ALEXANDER REINEFELD AND VOLKER LINDENSTRUTH[1]

How to Build a High-Performance Compute Cluster for the Grid[2]

[1] Universität Heidelberg, Kirchhoff Institut für Physik
[2] Invited paper for the 2nd Intl. Workshop on Metacomputing Systems and Applications (MSA'2001), Valencia, 2001.

ZIB-Report 01-11 (May 2001)

How to Build a High-Performance Compute Cluster for the Grid

Alexander Reinefeld
Konrad-Zuse-Zentrum für Informationstechnik Berlin

Volker Lindenstruth
Universität Heidelberg, Kirchhoff Institut für Physik

Abstract

The success of large-scale multi-national projects like the forthcoming analysis of the LHC particle collision data at CERN relies to a great extent on the ability to efficiently utilize computing and data-storage resources at geographically distributed sites. Currently, much effort is spent on the design of Grid management software (Datagrid, Globus, etc.), while the effective integration of computing nodes has been largely neglected up to now. This is the focus of our work.

We present a framework for a high-performance cluster that can be used as a reliable computing node in the Grid. We outline the cluster architecture, the management of distributed data and the seamless integration of the cluster into the Grid environment.

Keywords: Grid computing, resource management, cluster architecture, distributed data-intensive applications.

Contents

1 Introduction

Two paradigms are changing the way we do computing: Clusters and Grids. Both have been born by the need for a more economical means for high-performance computing: Clusters employ cost-effective commodity components for building powerful computers, and Grids provide a means for the better exploitation of computing resources that are interconnected via wide area networks like the Internet.

Much work has been spent on the design and implementation of middleware for computational Grids, resulting in software packages like Globus, Cactus, Legion, WebFlow, NetSolve, Ninf, Nimrod-G, Condor-G, and others. All these software packages have been successfully deployed and some of them are already used in the daily work—at least in the academic community.

Once the idea of Grid computing [5] grows beyond that of an academic experiment, the demand for computing cycles may by far outgrow the supply. It will then be no longer feasible to provide computer cycles on costly supercomputers in the Grid. Low-cost commodity clusters will become an attractive alternative—not only for high-throughput computing, but also for running parallel high-performance applications.

We present a framework for a low-cost, high-performance cluster that can be used as a reliable compute node in the Grid. The impetus for this work came from the EU project *Datagrid*, where we are collaborating on the design of Grid-aware clusters.

Besides being focused on the seamless integration of clusters into Grids, our work also concentrates on the management of large data sets, including such important topics as caching, replication, and synchronization within the bounds of the cluster and also in the 'outside' Grid environment. We target at an integrated design that takes both environments, local and global, into account. With the cluster architecture, we pursue three major goals:

- *low total cost of ownership* by providing improved fault tolerance and resilience as compared to conventional Beowulf clusters [10],
- *high degree of scalability* between 10^1 and 10^3 nodes,
- *maximum performance* for parallel applications by better exploiting commodity interconnects.

Building on the cluster as a compute node, we present middleware for the proper integration of clusters in Grid environments. Here, the focus is on the specification and management of distributed resources.

2 LHC Requirements

The *Large Hadron Collider (LHC)*, a high energy physics collider experiment is currently being prepared at the European research institution CERN. There are four LHC experiments planned (ALICE, ATLAS, CMS, and LHCb), which all have in common that a large number of particles is tracked and reconstructed, resulting in very high data streams [1]. During operation, the data stream of digitized data will exceed some TeraByte/s. This data stream is reduced by a complex hierarchy of data selection, filtering and compression systems to a few PetaBytes per year and experiment. This so-called *RAW data* is archived to permanent storage and further analyzed subsequently.

Fig. 1 shows an outline of the resulting data paths for the ATLAS experiment as an example. The data is processed in various steps producing a variety of data sets. This processing is iterated with an increasing rate as the end of the chain is reached. The first step resembles the calibration of the experiment and reconstruction of the detected particles from the measured or simulated RAW data, creating the *Event Summary Data (ESD)*. This step is computationally intensive as it requires multi-dimensional image processing. Assuming the steady continuation of Moore's law on the increase of processor speed, the processing of one such RAW event will take about 3.6 seconds on a fast computer in 2005 when the LHC experiments go online, while the simulation of one such event is 3 to 5 times slower. The ESD is further reduced to *Analysis Object Data (AOD)*, which is used as a basis for the physics analysis. In order to allow fast searches for particular scenarios, there is a small tag summarizing each event. From those AODs *Derived Physics Data (DPD)* is produced. These data sets are processed several times a day per user with different algorithms.

Aside from the data hierarchy, Fig. 1 shows the estimated number of processors[1] required in 2005 for each processing step and its aggregated data flow. Note that the estimated total computing power for the LHC experiments in 2005 exceeds 50,000 processors!

The large computational requirements for the first processing step (RAW $\rightarrow$ ESD) and the reduced event size results in an aggregate data stream of about 10% of any other stage in the analysis chain, roughly corresponding to an OC-12 link for ATLAS, which is state-of-the-art today.

Considering the total amount of processors required and the moderate

[1]based on today's benchmarks and assuming that Moore's law will hold true for the next years.

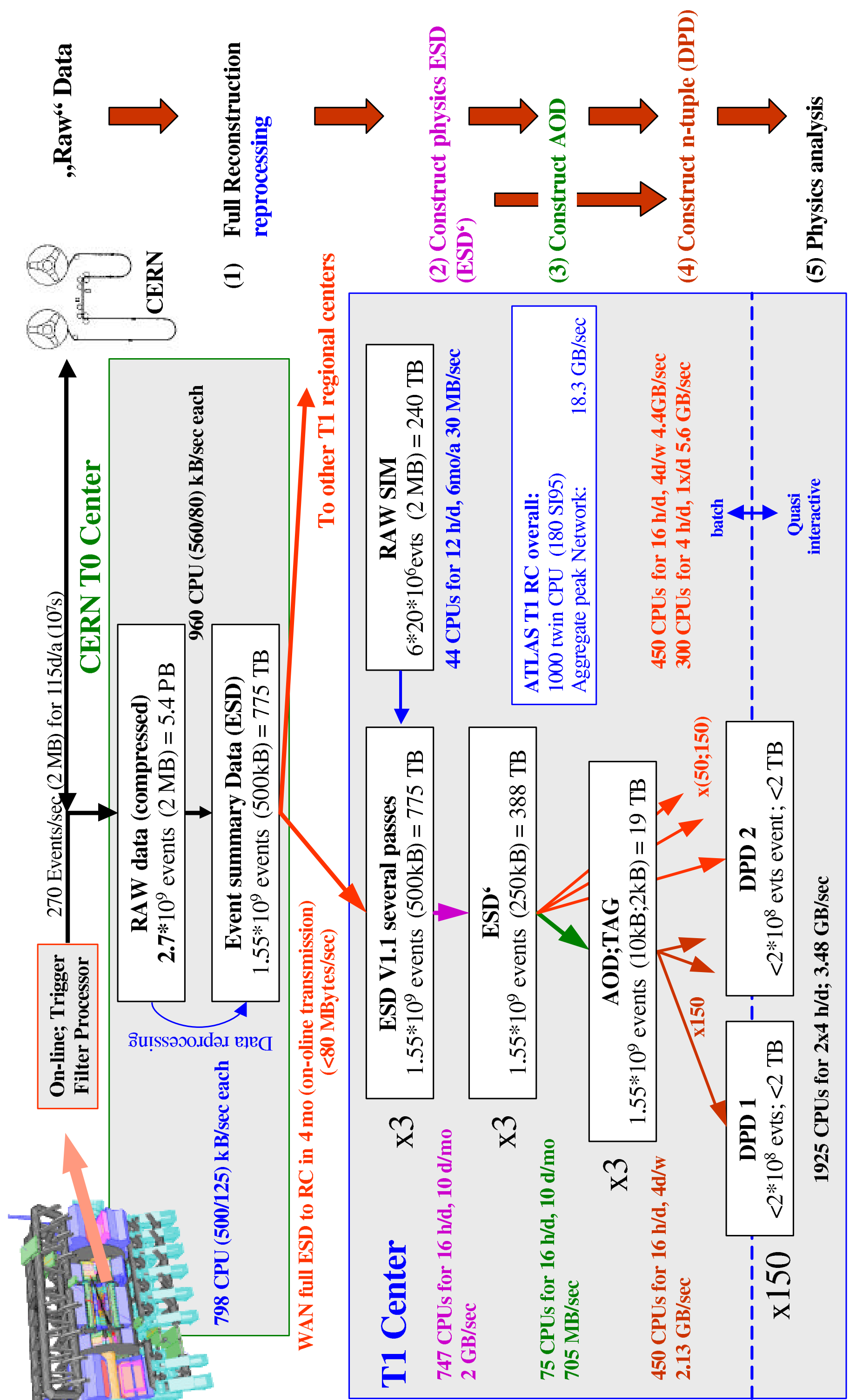
„Raw“ Data
(1) Full Reconstruction
reprocessing
(2) Construct physics ESD (ESD‘)
(3) Construct AOD
(4) Construct n-tuple (DPD)
(5) Physics analysis
CERN
On-line; Trigger Filter Processor
270 Events/sec (2 MB) for 115d/a (10⁷s)
CERN T0 Center
RAW data (compressed)
2.7*10⁹ events (2 MB) = 5.4 PB
Event summary Data (ESD)
1.55*10⁹ events (500kB) = 775 TB
960 CPU (560/80) kB/sec each
Data reprocessing
798 CPU (500/125) kB/sec each
WAN full ESD to RC in 4 mo (on-oline transmission)
(<80 MBytes/sec)
To other T1 regional centers
T1 Center
RAW SIM
6*20*10⁶evts (2 MB) = 240 TB
44 CPUs for 12 h/d, 6mo/a 30 MB/sec
ESD V1.1 several passes
1.55*10⁹ events (500kB) = 775 TB
x3
747 CPUs for 16 h/d, 10 d/mo
2 GB/sec
ESD‘
1.55*10⁹ events (250kB) = 388 TB
x3
75 CPUs for 16 h/d, 10 d/mo
705 MB/sec
ATLAS T1 RC overall:
1000 twin CPU (180 SI95)
Aggregate peak Network: 18.3 GB/sec
AOD;TAG
1.55*10⁹ events (10kB;2kB) = 19 TB
x3
450 CPUs for 16 h/d, 4d/w
2.13 GB/sec
450 CPUs for 16 h/d, 4d/w 4.4GB/sec
300 CPUs for 4 h/d, 1x/d 5.6 GB/sec
x(50;150)
x150
batch
Quasi interactive
DPD 1
<2*10⁸ evts; <2 TB
DPD 2
<2*10⁸ evts event; <2 TB
x150
1925 CPUs for 2x4 h/d; 3.48 GB/sec

Figure 1: Sketch of the data flow hierarchy for the example of the ATLAS experiment at CERN

data stream after the first processing step, the Europe-wide distribution of the LHC computing hierarchy seems feasible. The RAW data is produced, archived and processed at CERN, which also hosts the largest compute farm, the *Tier 0* (T0) center. The output is transmitted to three to five *Tier 1* (T1) centers per experiment, which are distributed throughout Europe.

Given the size of the various data sets ranging up to almost one PetaByte it becomes obvious that the required resource management has to expand from available computing resources to include the mass storage. The jobs must be submitted to the systems holding the required data, rather than sending the data to some available processors. For example, taking the smallest data sample, the DPD, and assuming an OC-12 link, the DPD data transmission time of about 7 hours exceeds the allowable processing time of 4 hours. The total size of the data sets drives up the size of the computing farms.

From the above it is clear that LHC computing will fail without the availability of Grid-enabled, high-performance, large-scale compute clusters. This aspect alone sufficiently explains the motivation for the here described research—aside from the many other application domains that are also expected to benefit.

In addition to the cluster architecture, the required resource management becomes particularly complex as the amount of different data objects and their versions to be taken into account exceeds by far the number and type of resources typically handled by existing resource management systems (see Sec. 5).

3 Outline of a Cluster Architecture

While for the high-throughput computing demands of the LHC experiments simple compute farms with a large number of powerful compute nodes may suffice, this approach is not feasible for conducting online analysis under quasi real-time conditions. Here, a fast system area network, more powerful computing nodes, and, most important, a cluster management system with effective real-time scheduling mechanisms are of prime importance. The following sections describe these components in more detail.

3.1 Computing Nodes

In former years, there was a clear distinction between low- and high-end processors. This is no longer true with the new processors from Intel (1.7 GHz Pentium 4 and IA64-Itanium), AMD (Athlon, 1,3 GHz) and Compaq

(Alpha 21264, 833 MHz) which are all targeted at the mass market. Today, the major difference lies in the choice of the chipset and the memory subsystem. The 4-fold discrepancy in memory bandwidth between the low-end SDRAM, the intermediate-level DDR-RAM, and the high-end Rambus memory is paralleled by a 4-fold increase in cost. As a result, a low-end dual Intel Pentium 3 node[2] node may sell at a moderate cost of \$1200, while a fully integrated Intel Pentium 4 system[3] with Rambus memory may cost \$4700. Which one to chose depends on the budget and the application characteristics.

Contrasting the wide range of different hardware platforms, there is only one choice of operating system: Linux. With its current release 2.4.x, Linux also supports SMP nodes with up to 16 processors. Experiments in using Microsoft Windows do not seem to be successful, which might be attributed to the less flexible system management and to the significantly higher cost for operating software licenses.

In summary there is a large variety of hardware options for the computing nodes to choose from in order to match the application requirements for memory size, bandwidth and architecture, cache architecture and size, and the type of processor. But given the wide support of PCI(-X) bus standard in basically all compute nodes this choice does not affect the choice of network architecture at all.

3.2 Interconnects and Protocols

Nowadays, also a wide variety of fast system area networks is offered: Myrinet, SCI, Atoll, Gigabit Ethernet, Giganet (cLAN), QsNet, ServerNet, Fibre Channel, and GSN. Which one to chose depends on the type of application and on the available budget.

For communication-intensive and latency sensitive, parallel applications the new SCI [9] or Myrinet [8] adapter cards with 64 bit wide, 66 MHz fast PCI bus connectors may be a good choice. Both SANs deliver a bandwidth in excess of 250 MB/s at a latency of less than 5 μs. The cost per node including a proportional number of switches, cables, software etc. is at about \$1800, which is, however, comparable to the price of a complete low-end compute node.

To keep the cost low, we used a Gigabit Ethernet (GbE) network in our cluster installation. However, it was clear that we had to adapt the maximum packet size (and later also the protocol itself), because the standard

[2]Tyan dual Intel P3, 800 MHz, 500 MB SDRAM, 45 GB disk

[3]Dell dual Intel Xeon, 1.7 GHz, 1 GB RDRAM, 40 GB disk

Ethernet MTU of 1500 byte, which is also used in GbE, is not adequate for the high transmission speed of GbE. In GbE, 1500 byte frames may cause interrupt rates of up to 100 kHz. Even with the non-standard jumbo frames of 9000 byte and interrupt coalescing, the CPU has to cope with interrupt rates beyond 10 kHz, yielding an almost 100% CPU load on an 800 MHz Intel P3, as illustrated by the '+' symbols in Fig. 2. Even so, TCP gives a throughput of almost 90 MByte/s at a relatively small packet size of 500 bytes (left curve in Fig. 2). It has to be stressed that this processing overhead does not scale with Moore's law as it is highly I/O bound.

The right curve in Fig. 2 shows the performance of the *Scheduled Transfer Protocol (STP)* [11] on GbE. STP is an ANSI-specified connection-oriented data transfer protocol that supports flow-controlled `read` and `write` sequences and non-flow controlled, persistent-memory `put`, `get` and `fetch` sequences. For all sequences, small control messages are used to pre-allocate buffers at the receiver side, so that the message transfer can thereafter be done at full speed directly from the physical network into the memory at the receiver side. Hence, the initial latency is higher, but once the buffer is established the maximum bandwidth of the network is fully utilized. Fig. 2 shows preliminary experimental results of STP on GbE. We expect the performance to improve when our `libst` library becomes available that supports zero-copy memory transfers under Linux.

Note the less than 5% CPU time overhead of STP which has to be compared with 60 to 100% in case of TCP/IP. The time spent in the network protocol limits the scalability of a compute cluster and therefore is an essential parameter to keep low. Assuming an overhead of 50% at 90 MB/sec, a total of 200 additional nodes will be required for a tier 1 regional center as sketched in Fig. 1.

Another advantage in using GbE as a cluster network is its universal applicability for SANs, LANs and WANs. We tested TCP on GbE over a 2 x 500 km WAN distance between Berlin and Nuremberg using two SGI Onyx2 (IRIX 6.5.4). With the standard MTU size we obtained a throughput of 45 MB/s and with jumbo frames 85 MB/s. Again, the frame rate (interrupts/sec) is the bottleneck. The latency of 5.7 ms corresponds to the speed of light within the fiber. We achieved a very promising zero packet error rate over 90 hours of stress test (sending 10^{10} frames).

3.3 Fault Tolerance

In very large scale clusters, the canonical trend to increase each component's reliability such that the overall system retains a high uptime cannot be

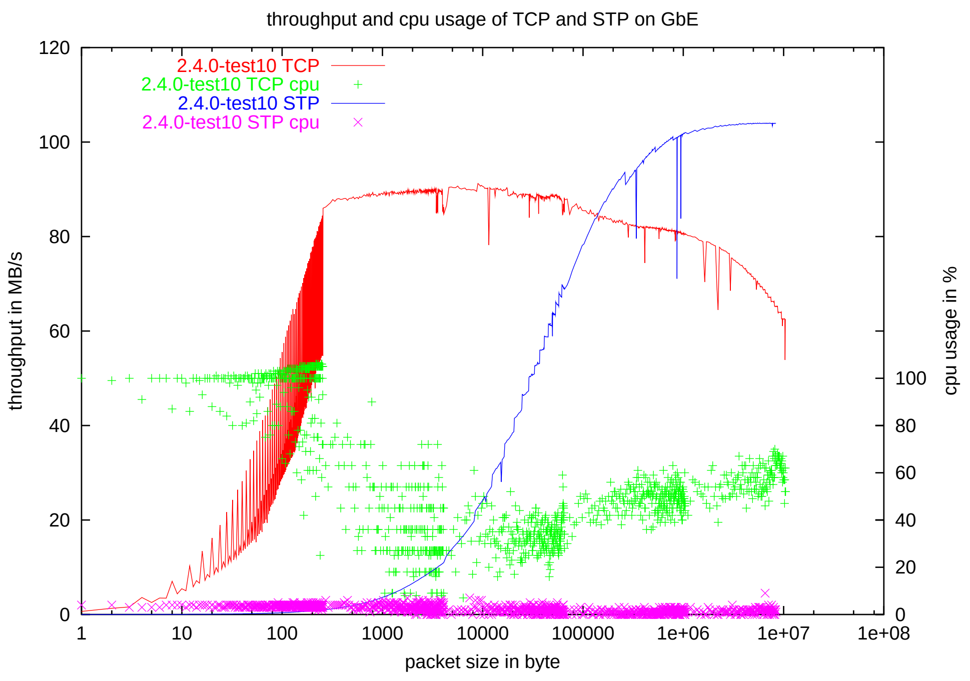

Figure 2: Bandwidth (left axis) and CPU time usage (right axis) for TCP (left graph) and STP (right graph) on Gigabit Ethernet—both using jumbo frames

applied, because the expected failure rate increases with the product of the number of components. Here, any single point of failure must be eliminated in the design and appropriate monitoring functions must be established to provide a status of the components and some means for remote maintenance and control.

We are currently implementing a fault tolerance system with a variety of sensor and actuator functions, basically allowing to measure any status within the system and to perform any function except physical replacement of components. This is accomplished by the design of a micro controller-based PCI device, which has access to all basic interfaces of the compute node and is operated through a private control network. This device senses the health status of the node and is capable of operating all functions up to including re-setting and power cycling, thus providing ultimate, standardized control over a given node. The use of PCI as the baseline bus guarantees the applicability of the device to almost any of today's computers. This device is capable of autonomously performing basic maintenance and repair functions.

However given the complexity of a large-scale compute cluster the variety of failures and race conditions requires more sophisticated repair functionality. Therefore, aside from logging any state changes to a monitoring data base, a hierarchical fault tolerance and repair server is being designed, that is able to handle any complex error conditions and instruct the fault tolerance agents in the front-end node.

Given the availability of a fault tolerance system, which enables the cluster to cope with failure rather than trying to avoid failure with additional costly hardware, the per-node-cost can be held as low as possible.

4 Management of Distributed Data

Another important aspect to be considered in the context of computing clusters is their mass storage subsystem. During the last decade the areal density of hard disks has increased by 60% p.a., allowing to attach a TeraByte of low cost disk capacity to any compute node using two or three hard disk drives during the next years. This scenario would result in a PetaByte of on-line disk capacity of a tier 1 center as sketched in Fig. 1, obviating the need for any mass storage system. The availability of significant online mass storage at the incremental cost of mass market hard disks has to be compared with the appropriate network attached disk servers or HPSS systems. Therefore a paradigm shift is to be expected here, similar to the shift from super computers towards compute farms.

The individual hard disk data transfer rate improves with 40% p.a. while the average access time based on rotation and head movement increases by 8% p.a. with typical ratings to date at 30 MB/s data transfer rate and 7 ms average access time. Those figures are independent of the storage architecture. However the improving raw throughput of the device also drives towards distributed architectures as already a 4x RAID L0 striping would result in an aggregate data stream of 120 MB/s, saturating a GbE link.

However, there is one fundamental issue related to a networked storage architecture as compared to a network attached storage: the reliability of the overall system. If a node is lost due to failure, the cluster looses an appropriate amount of compute power, but all online data remains unaffected. For a networked storage architecture, means to protect against data loss must be implemented. Cluster-local RAID systems, which are supported by the Linux kernel, are not feasible because they do not protect against the loss of a whole node. However the principle of RAID can well be adopted to a cluster.

The failure model of a networked mass storage system is rather simple as the error checking and recovery of the hard disk devices themselves is standard today and performed by the disk itself and any operating system locally. Therefore the data itself does not have to be checked for errors. Recovery mechanisms have to be provided only for the case of a node not being able to deliver a data block at all. In that case it has to be recreated from appropriate redundant data.

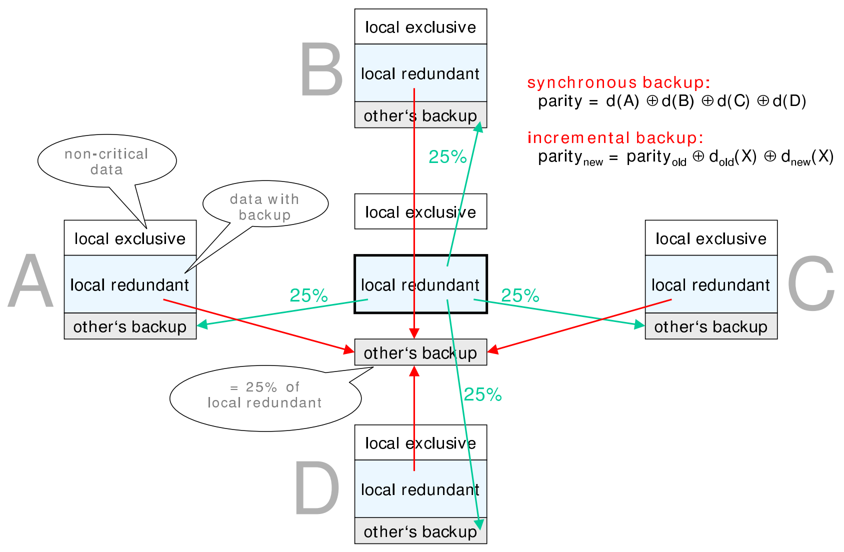

Figure 3: Sketch of a distributed cluster mass storage system

One possible scenario is outlined in Fig. 3. The concept is similar to RAID level 5. Here a parity block is generated for a group of four distributed data blocks. Consequently the loss of any node allows to recreate the appropriate data by using the remaining three valid blocks plus the parity block. In order to recover the irrecoverable loss of a node with 1 TeraByte of local storage to a new node, configured from scratch, a total of 4 times 1 TeraByte of data have to be moved across the cluster interconnect. Assuming 50% utilization of a GigaBit network, the transfer of a TeraByte corresponds to 5.6 hours. However the required XOR transaction on the four data blocks can be performed pair wise, allowing to overlap some of the operations on different nodes.

However, the affected data is not off-line during this recovery time, which can be performed in the background. Any data block can be recreated immediately on demand. For example recovering a 64 kB data block on any node in the cluster results in an additional latency of about 2 ms caused by the data transfer, which is small as compared to the average disk access time. Intelligent network interfaces can be used here to perform the parity operation without burdening the host CPU in order to further the efficiency of the recovery methodology.

The discussed scenario requires a data overhead of 25% for the parity blocks. However any level of redundancy and overhead can be implemented, trading the data overhead versus the tolerance with respect to multiple simultaneous failures and recovery network load and latency.

On the other hand not every data item has to be stored in a redundant fashion. For example page and swap, replica copies, local scratch space can be made available directly.

It should be noted that the presented scenario effectively implements a linear coding scheme, thus allowing reads to be served directly by the nodes owning the data. In case of writes only the parity block is required to be updated as sketched in Fig. 3, here, however, requiring the transmission of both the new and old data block for updating the parity block.

5 Integrating Clusters into the Grid: The Resource Management Problem

In Grid environments, the great variety of supported resources (computers, networks, instruments, files, archives, software) and their dynamic behavior poses a number of resource management problems:

- *Diverse local management systems and policies:* Various incompatible local management systems and policies must be supported.
- *Resilience:* The Grid management software must be able to cope with the changing availability and performance of components.
- *Co-allocation:* Multi-site applications (i.e., jobs running concurrently at different sites) must be supported by suitable co-allocation schemes.
- *Remote steering:* The execution of interactive applications must be supported by suitable monitoring information and online control.

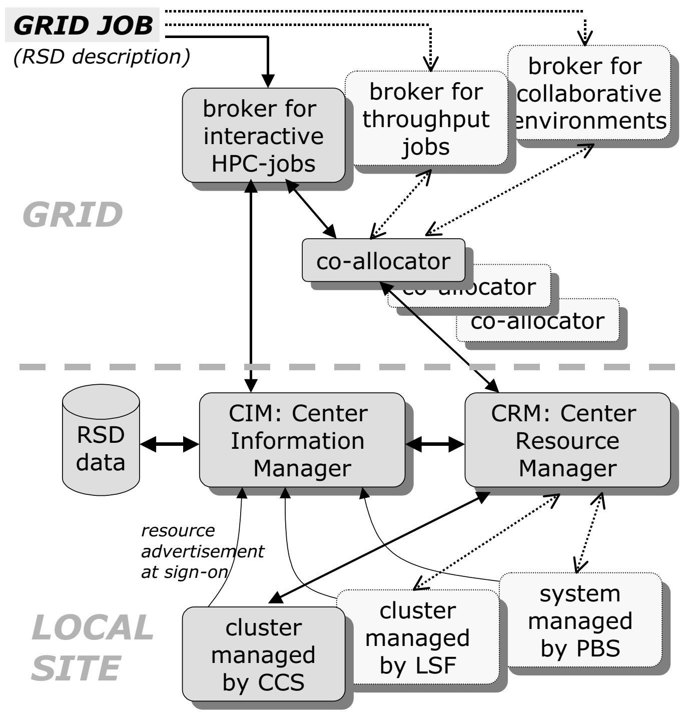
GRID JOB
(RSD description)
broker for interactive HPC-jobs
broker for throughput jobs
broker for collaborative environments
GRID
co-allocator
co-allocator
co-allocator
RSD data
CIM: Center Information Manager
CRM: Center Resource Manager
resource advertisement at sign-on
LOCAL SITE
cluster managed by CCS
cluster managed by LSF
system managed by PBS

Figure 4: Integration of the local cluster management system (lower part) into the Grid environment (upper part)

To deal with these problems, Grid environments are organized in hierarchical layers. In the following, we first outline the basic architecture of the Grid layer and thereafter the tools used in the local sites.

5.1 Resource Management at the Grid Level

Fig. 4 illustrates the framework of our Grid architecture. A Grid job, shown in the top part of the figure, is sent to a resource broker which searches for suitable resources. Note, that there may exist several brokers, each of them specialized on certain types of applications: e.g., one broker for handling interactive high-performance jobs, another for parameterized high-throughput jobs, etc. Resource requests are specified in RSD notation (see Sec. 5.2) and sent to the *Center Information Manager (CIM)* which holds a database on the available resources and the currently planned job schedule on the computers of a local site. The CIM checks the availability of resources and informs the broker, which decides whether to submit the job to the local site(s). In case of a positive decision, a co-allocator is asked to allocate the resource(s) just before runtime. This task is done in cooperation with the *Center Resource Manager (CRM)* which performs the pre- and post-processing (e.g. job setup, start of the runtime environment, monitoring, cleanup) to support the execution of the Grid job on the local system.

The described framework, which has been implemented for pharmaceutical drug design projects [3], is very similar to the scheme used in the Globus project [4, 5]. In the following, we discuss the specification of resources and thereafter the interface to the local cluster management.

5.2 Resource and Service Description

RSD (Resource and Service Description) [2] is a set of tools and services for specifying, registering, requesting and accessing any kind of resource in a Grid environment. In this context, the term “resource” is used in its broadest sense: A resource may be, for example, a compute node, a LAN link, a data archive, a graphics device, or a software package. Each of these resource objects may have attributes, e.g., a computing node may be characterized by its processor type, the clock rate, the amount of main memory, and any other useful information.

In contrast to the Globus approach [4], which uses two separate constructs for describing the available resources (MDS) and the resource requests (RSL), our approach is symmetrical. RSD is used by the application programmer to define the necessary resources for the job by means of a task graph and also by the site administrator to describe the available systems, their interconnection network and all available services. Resources and services are represented by hierarchical graphs with attributed nodes and edges describing static and dynamic properties such as communication bandwidth, message latency, or CPU load. This allows to run a graph mapping algorithm when matching the resource request with the available resources in the broker. Of course, the vertice’s attributes and the timing information (the schedule) is taken into account in the mapping process.

Fig. 5 shows an RSD specification of a cluster with eight Intel Pentium 3 nodes connected by SCI adapter cards in a unidirectional ring. Note that this RSD code is shown for illustrative purposes only. In practice, there is no need to use the somewhat clumsy RSD language for specifying resources, because there also exists a graphical user interface that generates RSD code. The code is translated to XML and stored by the CIM.

```
NODE cluster {
  CONST N = 8;          // number of nodes
  CONST CPU[] = (''P3'', ''P4'', ''IA64''); // CPU types

  // specify 7 nodes with Intel P3 and SCI ports
  FOR i=1 TO N-1 DO
     NODE i { PORT SCI; CPU=P3; MEMORY=512; OS=Linux; };
  OD
  // first node has 1 GB and an additional ATM port
  NODE 0 { PORT SCI; PORT ATM; CPU=P3; MEMORY=1024; OS=Linux; };

  // specify unidirectional SCI ring, 1 GB/s
  FOR i=0 TO N-1 DO
     EDGE edge_$i_to_$((i+1) MOD N) {
        NODE i PORT SCI => NODE ((i+1) MOD N) PORT SCI; BW = 1.0; };
  OD
  // virtual edge from node 0 to external hypernode
  ASSIGN edge_to_hypernode_port {
     NODE 0 PORT ATM <=> PORT ATM; };
};
```

Figure 5: RSD example: A cluster with unidirectional SCI topology

5.3 Local Resource Management

The seamless integration of local systems into the Grid still poses a big problem because many existing local management systems do not support necessary features like online control, site autonomy, or co-allocation. For clusters, the batch queuing systems PBS, NQE, LSF, LoadLeveler, Condor, and Grid Engine (formerly Codine) are most popular. They handle job submission by allocating resources via predefined queues. The batch system simply de-queues one job after the other when resources become available. This guarantees a high system utilization but does not allow distributed process control, online data scheduling, and co-allocation, which are all needed in Grid environments.

Our *CCS* (Computing Center Software) resource management system [7] copes with these problems. CCS is based on job mapping rather than job queuing. It primarily runs in space-sharing mode, thereby giving jobs exclusive access to resources at a specified time. This allows remote application steering, co-allocation with guaranteed access to resources at a given time and also data scheduling. In addition, CCS supports time-sharing mode as well, which is used for the efficient execution of parameterized jobs that are run in high-throughput mode.

With the fast innovation rate in hardware technology, we saw the need to encapsulate the technical aspects of the underlying systems and to provide a coherent interface to the user and the system administrator. Robustness, portability, extensibility, and the efficient support of space sharing systems, have been among the most important design criteria. This also caused us to design (as early as 1992) the described RSD language for the specification of resources. While initially there was just a textual interface, we later also included a graphical RSD editor for easier manipulation of resource requests. Also a site management layer (with the CIM and CRM illustrated in Fig. 4), which coordinates local CCS domains, was included.

In its current version (V4.03) CCS comprises about 120.000 lines of code. CCS is by itself a distributed software, running on any kind of UNIX system (Linux, Solaris, AIX). Its functional units have been kept modular to allow easy adaptation to future environments. More information on the architecture and features of CCS can be found in [7].

6 Summary

High-performance commodity clusters are getting more and more important as compute nodes in Grid environments. We have presented the framework of a Grid-aware cluster which supports site autonomy while providing the necessary hooks and services for the easy integration into the Grid, namely guaranteed scheduled access to resources, support of co-allocation and remote steering. Moreover, the cluster is resilient by means of monitoring and control functions and it provides a fault tolerant networked storage architecture without having to pay the price of a hardware RAID system.

Acknowledgements

Thanks to Jens Simon and Florian Schintke for providing valuable information and to Sebastian Heidl for running the tests on GbE. Part of this work was financially supported by the EU project Datagrid.

References

[1] M. Aderholz, et al. *Models of Networked Analysis at Regional Centres for LHC Experiments. Monarc Phase 2 Report.* March 2000.

[2] M. Brune, A. Reinefeld, and J. Varnholt. *A Resource Description Environment for Distributed Computing Systems.* HPDC'99, Redondo Beach (1999), 279–286.

[3] R. Bywater, J. Gehring, A. Reinefeld, F. Rippmann, and A. Weber. *Metacomputing in Practice: A Distributed Server for Pharmaceutical Industry.* FGCS 15,5/6, Elsevier 1999, 769–785.

[4] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. *A Resource Management Architecture for Metacomputing Systems.* JSSPP 1998, 62–82.

[5] I. Foster and C. Kesselman (eds.). *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufman Publ., 1999.

[6] K. Holtman and J. Bunn. *Scalability to Hundreds of Clients in HEP Object Databases.* Proc. of CHEP '98, Chicago, USA, August 1998.

[7] A. Keller and A. Reinefeld. *Anatomy of a Resource Management System for HPC-Clusters.* In: Annual Review of Scalable Computing, Vol. 3, Singapore University Press, 2001, 1–31.

[8] *Myrinet.* `www.myricom.com`

[9] *SCI.* Dolphin Interconnect: `www.dolphinics.com`

[10] T. L. Sterling, J. Salmon, D. J. Becker, and D. F. Savarese. *How to Build a Beowulf. A Guide to the Implementation and Application of PC Clusters.* The MIT Press, Cambridge, 1999.

[11] *STP on Linux.* `http://oss.sgi.com/projects/stp`.