDFMD.fMultiplicity.fData	#20 / 100	1	491521	491632
6799579	3285	149.66	AliESDFMD.fMultiplicity.fData	#99 / 100	1	491521	491632
52366	497	49.67	AliESDFMD.fEta.fData	#0 / 50	2	24578	24684
2735995	925	26.69	AliESDFMD.fEta.fData	#20 / 50	2	24578	24684
6802864	1069	23.09	AliESDFMD.fEta.fData	#49 / 50	2	24578	24684
3602099	3339	9.35	AliESDRun.fTriggerClasses	#0 / 2	53	30899	31214
6939656	3000	9.23	AliESDRun.fTriggerClasses	#1 / 2	47	27401	27692
6938090	120	4.14	AliESDRun.TObject.fUniqueID	#0 / 1	100	400	497

Table 1.1 An example of the heterogeneous nature of branches and their baskets intertwined in a file on a disk for a ROOT tree with 100 events. The branches have different numbers of baskets, different sizes of baskets, different compression factors.

To find out a pattern of the I/O requests of an analysis task, an AliRoot code has been developed, which produces statistics on how many read requests (calls) have been made per event, and how many bytes have been read as a result of these requests. In ROOT, the number of read calls is the number of the times the POSIX (Portable Operating System Interface for Unix) function read() has been called. This number defines how many requests have been issued by an application to the I/O subsystem of the OS. A distribution of the sizes of read requests for an exemplary AliESDs.root with 675 baskets is shown at the Figure 1.2. Almost 50% of read requests sizes are smaller than 512 bytes in size.

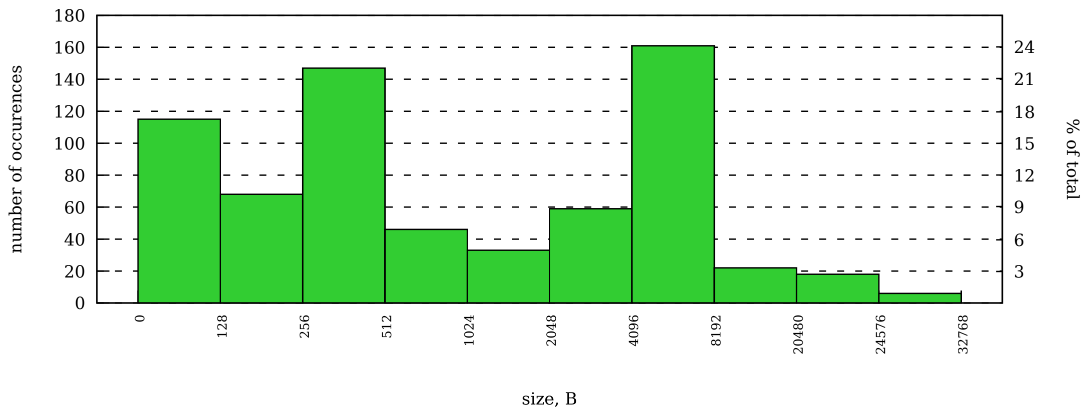

Figure 1.2 A histogram of the sizes of read requests for an exemplary AliESDs.root with 675 baskets. Almost 50% of the total are less than 0.5 KB in size. 24% are between 4 and 8 KB in size.

As seen in the Table 1.1, basket properties differ substantially from branch to branch of a typical ALICE ESD file. The size of a compressed basket on disk varies from 100 bytes up to 32000 bytes in an exemplary file, being on average around 3 KB. This, in turn, defines the size of a corresponding read request.

The way how well different user objects inside a basket can be compressed affects the uniformity of baskets' size on disk for compressed ESD files. When no compression is used the size of baskets on disk may be more homogeneous. However, the more uniform access comes at the price of inflated storage requirements needed for keeping uncompressed data. It is unfeasible to fill the baskets with data until the compressed size would reach a needed threshold, because substantial computing resources are required to check the size of the compressed basket for every user object to be added.

Let us examine the access pattern of an analysis task processing a typical ESD file of around 2 MB. Before every ESD file in a chain is processed 4 read calls are issued to the file for initialization. An event tree inside the file is comprised of 499 branches, with 484 of those resident in the file and stored in 675 baskets. Among those 15 missing from the file, 6 are in another “friend” file and 9 are in memory. Figure 1.3a shows that all 484 first baskets of each activated branch have been requested from the disk to read the first event. The baskets that are holding only one event have been requested for every single event. The baskets that are holding two events have been requested every second event.

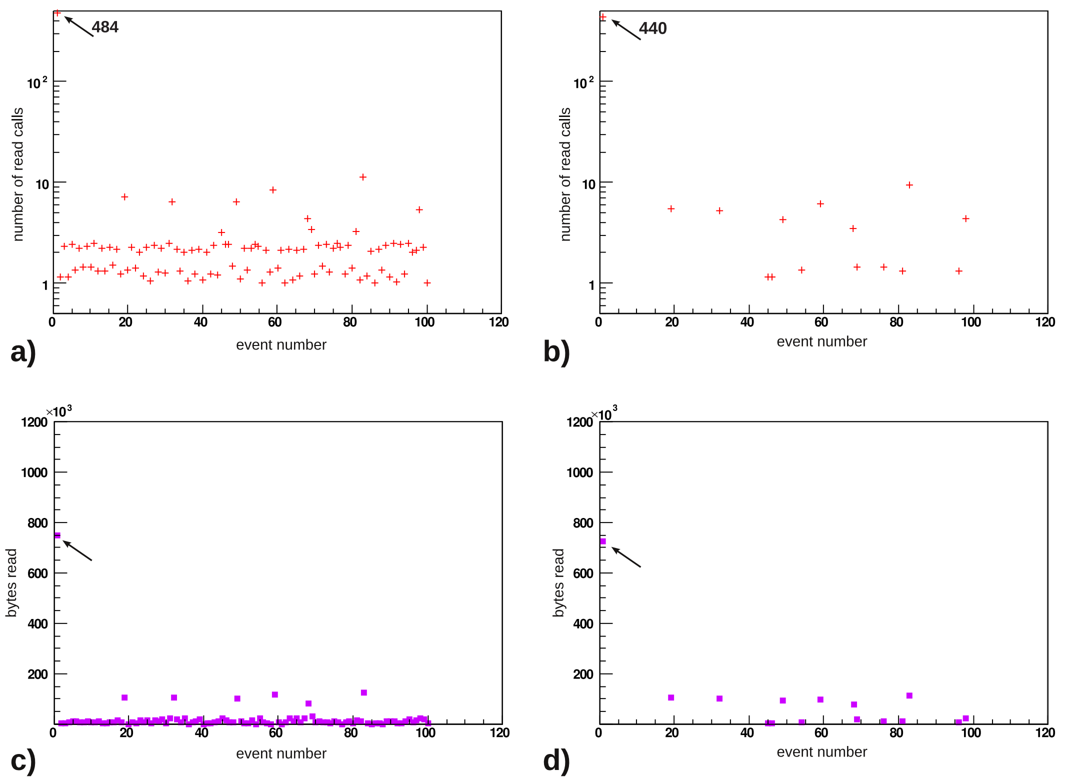

Figure 1.3 An analysis task processing 100 events in an AliESDs.root file. Read calls and bytes read for each processed event. a) Read calls, all branches activated. 484 requests for the first event. b) Read calls, 44 branches deactivated. Deactivation of particular branches significantly alters the requests pattern. c) Bytes read, all branches activated. d) Bytes read, 44 branches deactivated. Most of the data per file is read for the first event.

The majority of the branches, being stored in only several baskets each, need an equally low number of requests to be read completely. Baskets of those branches are requested at larger intervals. Requests for these baskets are visible on the Figure 1.3b, where the branches with 50 baskets, holding 2 events each, and 100 baskets, holding 1 event each, are deactivated and not read. That is a real example of how ALICE physicists are running their analysis.

Figures 1.3c and 1.3d show the amount of bytes read in each of the two scenarios, with and without disabled branches. For an analysis scenario, where all branches are read and requests are issued only to an AliESDs.root file (Figure 1.3a), the number of read requests equals the number of baskets.

File Name	File Size (B)	Single Task		Train	
		Read Calls	Bytes Read	Read Calls	Bytes Read
AliESDs.root	2 866 469	483	1 377 525	483	1 377 525
galice.root	6 058 811	101	683 172	101	683 172
Kinematics.root	6 150 493	403	320 596	7708	7 407 476
TrackRefs.root	5 728 700	403	199 110	2069	6 722 433

Table 1.2 Statistics for requested files per single AliESDs.root file processed with an analysis task and with an analysis train. Requests to the ESD file are greatly outnumbered by the requests to Kinematics.root and TrackRefs.root. The fact, that bytes read values for these files are larger than the respective file sizes, shows that some data are read more than once.

For the analysis train where additional Monte Carlo information is queried for each event from three other files the average request size is less than 2000 bytes. In addition to calls to AliESDs.root seen on Figure 1.3 there are numerous requests to galice.root, Kinematics.root, and TrackRefs.root (Figure 1.4(a,b), Table 1.2). The exact numbers also depend on the tasks themselves. Unlike for the ESD data, the same MC data can be read more than once during the analysis runtime. Table 1.2 shows that the number of bytes read from Kinematics.root during the train analysis is larger than the number of bytes that make up the file. One has to note that these numbers of bytes correspond to compressed files.

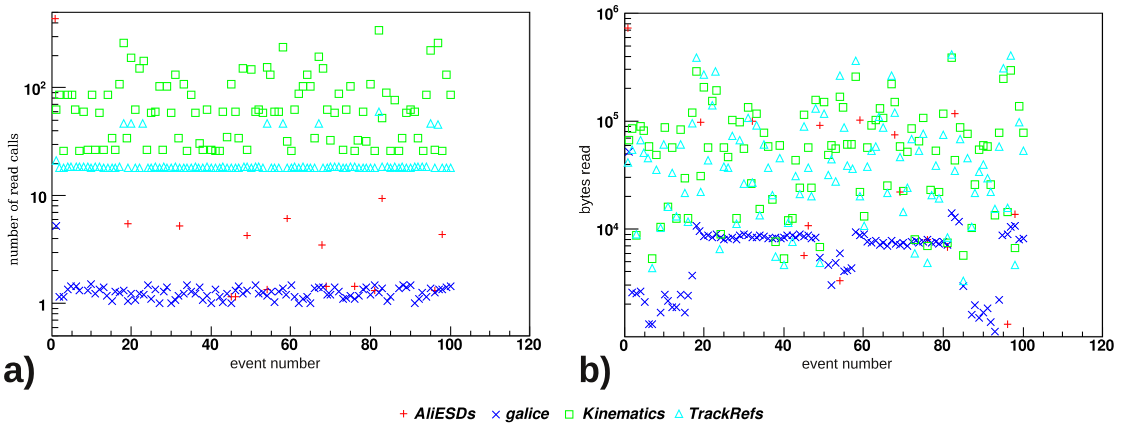

Figure 1.4 An analysis train processing 100 events in an AliESDs.root file. Auxiliary "MC" files are queried. a) Read calls. b) Bytes read. The access pattern is dominated by the read calls to the "MC" files, and not the ESD. Fluctuations may be attributed to varying event properties.

1.6 Evaluating Read Performance on the Level of the I/O Subsystem

The number of how many read requests have been issued eventually to the block device as well as many other relevant disk activity parameters can be monitored using the kernel file /proc/diskstats [iostat] and the iostat tool.

The following parameters of disk activity are shown among others in /proc/diskstats, all the values are accumulated since machine boot:

- the number of read requests completed,
- the number of read requests merged,
- the number of sectors read from a device,
- the number of milliseconds spent doing I/O.

Iostat provides the following I/O statistics on a per physical device basis:

- the average requests queue length,
- the average request size in sectors,
- the average time for I/O requests issued to the device to be served (this includes the time spent by the requests in queue and the time spent servicing them),
- the number of megabytes read from the device per second,
- the number of read requests that have been issued to the device per second,
- the number of read requests merged per second that have been queued to the device.

The average request size to a block device also depends on how well the requests from an application can be merged together. Requests to adjacent sectors on disk which get into the outstanding I/O requests queue are combined.

Table 1.3 shows the correspondence of the number and the size of the requests from the application via the read() calls to the number and the size of requests to the block device. One may observe that the average size of ROOT read requests for the analysis with the MC queries is 1.9 smaller than for the analysis without the MC queries. For the requests to the block device the situation is opposite with the ratio of 2.3. This means that the read() calls for the tiny MC data queries are merged into the larger queries to the block device even better than the read() calls for the ESD data. It would also explain why a train (with more MC queries) has a larger average block device request size than a task (with less MC queries).

	Task		Train (15 tasks)		Train (23 tasks)	
	Size (KB)	Number	Size (KB)	Number	Size (KB)	Number
ROOT read() calls						
No MC queries	3	49 105	3	49 127	-	-
MC queries	1.9	139 584	1.6	1 074 726	1.6	1 074 636
Block device requests						
No MC queries	74.5	3 608	74.5	3 607	-	-
MC queries	24.3	26 077	172	10 005	171.6	10 044

Table 1.3 Average request sizes and numbers of requests inside the application for tasks and trains, and the corresponding parameters for requests issued to a block device. Fewer, and of a larger size, requests are processed by a block device than are issued by the application. 100 files, each with 100 events, are analyzed. Eight tasks of the train cannot be run without "MC" queries, hence no values.

The missing statistics for a train with 23 tasks and no MC queries come from the fact, that only 15 out of 23 tasks of the current train can be executed without MC data, and the 8 remaining tasks crash the train when MC data are not available.

It is also worth to mention that tracing tools of varied complexity are developed to monitor what exactly an application is doing [Roselli]. Strace [strace] is capable of showing all system calls generated by an application. Dtrace [Cantrill], SystemTap [Prasad], and LTTng [Desnoyers] aim to provide dynamic binary instrumentation to trace any part of the system either in the kernel or the user space and process the tracing results dynamically while the application continues to run. By inserting markers inside the application code it is possible to enable logging for the application events which are triggered during the application's run.

1.7 Optimizing Read Performance

1.7.1 Test Environment

The tests are carried out on a 8 core Intel Xeon E5430 2.66 GHz with 8 GB of RAM. HP Smart Array P400 is used as a RAID controller. The Debian 4.0 OS [Debian] is stored on the array A, which is a RAID 1+0 (2 disks). The analysis data are stored on an aligned XFS file system installed on the array B, which is a RAID 5 of 932 GB (5 disks x 250 GB + 1 x 250 GB spare disk) with 64 KB stripe size and the block device read-ahead of 4 MB. The disks, HP GJ0250EAGSQ, have the speed of 5400 rpm. The array B can sustain sequential read operation at around 90 MB/s, as measured by hdparm [hdparm].

The jobs are the ALICE analysis train and task processes. To ensure for each job in the tests that all data files are read from the disks, the memory page cache is flushed between the test runs. Every job processes its own data set of 1000 files and utilizes a single core. A file contains 100 events, the size of a file varies between 2.3 and 8 MB. Events describe p+p collisions at 10 TeV.

The ALICE analysis train, used in the measurements and by the ALICE physicists at the time of research conduction, consists of 23 user-written tasks.

The used ROOT version is 5.23-02, and the AliRoot version is v4-16-Rev-08. Unless noted otherwise.

1.7.2 TTreeCache

To optimize the processing performance of the ROOT trees across wide area networks, the ROOT developers at CERN have introduced an ad hoc algorithm. Its goal is to fetch branch baskets into the memory before the analysis code requests them [Franco]. It became apparent that disk access performance may benefit from this caching mechanism too. The algorithm is implemented with a TTreeCache class in ROOT, and thus commonly referred to by that name. There are two ways to make the TTreeCache class know baskets of which branch to prefetch:

- to start the Learning Phase for a specified number of entries in a tree during which information on what branches are accessed by the analysis code is gathered,

- explicitly specify which branches should be read.

Then, the TTreeCache class, aware of all the baskets needed from the tree in the file, will sort and merge the disk access requests. The baskets are requested in the order of their appearance on the disk and those adjacent on the disk are fetched by a single request.

For example, let us examine how a chain of 3 files, each storing 100 entries, and holding a tree made of 440 branches, is processed with and without TTreeCache in use. The first file contains 483 baskets, second – 508, and third – 469. Without TTreeCache in use 483+508+469=**1460** requests are issued to read all the baskets of these files.

The following happens when TTreeCache is used. If all the branches are required for every entry, the learning phase can be set to last only for the first entry. It is clear that to process the first entry 440 requests are issued to read the first baskets of each branch. Then, the learning phase is stopped, and TTreeCache knows of all the branches in use. Next, remaining 43 baskets of the first file are prefetched in 18 requests. After the first file's 100 entries are processed 508 baskets of the second file are prefetched in 22 requests. The 469 baskets of the third file are prefetched after 200 entries by 14 requests. In total, with TTreeCache in use, 440+18+22+14=494 requests are issued instead of 1460. With every next file in the chain the difference grows (Table 1.4).

	TTreeCache enabled	TTreeCache disabled
Read Calls to AliESDs.root	2 708	49 006
Bytes Read from AliESDs.root	152 413 985	152 413 985

Table 1.4 Statistics for an analysis task processing a chain of 100 AliESDs files, each with 100 events. TTreeCache significantly reduces the number of read calls.

1.7.3 Prefetching

If all of the branches of an ESD tree are activated and all the baskets are going to be read, then the whole ESD file will be read from the RAID and put into the RAM. The file can be placed into the OS page cache in the RAM after a single read call. The data files are of such a size that they can easily be stored in the RAM in full. This allows to substitute the read requests to the disk with the read requests to the memory by invoking a function that reads the file sequentially with larger blocks into the memory before the file is accessed by the analysis process. Even if not all of the branches in a tree are required it may still be worthwhile. A speed-up due to the sequential read operation by a single read request can outweigh the side effect of reading the whole file.

A C++ code which implements reading of a file into the memory buffer is placed into the Notify() function of the AliRoot Analysis Framework for a test. It is invoked before a new file with a tree is opened. The file is read into the memory buffer with a single read request, and thus is placed into the page cache of the OS. The buffer is then freed and analysis code accesses the file from the page cache. This approach of prefetching the file results in redundant memory allocations, and the way to reduce them has to be examined.

There are plenty of read requests issued per one AliESDs.root data file to galice.root, and especially Kinematics.root and TrackRefs.root for an analysis which needs Monte Carlo data (Table 1.2). Indeed, read requests to these files greatly outnumber the read requests to AliESDs.root. Therefore, optimization of the access to these files will affect the overall data analysis speed the most. The proposed prefetching mechanism can be applied to these files too.

The test results (Figure 1.5) show that prefetching and TTreeCache optimization techniques both increase the throughput rate for parallel jobs which access exclusively AliESDs.root. However, since queries to auxiliary MC data files dominate the I/O, optimizing access to AliESDs.root is not enough to improve the data throughput rate for jobs, where auxiliary data are queried (Figure 1.6). TTreeCache is an optimization technique implemented for accessing the ROOT trees only.

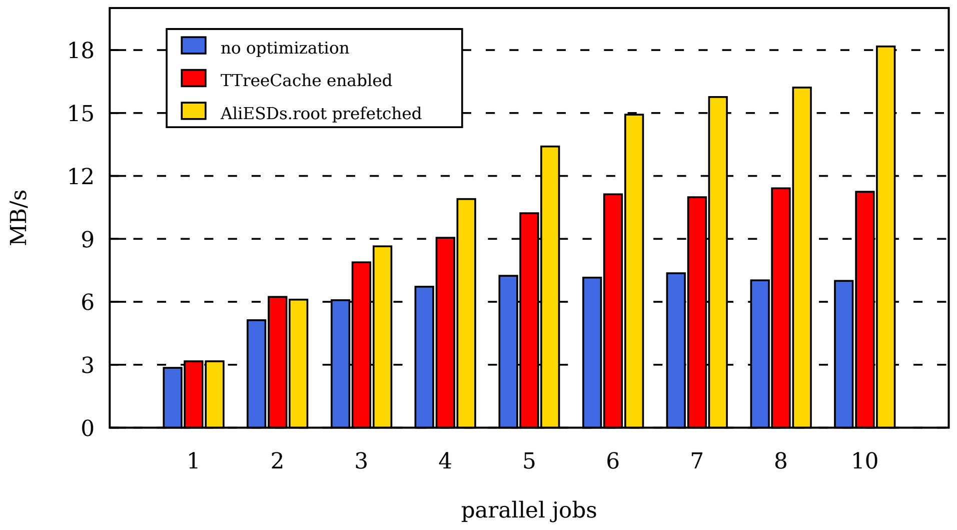

Figure 1.5 A comparison of the aggregate throughput rates for parallel jobs running a single analysis task. Only AliESDs.root is accessed. Both optimization techniques improve the performance, by factors of ~1,5 for TTreeCache and ~2 for prefetching, for eight parallel jobs.

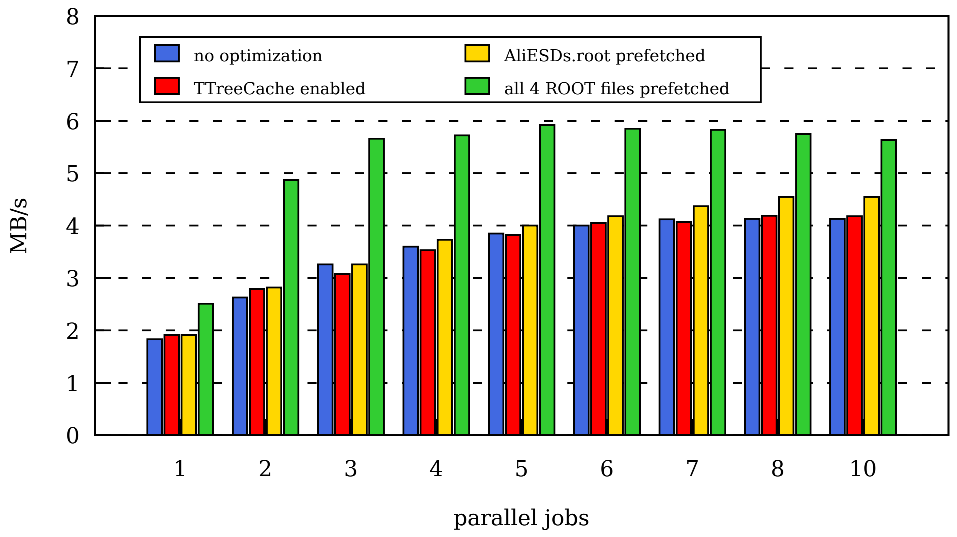

Figure 1.6 A comparison of the aggregate throughput rates for parallel jobs running a single analysis task querying the Monte Carlo data. AliESDs.root, Kinematics.root, galice.root, TrackRefs.root are read. To produce an impact, optimization must concern the Kinematics and Trackrefs files.

All described optimization techniques have no effect on the train jobs (Figure 1.7). This fact suggests that the analysis train of 23 tasks must be CPU bound, i.e. the time for a computer to complete an analysis train job is determined principally by the speed of the CPU. This is proven by the CPU utilization measurement. For analysis train of 23 tasks, CPU has a load of almost 100%, at 98% (Figure 1.8b). This allows the performance to stay on the same level with the addition of about 5 parallel jobs (Figure 1.8a). It goes down when the number of parallel jobs with concurrent disk access approaches and exceeds the number of cores, when the additional system overhead for context switching is introduced.

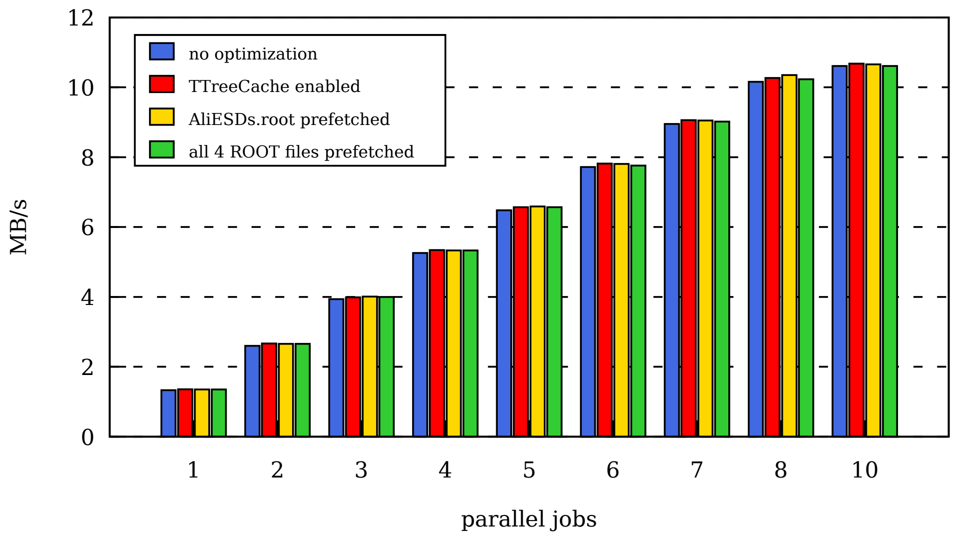

Figure 1.7 A comparison of the aggregate throughput rates for parallel jobs running an analysis train querying the Monte Carlo data. AliESDs.root, Kinematics.root, galice.root, TrackRefs.root are read. Optimization of the read access does not affect the performance of a CPU bound train.

On the other hand, the analysis task is I/O bound, i.e. the time for a computer to complete an analysis task job is determined principally by the period of time spent waiting for the I/O operations to be completed. With the addition of parallel jobs accessing the same block device, performance of a single job significantly decreases (Figure 1.8a).

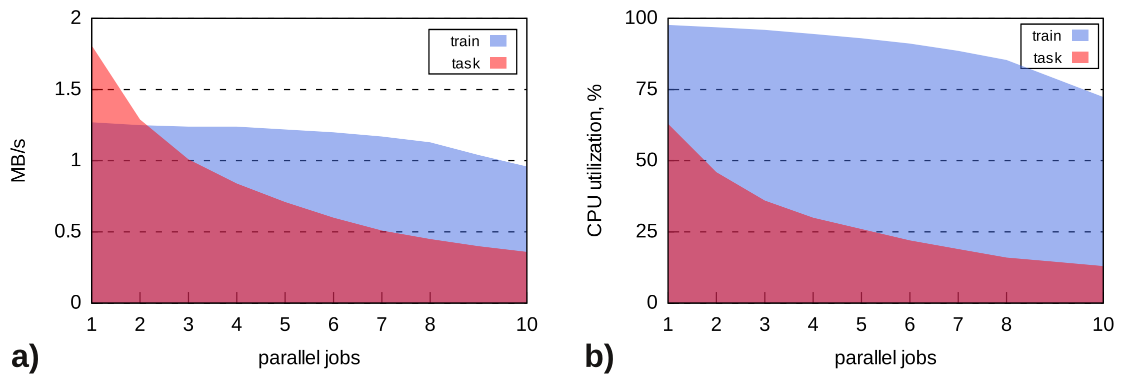

Figure 1.8 a) Variation of the data throughput rate of a single analysis train job and a single task job with the addition of parallel jobs on 8 cores. b) Variation of the CPU utilization of a single analysis train job and a single task job with the addition of parallel jobs on 8 cores. The tests prove analysis tasks to be I/O bound, and analysis trains - CPU bound.

1.7.4 Merging the Data Files

Another way of optimizing I/O access is to merge the data files. As a consequence, the number of data files is reduced, and their size is increased. In turn it cuts down the time spent on initialization of the files, and potentially larger blocks can be sequentially read from the disk. There are 2 options how the merging of the ROOT files can be done: default and ‘fast’. With the default option, source files’ baskets are unzipped, the data objects are merged into new objects which are put into new baskets written to the new file. For the ‘fast’ option, the source baskets are copied to the new file without being unzipped. The ‘fast’ option supports 3 sorting orders of the baskets in the file:

- by offset (default),
- by branch,
- by entry.

Sorting by offset means that the new baskets will be written to the new file in the same order as they are placed in the source file. In the case when sorting order ‘by branch’ is specified, all baskets of a single branch are placed in the file together, one after another. This order particularly suits the analysis scenario where only some of the tree’s branches in a file are read. It allows the baskets of a single branch to be read without additional positioning on the disk being done. When baskets are sorted by entry they are written to a file exactly in the order they are going to be retrieved from the disk. Similarly, no excess positioning is needed for the analysis where all branches are read from the file.

Using less files for the same number of events improves performance for the parallel jobs by a factor of 3 or higher (Figure 1.9). The fact, that default merging, via unzipping procedure, results in a lower analysis performance than the fast one, may be attributed to compression issues in the software and needs further investigation.

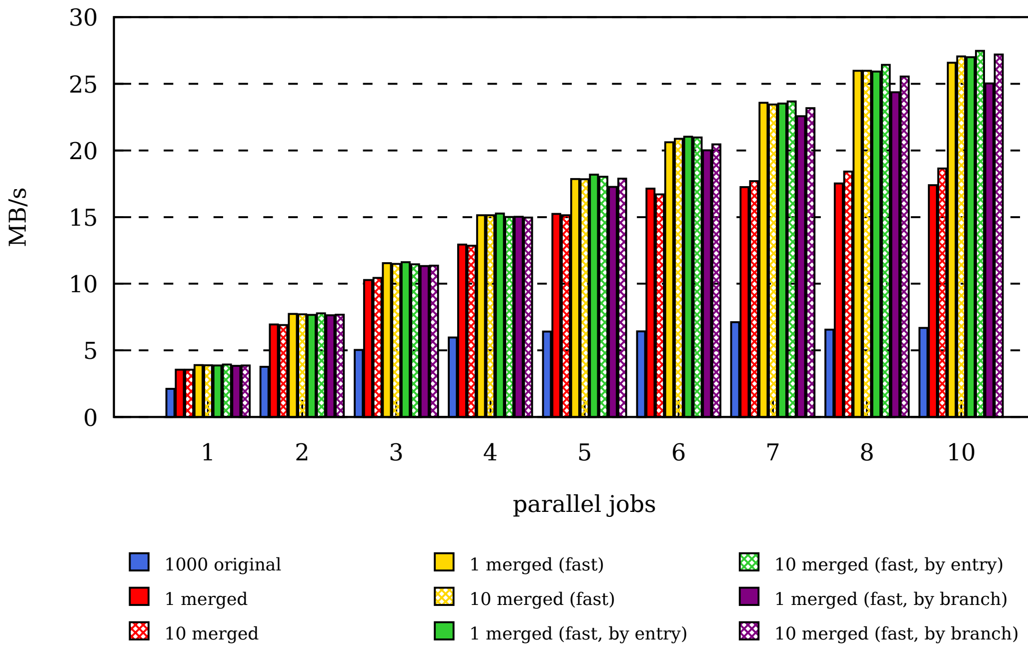

Figure 1.9 A comparison of the aggregate throughput rates for an analysis of the merged and original files. Merging of the data files is an efficient optimization technique. The analysis train of 15 tasks, with only AliESDs.root being accessed. One job processes 100 000 events.

1.7.5 Tuning of the I/O Subsystem

There is hardly a method other than empirical testing using a typical predicted workload to find the values for tuning parameters of the I/O subsystem that yield better performance. As it should be, the default values from the vendors usually produce the best results for most of the workloads.

At the level of the RAID controller, no difference in performance has been noticed for the various values of the stripe size, and hardly any difference between the RAID 5 of 5 disks and the RAID 6 of 6 disks.

At the block level, an important setting is the read-ahead. Read-ahead has to be definitely switched on as it gives a substantial performance increase (Figure 1.10), although there is a slight variation in performance for different values in the range from 64 KB to 16 MB. The size of the I/O requests queue is optimal at the default value of 128.

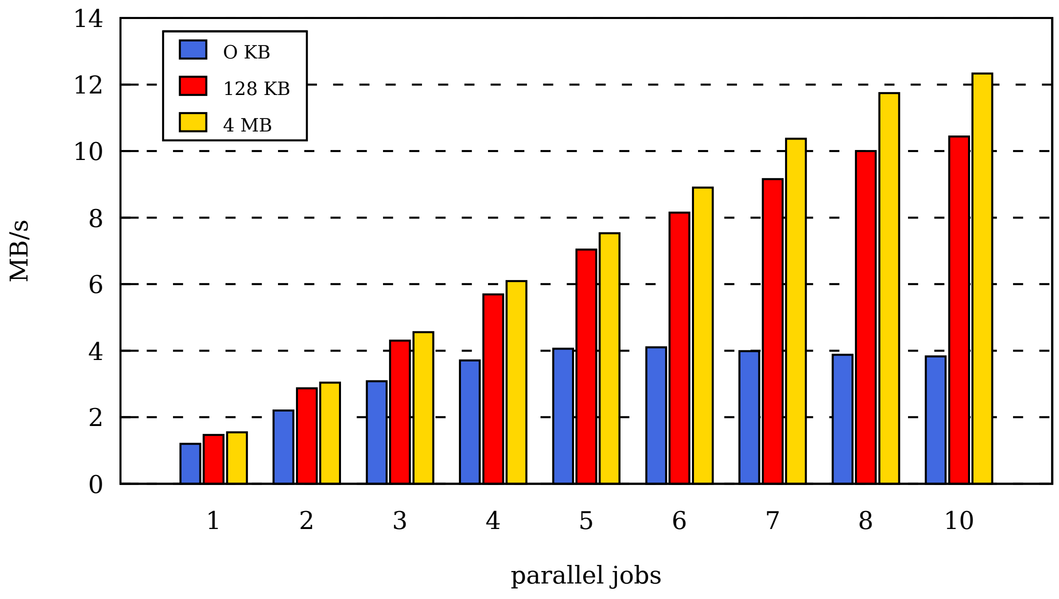

Figure 1.10 The data throughput rate at different values of the block device read-ahead. The analysis train with 15 tasks, with only AliESDs.root being accessed. Read-ahead substantially increases analysis performance for parallel jobs.

Whether the XFS file system has been aligned to the stripe sizes or not has not affected the performance. The number of the allocation groups did not have any large influence either.

1.7.6 Migrating to the Solid-State Drives

A solid-state drive (SSD) is one of the recent additions to the storage hardware technology. Being based on the flash memory and hence lacking any moving mechanical parts an SSD wastes no time on positioning and provides persistent storage with fast random access. An SSD can handle a much higher number of I/O operations per second than a hard disk drive (HDD). The weak points of the SSDs in comparison to the HDDs are lower capacity and higher cost per gigabyte. Whether these weak points would be overcome in the near future is debated [Hetzler].

In general, there are two ways to make use of the SSDs in the large data storage facilities. Either completely substitute HDDs with SSDs in current storage configurations, or to set up an intermediate tier of the SSDs between the RAM and the HDDs. While a complete migration to the SSDs would hardly be cost-effective at the present moment in the high-capacity storage facilities, there are some workloads which could benefit from an SSD tier in between the RAM and the disk storage [Narayanan]. In the latter case, an SSD could be used for a write-ahead log and a cache for read operations. Accessing data from the OS page cache in RAM is still much faster than accessing it from the SSD cache, but RAM is not persistent storage and costs significantly more per GB. So, for workloads requiring fast random data access an SSD could hold data which is not fitting into the OS page cache. A proper cache policy which results in enough cache hits is needed for making the SSD tier worthwhile.

While aforementioned issues slow the popularization of the SSDs among the high-capacity data centres down, the SSDs are quickly cutting into HDD share of the desktop PC (Personal Computer) and laptop markets where capacity requirements are orders of magnitude lower. For running the I/O bound ALICE analysis tasks on laptops or PCs migration to the SSDs proves to be worthwhile, substantially increasing the data throughput rate (Figure 1.11).

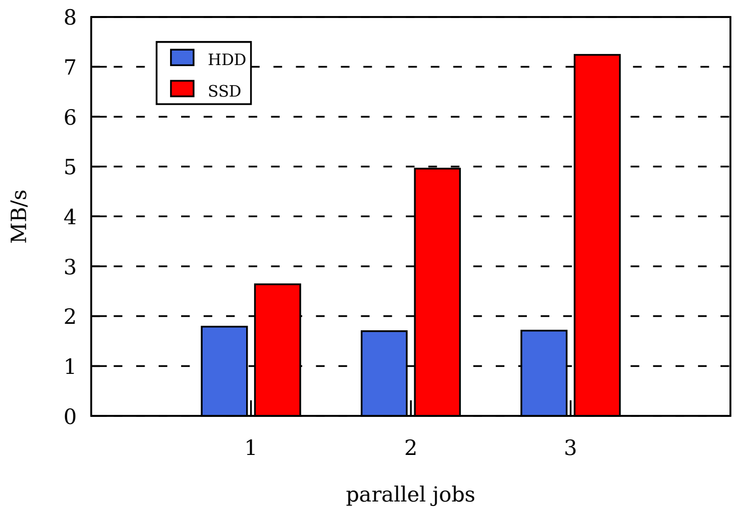

Figure 1.11 A comparison of a single HDD and a single SSD with regard to the data throughput rate of the ALICE analysis tasks querying the Monte Carlo data. SSD performance scales almost linearly with addition of parallel I/O bound jobs, while HDD performance is saturated at less than 2 MB/s.

The tested SSD is an Intel X25-E 2.5” SATA II SLC Enterprise Solid State Disk SSDSA2SH032G1 with a capacity of 32 GB. The compared HDD is an HP 3G SATA 5400 rpm 2.5” SFF Entry Hot Plug Hard Drive GJ0250EAGSQ with a capacity of 250 GB. The relatively small capacity of the SSD is an obstacle for running a higher number of parallel jobs each accessing an own data set. To decrease the sizes of the data sets is not an option, because, for consistency, the tests need to be run over a significant amount of time and be comparable with the HDD tests described above.

1.8 Conclusion of Part 1

Overall performance of the ALICE data analysis is directly affected by the way how fast the read requests to data storage are processed, since an analysis application cannot continue its run until its read request is fulfilled. The capabilities and the proper configuration of the hardware, and the I/O access patterns of the user applications play equally significant roles.

The time it takes to fulfill the disk access request includes the time spent on positioning and the data transfer. If hardware capabilities define the time spent on the data transfer from the disk, the I/O subsystem configuration of the OS and the I/O access patterns of the user application determine the time spent on positioning to start the transfer. Thus, the way to reach the best performance the hardware is capable of is to reduce the number of the positioning operations. This ultimately means that the number of requests for the disk access by the user application should be kept as low as possible. As it has been presented by the example of the AliRoot analysis tasks, this can be achieved either by sorting and merging disk access requests or by prefetching the data required by the application from the disk into the memory in an optimized way. The example of an analysis train shows an elegant way to avoid duplicate read requests by grouping the CPU workload of different tasks by the analysis data.

As a result, the direct cause of underutilization of hardware throughput capability is a large number of small, possibly random, read requests for the data on disks. The indirect causes leading to low I/O performance boil down to the following:

- a large number of relatively small ESD files,
- a large number of branches in the ESD tree, and, consequently, a large number of baskets inside the ESD files,

- numerous and superfluous queries to Kinematics.root, TrackRefs.root.

The described work provides the first overview of the physical layout of the ALICE data files and the I/O access patterns of the ALICE analysis code. The analysis code inside tasks and trains is user-written. Consequently, one cannot take for granted, that the code is going to be efficient with regard to disk access. The presented attempts to optimize performance concern the AliRoot analysis framework, in order to mitigate the inefficiency on a general level.

The results of this research have been presented at the ALICE Offline Week in October 2009, where the proposed optimization techniques have been approved for implementation, too. For the ALICE Tier-2 at GSI, these results have allowed to focus on analysis train as an effective way to utilize the local computing resources for analysis. A designated ALICE collaboration member collects the tasks from the ALICE physicists, attaches them to the analysis train and submits the train for execution.

2 Integration of Storage Resources

High-performance Computing, serving as a background for the topic of this thesis, implies heavy utilization of a large number of computing resources for intensive calculations. In order to provide a solution for advanced computing tasks that require large storage capacity and enough CPU power to get the results of the calculations quickly enough, computing resources have to be somehow integrated together. Integration should make it possible for resources to be used in parallel for a single purpose. Ideally, a result of such integration should be a set of resources which has a total power of a sum of powers of its units. Provided there is an interface, this set of resources can be regarded as a single virtual resource. It should be possible to dedicate the full capacity of this virtual resource equally easy to a single task or to a number of tasks running simultaneously. Practical solutions for integration which are considered in this chapter are limited to those available under a free software license.

2.1 Requirements

Thorough consideration of requirements for the desired computer system is needed before different solutions for integration of resources may be proposed and examined. The requirements for the distributed storage system of the ALICE Tier-2 centre analysis cluster can be divided into the following categories (based on [Oleynik]).

General:

1. The system must be available for Linux under a free software license.
2. The system must run on commodity hardware.
3. The system must be a general-purpose system.
4. The system must work over the Ethernet network.

Capacity:

1. The system must be able to grow indefinitely in data capacity with the addition of scalable units.

Functionality, performance:

1. The aggregate data throughput rate must scale up when more scalable units are added.

2. The system must support concurrent I/O access from hundreds to thousands of clients in the computing cluster.
3. The system must support a workload with a mixture of reads and writes, random reads, handle concurrent writes to a shared file.
4. The system must be able to queue the offered load before performance degradation, without dropping requests or deadlocking.
5. The I/O operations should be spread across the storage units and the underlying network infrastructure.
6. The system must not suffer from deadlocks, nor be significantly impaired by hung or deadlocked clients.
7. The performance of the system should not degrade with time due to data fragmentation.
8. The capability to control the number of WAN (Wide Area Network) and LAN (Local Area Network) transfers independently is desired.
9. The capability to integrate direct-attached storage of the computing nodes is desired.

Data integrity:

1. The capability to scan the system for file corruption without excessive impact on performance is desired.
2. The capability of the end-to-end integrity verification of the transfer protocols is expected.

Usability, maintenance:

1. It must be easy to add, remove and replace scalable units without causing performance degradation and stopping the entire system. Rolling upgrade feature is desired.
2. While some storage units may be offline, it must be possible to access those files on the remaining online units.
3. The maintenance and administrative operations must be amenable to invocation from remote systems. An application programming interface (API) is desired.
4. The system must be POSIX-compliant.

Global namespace:

1. The global namespace must be mountable and browsable on all nodes.

2. It must be possible to query namespace information from many clients simultaneously without affecting primary data operations.
3. It must be possible to analyze usage patterns to identify abusive users.
4. The global namespace must be scalable to support millions of files with no degradation in system performance.
5. The global namespace must be scalable to support hundreds of client operations per second.
6. It must be possible to recover the global namespace in the case of its crash. Backup possibilities have to be provided.
7. High-availability is desirable.
8. Quotas per user group are required.

Tape integration:

1. It must be possible to integrate the system with a tape back-end.

2.2 Classifying the Available Solutions

2.2.1 Definitions

As a starting point, the solutions to organize distributed storage systems are often divided into those based on the NAS (Network-Attached Storage) concept, and those based on the SAN (Storage Area Network) concept. Put simply, the NAS storage systems provide access to files on remote file systems united under single namespace. On the other hand, a typical SAN system allows its clients to enable file system access to remote storage blocks as if they are local to the client.

Interpretation of the definitions of the different file system architectures is generally confusing, and names are often misleading. Hence, the following list of definitions is presented in advance.

1. Network file system – any file system operating over a network as a NAS.
2. Distributed file system – any network file system which spans multiple independent storage servers and delivers them as a single storage unit.

3. Cluster file system – any distributed file system which can provide simultaneous I/O access to storage for nodes of a computer cluster.
4. Parallel file system – any distributed file system which can serve different I/O requests in parallel.
5. SAN file system / shared disk file system – any file system operating in a SAN.

Among other things, the POSIX norm defines a standard way for applications to obtain the I/O services from the I/O subsystem. This makes possible application portability [POSIX]. The POSIX semantics require coherence. The results of concurrent write operations have to be handled in such a way, that enables consequent read operations to reflect them. It is very expensive with regard to performance to provide the POSIX semantics for intensively shared files, since this requires a high amount of instant metadata updates. Certain extensions to POSIX for high-performance I/O are developed to address these issues and to relax some requirements [Welch].

2.2.2 SAN File Systems

Let us first consider the SAN file systems (also called shared disk file systems) used for the storage area networks [Tate]. A SAN is a specialized network, and its primary purpose is the transfer of data between computing nodes and storage. In a SAN, a storage device is not connected to a particular server bus but attached directly to the network. Thus, what a user can regard as a single block device on a local bus, in reality are multiple storage units connected through a dedicated network. SAN provides a network implementation of block storage protocols. Its main design characteristic is that client nodes are the only active elements of the system and are primarily responsible for the SAN file system operation.

The following properties of SAN may be regarded as advantageous:

- Storage is independent of applications and accessible through multiple data paths;
- Storage is accessible as a block device;
- Storage processing is off-loaded from computing nodes and moved onto a separate network resulting in higher application performance.

The following disadvantages are regarded as a trade-off:

- Management (formatting, partitioning, growing) of SAN would have to be done on client computing nodes;
- Gigabit Ethernet is the practical minimum;
- Major SAN protocols are not encrypted, hence security is an open issue.

Briefly described below are the SAN file systems GFS2 [Whitehouse] (developed by RedHat) and OCFS2 [Fasheh] (developed by Oracle). They are provided under the GPL (GNU General Public License), are POSIX compliant, and are both part of a Linux kernel. With their respective implementations of the Distributed Lock Manager (DLM) both file systems are able to provide simultaneous access to storage from multiple client nodes. Hence, every client needs to know which other clients can access the storage and keep locks on the resource.

The heartbeat technique is used by the clients to check each other's status and makes all clients periodically write into a special file in the system. To preserve the file system consistency the SAN file system has to isolate the client node which fails to produce the heartbeat. In this process, called fencing, the network connections to a misbehaving node are broken down, the misbehaving node is either shut down, restarted, or hung up in a state of kernel-panic. In such a way, the other nodes will not be stuck trying to access the resources for which the misbehaving node can have locks.

When the client nodes crash while the SAN file system is mounted, the file system journaling allows fast recovery. Recovery is done at the client node itself. However, if a storage device loses power or is physically disconnected, file system corruption may occur. Journaling cannot be used to recover from the storage subsystem failures. When that type of corruption occurs, one can recover the GFS2 file system by using the fsck command after unmounting it on all client nodes [GFS2].

GFS2 supports a maximum size of 25 TB. That is, to access storage capacity beyond 25 TB a client would need to mount more than one instance of GFS2.

File data in OCFS2 is allocated in extents of clusters, which can be from 4 KB to 1 MB in size. With the file addresses currently coded in 32 bits the limit for a file system size is 16 TB for 4 KB clusters, and 4 PB for 1 MB clusters [Mushran].

Once a GFS2 or an OCFS2 file system is created, its size cannot be decreased. Both file systems support quotas and the POSIX access control lists.

2.2.3 Aspects of NAS

There are plenty of different solutions for managing the network-attached storage. In contrast to SAN, the NAS solutions are operating on the file level. Fundamentally, the file systems of a NAS are called network file systems or distributed file systems. All of the solutions are variations of either the client-server or peer-to-peer models.

While choosing a particular cluster file system for a high-performance computing environment emphasis is put on two of the following features which are not necessarily present in all NAS file systems:

- fault-tolerance,
- parallel data access,
- scalability.

Fault-tolerance can be achieved through data replication among multiple file servers, and many file systems allow to specify the desired number of replicas. It is desirable to keep data replicas on devices on different RAID controllers, power circuits, network switches. Another way to achieve fault-tolerance is to set up the failover pairs of servers, where in case of a failure server's load would be transferred transparently to its mirror. However, a failover process sometimes takes longer than a simple reboot which may be enough to bring the server back online.

Parallel data access becomes possible when file system clients are able to perform a short query on a some kind of a metadata index to find out where to locate the data. It allows them then to directly connect to the server which stores the data for a lengthier transfer. Thus, multiple clients are able to concurrently and independently transfer data to and from multiple servers. Figure 2.1 illustrates that if the lookup queries 1 and 2 will be processed inevitably sequentially, transfer operations 3 and 4 can be processed in parallel.

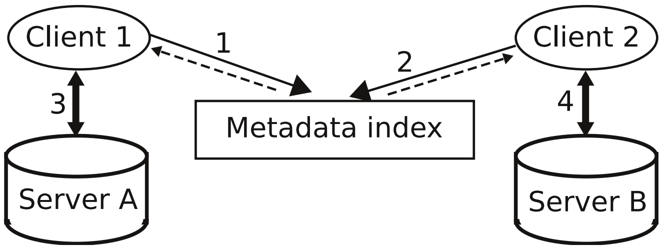

Figure 2.1 The origin of parallel access in the distributed file systems.

Scalability as a property of the distributed file system indicates that its capacity can be easily enlarged by adding new storage servers, and I/O access performance will scale proportionally. Among the issues that prevent distributed file systems to achieve scalability are provisioning of global namespace, and metadata management.

2.3 Metadata Management in Cluster File Systems

One of the crucial properties of a cluster file system is the way how the metadata requests, essential to operation of any file system, are handled. The metadata requests make up a large part of any user-driven workload, and affect performance and manageability.

The file system metadata are organized in a kind of an index of entries where each entry, called dentry, consists of a file's pathname and a corresponding inode number. The inode number points to an *inode*. In ordinary local file systems inode is a data structure which holds the information about the file like size, ownership, access permissions, and pointers to the contents of the file on the storage device. Different cluster file systems have different inode structures with different information available. For many local file systems whenever any file in the system is touched in any way its metadata have to be updated. For the cluster file systems this POSIX requirement for updating access times is relaxed.

This index which holds metadata of all the files in the system serves as a basis for providing a *global namespace*. A global namespace implements a unified view of the contents of all resources in the file system. It allows clients to see which files in which particular logical hierarchy are stored in the system without exposing their physical location. As the contents of a cluster file system grow, the index becomes larger, and its management offers a performance challenge. A simple search for a file in a large index introduces a latency.

With respect to the metadata management the cluster file systems may be symmetric, where metadata are spread among the nodes, or asymmetric with centralized metadata servers.

In an asymmetric system a stand-alone metadata server presents a single point of failure and a limit to system's scalability. Generally, the hardware requirements for such a server are especially high with regard to the I/O subsystem, and a failover pair is de facto a must. For cases where a single metadata server introduces a performance bottleneck, several metadata servers can be clustered together. Availability of multiple metadata servers enables parallel access to metadata, potentially relieves the load of a single server, and increases the rate of processed queries.

Keeping metadata safe and consistent is critical, since corruption or loss result in user data becoming unrecoverable. *Journaling*, the method adapted from the local file systems, is often used to cure inconsistency. In this method, pending updates are first written to a journal, a write-ahead log which can be used as a reference to complete the I/O operations. This allows a fast system recovery in case of a server crash before I/O completion. In cases where metadata are replicated, a difficult problem of keeping multiple instances in sync arises. Special algorithms for establishing the correct instance of metadata have to be in place.

At the example of GlusterFS [GlusterFS] one learns that it is possible to avoid metadata management in the cluster file systems altogether. In this case a cluster file system relies on a local file system at the servers, which would manage the metadata on its own. Whenever a file is written to a directory in GlusterFS, a hash number for the file is computed based on its name. According to the hash number a file is placed on one of the servers. Information to what servers the file can be dispatched is attached to the file’s parent directory. Thus, to retrieve a file or its metadata you have to recompute the hash.

Although this approach avoids the need to manage metadata centrally, it introduces several limitations to GlusterFS. In the case when a certain server for a given directory has crashed, it is not possible to write a file whose hash number would map it to the affected server. Besides, the request to show the contents of the global namespace with ‘ls’ command results in a broadcast of lookup queries to every server, which lead to a significant network traffic as the number of servers increases.

The files written via the GlusterFS interface are placed on to the local file systems of the storage servers in an ordinary way, without any transformation. When a storage server is deattached from GlusterFS, it is possible to easily access the files and recover the data via the local file system interface. Although, one has to note that GlusterFS does not support such scenarios where files are written through another interface to the local file system of a GlusterFS server.

As previously mentioned, the SAN file systems manage files on the level of blocks. On the other hand, the NAS open-source file systems such as Lustre [Braam], MooseFS [MooseFS], Ceph [Weil], PVFS [Carns] manage files as objects. When a file is written into such a cluster file system, it is split into multiple objects of a specified size. Object-based storage provides greater flexibility for achieving high performance and fault-tolerance for the cluster file systems. Objects of a single file can be spread across multiple servers, and thus can be accessed in parallel with increased cumulative throughput. Replication on the level of objects (for the sake of fault-tolerance) contributes to economic utilization of resources.

Whereas in such systems like Lustre the metadata server provides clients with list of objects' locations for a particular file, in Ceph a client is provided only with a cluster map, placement rules, and can compute a location of the objects of a file with a globally recognized function. This method avoids the need for maintenance of allocation tables and can increase performance.

The objects are stored as ordinary files on the local file systems such as ext3, which take care of allocating blocks to them on the storage devices. Without the metadata which keeps information on which objects make up the file, the objects are useless. So if the metadata server has crashed, it is not possible to recover the data.

2.4 Bringing the Workload to the Data

In general, the NAS solutions are tailored for the kind of computer cluster model, where the NAS clients reside on the farm of computing nodes across the network from storage servers staging the data. In this model data are moved to workload. However, even in such a model computing nodes are often installed with RAID controllers and a number of local disks. In this situation direct-attached storage remains mostly idle if computing nodes access software and data from remote locations.

The question whether to invest in direct-attached storage (DAS) or not often arises at the stage of cluster design, or at the stage of scaling the capacity up. The relevance of this dilemma is also strengthened due to a relatively low cost overhead for DAS in commodity computers. However, besides economic incentive for a cheap storage capacity increase, DAS gives a potential opportunity for jobs to perform read access on the local disks, to avoid network transfer of data, and thus to bring workload to data. Among the disadvantages of utilizing DAS is the management overhead of such system and effects of the misbehaving software on data availability.

Such highly scalable and fault-tolerant distributed file systems as Hadoop's HDFS [Borthakur] and CloudStore [CloudStore] are developed to run on commodity hardware and have a capability for parallel computation on local data. These systems are used for the "write once, read many” workloads and require applications to use the MapReduce software framework [Dean] which makes use of data locality. Another system, Sector/Sphere [Gu], uses the stream-processing paradigm, adopted from the general-purpose GPU (Graphics Processing Unit) programming models for similar workloads, and adapts it for wide-area distributed computing. It is complicated to use such systems for ordinary POSIX applications. It requires significant remodelling of application, but there is an ongoing research in this field [Molina-Estolano].

For Lustre, there are certain impediments for running a client and the object storage server on the same machine, because both processes are competing for the virtual memory. If the client consumes all the memory and then tries to write data to the file system, the object storage server will need to allocate pages to receive data from the client but will not be able to perform this operation due to low memory [Lustre manual]. This situation leads to a deadlock.

It is possible to run a client and a server of GlusterFS on the same machine. However, it is not possible for a client to know what data are stored locally on the server, and the majority of I/O operations would still go over the network. Hence, the only advantage of utilizing local disks’ capacity of computing nodes would be tarnished by aforementioned disadvantages. The same would apply to all NAS solutions which do not expose any API for data locality to detect local files.

2.5 Utilizing Direct-Attached Storage with PROOF

Several ad hoc tools for management of data comprised of the ROOT files can be of particular benefit to the environment of the ALICE Tier-2 centre. One such tool, designed for parallel processing of the ROOT data, is called PROOF (Parallel ROOT Facility) [Ganis]. Among its core properties is the ability to take advantage of data locality.

PROOF has been developed to run ROOT analysis on large data samples in an interactive mode with a real-time feedback and a fast response time. A PROOF system consists of clients, worker, submaster, and master nodes. To run an analysis in PROOF a client sends analysis code and a list of data files to the master. It then distributes the task of running the code among the submasters and they in turn distribute it among workers according to a scheduling policy. The scheduling policy ensures the following points:

- workers process local data first,
- the workload is distributed in such a way that all the workers are finished with processing approximately at the same time, regardless of their hardware heterogeneity or difference in load,
- resources are shared by different user sessions.

After workers are done with running the analysis the results are sent back to the submasters for merging. In turn, the merged results are sent for subsequent merging to the master before the client receives a final result. The submasters can control up to 64 workers each and form an intermediate tier between the master and the workers to scale the system up.

For its own infrastructure PROOF uses the Scalla toolkit [Scalla]. Scalla consists of tools to manage and access the ROOT files storage, commonly referred to as the xrootd cluster. The xrootd cluster is a stand-alone storage system, and is not necessarily used for running the PROOF analysis. The general scheme of a PROOF system running on top of an xrootd cluster is presented in the Figure 2.2.

Each node in an xrootd cluster runs both xrootd and cmsd daemons. An xrootd daemon, a core element of Scalla, implements the xroot protocol, which is a remote access protocol optimized specifically for the ROOT files. A cmsd daemon indicates the server status and enables the clustering of the data servers under the global namespace. Similar to the PROOF hierarchy, there is a redirector/manager node governing the data servers. Also similar to PROOF, intermediate redirectors/managers are used to scale the system up. It is worth to mention, that the PROOF master does not have to be run on the same node as the xrootd redirector/manager. The following happens when a client accesses a file on an xrootd cluster:

1. the client asks the xrootd redirector for the file A;
2. the xrootd redirector asks cmsd manager to find the file A;
3. the cmsd manager queries all subordinate cmsd instances to find the file A;
4. each cmsd instance queries xrootd instance attached on the same node for the file A;
5. if the file A is found, its location is sent back to the cmsd manager;
6. the cmsd manager determines according to the policy the optimal xrootd server and notifies the xrootd redirector;
7. the xrootd redirector instructs the client where to find the file A;
8. the client connects to the chosen xrootd server and accesses the file.

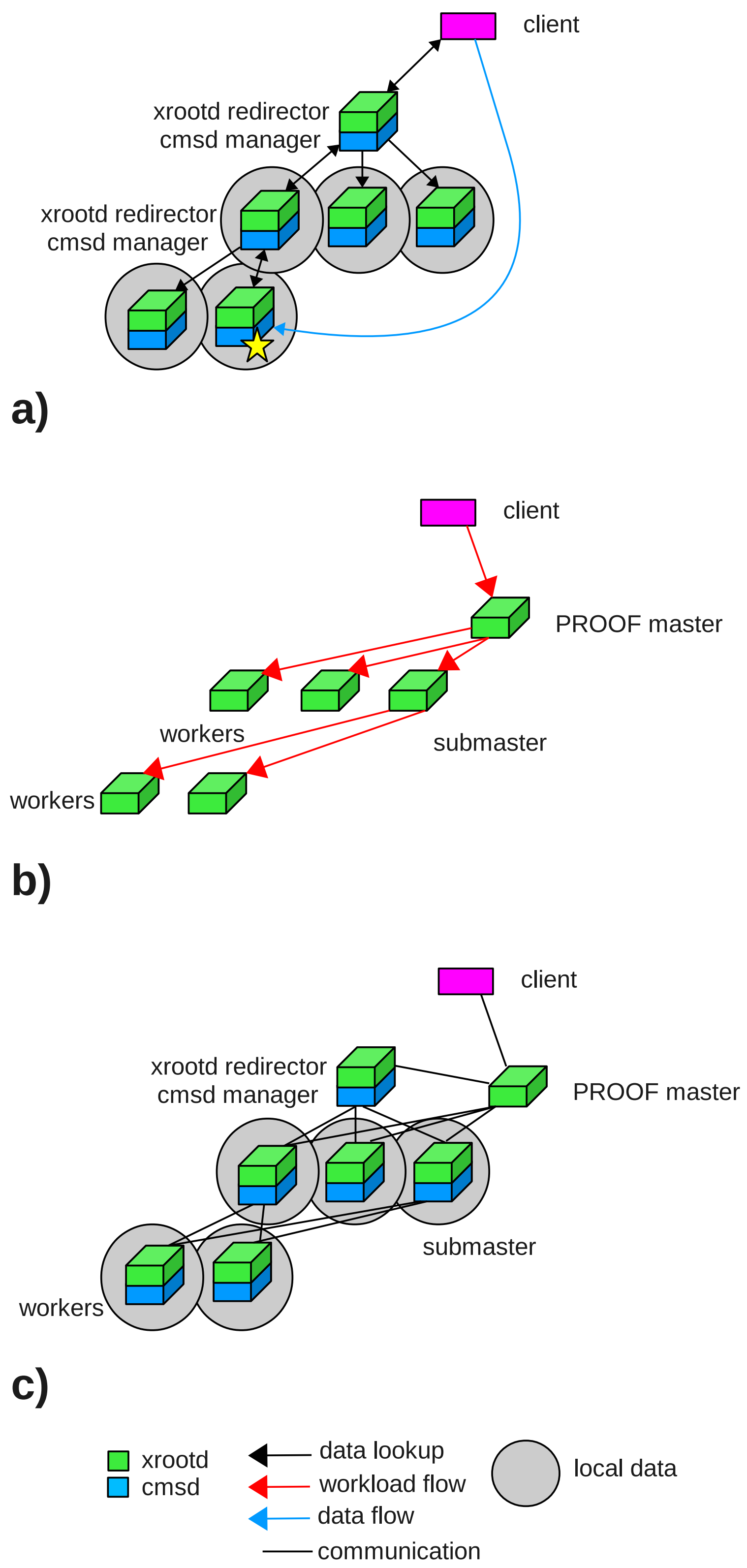

Figure 2.2 The PROOF and xrootd architectures: a) an xrootd cluster, b) a PROOF system, c) PROOF running on top of an xrootd cluster.

In addition to efficient access to ROOT files the xrootd cluster advantages include scalability and fault-tolerance. A new redirector/ manager node can be set up on demand, or in case of a crash. There is no file metadata store to grow. A crash of data server affects availability only of its local files. On the downside, the basic setup of an xrootd cluster lacks the POSIX interface, and critical data management abilities, e.g. file deletion.

2.6 A PROOF System Setup Options

A setup of CAF (CERN Analysis Facility) has demonstrated a first successful example of such PROOF system in operation [Grosse-Oetringhaus]. The data for analysis is staged from the tape back-end in the form of data sets by an ad hoc daemon on user request. Users are running a handful of concurrent PROOF sessions. A garbage collector is running, when disks are close to being completely full. It deletes data sets which have not been recently accessed.

To make data management on an xrootd cluster flexible, the XrootdFS [Yang] and XCFS [Peters][XCFS] file systems have been proposed. Both make use of the Linux kernel module FUSE to provide a possibility to mount an xrootd cluster as a file system. Although not all POSIX semantics are implemented, it is possible to run critical ‘ls’ and ‘rm’ commands. This gives an opportunity to avoid usage of the garbage collection mechanisms when data have to be deleted. Both file systems retain an ability to perform I/O access with the xroot protocol.

In XrootdFS, a dedicated daemon starts running a Composite Name Space (CNS). It stores metadata of all files in the xrootd cluster on the local file system. When changes occur on local data after xroot access, the xrootd daemon on a data server sends the update information to the CNS via a special plug-in. The CNS is updated after I/O accesses via FUSE too. The CNS is coupled with a simple server inventory (SSI), which provides an inventory for each data server. Both can be automatically recreated and replicated [Hanushevsky].

XCFS, on the other hand, is intended to be a full-fledged cluster file system, rich with features. It is possible to replicate data, to implement quota and access policies, to enable strong authentication. Metadata are stored on the metadata server which is the system’s single point of failure. Similar to XrootdFS, the metadata server holds inventories of each data server for consistency checks. It is worth to notice that it is not possible at the time of writing to attach XCFS to an already existing xrootd cluster. The local file systems on the data servers have to be formatted in a specific way.

A setup of GSIAF (GSI Analysis Facility) introduces a scenario where PROOF runs on the computing nodes which at the same time have been used by a distributed resource management system and have been executing batch jobs [Schwarz]. The PROOF workers are assigned a lower nice [nice] value than the batch jobs. If a PROOF session has been started when computing nodes are running batch jobs, PROOF workers are getting a higher priority for the CPU time.

Both setups of CAF and GSIAF are so-called static PROOF systems. Multiple user sessions are simultaneously coordinated by a single PROOF master controlling a fixed number of workers. In contrast, a dynamic setup is possible with the PoD (PROOF on Demand) toolkit which has been developed at GSI [Malzacher].

PoD provides a capability to start and fully control one’s own PROOF system with a desired number of workers. With the help of the batch system the specified number of workers can be distributed among computing nodes as regular batch jobs. After finishing the analysis, a user can easily bring its PROOF system down. Besides flexibility, another main advantage of using PoD instead of static PROOF is a complete decoupling of user sessions, which improves system reliability.

Thus, it is feasible to unite the DAS of computing nodes into a single storage pool of an xrootd cluster with a FUSE based file system on top. It allows to populate the local disks with data and run PROOF analysis processes on it. A PROOF cluster can be set up as a static system or a dynamic one by using PoD. In order for PoD workers to get distributed to the servers of an xrootd cluster and access data locally, a special batch queue can be set up. PROOF would distribute the workload with respect to data locality, and the xroot protocol would be used for accessing data. Either XrootdFS or XCFS could be used to manage and, most importantly, delete the data when it is not needed anymore.

2.7 Comparative Testing Results

The PROOF system has been used to assess and compare performance of GlusterFS, Lustre, and xrootd, as storage solutions. The configured PROOF cluster consists of 9 worker nodes with the tenth node playing the role of a PROOF master. Four tested setups are presented in Figures

2.3 and 2.4. Figure 2.3 shows the setups where data are accessed from a single node. On the other hand, Figure 2.4 shows, how data are accessed simultaneously from multiple PROOF worker nodes. PROOF uses the ROOT version 521-01-alice and a generated data set of ROOT files of an average 8.5 MB size with 5000 events, 20 branches, and without compression. The tests are made with the ProofBench benchmark analysis.

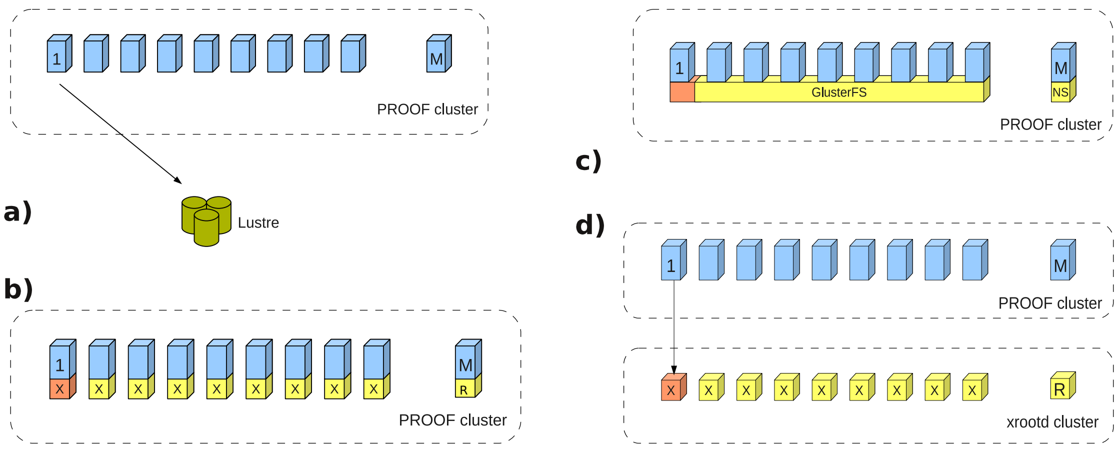

Figure 2.3 Storage test setups with a PROOF cluster. A single worker reads from a) a remote Lustre, powered by 30 storage servers, b) a direct-attached storage via xrootd, 'xrootd local', c) a direct-attached storage via GlusterFS, d) a remote node via xrootd, 'xrootd remote'.

GlusterFS (version 1.3.12) unifies the direct-attached storage of the PROOF nodes, with the namespace stored on the PROOF master node. The namespace in this version of GlusterFS is provided through the ‘unify’ translator, which makes sure that inode numbers are consistent across all the nodes in the ‘unify’ group.

Xrootd is represented with two setups. The first one is where the PROOF workers access direct-attached storage through xrootd, hence ‘xrootd local’. The second one is where the PROOF workers access a remote xrootd cluster, ‘xrootd remote’, with the same hierarchy of 9 data servers and a redirector.

Lustre (version 1.6) unifies capacity of 30 remote storage servers. This setup, in contrary to the 3 others, does not allow to flush the page cache on the servers between the test runs, and it is not known how the data are distributed among the servers. Consequently, the results of the testing should be viewed with a reservation, that Lustre has been used as a black box, powered by 30 file servers.

PROOF			Data rates (MB/s)			
Nodes	Workers per node	Total workers	GlusterFS	Lustre	xrootd local	xrootd remote
1	1	1	3	3	3	3
1	8	8	22	22	25	23

Table 2.1 The aggregate data throughput rates for PROOF analysis accessing data on a single node with a single worker and with a number of workers equaling the number of cores. The results show the same level of analysis performance for all four storage solutions, while 'xrootd local' with its native support of the ROOT trees proves to be slightly ahead.

For a single analysis PROOF process on a node, the results (Table 2.1) show no significant difference in performance between the setups. Although, notably, being optimized for the ROOT trees, xrootd sustains a slightly higher throughput rate when 8 workers on a node access the data simultaneously. In the same setup, but with the PROOF processes just reading the data from a single node (Table 2.2), for a single process, Lustre loses 25% in comparison to the rest. For 8 processes, Lustre achieves the best result with 50 MB/s. The result of xrootd for local access is accountably higher than the one of ‘xrootd remote’ and slightly better than the one of GlusterFS.

PROOF			Data rates (MB/s)			
Nodes	Workers per node	Total workers	GlusterFS	Lustre	xrootd local	xrootd remote
1	1	1	12	9	12	12
1	8	8	30	50	31	26

Table 2.2 The aggregate data throughput rates for PROOF accessing data on a single node with a single worker and with a number of workers equaling the number of cores. No analysis is performed by the benchmark, the data are just read. Read performance of Lustre scales up significantly better with the addition of jobs due to its aggregation of a three times higher number of storage servers than other solutions.

For multiple analysis PROOF processes accessing data simultaneously from all worker nodes (Figure 2.4, Table 2.3), ‘xrootd local’ provides by far the best throughput rate, significantly surpassing result of another local solution, GlusterFS. Lustre matches the result of ‘xrootd remote’ at 190 MB/s. The ‘xrootd local’ setup proves to be the most productive again for the similar setup with PROOF processes just reading the data at 107 MB/s (Table 2.4). The ‘xrootd remote’ setup sustains the higher throughput than the 'local' GlusterFS solution when there is a single process per node. With total 72 workers, 'xrootd remote' fails to surpass the local GlusterFS. Lustre's performance scales well, being higher than 'xrootd local' by 55%.

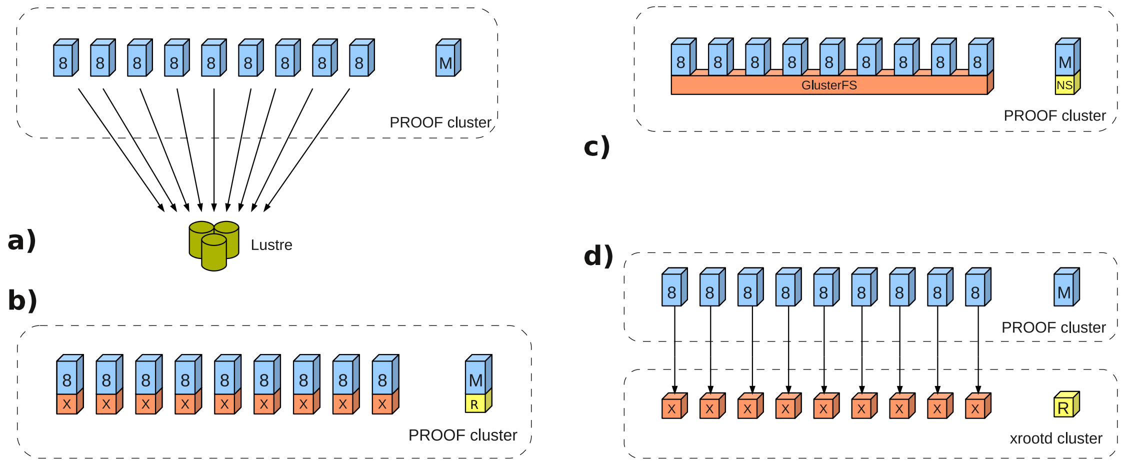

Figure 2.4 Storage test setups with a PROOF cluster. Multiple workers read simultaneously from a) a remote Lustre, powered by 30 storage servers, b) a direct-attached storage via xrootd, 'xrootd local', c) a direct-attached storage via GlusterFS, d) a remote node via xrootd, 'xrootd remote'.

PROOF			**Data rates (MB/s)**			
Nodes	Workers per node	Total workers	GlusterFS	Lustre	xrootd local	xrootd remote
9	1	9	27	26	28	27
9	8	72	172	190	220	187

Table 2.3 The aggregate data throughput rates for PROOF analysis accessing data simultaneously on multiple nodes with a single worker and with a number of workers equaling the number of cores. Local xrootd access for 72 workers significantly surpasses in analysis performance other data access solutions with remote Lustre access matching the result of remote xrootd.

PROOF			Data rates (MB/s)			
Nodes	Workers per node	Total workers	GlusterFS	Lustre	xrootd local	xrootd remote
9	1	9	89	72	107	103
9	8	72	219	421	271	216

Table 2.4 The aggregate data throughput rates for PROOF accessing data simultaneously on multiple nodes with a single worker and with a number of workers equaling the number of cores. No analysis is performed by the benchmark, the data are just read. Local xrootd access shows stronger read performance than other solutions. Only remote Lustre for 8 workers per node surpasses it, due to its aggregation of a three times higher number of storage servers and resulting enhanced performance scalability.

To summarize, the results conclude that, from the performance perspective, xrootd is the best solution for ROOT analysis and is specifically optimized for it. GlusterFS provides a reasonable performance for a direct-attached storage access. On the downside, a GlusterFS setup faces a significant management overhead trying to ensure that a process accesses data from the direct-attached storage. GlusterFS does not support this scenario. Finally, Lustre scales well and provides a generic cluster file system functionality.

2.8 Conclusion of Part 2. A Decision Taken at GSI

In a shared environment of a research facility with many supported experiments there is an incentive to consolidate all storage resources into a single file system. The obvious reason for this is that in case of fluctuating needs, and without implemented quotas, a single experiment can use more than its share of the total storage capacity. But, there is another reason that makes integration worthwhile to all user parties – increased data throughput through parallel access due to wider file distribution between file servers.

There are plenty of options on how to organize file storage at an ALICE Tier-2 Centre. To make a choice, compile the list of requirements similar to the one presented above, in the chapter’s introduction. Those solutions which meet the requirements should be carefully scrutinized with an emphasis on adoption history inside and outside of the community.

The systems which make it to the shortlist need to go through an intense testing phase. There is a fair chance of making a good decision. The importance of the initial decision is emphasized by the virtual impossibility of data migration for file systems of petabyte scale in a production environment.

At GSI, the choice has been made in favour of Lustre. Selecting the cluster file system functionality is necessary for feeding data to user analysis jobs processed in parallel at a computer cluster of several thousands cores (currently ca. 4000). With the first production version appearing in 2003, Lustre has since been popularized and installed at several of each year’s TOP 10 supercomputers (TOP500 list) most of which are used for research. The high-energy physics community has picked up the trend quickly, and Lustre has been discussed as a possible storage solution at each Computing in High-Energy Physics (CHEP) conference since 2004 and HEPiX meeting since 2005 [HEPiX] with all LHC experiments finding application for it [Carvajal][Wu]. Lustre is being the only open-source free cluster file system being under constant evaluation of the HEPiX Storage Working Group [Maslennikov].

A need for a general-purpose cluster file system with flexible data management capabilities, and immaturity of present POSIX interfaces are the main reasons why it has been decided not to put all storage resources into an xrootd cluster. Still, Scalla’s xrootd can be used for the WAN access to the ROOT files over a grid, and for a scratch space without long term management needs for analysis with PROOF.

A list of features, which includes the full POSIX compliance, scalability due to the absence of the metadata management overhead, and relatively easy deployment, have led to evaluation of GlusterFS. The biggest reason for considering and testing GlusterFS has been its potential to consolidate direct-attached storage of the commodity hardware. It has been of a particular interest to check whether it would have been possible to pair it with Scalla and make it provide a POSIX interface to an xrootd cluster. It has not been known, at the time, whether the HEP community has an experience with GlusterFS in this or any other regard.

Unfortunately, later, it has been determined that the GlusterFS client does not account changes to the local file system if they are implemented not through its own POSIX interface but with the xroot protocol. At this time, immaturity and its failure to provide a desired solution for integration of direct-attached storage of the cluster nodes have been the main arguments against the introduction of GlusterFS in GSI.

The tests results, described above and presented at the ALICE/FAIR disk storage workshop in 2009 [Zynovyev], have demonstrated how the

GlusterFS performance and the performance of Scalla are comparable to the Lustre performance on small scale installations. At the same time, the tests of an incomparably larger Lustre installation have produced highly satisfactory results. It has not been possible to reach performance saturation peak for the data throughput while the whole cluster has been busy with analysis jobs [Masciocchi]. This has paved the way for the final decision to stay with Lustre for the first years of the ALICE data-taking.

Because of the dedication of the GSI HPC group and the information exchange with the Lustre support team, the current installation of Lustre at GSI in its 4th year has a gross capacity of ca. 2 PB with a scheduled upgrade to 3.5 PB. It serves ca. 4000 computer cluster cores and stores ca. 100 million files. With the largest single production Lustre installation being approximately 26,000 nodes, and storage capacity of over 10 PB [Lustre size], there is some room to grow for Lustre at GSI.

Unfortunately, since Lustre is an object-based file system, its single metadata server (MDS) is the system's single point of failure and presents a performance bottleneck. Although, being long time on the roadmap, support of the clustered metadata is not implemented yet. To make it fault-tolerant, the MDS is set up in a high-availability pair configuration with a third server being on a cold standby. A single way to mitigate performance bottleneck is to enhance the MDS with more resources and GSI is planning to buy itself out of problems with an 48 core MDS with 128 GB RAM.

The clustered metadata and its inclusion in the Linux kernel under the LGPL (GNU Lesser General Public License) make the Ceph file system a valid alternative but no production Ceph systems are known at the moment, and its readiness is in question [Maltzahn].

The problems with Lustre include its relatively difficult diagnostics process, susceptibility of MDS to bugs which leads to frequent (for some versions) downtimes of the whole system, its clients operating in the kernel space of computing nodes and thus affecting nodes' availability. Besides, Lustre is very much dependent on stability of the underlying network. Not to mention, that frequent failures of the cheap storage hardware is the biggest reason for data unavailability in case of switched-off replication.

3 From Virtual Machines to ‘Infrastructure as Code’

Although the problems of the I/O access and storage with regards to the ALICE Tier-2 requirements have been examined above, it is rather clear, that most of the publicly funded scientific computing centres like GSI are serving more than one scientific experiment. Different scientific experiments have different requirements to computing, both to hardware and software. The ability to sustain operation of a heterogeneous environment and scale up should be envisaged in the computing centre’s architecture.

While the main goal of the infrastructure providers is to meet the requirements of all their users, the main goal of the users (scientific experiments) is to get their respective tasks accomplished (get results of computations). The infrastructure providers are interested in having the solution to satisfy the users as straightforward as possible. In turn, in most of the cases, the users are interested in accomplishing their tasks as fast as possible in a reliable way at a minimum cost.

Sharing the resources can lead to the mutual benefit relationship among the users. In case one of the users frees her share of resources, these resources can be utilized by another user. Through careful planning and negotiations users have a possibility to increase their resource capacity to satisfy peak demands by utilizing the resources originally provided by somebody else for different purposes.

The infrastructure providers should see their own incentive in fostering resource sharing among the users. For the users to be able to utilize each other’s resources, their applications’ requirements to the centrally provided infrastructure should converge. This in turn would lead to reduction in requirements to meet for the infrastructure providers and hence a more straightforward approach to administration of the resources.

Unfortunately, it is impossible to rely on the users settling on a single flavour of an operating system due to various reasons, epitomized by incompatible package dependencies. The concept of hardware virtualization is used to address this issue.

Hardware virtualization is a process of creating a simulated computer environment by host software for its guest software. One can distinguish paravirtualization, where guest software has to be modified to be aware of being in a virtual environment, and full virtualization, which allows guest software to be run without modifications and to be unaware of being in a virtual environment. Modern x86 hardware comes with architectural support for virtualization, hardware-assisted virtualization, effectively reducing performance overhead.

Host software manages guest software with the help of a Virtual Machine Monitor (VMM), also called hypervisor, which supervises access of guest software to CPU, RAM, and I/O devices. All unprivileged instructions can be executed natively on hardware without intervention of a VMM. However, all privileged instructions of the guest software generate a trap into the VMM. The VMM then takes care that those instructions are handled correctly and in a safe way for the system’s integrity. Those CPU intensive workloads that seldom use privileged instructions benefit from virtualization the most, while performance of I/O intensive workloads can suffer from overhead introduced by the VMM. However, the virtualization I/O overhead problem can be solved too [Liu].

3.1 Cloud Computing in a Scientific Research Environment

With decreasing performance overhead hardware virtualization has become broadly adopted as a helpful tool. It allows multiple users to exist on common hardware without knowing and hopefully without impacting each other [Abbott]. This has paved the way to a new trend in distributed computing, namely cloud computing.

Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [Mell]. The all-encompassing term of cloud computing leads often to confusion and it is very important to specify which of three service models is meant: SaaS (Software as a Service), PaaS (Platform as a Service), or IaaS (Infrastructure as a Service). The definitions of these three service models, as well as five essential characteristics of cloud computing, and four deployment models are formulated very well in [Mell].

Due to the heterogeneous nature of legacy scientific applications and their requirements to the OS, IaaS proves to be the most suitable cloud computing model to the environment of an academic research centre. Also from [Mell], in the IaaS model the capability provided to the user is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly limited control of select networking components (e.g. host firewalls).

The first IaaS providers have been those commercial companies which operated large capacity computing centres to meet their companies’ peak demands to computing and that could lease a part of their infrastructure to tenants for certain period between the peaks. Later it became profitable to scale up the infrastructure exclusively for leasing. Most of the pioneers of cloud-computing saw a unique possibility to drive down the costs for computing by avoiding the necessity to buy, install, operate, upgrade, and scale their own computing infrastructure.

Although the difference between scientific and commercial communities is substantial and is out of scope of this thesis, technically the IaaS model can suit them both.

3.1.1 Different Ways to Use Virtualization at a Computer Cluster

The advantages and disadvantages of adopting virtualization and an IaaS cloud model laid out below mostly concern batch and interactive processing. Whereas service hosting, although very important topic even in scientific environment, is out of the scope of this thesis.

If a computing facility, like an ALICE Tier-2 centre or a FAIR Tier-0 centre, is going to run applications inside virtual machines (VMs), there are several models to consider:

- with shared virtual machines,
- with private virtual machines controlled by a distributed resource management system (DRMS),
- an IaaS cloud,
- a hybrid model in which there are sets of physical resources reserved for distinct workflow models.

In the model with shared virtual machines one or a set of virtual machines are started on each physical host by system administrators. It can be done with the help of virtualization API and configuration management software (CMS) or with specialized cloud managing software. Once started, VMs are accessed and loaded by users, and stay online up to the moment when they crash or are shut down deliberately. Nothing changes for users in the way they run applications. System administrators can install a homogeneous software stack across all physical hosts in a computing farm. Homogeneity leads to easier operation of the infrastructure. The basic software stack needs at least the following:

- an operating system,
- a monitoring system,
- configuration management software,
- a virtual machine monitor,
- a virtualization API.

A set of virtual batch nodes with DRMS inside can be maintained alongside a set of interactive virtual nodes. Users submit batch job requests to a DRMS master service which may or may not be virtual and have no perception of running jobs in a virtual environment. Whether to run multi-core or single-core virtual nodes must be decided with respect to applications' requirements.

There are two potential shortcomings for users in the model with shared VMs:

- loss of native hardware performance, which can either be negligible or intolerable depending on the job workload;
- potential inability to use hardware accelerators or any other hardware devices, which VMMs do not support [Lagar-Cavilla].

On the other hand, the advantage for users is higher availability of resources that is the consequence of a shorter mean time to repair (MTTR) virtual infrastructure and potentially longer mean time to failure (MTTF) compared to a bare metal infrastructure.

The disadvantage of setting up an infrastructure with shared virtual machines compared to a conventional "bare metal" setup for system administrators is that virtual machines have to be provisioned, monitored, and maintained in addition to the physical hosts.

Another workflow model which takes advantage of virtualization is the one where a DRMS starts a VM for a scheduled batch job. A job description file would contain a requirement for a particular VM configuration and the DRMS would schedule the job for execution onto the worker node where such a VM could be started. The DRMS then starts the VM, executes a job inside the VM, and upon job completion and output transfer shuts the VM down.

The advantage of this workflow model in comparison to the one with a conventional DRMS and no virtualization lies in the ability to sandbox user jobs in a custom environment. In comparison to the model with shared VMs, this one does not require VM maintenance on the same level, since a VM would be short-lived and disposable upon job completion and possesses greater scheduling capabilities of existing DRMS. On the downside, the need to start a VM results in a delay prior to job’s execution.

3.1.2 Analysis of Virtualization Implications for Infrastructure Availability

Availability is a degree to which a system or a component is operational and accessible when it is required for use [Availability]. The way to measure availability which is presented below is taken from the very illuminating [Debois]. Availability is influenced by four major parameters (Figure 3.1):

- Mean Time to Diagnose (MTTD) – the average time it takes to diagnose the problem,
- Mean Time to Repair (MTTR) – the average time it takes to fix the problem,
- Mean Time to Failure (MTTF) – the average time there is correct behaviour,
- Mean Time Between Failures (MTBF) – the average time between different failures of the service.

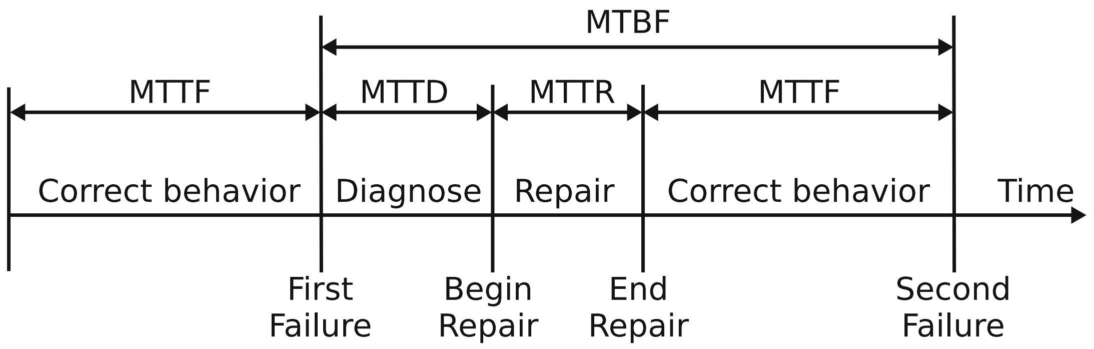

Figure 3.1 The parameters which influence availability.

Consequently, availability can be expressed as A = MTTF / MTBF = MTTF / (MTTF + MTTD + MTTR). On the other hand, availability can be calculated in terms of 9s as shown in Table 3.1.

Availability	Downtime
90% (one 9)	36.5 days/year
99% (two 9s)	3.65 days/year
99.9% (three 9s)	8.76 hours/year
99.99% (four 9s)	52 minutes/year
99.999% (five 9s)	5 minutes/year
99.9999% (six 9s)	31 seconds/year

Table 3.1 Availability in 9s.

To keep availability of a computing farm close to 1, or 100%, is obviously in the interests of users and is the duty of system administrators. Availability of a computing farm can be attributed to the service level objectives (SLO) such as CPU capacity (for example in HEP-SPEC), access and bandwidth (Mb/s) to the network storage system.

MTTR and MTTF are those parts of the equation which can be affected by virtualization. In case user batch jobs are distributed between the virtual machines, the users cannot crash with their processes the physical host. User batch jobs spawning child processes, which the DRMS can fail to control, or any kernel space code which can freeze the system, like in case of synchronous read requests without time-out to an unavailable file system, are not a threat to the physical host, but only to the virtual machine. When a virtual machine crashes or malfunctions, there is no need to repair it. It can be quickly deleted and re-provisioned. In another case, where there is a problem with a physical host (not related to the user workload) and the host has to be taken down for maintenance, virtual machines running on it can be migrated to a different physical host. In case of live migration, the VM does not even have to be shut down, if there is a storage system in place which is shared among the physical hosts. To summarize, the following features of virtualization technology lead to a shorter MTTR:

- easy and fast deletion and re-provisioning of a VM,
- migration of a VM.

Live migration capability results in a longer MTTF for virtual infrastructure too, since VMs can stay online, even when a part of the physical infrastructure goes down. If there are several VMs on a single physical host, one crashed VM will not affect the other ones running on the same host if they are not interdependent on the application level.

Because of clear resource boundaries (CPU, RAM, network I/O) there is no interference between user processes on different VMs. Even if configuration management software (discussed below) takes good care of keeping a VM clean, there is still a probability that with time the OS will get clogged with old obsolete software, configuration files, log messages, and remnants of user applications. This can lead to a system crash over time. There is little overhead in recycling an old or used VM, deleting the instance and starting a fresh one, once regular recycling procedure is set up. To summarize, the following properties of the virtualization technology lead to a longer MTTF:

- clear resource boundaries between VMs,
- live migration of a VM,
- easy recycling of a VM.

3.1.3 Analysis of the IaaS Benefits in a Scientific Environment

Another option to set up a computing infrastructure on virtual machines is to build a private IaaS cloud. On the contrary to public clouds, such as famous commercial Amazon Elastic Compute Cloud (EC2) or Rackspace, a private IaaS cloud is used only within a single organizational unit like a scientific research centre. Somewhere between public and private clouds there are also so-called community clouds shared by several organizations serving a community and closed for public access, such as IBM’s Research RC2 [Ryu].

For an IaaS cloud system administrators need to deploy the same basic software stack as for the infrastructure with shared virtual machines. But in IaaS cloud model system administrators do not provision VMs for users in advance, users deploy their own private VMs through the cloud interface on demand. In addition to those mentioned above for the model with shared virtual machines, the disadvantages of the private IaaS cloud model for users include the necessity to learn how to provision and maintain VMs themselves.

On the other hand, users would be able to enjoy some outstanding advantages:

- no issues with software platform compatibility for applications,
- freedom to have a personalized environment with all requirements met,
- total control of the execution environment,

- easy recovering (if there is a problem – just restart),
- dynamic scaling of infrastructure depending on the workload,
- easy migration between different IaaS cloud providers.

Certainly, user's ability to start VMs on demand requires regulation. A pricing model has to be implemented to motivate users to release resources which they do not utilize. To do such accounting in a private cloud no real money should be involved. With introduction of a billable unit like "instance-hour consumed" and accompanying rates users may be charged per time their particularly configured VM instance is occupying the resources. The rates would differ for the number of CPUs or RAM capacity per instance. In public clouds a similar model allows users to be charged in real money only for the resources they use.

Although to set up a private IaaS cloud takes a tremendous effort, once it is in place system administrators have a much more stable environment, prone to automation and efficient maintenance without the complexity of a conventional infrastructure on "bare-metal" machines which cracks under user software. In an IaaS cloud, maintaining the software stack within running virtual machines is largely a user's responsibility. System administrators can prepackage a VM image for users according to their custom requirements, make sure a VM instance is started, runs, and gets its share of resources. Once a VM instance falls into the user's hands, the user is free to install and run her own software, and the system administrator is no longer responsible to keep the software stack alive. If a VM crashes due to user actions, the user is free to delete or re-provision this VM according to a charging policy, and no other users are affected. Adoption of the IaaS cloud model gives infrastructure providers the following advantages:

- no interference between users, clear resource boundaries,
- no necessity of a shared software platform for all users,
- users cannot crash the physical host,
- user's infrastructure is not in the scope of the system administrators' responsibilities,
- it is possible to migrate a VM with no interruption in service,
- infrastructure is flexible enough to be easier adopted by users,
- higher number of users results in higher utilization and therefore cost efficiency,
- easier to keep pace with technological developments (easier to test and adopt new things).

3.1.4 SCLab. Implementation of an IaaS Cloud Prototype

Two aims have been pursued with the design and construction of an IaaS cloud prototype at GSI, called SCLab (Scientific Computing Lab):

- evaluation of free, open-source cloud technologies for production environment,
- creation of a cloud testbed with the purpose of studying how to use virtual machines to run applications.

To integrate SCLab prototype into the existing computing infrastructure at GSI, the decision has been taken to use preferably existing components and to introduce as few ad hoc changes as possible. Due to existing procedures for provisioning new servers, provisioning and maintenance efforts for physical hosts have been reduced to minimum.

KVM (Kernel-based Virtual Machine) [KVM] has been chosen as the VMM software for the SCLab physical hosts, because of its inclusion into the Linux kernel. This allowed to stay with the Debian OS, the default Linux flavour at GSI, and its Lenny version. The physical hosts have been centrally provisioned and maintained as all other servers in the computing farm of GSI.

KVM is a full virtualization solution distributed with Linux kernel since 2.6.20 release. For a long time, full virtualization solutions have been used to trade performance for compatibility. And although KVM had been losing out in performance to its famous paravirtualization counterpart Xen before, it has recently caught up [Deshane][Chierici], not least due to introduction of virtio, a Linux-internal API for virtual device drivers [Russel]. Virtio makes it possible for specific I/O drivers to be aware of running in a virtualized guest environment as in paravirtualization, and thus allows for efficient cooperation with VMM, which leads to a better performance.

For VM management the choice has been made in favor of OpenNebula, the open-source toolkit for cloud computing [Sotomayor]. OpenNebula is a virtual infrastructure manager that orchestrates storage, network, and virtualization technologies for deployment of any type of IaaS cloud. Eucalyptus [Nurmi] and Nimbus [Foster], which provide similar functionality, have been evaluated too, but at that time their respective versions have been using Xen as VMM technology and have been considerably harder to deploy in the given environment due to networking features. Feature-sets and design concepts behind all three toolkits have been thoroughly compared in [Sempolinski]. The result of the comparison proves that the decision to stay with network-agnostic OpenNebula for its customization abilities in the SCLab environment has been made correctly. A recent addition to these tools, Openstack [Openstack], looks promising and its evaluation is planned.

The interaction between OpenNebula and KVM on the physical hosts is done with the help of libvirt [libvirt], which plays the role of a common interface to different VMMs. Although it provides abstraction for all of the main VM operations, it fails to do so for some of the specific hypervisor features, like snapshots in KVM.

The architecture of the SCLab cloud built with OpenNebula is shown in the Figure 3.2. To provision VMs through the OpenNebula interface users log into the front-end node which plays the role of a cloud controller. It is possible to enhance OpenNebula with standard interfaces like the one of EC2 [EC2] or OCCI (Open Cloud Computing Interface) [OCCI], which can be called remotely. Before submitting a request for VM provisioning, all configurable VM parameters have to be specified in a configuration file. Parameters are divided into the following groups:

- capacity (amount of RAM, etc.),
- OS and boot options (boot device type, etc.),
- disks (image source, etc.),
- network (name of the network, etc.),
- I/O devices (VNC (Virtual Network Computing) port, etc.),
- context (files to include in a context device, etc.),
- placement (scheduling preferences, etc.),
- raw (arguments to pass to VMM, etc.).

After a request is submitted, a VM template disk image has to be transferred from an image repository to a particular physical host. OpenNebula aims to be agnostic to transfer mechanisms and it is possible to set up an own one, in addition to existing shared (network file system) and non-shared (secure shell) options.

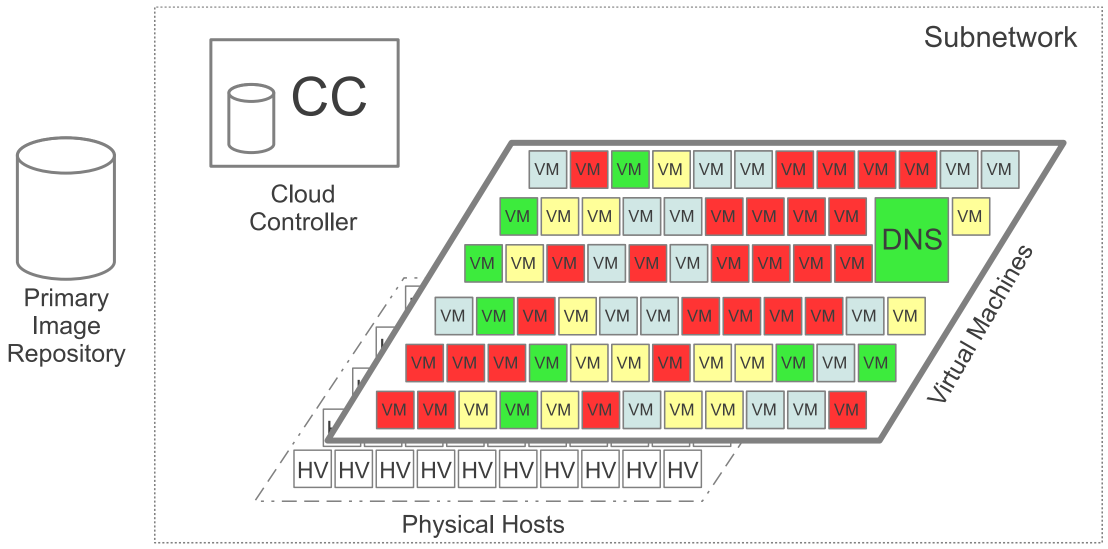

Figure 3.2 The basic private cloud architecture of SCLab. Virtual machines are colored to show heterogeneity. Physical elements are monochrome. A backup image repository is local to cloud controller. HV stands for hypervisor. DNS stands for Domain Name Server.

An image repository for SCLab is stored on a Lustre file system, which is shared between physical hosts and administered by another department in GSI. If it is online, the OpenNebula process copies it on the physical host to a particular local directory, but in case Lustre is not reachable, disk images are transferred from a backup repository at the cloud controller with the scp command. This improves SCLab availability.

The physical hosts and a shared file system are not the only things administered by another department that have been used for SCLab. The underlying network, which imposes constraints on the IaaS cloud architecture, is administered by another department too. An advantage of OpenNebula is that it does not rely on any particular network configuration, leaves the freedom of choice to system administrators, and provides the infrastructure to dynamically create new virtual LANs for VMs on the same physical network.

To be part of the GSI LAN and at the same time to avoid traffic interference with the external network, SCLab with its physical hosts and virtual machines has been moved to a dedicated class C subnet (for 256 IPv4 addresses). For such a static, relatively small scale cloud testbed, setup of a DHCP (Dynamic Host Configuration Protocol) server could be avoided. Instead of asking DHCP, VMs are assigned automatically generated IP (Internet Protocol) addresses through conversion at boot time of the last 4 pairs of hexadecimal characters of the MAC (Media Access Control) address to the 4 numbers of the IP address. Accordingly, 02:FF:0a:0a:06:32 is converted to 10.10.6.50 . The prefix 02:FF is reserved for VMs to distinguish them from physical devices. To be able to connect by hostnames to VMs from the GSI LAN, a DNS server for SCLab has been set up, and the DNS server for GSI has been modified to recognize it.

3.2 Efficient Computing Infrastructure Management

3.2.1 System Administration. A Key Obstacle for Infrastructure Scalability

Scalability, with regard to a computing infrastructure, can be either vertical or horizontal. Vertical scalability implies increase in resource capacity of a single infrastructure unit such as a computing server. On the other hand, horizontal scalability implies increase in the number of units comprising the infrastructure.

The precision of the highly complex ALICE analysis is limited by the amount of produced data, which can be physically stored, and the power of available computing resources to process it. In addition, the demands of the ALICE computing tasks are growing with the size of data produced by the experiment. Vertical scalability can not meet them because of the Von Neumann bottleneck, costs, and vague estimations for eventual capacity requirements. Thus, horizontal scalability with commodity hardware is inevitable.

The efficient scalability rate of the infrastructure can be defined by the scalability rate of its weakest link. With horizontal scalability being an issue every so often at an ALICE Tier-2 centre, the weakest link happens to be the manpower for system administration. If resource provisioning and maintenance require manual interaction of system administrators, then provisioning of new resources and maintenance of a scaled-up infrastructure require additional labour. It is hard to hire the costly highly skilled workforce on demand, on a short notice in a publicly funded sector. In addition, the increase in number of system administrators alone may not solve the problem of managing an evergrowing computing infrastructure. Rather, due to shared responsibilities, different skills and other workforce management issues, it may contribute to the complexity of the task. The problem of system administration can be mitigated by introducing those system administration techniques which foster automation, portability, and knowledge transfer. The more stages of the infrastructure life-cycle are optimized and automated, the less pressure is exerted on the manpower supply. Consequently, the system administration techniques are crucial to infrastructure scalability. Automation of system administration tasks is one of the key processes to make it possible.

3.2.2 Automation of Provisioning

The first step towards automation of resources provisioning in a conventional computing infrastructure is the introduction of a network boot, for example via PXE (Pre-boot eXecution Environment) [PXE]. In a typical network boot, the computing node’s network card broadcasts a DHCP request indicating that it is a PXE client; the DHCP server responds with IP information as well as the name and location of a network boot program to download via TFTP (Trivial File Transfer Protocol) [Sollins]. A network boot program, like FAI (Fully Automatic Installation) [Lange], would then install an OS with specified packages onto direct-attached storage.

In an IaaS cloud, automatic provisioning of resources is one of the design cornerstones. Arbitrary number of VMs can be started automatically from just a copy of a VM disk image with a pre-installed OS.

3.2.3 Automation of Maintenance

There comes a time when a fully functional computing infrastructure stops working properly. Something in the system breaks, and the clock for the MTTR is started. The system administrator becomes aware of the problem either after being contacted by a user, by looking at the related monitoring information, or by being alerted with an automatically sent message from the system itself. Although the third option may not be more complete than the first two, it is certainly the most reliable and prompt, since there is no human involvement.

Once notified, the system administrator diagnoses the problem. In case of a recurring or a well documented problem, it is very likely that there exists a script or a list of commands to execute. If the problem is new, it is very likely that such a script will be created to solve it. With time, the need to maintain scripts in an organized way becomes apparent. Using a version control system to keep track of changes to the code is necessary. In order for somebody to be able to reuse the script, it has to be well documented and written in a common language. For a better flexibility, and to facilitate sharing, a script must have a very precise indivisible purpose and be able to run across platforms. The time spent on finding correct commands or a script, writing a correct script, or adapting somebody else's script may result in an unnecessary delay to recovery.

The more steps on the road to recovery are automated, the shorter will be the MTTR. Automation saves time, increases infrastructure availability. It also allows to cope with infrastructure growth.

To be able to write a programming code in the form of a script or an application, which controls an infrastructure component or retrieves information from it, the component must have an API. Ideally, an API must be language-independent, so that it can be called from different programming languages, and accessible over the network, so that it can be called remotely, as a service.

As an example, one of the essential infrastructure components is a monitoring system, which gathers all kinds of relevant static and dynamic metrics of infrastructure components both on the system and application levels. It has to provide a network-accessible API, which can be remotely called for retrieval of a particular metric value. This would allow to write a code, which can detect and react on the change of the system state.

In a service-oriented architecture (SOA) (Figure 3.3), with the loosely coupled, abstract, reusable, autonomous, stateless, discoverable, composable services, one requires only orchestration to produce a new application, and, similarly, build an infrastructure.

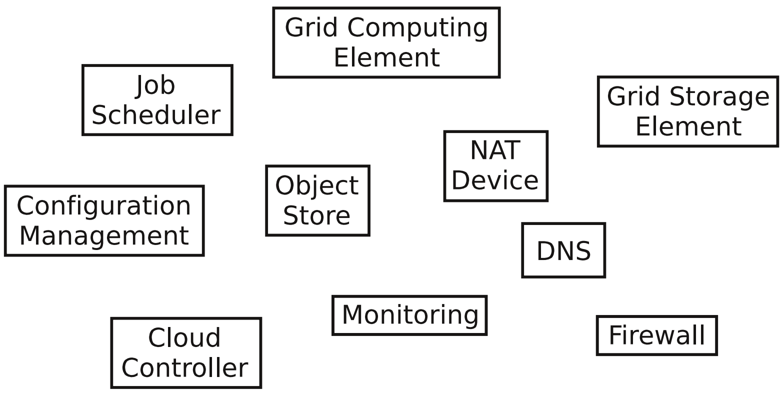

Figure 3.3 An example of a service-oriented architecture (SOA) with the loosely coupled, abstract, reusable, autonomous, stateless, discoverable and composable services. All the services require a remotely accessible API.

The desire for automation of provisioning and maintenance of an infrastructure and applications in a highly scalable network environment of an IaaS cloud and experience of SOA have led to the birth of the *Infrastructure as Code* (IaC) concept [Jacob]. IaC implies the breakdown of an infrastructure into modular network-accessible services for an easy and flexible integration into a functional, fully-automated system. With IaC, computing infrastructure is built by means of writing programming code. The “Holy Grail” of IaC is the reconstruction of a complete computing infrastructure from a source code repository and “bare metal” resources.

3.2.4 Solution. Adopting ‘Infrastructure as Code’

For higher degrees of flexibility, portability, scalability, recoverability, and transparency, it is necessary to automate as many stages of a computing infrastructure as possible. To address these issues in a scientific environment, the IaC concept from the commercial sector of Web operations is introduced.

Before turning to a solution of how IaC may be implemented in a scientific computing environment, it should be made clear that IaC is a general concept for infrastructure management and is not bound to any particular tools.

Unfortunately, not every infrastructure component, which has to be maintained, has an API. Besides, automation code is not that much of a help for continuous infrastructure operation and maintenance if it is scattered and has to be executed manually by system administrators. High reliability and availability are impossible without regular, organized maintenance checks. These issues are addressed by configuration management tools.

Configuration management, with regard to computing infrastructure, is a process of supervision of infrastructure components’ compliance with the assigned policy according to a specified protocol. Policy is a specification of what a component should do, and protocol is implementation of how it should be done. A basic architecture of a configuration management tool is presented in the Figure 3.4.

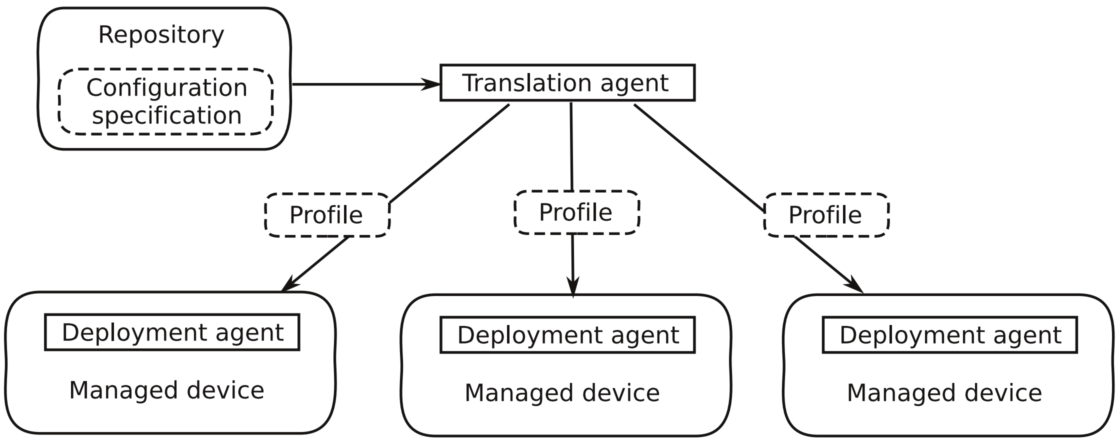

Figure 3.4 A basic architecture of a configuration management tool.

Starting with Cfengine [Burgess], most of the popular configuration management tools support a declarative syntax, abstract definitions, idempotence, and convergence. With a declarative syntax, for a policy to be executed, it is enough to specify its name, if its implementation is supported by the software. Definitions of policies are abstract and implementation details are hidden. Through idempotence configuration management tools take action only upon those resources which are not yet properly configured. Configuration can be gradual with each step bringing the resource closer to the state specified by the policy due to the intrinsic convergence.

The design principles and a comparison framework for the popular modern configuration management tools is presented in [Delaet]. Since for adopting IaC every infrastructure component should be a network-accessible service, a configuration management tool should also be modular, cooperative, composable, flexible, extensible, and repeatable.

Chef [Chef] is one of the configuration management tools which has been designed to meet all of these criteria. Its cooperative and composable origin and its reliance on general purpose Ruby programming language with a minimal custom DSL (Domain-Specific Language) distinguish it from the rest of configuration management tools. Chef, as a tool, epitomizes the IaC approach.

Using Ruby to write programs for infrastructure management, it is possible to interact with Chef as with any other Ruby library. Its rich network-accessible API makes it a suitable service for IaC. On the other hand, the fact that Chef uses Ruby, a popular open-source language, means that any kind of configuration management task can be expressed in code on any resources, as long as they support Ruby.

In order to configure with Chef some aspect of the system, a user writes *recipes*. Recipes are Chef’s fundamental configuration units. A Chef recipe is a Ruby DSL file that contains a list of *resources*, each of which represents a fraction of the system state which has to be implemented. Implementation is done by the *providers*, the OS specific abstractions of system commands and API calls. The most popular providers for variety of tasks are included into the Chef distributions, but ad hoc ones may be developed by the users themselves.

Cookbooks are the Chef’s fundamental units of distribution and are nothing more than collections of recipes, which are needed for configuration of a particular part of the system. Shown below are some recipes for the installation of ROOT. The Chef recipes for AliRoot cover similar installation steps and include a recipe for installing ROOT, which is to be run in advance. The way, how a particular installation recipe can be selected for a specific OS platform, is visible at the Listing 3.1.

```
case node[:platform]
  when "debian"
    include_recipe 'root::install'
    include_recipe 'root::config'
  when "scientific"
    include_recipe 'root::binary_install'
  else
    exit 1
end
```

Listing 3.1 A top-level Chef recipe for the installation of ROOT.

The Listing 3.2 shows how the compilation of ROOT has to be carried out. The selection of the dependency packages for a particular OS is done in the beginning. After a specified ROOT version is checked out of the Subversion repository, the code is built in the Bash shell with standard tools.

```
case node[:platform]
  when 'debian'
    [
     'g++',
     'make',
     'xorg-dev',
     'gfortran',
     'libxml2-dev',
     'libgl1-mesa-dev',
     'libglu1-mesa-dev',
     'subversion'
    ].each do |p|
      package p do
      action :install
    end
  end
  else
    exit 1
end

script 'build' do
  interpreter 'bash'
  cwd "/tmp/#{node[:root_tag]}"
  code <<-EOH
    ./configure --prefix=/opt/root/#{node[:root_tag]} \
    --with-pythia6-uscore=SINGLE --etcdir=/opt/root/#{node[:root_tag]} \
    --with-f77=gfortran
    make > build.log 2>&1
  EOH
  action :nothing
  not_if do File.exists?("/tmp/#{node[:root_tag]}/build.log") end
end

subversion "ROOT" do
  repository "http://root.cern.ch/svn/root/tags/#{node[:root_tag]}"
  revision "HEAD"
  destination "/tmp/#{node[:root_tag]}"
  action :checkout
  notifies :run, resources(:script => "build"), :immediately
  not_if do File.exists?("/opt/root/#{node[:root_tag]}/bin/root") end
end
```

Listing 3.2 A Chef recipe for compilation of the ROOT software.

The recipe at the Listing 3.3 documents how the ROOT program is installed into a specified directory, which is created beforehand, and how the logging files of all the installation steps are saved, before the temporary installation directory is deleted.

```
include_recipe 'root::compile'

directory "/opt/root/#{node[:root_tag]}" do
  owner 'root'
  group 'root'
  mode "0755"
  recursive true
  subscribes :create, resources(:script => "build"), :immediately
  not_if "test -d /opt/root/#{node[:root_tag]}"
end

script "install" do
  interpreter "bash"
  user 'root'
  cwd "/tmp/#{node[:root_tag]}"
  code "make install > install.log"
  subscribes :run, resources(:directory =>  \
              "/opt/root/#{node[:root_tag]}"), :immediately
  only_if do File.exists?("/tmp/#{node[:root_tag]}/Makefile") &&
          File.exists?("/tmp/#{node[:root_tag]}/build.log")
      end
end

script 'save_logs' do
  interpreter "bash"
  user 'root'
  cwd "/tmp/#{node[:root_tag]}"
  code "cp *.log config.status /opt/root/#{node[:root_tag]}"
  subscribes :run, resources(:script => "install"), :immediately
  only_if do
    File.exists?("/tmp/#{node[:root_tag]}/build.log") and
    File.exists?("/tmp/#{node[:root_tag]}/install.log") and
    File.exists?("/tmp/#{node[:root_tag]}/config.log") and
    File.exists?("/tmp/#{node[:root_tag]}/config.status")
  end
end

directory "/tmp/#{node[:root_tag]}" do
  recursive true
  subscribes :delete, resources(:script => 'save_logs'), :immediately
end
```

Listing 3.3 A Chef recipe for the installation of ROOT at the dedicated location.

The following code snippet at the Listing 3.4 shows how to invoke Ruby code from a recipe. Using Ruby, a general-purpose programming language, it is possible to build any kind of functionality into the recipes. In this case, the ROOT environment variables settings are placed to the dedicated OS file.

```
ruby_block "append_rootvariables_to_/etc/profile" do
  block do
    open("/etc/profile","a") do |f|
          f.puts "export ROOTSYS=/opt/root/#{node[:root_tag]}"
          f.puts "export LD_LIBRARY_PATH=$ROOTSYS/lib/root"
          f.puts "export PATH=$PATH:$ROOTSYS/bin"
          f.puts "export INCLUDE_PATH=$ROOTSYS/include/root"
    end
  end
  action :create
  only_if do
    open("/etc/profile") { |f| f.read.scan("export ROOTSYS=") }.empty?
  end
end
```

Listing 3.4 A Chef recipe code for configuring the ROOT environment variables.

Among other features, Chef facilitates the use of the distributed version control system *git* [git] for managing the source code of the repository and cookbooks. This allows to develop and exchange cookbooks in a flexible way. While general-interest cookbooks may be uploaded to the Opscode (Chef support company) website [Opscode] for a free user download, cookbooks, which contain information specific to a particular organizational unit, may be stored in the internal network for internal access.

The Chef clients are the deployment agents of Chef and run on the managed devices. They pull the configuration specifications from the central Chef server, which also stores the cookbooks and collects infrastructure’s metadata. The Chef clients can query the Chef server for the metadata, taking advantage of its search abilities, to dynamically integrate infrastructure parts. Pulling a configuration specification from a single server at regular intervals will eventually introduce a performance bottleneck at a large-scaled infrastructure, that is why Chef supports generation of random intervals for the Chef client pull mechanism.

A key capability of Chef is its powerful command line interface, called knife. Knife can be used remotely for virtually any Chef-related task, such as device configuration specification, infrastructure metadata query, bootstrapping target devices (installing Chef), and even provisioning of new virtual machines. It can be used to run any shell command on the Chef-managed device, and thus, effectively, substitutes such tools as Capistrano [Capistrano], which are used for executing commands in parallel via SSH (secure shell) on multiple remote machines. This functionality allows knife to be used for enforcing Chef action on demand without the need to wait for the next pull of the Chef client.

3.2.5 Virtual Clusters on Demand

Once multiple VMs are running on an IaaS cloud, the next step is to use them for parallel processing. For this purpose, the workload must be somehow distributed among the VMs, which would play the role of worker nodes. In case of ALICE computing jobs, this may be done either with PoD for PROOF workload or with a conventional DRMS for analysis or simulation batch jobs.

The steps, which a user has to take on a VM after it is scheduled and before it is ready to process the user workload, may be different for distinct workflows of private and public clouds. The difference comes from the way, how the VM image can be contextualized for use in the virtual environment. *Contextualization* is the name for a process of VM preparation, since its goal is to make the VM aware of its deployment context [Bradshaw]. Contextualization may be performed to various extent at any of the three stages:

1. The offline stage. Advance contextualization of a VM image.
2. The deployment stage. Contextualization performed by the cloud middleware at the VM boot time.
3. The online stage. Contextualization of a running VM instance.

If a user gets an opportunity to submit a VM based on her custom built VM image, then contextualization may be performed to some extent at the offline stage by a user herself. In this case, a user may preinstall the required software stack and create an own user account or store his credentials. If a user does not get this opportunity, he must rely on contextualization at the online stage.

A VM must be identified in the infrastructure network with an IP address for a user to get network access. How to set up a network interface of the VM depends on the cloud network configuration. If there is no DHCP available in the IaaS environment, then the VM image has to be edited with a static IP configuration or, like described for SCLab above, with a mechanism for IP address generation. Since such network information should be out of the user’s scope of knowledge, this kind of contextualizaton has to be performed by system administrators in advance. This is not so flexible as when a DHCP server is operating. In case of a functioning DHCP, IP assignment is done at the online stage of contextualization.

If a user would like to start a VM based on an image provided by an infrastructure provider, then contextualization performed by the cloud middleware at the deployment stage may be used for enabling user access to the VM, for example by generating or by placing provided SSH keys or certificates for the root account.

Implementation of contextualization at the deployment stage may be different from one cloud middleware to another, and may be completely transparent for the user. In case of OpenNebula, it requires the VM image to be modified in order for a VM to call a particular process that looks for information provided during deployment on the specified mounted device.

Since various cloud toolkits have different capabilities and various IaaS cloud providers have different image management policies, a user only gets full freedom for contextualization at the online stage, when a VM is running and accessible. This freedom is guaranteed on any IaaS cloud. That is why, when a cloud provider may be changed, it is of greater importance to be able to flexibly contextualize already running VMs with basic configuration. The IaC concept allows to automate this way of contextualization. It must be mentioned though that having as much of the required software preinstalled in the VM image as possible saves time.

To set up parallel processing for the ALICE analysis and simulation jobs with a DRMS, once user VMs are running and accessible, further contextualization is required. At this online stage VMs have to be assembled into a properly configured, functioning DRMS.

As in [Keahey], Torque [Staples] is used for this purpose. The Torque system is comprised of clients, worker nodes, and a server. For the proposed scenario, only the server and the worker nodes run on the VMs in the cloud and a client interface is installed on the server by default. The Torque configuration file for the server has to include the hostnames of the worker nodes. Configuration files of the worker nodes have to point to the hostname of the Torque server. Torque also requires ‘password-less’ authentication between the nodes for the user who submits the jobs. This is achieved by distributing the user’s public key among the nodes.

In addition, each of the worker nodes needs access to configured ROOT and AliRoot installations. A concept of CernVM [Buncic][CernVM] has introduced a VM image specifically designed for the LHC experiments, which stores only references to scientific software and downloads and caches a particular library from the central repositories only when it is called. The usage of CernVM allows to have compact size images, and does not require to maintain the image and the software stack, since it is done centrally at CERN. But in case of any other VM image, AliRoot has to be explicitly installed inside the VM or available on a mounted shared file system. At the same time a shared file system, like Lustre, may also be a source of data for analysis jobs.

The proposed architecture of a virtual cluster is presented in the Figure 3.5. Its implementation is done with Chef because of its IaC compliance. If there is a possibility to start VMs based on custom preconfigured VM images, these images could include installed AliRoot, the Lustre client, Chef software, and a configuration file for the Chef client with a desired role. But, if there is no such possibility, and one has to use generic, ‘naked’ VM images, the Chef software can be remotely bootstrapped, i.e. installed and configured, in the VMs with the knife command.

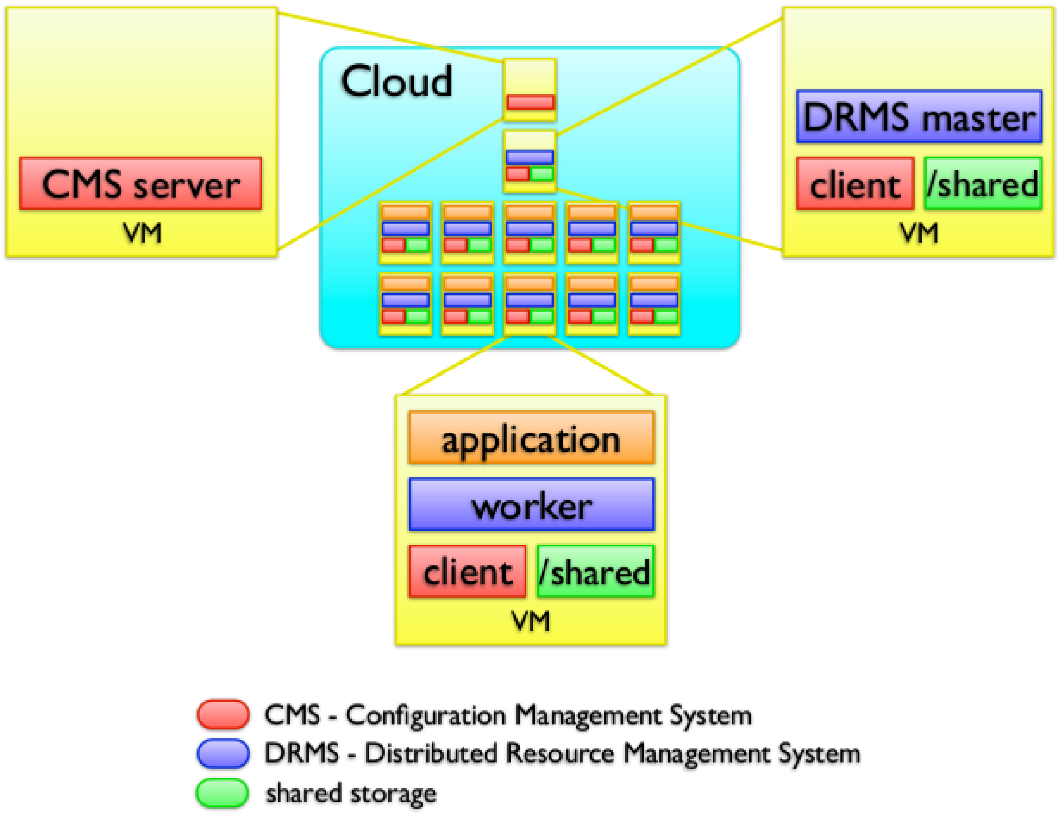

Figure 3.5 An architecture of a virtual cluster. There are three distinct VM roles: a CMS server, a DRMS master, and a worker node.

The Chef client should run on every submitted VM, and one VM must be reserved for the Chef server. To be able to connect to the Chef server to pull configuration information, the Chef client must be configured with a correct Chef server address. Placing of a correct server address to the right configuration file is done with knife at the online stage of contextualization.

In order to configure an AliRoot cluster with Chef, a Chef node for a Torque server VM needs a role, and a Chef node for a Torque worker needs a role. Roles define functionality for each of the nodes through recipes and attributes. It may be that one role, such as, e.g. aliroot_worker_node (Listing 3.5), is enough to set for a node, but it can be combined with other roles or specific recipes as well. The aliroot_worker_node role specifies cookbooks needed for the worker nodes and which include recipes for configuring a Torque worker, a Lustre client, and AliRoot. An aliroot_torque_server role can be composed similarly, but should have a recipe for a Torque server applied instead.

```
{
  "name": "aliroot_worker_node",
  "chef_type": "role",
  "json_class": "Chef::Role",
  "description": "The base role for a worker node with AliRoot",
  "override_attributes": {
   },
  "default_attributes": {
    "users": {{"id": "alivo000", "account" : { "uid" : "20000", "gid" : "20000"}},
           {"id": "alivo001", "account" : { "uid" : "20001", "gid" : "20000"}}},
    "root_tag": "v5-23-02" ,
    "aliroot_tag": "v4-16-Rev-08"
   },
  "run_list": [ "role[torque_worker]","recipe[aliroot]",
            "role[ganglia_agent]","recipe[lustre::mount_prod_system]" ]
}
```

Listing 3.5 A Chef role for a Torque worker node with AliRoot.

The recipe for setting up a Torque worker includes the code shown on Listing 3.6. So, when a Chef client applies the recipe at the worker node, it sends the query to the Chef server asking for a name of the node which has the “torque_server” role applied. Upon getting an answer, it puts the name inside the configuration file. Similarly, the Chef client on the Torque server asks the Chef server for the names of the nodes which have the “torque_worker” role applied (Listing 3.7). The Torque server process and the Torque worker processes are restarted only if the contents of configuration files have changed. Once the processes are restarted, Torque is ready for job scheduling.

```
server_name = search(:node, 'role:torque_server')[0].name
log "Query index for TORQUE master: #{server_name}"
# deploy the configuration files defining the TORQUE server
template '/var/spool/torque/server_name' do
   source 'conf/server_name.erb'
   mode '0644'
   variables :server => server_name
  notifies :restart, resources(:service => "pbs_mom")
end
```

Listing 3.6 A Chef recipe code snippet for setting a correct Torque server name.

```
workers = Array.new
search(:node, 'role:torque_worker') do |worker|
   workers << worker.name
end
log "Query index for TORQUE workers: #{workers.join(',')}"
template '/var/spool/torque/server_priv/nodes' do
   source "conf/server_priv/nodes.erb"
   mode '0644'
   variables :workers => workers
   notifies :restart, resources(:service => 'pbs_server')
end
```

Listing 3.7 A Chef recipe code snippet for populating the Torque server nodes list.

Besides configuration management system such as Chef, another necessary component for operating a virtual cluster is a monitoring system. Ganglia [Massie] has been chosen for the described example of a virtual cluster. With its master-clients hierarchy it is configured similarly to Torque. The role ‘aliroot_worker_node’ may then be combined with a role like ‘ganglia_agent’, and ‘aliroot_torque_server’ with ‘ganglia_collector’.

So once all VMs, a Chef server VM and Chef clients VMs, are running, the Ruby code shown on Listing 3.8 can be executed on the Chef server to configure a private virtual cluster.

```
# apply a role to a torque server node
system('knife node run_list add torque_server.domain "role[aliroot_torque_server]"')

# apply a role to torque worker nodes
data = `knife search node "name:worker*" -i`
data.scan(/^\S+$/).each do |k|
   `knife node run_list add #{k} "role[aliroot_worker_node]"`
end

# run chef clients on all aliroot nodes to apply the configuration
system('knife ssh "role:aliroot*" "sudo chef-client"')
```

Listing 3.8 A Ruby code snippet for configuring a virtual cluster.

3.3 Virtualization Performance for ALICE Analysis

Extensive benchmarking has been carried out to establish the performance overhead caused by virtualization for the ALICE analysis jobs. Using KVM and virtio, the same analysis train and task jobs as in Part I of the thesis have been processed inside virtual machines. Besides, the benchmarking tests aim to answer an interesting question, whether to process jobs in parallel inside a single multi-core VM or to process jobs in separate, running in parallel, single-core VMs.

In these tests the data for analysis have been stored on the Lustre file system and accessed over the network, as in the case with analysis jobs processed at GSI. This allowed to disregard KVM performance of I/O access to direct-attached storage. Although the VM's hard disk is stored on a host in a file, no intensive disk access has been monitored on the host for any of the tests described below. Since the AliRoot processes are not parallel, one CPU core is enough for a single analysis job. Two GB are empirically identified to be enough for the RAM requirement of a VM which processes an analysis job. The host has an Intel Xeon L5506 2.13GHz CPU with 8 cores and 24 GB of RAM, 1 Gbps full-duplex Ethernet, and the Debian 5 OS with the 2.6.26-2-amd64 kernel, KVM-72. The guests use Debian 5.0.4 with the same kernel version.

Figures 3.6, 3.7, 3.8, 3.9 reflect the results obtained while benchmarking the following four test setups:

1. Eight VMs with one core and 2 GB of RAM each execute in parallel a single job each, on the same given physical host,
2. Eight jobs are executed in parallel in a single 8-core VM with 8 GB of RAM,
3. Eight jobs are executed in parallel in a single 8-core VM with 16 GB of RAM,
4. Eight jobs are natively executed in parallel on the given 8-core physical host.

Figures 3.6 and 3.7 show performance and virtualization overhead for the ALICE analysis jobs, CPU bound trains and I/O bound tasks respectively. Figures 3.8 and 3.9 are there to crosscheck the results of the ALICE jobs on other similar workloads. Figures 3.6b, 3.7b, 3.8b, 3.9b show the overhead of virtualization in percentage to native performance on the physical host.

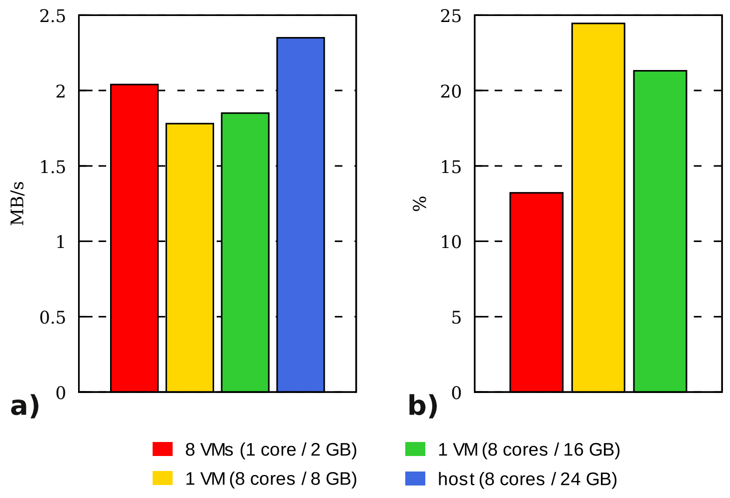

Figure 3.6 a) An average data throughput rate per a train job with 98% CPU utilization varies for different VM configurations. b) Performance overhead of virtualization relative to performance on a physical host is limited to 13% for train jobs each running in its own 1-core VM.

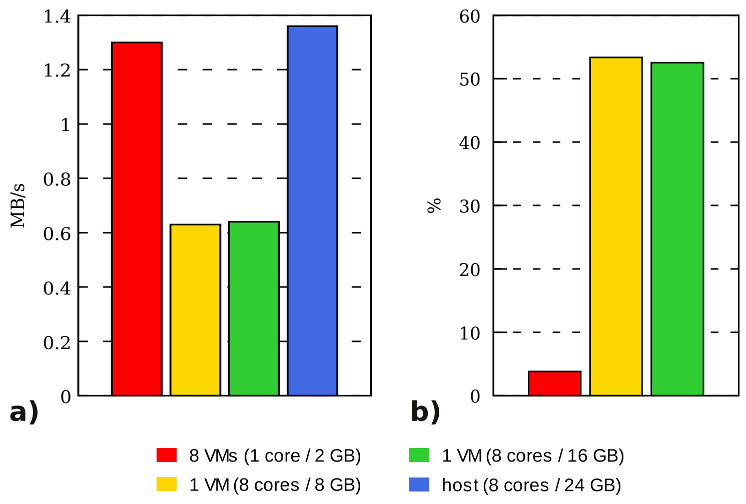

Figure 3.7 a) The difference in average data throughput rate between I/O bound task jobs running in separate 1-core VMs and natively on a physical host is negligible. b) Performance overhead of virtualization relative to performance on a physical host is an order of magnitude higher for task jobs running together in multi-core VMs than in separate VMs.

The HEP-SPEC benchmarking suite has been used to provide a reference for the virtualization overhead values for CPU bound workloads, with 100% CPU utilization. HEP-SPEC is used to evaluate performance of the computing resources of ALICE in particular, and the HEP sector in general. Its 06 version contains all of the C++ benchmarks from the industry-standardized SPEC CPU2006 suite [SPEC]. The exact version used in this thesis is HEP-SPEC06 v1.2. As seen in the Figure 3.8, the KVM hypervisor handles CPU bound workloads equally well for different VM configurations and keeps the overhead at around 16%. On the other hand, the results of running a custom C++ code which reads ESD files from the remote Lustre file system directly into RAM and spends no CPU time in user mode, 0% CPU utilization, provide a reference for the I/O bound workloads (Figure 3.9).

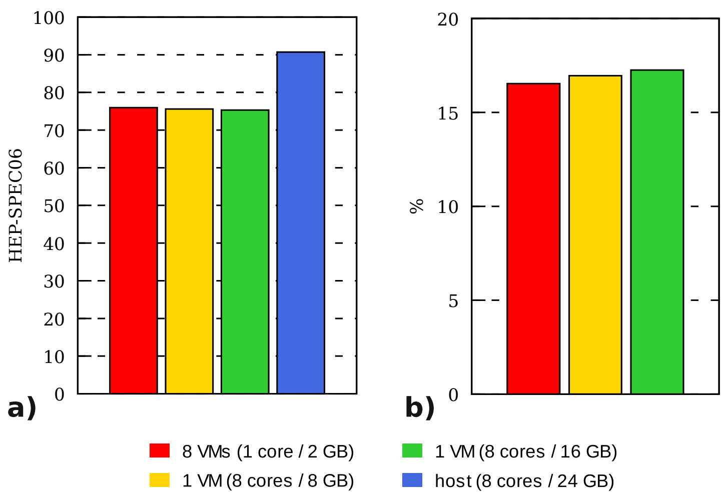

Figure 3.8 a) The KVM hypervisor handles CPU bound workloads, with 100% CPU utilization, equally well for different VM configurations. b) The virtualization overhead values are practically equal for all three setups for a CPU bound HEP-SPEC benchmark.

It is important to mention that CPU bound workloads, such as the HEP-SPEC benchmark and the ALICE analysis train, may include intensive memory management activities, which can induce performance overhead. The virtual memory addresses within the guest OS need to be translated to the specified physical addresses of the host. Besides, the hypervisor must track the changes to the page tables of the VM and control which memory the VM tries to access. These functions may be implemented in software of the hypevisor, with shadow page tables, or in hardware, with Extended Page Tables (EPT) for Intel and with Nested Page Tables (NPT) for AMD processors [Fisher-Ogden]. Consequently, performance of memory management activities inside VMs depends on the hypervisor implementation and virtualization support in hardware.

The results of synthetic benchmarks, such as SysBench [SysBench], which allow to fully load the CPU for large number calculations without significant memory utilization, show almost no performance overhead for purely CPU bound workloads. The fact that the overhead value is considerably lower for such number-crunching benchmarks than for the CPU bound workload of the HEP-SPEC benchmark is attributed to the difference in memory utilization.

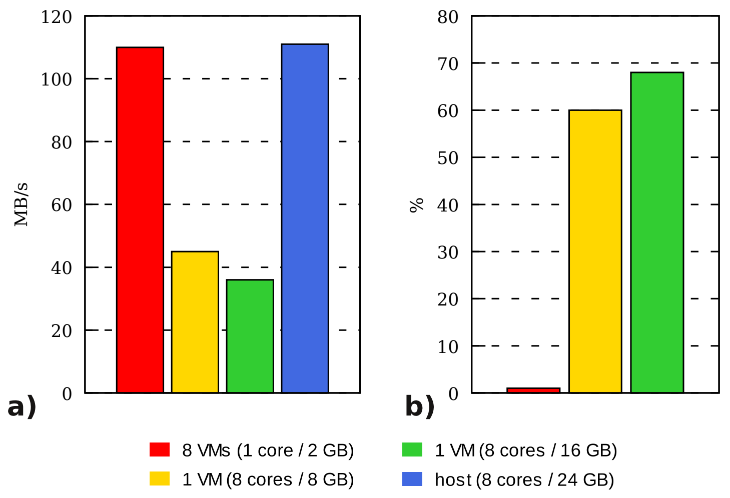

Figure 3.9 a) The data throughput rates for jobs which are purely I/O bound with infinitesimal CPU utilization confirm the results for the data analysis tasks in the Figure 3.7. b) A large overhead for multi-core VMs shows inadequate handling of parallel I/O inside multi-core VMs for the KVM hypevisor.

The conclusions based on the benchmarking results for the given hardware and software versions are summarized in the following paragraphs.

The lowest virtualization overhead for a CPU bound train is just around 13% (Figure 3.6b). Encapsulating jobs in separate single-core VMs avoiding parallel processing inside the multi-core VMs leads to a faster analysis train and task processing. When comparing the overhead values for the analysis jobs (Figure 3.6b) to the values for a CPU bound HEP-SPEC benchmark (Figure 3.8b), it becomes apparent that the virtualization overhead is affected by the workload's I/O utilization. For multi-core VMs, the overhead value increases when I/O activities are introduced into the workload. At the same time, for eight separate single-core VMs, the overhead decreases. This trend continues when the workload becomes even more I/O intensive in the case of analysis tasks (Figure 3.7b) and I/O bound workloads (Figure 3.9b).

Virtualization overhead becomes almost negligible, less than 4%, for I/O bound tasks in separate VMs (Figure 3.7b), whereas the overhead for parallel task processing in multi-core VMs increases more than twofold in comparison to train jobs. The results are confirmed with I/O bound workloads (Figure 3.9). This serves as a demonstration of good handling of the network by the network bridge on the host, good network I/O performance of the KVM hypervisor with the virtio driver for a single-core VM, and poor parallel I/O access within a multi-core VM.

As expected, those operations inside guests which require a trap into the hypervisor slow down the VM performance, increasing virtualization overhead. Changes to memory mapping, adjustment of page tables, cost additional CPU cycles. When more of these operations are required for multi-core VMs, the results of the corresponding benchmarking tests worsen. Memory intensive applications suffer from this the most. Optimization of virtualization support in memory management is a way to address this problem.

3.4 Practical Results for Scientific Computing

To prove the efficiency of the IaC concept, the accumulated techniques have been verified at four different cloud testbeds (Table 3.2). While SCLab and the derived LOEWE-CSC testbed, described below, are similar in architecture and usage aspects and differ only in scale, the other two testbeds have significantly distinctive characteristics.

A virtual cluster of the proposed architecture and assembly process has been deployed at the community cloud of the Frankfurt Cloud Initiative. The so-called Frankfurt Cloud uses VMWare ESX [VMWare] as a VMM, and CloudController [Incontinuum] and VMWare vSphere as cloud middleware, which are all proprietary products. The virtual cluster has been used for the FLUKA [Ferrari] simulation jobs for the “Radiation and Safety” department at FAIR. The FLUKA jobs require Scientific Linux as an OS, which is not provided at GSI, and are perfectly suitable for offloading to external clouds because of their high CPU and negligible I/O utilization.

	SCLab	Frankfurt Cloud	LOEWE-CSC	Amazon EC2
IaaS cloud type	private	community	private	public
Hypervisor technology	KVM	VMware ESX	KVM	Xen
Cloud middleware	OpenNebula	Incontinuum Cloud Controller, VMware vSphere	OpenNebula	EC2
Size of the deployed infrastructure in VMs	120	20	1000	3
API	+	-	+	+
VM image management control	absolute	restricted	absolute	absolute
Network connectivity control	restricted	restricted	restricted	absolute
Practical outcome	an ALICE grid site	radiation simulation for FAIR, nuclear structure calculations	PROOF benchmark	proof of concept

Table 3.2 The four testbeds, where the efficiency of the proposed IaC techniques has been verified.

The public Amazon EC2 cloud provides a paid service and grants the user absolute control of his virtual infrastructure. It is particularly important to note the availability of a flexible firewall configuration mechanism, which allows to open at a VM only necessary network ports. In contrast, at the Frankfurt Cloud, this functionality is hidden from a user, and one has to ask the responsible system administrator for changes. The absence of an API in general singles out the Frankfurt Cloud, where the VM image management capabilities are also restricted for common users and available only for system administrators, whom one would have to notify every time a change is needed.

The described approach to assembly of the virtual clusters has been used to set up at SCLab an AliEn grid site for ALICE. It has been used by the worldwide ALICE community since October 2010 (Figure 3.10), and its operation has been carried out according to the IaC principles. The number of VMs comprising its virtual cluster varies, since SCLab has been serving VMs to multiple users. New Torque worker nodes are created, and failed or old ones are deleted, while the rest of the virtual cluster continues to compute grid jobs. The Torque worker nodes run Scientific Linux 5 [SL], which is the main supported OS for the AliEn grid middleware. This allows to avoid potential dependency problems which are encountered every so often during AliEn installation on the versions of the Debian OS in GSI.

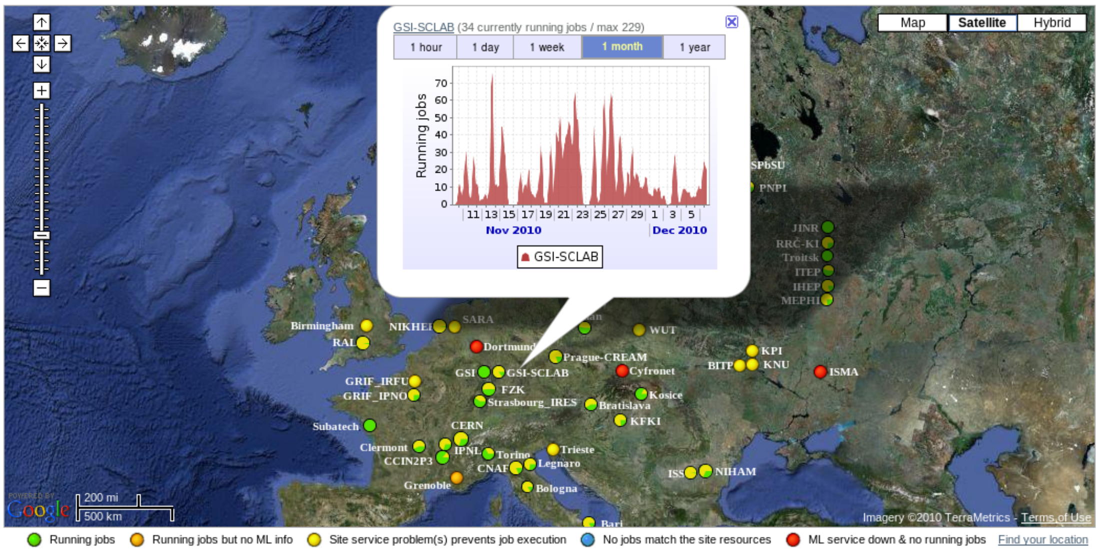

Figure 3.10 A virtual cluster at SCLab being in production for the ALICE grid.

For testing purposes, such virtual cluster has been successfully deployed at the Amazon EC2 with the publicly available OS images of Scientific Linux 5 and Debian 6. First, a Scientific Linux VM has been provisioned in the US region of the cloud, since a base OS image is available only there. Then, the bootstrapped version of the image has been saved to Amazon S3, Amazon's storage cloud. Subsequently, a VM could be started from this image in the European region of EC2.

The Chef server which has been used to configure the VMs with AliEn and Torque has been hosted by Opscode. This allowed to avoid installation of a Chef Server at the commercial cloud. Opscode provides for free a hosted Chef Server for infrastructures of less than 5 nodes.

The presented approach for building virtual clusters on demand, in contrast to the one described in [Keahey] and tuned to the grid environment, allows for utilization of any IaaS cloud, even those without contextualization services, and allows to dynamically change context. Besides, for the use case of the AliEn grid site, in contrast to [Harutyunyan], it allows to use any OS inside generic VM images, which supports Ruby, to use VMs without any preinstalled software on top, and to have total control over their configuration. The presented approach can be used for any workload capable of running in an IaaS environment. For the ALICE computing, it allows to process analysis or simulation tasks without any connection to the grid.

To complete the picture, it is worth mentioning that for an interactive parallel ALICE analysis with PoD no DRMS like Torque is needed. Using the SSH plug-in, PoD is capable of using any resources which run SSH. Certainly, the VMs must have the same software stack installed as on the PoD client. This can be implemented with Chef as described above. So, once VMs are running and remotely accessible by a user, no further contextualization is needed, since no configuration or awareness of any other dynamically addressed system component in the network is required for any particular VM. All what is left to a PoD user, before he can run his analysis in parallel, is to specify in the PoD client a list of hostnames or IP addresses of the running VMs. An opportunity to test this approach on a large scale has occurred during the deployment of a private cloud at the LOEWE-CSC supercomputer [CSC] for the Frankfurt Cloud project.

3.4.1 A PROOF Benchmark at LOEWE-CSC

The Frankfurt Cloud project has set the utilization of a subset of the LOEWE-CSC supercomputer resources as one of its goals. One of the transparent ways to accomplish it is to deploy on a part of the LOEWE-CSC resources a private cloud. The envisioned architecture is presented in the Figure 3.11. Abidance of the IaC principles during the deployment of SCLab and the resulting codebase have allowed to apply the SCLab architecture to a part of LOEWE-CSC.

Fifty out of 800 physical nodes, which make up the supercomputer, have been configured as hypervisors, using KVM. As a result, the private cloud at LOEWE-CSC could draw from the power of more than 1000 CPU cores. By practicing IaC, it is especially convenient to transfer the knowledge and deploy a private cloud similar to SCLab on available resources. Using VMs and isolating the infrastructure within a dedicated VLAN and subnet, it is possible to set up the OpenNebula software for VM orchestration at LOEWE-CSC just by deploying a Chef cookbook as it has been written and used for SCLab at GSI.

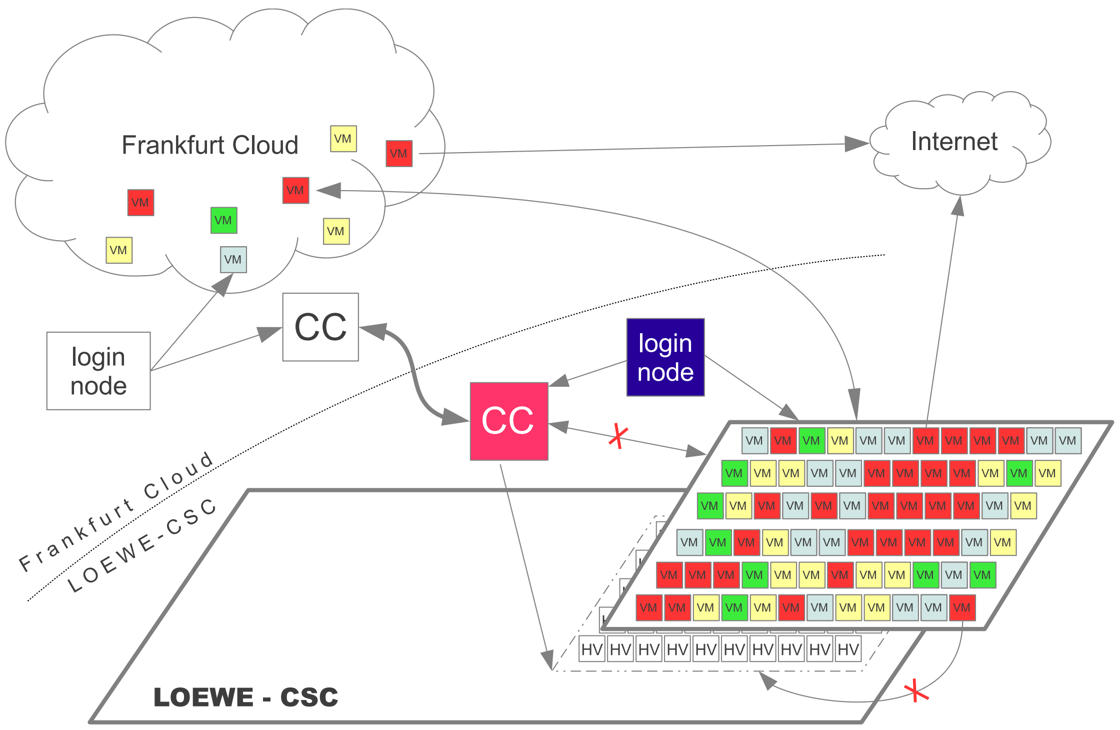

Figure 3.11 The envisioned architecture of the Frankfurt Cloud, incorporating a private cloud of LOEWE-CSC. Virtual machines are colored to show heterogeneity. Both cloud controllers (CC) could possibly schedule VMs between the domains. Crossed links are explicitly disabled. HV stands for hypervisor.

This infrastructure provides a unique opportunity to demonstrate the ability of PoD and its SSH plug-in to roll out a PROOF cluster on a large number of CPU cores at a cloud and the feasibility of a PROOF analysis with several hundreds of workers and a single master. The corresponding tests have set a record for a number of workers in a PROOF cluster. PoD has been used to deploy a cluster with 975 workers, which proves to be a limit for the ROOT version 5.30, since no test has succeeded to surpass this number.

The benchmarking results demonstrate that the scaling of the analysis rate is hampered at the level of 350 worker nodes (Figure 3.12a), and the efficiency of a cluster with more nodes and a single master goes down. The uneven distribution of the processed events among the workers (Figure 3.12b) indicates that the PROOF Packetizer may be at fault. The Packetizer is a PROOF module which schedules the workload between the worker nodes depending on the location of the data. Since some of the workers have received the load too late and have been idle until then, the single-threaded Packetizer might have failed to distribute the workload evenly in time. The PROOF and PoD developers, to whom the results have been reported, are set to eventually pinpoint the cause of this behaviour and get rid of a performance bottleneck.

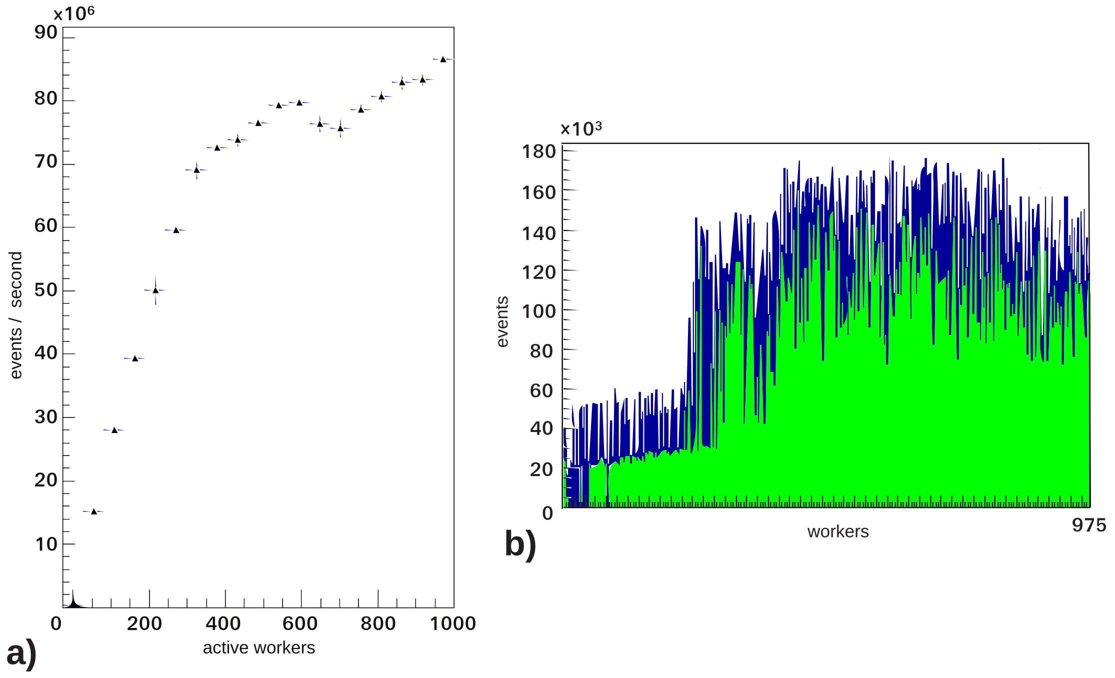

Figure 3.12 An outcome of the PROOF benchmark on a record 975-node PROOF cluster run with PoD on a private cloud at LOEWE-CSC. a) Event analysis rate. Efficiency threshold lies at the level of 350 workers. b) Uneven number of processed events among the workers shows that the single-threaded Packetizer fails to distribute the workload among all the workers in time.

To summarize, the unique results of the PROOF benchmark on a cluster of such a size have been instrumental in driving the improvement of the PROOF software for a distributed interactive ROOT analysis. This demonstrates a synergy of the technologies described in this thesis and serves as another example of the benefits of the IaC practice and the application of cloud computing technology in a scientific research environment.

3.5 Outlook. The Case for ‘Infrastructure as Code’

The greatest advantages of virtualization of computer resources in a scientific environment are the ability it gives users to run custom software stacks in a shared infrastructure, and the ability it gives system administrators to introduce homogeneity to infrastructure software installed on “bare metal” machines and hence operate infrastructure efficiently with higher availability.

Although it remains to be experienced at scale and measured, it is expected that a higher availability of computing resources outweighs the performance penalty of virtualization which for ALICE analysis train jobs amounts to 13% overhead.

An IaaS cloud is the most flexible of all architectures for managing multiple virtual machines. To utilize its flexibility in full and to automate deployment and operation of virtual machines, the IaC concept is proposed.

The benefits of IaC adoption, for both infrastructure providers and users, include easier infrastructure/applications management, faster recovery, higher reliability and availability, easier knowledge exchange, and portability.

Easier management is a result of automating all necessary stages of infrastructure and applications life-cycles with code and encapsulating the resulting code for every purpose into clearly defined units. Once a user specifies which infrastructure component to deploy, this component can be automatically bootstrapped into a correct running state. The codebase dedicated to a specific component must include programming code for at least installation, configuration, and launching on specified platforms. The same goes for user applications.

Faster recovery also stems from automation. If it is possible to query monitoring information for the state of an infrastructure, once a failure is detected, a program can react to it. This benefit of IaC becomes even more significant and precious when an infrastructure runs on virtual instead of physical machines. Virtual machines are easily disposable. So, in case of a failure and in case the dedicated codebase includes all the necessary code for bootstrapping a complete, functioning infrastructure component on top of a given VM image, it is easier just to recycle a VM by deleting it and then installing a new one from scratch.

Another side effect of IaC is a higher reliability of the infrastructure. As with faster recovery, if a program can *regularly* query the infrastructure state, it can recognize a premise for a failure, and take necessary actions to avoid it. Fast automatic recovery and higher reliability lead to a higher availability of the whole infrastructure.

The lack of documentation is often a critical problem for maintenance and operation of an infrastructure and applications. The result of not knowing what to do, and being unable to reach the right person for help is a delay during MTTR or any stage of the infrastructure life-cycle, starting with deployment. A written programming code and, as a good form, comments provided along make up the first source of documentation. Ability to write a generic code for a precise purpose that can be used in other environments, across platforms, by somebody else raises code’s value and fosters knowledge exchange and active collaboration in the community.

The portability of code makes it possible to reprovision infrastructure at the desired scale on any IaaS cloud resources. This allows to use different cloud infrastructure providers to meet peak demands, to ensure reliability, and to avoid lock-in. A change to the code can be avoided at all if migration is carried out to cloud with the same interface.

4 Summary

ALICE is a massively complex high-energy physics experiment with unprecedented demands for computing power and data storage. The data produced by the experiment is distributed through a dedicated grid and processed, as well as stored, all over the world. More than hundred scientific institutions with various computing infrastructures form this heterogeneous international collaboration. Usually, a multi-purpose commodity computer cluster at a scientific research facility taking part in the ALICE grid must serve the users beyond the ALICE community as well. The shared environment imposes restrictions on the ALICE computing infrastructure with its own ad hoc software and distinct requirements. From this perspective, this thesis reflects the bottom-up steps, which should be taken to prepare the ALICE Tier-2 centre's infrastructure for a shared computing environment and maintain its efficiency.

As a first step, it is crucial to understand the nature of the workload, in the case of an ALICE Tier-2 centre, how do analysis jobs utilize the resources. The answer to this question helps to design the infrastructure in a way as to avoid resource contention or underutilization. Initial tests have demonstrated that the level of disk I/O bandwidth utilization for analysis jobs is far lower than what can be sustained by hardware. It has been established that the reason behind this lies in the disk I/O access patterns. The methods to improve the data throughput for analysis jobs have been at the focus of the followed research. The results of the tests, presented in detail in the thesis, conclude that merging the I/O access requests and prefetching data from disk to memory are effective ways to address this problem. Increasing the data files in size improves the performance by a factor of up to 3. After those results and suggested solutions for improvement had been presented at the ALICE Offline Week meeting in October 2009, proposed changes have been implemented in later releases of the analysis code. Besides, this work provides the first picture of how exactly the ALICE analysis jobs access the data at the byte level.

The integration of storage resources is the next step up, and storage issues are addressed in Part 2. A list of requirements compiled for this thesis provides a framework for choosing a particular integration solution, and for an ALICE Tier-2 centre that is a cluster file system. To avoid performance bottlenecks or a single point of failure, a cluster file system has to either support clustering of metadata servers, like Ceph, or not to manage metadata at all, like GlusterFS. The management overhead is too much of a burden for the efficient utilization of the direct-attached storage in a cluster, if a system design does not foresee a mechanism to bring the workload to the data as in systems like Hadoop. PROOF addresses this problem for the ROOT analysis. The presented benchmarks have demonstrated that Scalla is a robust toolkit for efficient access to the ROOT files, but the immaturity of its POSIX interface and issues with file deletion make Lustre, a full-fledged cluster file system with a POSIX interface and fast network operation, preferable to use for diverse data processing tasks.

Conflicting user requirements and incompatible software package dependencies make it impossible to preserve a custom software environment for each user in a shared computing infrastructure. To solve this problem, it becomes necessary to turn to the hardware virtualization technology, which allows for flexible software migration between various hardware infrastructures even across administrative domains. Additionally, virtualization allows for an easier recovery and, thus, a higher availability of a computing infrastructure. Although the benefits come at a cost of a performance overhead, the tests have shown that the ALICE analysis jobs, which fetch data over the network, have it at an acceptable level of 4% for the tasks and 13% for the trains. The presented measurements prove that for a KVM hypervisor a single-core job per single-core VM is a more efficient configuration for parallel I/O intensive processing than multiple single-core jobs per single multi-core VM. With these results laying the foundation, it is possible to take the next step up the infrastructure issues ladder and examine the ways how to set up a large shared virtual infrastructure.

For infrastructures with more than a handful of virtual machines, it becomes crucial to implement efficient VM orchestration techniques. Cloud computing, and particularly its Infrastructure-as-a-Service (IaaS) model, provide an opportunity to do this. An IaaS cloud model, applied in this thesis to a commodity computer cluster hosting an ALICE Tier-2 centre, enables a user-driven on-demand instantiation of virtual machines for running applications sandboxed in a custom user environment. In this model, system administrators operate an infrastructure which runs a homogeneous software stack capable of scheduling and running VMs, and provide users with the VM management utilities. A prototype of an IaaS cloud has been successfully implemented with the OpenNebula cloud toolkit at GSI.

To efficiently handle the increasing complexity of an evergrowing computing infrastructure, it is necessary to automate as many system administration tasks as possible. This thesis introduces the ‘Infrastructure as Code’ concept (IaC) from the commercial world of Web operations to the scientific computing environment to achieve a higher degree of automation. The ambitious idea behind the IaC concept proposes to describe an infrastructure and its applications, including all matters of installation and maintenance, in code, preferably written in a common programming language. An efficient method of assembling virtual clusters for the ALICE analysis according to the IaC principles is demonstrated in this thesis. This work, for the first time, has allowed to process real ALICE grid jobs in a production mode with a completely virtual cluster deployed at an IaaS cloud prototype. It has been presented at the Large Installation System Administration conference in San Jose (USA) in 2010. The power of IaC is demonstrated, again, with a smooth deployment of a private cloud prototype at the LOEWE-CSC supercomputer, which has paved the way for a unique PROOF benchmark. The results of benchmarking a record-breaking 975-node PROOF cluster, deployed exclusively on virtual machines with PROOF-on-Demand (PoD), have highlighted the issues with PROOF workload distribution on a large scale and provided developers with necessary input information for further PROOF optimization.

“Conceptual Design of an ALICE Tier-2 Centre Integrated into a Multi-Purpose Computing Facility” is a broad topic, since the internal organization of a multi-purpose commodity computing facility poses countless research challenges from the lowest to the highest level. Ranging from efficient methods to improve disk access performance of the ALICE analysis tasks to manageable solutions for data storage integration, a number of important challenges are addressed in this thesis. The measured virtualization performance opens new grounds for the ALICE analysis. The IaC techniques presented in this thesis can be used to deploy virtual clusters for batch processing on demand at any IaaS cloud, as has been proven by the examples of private clouds at GSI and LOEWE-CSC, the Frankfurt Cloud, and the public Amazon EC2 cloud.

Zusammenfassung

ALICE ist ein komplexes Hochenergiephysik-Experiment mit bisher einzigartigen Anforderungen an Rechenleistung und Datenspeicher. Die Daten dieses Experiments werden weltweit auf ein spezielles Grid verteilt, verarbeitet und gespeichert. Mehr als hundert wissenschaftliche Institute mit verschiedenen Computing-Infrastrukturen nehmen an dieser heterogenen, internationalen Zusammenarbeit teil. Normalerweise bedient ein Mehrzweck-Computer-Cluster an einem wissenschaftlichen Forschungszentrum viele Benutzergruppen. Diese gemeinsame Umgebung stellt oft Einschränkungen an die ALICE-Computing-Infrastruktur dar, die eigene ad-hoc-Software und spezielle Anforderungen benötigt. Diese Dissertation beschäftigt sich mit der Planung einer effizienten ALICE-Tier-2-Infrastruktur in einer nicht dedizierten Computing-Umgebung.

Der erste Schritt besteht darin, die Nutzung von Rechenressourcen durch Analysejobs in einem ALICE Tier-2 Zentrum genau zu untersuchen. Ausgehend von den Ergebnissen kann die Rechnerinfrastruktur entsprechend gestaltet werden, um Ressourcenengpässe und nichtgenutzte Kapazitäten zu vermeiden. Erste Tests haben gezeigt, dass die tatsächlich durch Analysejobs genutzte Festplattenzugriffsbandbreite viel niedriger ist als die technisch mögliche Obergrenze der Hardware. Es hat sich herausgestellt, dass der Grund dafür ein ungünstiges Festplattenzugriffsmuster war. Ziel der darauffolgenden Forschung war die Verbesserung des Datendurchsatzes für Analysejobs. Die Ergebnisse von Tests, die ausführlich in dieser Arbeit beschrieben sind, zeigen, dass die Bündelung von Zugriffen und das Zwischenspeichern von Festplattendaten im Arbeitsspeicher am effektivsten ist. Eine Erhöhung der Dateigröße verbessert die Leistung um einen Faktor von bis zu drei. Die Ergebnisse und Lösungsvorschläge wurden auf der ALICE Offline Week im Oktober 2009 vorgestellt und in spätere Versionen der Analysesoftware übernommen. Überdies erhielt man das erste Mal ein Bild davon, wie genau ALICE Analysejobs Byte für Byte auf ihre Daten zugreifen.

Der nächste Schritt, welcher in Teil 2 beschrieben wird, ist die Integration von Speicherressourcen. Um die Auswahl eines Cluster-Dateisystem für das ALICE Tier-2 Zentrum zu ermöglichen wurde zunächst ein Anforderungsliste erstellt. Folgende Schlussfolgerungen wurden aus der Arbeit mit Dateisystemen gezogen. Um Performance-Engpässe oder einen “Single Point of Failure” zu vermeiden, muss das Cluster-Dateisystem entweder verteilte Metadatenserver (Ceph) unterstützen oder

gar ganz auf zentrale Metadatenverwaltung verzichten (GlusterFS). Der Mehraufwand an Verwaltung von Direct-Attached Storage in einem Cluster ist zu hoch, wenn es nicht möglich ist, die Analyse-Software in Datennähe auszuführen, etwa in Systemen wie Hadoop. PROOF löst dieses Problem für die ROOT-Analyse. Die vorgestellten Benchmarks haben gezeigt, dass Scalla ein robustes Toolkit ist um effizienten Zugriff auf ROOT-Dateien bereitzustellen. Dennoch kann Scalla aufgrund seiner unausgereiften POSIX-Schnittstelle und wegen Problemen beim Löschen von Dateien nicht mit Lustre mithalten. Seine POSIX-Schnittstelle und die Netzwerk-Performance gehören zu den Stärken von Lustre, und seine Anpassungsfähigkeiten machen Lustre zur sichersten Option für ein Cluster-Dateisystem.

Widersprüchliche Anforderungen der Nutzer und inkompatible Software Paket-Abhängigkeiten machen es unmöglich, eine eigene Software-Umgebung für jeden Benutzer in einer gemeinsam genutzten Computing-Infrastruktur zu erhalten. Um dieses Problem zu lösen, wird es notwendig sein, auf die Hardware-Virtualisierungs-Technologie umzusteigen, die eine flexible Software-Migration zwischen verschiedenen Hardware-Infrastrukturen auch über administrative Domänen hinweg erlaubt. Hinzu kommt, dass Virtualisierung einen einfacheren und schnelleren Wiederherstellungsprozess ermöglicht und somit die Verfügbarkeit einer virtuellen Rechnerinfrastruktur im Vergleich zu einer nativen erhöht. Allerdings werden die genannten Vorteile mit einem Performance-Verlust erkauft. Die Tests haben gezeigt, dass sich ALICE-Analysejobs, die Daten über das Netzwerk holen, auf einem akzeptablen Niveau von 4% Performance-Verlust für die Task-Jobs und 13% für die Train-Jobs bewegen. Die vorgestellten Messungen zeigen, dass für einen KVM-Hypervisor ein Single-Core-Job pro Single-Core-VM für parallele I/O-intensive Verarbeitung eine effizientere Konfiguration ist als mehrere Single-Core-Jobs für eine einzige Multi-Core-VM. Mit diesem Ergebnis als Grundlage war es nun möglich, einen Schritt im Infrastrukturdesign weiter zu gehen und Wege zu finden, wie man eine große, gemeinsam genutzte virtuelle Infrastruktur aufbauen kann.

Für Infrastrukturen mit mehr als einer Hand voll virtueller Maschinen ist es von entscheidender Bedeutung Methoden für eine effiziente VM Orchestrierung zu implementieren. Cloud Computing und insbesondere das Infrastructure-as-a-Service (IaaS) Modell lösen dieses Problem. Das IaaS Cloud-Modell, angewandt auf ein ALICE Tier-2 Zentrum, bedeutet eine benutzer- und bedarfsgetriebene Instanziierung von virtuellen Maschinen (VM), um Applikationen abgeschlossen in einer speziellen Benutzerumgebung zu betreiben. In diesem Modell stellen

Systemadministratoren eine Reihe von Softwarediensten zur Erstellung und Einteilung von VMs zur Verfügung und bieten den Nutzern die Instrumente zum Management ihrer VMs an. Es wurde ein Prototyp des IaaS Cloud-Modells in der GSI implementiert.

Um effizient die zunehmende Komplexität der ständig wachsenden Computing-Infrastruktur betreiben zu können, ist es notwendig, möglichst viele Aufgaben der Systemadministration zu automatisieren. Diese Arbeit nutzt das “Infrastructure-as-Code”-Konzept (IaC) aus der Welt des Web-Operating in der wissenschaftlichen Computing-Umgebung, um einen höheren Grad an Automatisierung zu erreichen. Die treibende Idee hinter diesem Konzept besteht in einer vollständigen Beschreibung sowohl der Infrastruktur und der Applikation, als auch der Installation und Wartung in Programmcode, möglichst in einer einheitlichen Sprache. Ein effizientes Verfahren zum Aufbau eines virtuellen Clusters für die ALICE-Analyse nach IaC-Grundsätzen wird in dieser Arbeit demonstriert. Es wurde bei der Large Installation System Administration Konferenz im Jahr 2010 in San Jose (USA) vorgestellt. Diese Arbeit hat es das erste Mal ermöglicht, echte ALICE Grid Jobs produktiv auf einem vollvirtualisierten Cluster, der in einer Prototyp-IaaS-Cloud eingerichtet wurde, zu rechnen. Die Macht des IaC-Konzepts wird außerdem mit dem erfolgreichen Einsatz eines Private-Cloud-Prototyps auf dem LOEWE-CSC nachgewiesen, der den Weg für einen in seiner Größe einzigartigen Benchmark geebnet hat. Die Ergebnisse des Benchmarks, erstellt auf einem 975-Knoten-PROOF-Cluster, ausschließlich auf virtuellen Maschinen mit PROOF-on-Demand (POD), haben die Probleme mit der PROOF Lastverteilung im großen Maßstab beleuchtet und haben den Entwicklern den notwendigen Input für die weitere Optimierung zur Verfügung gestellt.

“Conceptual Design of an ALICE Tier-2 Centre Integrated into a Multi-Purpose Computing Facility” ist ein weitläufiges Thema, denn aus der inneren Struktur einer Mehrzweck-Rechnerinfrastruktur ergeben sich auf allen Ebenen viele technische Fragen. Die Themen erstrecken sich von effizienten Methoden, um die Leistung des Festplattenzugriffs der ALICE-Analyse Jobs zu verbessern, bis hin zu handfesten Lösungen für die Datenspeicherung. In dieser Arbeit werden in diesem Zusammenhang eine Reihe schwieriger Herausforderungen gelöst. Die gemessene Leistung der Virtualisierung eröffnet neue Wege für die ALICE-Analyse. Die IaC-Methoden, die virtuelle Cluster für die Stapelverarbeitung auf Abruf bereitstellen und die in dieser Arbeit vorgestellt sind, können bei jeder IaaS Cloud genutzt werden. Das Konzept hat sich in privaten Clouds an der GSI und dem LOEWE-CSC, der Frankfurt Cloud, und der öffentlichen Amazon EC2 Cloud bewährt.

Acknowledgements

Foremost, I would like to thank my Mom and Dad for their care and wisdom, always when it matters most.

Second, I would like to thank Victor Penso for being such an influential IT guru and a helpful and serious colleague.

My gratitude goes to Dr. Peter Malzacher for continuous support during my PhD studies and his wise advices.

Prof. Dr. Volker Lindenstruth has given me this opportunity to challenge myself, and I am grateful to him for it.

Dennis Klein and Bastian Neuburger have helped me on numerous occasions and have shown me that IT is a source of fun and a passion. Dr. Kilian Schwarz had played a crucial role in bringing me for my PhD studies to GSI and has been supportive ever since. My direct colleagues at the “Grid Group” of GSI, such as Almudena Montiel, Anna Kreshuk, Anar Manafov, and Carsten Preuss, have always been ready to help. Thanks to the rest of the GSI IT too.

Dr. Silvia Masciocchi has graciously given a tour of the ALICE train.

Dr. Jan de Cuveland and Dr. Timm M. Steinbeck have provided me with precious comments on the thesis.

I would like to admit the essential role of the Bogolyubov Institute for Theoretical Physics in Kiev, whose scientists have introduced me to the world of ALICE, grid computing, and high-energy physics in general.

A thank you goes also to all my friends here and there.

Among other sources, I have drawn inspiration from the talents of U2 and Prof. Dr. Richard Dawkins.

Finally, this thesis would not be possible without those, who develop free and open-source software, and those, who use and promote it, sharing their experience on the Web, - thank you so much.

List of Figures

List of Tables

List of Acronyms

ALICE A Large Ion Collider Experiment
AliEn ALICE Environment
AOD Analysis Object Data
API Application Programming Interface
CAF CERN Analysis Facility
CERN European Organization for Nuclear Research in Geneva, Switzerland
CernVM CERN Virtual Machine
CFQ Complete Fair Queuing
CHEP Computing in High-Energy Physics
CMS Configuration Management Software
CNS Composite Name Space
CPU Central Processing Unit
DAS Direct-Attached Storage
DHCP Dynamic Host Configuration Protocol
DLM Distributed Lock Manager
DNS Domain Name System
DRMS Distributed Resource Management System
DSL Domain-Specific Language
EC2 Amazon Elastic Compute Cloud
ESD Event Summary Data
FAI Fully Automatic Installation
FAIR Facility for Antiproton and Ion Research
FLUKA Fluktuirende Kaskade
FUSE Filesystem in Userspace
GFS2 Global File System 2
GPL GNU General Public License
GPU Graphics Processing Unit
GSI GSI Helmholtz Centre for Heavy Ion Research in Darmstadt, Germany
GSIAF GSI Analysis Facility
HDD Hard Disk Drive
HDFS Hadoop Distributed File System
HEP High-Energy Physics
HPC High-Performance Computing
I/O Input/Output
IaaS Infrastructure as a Service
IaC Infrastructure as Code
IBM International Business Machines
IP Internet Protocol
IPv4 Internet Protocol version 4
IT Information Technology
JBOD Just a Bunch Of Disks
KVM Kernel-based Virtual Machine
LAN Local Area Network
LGPL GNU Lesser General Public License
LHC Large Hadron Collider
MAC Media Access Control
MC Monte Carlo
MDS Metadata Server
MTBF Mean Time Between Failures
MTTD Mean Time To Diagnose
MTTF Mean Time To Failure
MTTR Mean Time To Repair
NAS Network-Attached Storage

NAT Network Address Translation
NOOP No Operation Performed
OCCI Open Cloud Computing Interface
OCFS2 Oracle Cluster File System 2
OS Operating System
PC Personal Computer
PoD PROOF on Demand
POSIX Portable Operating System Interface for Unix
PROOF Parallel ROOT Facility
PVFS Parallel Virtual File System
PXE Pre-boot eXecution Environment
RAID Redundant Array of Inexpensive/Independent Disks
RAM Random-Access Memory
RC2 IBM Research Compute Cloud
SAN Storage Area Network
SCLab Scientific Computing Lab
SLO Service Level Objectives
SOA Service-Oriented Architecture
SPEC Standard Performance Evaluation Corporation
SSD Solid-State Drive
SSH Secure Shell
SSI Simple Server Inventory
TFTP Trivial File Transfer Protocol
VM Virtual Machine
VMM Virtual Machine Monitor
VNC Virtual Network Computing
WAN Wide Area Network
XCFS Xrootd Cluster File System

References

[Abbott]: Abbott M.L, Fisher M.T., “The Art of Scalability”. p. 433. Pearson Education. 2010.

[ALICE]: http://aliceinfo.cern.ch, last visited on 17.04.2011.

[ALICE Computing]: “ALICE Technical Design Report: Computing”, ALICE TDR 012 (15 June 2005), CERN-LHCC-2005-018.

[Availability]: “IEEE Std 610.12-1990”.

[Borthakur]: Borthakur D., “The Hadoop Distributed File System: Architecture and Design”, http://hadoop.apache.org/common/docs/r0.16.4/hdfs_design.html, last visited on 17.04.2011.

[Braam]: Braam P.J., “The Lustre storage architecture”, Cluster File Systems, Inc., August 2004. http://www.lustre.org/documentation.html, last visited on 17.04.2011.

[Bradshaw]: Bradshaw R., Desai N., Freeman T., and Keahey K., “A Scalable Approach to Deploying and Managing Virtual Appliances”, In TeraGrid 2007 Conference. 2007. Madison, WI.

[Brun]: Brun R., Rademakers F., “ROOT - An Object Oriented Data Analysis Framework”, Proceedings AIHENP’96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in Phys. Res. A 389 (1997) 81-86. http://root.cern.ch, last visited on 17.04.2011.

[Buncic]: Buncic P. et al., “CernVM - a virtual appliance for LHC applications”. Proceedings of the XII. International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT08), Geneva, 2008.

[Burgess]: Burgess M., “A Site Configuration Engine”, USENIX Computing Systems, Vol. 8, Num. 2, pp. 309-337, 1995.

[Cantrill]: Cantrill B.M., Shapiro M.W., Leventhal A.H., “Dynamic instrumentation of production systems”, Proceedings of USENIX ’04, 2004.

[Capistrano]: http://www.capify.org, last visited on 17.04.2011.

[Carns]: Carns P.H., Ligon III W.B., Ross R.B., Thakur R., “PVFS : A parallel file system for linux clusters”, In Proceedings of the 4th Annual Linux Showcase and Conference, pages 317–327,. Atlanta, GA, October 2000. USENIX Association.

[Carvajal]: Carvajal J.M.C., Schwemmer R., Garnier J.-C., Neufeld N., “A high-performance storage system for the LHCb experiment”, Proceedings of the Real Time Conference, 2009. RT ‘09. 16th IEEE-NPSS.

[CernVM]: http://cernvm.cern.ch, last visited on 17.04.2011.

[Chef]: http://www.opscode.com/chef, last visited on 17.04.2011.

[Chierici]: Chierici A., Veraldi R., “A quantitative comparison between xen and kvm”, Journal of Physics: Conference Series, 2010.

[Ciliendo]: Ciliendo E., Kunimasa T., Braswell B., “Linux Performance and Tuning Guidelines”, http://www.redbooks.ibm.com/abstracts/redp4285.html, last visited on 17.04.2011.

[CloudStore]: http://kosmosfs.sourceforge.net, last visited on 17.04.2011.

[CSC]: http://compeng.uni-frankfurt.de/index.php?id=86, last visited on 17.10.2011

[Dean]: Dean J., Ghemawat S., “Mapreduce: simplified data processing on large clusters”, Commun. ACM, 51(1):107–113, 2008.

[Debian]: http://www.debian.org, last visited on 13.05.2011.

[Debois]: Debois P., “Monitoring”, Allspaw J., Robbins J., “Web operations. Keeping the Data on Time”, Chapter 6, p. 82-85, O’Reilly Media 2010.

[Delaet]: Delaet T., Joosen W., Vanbrabant B., "A survey of system configuration tools", In Proceedings of the 24th Large Installations Systems Administration (LISA) conference, San Jose, CA, USA, 11/2010 2010. Usenix Association.

[Deshane]: Deshane T., Shepherd Z., Matthews J., Ben-Yehuda M., Shah A., Rao B. "Quantitative comparison of xen and kvm" Proceedings of the Xen Summit, Boston, MA, USA,June 2008, pp. 1–2. USENIX Association, June 2008.

[Desnoyers]: Desnoyers M., Dagenais M.R., "The LTTng tracer: A low impact performance and behavior monitor for GNU/Linux", Proceedings of the Ottawa Linux Symposium 2006, 2006.

[Domingo]: Domingo D., "Red Hat Enterprise Linux 5 IO Tuning Guide", https://www.redhatrenewals.com/docs/wp/performancetuning/iotuning/index.html, last visited on 17.04.2011.

[EC2]: http://docs.amazonwebservices.com/AWSEC2/latest/APIReference, last visited on 17.04.2011.

[FAIR]: http://www.gsi.de/fair, last visited on 17.04.2011.

[Fasheh]: Fasheh M., "OCFS2: The Oracle Clustered File System, Version 2", Proceedings of the 2006 Linux Symposium, pp. 289-302., 2006.

[FCI]: http://www.frankfurt-cloud.com, last visited on 17.04.2011.

[Ferrari]: Ferrari A., Fassò A., Ranft J., Sala P.R., "FLUKA: a multi-particle transport code", CERN-2005-10 (2005), INFN/TC_05/11, SLAC-R-773.

[Fisher-Ogden]: Fisher-Ogden J., "Hardware Support for Efficient Virtualization", PhD thesis, University of California, San Diego 2006.

[Foster]: Keahey K., Foster I., Freeman T., and Zhang X., "Virtual Workspaces: Achieving Quality of Service and Quality of Life in the Grid. Scientific Programming Journal, vol 13, No. 4, 2005, Special Issue: Dynamic Grids and Worldwide Computing, pp. 265-276.

[Franco]: Brun R., Franco L., Rademakers F., "Efficient Access to Remote Data in High Energy Physics", http://indico.cern.ch/contributionDisplay.py?contribId=284 &sessionId=31&confId=3580, last visited on 17.04.2011.

[Ganis]: Ganis G., Iwaszkiewicz J., Rademakers F., "Data Analysis with PROOF", Proceedings of ACAT 2008 Conference. PoS(ACAT08)007.

[GFS2]: "Red Hat Global File System 2. Edition 7." http://linux.web.cern.ch/linux/scientific5/docs/rhel/Global_File_System_2/ch-overview-GFS2.html, last visited on 17.04.2011.

[git]: http://git-scm.com, last visited on 17.04.2011.

[GlusterFS]: "Gluster File System Architecture" Whitepaper, 2009, http://www.gluster.com/products/gluster-file-system-architecture-white-paper, last visited on 17.04.2011.

[Grosse-Oetringhaus]: Grosse-Oetringhaus J.F., "The CERN Analysis Facility — A PROOF Cluster for Day-one Physics Analysis", J. Phys.: Conf. Ser. 119:072017 (2008).

[GSI]: http://www.gsi.de, last visited on 17.04.2011.

[Gu]: Gu Y., Grossman R.L., "Sector and Sphere: the design and implementation of a high-performance data cloud", Phil. Trans. R. Soc. A 367(1897): 2429-45, 2009.

[Hanushevsky]: "Scalla/xrootd 2009 Developments, presentation by Andrew Hanushevsky at CERN", 2009, http://xrootd.slac.stanford.edu, last visited on 17.04.2011.

[Harutyunyan]: Harutyunyan A., Buncic P., Freeman T., Keahey K., "Dynamic Virtual AliEn Grid Sites on Nimbus with CernVM", Journal of Physics: Conference Series, Volume 219, Issue 7, pp. 072036 (2010).

[hdparm]: http://hdparm.sourceforge.net, last visited on 17.04.2011.

[HEPiX]: "HEPiX Storage Presentations 2005", w3.hepix.org/storage/hepix.php?y=2005, last visited on 27.11.2010.

[Hetzler]: Hetzler S.R., "The storage chasm: Implications for the future of HDD and solid state storage", December 2008, http://www.idema.org, http://www.caiss.org/docs/DinnerSeminar/TheStorageChasm20090205.pdf, last visited on 17.04.2011.

[Incontinuum]: http://www.incontinuum.com, last visited on 17.04.2011.

[iostat]: http://www.kernel.org/doc/Documentation/iostats.txt, last visited on 17.04.2011.

[Jacob]: Jacob A., "Infrastructure as Code", Allspaw. J., Robbins J., "Web operations. Keeping the Data on Time", Chapter 5., p. 65-80. O'Reilly Media 2010.

[Jones]: Jones M.T., "Anatomy of the Linux File System. A layered structure-based review.", IBM DeveloperWorks, 30 October 2007. http://www.ibm.com/developerworks/linux/library/l-linux-filesystem/?S_TACT=105AGX03&S_CMP=ART, last visited on 17.04.2011.

[Kalos]: Kalos M., Whitlock M., "Monte Carlo Methods" 2nd edn. Wiley VCH, Weinheim, 2008.

[Keahey]: Keahey K., Freeman T., "Contextualization: Providing one-click virtual clusters", In eScience 2008, 2008.

[Kozierok]: Kozierok C.M., "Hard Disk Drives", The PC Guide, 2004, http://www.pcguide.com/ref/hdd/index.htm, last visited on 17.04.2011.

[KVM]: http://www.linux-kvm.org, last visited on 17.04.2011.

[Lagar-Cavilla]: Lagar-Cavilla H. A., Tolia N., Satyanarayanan M., De Lara E., "Vmm-independent graphics acceleration", In proceedings of VEE (2007), pp. 33–43.

[Lange]: Lange T., "Fully automatic installation of debian gnu/linux", Nov 2001, Linux Kongress, http://fai-project.org, last visited on 17.04.2011.

[libvirt]: http://libvirt.org, last visited on 17.04.2011.

[Liu]: Liu J., Huang W., Abali B., Panda D., "High Performance VMM-Bypass I/O in Virtual Machines", Proceedings of the USENIX 2006 Annual Technical Conference, 2006.

[Lustre manual]: http://wiki.lustre.org/manual/LustreManual20_HTML/SettingUpLustreSystem.html, last visited on 14.04.2011.

[Lustre size]: http://wiki.lustre.org/index.php/FAQ_-_Sizing, last visited on 27.11.2010.

[Maltzahn]: Maltzahn et al., "Ceph as a scalable alternative to the Hadoop Distributed File System", USENIX ;login magazine, vol. 35, no. 4 , August 2010.

[Malzacher]: Malzacher P., Manafov A., "PROOF on Demand", 2010 Journal of Physics: Conf. Ser. 219 072009.

[Masciocchi]: Masciocchi S., "Performance Tests on the GSI Batch Farm + Lustre", 2009. https://indico.gsi.de/getFile.py/access?contribId=10 &resId=0&materialId=slides&confId=450, last vistited on 27.11.2010.

[Maslennikov]: Maslennikov A., "Progress report 4.2010. HEPiX Storage Working Group", http://indico.cern.ch/getFile.py/access?contribId=11 &sessionId=3&resId=1&materialId=slides&confId=92498, last vistited on 17.04.2011.

[Massie]: Massie M.L., Chun B.N., and Culler D.E., "The ganglia distributed monitoring system: design, implementation, and experience." Parallel Computing, 30(7), July 2004.

[Mell]: Mell P., Grance T., "The NIST Definition of Cloud Computing", Version 15, 10-7-09. http://csrc.nist.gov/groups/SNS/cloud-computing, last visited on 17.04.2011.

[Merino]: Merino G. et al., "Transition to a new CPU benchmarking unit for the WLCG", http://indico.cern.ch/getFile.py/access?contribId=3&sessionId=0&resId=0&materialId=0&confId=49388, last visited on 20.05.2011.

[Molina-Estolano]: Molina-Estolano E., Maltzahn C., Brandt S., Gokhale M., May J., Bent J., "Mixing Hadoop and HPC Workloads on Parallel Filesystems", Proceedings of 4th Petascale Data Storage Workshop, Supercomputing '09.

[MooseFS]: http://www.moosefs.com, last visited on 17.04.2011.

[Mushran]: Mushran S., “OCFS2. A Cluster File System for Linux. User’s Guide for Release 1.4”, http://oss.oracle.com/projects/ocfs2/documentation, last visited on 17.04.2011.

[Narayanan]: Narayanan D., Thereska E., Donnely A., Elnikety S., Rowstron A., “Migrating server storage to SSDs: analysis of tradeoffs”, Proceedings of the 4th ACM European conference on Computer systems EuroSys’09, 2009.

[nice]: http://pubs.opengroup.org/onlinepubs/009695399/utilities/nice.html, last visited on 17.04.2011.

[Nurmi]: Nurmi D., Wolski R., Grzegorczyk C., Obertelli G., Soman S., Youseff L., Zagorodnov D., “The Eucalyptus Open-Source Cloud-Computing System”, Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, p.124-131, May 18-21, 2009.

[OCCI]: http://www.occi-wg.org, last visited on 17.04.2011.

[Offline Bible]: http://aliweb.cern.ch/secure/Offline/sites/aliweb.cern.ch.Offline/files/uploads/OfflineBible.pdf, last visited on 17.04.2011.

[Oleynik]: Oleynik G., “Storage system evaluation criteria”, http://cd-docdb.fnal.gov/cgi-bin/ShowDocument?docid=2576, last visited on 17.04.2011.

[Openstack]: http://www.openstack.org, last visited on 17.04.2011.

[Opscode]: http://www.opscode.com, last visited on 17.04.2011.

[Peters]: Peters A.J., “XCFS - An Analysis Disk Pool & Filesystem Based On FUSE And Xroot Protocol”, Proceedings of ACAT 2008 Conference. PoS(ACAT08)041.

[POSIX]: “Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API)”, ISO/IEC 9945-1, 1996.

[Prasad]: Prasad V., Cohen W., Eigler F.C., Hunt M., Keniston J., Chen B., “Locating system problems using dynamic instrumentation”, Ottawa Linux Symposium 2005, 2005.

[PXE]: “Preboot Execution Environment (PXE) Specification”, Intel, 1999 ftp://download.intel.com/design/archives/wfm/downloads/pxespec.pdf, last visited on 17.04.2011.

[ROOT user’s guide]: http://root.cern.ch/drupal/content/users-guide, last visited on 17.04.2011.

[Roselli]: Roselli D., Lorch J., and Anderson T., “A comparison of file system workloads”, In Proceedings of the 2000 USENIX Annual Technical Conference, pages 41–54, San Diego, CA, June 2000. USENIX Association.

[Russel]: Russel R., “virtio: Towards a De-Facto Standard For Virtual I/O Devices”, SIGOPS Oper. Syst. Rev., Vol. 42, No. 5. (July 2008), pp. 95-103.

[Ryu]: Ryu K.D. et al., “RC2 – A Living Lab for Cloud Computing”, Proceedings of LISA’10, USENIX. 2010.

[Saiz]: Saiz P. et al., “AliEn - ALICE Environment on the Grid”, Nucl. Instrum. Methods A502 (2003) 437-440; http://alien.cern.ch, last visited on 13.05.2011.

[Scalla]: “Scalla/xrootd. Presentation by Andrew Hanushevsky at OSG Storage Forum”, 2009. http://xrootd.slac.stanford.edu, last visited on 17.04.2011.

[Schwarz]: Schwarz K., “GSIAF, CAF experience at GSI”, PROOF 2007 Workshop CERN, http://indico.cern.ch/materialDisplay.py?contribId=19
&sessionId=5&materialId=slides&confId=23243, last visited on 17.04.2011.

[Sempolinski]: Sempolinski P., Thain D., “A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus”, IEEE International Conference on Cloud Computing Technology and Science, November, 2010.

[SL]: https://www.scientificlinux.org, last visited on 13.05.2011.

[Sollins]: Sollins K.R., “The TFTP Protocol (Revision 2)”, July 1992, IETF. RFC 1350. http://tools.ietf.org/html/rfc1350, last visited on 17.04.2011.

[Sotomayor]: Sotomayor B., Montero R.S., Llorente I.M., Foster I., “Virtual Infrastructure Management in Private and Hybrid Clouds”, IEEE Internet Computing, vol. 13, no. 5, pp. 14-22, Sep./ Oct. 2009.

[SPEC]: http://www.spec.org/cpu2006, last visited on 10.02.2012.

[Staples]: Staples G., “TORQUE resource manager”, SC ‘06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing, ISBN 0-7695-2700-0.

[strace]: http://www.linuxmanpages.com/man1/strace.1.php, last visited on 17.04.2011.

[SysBench]: http://sysbench.sourceforge.net, last visited on 17.04.2011.

[Tate]: Tate J., Lucchese F., Moore R., “Introduction to Storage Area Networks”, 2006, http://www.redbooks.ibm.com/abstracts/sg245470.html, last visited on 17.04.2011.

[Traeger]: Traeger A., Zadok E., Joukov N., and Wright C. P., “A nine year study of file system and storage benchmarking”, ACM Trans. Storage 4, 2, Article 5 (May 2008), 56 pages. DOI = 10.1145/ 1367829.1367831.

[VMWare]: http://www.vmware.com/support/pubs/vs_pages/vsp_pubs_esx41_vc41.html, last visited on 17.04.2011.

[Wang]: Wang F., Xin Q., Hong B., Brandt S.A., Miller E.L., Long D.D.E., and McLarty T.T., “File system workload analysis for large scale scientific computing applications”, In Proceedings of the 21st IEEE / 12th NASA Goddard Conference on Mass Storage Systems and Technologies, pages 139–152, College Park, MD, April 2004.

[Watts]: Watts D. et al., “Tuning IBM System x Servers for Performance”, http://www.redbooks.ibm.com/abstracts/sg245287.html, last visited on 17.04.2011.

[Weil]: Weil S., Brandt S.A., Miller E.L., Long D.D.E., Maltzahn C., “Ceph: A Scalable, High-Performance Distributed File System”, Proceedings of the 7th Conference on Operating Systems Design and Implementation (OSDI ‘06), November 2006.

[Welch]: Welch B., “POSIX IO extensions for HPC”, In Proceedings of the 4th USENIX Conference on File and Storage Technologies (FAST), December 2005. www.pdl.cmu.edu/posix, last visited on 17.04.2011.

[Whitehouse]: Whitehouse S., “The GFS2 Filesystem”, Proceedings of the Linux Symposium. Ottawa. 2007.

[WLCG]: http://lcg.web.cern.ch/LCG/resources.htm, last visited on 20.05.2011.

[Wu]: Wu Y. et al., “Utilizing Lustre file system with dCache for CMS analysis”, 2010 J. Phys.: Conf. Ser. 219 062068.

[XCFS]: https://svnweb.cern.ch/trac/CERNXROOT/wiki, last visited on 17.04.2011.

[Yang]: Yang W., “Introduction to XrootdFS”, http://wt2.slac.stanford.edu/xrootdfs/xrootdfs.html, last visited on 17.04.2011.

[Zynovyev]: Zynovyev M. “GlusterFS benchmarks”, 2009, https://indico.gsi.de/getFile.py/ access?contribId=6 &resId=0&materialId=slides&confId=450, last visited on 27.11.2010.

Curriculum Vitae

Personal information

First name / Surname	**Mykhaylo Zynovyev**
Also known as	Misha, Mike, or Michael
Place and date of birth	Kiev, Ukraine, 24/06/1984
Nationality	Ukraine
Gender	male
Place of residence	Frankfurt am Main, Germany

Occupational field — **Computer Science / IT Engineering**

Work experience

Dates	April 2007 – July 2012
Occupation or position held	PhD student
Main activities and responsibilities	postgraduate research on data analysis and design of a Tier-2 centre for the ALICE experiment at LHC
Name and address of employer	GSI Helmholtz Centre for Heavy Ion Research, Planckstr. 1, 64291 Darmstadt, Germany
Dates	2005 – March 2007
Occupation or position held	IT-engineer
Main activities and responsibilities	administration of grid-sites (AliEn and ARC middleware)
Name and address of employer	Bogolyubov Institute for Theoretical Physics, National Academy of Sciences of Ukraine, Metrologicheskaya street 14-B, 03143 Kiev, Ukraine
Dates	May 2005 – June 2006
Occupation or position held	programmer
Main activities and responsibilities	development of a C language compiler with eMAC support for the Freescale Coldfire microprocessor
Name and address of employer	Freescale Embedded Systems Lab, Information Software Systems Ltd, Bozhenko Str. 15, Kiev, Ukraine
Dates	January 2004 – April 2005
Occupation or position held	programmer
Main activities and responsibilities	analysis of C language compilers and implementation of optimization techniques using MAC/eMAC units for the Freescale Coldfire microprocessor
Name and address of employer	Freescale Embedded Systems Lab, National Technical University of Ukraine "Kiev Polytechnic Institute", Prospect Peremohy 37, 03056 Kiev, Ukraine
Dates	November 2002 – December 2004
Occupation or position held	programmer

Main activities and responsibilities	development of a wrapper for handling exceptions of a FPU of a MPC5xx microcontroller, and benchmarking
Name and address of employer	Motorola Embedded Systems Lab, National Technical University of Ukraine "Kiev Polytechnic Institute", Prospect Peremohy 37, 03056 Kiev, Ukraine

Education

Dates	2011 - 2012
Title of qualification sought	doctorate studies in Computer Science under supervision of Prof. Dr. Volker Lindenstruth
Name of organisation providing education	the Faculty of Computer Science and Mathematics, the Goethe University Frankfurt, Frankfurt am Main, Germany
Dates	2008 - 2011
Title of qualification sought	doctorate studies in Computer Science under supervision of Prof. Dr. Volker Lindenstruth
Name of organisation providing education	the Faculty of Mathematics and Computer Science, Ruprecht-Karls University of Heidelberg, Heidelberg, Germany
Dates	2004 - 2006
Title of qualification awarded	MSc in Computer Engineering, specialization in Computer Systems and Networks
Thesis title	The Means of Job Management in a Distributed Computing Environment
Name of organisation providing education	the Department of Specialized Computer Systems, the Faculty of Applied Mathematics, National Technical University of Ukraine "Kiev Polytechnic Institute", Kiev, Ukraine
Dates	2000 - 2004
Title of qualification awarded	BSc in Computer Engineering
Name of organisation providing education	the Department of Specialized Computer Systems, the Faculty of Applied Mathematics, National Technical University of Ukraine "Kiev Polytechnic Institute", Kiev, Ukraine

Training

Dates	7-12 November 2010
Description	training at LISA '10: 24th Large Installation System Administration Conference, San Jose, USA
Principal subjects/occupational skills covered	networking, Linux security and administration, storage
Name of organisation providing training	USENIX: The Advanced Computing Systems Association, Berkeley, USA
Dates	20-31 August 2007
Description	the CERN School of Computing, Dubrovnik, Croatia
Principal subjects/occupational skills covered	grid technologies, software technologies, physics computing
Name of organisation providing training	CERN, the European Organization for Nuclear Research, in collaboration with the University of Split, Croatia
Dates	September 2006
Description	training at ALICE in CERN, Geneva, Switzerland
Principal subjects/occupational skills covered	administration of the AliEn grid middleware
Name of organisation providing training	the Offline group, the ALICE experiment at CERN, European Organisation for Nuclear Research, Geneva, Switzerland

Dates	27 February - 3 March 2006
Description	International School "Grid Administration and Experiment Data Processing in ALICE", Dubna, Russia
Principal subjects/occupational skills covered	administration of the gLite grid middleware
Name of organisation providing training	JINR, the Joint Institute For Nuclear Research, Dubna, Russia
Dates	July, August 2005
Description	the CERN summer student program
Principal subjects/occupational skills covered	benchmarking in Linux, Tcl programming
Name of organisation providing training	the DAQ group, the ALICE experiment at CERN, European Organisation for Nuclear Research, Geneva, Switzerland
Dates	September 2004
Description	training at INFN
Principal subjects/occupational skills covered	LabVIEW programming, controlling thermal infra-red video camera
Name of organisation providing training	the ITS group, the ALICE experiment, INFN Sezione di Torino, Torino, Italy

Fellowships

Name (period)	Helmholtz Graduate School for Hadron and Ion Research "HGS-HIRe for FAIR" (2009 - 2012)
Name (period)	GSI Helmholtz Centre for Heavy Ion Research (2007 - 2008, 2012)
Name (period)	Freescale Inc. (2004 - 2006)
Name (period)	Motorola Inc. (2002 - 2004)

Conferences

Dates, location	2-6 May 2011, Darmstadt, Germany
Name of the attended conference	HEPiX Spring 2011
Contribution	Talk “Adopting 'Infrastructure as Code' to Run HEP Applications”
Dates, location	7-12 November 2010, San Jose, USA
Name of the attended conference	LISA '10: 24th Large Installation System Administration Conference
Contribution	Poster “Adopting 'Infrastructure as Code' in GSI”
Dates, location	21-22 June 2010, Geneva, Switzerland
Name of the attended conference	2nd Workshop on Adapting Applications and Computing Services to Multi-core and Virtualization
Dates, location	13-18 October 2006, Sinaia, Romania
Name of the attended conference	International ICFA Workshop on Grid Activities within Large Scale International Collaborations

Publications

Title	“Utilization of LOEWE-CSC within the Frankfurt Cloud”
List of authors	M. Zynovyev, D. Klein, A. Manafov, V. Penso
Periodical title and date of publication	GSI Scientific Report 2011, GSI Report 2012-01

Title	“Radiation Simulation and Nuclear Structure Calculation at Frankfurt Cloud”
List of authors	D. Klein, P. Malzacher, V. Penso, M. Zynovyev
Periodical title and date of publication	GSI Scientific Report 2010, GSI Report 2011-01

Title	“Cluster-Virtualization at GSI”
List of authors	D. Klein, P. Malzacher, V. Penso, M. Zynovyev
Periodical title and date of publication	GSI Scientific Report 2010, GSI Report 2011-01

Title	“Grid Activities at GSI”
List of authors	M. Al-Turany, M. Dahlinger, S. Daraszewicz, P. Malzacher, A.Manafov, A. Montiel Gonzalez, V. Penso, C. Preuss, K. Schwarz, F. Uhlig, M. Zynovyev
Periodical title and date of publication	GSI Scientific Report 2009, GSI Report 2010-01

Title	“Grid Activities at GSI”
List of authors	M. Al-Turany, A. Kreshuk, P. Malzacher, A. Manafov, V. Penso, C. Preuss, K. Schwarz, F. Uhlig, M. Zynovyev
Periodical title and date of publication	GSI Scientific Report 2008, GSI Report 2009-01

Title	“Grid Activities at GSI”
List of authors	K. Schwarz, P. Malzacher, A. Manafov, V. Penso, C. Preuss, M. Zynovyev, A. Kreshuk
Periodical title and date of publication	GSI Scientific Report 2007, GSI Report 2008-01

Title	“Ukrainian Grid Infrastructure: Practical Experience”
List of authors	M. Zynovyev, S. Svistunov, O. Sudakov, Y. Boyko
Periodical title and date of publication	Proceedings of 4th IEEE Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, 2007. IDAACS 2007

listed as co-author on multiple papers of the ALICE Collaboration (2009-2012)
http://aliweb.cern.ch/Documents/generalpublications

<u>Languages</u>

Mother tongue	**Russian**

Other languages

Self-assessment	**Understanding**				**Speaking**				**Writing**	
European level (*)	Listening		Reading		Spoken interaction		Spoken production			
English	C1	Proficient User	C1	Proficient User	C1	Proficient User	C1	Proficient User	C2	Proficient User
Ukrainian	C2	Proficient User	C2	Proficient User	B2	Independent User	C1	Proficient User	C1	Proficient User
German	B1	Independent User	B1	Independent User	B1	Independent User	B1	Independent User	A2	Basic User

(*) Common European Framework of Reference for Languages