Prototyping of
CMS Storage Management

PROEFONTWERP

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
Rector Magnificus, prof.dr. M. Rem, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op 29 mei 2000 om 16.00 uur

door

Koen Holtman

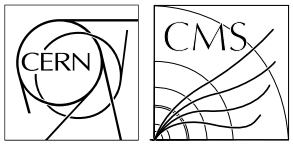
Geboren te Tegelen

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr. P.M.E. De Bra
en
prof.dr. R.H. McClatchey

Copromotor:
dr. P.D.V. van der Stok

/stan ackermans institute, center for technological design



The research reported in this thesis has been carried out at CERN, in the context of the CMS collaboration.

SIKS Dissertation Series No. 2000-2

The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Graduate School for Information and Knowledge Systems.

Druk: Universiteitsdrukkerij Technische Universiteit Eindhoven

Cover picture: a part of the CERN computer centre, in January 2000. On the right, in the back, are four Storagetek Powderhorn tape robots, with a total storage capacity of up to 1.2 Petabytes. See also figure 7.5.

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Holtman, Koen J. G.

Prototyping of CMS Storage Management/
by Koen J. G. Holtman.–
Eindhoven : Eindhoven University of Technology, 2000.
Proefontwerp. – ISBN 90-386-0771-7

NUGI 855
Subject headings: large databases / database organisation /
computers; storage-management / high energy physics
CR Subject classification (1998) : H.2.2, B.8.2, D.2.10, J.2

Contents

Chapter 1

Introduction

In this thesis I report on a designer's Ph.D. project, called a *proefontwerp* in Dutch, that was performed at CERN, the European laboratory for particle physics, with joint funding by the Eindhoven University of Technology.

The goal of a designer's Ph.D. project is to design an artifact that satisfies the needs and criteria of some user community [1]. Because of this goal, a designer's Ph.D. project differs from a normal Ph.D. project, where the goal is to do research that adds to the body of scientific knowledge. Like a normal Ph.D. project, a designer's Ph.D. project works on the boundaries of what is scientifically well-understood, as it has to demonstrate the ability of the Ph.D. candidate to do autonomous work on a scientific level. During the project, new scientific knowledge was created, but creation was driven by the needs of the environment, not by pure scientific curiosity. This implies some specific differences in the methodology used, and in the structure of the resulting thesis (see sections 1.3 and 1.5).

1.1 The CMS storage manager

The designer's Ph.D. project is concerned with the design of the *CMS storage manager*. CMS is a high energy physics experiment, the name stands for 'Compact Muon Solenoid'. This 'Compact Muon Solenoid' is a description of the technology of the CMS particle physics detector, which is the centrepiece of the experiment. The CMS detector will be built at CERN, the European laboratory for particle physics in Geneva, Switzerland. The CMS experiment will start operation in 2005. The organisational entity which is responsible for building and operating the CMS experiment is called the CMS *collaboration*. The CMS experiment is discussed in more detail in chapter 2.

To analyse the physics data it obtains, the CMS experiment will use a large computing system. This large computing system is called the *physics analysis system*. The CMS storage manager, that is the subject of this thesis, is one of the software components of the physics analysis system.

A layered view of the CMS physics analysis system is shown in figure 1.1. The storage manager acts as an intermediate between the physics-oriented software components

above it, and the largely commercial, general purpose software and hardware components below it. The layers above and below the storage manager are discussed in more detail in chapters 3 and 4.

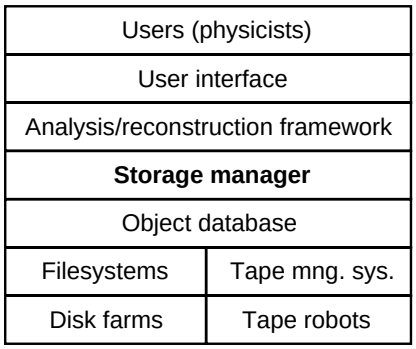


Figure 1.1: A layered view of the CMS physics analysis system.

1.2 Requirements for the CMS storage manager

The CMS storage manager is a software component that has to be finished in 2005, in time for the startup of the CMS experiment. This Ph.D. project is concerned with designing prototypes for the CMS storage manager. The prototypes aim at satisfying (subsets of) the requirements for the storage manager. In this section, the full requirements for the CMS storage manager are introduced. Prototypes are discussed further in section 1.4

The requirements for the CMS storage manager are as follows. The main task of the CMS storage manager is to offer physics data storage and retrieval services to the software layers above it. The storage manager should offer a high-level interface that is geared towards the specific needs of physics analysis. Data to be stored takes the form of *physics objects*, which are accepted from, or delivered to, the software layers using the storage manager. The storage manager should implement I/O optimisations that are specific to physics analysis workloads, and should shield its users from details like the physical storage location of objects.

The above requirements, centred around transparent data access and I/O optimisation, are not exceptional in themselves. The same requirements hold for many I/O layers that have been designed in the past, and will hold for many layers to be designed in future. What makes the CMS storage manager special are the size, workload, performance, and time scale parameters under which its requirements have to be fulfilled. Chapters 2 and 3 give detailed descriptions of these parameters. The most central constraints and parameters are introduced briefly below.

- Large timescales: the CMS physics analysis system, and its storage manager, have to be ready in 2005, so exact properties of the hardware they run on are uncertain. The system will have a lifetime of at least 15 years.
- Large data volume: 1 Petabyte of CMS detector data will be stored per year from 2005 onwards, over a period of at least 15 years. 1 Petabyte (PB) is 10^{15}

bytes or 1000 Terabytes (TB). The CMS data will be stored on tape robots and on large disk farms. Some of the disk farm capacity will be used as a cache for the data on tape.

- Large I/O requirements: when the experiment is running it requires 100 MB/s I/O capacity for storing the experimental data produced. Interactive physics analysis is expected to require at least a few GB/s of I/O capacity to support all concurrent users. From 2005 onwards, physics analysis workloads, and I/O requirements, will expand to take advantage of improvements in hardware price/performance. The software architecture needs to allow for such expansion.
- Parallel processing: the system will have hundreds of simultaneous users, and many CPU-intensive jobs will be executed on tens to hundreds of CPUs in parallel. Thus there will be at least hundreds of parallel I/O streams in the system.
- Atypical workload properties: physics analysis workloads differ in some significant ways from other database and I/O workloads that have been studied widely in industry and academia, like transaction processing workloads, CAD workloads, file system workloads, and video on demand workloads (see section 3.5. Because of these differences, much of the existing research on high-performance I/O does not apply.

The CMS data volume is much larger than the volumes found in currently deployed large databases. A 1998 survey [2] of in-production, commercial large databases ranks a 16.8 TB database as the largest, with all others below 7 TB. The size of the 'indexable web', all HTML pages that could potentially be indexed by web search engines, is estimated to be 15 TB in late 1999 [3]. Recently, it was reported that AT&T is setting up a call records data warehouse with 100 TB tape capacity [4]. A data warehouse system at Wallmart has 104 TB of disk capacity, storing multiple representations of a 31 TB data set of sales records [5]. The sales records are also backed up on tape, but tape does not play a direct role in supporting the data mining jobs on this system. Of the currently known production systems, the closest to the CMS data volume is the BaBar experiment, which has started in 1999, storing about 200 TB of physics data per year [6]. The NASA EOSDIS (Earth Observing System Data and Information System) system has similar data rates and timelines [7] [8].

All the above parameters imply significant risks, that have to be addressed by the CMS storage manager design. The set of risks is too large to be covered in a single designer's Ph.D. project. In fact, a large R&D effort, to study storage management and other issues, was already in place at CERN before this Ph.D. project was started. The Ph.D. project was set up to contribute to this R&D effort.

The large timescales mentioned above do not only influence the design of the CMS storage manager, they are also a main determining factor for the design methodology.

1.3 Overview of the design methodology

This section briefly presents the design methodology used in the project. A more detailed discussion, with a discussion of alternatives and justifications, can be found in chapter 8.

For its software efforts, the CMS experiment adopted a form of cyclic prototyping [9], with a cycle time of some 2-3 years [10]. In early cycles, the emphasis is not on producing a fully functional system, but on exploring and eliminating as many risks as possible. It is not expected that all code, or even most of the code, developed in one cycle is carried forward to the next cycle, but architectural elements and experience is carried forward.

The designer's Ph.D. project discussed in this thesis fills one of these 'big' 2-3 year cycles, as far as the design of the storage manager is concerned. Inside this one big cycle, cyclic prototyping was performed on a smaller scale, going through three 'small' cycles in which code re-use did happen. A small cycle starts with the identification of a new risk to explore. As discussed above, there is a large set of risks to choose from. The choice for a particular risk is guided by an assessment of which risks are most urgent. It is also guided by the need to coordinate with other researchers working inside and outside CERN: different groups of risks tend to end up being 'owned' by different people.

Once an urgent risk, not yet under close study elsewhere, is identified, it is expressed in terms of a research question. This research question is then addressed in a single project cycle. See section 1.4 for the three questions belonging to the three cycles in this project. The cycle is finished when the risk under study has been addressed to such an extent that residual risks are small compared to the still open risks.

Note that the above criterion for finishing a cycle is very different from the finishing criteria applied in 'pure science'. In pure science, research on some question is only finished when the correctness of the answer is firmly supported with evidence. This designer's Ph.D. project is performed in an 'applied science' or 'industrial environment'. The environment imposes a large set of risks that all have to be addressed within timing constraints, and this yields different criteria for finishing a cycle: it makes it necessary to finish the cycle with partial answers and residual risks. See chapter 8 for a further discussion of the risk-driven, cyclic methodology used for the three cycles in this designer's Ph.D. project. Inside the cycles themselves, an incremental development style was used. One exception to this was the use of a methodology centring around a 'big bang' integration step in the second half of the third cycle, corresponding to the second half of chapter 7. This 'big bang' methodology is discussed in section 7.4.

1.4 Prototyping work performed in this project

The risks addressed in this project centre around I/O performance. The connecting theme throughout the three project design cycles is the *clustering* of objects. The clustering of objects is the order in which object data is placed on physical storage media like disks and tapes. The performance of a physics analysis application depends crucially on having a good match between the object access patterns of the physics analysis jobs and the object clustering on the media. It was investigated how, and to what extent, clustering decisions can be made by the storage manager.

Also investigated was *reclustering*, the changing of a clustering arrangement. Reclustering, can be done by either re-arranging existing representations of physics objects,

or by clustering copies of these representations in a new arrangement. Reclustering is important if good I/O performance is to be maintained under changing physics analysis workloads.

Other important themes in the project are parallel processing and scalability, both for the storage management prototypes and for the software and hardware layers underlying the storage manager. In particular, during the project I made three visits to Caltech (the California Institute of Technology, Pasadena, USA) to do scalability studies on a 256-processor supercomputer.

The specific research questions addressed in the three design cycles are as follows.

1. How should physics data be mapped onto the object database, given requirements for scalability and clustering efficiency? This question is answered in chapter 5.
2. How can reclustering be used to optimise disk I/O? This question is answered in chapter 6.
3. Given likely physics data clustering arrangements on tape, how does one use caching on disk to reduce tape I/O? What about reclustering on tape? These questions are answered in chapter 7.

These questions follow an incremental development path: the results obtained in answering earlier questions are used as a basis for the next cycle.

1.5 Structure of this thesis

Chapters 2–4 discuss the environment of the project. Chapter 2 discusses the CMS environment, and the place of the project in it. Chapter 3 discusses the properties of physics analysis, that is the properties of the layers above the storage manager (see figure 1.2). Chapter 4 discusses the hardware and software layers below the storage manager.

Chapters 5–7 discuss the three subsequent design cycles in the project.

Chapter 8 concludes with a discussion of the project results and the design process.

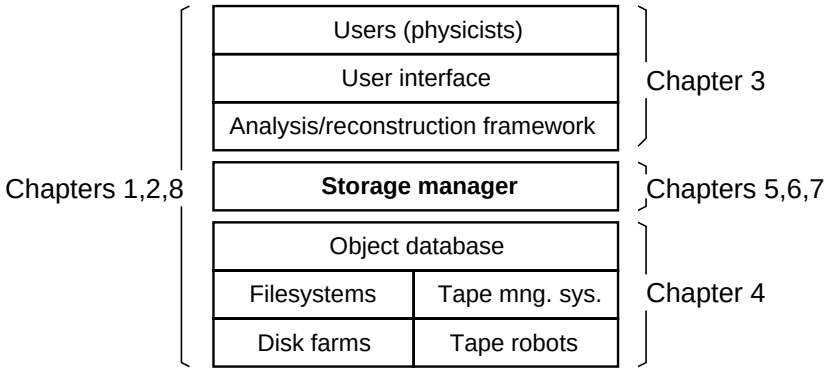


Figure 1.2: Relation between system layers and chapters.

Chapter 2

The CMS Environment

2.1 Introduction

This designer's Ph.D. project is concerned with the design of the CMS storage manager. CMS is a high energy physics experiment at CERN, the European laboratory for particle physics. This chapter places the project in the context of the CMS environment. The chapter first discusses CERN and the CMS experiment from a high level. Then it focuses on the physics analysis process and on the plans for the software infrastructure. Finally, it introduces some important organisational entities in the project environment.

2.2 CERN

At CERN, the European laboratory for particle physics, the fundamental structure of matter is studied using particle accelerators. The acronym CERN comes from the earlier French title: "Conseil Europeen pour la Recherche Nucleaire". CERN is located on the Franco-Swiss border west of Geneva. CERN was founded in 1954, and is currently being funded by 20 European countries. CERN employs just under 3000 people, only a fraction of those are actually particle physicists. This reflects the role of CERN: it does not so much perform particle physics research itself, but rather offers its research facilities to the particle physicists in Europe and increasingly in the whole world. About half of the world's particle physicists, some 6500 researchers from over 500 universities and institutes in some 80 countries, use CERN's facilities.

The study of matter with accelerators is part of the field of High Energy Physics (HEP). A lot of the technology, including software technology, used in the high energy physics field is developed specifically for high energy physics.

2.3 LEP and LHC

CERN currently has the largest particle accelerator in the world, the LEP, Large Electron Positron accelerator [11], which is a ring with a circumference of 26.7 km. The accelerator ring is located in an underground tunnel. This is mainly done to protect it from outside vibrations. The components of large accelerators require precision align-

ment over long distances, to say 0.1 mm over tens of kilometres.

The successor of LEP is called the LHC, Large Hadron Collider [12]. Currently, a large share of CERN's resources goes into the design and construction of the LHC accelerator, and the LHC detectors with which high energy particle collisions can be studied. The two general purpose LHC detectors are ATLAS, A Toroidal LHC ApparatuS [13], and CMS, The Compact Muon Solenoid [14]. Some special-purpose detectors, like ALICE [15] and LHCb [16], are also being built. The construction of the LHC and its detectors is scheduled to be completed in 2005. Figure 2.1 shows the layout of the LHC complex and its detectors.

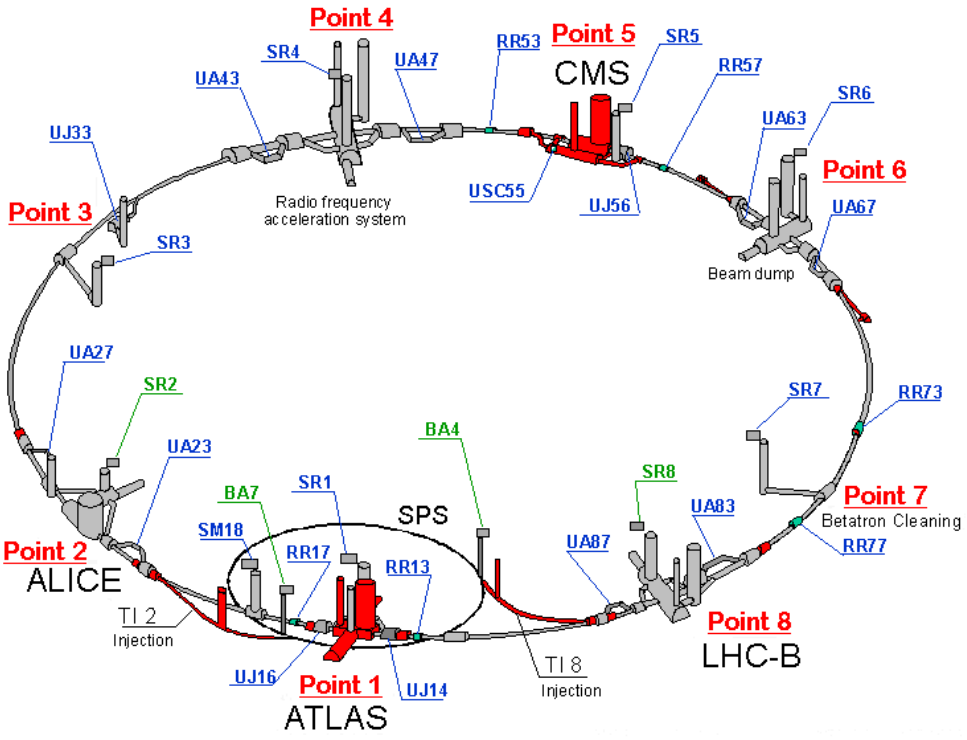


Figure 2.1: Layout of the LHC tunnel and cavern complex. The lighter parts have already been dug for LEP, the darker parts have to be newly dug for the LHC and its detectors.

2.4 CMS

In accelerators like LEP and LHC, particles can be accelerated to near-light speed, and collided head-on. Such high-energy collisions happen inside detectors, which detect some of the collision products (photons, electrons, muons, ...) as they emanate from the collision point.

The CMS detector (figure 2.2) is one of the two general purpose detectors of the LHC

accelerator. It is being designed and built, and will be used, by a world-wide collaboration, the CMS collaboration, that currently consists of some 2200 people in 145 institutes, divided over 30 countries. These institutes contribute funds and manpower to CMS, and will also be the users of the detector when it is finished. CERN is one of the institutes in the CMS collaboration.

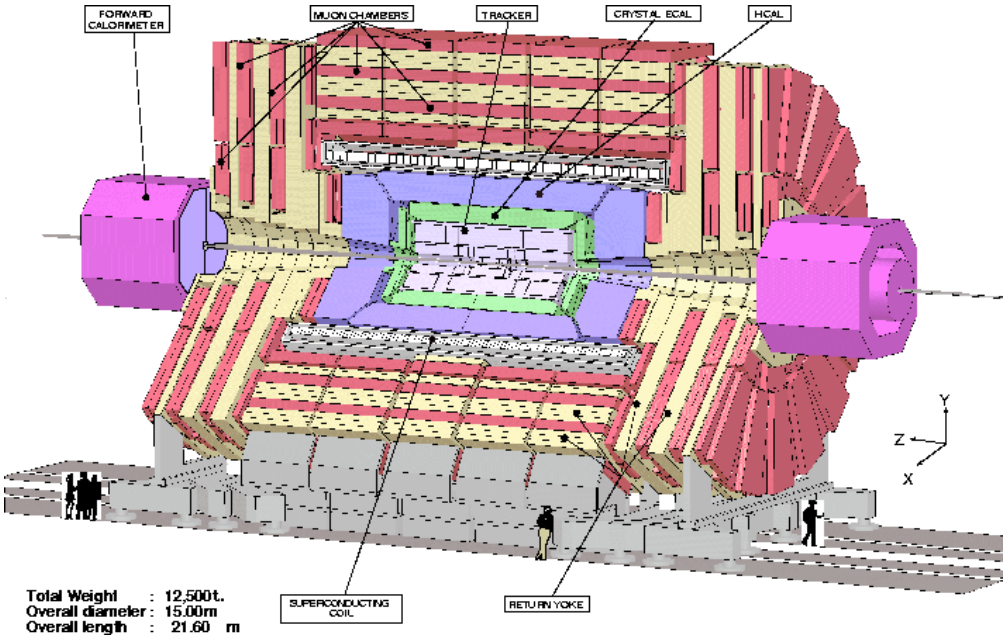


Figure 2.2: The CMS detector.

As shown in figure 2.2, the CMS detector consists of several 'subdetectors' layered around the collision point in the middle. These subdetectors all use different detection technologies.

The three inner subdetectors (called trackers) specialise in measuring the exact paths (tracks) of the particles as they emanate from the collision point.

Taking all subdetectors together, CMS has 15 million individual detector channels. The three outer subdetectors (called calorimeters) specialise in measuring the energies (momenta) that these particles possess. Moving outward, these subdetectors specialise in increasingly heavy particles, with larger energies. The CMS detector also contains a powerful (4 Tesla) super-conducting magnet. The magnetic field bends the tracks of charged particles as they emanate from the collision point, making them easier to identify.

In future operation, the LHC accelerator lets two bunches of particles cross each other inside the CMS detector 40.000.000 times each second. Every bunch contains some 10^{11} protons. In every bunch crossing in the detector, on average 20 collisions occur between two protons from opposite bunches. A bunch crossing with collisions is called an event. Figure 2.3 shows an example of an event. Note that the picture of the collision products is very complex, and represents a lot of information. The measurements of the event, done by the detector elements in the CMS detector, are called the 'raw event

data'. The size of the raw event data for a single CMS event is about 1 MB. This 1 MB is the size after compression. The uncompressed complete event signal from the 15 million detector channels is much larger than 1 MB, but it is a sparse block of data, because in every event, only some of the detector elements actually 'see' a collision product pass through them. The 'zeroes' from the remaining detector elements are removed using a 'zero suppression' lossless compression algorithm.

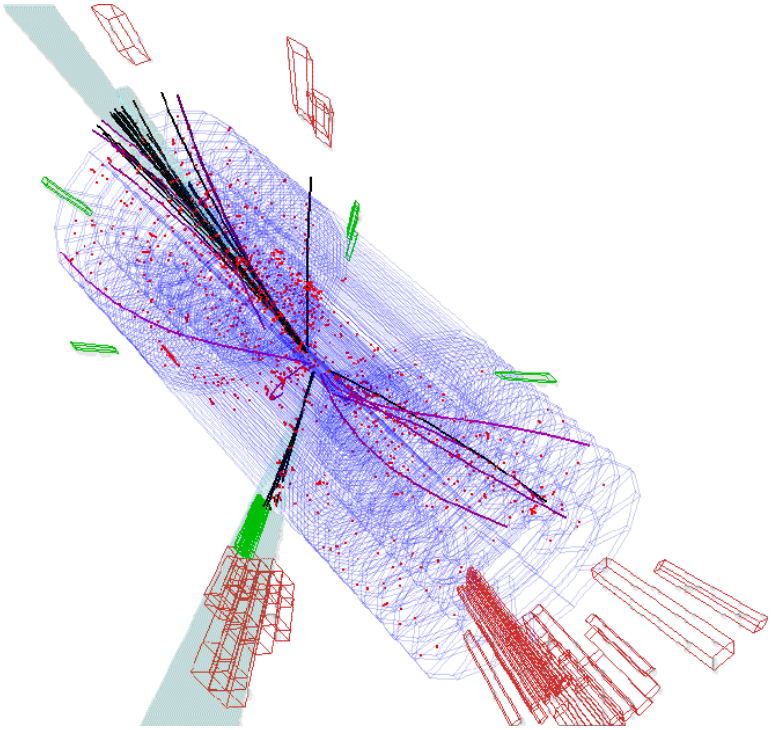


Figure 2.3: A CMS event (simulation).

Of the 40.000.000 events in a second, some 100 are selected for storage and later analysis. This selection is done with a fast real-time filtering system. Data analysis is done interactively, by CMS physicists working all over the world. Apart from a central data processing system at CERN, there will be some 5–10 regional centres all over the world that will support data processing. Further processing may happen locally at the physicist's home institute, maybe also on the physicist's desktop machine. The 1 MB 'raw event' data for each event is not analysed directly. Instead, for every stored raw event, a number of summary objects are computed. These summary objects range in size from a 100 KB 'reconstructed tracks' object to a 100 byte 'event tag' object, see section 3.2 for details. This summary data will be replicated widely, depending on needs and capacities. The original raw data will stay mostly in CERN's central robotic tape store, though some of it may be replicated too. Due to the slowness of random data access on tape robots, the access to raw data will be severely limited.

2.5 Physics analysis

By studying the momenta, directions, and other properties of the collision products in the event, physicists can learn more about the exact nature of the particles and forces that were involved in the collision.

For example, to learn more about Higgs bosons, one can study events in which a collision produced a Higgs boson that then decayed into four charged leptons. (A Higgs boson decays almost immediately after creation, so it cannot be observed directly, only its decay products can be observed.) A Higgs boson analysis effort can therefore start with isolating the set of events in which four charged leptons were produced. Not all events in this set correspond to the decay of a Higgs boson: there are many other physics processes that also produce charged leptons. Therefore, subsequent isolation steps are needed, in which 'background' events, in which the leptons were not produced by a decaying Higgs boson, are eliminated as much as possible. Background events can be identified by looking at other observables in the event record, like the non-lepton particles that were produced, or the momenta of particles that left the collision point. Once enough background events have been eliminated, some important properties of the Higgs boson can be determined by doing a statistical analysis on the set of events that are left.

The data reduction factor in physics analysis is enormous. The final event set in the above example may contain only a few hundreds of events, selected from the 4×10^{14} events that occurred in one year in the CMS detector. This gives a data reduction factor of about 1 in 10^{12}. As discussed in the next section, this data reduction is achieved by using large computing systems.

2.6 Online and offline systems

There are two main systems in CMS data processing: the online system and the offline system (figure 2.4).

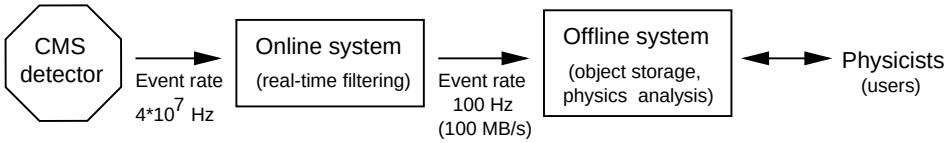


Figure 2.4: The CMS online and offline systems and their environment.

2.6.1 Online system

The online system is a real-time system, which has the task of selecting (filtering out) the on average 100 most interesting events out of the 4×10^7 events occurring every second in the CMS detector. Filtering is done by a multi-level system: a first filtering step is made by a fast filter implemented in hardware, called the first level trigger. Additional steps are made with a software-based system running on a large processor farm.

The output of the online system is a data stream of about 100 events, of about 1 MB each, per second. The detector takes data for about 100 days each year, which corresponds to about 10^7 seconds. Thus, in a year, the offline system will record 10^9 events or 1 Petabyte (1 PB, 10^{15} bytes) of (already compressed) raw event data. The total running time of the LHC is about 15 years. The LHC data volumes are the largest of any known project in the time frame involved [17]. One of the big challenges in CMS computing is to invent methods and techniques that scale to the Petabyte level.

2.6.2 Offline system

The stored events will be analysed by about 20 groups of physicists in the CMS collaboration, using a large computing system known as the offline system.

For the Higgs analysis example in section 2.5, with a data reduction factor of 10^{12}, filtering by the online system accounts for a reduction factor of about 10^6. The resulting event set stored in the offline system is then further narrowed down, by a factor 10^6, in a number of phases. In each phase, a *cut predicate* is developed, that 'cuts away' some of the events in the current set to obtain a smaller event set for the next phase. This cutting process is discussed in greater detail in chapter 3.

Some parts of physics analysis are highly CPU-intensive, other parts rely heavily on the I/O bandwidth that can be achieved when accessing precomputed results. It is expected that the physicists in the collaboration will be able to use whatever computing power the offline system makes available to them: an increase in computing power means an increase in the potential for interesting physics discoveries, because analysis jobs can look for more subtle effects in the same amount of running time.

The offline system relies on massive parallelism and special optimisation techniques to get the most out of standard hardware. The hardware will be upgraded through time to profit from new advances in computing technology. Performance is likely to grow with several orders of magnitude from 2005 to 2020 [10], and the system architecture and data model have to take this into account, by aiming for a very good scalability.

The offline system is a distributed system, in which storage and computing resources all over the world are tied together. As mentioned above, apart from a central data processing system at CERN, there are some 5–10 regional centres all over the world that support data processing. Further processing may happen locally at the physicist's home institute, maybe also on the physicist's desktop machine. Investigations into managing this distributed system, and the data exchange and replication between its components, are still in an early phase [18]. General replication issues are not considered in this thesis. The designer's Ph.D. project focuses on storage management issues at a single site, be it the CERN central site, a regional centre, or an institute processing facility. The use of desktop machines was not considered specifically, though some of the techniques developed are applicable to desktop machines too.

2.6.3 Offline system dimensions

Below are system dimension estimates for the initial offline system in 2005, taken from [10]. These figures should not be seen as performance goals: they are predictions that guide system design. Major sources of uncertainty are the exact needs of physics analysis, these depend on the exact features of the physics processes observed in a new energy region, and the exact price/performance figures for 2005 hardware. To some extent, system performance in 2005 will adapt to what can be feasibly bought at 2005 prices. Chapter 4 discusses these price/performance uncertainties in greater detail.

The estimated 2005 system dimensions in [10] are as follows.

Processing power. The offline system has about 10^7 MIPS of processing power, and relies heavily on parallelisation to achieve the desired speeds.

Storage capacity. The system has a robotic tape store with a capacity of several PB, and a disk cache of several hundred TB.

I/O throughput rates. For the central data store at CERN, the maximum integrated throughput from the disks to processors is of the order 100 GB/s, whereas the integrated tape-disk throughput is of the order 1 GB/s (32 PB/year) spread over some tens of devices. Most of this throughput is devoted to supporting physics analysis (chapter 3). Only 100 MB/s of the tape throughput is needed for storing physics data coming from the online system when the detector is active. Some few hundreds of MB/s of disk throughput will be used to buffer this data before it is stored on tape.

Taking the above figures as a starting point, some estimates of the amount of hardware implied have been made in the MONARC project [19]. For the offline facilities of the CMS experiment located at CERN in the year 2006, these are as follows.

CPU capacity. Some 1400 boxes with 4 processors each, in 160 clusters with 40 sub-farms.

Disk capacity. Some 5400 hard disks in 340 disk arrays.

Tape capacity. Some 100 tape drives.

Power requirements. Some 400 Kilowatts.

Space requirements. Some 370 m^2.

2.7 Object orientation and use of an object database

The CMS collaboration is using object oriented (OO) technology in its software development, with the terminology that comes along with it. This CMS OO terminology is used throughout this thesis. Pieces of physics data are thus always called *physics objects.* A *persistent object* is an object that has been saved on long-term storage, technically an object that continues to exist event after the program that created it has terminated.

The CMS offline software design is centred around the use of a single coherent persistent object storage system [10]. All CMS data (physics objects and other objects) will be in this object store, and all CMS programs will operate on these objects through a

high-level OO interface. This interface shields its users from details like the physical storage location of objects.

In the CMS computing strategy [10] the choice was made to implement the object store on top of a commercial object database management system (ODBMS). The Objectivity/DB object database [20] was chosen as the basis for all object storage prototyping efforts in CMS in the 1996–2001 timeframe. The final choice for a production database will be made at the end of 2001. Section 4.2 discusses some details of Objectivity.

2.8 Organisational environment

As seen in section 2.4, the CMS collaboration currently consists of some 145 institutes, which contribute funds and manpower. Because of this, the organisational environment of this designer's Ph.D. project is quite complicated. For detailed descriptions of the environment, see [21] or [10]. Below, only some important organisational entities that are mentioned elsewhere in this thesis are introduced.

The **CMS software/computing project** [22] performs all offline system computing R&D in the CMS collaboration. It has members both at CERN, and at other institutes throughout the world that participate in the CMS experiment. The designer's Ph.D. project has been performed inside the CMS computing project. The focus of the project was to work towards CMS computing group milestones.

The **RD45 collaboration** [23] performs R&D on persistent storage management for high energy physics. RD45 is a loose collaboration of computing researchers throughout the world's high energy physics community. The collaboration is centred around a core group at CERN's information technology division, which maintains a common software base centred around the Objectivity product, and organises workshops, in which the community meets to present progress and discuss alternatives. I participated in the RD45 collaboration. Some of the Ph.D. project work is based on research results from RD45, and Ph.D. project results were reported to RD45, some specifically satisfying RD45 milestones.

The **GIOD project** [24] is a collaboration between CERN, Hewlett Packard, and Caltech, centred around a core group at Caltech. The project is aimed at constructing a large-scale prototype of an LHC Computing Centre. I participated in the GIOD project, and visited Caltech a number of times to do scalability tests on a HP supercomputer located there. These tests contributed both to GIOD goals and to the goals of the Ph.D. project.

2.9 Conclusions

As discussed in Chapter 1, the project is concerned with the design of the CMS storage manager, which is one of the layers of the CMS physics analysis system. The CMS physics analysis system will start running in 2005. Because of this long timescale, and the major uncertainties involved, the CMS software/computing project is currently focusing on prototyping activities that explore technology options for offline software.

As such, the CMS software/computing project provides an ideal environment in which to perform designer's Ph.D. projects. The exploration of technology options performed in this project is both scientifically interesting from a computing science viewpoint, and practically necessary for the planning of the CMS experiment. The research performed in this project contributed directly to CMS milestones.

This chapter discussed the CMS environment, and the place of the project in it. The next chapter focuses on the layers above the CMS storage manager.

Chapter 3

Properties of Physics Analysis

3.1 Introduction

This chapter discusses the properties of physics analysis, that is the properties of the software layers above the storage manager. Discussed are the demands these software layers make on the services offered by the storage manager, both with respect to the type of data that must be stored and with respect to the access patterns to stored data. It is not considered how the storage manager could meet these demands, that is done in later chapters. Thus, this chapter is concerned with requirements on the storage manager only.

Very little documentation exists on data access patterns in physics analysis: knowledge about it is scattered and incomplete. In most current experiments, most steps in physics analysis efforts are performed on 'regional' or 'home' systems. Such systems are administrated by their end users, who are primarily concerned with getting physics results. Consequently, information about user activities in physics analysis efforts is only very rarely gathered and fed back to the groups responsible for software development. Only the initial, group-level and centralised steps of physics analysis are well-documented. This lack of knowledge about physics analysis in current experiments is one of the factors that contributes to the lack of hard numbers on future CMS physics analysis requirements. Other factors are the inherent uncertainties about the physics in a new experiment, and the large timescale with the resulting uncertainties about hardware performance. In the end, requirements are not stated in terms of fixed numbers, but rather as a desire to enable the use of whatever hardware is available. This state of affairs influenced the design efforts, in the sense that the prototypes had to work well over a large possible parameter space, rather than optimise for a particular well-defined set of parameters.

A considerable part of the project time was spent gathering a picture of the whole analysis process. The information in this chapter was gathered by talking to people working in CMS, from some work contributed to RD45 [25] [17], and by attending an overview lecture series targeted at physicists new to high energy physics data analysis [26].

First the structure of event data is considered. Then the structure of the physics analysis process is discussed. Finally the expected operations on the store and the expected object access patterns are discussed.

3.2 Structure of event data

The CMS object store contains a number of physics objects for each event, as shown in figure 3.1. Among themselves, these objects form a hierarchy. At higher levels in the hierarchy, the objects become smaller, and can be thought of as holding summary descriptions of the data in the objects at a lower level. By accessing the smallest precomputed summary object whenever possible, physicists can save both CPU and I/O resources.

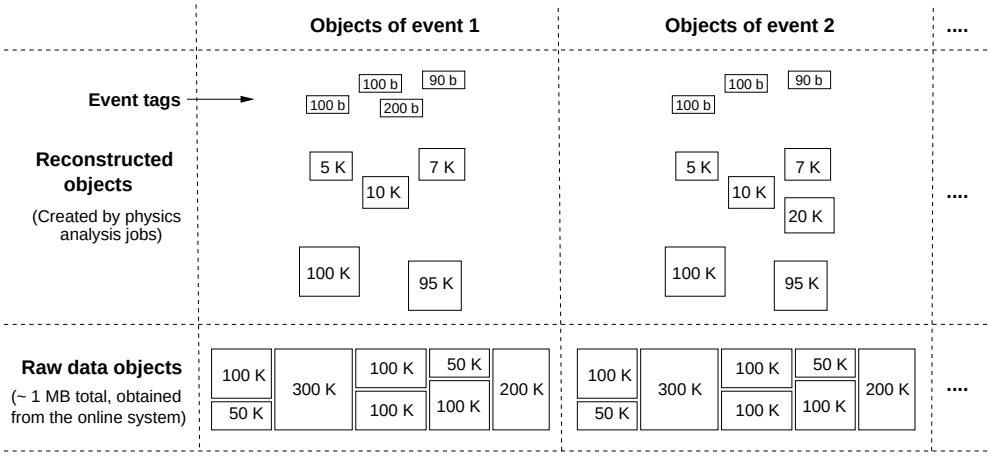


Figure 3.1: Example of the physics objects present for two events, as seen by users of the storage manager. The numbers indicate object sizes. The reconstructed object sizes shown reflect the CMS estimates in [10]. The sum of the sizes of the raw data objects for an event is 1 MB, this corresponds to the 1 MB raw event data specified in [10]. How this data will be partitioned into objects exactly is currently not known, the partitioning and sizes shown reflect an estimate based on the data model of the BaBar experiment [27].

At the lowest level of the hierarchy are *raw data* objects, which store all the detector measurements made at the occurrence of the event. Every event has about 1 MB of raw data in total, which is partitioned into objects according to some predefined scheme that follows the physical structure of the detector. Above the raw data objects are the *reconstructed objects*, they store interpretations of the raw data in terms of physics processes. Reconstructed objects can be created by physicists as needed, so different events may have different types and numbers of reconstructed objects. At the top of the hierarchy of reconstructed objects are *event tag* objects of some 100 bytes, which store only the most important properties of the event. Several versions of these event tag objects can exist at the same time.

3.2.1 Data dependencies

The physics objects for every event have a hierarchical arrangement among themselves: smaller, higher level reconstructed objects contain summaries of the data in

one or more larger (reconstructed or raw) objects below them. In fact, the smaller objects are usually computed on the basis of some next-larger objects below them in the hierarchy. Figure 3.2 shows possible data dependencies for some of the objects in figure 3.1. An arrow from A to B signifies that the value of B depends on A. The grey boxes represent physics algorithms used to compute the object: note that these all have particular versions. The lower-most grey box represents some 'calibration constants' that specify the configuration of the detector over time. Calibration is largely outside of the scope of this thesis, but see section 5.4.2 for a more detailed discussion of access to calibration data.

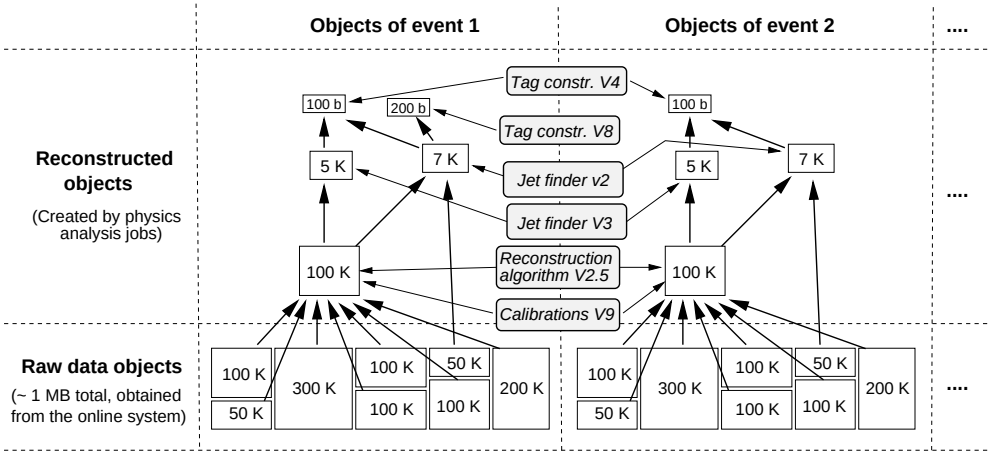


Figure 3.2: Data dependencies for some of the objects in figure 3.1. An arrow from A to B signifies that the value of B depends on A. The grey boxes represent physics algorithms used to compute the objects, and some 'calibration constants' used by these algorithms.

As is obvious from figure 3.2, the data dependencies for any physics object can become very complex. Note however that, for the two events in figure 3.2, the dependency graphs are similar. It is possible to make a single big dependency graph, in a metadata repository, that captures all possible dependencies between the physics objects of every event. Figure 3.3 shows such a metadata graph for the events in figure 3.2.

In the metadata graph, the physics objects are replaced by unique *type numbers.* These type numbers play an important role in the physics object addressing system that is developed in this thesis. When a physics analysis job starts up, the metadata graph is used to resolve a high-level description of the physics objects the job needs into the corresponding type numbers. These type numbers are then used to locate up the needed physics objects for every event.

3.2.2 Physics object addressing

A data model is used in which every physics object, that is every object shown in figure 3.1, can be uniquely identified by a tuple (*event ID , type number*). The first part of the tuple is the (unique) identifier of the event the object belongs to, the second the unique

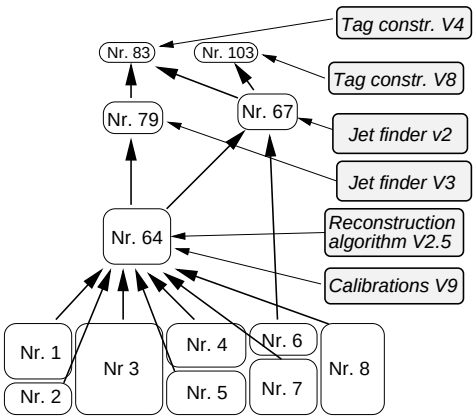


Figure 3.3: Data dependencies for all objects and events in figure 3.2, encoded as a metadata graph. The numbers in the nodes representing physics objects are globally unique type numbers.

identifier of the corresponding location in the object dependency metadata graph discussed above. This means that the term 'type' as used here is more specific than the concept of 'type' found in programming languages. As used here, the type encodes both the structure of the object and, for a reconstructed object, the exact method that was used to compute it.

The use of type numbers, together with the metadata graph, makes it possible to 'navigate' the physics object hierarchy for every event. Note that this navigational system does not rely on the presence of explicit pointers (object database associations) between the stored physics objects. By pushing the navigational complexity into the metadata system, the data model for stored physics objects is kept very simple. Without such simplicity, the implementation of reclustering type optimisations would be significantly more difficult.

3.2.3 I/O for tag and non-tag physics objects

Some existing physics analysis systems [28] [29] focus heavily on fast access to tag objects. In most existing systems, representations of tag objects are aggregated in specialised datastructures, which keep per-tag storage overheads to a minimum. Typically, the query function is evaluated against every tag in the aggregate tag collection. Special indexing structures, like trees that allow fast range searching, are not used, though research on their potential is currently ongoing [30]. Reasonably fast access to very large tag collections is very important, because the tags are used to make the initial event selection steps in a physics analysis effort.

It should be noted that, to store a collection of 100 byte tags for 10^9 events (one year's running of the detector), one needs about 100 GB of storage. With expected RAM chip prices in 2005, this means that, in 2005, tag collections could comfortably be kept in RAM memory. In this designer's Ph.D. project, it was decided to focus on disk and tape related I/O optimisations. This implies focusing on access to the larger, non-tag

physics objects. Scenarios in which only tags are accessed are not a consideration in the design. Physics analysis on tag objects has a great similarity to doing SQL queries on a single relational table. There is a lot of attention in academia and industry to optimising such SQL queries, see for example [31]. Following the risk-driven prototyping approach of this project, it therefore makes more sense to focus research on access to the larger non-tag objects, an area where more uncertainty exists.

3.2.4 Read-only nature of physics objects

Physics objects in the store have a read-only nature. For the raw data objects, which record measurements made by the detector, it is a requirement that they are never changed after being created. For the reconstructed objects, which record the output of a particular reconstruction algorithm, using particular settings for configuration constants and so on, there is no such strict requirement against modifying an object some time after creation. However, it is expected that modification of individual physics objects, except maybe of tags, will occur infrequently, if at all. If reconstruction algorithms or calibration constants are refined, this is expected to be accompanied by the creation of *new* physics objects, and the eventual deletion of the older versions, rather than the modification of the older versions. There are three reasons for this. First, when a new version is developed this is often accompanied by jobs that compare old and new versions: it is therefore necessary to keep the old versions around. Second, if many physicists are sharing a set of reconstructed objects for which some improved version becomes available, individual physicists only switch over to the new version after careful consideration, and at some convenient point in their analysis effort. This again means that the old versions should not be destroyed by the new versions.

Above, it was discussed how the nature of physics analysis leads to a preference for using read-only objects. If the use of read-write objects would give much better performance, physicists would easily drop their preference for read-only sets of physics objects. Updating an existing physics object set to reflect the results of a new algorithm, instead of creating a completely new set, could potentially be faster. However, there is some evidence [32] that, for physics objects with variable size fields, update operations are as expensive, or even more expensive, than creating new objects to reflect the changes. Thus, it is expected that the read-only nature of objects is retained even in cases where efficiency is the driving consideration.

Because of the above, only two types of object-level requests on the storage manager are expected to occur frequently:

- **store** this object of type T for event e,
- **retrieve** the stored object of type T for event e,

with **retrieve** occurring much more often than **store**. It is expected that **delete** operations will act on sets of objects of some type, rather than individual objects. As outlined above, it is not expected that an object-level **modify** operation will be used often, if at all. Efficient support for such modify operations was therefore not considered in this project. The 'implementation' of **modify** by the simple creation of a

new version, followed by the deletion of the original version, is considered efficient enough.

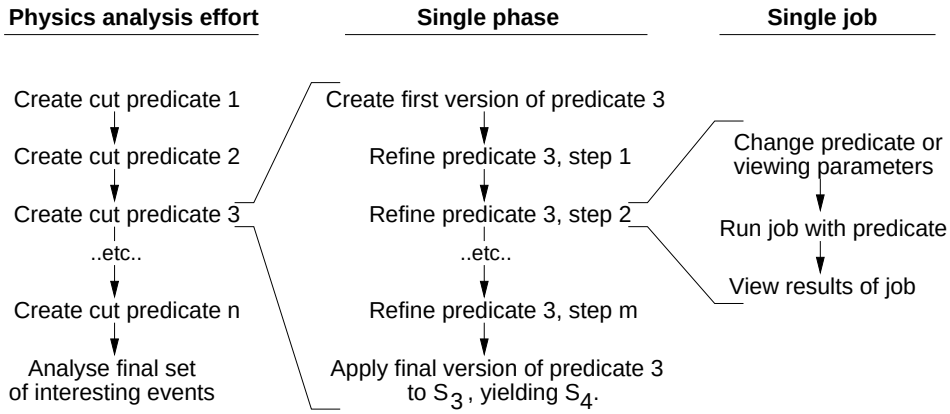


Figure 3.4: Overview of the physics analysis process.

3.3 Physics analysis process

As seen in section 2.6.2, a physics analysis effort on the offline system starts out with picking a large initial event set, containing say 10^9 events, which were selected by the online filters (section 2.6.1) and stored in the offline database. This set is then reduced to a smaller set of say 10^4 events whose corresponding particle collisions all displayed some particular interesting property. At the end this final set is analysed.

Figure 3.4 gives a graphical overview of the physics analysis process as performed on the offline system. Three levels of detail can be distinguished in this process.

1. The *analysis effort* level (left in figure 3.4): an analysis effort consists of several *phases.*
2. The *phase* level (middle in figure 3.4), a phase consists of several *jobs.*
3. The *job* level (right in figure 3.4).

Below, all three levels of detail are discussed in turn.

3.3.1 Analysis effort

A physics analysis effort primarily operates on *event sets.* Such event sets are usually represented as sets of event IDs: an event set representation does not contain any actual physics objects (figure 3.1) inside it. During an analysis effort, the physics properties of the events in larger event sets are examined to produce smaller sets. To examine the physics properties of an event in a set, a computation is made on one or more of the physics objects which belong to the event. These objects are obtained through the storage manager, navigating from the event IDs of the events in question (section 3.2.2).

A physics analysis effort on the offline system starts out with picking a large initial event set S_1. In a number of phases, this set is then reduced to a smaller set S_n. In each phase, a cut predicate p_i is developed, which is then applied to the events in the set S_i of the current phase to obtain the event set S_{i+1} of the next phase.

3.3.2 Phase

The goal of a phase i is to construct a new cut predicate p_i, on the events in the set S_i. This predicate should separate remaining 'interesting' from 'uninteresting' events.

The construction of a single cut predicate can take a significant amount of time: from weeks to months. In extreme cases, a team of physicists can spend more than a year constructing a cut predicate. One keeps refining the predicate (usually by tuning one or more of its constants), until its effects on the event set S_i under consideration are both appropriate for the physics goals under consideration, and well-understood.

To study the effect of the predicate p_i under construction, one runs jobs applying the current version of the predicate, or some component of the predicate, to the event set S_i or some subset thereof. Jobs can also be run over simulated events that have exactly known properties. In a single phase, one can thus expect database usage in which the same sets of (real or simulated) events are accessed over and over again in subsequent jobs, while various constants are tuned. Thus, there is an obvious opportunity for caching-type optimisations: one can speed up the system by keeping the sets that are revisited often on faster storage. Caching optimisations are simplified by the read-only nature of physics objects. The performance penalties that are usually associated with maintaining cache consistency if objects are modified are simply not a consideration in the caching of read-only physics objects.

3.3.3 Job

A typical job computes a, possibly very CPU intensive, function $f(e)$ for all events e in some set S. Different types of functions can be used, with function results processed in different ways. In some jobs, the function is a 'reconstruction algorithm' that computes a complex new object to be stored for later analysis. In other jobs, the function returns one or more floating point values, and the output of the job is a set of histograms of these values. Finally, the function can be a boolean cut predicate. Some jobs apply multiple functions to every event, for example combining an event cutting calculation with a histogramming calculation.

The I/O profile of a job is usually dominated by the reading of physics objects. The design work therefore focuses on the problem of optimising these read operations. In the remainder of this paragraph, the other parts of the job I/O profile are briefly discussed. The set S of events to which the function should be applied is generally represented as a set of event IDs, or as a predicate on a fast index of 'event tags'. In either case, visiting the set does not take much I/O resources. If the function computes a new object that has to be stored, there is some writing into the object store, but as function results are usually much smaller than the parts of the events needed to compute them, writing these results is seldom the dominant I/O factor. Jobs that store a new object for every

event are uncommon. Usually, function results are statistically aggregated, for example into a set of histogram objects, and this aggregation is a largely memory-based operation with low I/O requirements. In some physics jobs, time-dependent calibration constants need to be read, see section 5.4.2 for a short discussion of I/O techniques for calibration.

In a high energy physics experiment, there is no special correlation between the time an event collision took place and any other property of the event: events with similar properties are evenly distributed over the sequence of all events. In physics analysis jobs, events are treated as completely independent from each other. If function results are aggregated, the aggregation function does not depend on the order in which the function results are fed to it. Thus, from a physics standpoint, the order of traversal of a job event set does not influence the job result. This allows for massive parallelism in traversing the event set when executing a job, and creates important degrees of freedom in I/O scheduling.

3.4 Object access patterns in physics analysis

As discussed above, the I/O profile of a job is usually dominated by the reading of physics objects. The physics object access patterns for the subsequent jobs in a physics analysis effort can be approximated as in figure 3.5. In this figure, time runs from left to right in all three bar graphs concurrently. The bars represent the number of objects of a certain type read by each subsequent job. Each 'step' in the staircase pattern corresponds to a single phase. Figure 3.5 shows the access patterns on three different types of objects. It shows that in the first few phases, when the event set is still large, the jobs only reference the small (100 byte) tag objects for each event: the physicist avoids writing jobs that access the larger objects until all possibilities of eliminating events using the smaller objects have been exhausted.

It should be noted that figure 3.5 is an example only, and an example of a simple analysis effort. For different efforts, the widths and heights of the subsequent steps in the corresponding staircase pattern can vary enormously. Some analysis efforts split and merge event sets between phases. Some cut predicates read a variable number of objects for each event: after having read an object of type X, a cut predicate could determine that it does not need to read the type Y object to compute its results. Also, many analysis efforts, all working on different event sets, will be in progress at the same time, so that each type of object has many independent, but maybe overlapping, access patterns.

The design work performed in this project did not rely much on the specific details in figure 3.5. It relied on two key observations. First, that there will be sequences of jobs that access the same object set. Second, that object sets will not change gradually, gaining or loosing only a few objects, but that new object sets will be introduced abruptly.

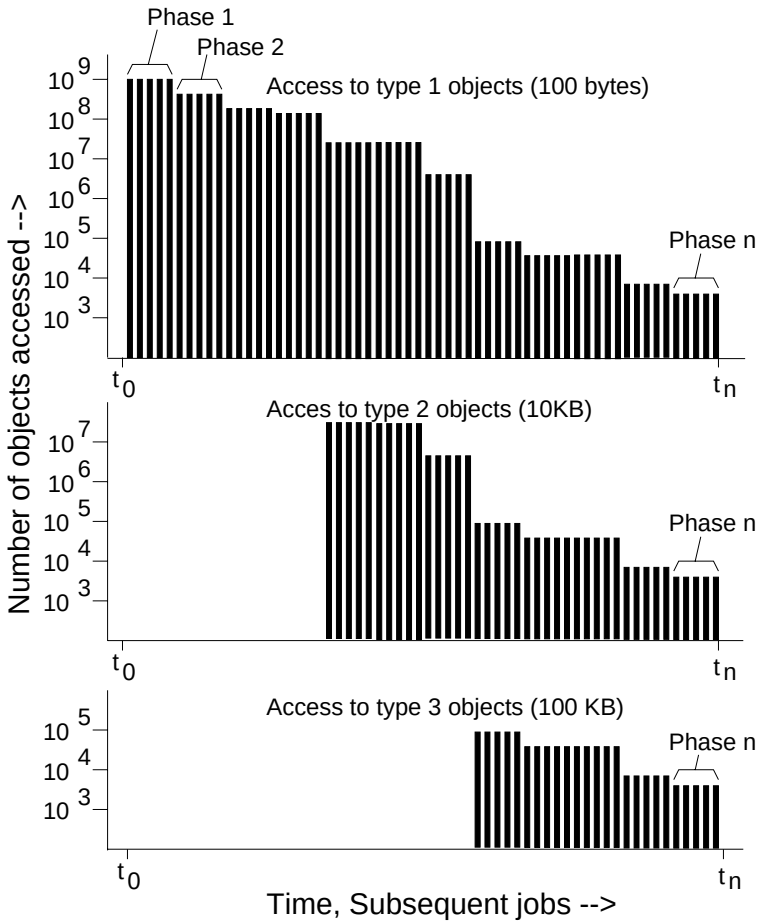


Figure 3.5: Numbers and types of objects accessed by subsequent jobs in a simple physics analysis effort. Time runs from left to right in all three bar graphs concurrently. In each of the three bar graphs, one bar represents the number of objects (of some type) read by one job. Different bars with the same position on the time axis belong to the same job.

3.5 Conclusions

This chapter discussed the data access patterns produced by physics analysis. The most important physics workload properties below are summarised below.

- Large database: the object store is exceptionally large, in the Petabyte range.
- Simple data model (figure 3.1): every event has a collection of physics objects in the database, these physics objects have a hierarchical structure with 1 MB of raw data objects at the bottom, and 100 byte tag objects at the top.
- Read-only nature of objects: modification of stored objects will be uncommon, if it happens at all.
- Independence of events: the absence of dependencies between events trivially

allows for massive parallelism, and creates important degrees of freedom in I/O scheduling.

- Large to very large jobs: jobs access thousands to millions of objects, Megabytes to Terabytes of data.
- Revisiting of large object sets: subsequent jobs will visit the same (large) object sets many times. Sets do not change gradually, but new sets are introduced abruptly.
- Use of cut predicates: a cut predicate is used to 'cut away' some of the events in the current event set to obtain a smaller event set for the next phase. These cut predicates can be very complex, and especially in the later phases of the analysis process, little can be predicted beforehand about the nature of the predicates.

As noted in section 3.2.3, for physics analysis on tag objects, there is a large body of related research and experience in the application area of data analysis on relational databases. Physics analysis workloads on non-tag objects, however, are atypical in that they differ in some significant ways from other database and I/O workloads that have been studied widely in industry and academia. Because of such differences, much of the existing research on high-performance I/O does not apply to the problem of optimising physics analysis on non-tag objects. Compared to transaction processing and CAD workloads, which have been researched extensively, the transactions (jobs) in physics analysis workloads are much larger, and they contain much less modification operations. The job size and the independence of events in physics analysis workloads make it possible to aim for obtaining sequential reading patterns on disk, rather than random patterns. This is generally infeasible for transaction processing and CAD workloads. However, because of the use of cut predicates, achieving sequential reading patterns on disk for physics analysis workloads will be more difficult than it is for file server workloads, another area where a lot of research into I/O optimisation has been done. Video on demand workloads [33] [34] have similarities to physics analysis workloads in their size and throughput parameters, but again there is nothing like a cut predicate in a video on demand system.

In the application area of scientific data analysis, the systems for the storage and analysis of large spatio-temporal datasets (for example satellite observation records), like EOSDIS [7] or Sequoia 2000 [35] [36] [37] [38], come perhaps closest to the size parameters for physics analysis in the LHC experiments. The data analysis jobs on such spatio-temporal systems generally examine subsets of the data which cover a small region of the whole spatio-temporal space occupied by the data set [36]. This means that the objects (data items) which are requested together are generally highly correlated, with a correlation that is known beforehand. This known spatio-temporal correlation can be exploited when optimising such systems, by building specialised multidimensional index structures [39] [40] [41] and by clustering the objects beforehand using a specialised clustering arrangement [38] [42]. In physics analysis workloads however, it is not possible to know beforehand which exact correlation between the objects (events) will be embodied in the event sets created while a physics analysis effort progresses. This lack of a high predictability means that specialised indexing and initial clustering optimisations (see section 7.5.2) are dramatically less applicable. Instead, more dynamic optimisation methods, which exploit correlations once they become

visible in the workload, take centre stage.

It can be concluded that non-tag physics analysis workloads do not fit the profile of any of the major high performance I/O application areas that have been studied widely in industry and academia. Nevertheless, there is still a lot of work that is at least related to the physics analysis application area, see sections 5.8, 6.8, 7.9 and 8.2.

This chapter discussed the nature of the demands made by the software layers above the storage manager on the services offered by the storage manager. The next chapter discusses the properties of the software and hardware layers below the storage manager.

Chapter 4

Layers below the storage manager

4.1 Introduction

The storage manager bridges the gap between the needs of physics analysis and the properties of the underlying software and hardware layers. This chapter discusses these underlying layers, as shown in figure 4.1. An important task of the storage manager is to perform I/O optimisations that are specific to physics analysis workloads. Because of this, a good understanding of the I/O performance characteristics of the underlying layers is a prerequisite for designing the storage manager.

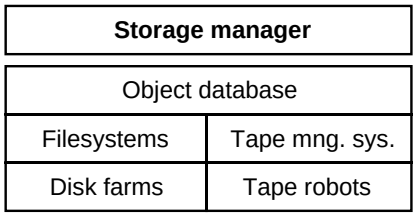


Figure 4.1: The storage manager and the layers below it.

Considerable project time was spent studying relevant performance properties of these layers, mainly by doing measurements on prototype implementations, but also by studying literature [43] [44] [45] [46] [47]. Apart from measuring the performance of current systems, literature on performance trends leading into 2005 was also studied.

This chapter first discusses the Objectivity database, which is the software layer directly below the storage manager. Then, the access to objects in cache memory, on disk, and on tape is discussed. The chapter ends with a discussion of possible future I/O technologies.

4.2 Objectivity/DB

As discussed in section 2.7, the Objectivity/DB object database management system [20] was chosen as the basis for all object storage prototyping efforts in CMS in the

1996–2001 timeframe. In early 1997, at the start of the designer's Ph.D. project, it was not yet known whether the use of an object database would introduce significant performance bottlenecks compared to earlier, flat-file based systems. It was also not known whether commercial object databases could scale to the needs of CMS in 2005. To settle these questions, a large number of tests were performed in the context of the RD45 collaboration (section 2.8). Some of these RD45 tests were performed as part of this project.

In April 1997, RD45 published a landmark report [17] that settled a large part of these questions. This report concludes the following. In essence, use of Objectivity for physics workloads does not introduce performance penalties compared to existing systems. Through its clustering directives, Objectivity allows for fine-grained control over physical data placement, so that optimisation strategies usable in a flat-file system can be adopted if necessary. RD45 identified Objectivity as the most suitable product for high energy physics needs, with Versant [48] as a commercial fall-back solution. Many other products explored did not have an architecture that would scale to meet high energy physics needs.

4.2.1 The ODMG standard

Objectivity is largely compliant with the ODMG standard [49] for object databases. The ODMG standard defines the applications programmer interface (API) for a compliant database. It does not cover performance or implementation internals. By using an ODMG compliant product, the CERN software gains some level of database vendor independence. The ODMG standard does not cover the whole interface of Objectivity, as is available to the application programmer. In particular, the performance-related Objectivity *clustering directives*, by which the programmer has control over physical data placement, are not covered.

From a C++ programmer's viewpoint, an ODMG compliant object database offers the service of creating and managing *persistent objects*. A persistent object has the property that, unless explicitly deleted, it continues to exist after the termination of the program that created it. Aside from that, persistent objects have all the features one can expect from a normal C++ object. Their class definitions can inherit from other objects, can have private and public data members, methods, and virtual functions. A non-persistent object can maintain references to persistent objects, and a persistent object can maintain references to other persistent objects. Compared to programming for a relational database, programming for an object database has the advantage that an object oriented data design maps naturally onto the object database facilities. There is no need for code that 'flattens' the object structure into something like relational database tables.

The ODMG standard defines a logical *storage hierarchy* for object databases. Individual database vendors can decide how to map this hierarchy to physical storage. The ODMG hierarchy is as follows, from the lowest layer to the highest:

- a *persistent object* is the smallest unit of storage visible to the programmer
- every persistent object must be stored in a *container*

- every container must be stored in a *database.*

Objectivity defines an additional layer on top of this:

- every database must be part of a *federation.*

The use of the term 'federation' by Objectivity differs from its usage in the broader database literature: the term 'federation of databases' usually implies a loosely coupled system made up of a number of different databases that may be implemented using completely different database products. An Objectivity federation is a strongly coupled set of databases, all of which are managed using the Objectivity product.

The sections below discuss how Objectivity maps the above logical hierarchy to physical storage.

4.2.2 Storage architecture

An Objectivity federation is represented on storage as a set of *database files*: there is one file for each database in the federation. The locations of all database files are maintained in a *federation catalog file.* Internally, Objectivity organises a database file as a set of *pages.* A page is a fixed-length contiguous block of space in the database file. The page size, from 2 to 64 KB, is set globally by the developer when a new federation is created.

The containers stored in a database are mapped to disjoint sets of pages in the database file. The persistent objects stored in a container are mapped to the pages of that container: objects larger than a page are broken up and mapped to multiple pages. This mapping happens transparently to the application programmer. Pages only become important in an analysis of application performance.

4.2.3 Fat database client architecture

Objectivity uses a 'fat database client' architecture. A database client, usually called a *client* for short in this thesis, is defined as a running process that accesses an Objectivity database by calling on the Objectivity applications programmer interface. Client executables are linked with a large Objectivity library that allows the client process to perform all major database operations by itself (figure 4.2). There is no central server that processes queries or serves data. All clients retrieve and process data by themselves. The only centralised component is a *lock server*, which manages the database locks for all running clients. The granularity of locking is at the container level in the ODMG hierarchy, not at the level of individual objects. Every running client maintains a TCP/IP connection to the lock server (figure 4.2).

In the Objectivity architecture, every persistent object, container, and database in a federation has an *object ID* (OID) that uniquely identifies it. Database clients open a federation at startup (they cannot open multiple federations), once the federation is open the clients can navigate all data in the federation without explicitly opening database files, containers, or objects: open operations are done transparently by the

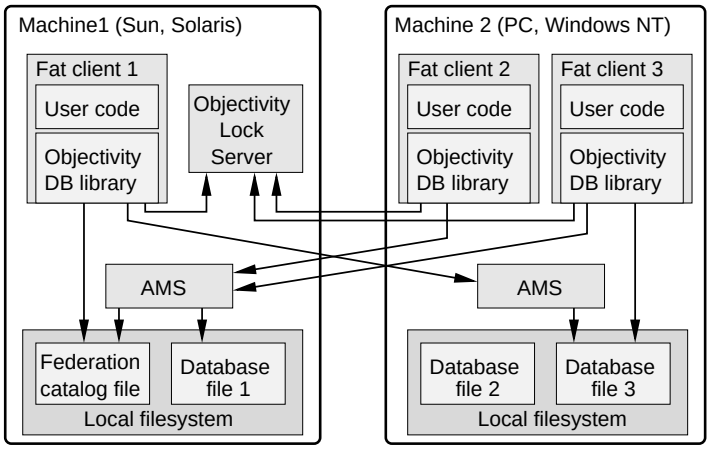


Figure 4.2: Components of the Objectivity/DB architecture: fat database clients, database files, the federation catalog file, the lock server, and AMS servers for remote database file access. The arrows represent TCP/IP connections and local filesystem access through the filesystem primitives.

Objectivity implementation. Read and write locking operations are done at the container level.

4.2.4 I/O architecture

Clients do reading and writing on database files in units of pages. All operations below the page level happen internally in client memory. Every client maintains an internal page cache, the size of which can be configured, to reduce database file I/O. Clients use the normal filesystem interface provided by the operating system to access database files on filesystems local to the machine on which they are running. To access database files on a remote host, a client contacts the so-called *Advanced Multithreaded Server* (AMS) that runs on the remote host. This AMS server is a program developed by Objectivity, which should be run on any machine containing database files that need to be accessed remotely. Clients contact an AMS over a TCP/IP connection using an Objectivity-specific protocol. The AMS uses the local filesystem interface to do I/O on behalf of the client. I/O via the AMS server again happens in units of database pages. If a remote database file is accessed by a client through an AMS, the file access pattern does not change.

Currently, there are a number of efficiency problems that make the use of remote database files in I/O-intensive applications very unattractive. First, the current generation of LAN technology typically delivers a throughput of 1 MB/s, which is low when compared to the 6 MB/s peak performance of a typical local disk. Second, the current AMS implementation has a number of problems that preclude scaling to high throughputs in multi-client scenarios even if the network is fast enough [6]. Product enhancements to eliminate these limitations are underway [6]. The prototyping efforts of this project, based on the current product, focused almost exclusively on measuring

the performance of accessing database files on local disks. Once AMS performance problems have been solved, the developed optimisation techniques for access to local database files should also be applicable to remote database access over a fast network, as the use of the AMS as an intermediate does not change the filesystem-level access patterns produced by the client.

4.2.5 Objectivity/DB future

Development of the Objectivity product is ongoing. The current product does not scale to supporting a multi-Petabyte object store like the one that CMS plans to have in 2005. However, it is expected that the remaining scaling limits will have been removed before 2005. Through the RD45 collaboration, the high energy physics community is actively providing feedback to the Objectivity developers, to make sure that the product will meet future physics needs.

From version 4 on, Objectivity has provided features for wide-area database replication. These replication features have not yet been extensively tested, though one project is performing such tests at the time of writing [18]. It is not yet clear whether the CMS replication strategies will be based on Objectivity-provided mechanisms.

CERN will make a final choice for a production database system at the end of 2001.

4.3 I/O on data cached in memory

Two kinds of in-memory caching can be distinguished when an Objectivity client is accessing data. First, the client can cache database pages in its own memory. The size of this cache is a configurable parameter. Access to objects on pages in the client cache is as fast as access to normal C++ in-memory objects. Second, a page may be in the in-memory filesystem cache of the operating system. If an object on such a page is accessed, and the page is not in the client cache already, the client retrieves this page through the filesystem interface, and puts it in its local cache. The Objectivity architecture does not include any type of server side in-memory caches, only client-side caches. The AMS server does not cache data internally.

The work in this designer's Ph.D. project focuses on disk and tape related I/O optimisations. It is aimed at optimising jobs that access datasets so large that they cannot (economically) be cached in memory. As such, a detailed study of the behaviour of in-memory caching was not made. In the prototype systems, the Objectivity client caches were configured to be small. Larger client caches would just waste memory without optimising performance.

For jobs where the dataset cannot be cached in memory, it would be efficient (in terms of memory usage) to disable filesystem caching too. However, on many of today's commercial operating systems, there is no mechanism by which a database client can disable filesystem caching of data read from some or all of the accessed database files. Such mechanisms are currently only known to be present in IRIX, the operating system on Silicon Graphics (SGI) machines, and in NTFS, the Windows NT File System. It is expected that mechanisms to disable filesystem caching will be more common in

future: they would also improve overall performance, by allowing for better memory utilisation, when the system is running multimedia applications that stream audio or video data from disk.

In performance tests on data sets that were about 1–3 times larger than the available memory, cases have been observed where attempts at filesystem caching by the operating system (Solaris or HP-UX) significantly decreased overall performance. In attempting to cache such datasets in memory, the operating system paged out parts of running programs, incurring large performance penalties when these had to be paged in again. It was observed that, when reading data sets much larger than the available memory, the internal cache management algorithms in the operating system (Solaris or HP-UX) would reach a steady state at one point, some time after the start of reading, after which such negative performance effects are absent. On the SPP-UX (HP Exemplar supercomputer) test platform, the filesystem cache was configured at a constant size. No adverse performance effects due to the filesystem cache competing for memory with the running applications were therefore seen. No detailed analysis of filesystem cache related performance losses on Solaris and HP-UX has been made. It is expected that this issue will have been solved on 2005 systems, probably by the introduction of explicit mechanisms to disable filesystem caching.

4.4 Disk I/O

This section discusses the performance characteristics of disk I/O through Objectivity, that is the performance when clients access objects in database files on local hard disks. A number of likely access patterns are considered.

CMS will use disk farms, that is large collections of disks interconnected in some way, to store data. A single disk farm will generally consist of a number of *disk arrays*. A *disk array* is defined as a set of disks which are interconnected to act as a single device, on which a single coherent filesystem can be created. Disk arrays containing some 5–20 disks are readily available on the market. The I/O interface bandwidths of such disk arrays are generally fast enough to support the I/O capacity of the disks. I/O busses never created bottlenecks in the tests that were done. In the disk performance discussion below, three classes of disk arrays are distinguished.

- In a *normal disk array* the filesystem space is split over all disks, in such a way that a contiguous set of blocks in the filesystem maps to a contiguous set of blocks on a single disk, or a contiguous set of blocks spanning one or more disk boundaries. Because of this filesystem contiguous to disk mapping, the performance characteristics of accessing a file on a normal disk array are generally the same as those for accessing a file located on a single disk. If heavy filesystem fragmentation destroys the contiguous mapping, the performance characteristics of a disk array are slightly better than those of a single disk, because of the increased seek (disk arm movement) parallelism made possible by having multiple disks. On a well-managed system, such heavy fragmentation will not occur in practice however.
- In a *striped disk array* the filesystem space is split over all disks in the array,

with subsequent blocks in the filesystem being assigned cyclically to subsequent disks. Except for very small files, the file contents are thus spread evenly over all disks in the striped array. When such a file is read sequentially, all disks are active at the same time, so that the I/O throughput is the sum of the throughputs of all disks. This is much better than in a normal disk array, where one obtains the throughput of the single disk holding the file.

- In a *RAID array* (more precisely RAID level 2 or higher), data is striped over the disks, but not in an exact 1-to-1 mapping. To protect against the failure of individual drives, some redundancy in the mapping scheme is introduced. Reading from a RAID array is generally as fast as reading from a striped disk array. Because of the redundancy scheme, writing is usually slower, the slowdown depends on the exact scheme used and the amount of dedicated hardware used to support it.

For every access pattern discussed below, the performance curve for a single disk is considered first, then it is discussed how this curve changes for different types of disk arrays.

4.4.1 Characterising the performance of modern disk drives

With memory becoming cheaper, modern hard disk devices [50] all contain enough internal RAM memory to hold the contents of several disk tracks. This memory is used to cache disk blocks before they are written, to buffer the current track contents as they pass under the read head, and as a buffer for read-ahead optimisations by the disk controller. As a result, modern disks show very smooth performance curves when workload parameters change. Older disks can show a large performance breakdown if, for example, the delay between subsequent I/O requests for a sector n and sector $n + 1$ is too large. If the delay is too large, then sector $n + 1$ will have just passed under the read head before the disk 'sees' the I/O request: in that case one will have to wait almost a whole platter revolution before sector $n + 1$ passes under the head again, and the I/O request is executed. To deal with such performance breakdown issues, a fairly precise parametrisation of the disk is required, as found for example in [43]. I/O bound programs that perform well on one particular older disk model could become very slow on another old disk model. Nowadays, with larger buffers smoothing out all events below the track level, such performance breakdowns are absent. For the remaining performance related disk parameters, there are no big discontinuities between disk models. Also (see section 4.4.7), these parameters are not expected to change fundamentally in the near future. For high-performance I/O, disk-specific optimisations are still needed, but it is not necessary anymore to tune or optimise for a single disk model.

4.4.2 Disk performance test platforms used

Considerable time was spent in the project doing disk-related performance tests. The primary goal was not to compile a complete catalog of performance curves. Rather, the goal was to formulate generic rules for achieving good performance. Performance

tests were used to discover such rules, and then to extensively validate them. As such, performance breakdown situations were studied less than situations in which performance was kept high.

Figure 4.3 lists all disk I/O performance test platforms that were used. The test reports below refer to these platforms by their identifiers.

Platform	Machine	Disks
SUNA	Sun, 4 CPUs	12 disks in normal disk array
SUNB	Sun, 6 CPUs	6 disks in normal disk array
HPA	HP, 1 CPU	single disk
HPB	HP, 4 CPUs	6 disks in RAID array
HPEX	HP Exemplar, 256 CPUs	4x16 disks in 16 striped disk arrays

Figure 4.3: Summary of disk I/O performance test platforms used.

The main testing and validation platform with respect to different disk access patterns was SUNB, a Sun Ultra Enterprise server, running SunOS Release 5.5.1, using a 2.1-GB 7200-rpm fast-wide SCSI-2, Seagate ST-32550W in a normal disk array, brand SPARCstorage. These disks can be considered typical for the high end of the 1994 commodity disk market, and the middle road of the 1996 market. HPEX, a HP Exemplar supercomputer, was used for extensive disk I/O scalability tests (see chapter 5). Details on the HP Exemplar are in section 5.7.1.

4.4.3 Sequential writing and reading

As seen in section 4.2.2, Objectivity maps objects to pages in a database file. This mapping behaviour was tested in detail, both through performance tests and system call tracing. It was determined that, if a sequence of objects is created in an empty container, these objects are mapped to subsequent pages in the database file, with these pages being written subsequently to the database file. Objectivity thus performs sequential write I/O through the filesystem interface when a large set of persistent objects is created in a single container.

If such a set of objects is read back in the order of object creation, the pages on which the objects are stored are usually also read back in sequential order, as expected. Sufficient conditions that need to be met for this to be the case have been determined and experimentally validated. The reading of objects in creation order causes sequential filesystem I/O if

1. the database page size is at least 8 KB (which is the default size), and
2. the objects stored are individually smaller than the database page size.

If both conditions are met, then reading objects in creation order achieves an I/O throughput close to the best possible disk throughput specified by the disk manufacturer, and equal to the throughput that can be achieved by any program doing sequential filesystem I/O.

The second condition above is the result of a bug in Objectivity, which has been reported to the vendor, and which may be resolved in a future version. The RD45 collaboration does not consider this bug to be a very serious one. The bad effects of the performance bug can be avoided by taking a large, say 64 KB, page size, and by mapping the (> 64 KB) physics objects (figure 3.1) onto a set of smaller objects, then storing those. The rest of this thesis only considers I/O use cases where both conditions above are met.

If a database file is accessed sequentially through the filesystem interface, this does not necessarily imply sequential reading on the disk hardware: this depends on how the database file contents are mapped to the disk. In tests that were performed in this project, mostly on SUNB and HPEX, it was found that modern filesystems do a good job at mapping files to sequential blocks on disk, unless the disk is almost completely full. Specific test results about filesystem mapping are not further shown in this thesis.

For all intents and purposes, considering Objectivity and operating system characteristics, the correct strategy for getting optimal (sequential) hardware performance is thus to simply read objects from containers in the creation order. One can ignore details like the mapping of objects to Objectivity pages, the mapping of containers to sets of pages in the database file, the mapping of database files to disk hardware, and the exact hardware parameters of the disk. This is an important result, that was validated on all platforms listed in figure 4.3. The result allows for the design work to proceed at a fairly high level of abstraction.

On a modern disk, the sequential reading throughput is of the order of 4–10 MB/s, and about equal to the sequential writing throughput. On a single disk, the throughput can differ by some 30% depending on whether the tracks holding the data are near the inside or the outside of the disk platter. On a normal disk array, throughput figures for a single client using the array on its own are equal to those of a single disk in the array (validated on SUNA, SUNB). If multiple clients read data that happens to be stored on two different disks in a normal disk array, then the throughput can be as high as that from the two disks combined (observed on SUNB). On a striped disk array, sequential reading and writing throughputs are those of all disks combined (validated on HPEX). With a RAID array, sequential reading proceeds at the combined throughput of all disks, writing speed can be lower, depending on the RAID array configuration and hardware characteristics (validated on HPB).

4.4.4 Random reading

Random reading is defined as a read pattern on a large set of objects, where the next object to be read is always randomly chosen. All objects in the set have an equal probability of being chosen next. Sequential reading represents the best case, random reading the worst case as far as performance is concerned. If an Objectivity client randomly reads objects from a large object set on disk, then performance can be very bad. As a first approximation, a current disk needs about 1/100th of a second to perform a seek operation to the page containing a randomly selected object. The seek operation consists of moving the disk arm with the disk read/write head to the disk track containing the page, synchronising the disk read-out electronics with the data stream on

the track, and then waiting until the page with the selected object appears under the read/write head. The disk then takes some additional time to actually read the page. This reading time is negligible, compared to the seek time, if the page size is below 16 KB. In summary, unless some of the objects are together on a single page, one can never randomly read more than 100 objects per second from a single disk.

Figure 4.4 gives a detailed overview of the performance effects of random reading. The figure plots the ratio between the speed of sequential reading and that of random reading for different object sizes. The figure shows a measured curve for 1994 disks in SUNB. An extrapolated curve for disks in the year 2005 is also shown. This curve was constructed from the 1994 curve, using a model based on an analysis of hard disk technology trends [21] [50] [51] [52].

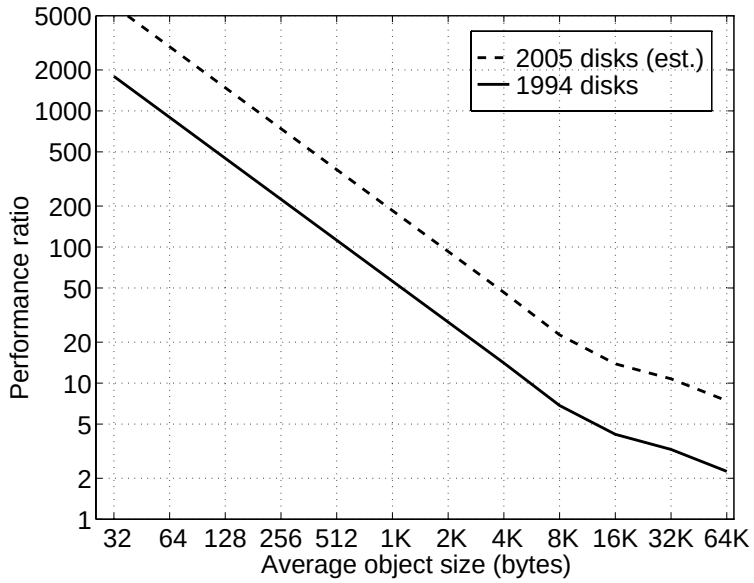


Figure 4.4: Performance ratio between sequential and random reading. The 1994 curve is based on measurements on SUNB. Performance effects were validated on SUNA and in literature. The 2005 curve is constructed from the 1994 curve, using a model based on an analysis of technology trends.

On all types of disk array, random reading by a single Objectivity client is as fast as random reading on a single disk. As the client waits issuing the next read request until after the current one is finished, only a single disk can be active servicing a read request at any point in time (validated on SUNB). If many clients perform random read requests at the same time, then, in the best case, all disks are able to perform seek operations in parallel. Multi-client random read scenarios were not analysed or measured in detail.

4.4.5 Selective reading

As discussed in chapter 3, the jobs in successive phases of a physics analysis effort read the physics objects for fewer and fewer events, as more cut predicates are applied to the original event set. If a container holds one physics object per event in the original event set, this leads to an object access pattern as in figure 4.5. Over time (downwards in the figure), the selectivity in reading objects grows. A container access pattern is called *selective reading* if the objects are read in the creation order, while some objects are skipped. The *selectivity* is the percentage of objects in the container that are read.

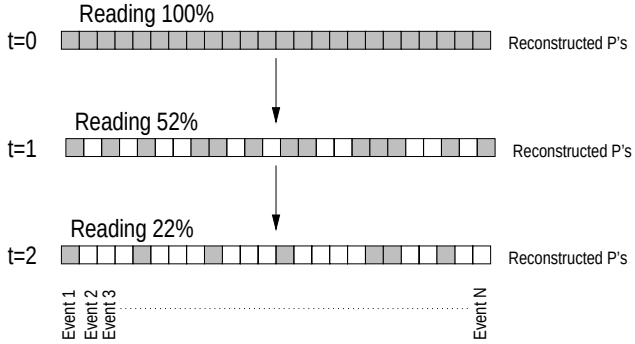


Figure 4.5: Increasing selectivity over time. The physics analysis effort advances as one moves down in the figure. The grey blocks represent objects that are being read by the jobs.

Considerable project time was spent in extensive performance tests for selective reading. This was done because the performance effects of selective reading are first, somewhat counter-intuitive, and second, of major importance to the design of the storage manager. From the point of view of communicating results to collaborators in the CMS software/computing project and the RD45 collaboration, this counter-intuitiveness implies that overwhelming evidence for the performance effects of selective reading is needed, before design decisions based on these effects are accepted as being valid and necessary. Measurements were also extensive because no performance results or models on selective reading for current disk hardware could be found in literature.

Figure 4.6 shows a typical performance curve, in this case for selective reading on a container with 8 KB objects, with an 8 KB database page size. The figure shows the wall clock running time needed to selectively read a subset of the objects in a container.

For selectivities from 100% down to some 20%, the running time does not decrease at all! This, initially surprising, lack of any speedup in reading a subset until the selectivity is low can be understood as follows. Consider the time it takes for the disk to perform a seek operation to the database page holding the next requested object if a few objects are skipped. When only a few objects are skipped, only a few pages are skipped (there is exactly one object per page), and there is a large probability that the requested page is on the same disk track as the previous page that was read. In that case the disk head does not need to move to get to the right position, and the fastest possible seek time equals the time needed to wait until the rotation of the disk platter

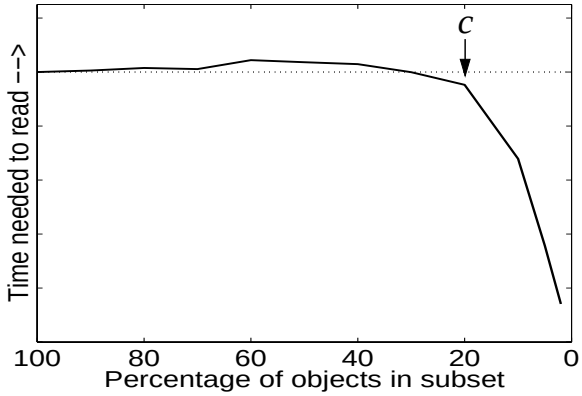


Figure 4.6: Wall clock running time needed to selectively read a subset of the objects in a container (for 8 KB objects). This curve plots measurements on SUNB. Performance effects were validated on SUNA, HPA, HPB, HPEX.

brings the requested page under the disk head. This rotational delay time is equal to the time it takes to read all intermediate pages.

For selectivities below 20% in figure 4.6, the running time improves. Below 20%, this measured time is actually the same as the measured running time needed to randomly read an object set of equal size.

Figure 4.6 shows a cutoff point c, which we define as the point at which running times start to get noticeably lower than the running time for reading 100% of the objects sequentially (the dotted line in the figure). The position of this cutoff point on the selectivity axis varies mainly as a function of the average size of the objects in the collection. The cutoff point c is visualised over a range of object sizes in figure 4.7. From figure 4.7 it can be concluded that for collections of small objects, selective reading may not be worth the extra indexing complexity over sequential reading and throwing away the unneeded data. A corollary of this is that one could pack several small 'logical' objects into larger 'physical' objects without getting a loss of performance even for a high selectivity.

Measurements on other types of disks, a study of disk technology, and a comparison of the disk specifications supplied by different manufacturers, indicated that the rather negative performance result for selective reading applies to all types of commodity disk hardware. For a normal disk array, the curve is the same as for a single disk in the array (validated on SUNA, SUNB). For striped and RAID arrays, both selective and sequential reading are faster, but the location of the performance boundary stays the same (validated on HPEX, HPB with 8 KB objects).

A study of disk technology trends showed that the curve in figure 4.7 will even move down, to smaller selectivities, in future. Of course, a radically new hardware innovation could change the trend by which this curve is moving down. The curve moves down because newer disks tend to have more data on a single track, leading to a higher probability that the page holding the next requested object is still on the same track.

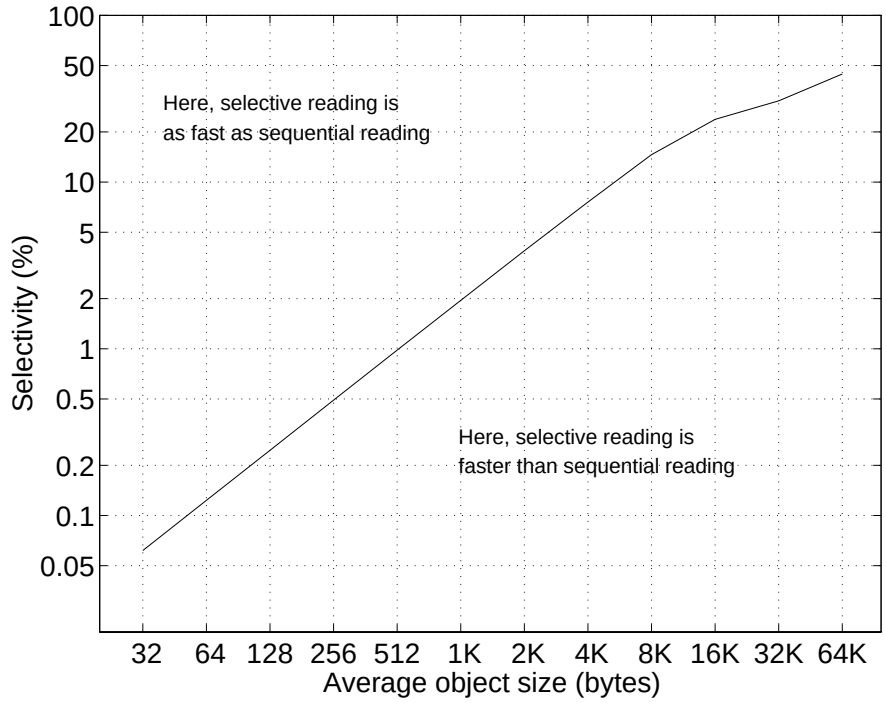


Figure 4.7: Selective reading performance boundary. This curve was constructed by applying a theoretical model to the 1994 curve in figure 4.4. Performance effects were validated on SUNA, SUNB.

Note that the commodity/desktop market, which is expected to drive innovation, is largely dominated by sequential reading: there is little market pressure to improve random and selective reading.

4.4.6 Parallel reading

Objectivity shows good scaling behaviour if multiple clients use the same disk or disk array in parallel. Below, some simple scaling studies that were performed at the start of the project are discussed. More elaborate scaling studies were performed later on, in validating the prototype designs (see chapter 5).

Figure 4.8 shows the aggregate throughput of 1 to 10 clients running in parallel, with each client sequentially reading objects from a different container, in the creation order. This test was run on SUNB, with all data located on a single disk in the disk array. The curves show no performance breakdown associated with the increased parallelism.

The operating system manages to avoid a performance breakdown in the parallel case by fulfilling multiple subsequent read requests from one client, before switching to a pending read request of any of the other running clients. By fulfilling multiple read requests before switching, the number of inter-collection seeks to be performed by the

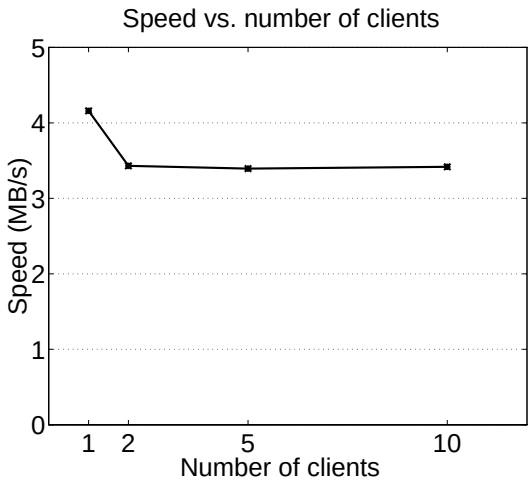


Figure 4.8: Aggregate I/O speed when running multiple clients in parallel. Curve plots measurements on SUNB. Performance effects initially validated with measurements on SUNA, later also on HPA, HPEX (see chapter 5).

disk is kept within reasonable bounds. This method of maintaining the performance is illustrated by the tests in figure 4.9.

Figure 4.9 plots two tests: one with a single client reading alone, and one with 10 clients reading concurrently, each from its own container. The tests were run on SUNB, using a disk array with 3-way striping, different from the array tested with SUNB in the rest of this chapter. In figure 4.9, each cross or dot plots the completion of a single read request. Time is on the x axis, and the y axis shows the sequence number of the object that has been read. For each of the two tests, the timing of a set of subsequent reads from a single client is plotted. In the first test, with one client running alone, individual reads happen quickly after each other, with only an occasional short delay. In the second test, with 10 clients running concurrently, short sequences reads are seen, with the same speed (slope) as for a single client running alone. These sequences are interspersed with longer delays, in which the disk array is serving read requests from the other clients.

The almost perfect scaling curve shown in figure 4.8 was not the result that was initially expected. It was expected that the curve would steadily drop, rather than staying horizontal, when going beyond 2 clients. The observed scaling result was very encouraging, and led to the conclusion that one can rely, to some extent, on the operating system to optimise the I/O traffic produced by multiple clients. The following design strategy was tentatively adopted: I/O reading patterns for each client were to be optimised separately, without introducing inter-client I/O optimisation mechanisms. Inter-client I/O optimisation would be left to the operating system.

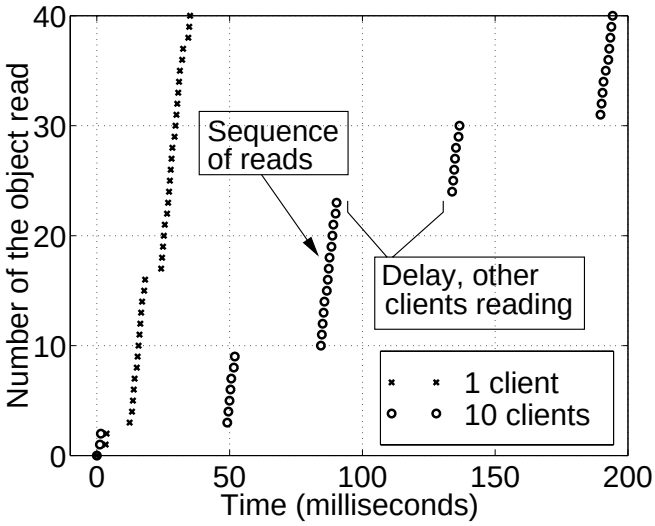


Figure 4.9: Timing of the completion of read requests in one client reading from a container. Two tests are shown: the first with a single client reading alone, the second with 10 clients reading concurrently. This figure plots measurements on SUNB.

4.4.7 Disk technology outlook

The hard disk market is big, and hard disks are a mature technology. Disk prices, in $ per GB capacity, have been dropping with a factor 1.6 every year [10], and this trend is expected to continue in the near future. RAM prices per GB are dropping faster than disk prices. A naive extrapolation of current curves yields that a GB of RAM capacity will remain more expensive than a GB of disk capacity until about 2010. Of course this extrapolation to 2010 has a very large error bar. But the 2010 result does make it reasonable to assume that in 2005, at CMS startup, RAM will not yet have replaced disks. Other serious long-term competitors for hard disks do not currently exist. In conclusion, without a major new technology breakthrough, one can expect that disks will still play an important role in 2005.

Disk I/O performance, in $ per MB/s, has only been improving with a factor of about 1.2 every year. I/O speeds are mostly improving as a side effect of the improvements in disk capacity. Capacity has been improved by improving the storage density (in bits/mm^2) of the disk platter. Suppose this density figure increases by a factor 4: this allows for twice as many tracks on a disk platter, each containing twice as many bits. Disk capacity quadruples, but with equal platter rotation speeds, the time to read a single track remains the same, so the I/O rate only doubles. Though disk rotation speeds have gone up in recent years, the major part of I/O speed improvements is due to increases in storage density. There are several obvious ways of improving disk I/O speeds with a few factors of 2: for example one could use the disk heads on multiple platters in parallel. However, it is unlikely that such techniques will be introduced in the near future. The disk market is very much driven by desktop systems, where disk

capacity per $, not disk speed per $, is the metric considered when comparing prices.

High performance computing is becoming more and more I/O bound. At CMS startup in 2005, CPUs will be about 100 times more powerful (in MIPS per $) [10]. It is expected that disks will only be about a factor 4 more powerful (in MB/s per $) at that point. Because of these trends, I/O optimisations become more and more important in high energy physics, which has traditionally been most concerned with optimising for CPU usage.

4.5 Tape I/O

Storing a large dataset on tape is some 10 times cheaper than storing it on disk [53] [54]. For massive datasets like those in high energy physics, the use of tertiary storage remains the only cost-effective option for the foreseeable future [55]. Tape storage has the disadvantage of introducing a major I/O bottleneck in the system: whereas CMS expects to have some 100 GB/s of disk I/O capacity in 2005, it expects to have only 1 GB/s of tape I/O capacity.

The 'industrial quality' tape drives in use for the storage of large scientific datasets, and for reliable backups in large computing centres, are very different, from 'consumer' tape drives, which are used for making occasional backups of the data on a single PC or a small LAN. They are different in price, storage density per tape, and reliability (mean time between failure). A typical industrial quality tape drive unit costs between $1000 and $60.000 [55], depending on the technology used. The cheaper (in $ per GB) tapes generally come paired with the more expensive drives. An industrial quality tape drive has a throughput of 3–15 MB/s. In $ per MB/s, this tape throughput is thus fairly expensive to very expensive when compared to disk throughput.

Typically, a tape cartridge used in an industrial quality tape system only has a single spool in it, on which the tape is wound. When the tape is mounted, inserted into the tape drive, the end of the tape in the cartridge is connected to a second spool in the tape drive itself. When seeking to a particular position on tape, the tape can be wound between spools at a rate of about 10 metres per second; the tape speed when reading or writing data is of course much lower. As there is only one spool in the cartridge, the tape always needs to be rewound completely (at about 10 meters per second) before it can be un-mounted. Seeking to a piece of data at the very end of the tape, after it is mounted, generally takes in the order of a minute. In a tape robot, it generally takes in the order of 10 seconds for the robot arm to move the tape between the drive and the shelve. A tape robot generally has a single arm to service many tape drives: contention on the robot arm is not generally an issue.

On a modern tape robot like the Storagetek Powderhorn used at CERN [56], a complete tape change operation takes about a minute on average. Most of this time is spent in rewinding the old tape before un-mounting, and advancing the new tape to the right position after mounting. The tape change time can therefore be improved most by making rewinding and seeking faster. It is considered infeasible [57] to make rewinding and seeking faster by spooling the tape faster. Mechanically, it is hard to improve much on the winding speed of 10 metres per second. There are two feasible ways of

improvement, and both are currently considered by tape manufacturers for future products. First, one can make tapes physically shorter. This need not necessarily lead to a lower capacity per cartridge, because storage densities on tape will increase in future. The second option is to use cartridges with two reels inside. This allows the tape to be unmounted without rewinding onto a single reel, and also lowers the average required seek time after mounting.

On the software side, the CMS collaboration plans to manage its tape robots with a commodity mass storage management software system, for example HPSS, the High Performance Storage System [58]. Such a mass storage system allows the tape robots to be accessed in terms of filesystem operations: data on the robots are organised in files. A single tape cartridge will generally hold many such files. If an I/O operation is requested on a file, the mass storage system 'stages' the file it first: the file is copied from its tape onto a disk farm that acts as a staging pool. The requested I/O operation on the file proceeds only after copying is complete: the operation is performed on the copy of the file that is on disk. If any writing is performed on this file, it is eventually copied back to tape, before being deleted from the disk farm to make space in cache replacement. Thus, when objects in an Objectivity/DB database file on tape are accessed, there is first a long delay for the file to be staged, and then the I/O characteristics of accessing objects in the database file are the same as for purely disk based database files.

To optimise tape I/O from the software side, one has to reduce mount and seek operations as much as possible, and maximise the amount of useful data in any file that is staged onto disk. Section 7.5.2 has a more detailed discussion of this optimisation problem, with respect to physics data.

4.5.1 Tape technology outlook

For completeness, the technology outlook for tape drives is briefly discussed below. Most of the material below is derived from [55] and [57].

In the tape drive market, one can see a split between cheap, relatively unreliable, tape drives in the consumer market, and the more expensive industrial quality tape drives and tape robots, which are used mainly for backup and archival purposes in large (corporate) computing centres. These industrial quality tape drives are also used to store large scientific datasets, in high energy physics or in other areas like climate modelling, astronomy, or earth observation through satellites. This scientific market segment is small, and has little influence on overall technology trends. The industrial tape drive market is developing only slowly, and prices remain relatively high compared to the hard disk or processor markets, which are driven by consumer needs. Opinions are mixed on the long-term viability of the industrial tape market. Some observers see a small, but stable market, reflecting a steady demand for industrial strength backup capacity, a demand which comes from buyers who are conservative by nature. Others see a downward trend in sales, and predict disappearance of the market within 5-10 years, though it is unclear from this prediction what would replace tape drives. Certainly, the need for backup and archival capacity will not disappear, in fact the amount of data that an average company maintains is growing steadily [5].

4.6 Possible new I/O developments and technologies

The CMS experiment was designed based on estimates for 2005 hardware, based on conservative extrapolations of current disk and tape trends. It is possible that new technologies appear before 2005, invalidating the conservative CMS predictions. Below, some possibilities are discussed.

- Consumer storage devices for multimedia type content may replace industrial tape technology in future. Writable CDs are much more expensive than the equivalent tape capacity. The new DVD technology is not yet considered to be a serious threat to the industrial tape market: the cost per GB of writable DVD storage is currently more than 10 times higher than that of tapes [55]. However, DVD is still a new technology and if prices for writable DVD disks go down a lot, it may become a serious contender to tape.
- So-called 'optical tape' technology could offer much cheaper storage capacity than the current magnetic tape systems. However, optical tape has not yet made the step to successful commercial products. Other systems, like 'focused ion beam' technology [55], are also in laboratory development, and could possibly make a successful step to market.
- The market for 'industrial quality' tape drives is driven by the needs of backup and archival systems, which are 'write once' and 'read almost never'. It is possible that in future, a new, cheap storage technology is developed that has a fast writing speed, but a slow reading speed. This technology would satisfy the needs of the backup and archival market, and could thus 'kill' tapes. Without mass storage products that can read at least as fast as tapes can read, CMS would find itself in a difficult position, as the physics analysis I/O profile is 'write once', 'read often'. Though it is unlikely that tape products disappear completely in such a scenario, they could end up being much more expensive than projected in the current CMS cost estimates.
- Disk technology is continuously being researched, and big breakthroughs in disk storage density are possible. In the laboratory, potentially better storage processes are developed quite often, a few to a dozen times per year depending on what one counts. The development of a new process is usually accompanied by some reporting in the technology media, with a calculation of disk capacities if the new process could be put into mass production. Breakthrough processes that actually reach the mass production stage are significantly less common than these media reports. A breakthrough, or a combination of breakthroughs, which improves storage density with a factor of 10 over the existing trend would allow disks to replace tapes as the preferred mass storage technology.
- If RAM prices per GB drop below disk prices, which is unlikely to happen before 2010, then one may see a disappearance of hard disk farms, leaving only RAM (with battery backup or an uninterruptible power supply) as the commodity storage medium of choice. Backups, archiving, and very large datasets would be supported by tape or optical disk robots.

If tape technology disappears, it will likely be replaced by disk technology, magnetic or optical. The disk based optimisation techniques developed in this thesis are not limited

to magnetic disk, but valid for any storage device with a rotating platter and movable read head. More generally, until purely solid-state (RAM) technology replaces all mechanical mass storage products, there will remain a gap between sequential and random reading from mass storage devices, and some of the reclustering techniques that were developed will remain applicable.

4.7 Conclusions

This chapter discussed the software and hardware layers underlying the storage manager. In designing this storage manager, a good understanding of the I/O performance characteristics of these underlying layers is a prerequisite.

Conditions have been identified under which the maximum, sequential, disk performance can be achieved using Objectivity: the objects in a container should be read back in the order of creation. The disk performance degradations connected to random and selective reading patterns have been discussed. Random and selective reading perform worse, often much worse, than sequential reading. In general, for any set of objects which are needed together, the further they are apart on a disk, the longer it takes to read them all. As a general principle, a good clustering policy should therefore aim to put objects which are needed together close together on disk.

Performance degradations connected the non-sequential reading patterns can be very high, especially when reading small ($<$32 KB) objects. Thus, when the application of new cut predicates in a physics analysis effort creates selective reading patterns on small objects, it becomes very attractive to recluster such small objects to re-create sequential reading patterns. The very large ($>$factor 10) performance losses associated with randomly or selectively reading small objects implies that considerable ($<$ factor 10) overheads are still acceptable when designing optimisations that can eliminate these performance losses. This observation, which was made very early in the project, led to a focus on investigating potential reclustering optimisations.

For objects larger than 64 KB, performance losses for random or selective reading from disk currently are no more than a factor 2 (see figure 4.4), so currently clustering optimisations are not essential for such objects. The 2005 curve in figure 4.4 shows however that the importance of good clustering, even for very large objects, only increases in future.

The design project is about optimisations for systems running in 2005. The optimisations focus on the case where system parameters are predicted by conservative technology estimates. This is in line with the strategy of doing research to minimise overall risks. Though one may hope for the best case, that is technology developments that eliminate the need for complicated optimisations, it is better to plan for a more conservative scenario. If technology developments make it economically feasible to replace tape by disk in 2005, then the tape-specific optimisations developed in chapter 7 are not needed. However, such a scenario does not invalidate the current reasons for developing tape-specific optimisations.

Considerable project time was spent on software and hardware layer performance tests, and on investigations into technology trends. The goal, however, was not primarily

to compile a complete and detailed overview of software and hardware parameters. Rather, the goal was to find sufficient conditions under which the performance of the layers below the storage manager would be good and predictable.

Chapter 5

Basic storage management policies

5.1 Introduction

The preceding chapters described the layers above and below the CMS storage manager. This is the first chapter about the storage manager itself. The storage manager is best seen as a software artifact that implements a *set of interrelated policies.* Some of these policies govern management at a very coarse-grained level, for example the placement of data on different disk farms. Others policies govern fine-grained actions like the reading of individual physics objects. This chapter describes the basic policies developed for the CMS storage manager. More advanced policies, which govern dynamic reclustering actions, are described in the chapters following this chapter.

The design process leading to the basic storage management policies was an iterative one, in which tentative design decisions were followed by validation studies. Decisions were based on the properties of the layers above and below the storage manager, as recorded in the preceding chapters, and on a study of policies in existing systems.

This chapter does not give a historic account of the design process. Instead, the policies are presented first, from coarse-grained to fine-grained, together with some considerations that led to their development. Then the validation work is discussed. The chapter ends with a brief discussion of related work.

5.2 Separation into chunks

The following two policies are used in all current large physics analysis systems [59] [60] [27]. The storage manager also uses them, without modification.

- The set of all events is partitioned into fairly large *chunks.*
- Farming, both for disks and CPUs, is done at the chunk level.

A chunk is a set of events. The introduction of chunks and farming is inevitable. It reflects the fact that the offline storage system is made up of multiple discrete hardware units, like disk arrays, tapes, and CPUs. Mapping to these requires a partitioning of the data volume and the system workload. The chunk system also represents a natural

separation of concerns. It separates most coarse-grained management issues, which are solved above chunk level, from fine-grained issues, which are solved below it.

5.3 Policies above the chunk level

This section discusses the CMS storage manager policies above the chunk level, along with some possible refinements. The issues above the chunk level are largely considered to be solved in the high energy physics community. The new work in this designer's Ph.D. project was mostly concerned with management issues below the chunk level, discussed in section 5.4 and onwards.

5.3.1 Creation of chunks

New chunks are created, and added to the offline system, continuously while the detector is running. The easiest scheme to create chunks would be to put the events occurring in subsequent time intervals in subsequent chunks. A more complicated scheme is used, however, to optimise the performance of later physics analysis jobs.

Data is buffered before being put into the offline system. The process that moves the data from the buffer into the offline system is responsible for allocating events to chunks. All events acquired over a short running period (in the order of a day) are examined and then partitioned into some 10 to 100 chunks. The goal is to optimise the assignment of events to chunks, so as to minimise the number of chunks that any future analysis job will need to access. To perform this optimisation, the physics properties of the events are examined, and events with similar properties are put in the same chunk. The underlying assumption is that analysis jobs always examine events whose properties are similar in some way. See [25] for a discussions of a chunk assignment policy, 'physics streams', used in many existing physics experiments. More advanced policies, possibly leading to still higher efficiency, are being researched [61]. Section 7.5.2 discusses this optimisation problem in some detail, in the context of placing chunks on tape.

The process that assigns events to chunks, and enters them into the offline system, can also assign the event IDs to the events. This makes is possible to have the event ID internally encode a chunk ID and a sequence number inside the chunk. Such structured event IDs allow for efficient lookup operations. They were used in most of the prototyping activities connected to the storage manager. Structured event IDs are not absolutely required: other mechanisms with equal lookup efficiency could be built. For example, the chunk numbers belonging to the event IDs could be cached inside the representation of the event set. The CMS storage manager design assumes that locating the chunks that are 'hit' by a given event set is a fast operation. The exact implementation mechanism is left unspecified.

5.3.2 Farming policies

A disk array, as defined in section 4.4, is a set of disks interconnected to act as a single storage device. Barring dramatic technology developments, the CMS central data processing facility will likely contain many such disk arrays. Figure 5.1 shows a likely hardware configuration, under the conservative assumption that network capacity will be too expensive to interconnect all disks and CPUs symmetrically.

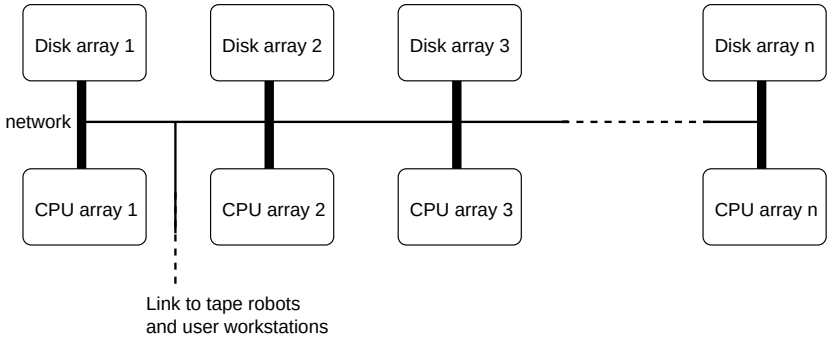


Figure 5.1: Massively parallel system with highly connected Disk/CPU array pairs.

To achieve locality of access, the storage manager uses the policy that all physics objects of the events in a single chunk are located on the same disk array. Physics analysis code that processes events from that chunk should be run on the connected CPU array.

To execute a physics analysis job over an event set S, containing events in the chunks $C_1, \cdots, C_n$, n chunk level subjobs are scheduled on the CPU arrays connected to the disk arrays holding the corresponding chunk data sets. Each subjob visits the events in S that are in its corresponding chunk. Subjobs are executed inside Objectivity/DB client processes that run on the CPU farms. See section 4.2.3 for a discussion of Objectivity clients. Note that the execution of n subjobs do not necessarily map to the execution of n client processes. A single client could run many subjobs sequentially. Conversely, a single subjob could also be parallelised to run on many clients. The exact mappings used will reflect hardware characteristics of the year 2005, and the chosen chunk sizes. These mappings are not part of the CMS storage manager design.

5.3.3 Load balancing

There will be many more chunks than disk/CPU array pairs. By assigning and re-assigning chunks judiciously to the different pairs, the system load can be balanced among them. The assignment schemes used in current physics experiments are fairly straightforward. Algorithms that re-assign chunks to dynamically re-balance array loads are readily available in literature, see for example [62] [63]. Such automatic re-balancing of disk resources is not used in currently deployed physics analysis systems, but it is an option for the future.

5.3.4 Chunk size

No detailed investigations have been made into the optimal chunk size. The optimal size depends a lot on hardware and workload characteristics for the system in 2005. Both are so uncertain that nothing definite can be said about the optimal chunk size now. For successful farming and load balancing, a certain minimum number of chunks is needed: this puts an upper bound on the chunk size. On the other hand, the chunks should not be too small. If chunks were so small that the average job would only visit, say, 10 events in every chunk, then the per-chunk overheads connected to starting and finishing subjobs would come to dominate the system.

5.4 Basic policies below the chunk level

This section starts the discussion of CMS storage management policies below the chunk level, which takes up most of the remainder of this chapter.

5.4.1 Type-based clustering

As discussed in section 3.2, every event in a chunk has many different physics objects associated with it. As seen in section 3.2, different objects for the same event are identified by their *type*. The following type-based clustering policy is used.

- A chunk is a set of events. Every event has different physics objects associated with it, distinguished by their type. The physics objects of all events in a chunk are partitioned into sets according to their type. Every set of physics objects of the same type within a chunk is clustered and managed separately.

The storage arrangement implied by this policy is shown in figure 5.2. The boxes represent coherent storage domains for each type. Recall that not every type necessarily has an object for every event: in figure 5.2, only some of the events have a 'reconstructed P' object.

The type-based clustering policy was inspired by the observation that physics jobs usually only read a single object of a single type, or a few objects of a few types, per event (section 3.4). By clustering all types separately, the storage space of unneeded types is never 'hit'.

The type-based policy implies an important separation of concerns: it mandates that the physics objects of each type should be managed separately, disregarding access to all other physics objects. This separation of concerns was a major step in the design of the CMS storage manager. Because of its importance, this design step is discussed in detail below. First, the advantage of applying this separation of concerns is, of course, that it yields a major simplification of the clustering problem that is left. Second, it should be recognised that there is a major risk associated with applying this separation: it could steer the design process into a dead-end direction, where no acceptable solution to the remaining design problems can be found. It is not immediately obvious that an efficient storage manager design can be made if the type-based clustering

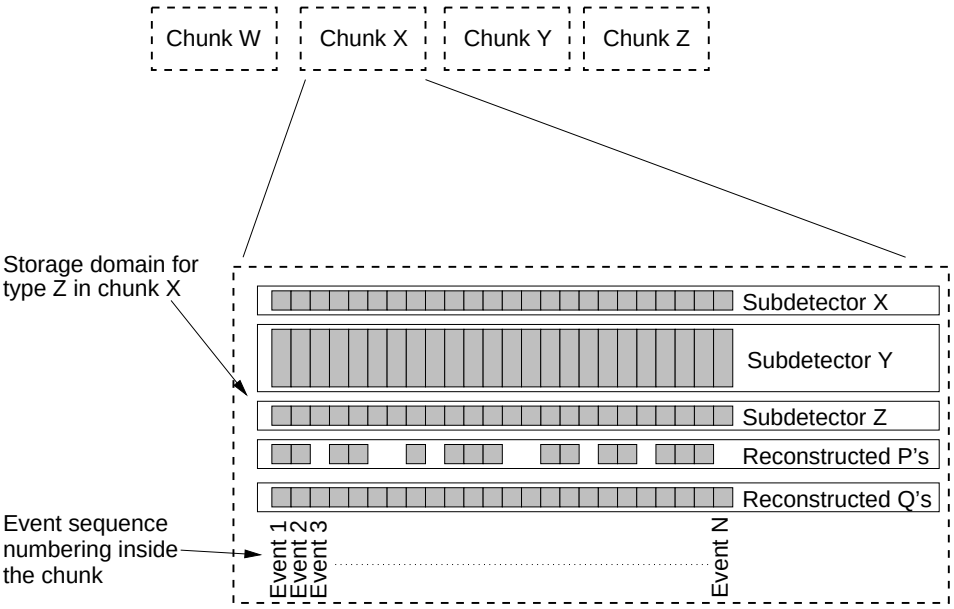


Figure 5.2: Type-based clustering of the physics objects in a chunk. Every solid box represents a separate storage domain.

separation of concerns is applied: consider the following. Some jobs read multiple physics objects of different types for every event. The type-based policy implies that these different objects are clustered separately, far apart, even though they are needed together. But in section 4.7, it was observed that to achieve good I/O performance, a clustering policy should aim to put objects which are needed together close together on disk. The type-based policy goes directly against this general principle. It could therefore well lead to unsolvable I/O performance problems.

At the time that the type-based clustering design decision was made in the project, the risk of encountering unsolvable performance I/O problems because of it was estimated to be low. The main reason for this favourable estimate was that, earlier, an unexpectedly good performance result was obtained for parallel reading by multiple clients, each client reading a single container (see section 4.4.6). With the risks estimated to be low, the design path defined by the type-based policy was followed further. The gamble payed off: a feasible CMS storage manager design based on type-based clustering could indeed be developed. As discussed in section 5.6.1, some performance problems in applying the type-based clustering policy were encountered at first, but it turned out that these problems could be solved.

5.4.2 Fixed iteration order

Another important policy is that of having a fixed iteration order.

- When a subjob iterates over all or some of the events in a chunk, this is always done in a fixed order, called the *iteration order.* On chunk creation, the events in

the chunk all get a sequence number: the iteration order is the order of increasing event sequence number.

This constraint on the iteration order affects all layers above the storage manager. However, it is easy for physics analysis software to live with this constraint. With respect to physics calculations, all events are independent, so any iteration order is equally good. Existing physics analysis systems all use a fixed iteration order. There are two important reasons for preferring a fixed iteration order.

The first reason is that many physics algorithms need to access detector calibration constants while processing events. These calibration constants are time dependent: some constants can change every minute or so while the detector is running. The constants are stored in a calibration database, keyed on the starting time of their validity interval. To make access to this database efficient, the events should be visited in strict chronological order. This allows calibration database access to be implemented as follows. For the first event, all calibration constants at the time index t at which the event occurred are retrieved and put in memory. For all subsequent events, these in-memory copies then only need to be updated occasionally with newer values as the time index advances along with event iteration. The lookup and retrieval workload on the stored calibration constants is thus kept to a minimum. More information about calibration databases can be found in [64] [65] [66].

The second reason for a fixed iteration order is derived from disk performance considerations. As seen in chapter 4, sequential and selective reading are usually much faster than random reading. A predictable, fixed iteration order helps the storage manager to achieve sequential and selective reading. Objects can simply be clustered in the iteration order. Without a known iteration order for future jobs, the clustering problem would become much harder.

In conclusion, the iteration order should be fixed, and it should be the time order in which the events occurred inside the detector.

5.5 Object collections

An object collection, or collection for short, is a set of stored physics objects. The collection concept is a result of an important separation of concerns.

- A separation of concerns is made between the policies governing a static clustering arrangement, and the policies for reclustering, that is policies for transforming one clustering arrangement into another.

The collection encodes the policies for a static clustering arrangement. These are as follows.

- A collection is a set of stored physics objects. All these objects have the same type, and they belong to a set of events in the same chunk.
- The physics objects inside a collection are statically and permanently clustered in the iteration order of their corresponding events. Access to the objects in a collection is therefore always in the order of iteration.

The relation between collections, types, events, and chunks is shown in figure 5.3.

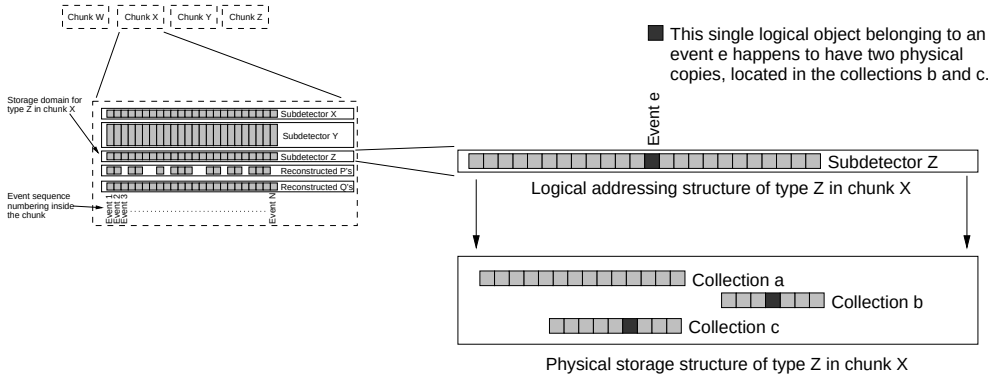


Figure 5.3: The relation between collections, types, events, and chunks. The left hand side of the figure is the same as figure 5.2.

The collection encapsulates the strategy, discussed earlier in section 4.4.3, of storing objects in an Objectivity/DB container and always reading them back in the storage order. This strategy leads to optimal hardware I/O performance, that of sequential reading, if all objects in the collection are read. If only some of the objects in the collection are read, the collection policies lead to selective reading (section 4.4.5), which is the optimum I/O strategy for sparse access to statically clustered data.

Collections also implement the following policy, which is connected to the policy in section 5.4.1 of managing and clustering different types of physics data separately.

- When iterating through its events, a subjob can read physics objects from multiple collections. The implementation of the collections ensures that this can be done without significant performance overheads.

In other words, the time needed for reading a set of objects, accessing multiple collections at the same time, should not be much higher than the time needed to read the same set of objects, accessing these collections individually, one after another. The implementation mechanisms used to ensure this are discussed in section 5.6.1.

5.5.1 Use of collections in reclustering

The following policy governs reclustering.

- To recluster some physics objects of a single type in a chunk, these objects should be re-distributed over a new set of collections.

Figure 5.4 shows some sample reclustering operations, as envisaged when collections were initially designed. Of course, many variations on this theme are possible: the policy of implementing reclustering by re-arranging objects over collections still leaves many degrees of freedom. The specific reclustering policies and operations used by the storage manager are discussed in the chapters following this one.

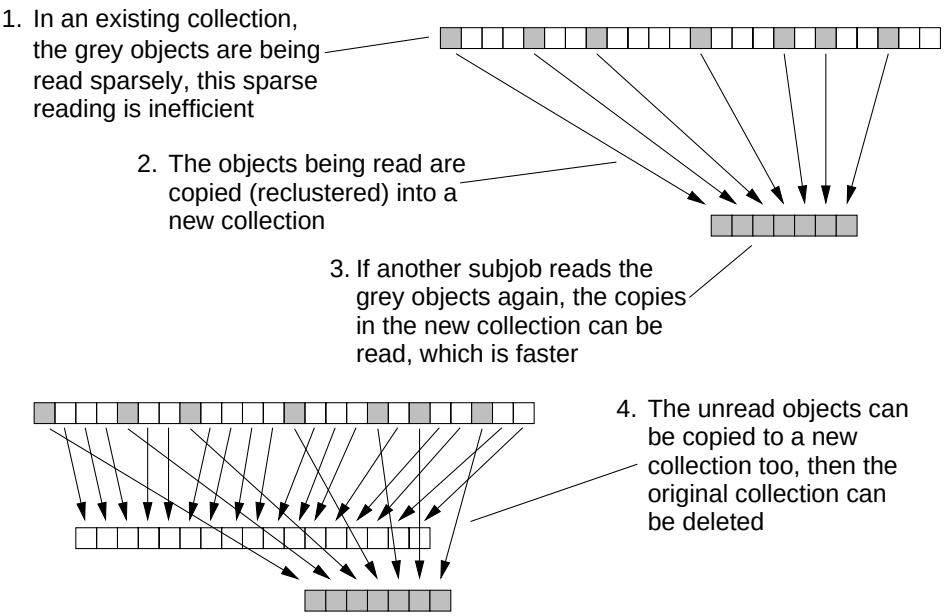


Figure 5.4: Sample reclustering operations. The horizontal bars of boxes represent collections, every box is a physics object.

Note that the operations in figure 5.4 imply that many stored copies of the same physics object can exist at the same time. If a stored object were modified, the storage manager would have to ensure that all stored copies are modified. This synchronisation of copies would add some complexity. However, as outlined before in section 3.2.4, there is no great need for modify operations on stored individual physics objects. Modification operations were therefore not considered in this project.

- The storage manager does not support modify operations on stored physics objects: once stored they become read-only.
- By extension, collections also do not support modify operations on the objects they store.

5.5.2 Operations on collections

Collections shield their users from many of the details concerning the use of Objectivity/DB and the system layers below it. Reflecting the storage management policies above, a collection object only has to offer a very limited set of operations to its users. The exact operations that are needed were identified during the design phases that refined reclustering policies (chapters 6 and 7). These operations are listed below.

- **Creation and filling.** A collection can be created on a specific disk array, and then be filled with physics objects. On filling, the physics objects must be passed to the collection in the iteration order of the corresponding events. With each object, the corresponding event ID must be passed too.

- **Index access.** A filled collection can be queried for its size, both in MB and in the number of physics objects stored. A list of the event IDs corresponding to the stored physics objects, sorted in iteration order, can be accessed.

- **Access.** The physics objects stored in a collection can be accessed through a *collection iterator*. Only access in the fixed iteration order is supported. To obtain the next physics object of interest, the ID of the corresponding event can be passed to the iterator. A handle of an in-memory copy of the object is then returned. This system of iteration by event ID allows the collection to perform special I/O optimisations in the case of sparse access. The iterator also supports the more traditional 'get next stored object' form of iteration.

- **Speculative access.** Any event ID can be legally passed to the collection iterator. If the event ID has no corresponding physics object stored in the collection, the iterator returns a *NULL* handle. This speculative access facility can be convenient in implementing software components that access many collections. It often makes it unnecessary for such components to keep track of exactly which physics objects are in which collection.

- **Deletion.** A collection can be deleted, this also deletes all physics objects contained in it.

- **Migration to tape.** Migration to tape is supported through collection copy operations. A collection can be copied, as one unit, to tape. A copy on tape cannot be accessed directly, but it can be copied back to a new collection on a disk array. This new collection can then be accessed in turn. More details about collection migration are in chapter 7, where the tape handling mechanisms of the storage manager are discussed in full.

The above operations are sufficient to support all storage manager prototypes discussed in this thesis.

5.6 Implementation of collections

Collections were implemented in this project as part of several storage manager prototypes. Much of the implementation is fairly straightforward.

It was already mentioned in section 5.5 that collections encapsulate the strategy of storing objects in an Objectivity/DB container, and always reading them back in the storage order. On creation of a collection, an Objectivity container is therefore created. Filling the collection amounts to placing the subsequent physics objects passed to the collection in the container. This placement is controlled by the use of Objectivity clustering directives. Objectivity containers have a certain maximum size. To create collections larger than this size, a *multi-container* strategy developed elsewhere in RD45 can be used [67]. If the current container is nearly full, a new one is created and writing continues there.

Indexing of the objects inside the collection is straightforward. As the collection is always traversed in iteration order, the index does not have to support fast random lookup operations. The collection index is simply a list of pairs

(Event ID , Objectivity/DB object ID of the corresponding stored physics object).

The Objectivity/DB object ID can be used by the collection implementation to retrieve the actual physics object data. The list of pairs is sorted on the event ID, in iteration order. This allows the collection iterator to look up the next object needed from a collection in, on average,

O(number of objects in the collection/number of objects retrieved from the collection)

time. The collection iterator simply advances a 'lookup cursor' over the list of pairs, keeping in step with the progress of event iteration. Use of this technique assumes that two event IDs of the same chunk can be quickly compared to find out which one is earlier in the iteration order. This is the case for the event IDs in all prototypes built in this project.

5.6.1 Read-ahead optimisation

Most of the collection implementation is straightforward. The only unusual part of the implementation is the *read-ahead optimisation*. This optimisation is therefore covered in depth below. The read-ahead optimisation is needed to get good I/O performance when a single subjob is accessing many collections concurrently.

Figure 5.5: Reading two collections: logical pattern (left) and preferred physical pattern (right).

Consider a subjob, running inside a single Objectivity database client, that reads from two collections while iterating over its events. Such a subjob has a logical object reading pattern as shown on the left in figure 5.5. For every subsequent event, first one physics object is read from the first collection, then another physics object from the second collection. If this reading pattern were performed directly on a disk array containing the two collections, then the I/O workload would be completely dominated by the 'long seeks' between these collections, in which the disk arm moves back and forth between the two areas on disk where the collections are stored.

To achieve near-sequential throughput accessing the collections, the logical pattern on the left in figure 5.5 needs to be transformed into the physical pattern on the right: here long seeks are infrequent and most time is spent in sequential reading. In prototyping tests it was found that this transformation was not performed by any layer below the storage manager: neither by Objectivity/DB, nor by the operating system (both SunOS and HP-UX were tested), nor by the disk hardware, for various tested commodity

disk brands. It was therefore decided to implement this transformation by adding the read-ahead optimisation inside the collection iterators. This optimisation causes the collection iterators to read objects from the collection in bursts of typically a few hundred KB. The objects are buffered in the Objectivity/DB database client cache until they are needed by the subjob.

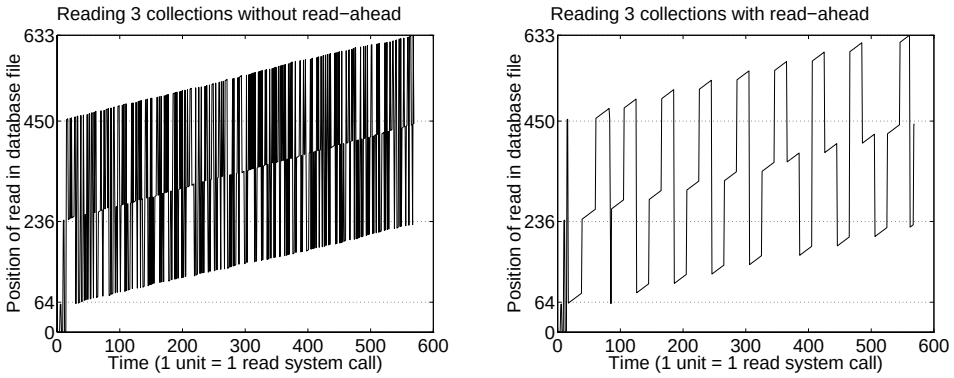


Figure 5.6: Read patterns produced by the database.

Figure 5.6 shows the effect of the read-ahead optimisation at the operating system interface. The left hand graph plots the `read()` system calls performed by an Objectivity/DB client that runs a subjob reading from 3 collections in its event loop, without the read-ahead optimisation. The right hand graph shows the calls for the same job, with the read-ahead optimisation in place. As is obvious from the graphs, the read-ahead optimisation reduces the number of long seeks at the operating system interface (vertical lines in the graphs) dramatically.

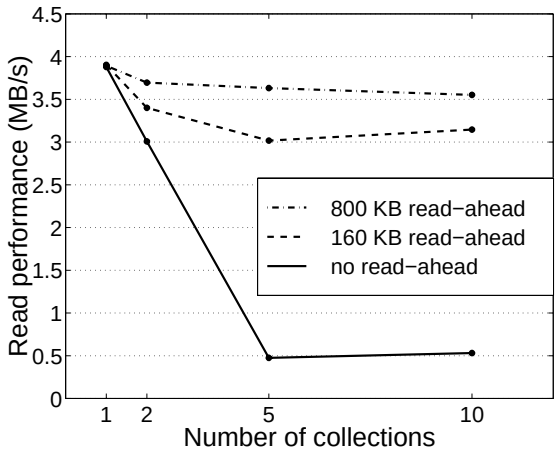


Figure 5.7: I/O scalability of one subjob reading multiple collections of 8 KB objects from a single disk. The curves plot measurements on SUNB.

The scalability effects of the read-ahead optimisation are shown in figure 5.7. Without the read-ahead (bottom curve), the performance drops dramatically as more collec-

tions are read from during iteration. With the read-ahead, the overheads (performance losses) associated with reading multiple collections are kept very low. The two topmost curves show the read-ahead scalability for two different read-ahead buffer sizes: 160 KB and 800 KB per collection. With a larger read-ahead buffer, the number of long seeks between the collections is lower, causing the performance to stay more closely to that of reading a single collection.

Though the measurements in figure 5.7 show that the read-ahead optimisation works in keeping the performance high, these measurements do not directly show that the read-ahead optimisation achieves this in the expected way, by reducing the number of long seek operations. Maybe the read-ahead optimisation instead corrects a different performance breakdown problem, somewhere above the disk hardware level. To validate the assumption that the read-ahead optimisation indeed works by reducing the number of long seeks, this assumption was used as the basis of a simple mathematical model for predicting the performance effect of the read-ahead optimisation. The predictions of this model were then compared to the actual measurements from figure 5.7. This comparison is shown in figure 5.8: the model and the measurements match quite closely, so it is concluded that the read-ahead optimisation indeed works in the expected way, by reducing the number of long seeks.

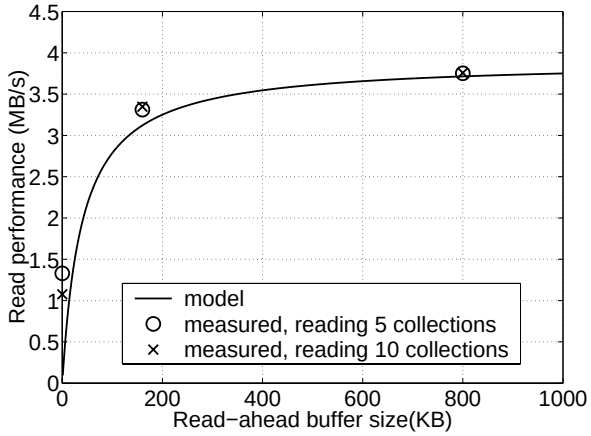


Figure 5.8: Comparison between the predictions of a simple mathematical model for the performance effect of the read-ahead optimisation, and the measured performance effects (for reading 5 and 10 collections) in figure 5.7. The two measurements plotted at a read-ahead buffer size of 0 are the two 'no read-ahead' measurements for reading 5 and 10 collections in figure 5.7. Projected onto the curve, these two measurements give an apparent read-ahead buffer size of 15–20 KB. This is larger than the database page size and the filesystem block size, which are both 8 KB. So apparently, an optimisation mechanism below the level of Objectivity/DB, probably read-ahead caching in the disk controller, is creating a very small read-ahead buffering effect, limiting the number of long seeks to about one for every two database pages read.

The simple mathematical model in figure 5.8 is constructed as follows. It is assumed that the time needed by a job to read n collections concurrently, using the read-ahead optimisation, equals the time needed to read all these collections sequentially after each other, plus the time needed to perform all long seek operations between the collections. The first time contribution is calculated by combining the size of the collections with the 3.9 MB/s speed for sequential reading of a single collection, as measured the tests of figure 5.7. For the second time contribution, the expected number of long seeks is multiplied with the time needed for a single long seek. The time for a single long seek was measured earlier to be 1/100th of a second for SUNB (see also section 4.4.4). With a read-ahead buffer size of n KB for every collection, the expected number of long seeks is one long seek after every read-ahead operation, so one long seek for every n KB that is read. (This latter model assumption is a slight simplification from reality: in practice, there is a small probability that the next read-ahead operation will be on the same collection as the last one, so that no long seek is needed.) Taking all this together, the curve in figure 5.8 can be computed. As noted, this model curve fits the measurements quite closely. Second-order effects, not captured in the model, are apparently quite small.

RAM cost considerations

When reading from 10 collections with a read-ahead buffer of 800 KB per collection, 8 MB of RAM is used for buffering. With a disk array containing n disks, n times as much buffer memory would be needed to get the same performance results. By today's standards, 8 MB per disk is a small enough amount of memory to devote to I/O optimisations. At the time of writing this sentence (August 1999), 8 MB of RAM costs some 10–25 dollars, much less than the cost of the hard disk it would optimise. With memory prices dropping faster than hard disk prices, the equation will be still more favourable in future.

By using a modest amount (given current RAM prices) of memory, the read-ahead optimisation prevents a performance degradation if two or more collections are accessed in the event loop. This lack of a performance degradation makes possible a separation of concerns: different types of physics objects can be managed and clustered independently. The lack of a degradation also makes it possible to recluster objects of the same type by re-dividing them over many collections. One could therefore argue that the RAM used in the read-ahead was traded away to gain simplicity in the system design, as much as it was traded away to gain performance.

Because of the comparably low price of the RAM needed for read-ahead, research into the fine-tuning of the read-ahead buffer sizes to maximise the price/performance of the system has not been done.

5.6.2 Read-ahead optimisation and sparse reading

The collection iterator has no information about the exact physics objects that are read by the subjob in future. Therefore, the read-ahead optimisation inside the collection iterator always just reads ahead the next set of contiguous objects stored in the col-

lection. Thus, the read-ahead optimisation causes the collection implementation to do sequential reading on the database container, even when the subjob does selective reading on the collection. This sounds like a source of inefficiency, but it is not. As seen in figure 4.7 of section 4.4.5, sequential reading is as fast as selective reading, up to a certain performance boundary. This boundary depends on the selectivity, the average object size, and of course on the disks used. The collection iterator can monitor whether this performance boundary is crossed: in that case the read-ahead optimisation should be switched off for that collection, because selective reading of individual objects would be faster. Past the boundary, the performance of selective reading is actually indistinguishable from that of random reading: every seek to the next object in the collection has become a 'long seek', similar to the long seeks between the different collections. Switching off the read-ahead in one collection is therefore neutral with respect to the performance considerations in all other collection iterators. No global coordination about switching on or off the read-ahead optimisation is needed.

Though a mechanism for switching off the read-ahead optimisation if the performance boundary is crossed was designed, it was never implemented in any running prototype of this project. It turned out that automatic reclustering in these prototypes would hardly ever let the selectivity of access to a single collection drop far enough to pass the boundary where switching off was beneficial. With today's disks and with 8 KB objects, for example, the selectivity on a collection has to drop below 15% for switching off read-ahead to become interesting.

5.7 Validation of the basic storage management policies

The efficiency of the basic storage management policies presented in this chapter, together with the layers underlying the storage manager, was validated in a number of implementation tests. This section concerns a set of validation tests focused primarily at the *scalability* of the policies. It is shown that the policies work well on a very large hardware configuration, with many CPUs and disk arrays. The policies lead to an efficient utilisation of all allocated hardware under a range of likely physics data processing scenarios.

The results of additional validation tests, beyond the tests in this section, can be found in section 5.6.1 figure 5.6 for read-ahead, and section 6.7 for the basic reclustering policies.

Validation tests are necessary because the efficiency of the basic storage management policies cannot easily be established by analytical means. The policies have some strong circular interdependencies that make an analytical approach difficult. Furthermore, the policies all interact with layers below the storage manager. Chapter 4 contains a large body of knowledge about these layers, which was indispensable in guiding the design process. But based on the information in chapter 4 alone, one cannot analytically rule out the appearance of a performance breakdown in the underlying layers when all policies are combined. Therefore, new empirical validation tests were performed.

The CMS physics analysis system needs to scale to unusually large hardware configu-

rations (see section 2.6.3). This implies that credible validation tests of scalability will be far from trivial. Significant project time was spent in these validation tests, and, as shown below, unusually large hardware configurations were used.

5.7.1 Testing platform and software

To test the storage management policies and the layers underlying the storage manager, a testing platform was sought. This platform should contain as much hardware as possible, approximate likely CMS hardware configurations in 2005, run the Objectivity/DB software, and be available for use by the CMS collaboration. The platform found was a HP Exemplar supercomputer located at Caltech (the California Institute of Technology, Pasadena, USA). Figure 5.9 shows the Exemplar in the Caltech machine room. The tests performed contributed both to the goals of this designer's Ph.D. project, and to those of the GIOD project at Caltech (see section 2.8). System time on the HP exemplar is a scarce resource: the time was obtained by the GIOD project from the NPACI [68], a US organisation overseeing the allocation of supercomputing resources. Performance tests were mostly done during three visits to Caltech, while the Exemplar was in dedicated mode for exclusive use by these tests. Some preparatory and additional tests were done over the Internet from CERN, accessing the Exemplar in shared mode.

Figure 5.9: The HP Exemplar at Caltech.

The HP Exemplar is a 256 CPU SMP machine of some 0.1 TIPS. The machine consists of 16 nodes, which are connected by a special-purpose fast network called a CTI (see figure 5.10). Each node contains 16 PA8000 processors and one node file system. A node file system consists of 4 disks with 4-way striping, with a file system block size of 64 KB and a maximum raw I/O rate of 22 MB/s. An analysis of the raw I/O behaviour

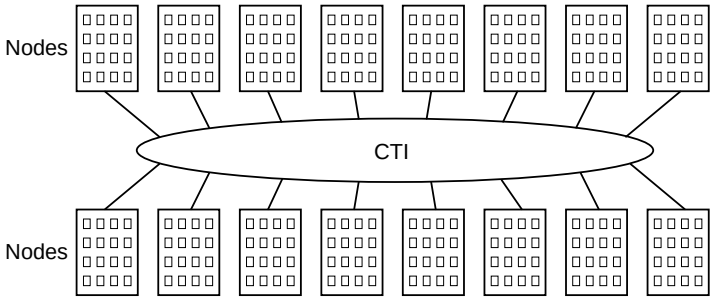


Figure 5.10: Logical configuration of the HP Exemplar at Caltech.

of the Exemplar can be found in [69].

The Exemplar runs a single operating system image, and all node file systems are visible as locally mounted UNIX file systems to any process running on any node. If the process and file system are on different nodes, data is transported over the CTI. The CTI was never a bottleneck in the validation tests that were performed. The CTI was designed to support shared memory programming, and can easily achieve data rates in the GB/s range. As such, the Exemplar can be thought of as a set of 16 16-processor UNIX machines with cross-mounted file systems, using an 'infinite' capacity network. Though the Exemplar is not a good model for current UNIX or PC farms, where network capacity is a major constraining factor, it could in fact be a good model for future farms that use GB/s networks like Gigabit Ethernet [70] or Myrinet [71] as an interconnect.

The Exemplar runs the SPP-UX operating system. SPP-UX is a specialised version of HP-UX, and binary-compatible with normal HP-UX software. The validation tests used the HP-UX version of Objectivity/DB v4.0.10. The test setup did not use the so-called Objectivity AMS server for remote database access: all database file I/O took place directly between the Objectivity database clients and the operating system. See section 4.2.4 for a more detailed discussion of the AMS.

The test loads were generated with the TOPS framework [72]. TOPS, the Testbed for Objectivity Performance and Scalability, is a specialised testing framework developed in this designer's Ph.D. project, and contributed to the RD45 collaboration. TOPS runs on top of Objectivity/DB. It can be used to test the performance of Objectivity and the underlying operating system and disk systems, under various clustering models and access patterns. It can also be used to test scenarios with many concurrent jobs. TOPS implements the basic storage management policies discussed in this chapter, and also contains physics workload generators to simulate the layers on top of the storage manager.

In the tests on the Exemplar, two things in the Objectivity architecture were of particular concern. First, the Exemplar node file systems, which have a block size of exactly 64 KB, and Objectivity does not support a matching database page size of 64 KB. It only supports page sizes up to 64 KB minus a few bytes. The exact reason for this limitation is not known. Probably the Objectivity implementation internally reserves some special case values in the 16 bit datatype for in-page offsets. As it does not support 'real' 64 KB pages, Objectivity does not match perfectly to the Exemplar

I/O system. After some experiments it was found the best I/O performance could be had with a database page size of 32 KB. This page size was used throughout the tests below. The second point of concern in the Objectivity architecture is the lock server. As discussed in section 4.2.3, Objectivity uses a single lock server process to handle all locking operations. This lock server could become a bottleneck when the number of (lock requests from) clients increases.

5.7.2 Scalability of the Objectivity lock server

In all tests described below, the Objectivity lock server was not a bottleneck. Lock server traffic was closely studied in a number of tests and experiments. The results are as follows. Writing loads the lock server more than reading. A client contacts the lock server whenever it resizes an Objectivity/DB container that is being filled with objects. The frequency at which container resizing operations occur can be tuned, by the application programmer, through two parameters. These are the initial container size, and container growth factor that determines how much space is added in each resizing operation. In all writing tests below (sections 5.7.3 and 5.7.5), the collection implementations were configured to use a large initial container size (200 pages) and a large container growth factor (20%). With smaller growth factors, resizing happens more frequently, leading to more lock server traffic, so that the lock server could become a bottleneck. From a study, performed in this project, of lock server behaviour under artificial database workloads with a high rate of locking, it is estimated that, even with large resizing factors, lock server communication may become a bottleneck in a DAQ scenario above 1000 MB/s [73].

The purpose of Objectivity read locks is to stop writers from writing too early. Most data in the CMS data store is read-only. In principle, when such data is accessed, no read locks are necessary because there are never any writers. Objectivity/DB nevertheless creates the read locks: there is no special mechanism by which the programmer can prevent the creation of read locks for particular types of data. This unnecessary creation of read locks loads the lockserver, but in the tests performed this did not endanger scalability.

The CMS computing technical proposal [10] foresees that hundreds of physicists will be using the CMS object store at the same time for interactive physics analysis. If the object store is implemented as an Objectivity federated database, this would lead to a database workload with hundreds of physicists performing short transactions. The scalability of the Objectivity lockserver under such short transaction workloads was not studied in this project.

5.7.3 Reconstruction test

The system was first tested under a ’reconstruction’ workload with up to 240 clients, each running a single subjob. The workload parameters were loosely inspired by the parameters of ’full event reconstruction’ as specified in the CMS computing technical proposal [10].

Every subjob in the test reads from 3 private collections and writes to 1 new private

collection. All objects read and written are 10 KB physics objects, filled with dummy data. As they all deal with private data, subjobs do not synchronise their reading or writing operations. The event loop in each subjob does the following.

- Reading: in every iteration, one object is read from the first collection. In every second iteration, one object is read from both the second and third collection.
- Writing: in every tenth iteration, one object is written into the new collection.
- Computation: with respect to computation, two tests with alternative scenarios were run. In the first test, the subjob spent 4 MIPSs per iteration in computation. This corresponds to 0.1 CPU seconds on one Exemplar CPU for every iteration, and $2 * 10^3$ MIPSs per MB read. In the second test, 2 MIPSs per iteration were spent, this corresponds to $1 * 10^3$ MIPSs per MB read.

The above parameters have the following relation to 'full event reconstruction' as specified in the CMS computing technical proposal [10]. Full event reconstruction reads 1 MB per event, and writes 100 KB, this yields the same 1:10 ratio between reading and writing as used above. It is expected that the objects read and written in full event reconstruction will be somewhat larger than 10 KB. The small 10 KB object size was chosen to make the test more sensitive to possible disk I/O performance degradations associated with non-sequential reading patterns. As seen in chapter 4, the performance gap between sequential and random reading is larger for smaller objects. The CMS computing technical proposal predicts a computation time of $2 * 10^4$ MIPSs per event (and thus per MB read) for full reconstruction. However, it also predicts that CPUs will be some 100 times more powerful (in MIPS per $) at CMS startup in 2005. The tests used computation times of $2 * 10^3$ MIPSs per MB read and $1 * 10^3$ MIPSs per MB read as a compromise.

In the reconstruction test setup, the chunks (and therefore the collections) are divided over four Exemplar node file systems. The Objectivity/DB federation catalog (see section 4.2.2) was placed on a fifth file system. The read-ahead buffer sizes were 4 MB per collection. Note that each collection was stored on a disk array containing 4 disks, so this gives a buffer of 1 MB per collection per disk, comparable to the 800 KB in figure 5.7.

The test results are shown in figure 5.11. The two curves represent the two tests with different amounts of computation per event. Both curves show almost linear scaling in the number of clients, indicating efficient use of the allocated disk and CPU resources.

In the test corresponding to the solid curve, the system remains predominantly CPU bound. In part of this curve below 150 clients, 91% of the allocated CPU resources are spent running actual reconstruction code. With 240 clients, 83% of the allocated CPU power (240 CPUs) is used for physics code, yielding an aggregate throughput of 47 MB/s, using about 0.1 TIPS.

In the test corresponding to the dashed curve, with less computation per MB read, the curve shows a clear shift from a CPU-bound to a disk-bound workload at 160 clients. At that point, the available I/O resources (16 disks in 4 striped file-systems) are saturated. The maximum throughput is 55 MB/s, which is 63% of the maximum raw throughput of the four allocated node file systems (88 MB/s). Overall, the disk

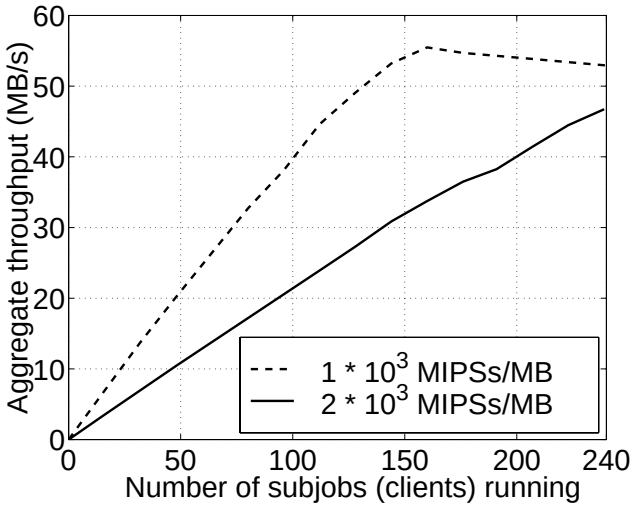


Figure 5.11: Scalability of reconstruction workloads.

efficiency is less good than the CPU efficiency. The mismatch between database and file system page sizes discussed in section 5.7.1 is one obvious contributing factor to this. On tests with some ten clients on a different platform, where a better match could be obtained between the 16 KB device block size of that platform and the database page size, higher disk efficiencies for similar workloads have been obtained. In the dashed curve, the transition region around 160 clients is relatively narrow, and the performance degradation if more clients above 160 are added is very low. Both are signs of very good scalability. Because Objectivity/DB architecture was not developed with scalability to hundreds of active clients as a design goal, a result this good was not expected at the start of the tests.

5.7.4 The read-ahead optimisation

The read-ahead optimisation was developed to optimise I/O performance for a client reading from multiple collections in its event loop. During the scalability tests with many clients, an interesting observation was made. It turned out that, the read-ahead optimisation improved I/O performance on the Exemplar even if every client was reading only a single collection.

Figure 5.12 shows a scalability test with n clients reading n collections, with each client accessing one collection only. The computation in each client is again $2 * 10^3$ MIPSs per MB read. The collections are placed on two node file systems, which have a combined raw throughput of 44 MB/s. Two tests were done: one with read-ahead enabled in all clients, one without read-ahead.

Figure 5.12 shows that without the read-ahead optimisation, the workload becomes disk-bound fairly quickly, at 64 clients. At 150 clients, meaning 75 concurrent readers per node filesystem, the lack of a read-ahead optimisation degrades the I/O performance with a factor of two.

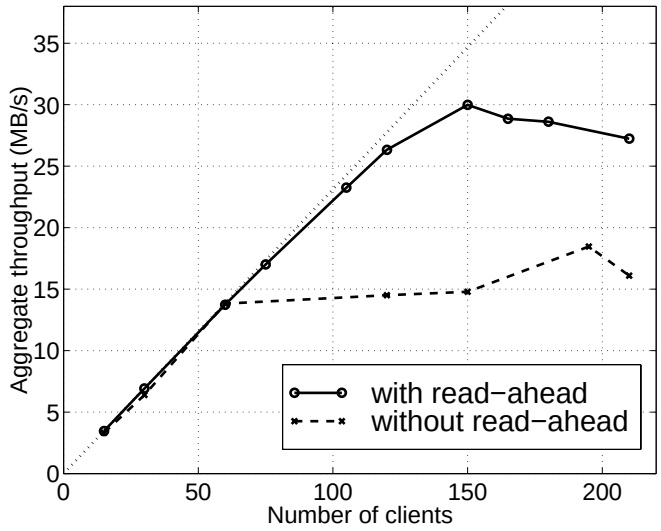


Figure 5.12: Performance of many clients all performing sequential reading on a container. The dashed straight line plots the maximum theoretical performance, computed by assuming that I/O overheads are zero and that every running client can spend 100% of one CPU in physics computations. Note that in this test, only two node filesystems were used. This explains the lower maximum when compared to figure 5.11.

The test result of figure 5.12 is consistent with that in section 4.4.6 figure 4.8, showing *no* performance degradation, without a read-ahead optimisation, up to 10 clients reading a single container each. For figure 4.8, this lack of a performance degradation was explained by noting that the operating system used a strategy of fulfilling multiple read requests from each client before switching to the next, thus reducing the number of 'long seeks' that have to be done. This simple strategy of fulfilling multiple read requests from one client before switching to the next cannot fully optimise I/O performance in the 'without read-ahead' test of figure 5.12. Because of the significant computation between I/O requests in each client, a single client on its own cannot produce I/O requests fast enough to saturate the I/O device. To fully optimise I/O, to the level achieved in the 'with read-ahead' curve, would have to use a more advanced strategy. As is apparent from figure 5.12, such a more advanced strategy was not available to the OS.

5.7.5 Data acquisition test

The system was also tested for a workload consisting exclusively of database write operations. This workload was inspired by scenario of filling the offline system with new data from the detector data acquisition (DAQ) system [10].

The system was tested with a quasi-realtime, continuous data acquisition workload up

to 238 clients. In this test, each client was writing a stream of 10 KB objects to its own collection. Once a collection reached a size of about 400 MB, it was closed and deleted by the client. Then the client created and filled a new collection. This was done to avoid running out of free file system space during the continuous tests. In a real data acquisition system, periodic switches to new collections would also occur, while retaining the old collections.

Per MB data written, a client spent about 180 MIPSs (0.45 CPU seconds on the Exemplar) in simulated data formatting. For comparison, for the same MB of writing, Objectivity spent 0.20 CPU seconds in the tests, and the operating system spent 0.01 CPU seconds. The database files are divided over eight node file systems, with the federation catalog on a ninth file system.

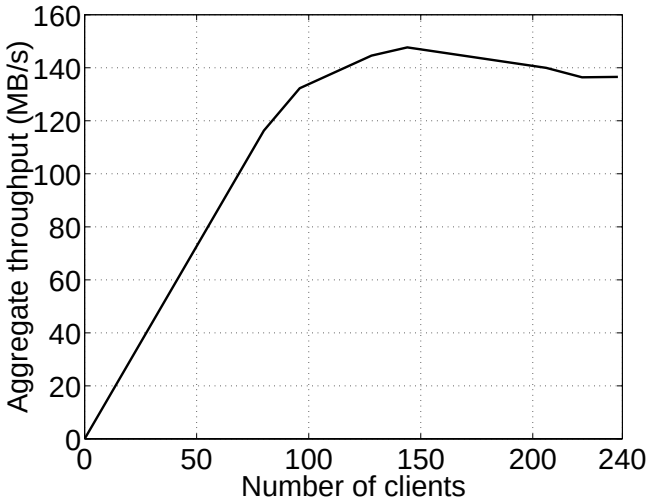


Figure 5.13: Scalability of a data acquisition workload.

The test results are shown in figure 5.13. Again, there is a transition from a CPU-bound to a disk-bound workload. The highest throughput is 145 MB/s at 144 clients, which is 82% of the maximum raw throughput of the eight allocated node file systems (176 MB/s).

Above 100 clients in this test, the system is I/O bound and the node file systems are saturated with write requests. When saturated with write requests in this way, the file systems show some surprising behaviour. It can take very long, up to minutes, to do basic operations like or creating a new (database) file, or committing all remaining data that was written to a (database) file from the operating system write buffers to the disks. It is believed that this is due to the I/O write request buffering facilities in the Exemplar operating system. Apparently, the operating system is buffering many megabytes worth of pending write requests for the disks in each node file system that is saturated. This buffering could be beneficial in some applications, but it is a problem for the data acquisition application. Buffering does not affect the throughput, but does affect the latency of operations that won't finish until data has been actually committed to the disks, and such operations are performed by a client when switching to a new database file. During the test, other file systems not saturated with write requests

still behaved as usual, with no exceptional latencies. From this test, one can therefore conclude that one should be careful in saturating disk arrays with write requests: unexpectedly long slowdowns may occur.

5.7.6 Validation conclusions

The CMS physics analysis system needs to scale to unusually large hardware configurations. This implies that credible validation tests, to show the scalability of the basic storage management policies, are far from trivial. Yet, without such tests, the correctness of the basic policies cannot be established, so significant project time was spent to perform these tests.

The validation tests show that the basic storage management policies work well in achieving good resource utilisation. The storage management policies, and the components below the storage manager, show almost ideal scalability, up to 240 clients, under CMS reconstruction and data acquisition workloads. The tests show excellent utilisation of allocated CPU resources, and satisfactory utilisation of allocated disk resources on the HP exemplar.

Additional scaling test results for Objectivity/DB on the Exemplar are documented in [73] and [74].

5.8 Related work

As noted before, a separation into chunks to implement resource farming is used in all current large physics analysis systems [59] [60] [27]. Most of these systems do not use the word 'chunk' but another term like 'ntuple file' or 'FileAtom'. Work on policies for assigning objects to chunks [25] [61] has been briefly discussed in section 5.3.1.

Type-based clustering is used in the BaBar system for storing physics objects [27]. The BaBar approach does not support reclustering. The BaBar storage structure for the physics objects of a single type in a chunk is roughly equivalent to storing objects permanently in a single collection. At the time of writing this sentence (September 1999), the BaBar system does not implement the equivalent of a read-ahead optimisation, or the equivalent of reclustering. BaBar plans to perform a set of live measurements of physicist's access patterns, and associated performance bottlenecks, before developing and implementing possible performance enhancements. BaBar uses Objectivity/DB as a basic storage infrastructure, and has performed a number of scalability studies of their storage management policies on top of Objectivity/DB [75]. The scalability of Objectivity/DB in database size, rather than throughput, has been studied in the ATLAS experiment [76].

The need for a read-ahead optimisation on top of Objectivity/DB was first identified in this designer's Ph.D. project. Some video on demand systems, that read multiple audio and video streams concurrently from the same disk array, also use read-ahead buffering [33] [34]. In these systems, the primary motivation for buffering in RAM is to hide disk delays, because streams need to be delivered at a fixed, continuous rate. However, the fixed-size buffers also optimise hard disk performance by minimising

seeks.

The ROOT physics analysis system [29] provides a 'tree' type for storing physics objects. This tree type incorporates policies similar to type-based clustering with read-ahead. ROOT allows the programmer to partition the physics objects associated with the events into several sets called 'branches'. On creating a tree with, say, 5 branches, ROOT maintains 5 buffers in memory, one for every branch. The sizes of these buffers are determined by the application programmer; the ROOT documentation recommends a size of 32 KB on the average machine. As the events are iterated through, the buffers are filled with objects in the corresponding branch. Once a buffer is full, it is written to disk. Buffer contents are written sequentially to the same file. Optionally, the buffer contents can be compressed before writing. When reading physics objects from a tree, reading on disk also happens in (maybe compressed) blocks, corresponding to the buffer size. When only one or a few branches are accessed in reading, the resulting access pattern on disk is a mix of sequential reading and medium to long seeks. This pattern, similar to the sequential reading interspersed with long seeks of the CMS storage manager, ensures reasonable disk performance. Note that the recommended 32 KB buffer size is relatively small if the goal is to spend the majority of the disk time on sequential reading, not seeks. Results in chapter 4 (e.g. figure 4.4) suggest that with a 32 KB buffer size, up to 3/4 of the time may be spent in seeks. Especially on a disk array with many disks, larger buffer sizes would be preferable.

5.9 Conclusions

In this chapter the basic policies of the CMS storage manager were discussed. The set of all events is partitioned into chunks, with storage management policies above and below the chunk level. The physics objects for all events inside a single chunk are further partitioned by type, into type-based storage domains. Sets of objects of a single type can be stored in a collection. Inside a collection, the objects are clustered in the fixed iteration order. Collections play an important role in reclustering operations. To recluster some physics objects of a single type T in a chunk X, these objects are re-distributed over a new set of collections. Collections use a read-ahead optimisation, which maintains performance when multiple collections are accessed in the same subjob.

The efficiency of the basic storage management policies presented in this chapter, together with the layers underlying the storage manager, was validated in a number of implementation tests. It is shown that the policies have excellent scalability and good efficiency on a very large hardware configuration, with many CPUs and disk arrays.

This chapter dealt with the basic policies that underly the CMS storage manager. The next chapter is about specific policies that govern reclustering.

Chapter 6

Reclustering on disk

6.1 Introduction

The previous chapter discussed basic storage management policies. Section 5.5.1 introduced the basic policy for reclustering: to recluster some physics objects, they should be re-distributed over a new set of collections. Figure 5.4 showed some sample reclustering operations.

This chapter expands on the basic reclustering policy. A complete system for reclustering disk based physics data is developed, and the results of a prototype implementation are shown. This system is called the *disk based prototype* for short. With respect to the previous two chapters, this chapter and the next represent a shift in scope: storage management is considered at a much higher level. The previous two chapters were mostly concerned with optimising access to physics objects at the subjob level. Here, the scope shifts to optimising long *sequences of jobs* through reclustering. In this chapter, the scope is still limited to reclustering a dataset that is fully on disk. In the next chapter, the scope widens again to include tape.

This chapter first covers the reading of reclustered objects, that is how a subjob can access, as fast as possible, a set of physics objects scattered (reclustered) over multiple collections. After that, the problem of developing reclustering strategies is discussed. The specific reclustering mechanisms developed for the disk based prototype are discussed next. Then, implementation and validation of the prototype are discussed. The chapter ends with an overview of related work.

6.2 Reading reclustered objects

This section deals with the reading of reclustered objects. More formally, it covers the following problem. Consider the set O of all physics objects of type T in a chunk X. As seen in chapter 5, the objects in O are (re)clustered by dividing them over several collections. Some of the objects in O may appear in more than one of these collections. Consider a subjob that needs to read a subset S of the physics objects in O. The problem is: which of the collections should the subjob read to gain good efficiency? For the purpose of this section, the 'optimal' way of reading a set of objects is the defined as the way which gives the highest efficiency, the lowest overall reading time.

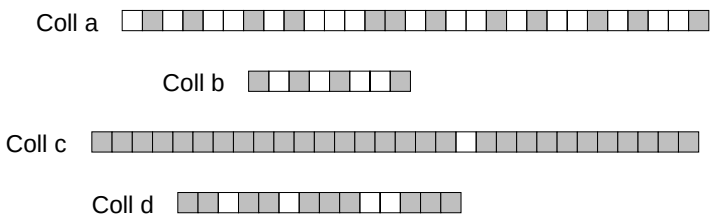


Figure 6.1: An instance of the reading problem: 4 collections, each containing some (grey) objects needed by a subjob.

Figure 6.1 shows an instance of the above problem, with 4 collections. The objects to be read by the subjob are grey.

6.2.1 Even distribution of objects

Inside each collection, the grey objects are more or less evenly distributed. This is because the cut predicates used to select events, and thus objects to read, are independent of the iteration order in which the events are clustered. These cut predicates select on physics properties, and inside each chunk, the physics properties do not depend on the iteration order. As seen in section 5.4.2, the iteration order inside a chunk is the time order in which the events occurred, and the event's physics properties are time independent.

If significant clumping of grey objects inside collections occurs, this would make it attractive to use optimisation strategies that treat collections differently depending on the progress of iteration. However, unless the reclustering process itself introduces an iteration order dependent bias, such clumping is statistically unlikely, so this specific optimisation opportunity is not explored further here.

6.2.2 Possible duplication of objects

Recall that some physics objects may appear in multiple collections after reclustering. Because of this duplication, the problem representation in figure 6.1 does not contain enough information to see the optimal solution, the solution with the fastest reading time. Figure 6.2 shows a different representation of the same problem, with the collections plotted as a Venn diagram showing the overlap in object sets. The figure decomposes the 4 collections into 6 distinct areas. In every distinct area, all events are members of exactly the same collections. Each area is labelled with the percentage of objects contained in that area that are needed by the subjob.

From the greyscale colouring, corresponding to the percentages, in figure 6.2, it is obvious to a human, that the subjob should read the objects it needs from collections c and d. The subjob should never attempt to access collections a and b, even though they do contain some of the objects needed. As discussed in section 5.5, the subjob will read collections c and d using their collection iterators. Together, these collections contain all objects needed.

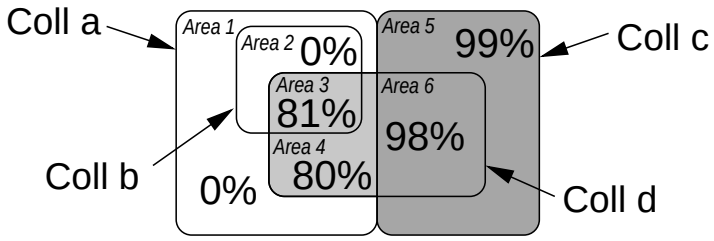


Figure 6.2: A Venn diagram of the objects in 4 collections. The collections decompose into 6 distinct areas, each area is labelled with the percentage of the objects therein that are needed by the subjob. The greyscale colouring of the areas also represents this percentage.

6.2.3 Optimal reading of objects

Generalising the above example, the reading of collections can be optimised by computing a set of collections C, and instructing the subjob to read objects from these collections only. C must be computed so that:

- the collections in C together contain all objects needed by the subjob, and
- the sum of the sizes of the collections in C is minimal.

For the collections in figure 6.2, this computation yields $C = \{\textit{Coll c}, \textit{Coll d}\}$.

To see why the above cost function 'sum of the sizes of the collections in C' is appropriate, one must consider the time it takes for the subjob to iterate though all collections in C at the same time. First, consider the case where, if the optimal way of reading is used, all concerned collection iterators have their read-ahead optimisations switched on. As discussed before (section 5.6.2), prototyping studies have shown this is the most frequently occurring case. The results in section 5.6.1 figure 5.7 show that aggregate throughput of the collection iterators in the subjob roughly equals the throughput of sequential reading. This implies that the running time of the subjob (if I/O bound) equals the running time of reading all collections sequentially after each other. This time in turn is proportional to the sum of the sizes of the collections. Thus, by minimising the sum of the sizes, the running time (or the I/O resource usage) of the subjob is minimised.

In the case that the optimal solution does *not* involve switching on the read-ahead optimisation in all collection iterators involved, the above cost function does not necessarily minimise the subjob running time. This more complicated case is considered in section 6.2.5.

Note that, in the usual case when all read-ahead optimisations are switched on, the subjob only needs to be instructed to read objects from the collections in C. If an object that is needed exists in multiple collections in C, it is usually immaterial from which one collection this object is actually read, because the read-ahead optimisations of all these collections will usually have pre-fetched this object into the in-memory read-ahead buffers of all these collections already. In theory, if one of these collections

does not have the object in its read-ahead buffer already, then reading the object from this particular collection should be avoided, because this yields a tiny performance benefit. This benefit would be that the next read-ahead operation for this collection would start reading after, not at the position of the object in question, saving a bit of I/O resources. In practice, these tiny savings would be lost in the noise. In prototyping efforts, this theoretically possible optimisation was therefore not implemented.

The above cost function sometimes leaves open some freedom in choosing a C. If there are three collections C_a, C_b, and C_{ab} with $C_a \cap C_b = \emptyset$ and $C_{ab} = C_a \cup C_b$, then $C = \{C_a, C_b\}$ and $C = \{C_{ab}\}$ have equal costs. In this case, the prototype implementations use a tie breaker, which chooses the C with the smallest number of collections in it.

6.2.4 Set covering problem

The calculation of C is an instance of the *set covering problem*, which is NP-complete, but for which good approximation algorithms exist [77] [78]. The set covering problem can be defined as follows:

> Take finite sets $S_1 \ldots S_n$, and a set $T \subset S_1 \cup \cdots \cup S_n$. A cover C for T is a subset C of the sets $S_1 \ldots S_n$ so that $T \subset \bigcup_{S_i \in C} S_i$. Find the cover C that minimises $\sum_{S_i \in C} size(S_i)$.

Variants of the problem minimise $size(C)$ and $\sum_{S_i \in C} w_i$, where w_i is a 'weight', a number attached to every set S_i. All of these variants have equivalent complexity.

In prototyping efforts, it was found that the instances of the set covering problem that occurred in practice when calculating the C above were almost always small enough, and simple enough, to compute the optimal solution with brute force. Problem sizes were usually below 10 sets.

In the prototypes that were made, an algorithm is used that will either solve the set covering problem optimally, or approximate the optimal solution. Figure 6.3 shows the algorithm in high-level pseudo-code. The algorithm does a branch-and-bound search of the solution space, and uses a timeout mechanism to stop the search if the solution space turns out to be too big to explore within a reasonable time. The timeout used was typically 4 seconds. If the timeout occurs, then the best solution found up to then is used as an approximation of the optimal solution. The branch-and-bound algorithm uses a 'greedy' search strategy as follows: when seeking to extend a partial solution, the candidate set containing the largest number of needed objects should be added first. This 'greedy' approach greatly reduces the computation time needed to find the optimal solution, or else increases the quality of the best solution found at timeout.

```
typedef Set<EventID> Collection;
typedef Colls Set<Collection>

// Solve the set covering problem
// with the sets C = {S_1, ..., S_n} and target set T.
Colls setcover(Colls C, Set<EventID> T) {
 Colls best_solution;
 int best_cost = ∞;
 extend_cover({ }, 0, C, T, best_solution, best_cost);
 return best_solution;
}

// Auxiliary function used by setcover().
void extend_cover(Colls partial_solution, int cost_of_partial_solution,
                  Colls candidates, Set<EventID> T, Colls &best_solution, int
&best_cost)
{
 // If the partial solution is a cover, then see if it is the best,
 // and in any case return to avoid extending the solution further
 if(T ⊂ ⋃ partial_solution)
   {
    if(cost_of_partial_solution < best_cost)
      {
       best_solution = partial_solution;
       best_cost = cost_of_partial_solution;
      }
    return;
   }

 if((best_cost != ∞)&& 'The timeout has passed') return;

 Collection biggest = 'find the biggest collection in candidates';
 Colls rest = 'all candidates that are left after removing best ' ;

 // If biggest extends the partial solution, then search
 // partial solutions with biggest, but bound the branching
 if('biggest has events in T that are not in ⋃ partial_solution)'
  if(cost_of_partial_solution + size(biggest) < best_cost )
     extend_cover(partial_solution ∪ { biggest },
      cost_of_partial_solution + size(biggest), rest, T, best_solution, best_cost);

 // Search partial solutions without best
 extend_cover(partial_solution,
    cost_of_partial_solution, rest, T, best_solution, best_cost);
}
```

Figure 6.3: Pseudo code of the algorithm used to solve or approximate the set covering problem in the prototypes.

6.2.5 Optimal solution for the case of sparse reading

The straightforward optimisation approach discussed above finds the optimal solution for reading under the assumption that this optimal solution involves switching on the read-ahead optimisations in all collection iterators involved. In prototyping studies it was found that this assumption was almost always true in practice. Nevertheless, for theoretical completeness, this section discusses a more involved optimisation approach, that takes the option of switching off read-ahead into account.

As discussed in section 5.6.2, the read-ahead optimisations should be switched off in a collection iterator if the sparseness of reading drops below a certain level. With today's disks and with 8 KB objects, for example, the selectivity on a collection has to drop below a 15% to make switching off read-ahead interesting. Section 4.4.5 shows that, when reading less than 15% with read-ahead switched off, the speed of reading is equal to the speed of random reading. Furthermore, this section shows that with the read-ahead cutoff at a selectivity factor c, say $c = 0.15$ in the above example, the speed difference between sequential and random reading is $1/c$. To summarise the above results, reading n objects from a collection of size s takes a time proportional to

s, if $n/s \geq c$, with read-ahead switched on,
$1/c * n$, if $n/s \leq c$, with read-ahead switched off.

As discussed before in section 5.6.2, the cutoff factor c depends on the (average) object size of the objects stored in the collection, and on the type of disks used.

Consider the distinct areas in figure 6.2. For each area that contains some needed objects, there are two options for reading these objects. They could be read by a collection iterator for an enclosing collection that has the read-ahead optimisation switched on, and by an iterator for an enclosing collection with the read-ahead optimisation switched off.

Taking this into account, the optimal schedule can be represented by two sets, C and D, of collections. The corresponding instructions to the subjob are as follows.

- For all collections in C, collection iterators are created with the read-ahead optimisation turned on.
- For all collections in D, collection iterators are created with the read-ahead optimisation turned off.
- If an object to be read is present in a collection in C, then the subjob should read it from one of those collections. If not, the object should be read from any one of the collections in D.

The sets C and D of collections must be computed to satisfy the following criteria.

- the collections in C and D together contain all objects needed by the subjob.
- The following cost function is minimised:

  the sum of the sizes of all collections in C
  + $1/c *$ the number of objects needed by the subjob that are not in any of the collections in C.

To show that this optimises the performance of the subjob, it is necessary to show that the above cost function is proportional to the running time of an I/O bound subjob. This can be shown as follows. First, note that section 5.6.2 shows that switching on or off the read-ahead in one collection is neutral with respect to the performance considerations in all other collection iterators. This implies that the running time for all iterators can be calculated by simply adding the projected running times of individual iterators running on their own. As already seen earlier, the first term above, the sum of the sizes of the collections with read-ahead, correctly gives a value proportional to the time it takes the iterators for the collections in C to traverse their collections. The second term above can be decomposed as follows, with $D_1 \ldots D_m$ the collections in D:

'second term' $=$

$$\sum_{i \in 1 \ldots m} 1/c * \text{ the number } n_i \text{ of objects read by the subjob from collection } D_i.$$

Above it was shown that the term for every D_i is proportional to the time it takes to read the n_i objects from D_i *provided that* $n_i/s_i \leq c$, with s_i the size of D_i. This provision is always true: it can be shown as follows.

> To show that the provision $n_i/s_i \leq c$ is always true, this proof assumes otherwise and derives a contradiction. Assume that $n_i/s_i > c$. In that case, $1/c * n_i > s_i$ also holds (multiply both sides of the former with s_i/c, note that s_i/c is a positive number). But if $1/c * n_i > s_i$, then moving the collection D_i from D to C would give a lower value of the cost function. But this is impossible because C and D minimise the cost function. QED.

It is interesting to note here that the above cost function leaves a lot of freedom in choosing the set D: any set D of collections not already in C, with these collections together containing the 'missing' objects, will do. A second constraint, to minimise the sum of the sizes of the collections in D, can be added. This constraint will minimise the collection indexing overheads in the subjob. Again, a tie breaker to choose the smallest D if there are multiple candidates left can be used.

The computation of the best C and D according to the above rules can again be mapped to an instance of the set covering problem.

6.2.6 Conclusions on reading reclustered objects

In this section, a method was developed for reading reclustered objects in an optimal way, that is as fast as possible. A concrete implementation of the method is discussed in section 6.6.

With the method developed here, the optimal way of reading is calculated at the collection level, rather than at the level of individual objects. The subjob reading the objects does not have to perform an expensive optimisation calculation for every object encountered. The object level computations are limited to finding the first collection in which the object appears, searching C and possibly D.

The optimisation method developed here is robust, in that it can find the optimal way of reading for any possible configuration of reclustered data. This robustness is an important property. Because of robust reading, there is great freedom in the design of actual reclustering mechanisms. If the optimiser for reading were fragile, this would have to be taken into account when designing the reclustering algorithms: one would have to ensure that the clustering arrangements produced would always be simple enough for the reading optimiser. However, the reading optimiser presented in this section will even handle very complicated clustering arrangements correctly. Thus the designer of reclustering algorithms does not have to worry about producing, intentionally or unintentionally, arrangements of great complexity. The reclustering design can effectively be decoupled from the problem of reading back reclustered data.

6.3 Reclustering strategies

Section 5.5.1 introduced the policy for reclustering. In this section, specific refinements of this policy are considered. Note first that the policy of section 5.5.1 decomposes into two primitive operations:

1. copying some physics objects into a new collection,
2. deleting an existing collection.

Many reclustering strategies are possible. The simplest strategy, from an implementation standpoint, is to make the end user responsible for specifying and initiating all reclustering operations. Most existing physics analysis systems allow for a crude form of user-guided reclustering: the user can copy a subset of an existing set of events into a new set, and then manually redirect new jobs to this new set. In the context of the CMS storage manager, one could allow end users to explicitly issue commands that create new collections, and delete old collections. Manual redirection of jobs would not be necessary. Selection of the right collections can be made automatic, using the method for optimal reading developed above.

Manual reclustering is an attractive technique to physicists. It is currently in widespread use. Manual reclustering is well understood, so it is not very interesting anymore from a research standpoint. Therefore, this thesis focuses on exploring *automatic* reclustering. All prototypes developed implement automatic reclustering only. However, the CMS storage manager built in 2005 will likely have to support both manual and automatic reclustering. Manual and automatic reclustering are compared further in section 6.8.2.

With respect to automatic reclustering, again many strategies are possible. Optimisers that analyse the physics properties of events could be used to cluster events with similar properties together. This option was already discussed for optimisations at the chunk level in section 5.3.1. To some extent, this option still relies on a 'by hand' advance encoding of physics properties likely to be important. In this designer's Ph.D. project, this option of identifying important physics properties was not pursued. Work on identifying such properties is better done at some time shortly before the startup of the CMS experiment in 2005. At that time, the knowledge about these properties will be much higher than it is currently, so a better job can be done. There is little value in

having an encoding of important properties at this point in time, years before it needs to be applied to actual data. This thesis is therefore mainly concerned with another option, which does not require advance encoding of physics knowledge. The option is to observe the objects accessed by current jobs in order to predict which objects are accessed by future jobs. As seen in chapter 3, physics analysis efforts contain sequences of jobs that all iterate over the same object set. Exploiting this repetitiveness in the workload is very attractive. By observing actual physics jobs as they happen, rather than relying on advance guesses about which physics properties will be most relevant, this technique in some sense offers a complimentary addition to the various possible 'by hand' optimisations.

6.3.1 Reclustering as an investment

Because of the copying of objects involved, reclustering operations are expensive, both in resources needed to perform the copying, and in the space needed to keep the copy. As concluded in section 4.7, these expensive operations are only attractive because sparse reading by physics analysis jobs can be even more expensive.

Basically, a reclustering operation is an *investment*, which can be made to avoid the high costs of repetitive sparse reading operations that would otherwise be expected in future. Reclustering is based on expectations of sparse access, not the certainty that it will occur. In other words, reclustering is an investment game where actions are performed because of a *possibility* for a future payoff.

6.3.2 Reclustering strategy of the disk based prototype

In this chapter, a complete system for reclustering disk based physics data, called the disk based prototype, is developed, and the results of a prototype implementation are shown. The prototype uses two different types of reclustering, called 'filtering' and 'batch reclustering'. Filtering and batch reclustering operations are discussed in the next two sections.

6.4 Filtering

A filtering operation is a reclustering action that runs concurrently with a subjob. In filtering, some or all of the objects read by the subjob are copied to a new collection. Two examples of filtering operations are shown in figure 6.4.

Seen as an investment, filtering speculates on the possibility that in future, in a large fraction of the cases (see figure 3.5), some subjobs will be run that accesses exactly the same object set as the current subjob. After filtering, these subjobs on the same object set, if they occur, can read the objects more quickly from the new collection.

Filtering is done automatically, transparently to the physics application code running in the subjob. The filtering actions of a subjob are governed by a *filtering function* F. This function yields, for every specific object read by the subjob, the (boolean) decision whether to filter the object to the new collection. The function is computed

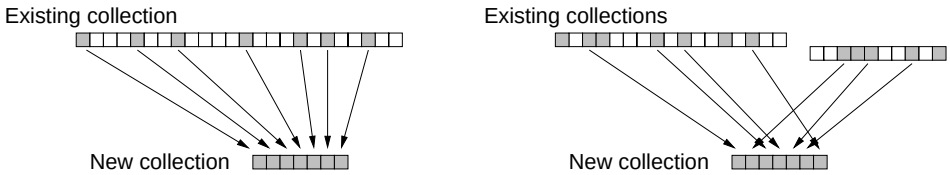


Figure 6.4: Two examples of filtering operations. On the left is the most basic form, on the right a more complicated operation filtering some of the objects from two collections. The objects read by the subjob are grey.

by an optimiser just before the subjob starts its iteration over the objects. By using the filtering function, the subjob can make sophisticated object-level decisions without performing expensive calculations for each object encountered.

As its input, the filtering function takes, for every particular object, the set of all collections in which this object is currently present. Figure 6.5 shows an example of a filtering function, in this case there are two existing collections (A and B) that contain some objects needed by the subjob, and a third collection is filtered from them. The function values $F(\{A\}) = false$ and $F(\{A, B\}) = false$ prevent the objects already in the small collection A from being duplicated, which would be wasteful.

Note that a filtering function F that yields $false$ for every input specifies that no filtering is to be done at all.

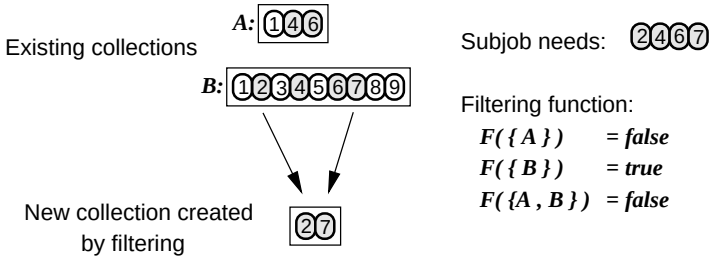


Figure 6.5: Example of a filtering function.

6.4.1 Choice of the best filtering function

As explained above, a filtering operation is an investment. The best investment is that which gives the highest profit. Following this reasoning, a way of estimating the profit associated with applying a filtering function was developed. The function with the highest profit estimate is the best one. With knowledge about the future being inexact, estimating profits is necessarily an inexact activity. Many estimation methods are possible, and the methods developed below are not the only plausible ones. The methods below have therefore been applied in prototyping experiments. They proved to work well in practice.

The profit estimate for a filtering function F is developed as follows. Most basically,

profit = savings $-$ price of investment.

The *savings* obtained by a filtering operation are due to having less sparse reading in future subjobs. Consider a single future subjob that reads exactly the same objects as the current subjob. The savings to this subjob if the filtering function F were applied can be calculated as:

running time R_0 for the subjob given the current set of collections, if no filtering is done at all
$-$ running time R_F for the subjob if the filtering function F were applied first

Ways to approximate such running times were already discussed in section 6.2. Following the approach in section 6.2, R_0 and R_F are defined as follows.

R_0 = the smallest sum of the sizes of collections containing all objects needed by the subjob, choosing collections from the set that currently exists.

R_F = the smallest sum of the sizes of collections containing all objects needed by the subjob, choosing collections from the set that currently exists, plus the collection created by applying the filtering function F.

Thus, the savings due to the running of a single subjob that accesses the same objects, after F is applied, are $R_0 - R_F$. The *total* savings depend on how many of such subjobs will be run, and how many similar subjobs, for example on subsets of the given object set, will be run. It was chosen to estimate the total savings as

$$r * (R_0 - R_F)$$

where r is a tuning parameter. This r is a measure of the 'repetitiveness' of the system workload, it corresponds to an estimate of how often subjobs on the same object set are repeated, and how often similar subjobs occur, before the clustering arrangement is radically altered again. With r a dimensionless constant, note that the value of the savings measure $r * (R_0 - R_F)$ is in 'bytes read from collections'.

The *price of investment* factor in the profit equation is constructed as follows. Consider that the price consists of (1) the time needed to create the new collection in copying, plus (2) the storage capacity needed for keeping the new collection until deleted. The time needed to create the new collection is proportional to the size S_F of this collection. The unit of S_F is 'bytes written to a collection', assuming that collection size is measured in bytes. This unit is compatible with the 'bytes read from collections' unit of the savings equation, as sequential reading and writing of equal amounts of data takes about equal amounts of time (section 4.4.3). The second term, the storage space needed, is also proportional to size S_F of the new collection. However, here the units do not match. Therefore, a tuning factor s is introduced that converts 'cost of storing bytes' to 'cost of reading bytes'. This makes the total price of investment to be

$$S_F + s * S_F.$$

Putting everything together, the following profit estimate is obtained.

profit in applying F = $r * (R_0 - R_F)$ $-$ $(S_F + s * S_F)$.

Note that R_0 is the same for every F. The F that maximises profit is therefore the F that *minimises*

$$r * R_F \ + \ (1 + s) * S_F.$$

The above value can be re-scaled to replace the two tuning factors r and s by one factor t. Thus the following cost function for F is obtained:

$$R_F \ + \ t * S_F,$$

the F that minimises this function is the best one. For the tuning factor t, values in the range 1 to 4 were found to work well in prototyping efforts. The factor t reflects both the repetitiveness of the workload (more repetitive means that a lower value is better in optimising the system), and the cost of storage space (more costly means that a higher value is better in optimising the system).

As an example, say that $t \ = \ 2$, and say that currently only a single collection C_1 exists, containing all $100\,000$ objects in a chunk. Say that a subjob reads $20\,000$ of these objects. With one collection, there are only two possible filtering functions, F_1 and F_2.

- F_1 is the function that mandates that no filtering is to be done at all: $F_1(\{C_1\}) = false$. The cost function value of F_1, according to the above formulae, is $100\,000 + 2 * 0 \ = \ 100\,000$.
- F_2 is the function that mandates that the $20\,000$ objects read are filtered into a new collection: $F_2(\{C_1\}) = true$. The cost of F_2, according to the above formulae, is $20\,000 + 2 * 20\,000 \ = \ 60\,000$.

In this example, F_2 has the lowest associated cost, so filtering will be performed with F_2.

With n collections containing objects needed by a subjob, there are $2^n - 1$ possible inputs for a filtering function (2^n minus one because the empty set can never occur as an input). The function can yield *true* or *false* for any of these inputs, so there are $2^{(2^n-1)}$ possible filtering functions. Finding the best filtering function is again an NP-complete problem. In the prototypes, this problem is solved by first mapping it to a set covering problem, and then solving the set covering problem with the greedy branch-and-bound algorithm outlined in section 6.2.4. A timeout of 4 seconds was used in searching the solution space. Problem instances usually stayed below $n = 10$ collections, and the algorithm only ran into the timeout occasionally. Most of the time, the optimum F was found.

As already noted at the start of this section, there exist many plausible cost functions by which to select the best F. Early versions of the disk based prototype used a two-sided selection strategy of maximising the *savings* estimate as above, while also ensuring that the return on investment *price of investment* / *savings* was good. A function F with slightly lower savings, but with a much better return on investment, could be chosen over the one with the highest savings. Compared to just maximising the profit, this reflects a more 'cautious' investment policy. The strategy worked reasonably well, but in the end it was dropped in favour of the strategy of just maximising profit. This was

done mainly because with the latter strategy, large problem sizes with many collections could be optimised better in much less computation time.

A filtering operation will allocate extra disk space in order to increase efficiency. In the disk based prototype, this space can be recovered, without losing the efficiency, with a batch reclustering operation.

6.5 Batch reclustering

Take all objects of a particular type T in a chunk X: these objects may be distributed over many collections, possibly with some duplicates. A batch reclustering operation works on the complete set of objects of a particular type T in a chunk X: it removes any duplication of objects caused by filtering operations, and globally optimises the clustering arrangement, taking recent access patterns into account. The operation is called 'batch reclustering' because it is not performed as a side-effect of a subjob. Rather, it is run on its own, typically whenever there are some free system resources to spare. In the disk based prototype, the database is not locked during batch reclustering: physics analysis jobs can still access the objects concerned while batch reclustering is in progress.

Batch reclustering operations are based on previously recorded access logs of physics analysis jobs. Whenever a subjob reads some objects of a type T in a chunk X, a log is produced that records the exact set of objects read. To recluster the objects of type T in chunk X, the batch reclustering operation will first process the recent logs for the chunk. The result is a view of all recent access patterns on the chunk, in terms of possibly overlapping sets. Such a view is shown on the left in figure 6.6. The view in this figure corresponds to a situation in which the access logs show exactly two different access patterns, presumably produced by jobs in two independent analysis efforts.

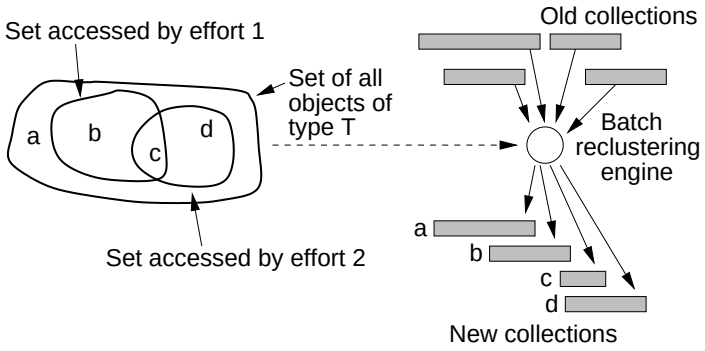


Figure 6.6: Batch reclustering.

In the set view, four different non-overlapping areas a, b, c and d are present. After computing the set view, the batch reclustering engine creates a new collection for each area in the view, each collection containing all objects in its area. This is shown on the right in figure 6.6. The objects for the new collections are copied from the old, existing collections. When the new collections have been created, all old collections

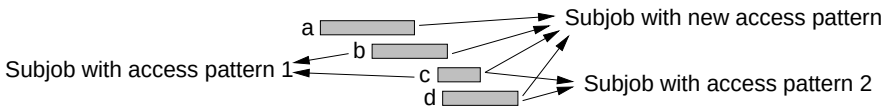


Figure 6.7: Use of batch reclustered collections by different jobs.

are deleted. If a running subjob is still accessing such an old collection, the batch reclustering operation will postpone deleting it until the subjob has run to completion.

The end result of batch reclustering is that the storage space occupied by the objects of type T in chunk X is reduced to the absolute minimum. After batch reclustering, the collections can be used by subsequent subjobs as shown in figure 6.7. Note that, for both access patterns on which the batch reclustering operation was based, the clustering arrangement is optimal, in that both these patterns will cause pure sequential reading on the two collections. A subjob in a completely new analysis effort, with a completely new access pattern, will usually have to read objects from all four collections, and could invoke filtering on any four of the collections while doing so.

The batch reclustering engine implemented in the disk based prototype contains an optimiser to suppress superfluous data copying in batch reclustering. The optimiser will suppress the reclustering of data from old collections, and the subsequent deletion of these collections, in cases where this reclustering action would produce little or no performance gain for the access patterns under consideration.

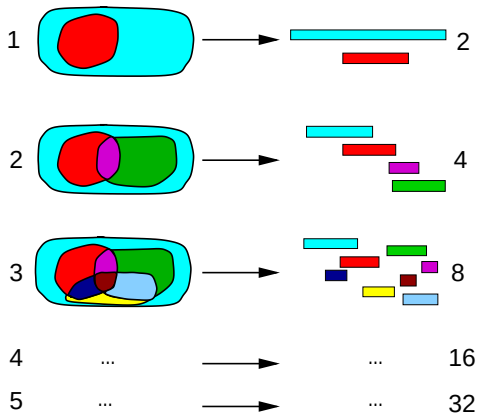


Figure 6.8: Exponential scaling in batch reclustering: n access patterns lead to 2^n collections.

6.5.1 Limitation of batch reclustering

One important limitation of the batch reclustering scheme is that it does not scale well in the number of different access patterns for which reclustering is done. In the worst case, n different patterns will produce 2^n independent collections after batch reclustering. Figure 6.8 illustrates this scaling effect.

For the disk based prototype, it was found that going beyond concurrent access to 20 collections per type produced overheads that were too large. Thus, the batch reclustering operation in the prototype is limited to optimising a maximum of 4 independent access patterns. The most recent patterns are chosen if more than 4 different patterns have been logged.

The prototype limits itself to batch reclustering for at most 4 different access patterns. Because of this, at most 16 collections are present after batch reclustering. By producing at most 16 collections without duplication, batch reclustering therefore also sometimes serves to simplify a very complex arrangement of collections created by many preceding filtering operations.

For a discussion of ways beyond the limitation to 4 independent access patterns, see section 7.2.

6.6 Implementation of the disk based prototype

In this chapter, a complete system for reclustering disk based physics data, called the disk based prototype, is developed. The prototype has the following main properties.

- It performs reclustering automatically, transparently to the user physics code, and the interface exposed to the physics code is as simple and narrow as possible.
- It can optimise clustering for four different access patterns at the same time.
- It keeps the use of storage space within bounds by avoiding the duplication of data.
- It supports multiple concurrent readers and writers.
- It supports 'on-demand reconstruction'.

In on-demand reconstruction, if an event e, encountered during subjob iteration, does not yet have a physics object of type T associated with it, this object is computed and used on the fly, then also stored for later use. The CMS computing model [10] places particular emphasis on the use of on-demand reconstruction as a way to optimise some types of physics analysis. On-demand reconstruction is not currently used on a large scale, if at all, in high energy physics.

The prototype has been implemented on top of Objectivity/DB. Two concrete implementations have been made. The first implementation was integrated with the TOPS framework [72], which was discussed earlier in section 5.7.1. Integration with TOPS allowed for extensive performance measurements based on simulated physics scenarios. A second implementation [79], largely re-using the code of the first, integrated the prototype with the LHC++ framework for physics analysis [80]. This second implementation validated that the prototype would indeed fit as a storage manager inside a complete physics analysis system, and that it would correctly operate as part of it.

The disk based prototype implements both filtering and batch reclustering. The implementation of batch reclustering is straightforward. Because of the narrow interface to the physics code, the implementation of filtering is more interesting. This implementation is discussed in detail below.

6.6.1 Transparent, narrow interface

As mentioned above, the prototype exposes a transparent and very narrow interface to the running physics code. The interface works as follows. At the start of the subjob, the physics code must create a *store accessor* for each type T of physics object that needs to be accessed. Creation of the store accessor just involves supplying the identifier of type T to the storage manager. The physics code does not have to specify beforehand which objects will be accessed. Instead, the specification of which objects to read happens on the fly, as is usual in most disk-based physics analysis systems. During iteration, if an event e is reached, the physics code can obtain the corresponding object of type T by using a 'get object belonging to e' operation. This operation is provided by the store accessor of type T, and returns a pointer to an in-memory copy of the object.

The narrow interface does not give the store accessor any prior knowledge about which objects of a type T will be read. To optimise reading and filtering, the store accessor implementation therefore needs to use predictions based on statistical methods. This use of statistics is necessary even though exact information about the set of objects to be read is usually available, at the start of subjob iteration, on the physics code level. Nevertheless, it was decided develop a prototype with the above very narrow interface. This made the prototyping efforts more meaningful as an exercise in exploring the possible limits of reclustering schemes that are fully transparent to the user.

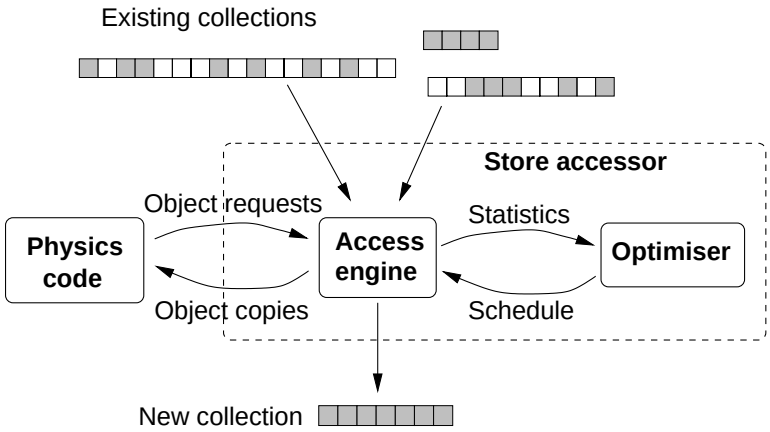


Figure 6.9: Implementation of the store accessor, with object and data flows. In this figure, objects are read from two collections, and filtered to a new collection.

6.6.2 Implementation of filtering

In the implementation of a store accessor, two distinct mechanisms are involved: the *access engine* and the *optimiser* (figure 6.9).

The *access engine* performs all tasks on the level of individual objects. It processes the requests for objects from the physics code by reading objects from the existing

collections. If filtering is to be done, the access engine copies the individual objects to the new collection.

The *optimiser* is run only once per subjob. It computes a *schedule* for the access engine that governs reading and filtering at the subjob level. The optimiser uses the methods in sections 6.2 and 6.4 to calculate the schedule. The calculations are based on statistical information on the relation between the set of objects read by the physics code, and the sets of objects contained in the existing collections. To cope with the lack of advance information about these sets at the start of subjob iteration, the access engine works in two phases. Statistics gathered in the first, short, phase are sent to the optimiser, which then returns a schedule to fully optimise the second, longer phase. The complete sequence of events in the access engine is as follows.

- *Initialisation*: Locate all collections of objects of type T (defined by the corresponding store accessor) in chunk X (defined by the subjob).
- *Phase 1*: (First few hundred requests from the physics code) Locate the subsequently requested objects in the collections. If a requested object has copies in multiple collections, choose the copy to read with a simple tie-breaking algorithm. Meanwhile, gather statistics for the optimiser.
- *Transition*: Send statistics to the optimiser. Receive schedule. Perform filtering for the objects requested in phase 1, if necessary, according to the filtering function obtained from the optimiser.
- *Phase 2*: Handle all subsequent requests, performing filtering, if necessary, according to the filtering function contained in the schedule. A different part of the schedule is used to optimise reading.

This two-phase approach works well because events with similar properties are distributed evenly over the collections (section 6.2). This makes it possible to make a good schedule based on the statistics for the few hundred initial requests.

The schedule supplied by the optimiser has two parts: one that governs reading and one that governs filtering.

In computing the *reading* part of the schedule, the optimiser uses the first method described in section 6.2: this method yields a set C of collections with the understanding that all requested objects should be read from the collections in C. The method in section 6.2 works on the basis of exact information. The statistics gathered by the access engine in the first phase are inexact however, and this could lead to the computation of a C that leaves out a collection that nevertheless needs to be read from, because it is the only collection containing a particular object that will be requested. To cope with this possibility, the access engine uses the following two rules for reading objects based on C. In order of priority, these are:

1. reading from a collection in C is preferred,
2. reading from a smaller collection is preferred.

The first rule implies that collections outside C may also be read from, the second rule optimises this 'emergency reading' to some extent.

In prototyping tests over a range of scenarios it was found that, with 100 requests in the first phase to base statistics on, there was never a case in which collections not in C had to be read from. Statistically, by choosing a large enough first phase, the probability that there is a discrepancy between the statistics and reality resulting in a sub-optimal schedule, and the average performance loss due to such schedules, can be made arbitrarily low. Of course, the access engine is most efficient in the second phase, so the first phase should not be made too long. A complete statistical analysis of this phase length tuning problem was not made. Given that likely subjobs will request at least thousands of objects, the overhead of a 100 request long first phase is lost in the noise, so it was not explored whether this phase could sometimes be feasibly reduced to, say, only 50 requests.

The *filtering* part of the schedule is computed using the methods in section 6.4, again based on statistics instead of exact information. In most runs with the disk based prototype, the tuning constant t was kept fairly high, at values of 4 or more. This has the effect of inhibiting most filtering operations. The high constant implies that the reading on existing collections has to be very sparse for filtering to be done. Thus, there is only filtering if this results in immediate, very high benefits to subsequent jobs. It was reasoned that frequent batch reclustering operations, executed when there are some resources to spare, would take care of obtaining the smaller benefits.

6.7 Validation

The disk based prototype was tested extensively with synthetic workload scenarios. The tests aimed to validate the overall design and to confirm the performance expectations underlying the design. Also, tests were done to analyse the complexity of the clustering arrangements produced, and the ability of the prototype to handle them. Overall the prototype proved to be robust under a wide range of workloads. Some specific robustness results have been mentioned in the previous sections. The remainder of this section centres on performance, rather than robustness results.

The main platform for the validation tests was SUNB (see section 4.4.2), with most results being cross-validated by running the same test again on HPB. The performance results plotted in the figures below are all for jobs that access 8 KB objects of a single type T. The Objectivity/DB page size was also 8 KB. The jobs all execute as a single subjob over a single chunk. The size of the physics object set used is 250 MB.

6.7.1 Performance without reclustering

To have a baseline for performance comparisons, a small prototype that does not do any reclustering was developed. The prototype stores all objects in a single collection, and does sequential or selective reading, with the read-ahead optimisation disabled. The use of read-ahead in this prototype is not necessary because, with a single collection only, the overheads due to reading multiple collections at the same time, which read-ahead was designed to minimise, can never occur.

A test scenario that spans a single physics analysis effort was developed. The effort in

the scenario has 4 phases, with each phase having 12 jobs. In each phase, the event set under consideration gets smaller. In the first phase, 100% of the events are considered, which means that 100% of the stored physics objects are read. In the next phases, the numbers drop to 50%, 30%, and finally 10%.

The test scenario was fist run on the prototype without reclustering. Figure 6.10 plots the results. Each pair of bars in the figure represents a single job. The height of the black bar in front represents the size of the event set over which the job was run. The height of the grey bar behind it represents the (wall clock) running time of the job. All jobs were I/O-bound. The jobs were run on a machine that was nearly empty, but not completely empty: this explains the slight variation in runtimes between subsequent jobs in the same phase. To make the extent of the variations in job runtime visible, figure 6.10 (and the similar figures 6.11 and 6.13 later on) plots jobs in a single run of the scenario only, not runtime averages for jobs over many runs of the scenario.

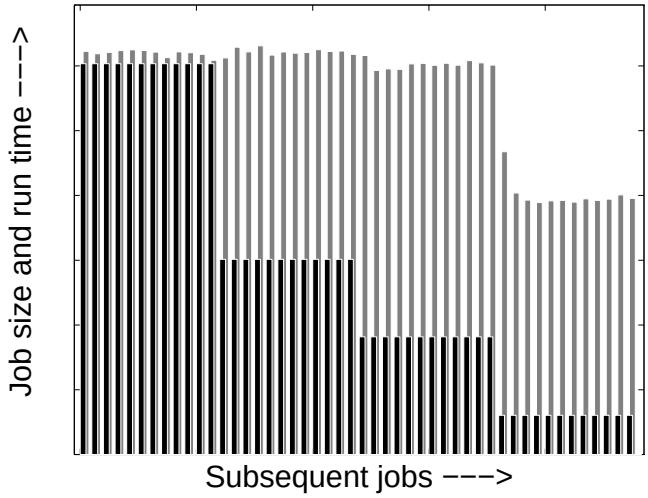


Figure 6.10: Performance of prototype without reclustering under a test scenario. Each pair of bars in the figure represents a single job. The black bar represents the number of objects accessed by the job, the grey bar the (wall clock) running time of the job. This figure shows a test run on SUNB, performance effects were validated on HPB.

In the first phase of the analysis effort, 12 jobs are run that all access 100% of the stored objects during iteration. These jobs produce a nearly sequential database page reading pattern on disk, the only perturbation is an occasional excursion due to the reading of a database-internal indexing page. The jobs in this phase achieve an I/O throughput of about 5 MB/s, not noticeably lower than the maximum throughput of raw sequential disk I/O for SUNB.

In the second phase, 12 jobs are run over a set containing only 50% of the events. The runtime of these jobs does not drop at all, even though the jobs read only half of the objects (and therefore only half of the database pages). For the next phase, in which the jobs read 30% of the objects, the same effect is seen. Only the jobs in the last phase, where 10% of the objects are read, are somewhat faster. These performance

results are consistent with those obtained earlier for selective reading on SUNB: see section 4.4.5 figure 4.6.

6.7.2 Performance effects of batch reclustering

Figure 6.11 shows the same test scenario as above, this time run with the disk based prototype as the storage manager. Filtering was completely disabled for this test, so that the performance effects of the batch reclustering operations alone can be seen in isolation. The wide grey bars in figure 6.11 represent the running of a batch reclustering operation, which, in this scenario, is run between jobs. The runtime of the batch reclustering operation is represented by the surface, not the height, of the wide bars: they are twice as wide as the other bars.

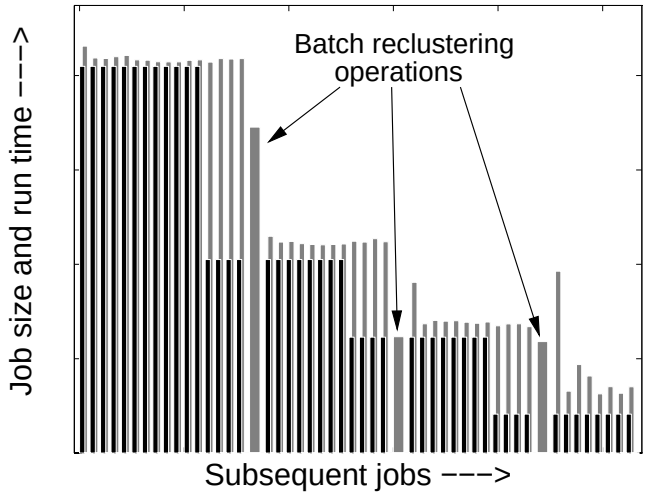


Figure 6.11: Performance of prototype with automatic reclustering under the same test scenario as in figure 6.10. This figure shows a test run on SUNB, performance effects were validated on HPB.

At the start of the test run, the objects in the database are clustered in a single collection, in the same way as for the prototype without reclustering above. The first 12 jobs again achieve a throughput not noticeably lower than the maximum throughput of sequential I/O on SUNB. For the next 4 jobs, the running time is again similar to that in figure 6.10. Then, the first batch reclustering operation is run. The operation examines the logs of the access patterns of the preceding jobs, finds two distinct patterns (reading 100% and reading 50%), and reclusters the database to optimise for these patterns, as shown schematically in figure 6.12.

After batch reclustering, the running time of the next 8 jobs, which access exactly the same data as the preceding 4 jobs, has improved. The time is again nearly proportional to the amount of data accessed, yielding a throughput close to that of sequential I/O. The first 4 jobs of the next phase have comparable runtimes, then the second batch reclustering improves runtimes again. In the jobs immediately after the second and third batch reclustering operations, spikes in the runtime can be seen. These longer runtimes

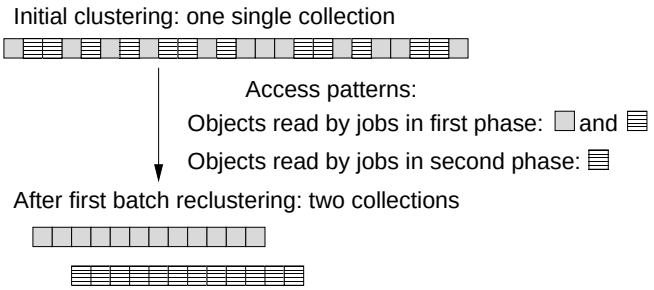


Figure 6.12: State of the database before and after the first batch reclustering operation in figure 6.11.

are caused by the emptying of the operating system write cache during the running of these jobs. This cache was filled by the preceding batch reclustering operation; the job following this operation was started immediately after the operation had finished, so there was no time for the operating system to empty the cache first.

6.7.3 Performance effects of filtering

A filtering operation will allocate extra disk space to increase efficiency. This space can be recovered, however, without losing efficiency, by running a batch reclustering operation later. A test run with such a scenario is shown in figure 6.13. In this test run, filtering is enabled.

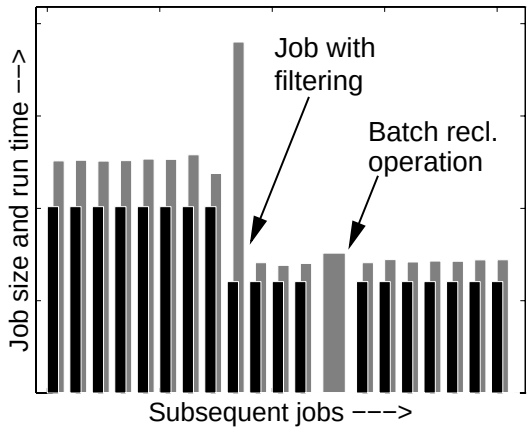


Figure 6.13: Filtering operation, triggered by the occurrence of a new access pattern, followed later by a batch reclustering operation. This figure shows a run on HPB, performance effects were validated on SUNB.

Figure 6.13 shows the last 8 jobs of a phase accessing 50% of the objects, then all 12 jobs of a subsequent phase accessing 30% of the objects. In the first job of the 30% phase, represented by the leftmost lower black bar, a filtering operation is performed.

Note that this operation increases the job runtime: additional I/O resources are spent in writing to the new collection. The second job in the 30% phase accesses the new collection rather than the old one, and so again has a run time proportional to the amount of data read. The batch reclustering operation after the fourth job in the second phase shrinks the store down to its original size by removing duplicate objects. It can be seen that the batch reclustering operation does not change the job running time.

6.7.4 Validation conclusions

From the measurements shown in the graphs above, it is clear that the filtering and batch reclustering operations have the performance effects predicted by chapter 4. The difference in the grey bars of figures 6.10 and 6.11 vividly illustrates the utility of reclustering. Both figures show the same analysis effort (black bars), but the job runtime (grey bars) needed to perform this effort is dramatically lower in the second figure, with reclustering in place.

6.8 Related work

Three kinds of related work are considered here: work in the object database community, work in the high energy physics community, and work on 'view materialisation', a database optimisation which is similar in some ways to the filtering operation. For related work on clustering, not reclustering, see section 5.8.

6.8.1 Related work in the object database community

Clustering and reclustering to increase performance is an important research topic in the object database community. However, this research is generally directed at what are considered 'typical' object database workloads. Such workloads consist of small unrelated transactions, where each transaction typically accesses tens of objects. Proposed general-purpose object database reclustering mechanisms, for example [81] and [82], take such workloads as a starting point. They make reclustering decisions based on statistics like object access frequency and link traversal frequency. These statistics are aggregated over many small jobs before reclustering is done. The disk based prototype does not use such statistics, but instead makes reclustering decisions based on either logs of, or real-time observations of, the exact access patterns of individual jobs.

General-purpose reclustering algorithms usually optimise the mapping of objects to database pages or chunks of database pages in order to increase the database cache hit rate. The disk based prototype aims to optimise both cache hit rate and the pattern of disk reading produced by cache misses. To optimise both, it does not map objects to pages, but to the collections developed in chapter 5. This feature of mapping to collections, and making optimisation decisions in terms of sets of collections, seems to be genuinely new. It was not found in any of the general-purpose systems that have been studied.

Compared to the 'typical' object database workload, which most general-purpose re-

clustering research aims to optimise, physics analysis workloads are highly atypical. As a result, the general-purpose reclustering mechanisms proposed in the object database community are not very applicable to physics workloads. They could optimise physics analysis to some extent, but they will not produce the efficiency gains of the disk based prototype in this chapter, because they fail to exploit some useful characteristics of physics workloads. Specifically, physics workloads mostly consist of read-only access, not update operations, transactions routinely access millions of objects, and most importantly the workloads lend themselves to 'streaming' type optimisations that preserve sequential reading speeds. An very early paper discussing the desirability of sequential access in databases is [83].

So far, the general-purpose reclustering algorithms proposed in academia have not yet made it into commercial products. It is conceivable that, at some future time, vendors will add some general-purpose automatic reclustering facilities to their products. It is not expected however that such future products will be able to provide efficient reclustering for physics workloads. As far as reclustering is concerned, physics analysis is too atypical to be provided for by the market. Therefore, it is concluded that the high energy physics community has to develop its own reclustering systems. This development work could conceivably be shared with some other communities that have data analysis applications similar to those in high energy physics. Examples of such applications are some types of satellite image analysis (both for earth observation and astronomy) and the study of climate modelling data [84].

6.8.2 Related work in the physics community

'Manual' methods of reclustering, in which users or administrators explicitly create new datasets ('data summary tapes'), and redirect jobs to them, have been common in high energy physics for a long time. They are also found in other application areas where users deal with very large datasets.

With respect to manual reclustering, the main benefits of an automated system like the disk based prototypes are expected to be (1) a higher degree of transparency for the end user physicist writing physics code (2) time savings for users and administrators, (3) higher performance because optimisations are applied more consistently and at a finer-grained level, (4) a quicker response to user needs in systems where only administrators can create smaller datasets, and (5) a global optimisation of the use of disk space, rather than a local one, in systems where all users can create and manage smaller datasets themselves.

A disadvantage of an automated system can be that some user knowledge about likely future queries will not be exploited. Note however that, if a user would have created a new 'data summary tape' with the objects O in a manual system, an equivalent, and usually as effective, action with the disk based prototype would be to run a 'dummy query' that reads the objects O, and thereby triggers a filtering operation for these objects.

Automatic reclustering has never been deployed in the high energy physics community. Significant research on reclustering (automatic or otherwise) was only started with the start of this designer's Ph.D. project. Shortly after the start of this project,

a strong case for reclustering research was made, based on the analysis of the performance of selective reading in object databases (see section 4.4.5). As a result of this, (re)clustering was later identified [85] as a major open area of research in the RD45 collaboration, which serves as the meeting ground for storage management R&D in the high energy physics community (see section 2.8).

Reclustering research was done in this designer's Ph.D. project, and also in a second Ph.D. project by Martin Schaller [86] [87]. In [86], a disk based reclustering prototype is also developed. Here, this system is called the HAMMING system for short. The HAMMING system uses job access logs to re-organise the clustering arrangement in a single operation, like batch reclustering does. The main difference in the HAMMING system's approach is that a fixed iteration order is *not* assumed. Instead, the HAMMING system may change the iteration order as part of the reclustering optimisation. This extra degree of freedom allows HAMMING to optimise for many (15 – 40) independent access patterns while keeping space occupancy minimal and the I/O performance reasonable. The exact number of access patterns that can be optimised for at the same time depends on parameters like the average job size and the object size, this is discussed further in section 7.2.1. The work in [86] deals with reclustering for a single type only. For jobs that access many types for every event, the possibility to change the iteration order yields a large number of new tradeoff options: these were not studied extensively in the work leading to [86]. Consequently, if multiple types are involved, the nature of the theoretical performance limits is an open issue.

6.8.3 View materialisation

In its most general form, a 'materialised view' is a relatively small summary or excerpt of a relatively large original data set. View materialisation, the operation of creating a materialised view, is therefore similar, in intent and effect, to a filtering operation.

The optimisation strategy of view materialisation has been under study in the database community for some time, see for example [88]. Recently, this strategy has received a lot of attention as an optimisation for data mining systems [89] [90] [91] [92] [93]. In view materialisation research, the original data set under consideration is usually a relational database consisting of tables, not an object database structured as a set of objects. The materialised view need not be a simple subset of all values in this larger set. In a data mining context, for example, if the larger original data set is the set of receipts of all sales transactions at various locations, one possible materialised view is a table that only has the total sales for every location.

A lot of research on materialised views is concerned with computing an optimal, static set of views, based on advance knowledge about access patterns, see for example [90] and [91]. Another significant research area is concerned with the problem of updating views efficiently, to reflect small updates of records in the base data set, see for example [92] and [93]. This 'view updating' research has little relation to CMS storage management issues however, because update operations on a few physics objects in a large existing set will occur seldom, if ever (section 3.2.4). There is also some existing research on mechanisms for automatic, *dynamic* creation and deletion of views, in the context of optimising SQL queries [94] and data warehousing (data cube) operations

[89]. The dynamic management or materialised views is a research problem very similar to that of the dynamic creation and deletion of (filtered) collections. This similarity leads to similar architectural patterns. For example, the systems in [94] and [89] both create new views as a side-effect of running a query, similar to the filtering operation which creates a new collection as a side-effect of running a physics analysis job. Section 7.9 discusses some striking similarities between the system in [89] and system for reclustering on disk and tape that is developed in chapter 7.

6.9 Conclusions

In this chapter, the basic reclustering policy introduced in section 5.5.1 has been developed further into a complete system for reclustering disk based physics data. An optimal policy for reading objects from collections has been developed, this policy is optimal in the sense that it maximises the I/O speed for any given access pattern on any given clustering arrangement. Then, reclustering in general was discussed, and two specific reclustering mechanisms, filtering and batch reclustering, were developed. Filtering and batch reclustering are automatic mechanisms. They work by observing the physics jobs that are run, rather than by being fed advance knowledge about the jobs that will be run. A validation effort shows that these reclustering operations work well, in that they greatly improve the I/O throughput when compared to a system with no reclustering. However, the reclustering mechanisms were not constructed or proven to be 'optimal' in the same way that the policy for reading was constructed to be optimal. The construction of truly optimal reclustering mechanisms, which would provably minimise the (average future) resource usage over long physics analysis efforts, is considered infeasible for a number of reasons. First, optimal mechanisms would have to take the complete wealth of detail in the physics analysis process (chapter 3) into account. Second, optimal mechanisms designed now would have to be extremely generic because of the large uncertainties surrounding the parameters of the physics analysis process, as discussed in chapter 3. Third, the many degrees of freedom in reclustering produce an exponential number of possible clustering solutions, and always choosing the optimal solution in a reasonable time is likely to be infeasible.

The policies for reading and reclustering are implemented in a disk based prototype. This prototype features a very narrow interface to the physics code. This narrow interface is possible because statistics can be used to steer optimisation. The prototype has been tested for robustness and performance. Test results show that filtering and batch reclustering have the performance effects predicted by chapter 4. The performance of the disk based prototype was compared to that of a prototype without reclustering. The prototype has also been integrated with the LHC++ framework for physics analysis [80]. This integration shows that it can indeed work as a storage manager inside a complete physics analysis system.

The reclustering techniques developed in this chapter differ significantly from those in proposed general-purpose object database reclustering systems. The difference is large because the techniques developed here successfully exploit the specific characteristics of physics analysis workloads in order to maintain near-sequential I/O performance.

The disk based prototype has a limitation, in that it can only optimise the clustering

for 4 independent access patterns any one time. In the next chapter, ways to go beyond this limitation are discussed. The next chapter also widens the scope from a purely disk based system to a system that manages data on both disk and tape.

Chapter 7

Reclustering on disk and tape

7.1 Introduction

In this chapter, a complete system for reclustering on disk and tape is developed. This system will be called the *tape based system* for short. With respect to the previous chapter, this chapter widens the scope of the design effort in a number of ways. First, both disk and tape are considered. Second, system workloads with tens to hundreds of independent access patterns, created by tens to hundreds of concurrent users, are examined. Third, some parts of the tape based system deal with many chunks and subjobs at the same time, rather than always being limited to a single chunk.

This widening of the scope implies that a different design methodology has to be used. The system and its workloads are now so complex that simulation with likely workloads is the only feasible way to refine the schedulers and optimisers in the system. The optimisation strategies can no longer be completely developed analytically from first principles, as was done in chapter 6.

This chapter first discusses the issue of scaling beyond the 4 independent access patterns of the disk based system in chapter 6. Then the basic design of the tape based system is introduced, followed by the choice of a design methodology for the detailed design. This methodology relies heavily on simulation: the system and workload parameter space used in the simulations is considered next. Following, the system components and schedulers obtained in the detailed design phase are described. Finally, validation results and related work are covered.

7.2 Beyond four independent access patterns

In the disk based prototype of chapter 6, it was decided to limit batch reclustering to 4 independent access patterns. With hundreds of CMS physicists, many more than 4 independent access patterns will need to be supported. When going beyond 4 independent access patterns, a number of strategies are possible. For example, to optimise for 8 independent patterns, one could simply make two copies of the original object set, and have each copy managed by one instance of the disk based prototype, dividing the users between the instances. This strategy trades away storage space to maintain performance for more access patterns. Ideally, one would like to trade away as little storage space as possible, preferably none. In the remainder of this section, the issue

of trading away space is explored from two angles. First, the limits of systems that never trade away space are discussed. Second, for systems that do trade away space, the likely space overheads are estimated.

7.2.1 Limits of systems that never trade away space

As already discussed in section 6.8.2, the HAMMING disk reclustering system made by Martin Schaller [86] never trades away space: the original object set is reclustered (reshuffled), but without any replication of data. This disk reclustering system maximises clustering efficiency to near the theoretical limit, for any number of independent access patterns. This means that the object set is clustered in such a way that the speed with which all objects in a pattern can be read, averaged over all supplied access patterns, is maximised. If this system is supplied with a few, say 4, access patterns, then the (average) speed obtained is close to that of sequential reading, as it is for the disk based system of chapter 6. If more than 4 patterns are supplied, then the average speed achieved by the system will slowly drop. When more and more patterns are added, the speed will eventually level out at the speed of selective or random reading an object set in the original storage order. At that point, using the HAMMING system is not interesting anymore: not reclustering data at all will serve the workload with these many access patterns equally well. Because the HAMMING system maximises efficiency to near the theoretical limit for any system that does not trade away space [86], this work also provides a good indication of the efficiency limits of *all* possible systems that do not trade away space.

In [87], Martin Schaller calls a reclustering system *efficient* if it can maintain an average reading speed, for all supplied access patterns, of at least 75% of the speed of sequential reading. Note that the disk based system, which can maintain a speed near that of sequential reading, is efficient in this sense, though at the cost of scaling only to 4 independent access patterns. The HAMMING system can go beyond 4 patterns, but will become *in*efficient, in this sense, if the number n of access patterns becomes too high. The value of this efficiency upper bound n depends on the selectivity of the average access pattern, the total number of objects in the collection, the properties of the disks used, and the average object size [87]. Figure 7.1, adapted from [87], gives some values for n.

As the HAMMING system maximises the average I/O speed to near the theoretical limit for any system that does not trade away space [86], figure 7.1 provides a good indication of the efficiency scalability limit, in the number of access patterns that can be efficiently handled together, of *all* possible systems that do not trade away space.

Figure 7.1 shows that the disk based prototype, with its limit to 4 access patterns, is often quite far away from the access pattern scalability limit of Martin Schaller's system, and thus often quite far away from the access pattern scalability limit of all possible systems that do not trade away space. As discussed in section 6.8.2, one reason for this is that the disk based prototype was designed to satisfy a constraint that is not satisfied by the HAMMING system design. The disk based system supports the constraint that event iteration has a fixed order, an iteration order which was determined by when the events were entered into the offline system (section 5.3.1). The HAMMING system

Total number of objects	Average selectivity 1%	2%	5%	10%	50%
10^6	46	33	21	16	13
10^9	110	68	42	30	22

Figure 7.1: Upper bound to the number of access patterns that can be efficiently clustered for at the same time in the HAMMING reclustering system. 'Efficient' means that the average I/O speed should remain at least 75% of the speed of sequential reading. The numbers in this table are for object sizes from 1 – 8 KB, with data located on a Seagate Elite disk. Figure adapted from [87].

does not follow a fixed iteration order determined from the outside. Instead, in an analysis framework based on the HAMMING system, the HAMMING system layer has the freedom to dictate the event iteration order for any job. The HAMMING system makes good use of this ability to change the iteration order: this degree of freedom enables it to maximise clustering efficiency to near the theoretical limit.

The disk based prototype can (re)cluster many types independently, exactly because it uses a fixed, globally determined event iteration order. Reclustering many types independently, while still allowing objects from different types to be accessed in a single job, is not supported by the HAMMING system, which deals with jobs on a single type only. As noted before in section 6.8.2, the theoretical limits of multi-type reclustering systems are not well understood, but their limits are likely to be significantly below the upper bounds in figure 7.1.

Figure 7.1 shows that, if one is to scale to hundreds of independent access patterns, then any system that never trades away space will not be able to achieve efficient clustering performance. It can thus be concluded that, if a system is to scale to hundreds of independent access patterns, while keeping the I/O performance efficient, that is while keeping an I/O performance of at least 75% of that of sequential reading, this system will have no other option than to trade away storage space.

The tape based system developed in this chapter does indeed trade away space. The design decision to trade away space was made, however, before the results in [87], on which the above strong conclusion is based, became available. The decision was made earlier, based on discussions with Martin Schaller about some of his preliminary results and observations.

7.2.2 Trading away space

This section briefly covers some simulation experiments, performed in this designer's Ph.D. project, which were done to estimate the space overheads in systems that trade away space to preserve I/O performance. The experimental results shown below are for 100 access patterns on a large object set. The access patterns were modelled to reflect physics analysis characteristics as discussed in chapter 3. It was found that the size of

the access patterns, that is the number of objects in them, is an important determining factor for the space overhead. This factor was varied in the experiments. Different sets of 100 patterns were constructed. The one with the largest patterns had patterns that select 50% down to 5% of all elements in the object set, the one with the smallest patterns had patterns from 0.1% down to 0.01%. Figure 7.2 shows the experimental results for different clustering systems. The 'storage size' shown is in multiples of the original object set size.

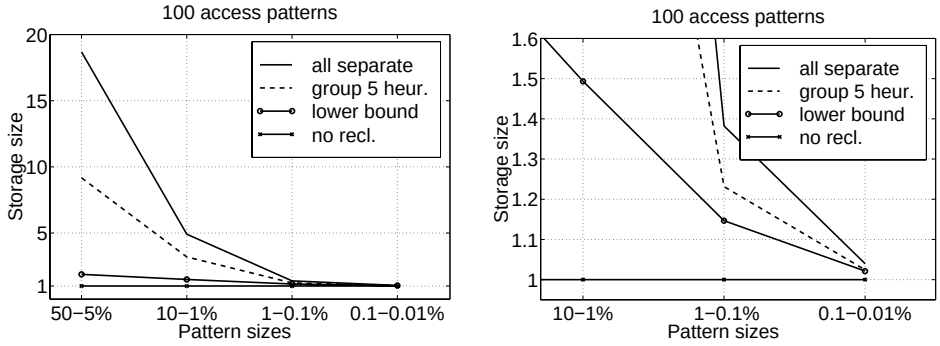


Figure 7.2: Storage space overheads, caused by different clustering systems that trade away space, when clustering for 100 access patterns of various sizes. On the 'storage sizes' axis, each of the four points is labelled with a range of access pattern sizes: this is the range in which the sizes of the 100 access patterns in question lie. An access pattern size (number of objects in the pattern) is expressed as a percentage of the size of the original object set. The right hand graph shows a detail of the left hand graph. The graphs were created in simulation experiments. See the text for an explanation of the four different curves shown.

In figure 7.2, the **no recl** curve shows the storage efficiency of doing no reclustering at all: the storage size remains at 1, which is the size of the original object set. With no reclustering being done, and the original object set being stored in a single collection, the I/O speed for any particular access pattern is the speed of selectively reading the objects in the pattern from the single collection. Compared to the optimal speed of sequential reading, the selective reading speed can be very low, depending on the object size and the selectivity (see section 4.4.5).

The **all separate** curve shows the storage efficiency when the objects in each pattern are clustered separately: each pattern has its own collection, containing all objects in that pattern. This implies that the I/O speed is close to that of sequential reading for every pattern.

The **lower bound** curve shows the (theoretical) storage efficiency of a system that maintains, in addition to the original data set, some collections in which the objects in all access patterns are stored such that every object only has a single copy in these collections. As discussed in section 7.2.1, it is not clear what the I/O performance of this theoretical system would be.

The **group 5 heur** curve shows the storage efficiency of a system in which a heuristic is used to divide the 100 access patterns into groups of 5, so that the members of each group have a large overlap. Then, a batch reclustering type strategy is used to cluster the objects in each group of patterns into at most 31 (2^5-1) collections. This batch type reclustering strategy ensures that, inside each group, every object is only stored in one of the collections for that group. Also, the I/O speed will be close to that of sequential reading. A detailed discussion of the heuristic and optimisation used is outside the scope of this thesis. The **group 5 heur** curve is included in figure 7.2 because it gives a reasonable indication of the space overheads in a viable reclustering system for many access patterns. Some other viable systems were also simulated, most produced curves not much different from the **group 5 heur** curve, none produced noticeably better curves.

Looking at figure 7.2, it can be seen that, when the average size of the access patterns goes down, the difference in storage overhead between the worst case (**all separate**) and best case (**lower bound**) becomes smaller. This is caused by the fact that, as the patterns get smaller, the probability of having a large overlap between the object sets in different patterns goes down.

The conclusions that can be drawn from the **group 5 heur** curve in figure 7.2 are somewhat disappointing. First, in the case of very large access patterns (left in the left hand graph), a viable reclustering system for 100 access patterns seems simply too costly in storage space: compared to a system with no reclustering, it adds a disk space overhead, and thus disk cost overhead, of a factor 5 or more. Second, in the case of very small access patterns, the **group 5 heur** curve does not outperform the storage efficiency of the **all separate** curve much. It is hard to see how one could justify the added software complexity of the **group 5 heur** system: just storing all access patterns separately seems to be a reasonable strategy too.

7.2.3 Design choices

Based on the above observations, the following design choices have been made while developing the tape based system. First, as the goal was to scale to tens to hundreds of access patterns, the system is designed to trade away storage space to gain speed. Second, the focus is on optimising for access patterns that are small: in the range of 10% down to 0.01% of the complete object set. The large size difference between the whole data set and the patterns makes it natural to consider both the disk and the tape storage layers. Third, the scope of the design is restricted: it is designed that the system design effort should not aim at developing new, complicated methods by which to eliminate the duplicate storage of objects present in multiple patterns.

Recall that two reclustering strategies were developed in chapter 6: filtering and batch reclustering. Batch reclustering eliminates duplicate storage, but it has a scalability limit to 4 access patterns. The third decision implies that no improvement on batch reclustering would be developed for incorporation in the tape based system. The tape based system only incorporates the filtering operation from chapter 6. This filtering operation works on the 'current' access pattern only, it therefore has no direct scalability problems when the number of overall access patterns to be optimised increases.

7.3 Basic design of the tape based system

The basic design of the tape based system is created by combining (a) elements of 'normal' tape management systems, with (b) the basic storage management policies in chapter 5, and (c) the filtering optimisation in chapter 6.

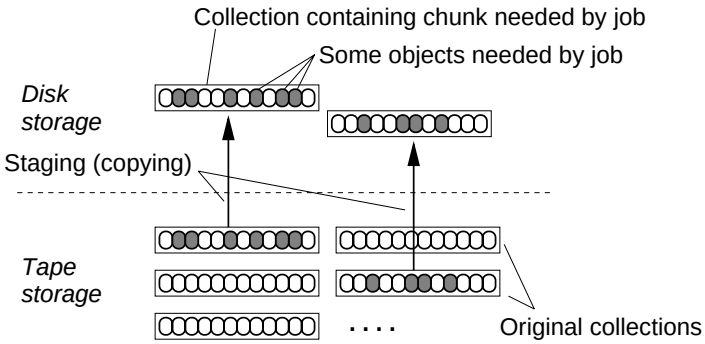


Figure 7.3: Storage organisation of a normal tape management system. All the elements in this figure are also present in the tape based system. The names and terminology used in this figure are those of the tape based system.

7.3.1 Elements derived from normal tape management systems

Figure 7.3 shows the elements of a 'normal' tape management system. Systems with these elements are currently in production use at sites that store large scientific datasets. For example, CERN is currently using the HPSS [58] system, which has the above structure. Some sources on the use and optimisation of these systems are [95] [96] [97] and [58].

All the elements of a normal system in figure 7.3 are also present in the tape based system. In the figure, the elements are also labelled using the terminology of the tape based system. The figure shows two storage levels: disk and tape. The tape level contains a set of collections (called files in most tape management systems), which together hold all objects in the system.

The unit of transfer between the disk and tape level is the collection. If an object is needed by a job, the collection holding it is staged (copied to the disk level) first, and then accessed by this job. In this chapter, staging is defined as a copy operation: staging of a collection on tape creates a new collection on disk, with contents that are identical to those of the collection on tape.

7.3.2 Basic clustering policies

In the tape based system, as in figure 7.3, objects are clustered into collections according to the basic storage management policies in chapter 5. Clustering is by chunk and

by type: every collection holds objects of a particular type, that exist for the events in a particular chunk.

The tape based system maintains a set of *original collections* on tape. Conceptually these collections hold the 'original copies' of all physics objects present in the system. For every chunk ID and type ID combination that exists with objects in the system, there is exactly one original collection on tape that contains *all* physics objects present for that chunk and type. The original collections are never deleted. The tapes they are stored on can even be physically write-protected: this makes the original collections serve as a kind of built-in backup set, which protects against software failure.

7.3.3 Farming policies

To achieve I/O and CPU parallelism, the tape based system uses the farming policies outlined in section 5.3.2: a job over an event set S is divided into chunk-level subjobs. There is one subjob for each chunk C that contains any events in S. A physics job on the tape based system specifies, at the start, the complete contents of its S, and the (type identifiers of the) objects it wants to read for every event in S.

7.3.4 New elements in the tape based system

Using the filtering operation in chapter 6, the normal system in figure 7.3 is extended with two new optimisations: *cache filtering* and *tape reclustering*. The resulting storage organisation is shown in figure 7.4. Note that both operations create new collections, but in a way which preserves the clustering constraints dictated by the basic storage management policies in chapter 5: in that every new collection still only contains objects of a single chunk and a single type.

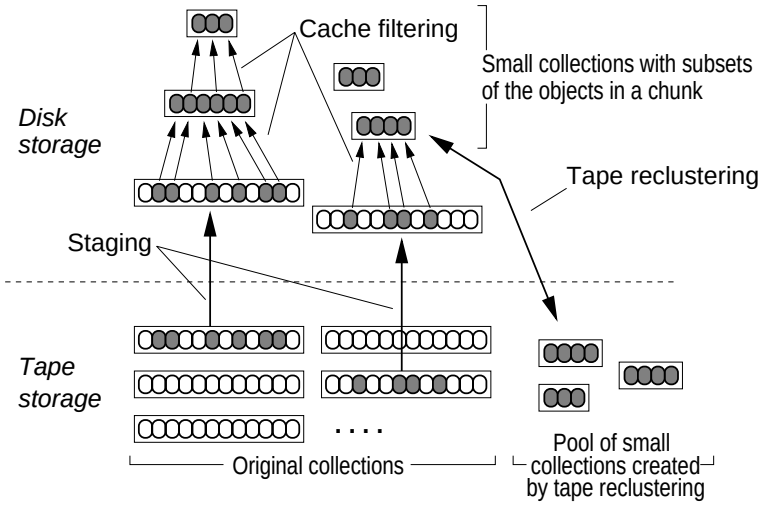


Figure 7.4: Storage organisation in the tape based system.

7.3.5 Cache filtering

A cache filtering operation is done while a subjob is running: some or all of the objects read by the subjob are copied into a new collection. Cache filtering is done with the filtering operation described in section 6.4. It is called *cache* filtering here because the operation only acts on collections in the disk cache, not on those on tape. As illustrated in figure 7.4, cache filtering can be done recursively. The filtering operation from section 6.4 suppresses the filtering of objects read by the subjob to the new collection if these objects are already present in another collection on disk. Thus, to some extent, cache filtering will prevent the needless duplication of objects that are present in multiple access patterns. Its storage efficiency is therefore better than that of the **all separate** system in figure 7.2. This better storage efficiency is probably not critical to the success of the tape based system, but it is too nice a 'spin-off' from the disk based system to ignore.

In the tape based system, the cache filtering optimisation can improve performance in three ways. First, like the filtering operation of the disk based prototype, it improves the I/O efficiency of disk access by producing a better clustering arrangement. Second, cache filtering produces the collections that can be used in tape reclustering. Third, cache filtering can be used to improve the storage efficiency of the disk cache. Following the usual caching terminology, an object is called 'hotter' if it is accessed more often. The storage efficiency of the disk cache is better if more 'hot', and fewer 'cool' objects are stored in it. In the tape based system, the 'hot' objects are those contained in the (currently used) access patterns. In figure 7.4, the 'hot' objects are grey, the cold ones are white. A cache filtering operation concentrates 'hot' objects into a new collection. If the old collection from which these hot objects were extracted is then deleted, the storage efficiency has increased. The cool objects present in the old collection have been removed from the disk cache, so it effectively uses less space to store the same set of hot objects.

7.3.6 Tape reclustering

In tape reclustering, comparatively small collections on disk, containing some hot objects, are copied to tape, before they are deleted from disk in cache replacement. If some of the objects in such a tape reclustered collection on tape are needed again, while they are not present anymore on disk, then this tape reclustered collection can be staged again, instead of staging the comparatively larger original collection. This way, tape reclustering makes tape I/O more efficient: more 'hot' objects are transferred for every MB that is staged. The exact policies for tape reclustering are discussed in section 7.7.3.

7.3.7 Indexing system used

As noted above, every collection in the tape based system contains objects of a single type, for events in a single chunk, only. This strict chunk-level separation makes indexing and scheduling problems much more manageable. The indexing of the collections present in the system is straightforward. A high-level index is maintained,

which can resolve the combination of a type ID and a chunk ID to the set of all collections which contain objects of that type for events in that chunk. For every single collection, whether on disk or tape, a *collection content index* is kept on disk. This collection content index is simply a list of all event IDs for which the collection contains a physics object. The content indices can be used to decide, for example, which collections should be read by a subjob, which objects should be filtered, and which collections should be staged.

7.3.8 Type-level separation

Like the disk based system, the tape based system manages the objects of each type independently, in separate sets of collections. If a subjob needs more than one type of physics object per event, the tape based system will need to synchronise access to multiple collections containing objects of different types. As seen in section 5.6.1, this synchronised access does not cause any I/O overhead if all the collections are on disk. However, if some collections on tape are also needed, these will have to be staged first. The staging of different collections, for the same chunk but for different types, would have to be synchronised to achieve the best storage efficiency in the disk cache: if staging operations are too far apart, then the already-staged objects will just sit on disk for a long time, taking space, without being useful yet. In the prototyping efforts for the tape based system, this synchronisation issue was not explored in depth. Given the time constraints, other issues, with higher associated risks, were considered to be more important. The current system design, and the validation studies that were performed, assume that every job needs only one type of physics object per event.

In literature, some detailed research exists on the synchronisation issue, see for example [37].

7.4 Planning of the detailed design phase of the tape based system

In the design of the tape based system, the creation of the basic design was relatively straightforward. Given the initial design choices of section 7.2.3, the creation of the basic design, as shown in figure 7.4, only took a few days. The hard part of the design process was the creation of the schedulers that steer the different data movement processes found in figure 7.4, so as to optimise the overall performance.

At the start of the detailed design phase, the problem of creating these schedulers was considered carefully, and a particular design methodology was chosen as a result. The methodology relies heavily on simulation. Below, the design problem is considered first, then the design methodology and simulation framework are introduced, followed by a risk analysis.

7.4.1 Designing the schedulers

The tape based system needs a number of schedulers to steer the different data movement processes found in figure 7.4. The main problem in designing these schedulers is that effects of the individual schedulers will strongly interact with each other. The interactions will be strong no matter how the scheduling problem is decomposed into tasks for individual schedulers. For example, the scheduling algorithm that steers the staging process influences the set of collections present on disk, and the properties of this set influence cache filtering. Cache filtering requires disk space, this leads to cache replacement, and the cache replacement rate again influences the demands on the tape staging process. Feedback loops like the one above are present for most major scheduling activities in the system.

In this chapter, a separation of concerns is called *clean* if the individual concerns (design problems) obtained from it can be treated in full isolation of each other. In the detailed design phase, a separation of concerns has to be made: the whole scheduling task has to be decomposed in subtasks for different individual schedulers. However, this separation of concerns cannot be a clean one. Because of the feedback loops between most major scheduling activities, the actions of most schedulers cannot be treated in full isolation from the actions of most other schedulers. The separation step can try to minimise the circular dependencies between design problems for the individual schedulers, but it cannot fully eliminate them.

The lack of a clean separation between schedulers rules out the use of some attractive design methods which could otherwise be used. If a design problem is cleanly separated, one can often use a design method that constructs a provably good, or even optimal, solution from first principles. For example, for the disk based prototype design problems in sections 6.2 and 6.4, a clean mathematical description could be found, allowing further development along formal lines, into a provably good or even optimal solution. For a tertiary storage management system, in which sets of pending requests are present, it is sometimes possible to use formalisms like queueing theory do derive a provably good or optimal solution for a request scheduling problem. This presupposes however that the problem is simple and isolated, which is not the case for the scheduler design problems in the tape based system.

With the use of 'first principles' methods, leading to provably good or optimal solutions, considered infeasible, a stepwise refinement methodology was chosen instead. In the chosen methodology, simulation experiments are used to drive the refinement of the schedulers.

7.4.2 Methodology used in the detailed design

The concrete steps followed in the detailed design of the tape based system were as follows.

- As a first step, the basic system design of figure 7.4 was extended with a number of active components and schedulers that would steer the dynamic processes in the basic design.

- Then, detailed designs were made for the components, and initial designs of the schedulers, while completely disregarding most interactions between different processes in the system. The detailed designs of the active components are described in section 7.6.
- The creation of the individual components was followed by a 'big bang' integration step. All components were put together, and the tuning parameters in all schedulers were set to some plausible initial values. This way, a first version of the whole system was obtained. In simulations, this version was able to process a query workload successfully, though not very efficiently.
- Following the 'big bang' integration step, the schedulers were refined to globally deliver good performance, using simulations with likely workloads. This last phase of the design process was an iterative one, with many simulate-adjust cycles. After integration, simulation was used to refine the schedulers and tune the system into efficiency. The resulting scheduling algorithms and tuning parameter settings are described in section 7.7. The simulation phase was supported by a simulation framework developed specifically for that purpose, which allowed parameters and algorithm details to be altered quickly.

Of course, design methods which involve a 'big bang' integration step have a deservedly bad reputation. A particular disadvantage of the above method is that there is little feedback on the feasibility of the system design until the 'big bang' step, when all components are integrated and the first complete simulation can be run. Before embarking on this route, considerable time was spent in searching for more cyclic, gradual methods (without much success), in analysing the associated risks (see below), and in developing techniques to reduce these risks. Risks were reduced by maximising the probability of having a successful 'big bang' step, and by minimising the amount of work that would be lost if the 'big bang' step would be unsuccessful. The amount of work at stake was minimised by aiming at the implementation of a full simulation of the tape based system, rather than the implementation of a full 'production' version that would act on real physics objects. A full simulation takes much less work to implement, while yielding almost as much knowledge about the feasibility of a 'production' system in the end. The risk of having a failing 'big bang' step was mitigated by making robustness of individual system elements an important design goal of the first two steps. The active components and scheduling algorithms were designed so that they would keep working, and ensure some degree of progress, no matter how bad the decisions of other scheduling algorithms are. The execution framework for the active components was also designed for robustness. In particular, global deadlocks between active components, which might be caused by bad scheduling, are detected by the framework, and resolved by a global deadlock resolution algorithm. Starvation of individual system components is also prevented. However, the framework does not prevent all forms of non-progress, no matter how bad the decisions of the scheduling algorithms are. For example, there will be a live-lock situation if the scheduling algorithm for staging operations decides to always stage collections that are not needed: in that case there will be a lot of tape I/O, but no overall progress.

7.4.3 Simulation framework

As discussed in chapter 3, the exact properties of the CMS physics analysis workloads, to be put on the CMS physics analysis system from 2005 onwards, are currently not known. Some workload parameters are known fairly well, others have large error bars. To make sure that the design fits the future workloads, it therefore has to be evaluated over a very large parameter space. If the design uses simulation, this implies that a large number of simulation runs is necessary, so that the whole space is covered.

In the design effort for the tape based system, the creation of a large set of likely physics workloads, by which to tune and validate the system, was a significant undertaking itself. The simulation driven development work was done using some 6 sets of 30 workloads each, with each set covering a 4-dimensional workload parameter space. A special framework was built to support massive simulation experiments. Typically, a single simulation experiment would involve some 300 workload and system parameter combinations. For each combination, a discrete event simulation of the tape based system under these parameters was run. Custom tools were developed to visualise the results of a single simulation experiment, consisting of some 300 discrete event simulation runs for different parameter combinations, on a single page of graphs. Using additional tools, the differences between two simulation experiments could be visualised.

In exploring the workload and tuning parameter space to investigate design options, some 140 simulation experiments were run, consisting of some 25000 individual discrete event simulations of the tape based system, using about 600 CPU hours. Section 7.5 has more details on the system workloads used in the simulations.

7.4.4 Risk analysis for the detailed design

The specifics of the above design methodology were largely developed on the basis of a risk analysis effort, done before the detailed design of the system was started. The decision to actually proceed with the design was made after estimating the risk of failure in proceeding. The risk of failure considered to be sufficiently low to make proceeding a good choice. The failure estimate was based on a number of observations. The three most important ones are discussed below.

The first observation is connected to the simulation experiments for trading away space (section 7.2.2). During these studies, various input parameters were varied to study their effect on the resulting space overhead. Some of these parameters were the average size of an access pattern (shown in figure 7.2), the average percentage of events eliminated in a single cut, and the similarity between analysis efforts by different users. It was found that changing these parameters almost always had a linear effect on the result. Only rarely did a small change in a parameter result in a large change in the resulting space overhead, and in the differences between space overheads for different clustering systems. The lack of non-linear effects made the analysis of space overheads over a large parameter space tractable: it made it possible to look at different parameters in isolation. This suggested that tuning the tape based system into efficiency, based on workloads covering a large parameter space, would be a tractable problem

too. It was guessed that there was only a low probability of encountering intractable, non-linear effects when changing parameters in the tape based system. In the end this guess turned out to be right: only a few non-linear effects were encountered.

The second observation was made after studying the literature on existing and proposed normal tape management systems [58] [95] [96] [97]. Looking at these sources, it was estimated that the development efforts for the complete architectures presented in them took in the order of 4 to 24 man-months, rather than many man-years. The time available for the detailed design of the tape based system was about 6 months. It was estimated that this would be long enough to develop at least an initial version of the tape based architecture, to a state where at least some initial conclusions could be drawn. This estimate also turned out to be right: some definite conclusions could be drawn at the end of the 6 months.

The third observation was that the tape based system can be seen as a caching system, and that caching systems are often amazingly robust. Very simple cache replacement algorithms can do a good job for very complex workloads. Completely different cache replacement algorithms often yield about the same performance. This is often frustrating for caching researchers: new replacement algorithms that look dramatically more 'intelligent' than plain LRU (Least Recently Used) often turn out to yield only slightly better overall performance. But from a risk analysis standpoint, it is a good thing: given this state of affairs it was concluded that it would be likely that even a very naive LRU strategy, when encoded as the initial cache replacement algorithm for use in the 'big bang' integration, would give reasonable first results. This guess turned out to be correct, in that the initial caching performance in the 'big bang' system was reasonable, and in that further tuning was a tractable problem.

The active work on the above risk analysis took some two working weeks. However, doing a risk analysis is also partly an intuitive process, and intuition also cannot be rushed. For the whole risk analysis process, the total time between the first consideration of options and the final decision was some three months.

7.5 Simulation details

This section describes the system and workload parameter space used to tune and validate the tape based system. As discussed in section 7.4.3, the creation of a large set of likely workloads was a significant undertaking in itself. During the simulation effort, the parameter space under consideration was refined a number of times. In these refinement steps, some parameters that turned out to only slightly influence the system performance were eliminated from consideration, while some new parameters were added, and existing parameters were extended to cover a wider range. This section only describes the final parameter space, the space used for final tuning and for the validation studies. This space has four dimensions, corresponding to the following four parameters.

- The size of the disk cache (see section 7.5.1).
- The initial clustering efficiency (see section 7.5.2).
- The workload type (see section 7.5.3).

- The average number of times that a job is performed on the same object set (see section 7.5.3).

These four parameters are discussed in more detail below. In the simulations, all other hardware and workload parameters are either fixed at a single likely value (for example the 'size of a tape in GB' parameter), or they vary stochastically over a likely range (for example the 'size of the object set requested by a job' parameter).

In the remainder of this section, the hardware and size parameters are discussed first. Then the 'initial clustering', the initial assignment of events to chunks, is discussed. The quality of the initial clustering is an important determining factor in the system performance. Finally, the workloads themselves, that is the simulated physics jobs, are discussed.

7.5.1 Hardware and size parameters

After 1 year of running the CMS detector, the CMS physics analysis system will hold about 1 Petabyte of physics data, with the events divided over some thousands of chunks. To keep the simulation effort manageable, it was decided to simulate only a 'slice' of the CMS physics analysis system. The slice used in the simulations only concerns the access to one type of physics object for every event, and the slice contains exactly 200 chunks. This number of 200 was found to be high enough so as not to bias the results towards small systems, and low enough to keep the simulation time manageable.

Number of chunks in simulated slice: 200 Chunk size: 1 GB Object size: 100 KB
Tape space for original chunk files: 200 GB Disk space for staging/caching: 4–40 GB Tape space for tape reclustering pool: 40 GB
Tape robot: Storagetek Powderhorn Tape drive(s): Storagetek Redwood SD3 Tape size: 10 GB Tape speed for sequential reading: 11.1 MB/s Tape change time: 19 sec Seek from front to end of tape: 42 sec
Disk speed for sequential reading: 555 MB/s

Figure 7.5: Size and hardware parameters of the simulated system slice.

Figure 7.5 shows the full set of size and hardware parameters used in the simulations. In the simulations, it was found that tape, not disk, was usually the bottleneck. The disks were found to be very lightly loaded in the majority of simulations. As they almost never formed a bottleneck, the disks were not simulated in great detail.

A problem with choosing the tape parameters in figure 7.5 was that no predictions of tape parameters in 2005 were available. Predictions being made by technology experts,

inside CERN [55] and outside CERN [57], generally contain no hard numbers for tape price and performance beyond a few years in the future. This reluctance to make concrete long-term predictions can be explained partly by the smallness of the 'industrial strength' tape drive market (see section 4.5.1), it might easily be disturbed by 'turbulence' from events in the much larger hard disk and consumer multimedia storage markets, and partly by the fact that all past predictions that did put a specific year, before 2000, on the availability of breakthrough optical or holographic storage products have turned out to be wrong. Nevertheless, experts agree that, if one disregards the possibility of dramatic future technology breakthroughs, the best guess is that speed and capacity advances in magnetic tape technology will keep pace with the speed and capacity advances in hard disk technology [57] [54]. This is because the continued improvements in hard disk read/write head technology, which yield the speed and capacity advances for hard disks, are usually directly applicable to tape drive read/write heads too. The R&D efforts of tape manufacturers, with respect to tape drive heads, are largely limited to doing the necessary adaptations of new hard disk head technology. It is therefore not expected that future developments of tape will out-pace the hard disk curve, nor is it expected that these developments will fall far behind. In summary, though there are no long-term predictions of tape parameters available, one can assume, as a first order approximation, that these parameters will keep pace with hard disk parameters. In the simulations for the tape based system, the tape and hard disk parameters of today were therefore used. This yields the best possible prediction of the relation between the parameters of disks and tapes in 2005. Both tuning or validation are based on comparing alternative system configurations, rather than looking at absolute I/O speeds. A conversion factor for the speed of the simulated hardware to the speed expected of 2005 hardware is therefore not used in the simulations. As an aside, it is expected that disks will be about a factor 4 more powerful (in MB/s per $) in 2005, see section 4.4.7.

The tape parameters chosen in figure 7.5 are those of a Storagetek system recently taken into production use, at CERN, for the storage of physics data from some current experiments with high data rates. The chunk size was chosen to be a reasonable one for this tape system.

The 'size of the disk cache' parameter in figure 7.5 varies from 4 to 40 GB, this number is computed to be 2% to 20% of the tape size. Note that the CMS experiment estimates to have a disk space size of 'several hundred TB' in 2005, see section 2.6.3. A part of this will be used as a cache for tape based data: a good estimate would be 100 TB, or 10% of the size of the 1000 TB dataset on tape. The current the MB per $ price ratio between disk and tape is about a factor 10 [53] [54]. In conclusion, the most likely estimate for the size of the disk cache is therefore 10% of the tape size.

The difference in I/O speed between disk and tape is taken to be a factor 50: this reflects the current MB/s per $ price ratio between disk and tape [54]. The factor 50 implies that the disk speed for sequential reading is $50 * 11.1 = 555$ MB/s. Note that the expected I/O bandwidth of a 4–40 GB commodity disk farm is lower than 555 MB/s: it is in the range 5-200 MB/s, depending on the number and size of the hard disks used. This discrepancy indicates that, because of economic reasons, a system slice with 200 collections of 1 GB is unlikely to get exclusive use of a single tape

drive. This fact could be reflected in the simulations by modelling long periods in which the tape drive is not available for the slice. This level of detail was not added to the simulation framework however. After studying the timing of disk and tape access in the simulations, and considering that the simulated systems were mostly tape-bound, it was concluded that adding such 'not available' periods would slow down all possible alternative system configurations about equally much. Both tuning or validation are based on comparing alternative system configurations, rather than looking at absolute run times. Thus, tuning and validation results would not change if this level of detail were added.

7.5.2 The initial clustering

The *initial clustering* is defined as the assignment of events (physics objects) to chunks, and original collections, storing these complete chunks, to tapes. It is assumed that this initial clustering remains fixed over the lifetime of the tape based system, or at least over the time windows of the simulations performed. Changing the initial clustering of a huge scientific dataset on tape would be a very expensive operation; currently no known sites with such scientific datasets perform such operations.

The initial clustering should be chosen to minimise number of chunks needed ('hit') by any future job. If fewer chunks are needed, less staging operations are needed, and the disk cache efficiency immediately after staging will be higher too. The quality of the initial clustering, the degree to which the number of chunks hit by any job is minimised, is thus an important determining factor in the overall system performance.

The problem of creating a good initial clustering is not limited to tape based physics analysis systems: many tape based applications with query-like workloads have the same optimisation problem. Consequently, the construction of good initial clusterings for various data types and workloads is an active area of research [95] [61] [38]. Such research is not necessarily aimed at datasets on tape, but often applies to any storage level, even to the internal storage structure of a large index, see for example [39] [40] and [41] for overviews.

Initial clustering algorithms for n-dimensional spatial or spatio-temporal datasets have been researched widely. Such algorithms usually interpret the objects in the dataset as points in some n-dimensional space. They then try to put objects which are close together in this space in the same chunk, and chunks with close-by objects near each other on the same tape. This is a natural approach when jobs analyse sets of objects located in a particular region of space.

For other types of datasets and workloads, one generally tries to map objects into a similar n-dimensional space, using object attributes to construct object coordinates, so that jobs again analyse sets of objects located in a particular region of this constructed space. In the HENP GC project (the High Energy and Nuclear Physics Grand Challenge project) [61], this option is being explored for high energy physics datasets. A special 'workbench' has been developed to help in finding the best possible attributes to be used in constructing the space for the clustering algorithm.

For any application, the efficiency of the initial clustering that can be achieved has a high dependency on the type of data and the workload. In the optimum case, a job that needs to read $y\%$ of the objects hits exactly $y\%$ of the chunks. Several factors make this optimum unattainable for most practical systems. First, the exact object properties that jobs will use in their object selection are often not known in advance, and thus have to be guessed. Second, if there are many, say 50, possible object selection criteria, then an initial clustering performed using the implied 50-dimensional space is only efficient if all jobs always use narrow ranges on all selection criteria. If this is not the case, if many criteria (dimensions in the space) are left unconstrained, then the number of hit chunks in a 50-dimensional space will be very large, sometimes even approaching the total number of chunks. To keep this effect within bounds, one will have to lower the dimensionality of the space, eliminating most of the 50 criteria as dimensions. So one has to make a tradeoff between keeping the dimensionality low on the one hand, and reflecting as many selection criteria in the dimensions as possible on the other hand. Third, if there are many jobs that request small object sets, object sets maybe even smaller than the chunk size, this inevitably leads to inefficiencies due to a low object hit rate inside the hit chunks. One can choose to make the chunks smaller in such cases. However, chunks cannot be made arbitrarily small. Tape systems leave a gap between each file (collection representation) on tape for indexing and synchronisation purposes. If chunks are too small, the files are too small, and most of the tape space will be used for such gaps, rather than for storing actual data. In [98], gap sizes from 64 KB to 141 MB are reported for different tape systems.

For physics datasets, all three of the above causes for a decreased clustering efficiency apply. No detailed estimates of the clustering efficiency achievable in high energy physics data analysis have been made. In this designer's Ph.D. project, the issue was not investigated in depth, because the HENP GC project [61] was studying it already. The HENP GC project, which also takes part in the RD45 collaboration (see section 2.8), includes some of the world specialists on initial clustering algorithms. Unfortunately, at the time of creating the workloads for the tape based system, and also at the time of writing this chapter, the HENP GC project had not yet drawn any conclusions on the initial clustering efficiencies achievable on tape in high energy physics applications.

To cope with the uncertainty as to the initial clustering efficiency in the tuning and validation of the tape based system, three different initial clustering arrangements were used in the simulations. Figure 7.6 shows the average chunk hit rates of these initial clusterings, called (a), (b), and (c), for the jobs in the workloads of section 7.5.3. The 'job size' on the x axis of the figure is the size of the object set requested by the job.

Initial clustering (a) was constructed as a 'best realistic case', by simulating job access patterns on a two-dimensional dataset. The simulated jobs always inspect sets of objects lying within a rectangle in the space, and the initial clustering algorithm 'tiles' the dataset space into 200 rectangular chunks. Curves (b) and (c) were constructed by systematically worsening the clustering arrangement of (a). These curves are more representative of the expectations for high energy physics workloads. Curve (c) is not a 'worst case' curve: for the real worst case curve, the chunk hit rate would always be 100%, no matter what the job size.

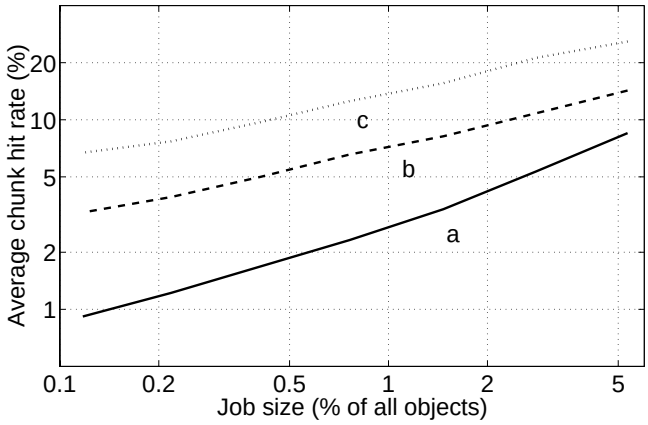


Figure 7.6: Chunk hit rates of three initial clusterings.

From the chunk hit rate curves in figure 7.6, one can compute curves for the average object hit rate *inside* the hit chunks. This number equals the size of the object set requested by a job divided by the sum of the sizes of the chunks hit by the job. Curves for this object hit rate are plotted in figure 7.7.

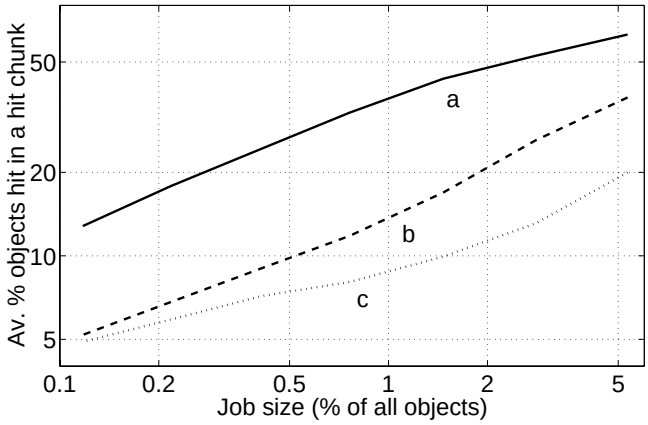


Figure 7.7: Object hit rates inside hit chunks of three initial clusterings.

For the purpose of evaluating the optimisations in the tape based system, this object hit rate inside the hit chunks is a better measure of the initial clustering quality than the chunk hit rate of figure 7.6. The object hit rate in a hit chunk is a direct measure of the inefficiency associated with caching that whole chunk in the disk cache. If the hit rate in some chunk for a job is 10%, then 90% of the disk space is wasted if that whole chunk is cached for the purpose of satisfying future jobs on the same object set, so cache filtering could potentially save 90% of the cache space used. For the mix of jobs in the workloads, these three clustering arrangements give average object hit rates inside the hit chunks of 30% (a), 13% (b), and 9% (c).

7.5.3 Simulated workloads

The simulated workloads used in the tuning and validation of the tape based system are multi-user workloads with the following properties. A user will pick an object set (access pattern), and then run a number of jobs on this object set. The user waits for each job to finish before submitting the next job. The average number of jobs run by a user on a single object set is a workload parameter, which varies from 2 to 32. When these jobs are all run, a new object set is picked, and so on.

In every workload, the sizes of the object sets picked by the users vary over a very broad range: from 0.03% to 6% of the complete dataset, with an average of 0.34%. As seen in chapter 3, this broad range is a very fundamental property of high energy physics workloads.

A user working on small object sets is expected to have a short think-submit-wait cycle, with most or all of the queried objects remaining in the disk cache. Such a user is expected to submit more jobs per day than one working on large object sets, where jobs are tape bound and take hours or days to run. The workloads reflect this expectation as follows: in every particular workload, jobs of size $n/2$ occur twice as often as jobs of size n. The simulated users do not actually submit more jobs if they encounter a system with a faster response time. To allow for a meaningful comparison between different systems running the same workload, the number and frequency of jobs in a particular workload is the same no matter what the system performance. The simulated system response time only influences the actions of the simulated users to the extent that a simulated user always waits for the current job to finish before submitting the next job.

When a user works on a small object set, the shorter think-submit-wait cycle of the associated jobs is also expected to have the result that the user will run more jobs to study the object set, before moving on to the next object set. In the workloads, this expectation is reflected as follows. The average number of jobs run per object set increases with a factor 1.17 if the object set is twice as small. Coupled to this factor, the number of different object sets with a certain size increases with a factor $2/1.17 = 1.71$ if they get twice as small.

The amount of job parallelism in the system is bounded with the constraint that all users together never have more than 10 jobs submitted concurrently. On average, in the period that one user is running jobs on a chosen object set, about 12 other users will pick new object sets and start running jobs on them.

Two types of workloads are used in the simulations: *physics workloads* and *generic workloads*. The physics workloads have inter-job dependencies as found in physics analysis, the generic workloads have no such dependencies.

In a physics workload, the different object sets picked by users are not picked independently. Instead, following the physics analysis process properties described in section 3.3, smaller object sets are created by applying a cut predicate to a larger object set, a set about twice as large. Only the largest, initial, object sets are chosen independently. The collaboration between physicists in doing physics analysis is also modelled as follows. The initial, largest object sets are picked and analysed by large groups of users, who jointly prepare and run jobs. These groups split into smaller groups, and later

into individual physicists, as the remaining object sets, obtained by applying cut predicates, become smaller. The object set interdependencies in the physics workload very much improve cache efficiency: not only are jobs highly correlated, so that less space is needed to cache all the object sets of many jobs, but the first job on a newly obtained, smaller object set will also generally find all objects it needs already in cache, because the new object set was obtained by applying a cut predicate to an object set that was studied in the recent past.

Generic workloads were introduced to analyse the performance of the tape based system for non-physics applications. In a generic workload, users pick new object sets independently and at random. The workloads do not have any 'hot spots' in the complete object set: all objects are equally likely to be picked when a user chooses an object set to run jobs on. The absence of hot spots is somewhat unrealistic: most applications will have them. Some simulations with workloads including such hot spots were performed: it was found that including them slightly decreased the speedups coming from the introduction of cache filtering and tape reclustering. The detailed performance results in section 7.8 are for generic workloads without hot spots, because no compact way was found of reporting on the exact parameters of the simulated hot spots, and their interaction with the initial clustering system. Because of the way in which the initial clustering arrangements were simulated, tests with the generic workloads are somewhat similar to tests with query type 3 in the Sequoia 2000 storage benchmark [36].

7.6 Active components in the tape based system

Figure 7.8 shows the different active components (rounded rectangles) in the tape based system, and their interactions with each other and with the collections in the store. The components invoke schedulers (not shown) to make their optimisation and scheduling decisions. The schedulers are discussed in section 7.7. In the remainder of this section, the different active components in figure 7.8 are discussed.

7.6.1 Job component

A physics analysis job specifies an event set S, a physics object type T, and a piece of physics analysis code. The job semantics are that for every event in S, the system must deliver the physics object of type T that belongs to this event to the physics analysis code.

A job, after being submitted and scheduled for execution, does little more than creating one subjob for each of the chunks that have events in its object set S. Below, these chunks are called the chunks 'hit by the job'. The job then waits for these subjobs to complete. When all subjobs have completed, the subjob results can be aggregated into the overall job result, which can be returned to the user. Many jobs can be executing at the same time. The system does not limit the number of concurrently executing jobs: concurrency limits are only placed on subjobs. Because of this, the limits in job concurrency are determined by size and behaviour of the user community: users

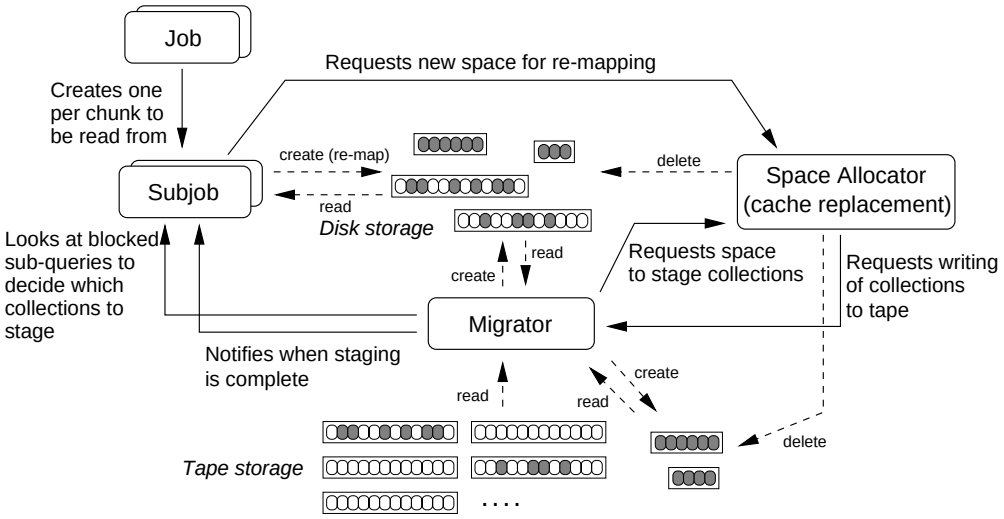


Figure 7.8: System components and their interactions in the tape based system.

generally won't submit a new analysis job until they have seen the result of their current job.

7.6.2 Subjob component

A subjob works at the level of a single chunk. It reads the objects in that chunk which are needed by its parent job, and feeds them to the physics code for processing. Apart from delivering objects to the physics code, a subjob can also perform a cache filtering operation.

The subjob acts as a framework in which the physics code executes. The subjob controls the iteration over the requested events in its chunk. As in chapter 6, iteration is always in the object storage order of the original chunk. For every requested event e, the subjob event loop calls the physics code, supplying the event ID of e as an argument. The physics code in turn will call the storage management layer in the subjob framework, to request a copy of the object of type T for this event e.

When started by its parent job, a subjob will immediately examine the collection content indices of the collections on disk, to determine if all objects it needs are present on disk. If not, the subjob 'blocks', it will suspend its execution to wait until all needed objects are present. It is the responsibility of the migrator (see the next section) to ensure that blocked subjobs eventually become un-blocked.

When a subjob, possibly after having been blocked, finds all needed objects present on disk, it invokes a scheduler to compute from which collections to read these objects, and whether to do any cache filtering. The scheduler uses the first algorithm from section 6.2 to decide from which collections to read, and the algorithm from section 6.4 to compute the filtering function to be used. The tuning of the latter algorithm is

discussed in section 7.7.1.

After having locked the collections to be read against deletion by cache replacement, the subjob requests permission from a central scheduler (not shown in figure 7.8) to start reading objects. The central scheduler ensures that not too many subjobs are busy doing disk I/O at the same time. For example, on the system configuration in section 5.10, a good limit would be to restrict the concurrency to a maximum of some 400 subjobs. In the simulation experiments of this chapter, a limit of 10 subjobs was used. When permission to read is obtained, the subjob will first request some free disk space if cache filtering is to be done. Then, the subjob iterates over the needed events in its chunk. Objects are read from existing collections, processed by the physics code, and possibly written to a new collection in cache filtering.

7.6.3 Migrator component

The collection migrator manages the tape drive(s) in the system, and migrates collections between disk and tape.

To decide which collection to stage next, the collection migrator examines all blocked subjobs. Many hundreds of subjobs may be blocked at the same time. Sometimes, many subjobs (of different jobs) are all blocked, waiting for the staging of objects from the same chunk. The collection migrator partitions the blocked subjobs into clusters. Every cluster is a group of blocked subjobs waiting for objects of the same type in the same chunk. For every cluster, the collection migrator identifies a single collection on tape, whose staging would allow all subjobs in the cluster to un-block. This pooling of tape requests from different sources is known as *query batching*, and it can lead to dramatic savings [99], especially for workloads with many concurrent large jobs.

The scheduling policies used by the migrator are discussed in section 7.7.4.

7.6.4 Space allocator component

The space allocator manages the free and allocated space on disk and tape. The space allocator is also responsible for choosing which collections to migrate down to tape in tape reclustering. The space allocator will satisfy requests for disk space in the order of arrival. In all but extreme cases, the requests can be satisfied immediately. The space allocator can fail to grant requests for space in two cases: (1) all collections on disk are currently locked for reading, so that none can be deleted and (2) some collections could be deleted, but the space allocator wants to copy them to tape in chunk reclustering first, before deleting them.

The various scheduling policies used by the space allocator are discussed in sections 7.7.2 and 7.7.3.

7.7 Schedulers in the tape based system

As discussed in section 7.4, the design of the schedulers in the tape based system involved an extensive simulation phase, in which the scheduling algorithms and tuning

parameters were refined to work well over a large workload parameter space. This section documents the end result of this process, the final scheduling algorithms and parameters. It also describes some interesting parts of the design process itself. Often, a simulation using 'what if' scenarios could be used to drive a design decision. This section does not describe all design alternatives considered. For example, some 14 variant cache replacement policies for collections on disk were considered, only a few are discussed below.

7.7.1 Tuning of cache filtering

The scheduling algorithm from section 6.4 is used to steer cache filtering. This algorithm has a single tuning parameter t, which controls how eager the system is to initiate a filtering operation. If the value is higher, the access to existing collections needs to be more sparse to trigger a filtering operation. Thus, the higher the value, the less filtering is performed. See section 6.4 for the definition of this parameter. In a tuning exercise, the value $t = 2$ was chosen: this value gave the best overall performance for the workloads considered. In a simple case, say if there is only one existing collection with x objects on disk, the value $t = 2$ implies that a subjob only filters the objects from that collection into a new collection if it needs less than $\frac{1}{3}x$ objects from that collection. Compared to $t = 2$, the value $t = 4$ worked almost equally well: surprisingly, performance was not found to be very sensitive on the tuning factor t.

7.7.2 Cache replacement on disk

This section describes cache replacement algorithm for the collections on disk, as used by the space allocator. Replacement of data from the disk cache is done at the collection level, as true object-level decisions would be too expensive to compute. Also, object-level cache replacement might lead to extreme fragmentation of data on the disk cache, degrading disk performance to that of random disk access. Such a fragmentation risk is absent with the collection-based replacement policy that is used: related hot objects are kept packed together in a few collections, and these collections can be accessed using sequential or sparse reading.

The cache replacement policy on disk has to achieve some conflicting aims. First, a recently used collection should of course be retained as long as possible. But second, a collection from which objects were recently filtered should be deleted as quickly as possible, so that an important goal of the cache filtering operation, creating a tighter packing of hot objects in the disk cache, is actually achieved. Thirdly, if a job with a size many times that of the disk cache size is executed, no attempt at caching the collections staged for this job should be made, but they should be deleted as quickly as possible, to maximise the available cache space for smaller jobs. To reconcile these conflicting aims, a special cache replacement policy called UBC (Usage Based Cooling) was developed, and refined through simulation.

The basic idea that underlies Usage Based Cooling is that different collections should have different cache replacement rates, depending on their use. For example, if a collection is staged from tape, or created by a subjob that was blocked on tape, it should

be replaced from cache soon, to satisfy the third goal of cache replacement above. It should be replaced much sooner than a collection that was recently accessed by a subjob that did not block on tape. Below, the complete UBC algorithm is described, with the tuning parameters optimised for the system and workload parameter space of section 7.5.

In UBC cache replacement, any use of a collection adds a 'heated token' to this collection (figure 7.9). A heated token has an initial temperature t_i and a cooling rate c. When the token is t units of time old, its temperature will have dropped to $\frac{t_i}{c \cdot t+1}$. This cooling function approximates a physical cooling process. Similar decreasing functions would also work well in the replacement algorithm. Tokens added in different types of use get different t_i and c values. Specific values were chosen in a tuning exercise; it was found that performance is not very sensitive to the specific values, some examples of this are included in the discussion below.

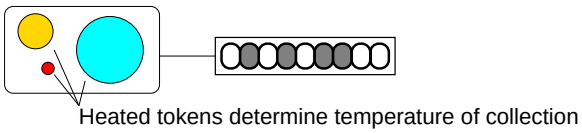


Figure 7.9: Adding heated tokens to a collection.

In UBC cache replacement, the 'coldest' collection is replaced first. The temperature of a collection is defined as the temperature of its hottest token plus half the temperature of its second-hottest token. This (somewhat arbitrary) temperature definition was found to work best in a tuning exercise, but just taking the temperature of the hottest token also works well.

If a subjob reads a collection, it adds a 'standard' token with $t_i = 1$ and $c = 1$. An additional token twice as hot ($t_i = 2$) and cooling twice as slowly ($c = \frac{1}{2}$), is added to a collection if immediate re-execution of the subjob would cause the collection to be read again. The effect of this is, for example, that a new collection created by cache filtering lasts about four times as long as the collection(s) read to create it. The cooling rate c of an added token is multiplied by 40 if the subjob was blocked on the staging of a collection. This much faster cooling rate causes an object which has been freshly staged to remain in the cache for a short while only, unless the object is quickly accessed again by a subsequent job. If five jobs are run subsequently on a small set of objects, the first job stages these objects and adds fast-cooling tokens to their collections. The second job, if run quickly enough afterward, before the collections are replaced from cache, adds slow-cooling tokens. These slow-cooling tokens keep the collections in cache for a much longer time, protecting them against replacement by collections staged when processing subsequent large jobs. The cooling rate multiplication factor of 40 was obtained in a tuning exercise, it was found that a factor of 200 worked almost equally well.

It should be noted that the tuning parameters above were chosen to work well over the whole workload parameters space used. If the system is deployed with a particular workload set covering a smaller parameter range, it would make sense to re-tune the above UBC parameters. Some experiments were performed with a 'weighted' UBC

replacement policy, in which tokens added to larger collections cool faster. It was found that a switch to such a policy had little effect on performance.

7.7.3 Tape reclustering

Tape reclustering is done by copying some of the collections on disk, which were produced by cache filtering, to tape. When the space allocator, which is also responsible for cache replacement, determines that a collection is soon to be replaced (deleted) from the disk cache, it invokes a scheduler to determine if the collection is to be copied to tape. If the collection is to be copied, the space allocator includes this collection in a batch of write requests to the migrator, and will then hold off deleting the collection from disk until it has been written to tape.

Selection of collections

There is no obvious method of deciding whether or not a collection should be copied to tape in tape reclustering before deletion on disk. Obviously, only the 'best' collections should be copied to tape, but when the space allocator offers a collection up for consideration, it has already decided itself that this is one of the 'worst' collections it has on disk!

In searching for a good selection algorithm, the following approach was used. Simulation is used to find some predictive characteristics of collections, characteristics that could be successfully used to select collections for tape reclustering. A system was simulated in which, unrealistically, all collections about to be deleted from disk are copied to tape, at zero cost. To deal with the tape space requirements of this indiscriminate copying, the pool of tapes set aside for tape reclustering is relatively large. In simulation runs of this system, the collections that are actually staged back onto disk were examined, to find some identifying characteristics for these collections, separating them from the collections that are never staged back. However, a good predictive identifying characteristic, that could be used in a selection method, was not found. Characteristics explored were the collection size, the collection access history, the way in which the collection was created on disk (filtering or staging), and various properties of the heated tokens present on the collection.

With a sophisticated method not being found, a very simple selection heuristic was adopted instead. This heuristic is based on the observation that a very large collection C, containing, say, 80% of all objects in the corresponding chunk, should obviously not be written back to tape. Instead of staging the contents of C, when objects in C are needed, the corresponding original collection can be staged, at only a slightly higher cost in tape I/O and disk space. The benefit of having C on tape is relatively low, so it is very unlikely that the investment made in copying C to tape in the first place can be recuperated, let yield a profit. Following this reasoning, the selection heuristic is based on a size cut-off. All collections smaller than 20% of the chunk size (the original collection size) are selected for copying to tape. This value of 20% was found in a tuning exercise. It was found that 40%, for example, worked almost as well. Some heuristics that were more complicated than a size cut-off were tried, but

none performed noticeably better. Under the principle that no unnecessary complexity should be added to the system, the simple size cut-off heuristic was kept as the final choice.

Tape management

For the pool of reclustered collections on tape, the following management policy is used. A set of tapes is reserved exclusively to hold these collections. One tape at a time is filled, collections are written on this tape sequentially. When all tapes in the pool are full, the oldest tape is recycled: all collections on it are deleted and writing starts again from the front of the tape.

The above scheme amounts to a 'least recently created' cache replacement policy: the collections that are least recently created are deleted first. Of course, a collection deletion policy closer to 'least recently used' would potentially be more effective at maintaining a set of useful collections on tape, if a way could be found to keep the associated free space fragmentation on tape in check. To investigate the potential benefits of other replacement policies on tape, a 'what if' simulation was used to determine the performance of 'least recently used' replacement under the (unrealistic) assumption that there is no performance loss due to fragmentation. The 'least recently used' policy, simulated under this assumption, outperformed 'least recently created' with factors of 1.0 (no improvement) up to 1.2 for the system and workload parameter space of section 7.5. From these small improvement factors, it is concluded that the simple 'least recently created' tape replacement policy is an appropriate choice. A better policy may be possible, but it is unlikely to be better by more than a factor 1.2.

7.7.4 Staging of collections

In any tape-based system, it is important to minimise the randomness of tape I/O as much as possible, because tape mounts and seeks are expensive operations. The staging policy of the tape based system has to make a tradeoff between (1) minimising mount and seek operations and (2) optimising the order in which subjobs are serviced. To investigate the tradeoff between (1) and (2), a 'what if' simulation was used to determine the performance of various staging policies, under the (unrealistic) assumption that there are no mount and seek overheads. This assumption eliminates the effects of the staging policy on (1), so that the effects of various policies on (2) can be studied in isolation. Surprisingly, it was found that the choice of staging policy had very little effect on performance in these 'what if' simulations: even a purposely bad policy (least work first) would only decrease performance with a factor 1.2 compared to the best policy found.

Based on the above results, the following simple staging policy was chosen. This policy ignores the considerations of (2) and just aggressively minimises tape mounts and seeks (1). The policy cycles over all tapes in the system in a fixed order. When the next tape is reached, and a tape drive becomes available for reading, the stager inspects all collections on the tape to see if any of them could satisfy the staging needs of any cluster of blocked subjobs. (See section 7.6.3 for a discussion of subjob clustering.)

If at least one such collection exists, the tape is mounted. Then, all collections which have blocked clusters of subjobs waiting for them are staged, in the order of their position on tape. This results in a sequential pattern of seeking and reading on the tape. When the last collection has been staged, the tape is rewound and unmounted. Note that this policy works with any number of tape drives. The fixed cycling order ensures that subjobs are never blocked indefinitely.

When it stages a collection, the migrator adds a special heated token to the staged copy of this collection. The token is very hot, but has a fast cooling rate. It therefore has the effect of locking the copy of the collection into the disk cache for a short time, so that all blocked subjobs in the corresponding cluster get a chance to un-block, and lock the copy of the collection for reading, before it is deleted by cache replacement.

7.8 Validation

When the tuning of the tape based system was complete, a validation study was made to determine the usefulness of its new optimisations, cache filtering and tape reclustering. The usefulness of these optimisations is determined by comparing, for the same workloads, the speed of the tape based system, which has these optimisations, with the speed of a more basic system, the *baseline system*, which lacks these optimisations. For any single workload, the comparison yields a speedup factor, the speed difference between these two systems. The higher the speedup factor, the more effective the optimisations.

7.8.1 Baseline system

The 'baseline' system is obtained from the tape based system by disabling both optimisations. The baseline system is thus a 'normal' tape management system in the sense of section 7.3, with a storage organisation as shown in figure 7.3: the baseline system always deals with complete (copies of) original collections.

The cache replacement policy for the baseline system is LRU (Least Recently Used). Interestingly, when trying UBC cache replacement in the baseline system, it was found that it performed worse than LRU, in fact it performed about as bad as random cache replacement. Conversely, LRU performs reasonably well, though not as well as UBC, in the full tape based system, especially for larger cache sizes. The tape staging policy for the baseline system is the same as that of the full tape based system. Through tuning experiments it was determined that this staging policy is a good one for the baseline too.

7.8.2 Speedup factors found

Figure 7.10 show some simulation results for different workload parameter combinations, all with the disk cache size parameter set at 10% of the dataset size. The columns in the tables are as follows.

Physics workloads, cache size = 10% (20 GB)						Generic workloads, cache size = 10% (20 GB)					
Av jobs/ oset	Av % o hit/ hit ch	CF	TR	CF+ TR	Cx2	Av jobs/ oset	Av % o hit/ hit ch	CF	TR	CF+ TR	Cx2
2	30 (a)	1.3	1.0	1.3	2.2	2	30 (a)	1.3	1.0	1.3	1.3
2	13 (b)	2.3	1.1	2.5	1.7	2	13 (b)	1.3	1.0	1.3	1.2
2	9 (c)	2.4	1.2	2.7	1.4	2	9 (c)	1.2	1.0	1.1	1.1
8	30 (a)	1.5	1.0	1.5	3.4	8	30 (a)	1.7	1.0	1.8	2.4
8	13 (b)	4.1	1.0	4.0	2.9	8	13 (b)	2.9	1.0	2.9	1.4
8	9 (c)	11	1.1	12	1.7	8	9 (c)	2.8	1.0	3.0	1.3
32	30 (a)	1.8	1.0	1.8	3.0	32	30 (a)	2.9	1.1	3.1	3.4
32	13 (b)	5.8	1.1	6.2	4.9	32	13 (b)	6.0	1.1	6.7	1.6
32	9 (c)	47	1.0	48	2.1	32	9 (c)	7.2	1.1	8.2	1.4

Figure 7.10: Speedup factors for physics workloads and generic workloads.

- **Av jobs/oset** is a workload parameter: the average number of times that a job is performed on the same object set.
- **Av % o hit/hit ch** is the 'initial clustering efficiency' workload parameter. The value is the average percentage of objects hit per hit chunk for the initial clustering, with the designation of the initial clustering, see section 7.5.2, in brackets.
- **CF** is the speedup factor due to cache filtering. It is the speed difference between the baseline system and the tape based system with only cache filtering enabled, and tape reclustering disabled.
- **TR** is the speedup factor due to tape reclustering. It is the speed difference between the tape based system with only cache filtering enabled, and the full tape based system with both cache filtering and tape reclustering enabled.
- **CF+TR** is the speedup factor due to both cache filtering and tape reclustering. It is the speed difference between the baseline and the full tape based system with both optimisations enabled.
- **Cx2** is the speedup factor of doubling the disk cache size in the baseline system. If this factor is very low, then the workload in question apparently will not benefit much from better caching. In that case, no big benefits should be expected from cache filtering and tape reclustering too, as both are caching-type optimisations.

For the physics workloads in figure 7.10, the speedup factor of adding cache filtering and tape reclustering **CF+TR** ranges from 1.3 to 48. Physics workload speedups are low in the best initial clustering case (a), but as noted in section 7.5.2, initial clustering (a) is unrealistic for physics workloads. It can thus be concluded that, for likely physics workloads, the tape based system shows good speedups over the baseline, from 2.5 to 48.

For the generic workloads in figure 7.10, the speedup factor of adding cache filtering and tape reclustering **CF+TR** ranges from 1.1 to 8.2. For generic workloads in which

there are on average only 2 jobs that visit the same object set, the speedups are very low, from 1.1 to 1.3: this is true both for **CF+TR** and for just adding more physical cache to the baseline, **Cx2**. It is concluded that these workloads do not have sufficient repetitive access to respond well to caching: they could be processed equally well with a tape staging system which does not include a cache. It is interesting to note that, for the physics workloads in which objects sets have two jobs on average, the speedups for **CF+TR** and **Cx2** are much higher, from 1.3 to 2.7. Apparently, the dependencies between the different object sets chosen in physics workloads produce enough repetitiveness in object access to allow for performance gains through caching, even if every object set is only visited twice on average.

Comparing the **CF** and **TR** columns in figure 7.10, it is clear that cache filtering is responsible for the major share of the speedups in the tape based system, at least for the disk cache size parameter value of 10% of the dataset size. Tape reclustering shows very disappointing speedups in these tables, from 1.0 to only 1.2. Section 7.8.5 looks at the case of tape reclustering more closely.

The **Cx2** column in the tables shows that switching from the baseline system to the tape based system usually outperforms doubling the disk cache size of the baseline: it outperforms doubling the disk cache size for all initial clustering configurations except (a).

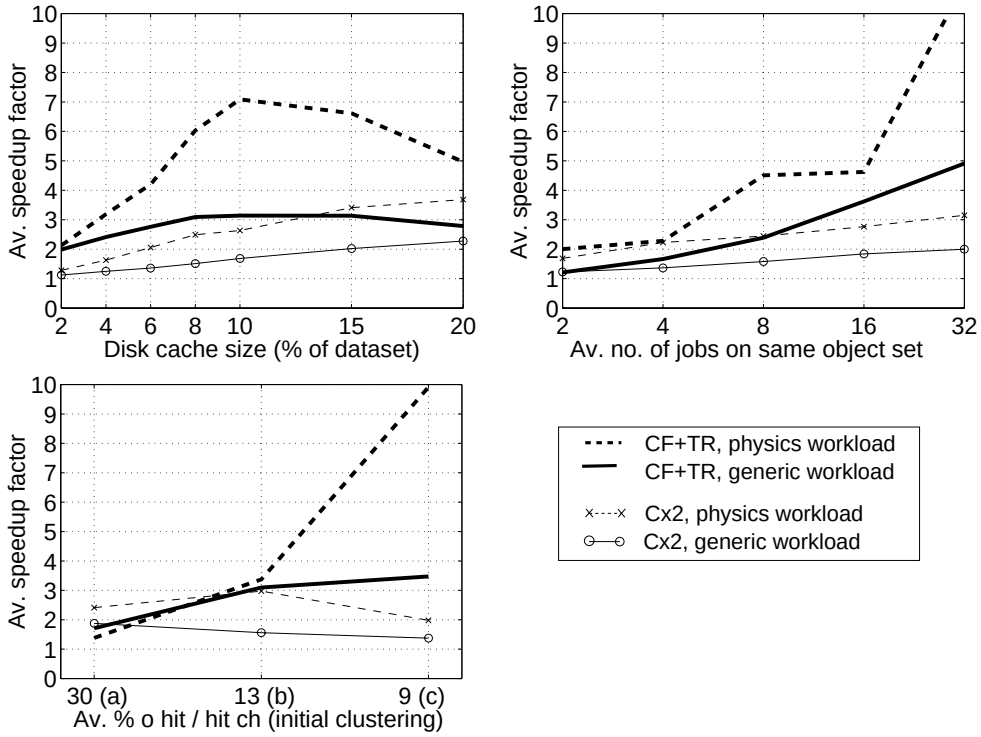


Figure 7.11: Dependence of different speedup factors on several parameters. Every speedup factor shown in a graph is the average over all combinations of the parameter values on the x-axes of the other two graphs.

7.8.3 Effects of workload parameters

The graphs in figure 7.11 provide an alternate view of how various workload parameters affect the speedup factors. In the top, left side graph, which shows the dependence on the disk cache size, speedup factors first increase, and then decrease again as more cache space is added. This later decrease is as expected: it can be explained as follows. Note that the performance gain of adding more cache space inevitably moves to zero as the cache size approaches the size of the original dataset. This is true for both the tape based system and the baseline system, though the baseline will generally reach the point of zero gains later. Once both reach this point, the systems should be equally fast, because in processing the same workload, both only need to stage all initial collections needed in the workload once. Therefore, at this point the speedup factor for the tape based system is exactly 1. In the top, left graph of figure 7.11, all curves must thus decrease to reach 1 eventually.

The top, right side graph in figure 7.11 shows the dependence of the speedup on the number of times a job is repeated on the same object set. Again, results are as expected: if jobs are repeated more often, the workload will respond better to caching, and thus better to the improved caching of the tape based system. The bottom, left side graph in figure 7.11 shows, as expected, that the tape based system gives higher speedups if the initial clustering efficiency is lower. This graph also shows that the dependence of the speedup factor on the initial clustering efficiency is high for physics workloads, and low for generic workloads. At the time of writing, no explanation for this difference in dependence has been found.

7.8.4 Tape access overheads

In table 7.5, the hardware performance characteristics of the simulated tape system are given. For example, the time needed to change a tape is 19 seconds, excluding the time needed to rewind the currently mounted tape first. In the simulations, the tape mount, seek and rewind overheads *per staged collection* were studied. Results are as follows. First, as expected, the scheduling policies of the tape based system generally cause many collections to be staged from a single tape when it is mounted. This leads to an average tape mount/seek overhead of about 10 seconds for staging an original collection, and 5 seconds for staging a collection created in tape reclustering. The per-collection overhead for the tape reclustered collections is less because a larger number of collections is staged, on average, from a tape holding tape reclustered collections. The mount, seek, and rewind overheads were fairly insensitive to the simulation parameters. Note that the time needed to read the data in a complete original collection from tape, assuming mounts and seeks have been done, is 90 seconds. This number of seconds is much more than the overheads. It can therefore be concluded that the tape robot hardware is used efficiently: most of the time is spent in useful data transfer. The use of tape cartridges with a lower storage capacity, which have a better average seek and rewind speed because the tapes are shorter, won't make the tape based system (or the baseline) much faster. This is important because longer tapes have a better GB per $ price/performance ratio.

7.8.5 The case of tape reclustering

It is clear from the **CF** and **TR** columns in figure 7.10 that cache filtering adds much more to performance than chunk reclustering. Note that tape reclustering can only be added to a system that already has cache filtering, as it depends on cache filtering to create the relatively small collections it moves to tape. Being able to stage these relatively small collections, instead of the larger original collections, does save tape resources. However, the resources needed to write the smaller collections to tape in the first place are considerable. Typically, in the simulations, half to all of the time saved by reading the smaller collections instead of the original collections is spent in writing the smaller collections to tape.

Physics workloads, cache size = 4% (8 GB)					
Av jobs/ oset	Av % o hit/ hit ch	CF	TR	CF+ TR	Cx2
2	30 (a)	1.2	1.1	1.3	1.4
2	13 (b)	1.5	1.2	1.8	1.3
2	9 (c)	1.5	1.4	2.1	1.2
8	30 (a)	1.4	1.1	1.6	2.2
8	13 (b)	2.5	1.3	3.1	1.6
8	9 (c)	2.5	2.1	5.4	1.3
32	30 (a)	1.8	1.1	2.0	2.9
32	13 (b)	4.7	1.3	6.2	2.0
32	9 (c)	5.4	2.0	11	1.4

Generic workloads, cache size = 4% (8 GB)					
Av jobs/ oset	Av % o hit/ hit ch	CF	TR	CF+ TR	Cx2
2	30 (a)	1.2	1.1	1.3	1.1
2	13 (b)	1.2	1.1	1.3	1.1
2	9 (c)	1.2	1.0	1.2	1.1
8	30 (a)	1.6	1.1	1.8	1.4
8	13 (b)	2.0	1.3	2.7	1.3
8	9 (c)	2.0	1.3	2.6	1.2
32	30 (a)	1.7	1.3	2.2	1.5
32	13 (b)	2.6	1.6	4.2	1.3
32	9 (c)	2.6	1.8	4.6	1.2

Figure 7.12: Speedup factors for a relatively small disk cache size.

The addition of tape reclustering only causes good performance gains if the disk cache size is small. Figure 7.12 shows **TR** speedup factors with a disk cache size of 4%: these range from 1.0 (no speedup) to 2.1. The speedup factor of tape reclustering gets even better if the disk cache size is still smaller. Figure 7.13 shows the speedup gain with respect to the disk cache size in more detail.

In section 7.5.1, it was noted that the most likely estimate for the size of the disk cache is 10% of the full dataset size. Thus, it must be concluded that the tape reclustering optimisation is most likely uninteresting for the CMS storage manager in 2005. Put in another way, based on current estimates and results, I would advise that only the cache filtering optimisation is implemented in the 2005 production system, not tape reclustering, unless implementation manpower is very abundant.

There are a number of cases under which tape reclustering would become interesting to implement in the CMS storage manager. Some of these are outlined below.

First, if the disk cache size is 4% or lower than the size of the dataset, then implementation would be interesting. The difference between this 4% and the most likely disk cache size of 10% is only about a factor 2, well within the error bars of the estimates of the disk space available to CMS in 2005, and well within the error bars of the estimate of the fraction of the disk space that will be available as a disk cache. So it is

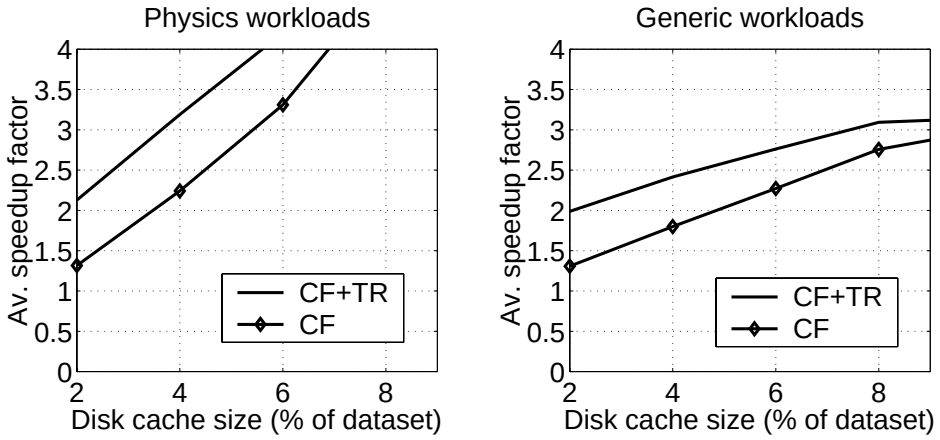


Figure 7.13: Average speedups for CF *and* CF+TR*, for physics and generic workloads.*

entirely possible that the size of the disk cache for the tape based system be 4% or less of the size of the dataset on tape, in which case tape reclustering would be interesting to implement.

Second, even if the disk cache size is 10%, the actual workload parameters for CMS in 2005 may lie beyond the likely ranges assumed in the simulations. Trends in the simulation results suggest that higher performance gains for tape reclustering, with a disk cache size of 10%, are possible if the initial clustering is less efficient than the worst initial clustering simulated here, or if the average number of jobs on a single object set is higher than 32.

Third, tape reclustering may be attractive for reasons other than performance. Even with a **TR** speedup factor of 1.0, tape reclustering does have the effect of lowering the number of times that a tape holding original collections needs to be mounted. Lowering this number could be attractive for safety reasons: it reduces the probability that original data will be irretrievably lost because a tape with such data is damaged irretrievably due to mechanical problems in the tape drive.

Fourth, tape reclustering, even with a speedup factor of 1.0, can be attractive because it allows for greater flexibility in the use of tape hardware. If tape drives which are incompatible with the tapes holding the original collections happen to become available, these tape drives could be added (maybe temporarily) to the system, and used with their own set of tapes for tape reclustering. With more overall I/O capacity, the system performance is increased.

Fifth, it could make economic sense to store tape reclustered data on a separate tape system (or optical disk system) that is small, but fast compared to the tape system holding the initial chunk collections. If tape reclustered collections can be accessed much faster, the performance gains of tape reclustering may improve enough to make its use economically attractive as an alternative to buying more disk cache space, or more tape drives in the tape system holding the original collections. The economic trade-

offs are complex however, and have not been fully explored in this project. In some initial studies, no hardware configuration was found with parameters good enough to successfully exploit this opportunity.

Sixth, tape reclustering might be made more efficient by giving the migrator (section 7.6.3) the option of staging two or more separate tape reclustered files, instead of only one, to satisfy the needs of a cluster of blocked subjobs. Staging two collections to satisfy a cluster implies however that the first collection has to be kept in the disk cache until the second collection has also been staged. This decreases the amount of cache space available for other purposes, so there is a tradeoff involved. To estimate the maximum performance gains that could be obtained by allowing the migrator the option of staging multiple collections for one cluster, a 'what if' simulation was done with a system in which this option was provided, and in which, unrealistically, the already-staged collections in cache would use up zero cache space until they were actually used. Over all workload and system parameter combinations shown in figure 7.11, a maximum speedup of 1.4 was found with respect to the current tape based system, with an average of 1.2. It can therefore be concluded that, more research on this option might yield improvements in tape reclustering performance which are high enough to make tape reclustering attractive over a wider range of disk cache sizes.

Finally, more research into a selection algorithm for tape reclustering (see section 7.7.3) could yield an algorithm that is much better than the simple heuristic used by the tape based system. Such a better algorithm could make the use of tape reclustering attractive in cases where it currently yields little additional performance. More research in the current tape based system framework may discover a possibility that was overlooked in the effort described in section 7.7.3. Also, the selection process might improve if predictive 'hints' supplied by end users were introduced in the system. The option to use such 'hints' has not been explored.

7.8.6 Validation conclusions

The validation shows that using the tape based system for CMS storage management will likely give worthwhile improvements over a normal, baseline tape management system. For likely physics workloads, the tape based system shows good speedups over the baseline, from 2.5 to 48. The cache filtering optimisation is responsible for the largest part of these speedups. The tape reclustering optimisation shows somewhat disappointing speedup factors for the most likely disk cache size. For relatively small disk cache sizes, of 4% or less of the dataset size, tape reclustering does show some worthwhile speedups.

7.9 Related work

The tape based system builds on experience from existing tape management systems [99] [96] [37] [95] [97] [58], especially with respect to cache replacement and staging policies. Other related work is the research on initial clustering algorithms, as discussed in section 7.5.2.

Apart from publications produced earlier in this designer's Ph.D. project [100] [101], there is no known related work on tape management system optimisations similar to cache filtering and tape reclustering. Caching at a granularity below the chunk level, when data moves to higher levels in the storage hierarchy, occurs in many systems, see for example [96]. At least one existing mass storage management product [102] structures data into small units (atoms, like our physics objects), and allows the application programmer to request (sets of) such data units, rather than complete files. This product uses a caching granularity below the staging granularity, but it does not go down to the 'atomic' level in its caching mechanisms. The use of a true object-level granularity in the tape based system, coupled to optimisers that work at the level of object sets, collections, seems to be new.

Section 6.8 discusses work related to the disk based system. Most of this work, in particular that on 'data summary tapes' and 'view materialisation', is also related to the tape based system.

As noted in section 6.8.3, the dynamic management or materialised views is a research problem very similar to that of the dynamic creation and deletion of (filtered) collections, and this similarity leads to similar architectural patterns. The remainder of this section discusses some striking similarities between the tape based system and the 'DynaMat' system [89], which was designed to dynamically materialise views in a data mining environment. In the DynaMat system, views are dynamically created, either from the original dataset or from existing, larger views, then stored in a cache, and eventually deleted with a cache replacement algorithm. The views in question are called 'multidimensional range fragments': these do not have an 'object' granularity like the collections in the tape based system, but they have a much more fine-grained granularity than the views proposed elsewhere in the data mining literature [89]. The data mining workloads for which the DynaMat system design in [89] is tuned are not directly comparable to the workloads of the tape based system considered in this chapter. Because of this, the best cache replacement algorithm found after tuning in [89] is very different from the UBC algorithm in the tape based system. The work on DynaMat [89] was well received in the database community, so it is possible that it will spark significant future research efforts in this direction. Workloads more closely related to those of the tape based system also occur in data mining. It is therefore possible that future work on systems like DynaMat will yield results that are directly relevant to the architecture of tape storage management systems for high energy physics.

7.10 Conclusions

In this chapter, a complete system for reclustering on disk and tape was developed. The optimisations in this tape based system scale to workloads with tens to hundreds of independent access patterns, created by tens to hundreds of concurrent users.

To prepare for the design of the tape based system, the issues in scaling to hundreds of access patterns were studied. The results of these studies, and also later relevant results obtained elsewhere in the RD45 collaboration, are documented in section 7.2. It can be concluded that, if a system is to scale to hundreds of independent access patterns while keeping the disk I/O performance efficient, this system will have no other option

than to trade away storage space, to use more space than is needed just to store the original dataset.

The basic design of the tape based system was created by combining (a) elements of normal tape management systems, with (b) the basic storage management policies in chapter 5, and (c) the filtering optimisation in chapter 6. A 'big bang' design approach was used for the detailed design, after which the system schedulers were refined to globally deliver good performance, based on simulations with likely workloads. The tape based system incorporates two novel optimisations, cache filtering and tape reclustering, and a special-purpose cache replacement algorithm, Usage Based Cooling, which reconciles the sometimes conflicting aims of caching, cache filtering, and staging.

In section 7.8 the performance of the two optimisations in the tape based system is analysed over a large parameter space. It is concluded that cache filtering yields good speedups in many cases, and very good speedups for likely high energy physics workloads. Cache filtering is especially effective if the initial clustering efficiency is low.

Based on current knowledge, it seems most attractive, for the first production CMS storage manager in 2005, to implement only cache filtering, and not tape reclustering, unless implementation manpower is very abundant. However, if the expected disk cache size drops from the current 10% of the dataset size to 4% or lower, it would also be attractive to implement tape reclustering from the start. If tape reclustering is not implemented from the start, it seems attractive to study the actual workloads, when the first production version of the CMS storage manager runs, to determine if the implementation of tape reclustering would yield a worthwhile performance improvement.

Chapter 8

Discussion and conclusions

This chapter discusses the design methodology chosen for this designer's Ph.D. project, and reviews the project results in a wider context. First, the reasoning behind the choice of the risk-driven design methodology is presented. Then, the body of risks associated with the construction of the CMS computing infrastructure is reviewed. This review focuses on computing and storage management risks, and identifies the risk related contributions of this designer's Ph.D. project. Then, the software artifacts created in the project are described, followed by a discussion of some spin-off results. The chapter ends with an evaluation of the design process.

8.1 Focus on risk reduction

In a normal Ph.D. project, the focus is on *scientific results*, the main output of the project is new scientific knowledge. In a designer's Ph.D. project, the main output is usually an *artifact*, and the focus is on the design of this single artifact. In this designer's Ph.D. project, the goal was not to create a single artifact. A complete, production-quality CMS storage manager implementation is only needed from 2005 onwards. In this project, the focus was on reducing risks through cyclic prototyping [9]. The primary output of the project is therefore the *reduction of the risks associated with CMS storage management* that has been achieved. The artifacts and spin-offs of the project are secondary.

There is an interesting paradox here. The artifact produced in a designer's Ph.D. project, even if it is a software artifact, is usually a very tangible, discrete object. But the 'reduction of risks' that is produced in this project is a very intangible object: it is a change in a probability. It is not even a change in a probability that can be quantified well. Quantifying risks should be possible for 'repeatable' software processes. But the CMS software process is such an uncommon one that a statistical aggregation over similar projects with known results, an aggregation that would yield some numbers, is not possible.

Why should one take something as intangible as the reduction of risks as the main project goal? There are certainly many disadvantages in doing so. For example, there is no clear list of requirements by which to steer the designer's Ph.D. project. End users are seldom interested in risks which have not yet turned into disasters. If you do your work well, the project result is that nothing dramatic will happen in future.

It is hard to 'sell' such project results. Selling results usually amounts to showing the solution first, and then arguing how the solution is important because of this risk.

Given these disadvantages, good reasons are needed for choosing risk reduction as the main project goal. In the case of this project there were a number of reasons: some very compelling, some of a more secondary nature. Most of these reasons follow directly from the properties of the environment in which the project was performed.

The main design parameters of the CMS storage manager (section 1.2), the parameters of the whole CMS physics analysis system even, imply significant risks. The CMS collaboration has recognised this, and has therefore adopted a cyclic prototyping approach for the whole CMS software effort [10]. Risk management is a major theme in the initial prototyping cycles, which last to 2002. With its focus on cyclic prototyping and risk management, CMS is of course following best practices that are increasingly common in industry [103]. Apart from structuring its own software process in a risk driven way, the CMS collaboration also helped initiate, and participates in, a large common R&D effort called the RD45 collaboration (section 2.8), which studies storage management issues. From a CMS point of view, the main purpose of setting up this designer's Ph.D. project was to add manpower to the efforts in RD45. Thus, a focus on risks was already well established at the start of this project.

It would have been possible, in starting up the designer's Ph.D. project, to ignore the focus on risks that was inherent in the initial project proposal written by CMS. It would have been possible to re-define the project mandate, so that it would focus on the creation of some concrete short-term production software artifacts instead. For example, the CMS collaboration is currently performing 'test-beam' experiments to prototype particle detection elements that will be used in the CMS detector. These 'test-beams' require the construction of special-purpose data acquisition software. A focus on such software artifacts would have made the Ph.D. project into a requirements-driven or user-driven one, rather than risk-driven one. Such a re-definition of the project mandate, though probably possible, was not performed for a number of reasons. First, it was felt by the designer's Ph.D. project participants that CMS was entirely right in its focus on risk management, and in its decision to add manpower to risk related R&D activities. Second, the short-term software needs for activities like test-beams, which could offer an alternative, were judged to be changing too fast (in yearly cycles) to offer a stable basis for performing a Ph.D. project, which would last several years. Third, doing research on long-term risks it would allow the Ph.D. project to do justice to both of the main requirements for a designer's Ph.D. [1]: to satisfy the real needs and criteria of some user community, and to demonstrate the ability of the Ph.D. candidate to do autonomous scientific work.

The designer's Ph.D. project is not only risk driven at the highest level: it also uses risk-driven cycles inside. Cyclic prototyping was used, and the project went through three cycles. A cycle starts with the identification of a new risk to explore. There is a large set of risks to choose from. The choice for a particular risk is guided by an assessment of which risks are most urgent. It is also guided by the need to coordinate with other researchers working in CMS and the RD45 collaboration. Once an urgent risk, not yet under close study elsewhere, is identified, it is expressed in terms of a research question (see section 1.4). This research question is then addressed in a single project cycle. The

cycle is finished when the risk under study has been addressed to such an extent that residual risks are small compared to the still open risks. This approach of performing short cycles, driven by the identification of an urgent risk, again corresponds to some best industry practices. The execution of an individual cycle in this designer's Ph.D. project can be compared to the execution of a 'tiger team' sub-project in the sense of Booch [103]. These 'tiger team' projects are initiated inside a larger software project to attack a single risk when the need arises. The sub-project ends and the team is disbanded again when the risk has been sufficiently addressed. Such tiger teams act independently of the mainstream software activities in the larger project, and feed their results back into this mainstream when they are done.

As an alternative to the above risk-driven cycles, a different approach could have been used. It would have been possible to identify some specific risks at the start of the project, and then translate these into requirements for a single software artifact to be created, having the CMS collaboration 'sign off' on these requirements at the start. This way, the project could have been requirements-driven inside, with the decision of mapping risks to requirements being made outside of the project, by the whole CMS collaboration. This option was not chosen however, because it would have made the designer's Ph.D. project much less flexible in synchronising with the environment and coping with unexpected results. As part of his work in an earlier designer's Ph.D. project at CERN [104], Erco Argante also concluded from experience that, when performing a research-like project with no strictly defined up-front requirements, an evolutionary (cyclic) approach is more appropriate than a 'waterfall' approach that starts with an attempt to define complete, detailed requirements first.

8.2 Review of risks, contributions of this project

It is important to note that this designer's Ph.D. project was performed in the context of several larger projects: the CMS software/computing project and the RD45 research and development collaboration (section 2.8). The specific risks addressed in this designer's Ph.D. project were chosen from the larger body of risks associated with constructing the CMS physics analysis system, of which the storage manager is a part. This larger body of risks is reviewed in this section, so as to place the designer's Ph.D. project into context. The review also identifies the contributions, in terms of risk reduction, of this designer's Ph.D. project. The review first discusses the risks related to CMS software as a whole, then the specific risks related to storage management.

8.2.1 Risks related to CMS software

Compared to past software projects in physics, the CMS software/computing project again has a larger scale in people and time than any previously completed project. For example, at the time of writing, a list of 'main coordination responsibilities' for the CMS software/computing project lists 45 responsibilities, and allocates these to 32 different people. The CMS software/computing project is aggressively pursuing the use of the Web and video conferencing as tools to compensate for this increased scale. The use of OO technology is also seen as a key to keeping the coordination

effort manageable. Finally, there is a lot of attention to the scaling and quality of the software repository system that is used.

The CMS software strategy relies heavily on the use of commercial software components in building the system: there is simply not enough manpower to build all components in-house. This use of commercial components has a number of significant risks. First, a selected component may turn out to have insufficient performance or scalability, or simply have too many bugs to be usable. Significant manpower is therefore spent on selecting components, and on validating selected components. The work in chapter 4, on testing objectivity/DB as a layer underneath the storage manager, contributed to this component validation effort. Some bugs in Objectivity/DB were found in this designer's Ph.D. project, but overall the RD45 collaboration has judged the level of bugs to be low enough, and the customer support to be good enough, to make Objectivity/DB a workable solution. Another significant risk with commercial components is that the running time of the CMS experiment, at least 15 years starting from 2005, is much larger than the lifetime of most commercial software products. To cope with this, CMS is focusing heavily on the use of software components for which some kind of standard exists. For example, standard C++ with its standard libraries is used as the implementation language: it is likely that standard C++ programs can still be compiled 20 years from now.

In 1996, the RD45 collaboration considered object databases to be an emerging major software market, a market which would be held stable over a long term, with mutually compatible products, by the ODMG standard which was then being developed (section 4.2.1). Because of this, the strategy of using a commercial object database project as a basis for the CMS object store was formulated, and Objectivity/DB was selected by CMS as the basis for all object storage prototyping efforts in the 1996–2001 timeframe. Currently, in 1999, it is clear that object/relational databases have won the battle against pure object databases: the object database market is currently a niche market at best [105]. Therefore, the work on object persistency in the RD45 collaboration, and in CMS, is shifting more and more towards the production of a *persistency interface*, which should isolate most of the code base from any changes in the database product or technology that is used. Also, the alternative of creating a home-grown object persistency service is being explored [105]. Such a home-grown service would cost significant manpower to produce, but it would have the advantage of ensuring that the persistency interface, and the format in which data is stored, can remain stable during the entire lifetime of the CMS experiment. Stability, or at least backwards compatibility, in the storage format is crucial because it will be extremely expensive to convert many Petabytes of physics data on tape into a new format. Doing one complete conversion every five years or so will be possible, but doing it more often would take too large a fraction of the expected tape I/O capacity.

In any software project, there is the risk of having a software process breakdown. This is a very real risk for the CMS software/computing project: software process breakdowns have occurred in many comparable software projects in high energy physics. Such a process breakdown generally sets back progress by at least a year, and nullifies all organisational and architectural decisions made up to the point of the breakdown. The risk of a breakdown is substantial in high energy physics projects for a number of

reasons. First, only some of the software project members are computing professionals: most are trained as physicists, and they often work only part-time on software. Second, the project participants are geographically dispersed, which makes communication more difficult. Third, compared to the project lifetime, many participants stay only for a short time. Ph.D. students typically stay some 3–5 years, and participants with post-doc type positions usually have little chance of seeing their short-term (2–4 year) contracts extended into full-time appointments. Fourth, the consensus building mechanism which is used to coordinate the activities of the different project participants works slowly, making it difficult to adapt quickly to changes in the environment.

In [103], Grady Booch says the following about very long software projects.

> *Projects of 5 to 10 years exist, though I have seen very few of them succeed, at least measured against the initial scope of the project. In any case, geopolitical issues – not technical ones – tend to dominate, mainly because there is so much at stake.*

These observations apply to some extent to the CMS software/computing project. The CMS computing technical proposal [10] defines ambitions and far-reaching system capabilities, not just in the quantitative system performance parameters (see section 1.2), but also in the qualitative parameter of end user convenience. It is widely recognised however that, while most quantitative goals are hard, the user convenience goals written down in the CMS computing technical proposal are not. User convenience will be offered subject to the constrains of the available hardware performance and the available manpower for the implementation effort. It will not be considered a sign of failure if the actual end user convenience falls short when measured against the initial goals.

With respect to the second observation: political issues do indeed play a large role in the CMS software/computing project. The consensus building process which coordinates the project is very much a political activity. This existence of a political component does not imply however that the whole project is doomed from the start. In stead it implies that the technical participants will have to work hard to create a strong correlation between the political issues and the technical issues. I want to argue that a software engineer involved in a consensus driven project should not stay aloof from the politics. Instead, this software engineer has a professional obligation contribute to the political part of the consensus process, because the shape of this process has a major effect on the technical direction and health of the project.

8.2.2 Risks related to CMS storage management

There are many risks related to CMS storage management. Perhaps the most obvious risks are those of data loss and data corruption, caused by hardware or software failure. These risks were not directly investigated in this designer's Ph.D. project, though it should be noted that the tape based system architecture in chapter 7 has elements that offer some degree of protection against hardware or software failure. In the tape based system the original collections can be kept on tapes that are physically write protected, and the organisation of the disk level as a cache allows for fault tolerance and graceful degradation in case of disk hardware failure.

Another risk, in a database with the size and number of users of the CMS object store, is that useful or even critical information cannot be found, even though it is stored somewhere. In other words, this is the risk of having insufficient, inexact, or improperly used meta-data for information discovery. The lack of good meta-data is often a major practical problem in the physics analysis efforts of past and current experiments. To combat the meta-data risk, CMS is aiming towards a system design that keeps all data used and produced by the collaboration in a single coherent object store, and allowing end users to access objects in terms of a uniform, physics oriented datamodel, a datamodel in which hardware details like data location are hidden. Together, these system properties should make it much easier to keep all physics results consistently annotated with meta-data. However, the success of this strategy depends for a large part on whether these properties can be implemented without adversely affecting the performance that can be obtained. If physicists using the system can get much better performance by taking data out of the common store into a private object store, or by creating private physics object types which force separate storage, they are likely to do this, resulting in 'islands of information' and a loss in the coherence of the data and meta-data.

This designer's Ph.D. project makes two important contributions towards lowering the above meta-data risks. The first contribution is in the creation, in section 6.2, of a mechanism for reading reclustered objects in an optimal way, no matter how these objects are physically divided and replicated into different collections. This mechanism makes it unnecessary for users to create private, separate object stores or private physics object types to ensure that their physics analysis software chooses the best way of reading the required objects from all available collections. The mechanism in section 6.2 assumes that all collections which are considered are on the same disk array. The mechanism has to be generalised in order to make access optimisation decisions for the complete CMS object store, which has collections on many storage subsystems with different geographical locations and access characteristics. However, making this generalisation is relatively straightforward [106]. The second contribution to meta-data risks is that *automatic* mechanisms were created, which transparently perform re-clustering operations (chapter 6) and staging and filtering operations (chapter 7), operations which would otherwise have to be performed by hand. By reducing or eliminating the need for the end users to pay attention to the physical storage model of the data, these operations allow the end users to devote more time and attention to maintaining the quality of the logical, physics oriented datamodel and its meta-data.

Most of the work in this designer's Ph.D. project is driven by the risk that the CMS physics analysis system, in production from 2005 on, will deliver insufficient I/O performance to support the physics analysis goals of the experiment. There are two sides to this: how much performance is needed, and how much can be feasibly offered. The needs are discussed in chapters 1, 2, and 3. Unfortunately, as discussed in chapter 3, there are huge error bars on the estimates of the I/O needs for physics analysis, both with respect to the absolute throughput needed and with respect to the exact parameters of the access patterns that need to be supported. Some of this uncertainty is inevitable. For example, the offline physics analysis I/O performance needed to find the Higgs boson, and determine its mass, depends for a large part on the mass of the Higgs boson. Other parts of this uncertainty could be feasibly reduced, for example by a closer study

of the end user activities in current physics analysis efforts. As discussed in chapter 3, the main thing that is holding up such studies is the lack of hard information like access pattern logs. In this designer's Ph.D. project, all information that was available was collected and studied (see chapter 3), and some efforts were made to obtain new access pattern logs. Unfortunately, these efforts have not been too successful so far. Obtaining logs from a running experiment is not just a matter of talking to the right person. Such logs, if they are made at all, are considered sensitive information, because they reveal things about the physics results being obtained by the experiment. It is not customary for experiments to release any sensitive information to outsiders, and to do it at all requires high-level political support in the experiment concerned. The importance assigned to this problem in political circles is increasing, but it is not yet so high that any detailed access patterns logs have been released to outsiders.

The uncertainty in the estimates of the I/O needs will become somewhat smaller as the CMS experiment approaches the start of data taking in 2005, but it is not expected to become very much smaller. Thus, before the start of data taking, the best strategy to meet the I/O needs for CMS is to maximise the I/O performance that can be offered in 2005 as much as possible. The need to maximise performance should be balanced however against the need to keep the complexity of the storage manager software within bounds. It is important to find the effective optimisations among the huge number of optimisations that can be thought up. If too many optimisations are implemented, the software will collapse under its own complexity. Finding effective optimisations is made harder, but not impossible, by the lack of detailed knowledge about future system and workload parameters. As seen in chapter 7, for some optimisations it can be shown that they are effective over a large parameter space. Others turn out to be only effective in some parts of the parameter space, so that their utility for the CMS physics analysis system of 2005 remains questionable. In this project, the rule of thumb was used that an optimisation should only be pursued, and recommended for implementation, if it can yield a performance improvement of at least a factor 2 over a large parameter space.

The I/O performance that can be offered depends for some part on the hardware performance in 2005, so hardware performance was studied in chapter 4. When extrapolating current trends for hard disks, and combining these with the likely computing budget, it seems likely that a disk performance of some 100 GB/s in sequential I/O can be available to CMS in 2005 (chapter 2). The error bar on this I/O performance estimate is much smaller than the error bars on the estimates of the I/O performance needs. However, as seen in chapter 4, the performance figure for sequential I/O does not tell the whole story. Other I/O patterns, like random and selective reading (sections 4.4.4 and 4.4.5), yield an I/O performance that is dramatically lower than that of sequential I/O, both on current and future disk hardware. For example, the performance difference between sequential and random reading of 1 KB objects is a factor 50 on a commodity hard disk typical for the high end of the 1994 market, and is expected to increase to a factor 200 on a disk bought in 2005 (figure 4.4). The I/O performance delivered by the disks to the CMS physics analysis system thus depends strongly on the nature of the disk access pattern produced by the system. Because of their crucial importance to I/O performance, access patterns became a major focus of this designer's Ph.D. project. The properties of physics analysis, in terms of logical access patterns on physics data,

were studied closely in chapter 3. When combining the results of chapter 3 with the performance measurements for physical access patterns in chapter 4, it was concluded that a disk reclustering optimisation is needed to maintain good I/O performance over the progress of physics analysis effort. As a result of the work in this designer's Ph.D. project, (re)clustering was identified [85] as a major area of research in the RD45 collaboration. An automatic, dynamic disk based reclustering system was developed in chapter 6. The reclustering optimisations in this system greatly reduce the risk that the effective disk I/O performance of the physics analysis system lies far below the optimum performance of sequential I/O. The reclustering optimisations work on top of the basic storage management policies developed in chapter 5. These basic policies also include the use of a read-ahead optimisation (section 5.6.1), to avoid a performance breakdown when some particular disk access patterns occur. These patterns occur, for example, when a physics analysis job reads two or more physics objects of different types per event. The need for a read-ahead optimisation on top of Objectivity/DB, in case these patterns occur, was first identified in this designer's Ph.D. project.

When reading physics objects from a set of tapes in a tape robot, a bad access pattern can easily degrade performance by a factor of 1000 below the optimum. The issue of optimising I/O performance in physics analysis efforts with data on tape was investigated in chapter 7. This work builds on existing research and experience with 'normal' tape management systems (section 7.3), and is complementary to other work on tape issues in high energy physics [61]. The main contribution made in this designer's Ph.D. project is that it explored the potential of using reclustering operations to optimise a tape based system. It turns out that cache filtering (section 7.3.5), a particular reclustering operation in the disk cache, can often greatly improve performance. However, over the range of likely system and workload parameters that was explored, the benefits of tape reclustering (section 7.3.6), reclustering data directly on tape, are low or even absent. Though the work on improving tape I/O performance in chapter 7 contributes to lowering the tape I/O risks and uncertainties, it does not eliminate them. The quality of the initial clustering on tape therefore remains a very important parameter in determining the overall performance of tape based physics analysis. Research on improving this parameter is ongoing outside this designer's Ph.D. project [61]. Apart from improvements, it is expected that this research will lower the error bars on the initial clustering quality parameter for CMS. These error bars are very high at the moment.

Data transport over the network also is a factor in determining the I/O performance of the CMS physics analysis system. Local area network (LAN) speeds are improving very quickly compared to the speed of I/O devices, so LAN performance is not expected to be a bottleneck in future physics analysis systems [54]. LAN issues were not specifically researched in this project. The picture for the wide area network (WAN) risks is more complicated. Recall that the complete CMS physics analysis system is a distributed system, with sites all over the world, interconnected by WAN links. The estimate of the WAN performance needed by the CMS system from 2005 onwards has large error bars, and so has the estimate of the WAN performance that can be economically obtained at that time. In this designer's Ph.D. project, WAN issues were never studied directly, mainly because they were considered to be out of scope in the RD45 collaboration, of which this project was a part. WAN issues are currently being ad-

dressed in the MONARC project [18], which was started in late 1998. This designer's Ph.D. project made some contributions to MONARC. Specific technical knowledge on Objectivity/DB and the layers underneath it as summarised for MONARC [107] [108], this knowledge was produced both in this designer's Ph.D. project and elsewhere in the RD45 collaboration. Also, some experience gained in simulating the tape based system of chapter 7 was passed on to the MONARC Simulation Workgroup [109].

There are significant scalability goals in CMS storage management (chapter 1), and these imply significant risk. It is not possible to demonstrate now, with tests on running prototypes, that the system design will scale to the size parameters needed in 2005. The hardware configurations that are economically available to CMS are simply not large enough to demonstrate scaling to the 2005 target configuration. Scalability risks with respect to the size, in Petabytes, of the object store were mostly investigated outside this designer's Ph.D. project. This project has made contributions with respect to the scalability in the MB/s of throughput, and the scalability in the number of concurrently active clients. In chapter 5, these scalability issues were addressed by developing storage management policies which show not merely good, but excellent scalability to the size of current very large hardware configurations, like the HP Exemplar supercomputer (see section 5.7.1). The excellent scalability of these policies gives a safety margin for the efficiency when scaling to still larger future configurations.

8.3 Software artifacts created in this project

In chapter 5, a set of basic policies for the CMS storage manager was developed, including basic reclustering policies. Some of the policies govern storage management at a very coarse-grained level, for example the placement of data on different disk farms. Other policies govern fine-grained actions like the reading of individual physics objects. Particular emphasis is placed on scalability. It is shown that the policies have excellent scalability and good efficiency on a very large hardware configuration, with many CPUs and disk arrays. By developing the basic policies to take care of scalability, chapter 5 lays the groundwork for the design of scalable, more sophisticated storage management policies in the later chapters.

A specialised testing framework called TOPS [72], the Testbed for Objectivity Performance and Scalability, was developed and implemented on top of Objectivity/DB. TOPS implements the basic storage management in chapter 5, and contains physics workload generators to simulate the layers on top of the storage manager. TOPS also provides the tools to run and evaluate large scalability tests, for hundreds of database clients, on machines like the HP Exemplar.

In chapter 6, a complete system for reclustering disk based physics data was developed. The system works automatically, based on a few tuning parameters, and transparently to the end user code. Two concrete implementations of this disk based system have been made. The first implementation was integrated with the TOPS framework. This implementation was used to validate the system design, through extensive performance measurements based on simulated physics scenarios. A second implementation [79], largely re-using the code of the first, integrated the disk based system with the LHC++ framework for physics analysis [80]. This second implementation validated that the

prototype would indeed fit as a storage manager inside a complete physics analysis system, and that it would correctly operate as part of it. The second implementation is the only 'production quality' software artifact developed in this designer's Ph.D. project. Here, 'production quality' means the requirement that the software should be useful for end users, people other than the original developer, without requiring a strong interaction with the original developer. This implies that documentation must be written, that a clean installation procedure must be developed, that the system must be robust in the face of software and hardware failure, and that it must generate error messages intelligible to end users. Brooks [110] gives as a general rule that, compared to an implementation only to be used by the original developer, creating such a 'production' version takes at least three times as much effort. This rule turned out to be true for the two implementations of the disk based system. The second implementation was developed up to a 'production quality' level because, at that time (winter 1997–1998), a group inside the CMS collaboration, was planning to perform a large-scale integration effort leading to a fully integrated prototype CMS physics analysis system. A storage manager with automatic reclustering facilities was one of the requirements for this system. This integration effort, to be completed in the summer of 1998, was planned to satisfy a major milestone in the CMS software effort. However, the integration effort was first postponed, and then scrapped in the late Autumn of 1998, because of a lack of manpower. The interpretation of the corresponding CMS milestone was scaled down to account for this.

In chapter 7, a complete system with reclustering optimisations for tape based physics analysis was developed and validated. This system is also useful for some non-physics data analysis workloads. The system was not fully implemented, only an event driven simulator for the system was implemented. This simulator did however contain full implementations of the schedulers that would also run in a fully implemented tape based system. The event driven simulator is the main component in a simulation framework (section 7.4.3) that was also purpose-built in this designer's Ph.D. project. This framework supports massive simulation experiments covering a large parameter space. It is interesting to note that some parts of the TOPS framework were re-used in creating this simulation framework. The re-used parts of TOPS are the tools and procedures to create the individual configuration parameters for hundreds of Objectivity/DB clients running in parallel. The simulation framework uses these tools and procedures to create the individual configuration parameters for the hundreds of event driven simulations making up a single simulation experiment. As an added bonus, the TOPS framework tools made it trivial to run many event-driven simulations in parallel.

The uncertainties related to CMS storage management are not all negative. This project developed optimisations that focus on a conservative scenario for future developments in hardware. However, as discussed in section 4.6, there is some chance that, before 2005, a radically improved storage technology will replace tapes or hard disks. In this more optimistic scenario, CMS storage management would be a lot easier, though at the same time some of the mechanisms developed in this project could become obsolete. However, this more optimistic scenario does not invalidate the reason why these mechanisms were developed, which is to lower the current overall risk over all possible scenarios. Section 4.6 also mentions a very pessimistic scenario, one in which the market for 'industrial quality' tape drives disappears, leaving no replacement tech-

nology for high-volume physics data storage at equal cost. The work in this project does not directly deal with this pessimistic scenario. Most likely, in this scenario, the CMS collaboration will have to spend much more funds than currently foreseen to buy additional disk storage space that makes up for the missing tape space.

8.4 Spin-offs

The work in this designer's Ph.D. project produced a few spin-off results, which do not directly fit the scope of the initial project definition. Some contributions to the MONARC project [109] [107] [108] were already mentioned above. Also mentioned was the generalisation of the methods developed in section 6.2 to distributed systems with replication [106]. At the start of the project, some Objectivity/DB related software integration issues were studied [21] [111].

The most interesting spin-off is the work done on achieving a particular database milestone of the CMS software/computing project [112] [74]. This milestone was called 'ODBMS filling with raw data at 100 MB/s': to fill an object database with raw CMS event data (see section 2.4), with a speed of at least 100 MB/s, the raw data production rate of the CMS detector. The milestone thus aims to demonstrate the ability to store all experimental data produced when the experiment is running. The work on this milestone was done by the author in collaboration with other participants in the GIOD project [24]. The milestone test was run on the HP Exemplar at Caltech, the machine that was used earlier in the validation studies of section 5.7. The Exemplar was chosen because it was the only configuration available that has both sufficient hard disks to write to disk at 100 MB/s, and sufficient CPU power to support the object creation process in Objectivity at 100 MB of objects per second [107]. The raw CMS event data used were produced earlier in the GIOD project, using an object model that was also defined earlier in GIOD. The contributions from the side of this designer's Ph.D. project were the TOPS framework, used for setting up the milestone test run and evaluating its result, some of the basic storage management policies in chapter 5, knowledge about the locking protocol of Objectivity/DB, and analysis of system requirements, and experience in configuring and running such scalability tests. The test setup was similar to that of the test described in section 5.7.5. The differences were that 'real' raw data objects were written (after being pre-loaded into memory), that less computation was specified per event, and that all available disk arrays (containing 49 disks) on the HP Exemplar were used, so as to maximise the I/O performance obtained. See [74] for a complete description of the test setup. In the end, a maximum sustained I/O performance of 172 MB/s into Objectivity/DB database files on disk was obtained, well above the 100 MB/s set by the milestone.

Some new technical knowledge was gained from the 100 MB/s milestone test, but not very much compared to the knowledge gained in the earlier scaling tests discussed in section 5.7. However, the milestone test was not designed to deliver technical results, but for its publicity value, to deliver a result that would allow project managers in CMS and RD45 to demonstrate that appropriate technology choices were being made and that the technical work in their projects was in good shape. The publicity value of the milestone test turned out to be very high. The test result was used in particular

to show that the choice for Objectivity/DB was appropriate, that Objectivity/DB could handle the CMS raw data-rate. The 172 MB/s figure has been quoted by the project management of the CMS software/computing project in high-level external reviews of this project. It has also been quoted by the RD45 collaboration in a high-level external review of RD45. In addition, the 172 MB/s figure found its way into the background sections of at least two funding proposals for CMS related computing research projects in the USA.

8.5 Evaluation of the design process

Project results versus the project proposal

When comparing the project results to the original project proposal [113], one should note that the project proposal did not specify a fixed set of quantifiable targets. The need for a cyclic prototyping approach was already well established when the proposal was written, so instead of fixed goals the proposal specifies that cyclic prototyping has to be used to address some or all of the stated problems in CMS storage management. This goal set by the original project proposal has been satisfied: the project has produced a significant reduction in the risks for CMS storage management, and a significant increase in knowledge.

The project proposal also specified that the end result of the project should be a 'fully functional prototype', to be used in validation studies and in the 'OO pilot projects' involving end users that were planned by CMS. These two project goals were dropped in the late Autumn of 1998, when the CMS software/computing project dropped its goal to perform any such OO pilot projects in the timeframe up to 2000 (see also section 8.3). The dropping of these goals in this designer's Ph.D. project is visible in this thesis as a discontinuity between chapters 6 and 7. Most of the results up to and including chapter 6 were created before these two goals were dropped. Most of the results in chapter 7 were produced after. Up to chapter 6, all results are validated with running prototypes, and in chapter 6 a 'production quality' version of the disk based system is produced, a version that is suitable for OO pilot projects. In chapter 7, a fully functional, running prototype is no longer created, instead the design is validated using simulation.

Software engineering approach

The risk-driven, cyclic prototyping approach [9], taken as the software engineering methodology for the project, turned out to work well. The cyclic approach implies that the goals of a cycle are chosen at the start of the cycle, not at the start of the project. This flexibility turned out to be crucial in keeping the project well-integrated with the ongoing efforts of other participants in the CMS software/computing project and the RD45 collaboration, of which this designer's Ph.D. project was part.

Inside the cycles themselves, an incremental development style, similar to the spiral model when using very small cycles [9], was used. One exception to this was the use of a 'big bang' approach in the detailed design phase of the tape based system, the

work that corresponds to the second half of chapter 7. See section 7.4 for a complete discussion and evaluation of this 'big bang' approach. The 'big bang' approach is a very risky design methodology, relying on the success of a single 'big bang' integration step. The approach was only used because no alternative, less risky design approach could be found. Even then, considerable effort was spent in developing techniques to reduce the associated risks as much as possible. In the end, taking the risk in this approach paid off: the 'big bang' integration step turned out to be successful.

The cyclic approach followed in this designer's Ph.D. project made it possible to deal with unexpected research results. At two occasions during the project, the tentative plans for subsequent activities were changed radically because of such unexpected research results. The fact that this happened shows that the choice for a flexible, cyclic approach was correct. If a more rigid methodology had been chosen, the cost of changing the plans would have been much higher.

The first occasion at which plans were radically changed was soon after the start of the project. At that time, the assumption in the RD45 collaboration was that selective reading of physics objects (see section 4.4.5), a technique that was not used in the then-popular existing physics analysis systems, would be the key to improving performance over these existing systems. The major open issue with selective reading was believed to be whether Objectivity/DB allowed the building of efficient indexing mechanisms, that would give low indexing overheads in locating the selected objects even for highly selective reading. The selective reading tests in section 4.4.5 were performed with the intention to prove that selective reading indeed was the key to improved performance. It was expected that these tests would show that, if only half of the objects are read, a physics analysis job will run twice as fast. In fact, the tests of section 4.4.5 showed something completely different. They showed that the job will generally not run any faster at all. Furthermore they showed that this lack of a performance improvement is not caused by too-high indexing overheads in Objectivity/DB itself. As explained in section 4.4.5, the lack of an improvement is caused by the way in which hard disks work. Because of this unexpected result, the then-existing plan for producing a prototype was dropped. The original plan was to create a prototype that would centre around the use of a selective reading mechanism to optimise performance. Instead of following the original plan, reclustering was identified as a major focus for the designer's Ph.D. project.

The second occasion at which plans were radically changed was during the studies for scaling disk reclustering operations beyond four independent access patterns by trading away disk space (section 7.2). When these studies were started, it was believed that good results could be obtained by looking for ways to minimise the amount of space that would have to be traded away in order to maintain a good I/O performance. The plan was to spend the remainder of the research time in the project on developing smart heuristics, which would minimise the space that is traded away while maintaining good I/O performance. During the work on developing such good heuristics, it became more and more clear that even very good heuristics would have little chance of improving any performance measure by a factor 2 or more. Recall that such a factor 2 is the criterion that was adopted in this project for deciding whether an optimisation should be pursued (see section 8.2.2). As a result of this observation, the activity on

developing these heuristics was stopped. It was then chosen instead (section 7.2.3) to spend the remainder of the research time on a system that would scale beyond four independent access patterns without this system incorporating any new smart heuristics to reduce the amount of space traded away.

8.6 Conclusions

This thesis reported on a designer's Ph.D. project called *prototyping of CMS storage management*. The project has been performed at CERN, the European laboratory for particle physics, in collaboration with the Eindhoven University of Technology. The work was done in the context of a larger R&D effort aimed at developing the physics analysis systems for CERN's next generation high energy physics experiments. Research on CMS storage management is needed to deal with the extraordinary performance and scalability requirements for the CMS physics analysis system, and to ensure that the atypical properties of the physics analysis process are accounted for. This designer's Ph.D. project used a risk-driven, cyclic prototyping approach to reduce the risks associated with CMS storage management. The work centred around risks related to I/O performance. The main contributions of this project are in the development and validation of scalable clustering policies, and of scalable reclustering optimisations for disk based and tape based physics analysis.

Summary

This thesis reports on a designer's Ph.D. project called *prototyping of CMS storage management*. The project has been performed at CERN, the European laboratory for particle physics, in collaboration with the Eindhoven University of Technology. CMS, the Compact Muon Solenoid, is a next-generation high energy physics experiment at CERN, which will start running in 2005. The data storage requirements for CMS are exceptional in their scale. In one year of running, one Petabyte of physics data is produced, that is 10^{15} bytes or 1000 Terabytes. Interactive physics data analysis is expected to require at least a few GB/s of I/O capacity to support all concurrent users. Research on CMS storage management is needed to deal with this exceptional scale, and to ensure that the atypical properties of the physics analysis workload are accounted for.

The work in this designer's Ph.D. project was done in the context of a larger R&D effort aimed at developing the physics analysis systems for CERN's next generation high energy physics experiments. The project used a risk-driven, cyclic prototyping approach to reduce the risks associated with CMS storage management. The work focused on risks related to I/O performance.

The main results of the project are as follows. A rigorous study has been made of the I/O performance issues surrounding physics analysis software, with a particular emphasis on the later stages of physics analysis. As a result of this study, *clustering* and *reclustering* were chosen as the main focus of the project. The *clustering* arrangement of data is the order in which data are placed on physical storage media like disks and tapes. A *reclustering* operation is one that changes the clustering arrangement. Reclustering is crucial if good disk performance is to be maintained under changing physics analysis workloads. A set of basic physics-oriented storage management policies has been developed. The scalability of these policies was validated on a large hardware configuration, with up to 240 concurrent worker processes and up to 145 MB/s of throughput. On top of these basic policies, scalable, more sophisticated storage management policies like 'disk reclustering' and 'cache filtering' were designed. A running prototype of a disk based storage management system, which includes reclustering optimisations, was created and validated. A full-scale storage management system to optimise disk and tape access for CMS was designed, and the performance of this system was analysed using simulation over a range of physics workloads.

Through the use of risk-driven, cyclic prototyping, this project has produced a significant reduction in the risks for CMS storage management, and a significant increase in the knowledge base available to the CMS collaboration in the further development of its physics analysis system.

Samenvatting

Dit proefschrift documenteert een proefontwerp project genaamd *prototyping of CMS storage management*, het prototypen van databeheer in CMS. Dit proefontwerp werd uitgevoerd op CERN, het Europese laboratorium voor elementaire deeltjes fysica, in samenwerking met de Technische Universiteit Eindhoven. CMS, de Compact Muon Solenoid, is een hoge-energie fysica experiment van de volgende generatie op CERN, dat opstart in 2005. In een looptijd van een jaar produceert CMS een Petabyte aan gegevens, dat is 10^{15} byte of 1000 Terabyte. Naar verwachting vereist de interactieve data-analyse ten minste een paar GB/s I/O capaciteit. Onderzoek naar databeheer in CMS is nodig om deze grootte-ordes te bereiken, en om er zeker van te zijn dat er rekening gehouden wordt met de atypische eigenschappen van het data-analyse proces.

Dit proefontwerp project werd uitgevoerd als deel van een groter R&D project gericht op databeheer voor volgende generatie experimenten op CERN. In het proefontwerp project zelf werd er gebruik gemaakt van een risico-gedreven, cyclisch prototyping proces, om de risico's voor databeheer in CMS te verkleinen. Het werk richtte zich voornamelijk op risico's gerelateerd aan I/O performance.

De belangrijkste resultaten van het proefontwerp project zijn als volgt. Het I/O performance probleem voor data-analyse in CMS is diepgaand bestudeerd, met bijzondere aandacht voor de latere fasen in de data-analyse. Als gevolg van die studie werden *clustering* en *reclustering* gekozen als zwaartepunt van het project. De *clustering* van gegevens op opslagmedia als hard disks en tapes is de rangschikking waarin die gegevens stoffelijk opgeslagen zijn. Een *reclustering* operatie verandert de clustering. Reclustering is essentieel in CMS voor het behouden van een goede I/O performance terwijl de toegangspatronen van het data-analyse proces veranderen. Een stelsel van basisregels voor databeheer in CMS is ontworpen. De schaalbaarheid van deze basisregels is aangetoond op een groot computersysteem, met tot 240 tegelijk werkende processen, en tot 145 MB/s aan throughput. Uitgaande van deze basisregels zijn schaalbare, en meer geavanceerde methoden voor databeheer zoals 'disk reclustering' en 'cache filtering' ontwikkeld. Een werkend prototype van een databeheersysteem met reclustering voor hard disks is gebouwd en op werking getest. Een volledig databeheersysteem, voor het optimaliseren van disk en tape toegang in CMS, is ontworpen, en door middel van simulatie is de performance van dit systeem onderzocht over een reeks van parameters.

Via een risico-gedreven, cyclisch prototyping proces is er in dit project een aanzienlijke verkleining van de risico's voor databeheer in CMS bereikt. Ook is de kennis die beschikbaar is voor het verder ontwikkelen van het CMS data-analyse systeem aanzienlijk vergroot.

Acknowledgements

I thank my daily supervisors, Peter van der Stok and Ian Willers, for their enthusiastic support, and for having created this project in the first place. I thank Peter in particular for his insights and support during the writing of this thesis, and his willingness to proof-read chapters whenever I could finish them. Ian was always there to share his insights concerning the organisational and political angles of my work at CERN. I want to thank Marloes van Lierop for her support from the side of the Stan Ackermans Institute, and Martti Pimiä and Werner Jank for their managerial support from the side of the CMS software/computing project, and for making the CERN EP/CMC group such a nice place in which to work. I thank my Ph.D. supervisor Paul De Bra for his insights and his enthusiastic support of this project. Thanks also to Richard McClatchey for his thoughtful comments on different versions of this thesis.

The scalability tests on the HP Exemplar supercomputer, documented in chapter 5 and section 8.4, were done in collaboration with Harvey Newman, Julian Bunn, and Rick Wilkinson at Caltech, whom I thank for their insights and enthusiasm, and for making me feel welcome whenever I visited them. Thanks go also to the CACR system management at Caltech for their cheerful and comprehensive support during these scalability tests.

I want to thank all my colleagues and friends at CERN, in the CMS software/computing project, and in the RD45 collaboration for their support, for having done the work that I could build on, and for many pleasant coffee breaks. This work profited from numerous discussions I had with my fellow Ph.D. students at CERN, in CMS and RD45: Erco Argante, Johannes Gutleber, Martin Schaller, Wolfgang Hoschek, Heinz Stockinger, Kurt Stockinger, and Frank van Lingen. My predecessor Erco Argante deserves special recognition for his help in getting me started working at CERN and living in the Geneva region.

This project was jointly funded by the Technical and Doctoral Student Programme at CERN, and by the Stan Ackermans Institute at the Eindhoven University of Technology.

Curriculum Vitæ

Koen Holtman

25 januari 1971	Geboren in Tegelen
1983 – 1989	Gymnasium-β Bisschoppelijk College Broekhin in Roermond
1989 – 1995	Technische Informatica Technische Universiteit Eindhoven
1995 – 1997	Ontwerpersopleiding Technische Informatica Stan Ackermans Instituut, Technische Universiteit Eindhoven
1997 – 2000	Promotie op Proefontwerp (als uitbreiding van eindproject in de ontwerpersopleiding) Stan Ackermans Instituut, Technische Universiteit Eindhoven uitgevoerd bij het CERN in Geneve

References

[1] Bestuurscommissie Ontwerpers- en korte Onderzoekersopleidingen (BCO). Op weg naar promotie op proefontwerp, Handreiking van de BCO aan het College van Decanen. Eindhoven University of Technology, February 1994.

[2] Winter corporation 1998 VLDB Survey Program. See http://www.wintercorp.com

[3] C. L. Giles. Searching the web: can you find what you want? Keynote address at the Eighth International Conference on Information and Knowledge Management (ACM CIKM ’99), Kansas City, USA, November 2-6, 1999.

[4] Talk by Filetek, Inc. on ’Atomic Data Store’ at the RD45 Workshop, July 12-15 1999. See http://wwwinfo.cern.ch/asd/rd45/workshops/july99/index.html

[5] J. Scoggins. A Practitioners’s View of Techniques Used in Data Warehousing for Sifting Through Data to Provide Information. Keynote address at the Eighth International Conference on Information and Knowledge Management (ACM CIKM ’99), Kansas City, USA, November 2-6, 1999.

[6] A. Hanushevsky. Developing Scalable High Performance Terabyte Distributed Databases. Proceedings of CHEP’98, Chicago, USA.

[7] L. Kerschberg, H. Gomaa, D. Menascé, J. Yoon. EOSDIS Data and Information Architecture. Proceedings of the 1996 International Conference on Statistical and Scientific Databases, Stockholm, Sweden.

[8] M. Csenger. Two Terabytes Per Day. Network World, 27 July 1998. Also available at: http://www.nwfusion.com/news/0727feat.html

[9] B. Boehm. A Spiral Model of Software Development and Enhancement. ACM Sigsoft Software Engineering Notes, Vol 11 Nr. 4 p. 22–42, 1986.

[10] CMS Computing Technical Proposal. CERN/LHCC 96-45, CMS collaboration, 19 December 1996.

[11] The LEP accelerator at CERN. General information: http://www.cern.ch/CERN/Divisions/SL/welcome.html

[12] The LHC Conceptual Design Report. CERN/AC/95-05(LHC). General information: http://wwwlhc01.cern.ch/

[13] ATLAS Technical Proposal. CERN/LHCC/94-43 LHCC/P2. General information: http://atlasinfo.cern.ch/Atlas/Welcome.html

[14] CMS Technical Proposal. CERN/LHCC 94-38 LHCC/P1. General information: http://cmsinfo.cern.ch/cmsinfo/Welcome.html

[15] ALICE - Technical Proposal for A Large Ion Collider Experiment at the CERN LHC. CERN/LHCC/95-71, December 1995. General information: http://www1.cern.ch/ALICE/welcome.html

[16] LHCb technical proposal, A Large Hadron Collider Beauty Experiment for Precision Measurements of CP Violation and Rare Decays. CERN LHCC 98-4, LHCC/P4, 20 February 1998.

[17] Using an object database and mass storage system for physics analysis. CERN/LHCC 97-9, The RD45 collaboration, 15 April 1997.

[18] The MONARC Project, Models of Networked Analysis at Regional Centres for LHC Experiments. Project homepage: http://www.cern.ch/MONARC/

[19] L. Robertson. Mock-up of an Offline Computing Facility for an LHC Experiment at CERN. Presentation in the MONARC meeting on July 26, 1999. http://www.cern.ch/MONARC/plenary/1999-07-26/robertson/index.htm

[20] Objectivity/DB. General information: http://www.objy.com/

[21] K. Holtman. Prototyping of CMS Storage Management. CMS NOTE/1997 - 074, CERN.

[22] The CMS software/computing project. More information: http://cmsdoc.cern.ch/sw.html

[23] RD45, A Persistent Storage Manager for HEP. Project homepage: http://wwwcn.cern.ch/asd/cernlib/rd45/

[24] The GIOD project, Globally interconnected object databases. Project homepage: http://pcbunn.cithep.caltech.edu/

[25] D. Baden et al. Joint DØ/CDF/CD Run II Data Management Needs Assessment. CDF/DOC/COMP_UPG/PUBLIC/4100, DØ Note 3197, March 20, 1997.

[26] R. Jacobsen. From Raw Data to Physics Results. CERN summer student lecture. See also http: //webcast.cern.ch/Projects/WebLectureArchive/jacobsena/sld001.htm

[27] J. Becla. Data clustering and placement for the BaBar database. Proceedings of CHEP'98, Chicago, USA.

[28] R. Brun et al. PAW Physics Analysis Workstation. CERN/CN Long Write-Up Q121, 1989. See also http://wwwinfo.cern.ch/asd/paw/

[29] R. Brun, F. Rademakers. ROOT - An Object Oriented Data Analysis Framework. Proceedings of AIHENP 1996, Laussanne. General information: http://root.cern.ch/

[30] W. Hoschek. Partial range searching in OLAP data warehouses. Submitted for publication. See also
http://www.cern.ch/CERN/Divisions/EP/HL/Papers.html

[31] P. Boncz, S. Manegold, M. Kersten. Database Architecture Optimized for the New Bottleneck: Memory Access. Proceedings of the International Conference on Very Large Data Bases (VLDB), p. 54–65, Edinburgh, United Kingdom, September 1999.

[32] J. Bunn. Personal communication.

[33] J. Korst, V. Pronk, P. Coumans. Disk scheduling for variable-rate data streams. Proceedings of the European Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services, IDMS'97, Darmstadt, Germany, September 10–12 1997.

[34] J. Korst. Random Duplicated Assignment: An Alternative to Striping in Video Servers. Proceedings of ACM Multimedia 97, November 8-14, 1997, Seattle, USA.

[35] M. Stonebraker. Sequoia 2000– A Reflection on the First Three Years. Sequoia 2000 Technical Report 94/58, University of California, Berkeley, September 1994.

[36] M. Stonebraker, J. Frew, K. Gardels, J. Meredith. The Sequoia 2000 Storage Benchmark. Sequoia 2000 Technical Report 92/12, University of California, Berkeley, CA, June 1992.

[37] S. Sarawagi. Query Processing in Tertiary Memory Databases. Proceedings of 21st VLDB Conference, Zurich, Switzerland, 1995, p. 585–596.

[38] S. Sarawagi, M. Stonebraker. Efficient organization of large multidimensional arrays. Proceedings of 10th Int. Conf. on Data Engineering, p. 328–336 (1994).

[39] S. Berchtold, H. Kriegel, C. Böhm. The Pyramid-Tree: Breaking the Curse of Dimensionality. Proc. of the ACM SIGMOD Int. Conf. on Management of Data, 1998. Available at http://www.research.att.com/~berchtol

[40] S. Berchtold, D. Keim. High-Dimensional Index Structures - Database Support for Next Decade's Applications. Tutorial at the ACM SIGMOD Int. Conf. on Management of Data, 1998. Available at
http://www.research.att.com/~berchtol

[41] V. Gaede, O. Guenther. Multidimensional Access Methods. Technical report, International Computer Science Institute TR-96-043, October 1996. Available at http://www.wiwi.hu-berlin.de/~gaede/vg.pub.html

[42] H. Ferhatosmanoglu, D. Agrawal and A. Abbadi. Clustering Declustered Data for Efficient Retrieval. Proceedings of the Eighth International Conference on Information and Knowledge Management (ACM CIKM '99), Kansas City, USA, November 2-6, 1999.

[43] G. Wiederhold. Database design, 2nd ed. London, McGraw-Hill, 1983. ISBN 0-07-070132-6.

[44] J. Hennessy, D. Patterson. Computer organization and design: the hardware/software interface. San Mateo, Morgan Kaufmann, 1994. ISBN 1-55860-281-X.

[45] D. Patterson, J. Hennessy. Computer architecture: a quantitative approach, 2nd ed. San Mateo, Morgan Kaufmann, 1995–1996. ISBN 1-55860-329-8.

[46] R. Cattell. Object Database Management: Object-Oriented and Extended Relational Database Systems. Revised ed. Addison-Wesley 1994. ISBN 0-201-54748-1.

[47] C. Ruemmler, J. Wilkes. An introduction to disk drive modeling. IEEE Computer 27(3):17-29, March 1994.

[48] The Versant object database. Vendor homepage: http://www.versant.com/

[49] R. Cattell, D. Barry (eds), The Object Database Standard: ODMG 2.0. Morgan Kaufmann Publishers. May, 1997. General information: http://www.odmg.org/

[50] Quantum Corporation. Storage Industry History And Trends. Web site: http://www.quantum.com/src/history/

[51] LHC Computing Technology Tracking Teams. General information: http://wwwinfo.cern.ch/di/ttt.html

[52] Pasta - The LHC Technology Tracking Team for Processors, Memory, Architectures, Storage and Tapes. Status Report, CERN, August 1996.

[53] J. Gray, G. Graefe. The Five-Minute Rule Ten Years Later, and Other Computer Storage Rules of Thumb. SIGMOD Record 26(4): 63–68 (1997).

[54] Personal communication with technology trackers of the CERN IT division.

[55] Some tape aspects of PASTA, revisited. Internal CERN technology tracking report, 1999.

[56] StorageTek (Storage Technology Corporation). Vendor homepage: http://www.storagetek.com/

[57] C. Farel. The State-of-the-Art in Magnetic Tape Data Recording. CERN Computing colloquium, 21 October 1999.

[58] HPSS – The High Performance Storage System. Project homepage: http://www.sdsc.edu/hpss/

[59] F. Rademakers. The Parallel Interactive Analysis Facility. CERN Report, CERN/CN/94/9 (1994).

[60] M. Ogg, F. Handfield, A. Ricciardi. Nile: Large-Scale Distributed Processing and Job Control, Proceedings of CHEP'98, Chicago, USA.

[61] Grand Challenge Application on HENP Data. Project Homepage: http://www-rnc.lbl.gov/GC/

[62] P. Scheuermann, G. Weikum, P. Zabback. 'Disk Cooling' in Parallel Disk Systems. Data Engineering Bulletin 17(3): 29–40 (1994).

[63] T. Nemoto, Y. Sato, K. Mogi, K. Ayukawa, M. Kitsuregawa, M. Takagi. Performance evaluation of cassette migration mechanism for scalable tape archiver. Digital Image Storage and Archiving Systems, Philadelphia, PA, USA, 25-26 Oct. 1995, p.48–58.

[64] E. Arderiu. Calibration DB. Talk at the RD45 workshop, October 27-29 1998. http://wwwinfo.cern.ch/asd/rd45/workshops/oct98/presentations/eva/calibration/index.htm

[65] M. Pollack. Database Solutions for the PHENIX experiment. Proceedings of CHEP'98, Chicago, USA.

[66] The CDF Run II Calibration Database. http://www-cdf.fnal.gov/upgrades/computing/calibration_db/orac_calibdb.html

[67] HEPODBMS reference manual. http://wwwinfo.cern.ch/asd/lhc++/HepODBMS/reference-manual/index.html

[68] National Partnership for Advanced Computational Infrastructure. Homepage: http://www.npaci.edu/

[69] R. Bordawekar. Quantitative Characterization and Analysis of the I/O behavior of a Commercial Distributed-shared-memory Machine. Proceedings of the Seventh Workshop on Scalable Shared Memory Multiprocessors, June 1998. Also CACR Technical Report 157, March 1998, available from http://www.cacr.caltech.edu/~rajesh/

[70] S. Saunders. Data Communications Gigabit Ethernet Handbook. 1998, The McGraw-Hill Companies, Inc, New York, USA, ISBN 0-07-057971-7

[71] Myrinet network products. Vendor homepage: http://www.myri.com/

[72] TOPS, Testbed for Objectivity Performance and Scalability, V1.1. Available from http://home.cern.ch/~kholtman/

[73] K. Holtman, J. Bunn. Scalability to Hundreds of Clients in HEP Object Databases. Proceedings of CHEP'98, Chicago, USA.

[74] J. Bunn, K. Holtman, H. Newman. Object Database Scalability for Scientific Workloads. Technical report. Available from http://home.cern.ch/~kholtman/

[75] Status report of the RD45 project. CERN/LHCC 99-28, LCB Status Report/RD45, The RD45 collaboration, 21 September, 1999.

[76] R. Schaffer. Summary of 1 TB Milestone. Talk at the ATLAS database meeting.
http://home.cern.ch/s/schaffer/www/slides/db-meeting-170399-new/

[77] V. Chvatal. A greedy heuristic for the set-covering problem. Math. of Oper. Res., 4:233–235, 1979.

[78] T. Grossman, A. Wool. Computational experience with approximation algorithms for the set covering problem. Euro. J. Operational Research, 101(1):81-92, August 1997.

[79] Reclustering Object Store Library for LHC++, V2.1. Available from http://home.cern.ch/~kholtman/

[80] Libraries for HEP Computing – LHC++. Project homepage: http://wwwcn1.cern.ch/asd/lhc++/index.html

[81] W. McIver Jr., R. King. Self-Adaptive, On-Line Reclustering of Complex Object Data. Proceedings of the SIGMOD Conference 1994: 407-418.

[82] J. Cheng, A. Hurson. Effective Clustering of Complex Objects in Object-Oriented Databases. Proceedings of the SIGMOD Conference 1991: 22-31.

[83] S. Ghosh. File Organisation: The Consecutive Retrieval Property. CACM vol. 15, no. 9, p. 802–808, September 1972.

[84] A. Shoshani, L. Bernardo, H. Nordberg, D. Rotem, A. Sim. Multidimensional Indexing and Query Coordination for Tertiary Storage Management. Proceedings of SSDBM'99, Cleveland, Ohio, July 28-30, 1999.

[85] Status report of the RD45 project. CERN/LHCC 98-11, LCB Status Report/RD45, The RD45 collaboration, 6 April, 1998.

[86] M. Schaller. Reclustering of High Energy Physics Data. Proceedings of SSDBM'99, Cleveland, Ohio, July 28-30, 1999.

[87] M. Schaller. Persistent ATLAS Data Structures and Reclustering of Event Data. Ph.D. thesis, Geneva, September 1999.

[88] N. Roussopoulos. View Indexing in Relational Databases. ACM Transactions on Database Systems, Vol 7, No 2, June 1982, p. 285–290.

[89] Y. Kotidis, N. Roussopoulos. DynaMat: A Dynamic View Management System for Data Warehouses. Proceedings of ACM SIGMOD, May 31 - June 3, 1999, Philadephia, USA.

[90] H. Uchiyama, K. Runapongsa, T. Teorey. Progressive View Materialization Algorithm. Proceedings of DOLAP'99, Kansas City, USA, November 6, 1999.

[91] G. Chan, Q. Li, L. Feng. Design and Selection of Materialized Views in a Data Warehousing Environment: A case study. Proceedings of DOLAP'99, Kansas City, USA, November 6, 1999.

[92] Y. Zhuge, H. Garcia-Molina, J. Hammer, J. Widom. View Maintenance in a Warehousing Environment. Proceedings of the ACM SIGMOD International Conference on Management of Data, p. 316–327, San Jose, CA, May 1995.

[93] D. Quass, J. Widom. On-Line Warehouse View Maintenance. Proceedings of the ACM SIGMOD International Conference on Management of Data, p. 393–404, Tucson, Arizona, May 1997.

[94] P. Scheuermann, J. Shim, R. Vingralek. WATCHMAN: A Data Warehouse Intelligent Cache Manager. In Proceedings of the 22th VLDB Conference, p. 51–62, Bombay, India, September 1996.

[95] L. Chen, R. Drach, M. Keating, S. Louis, D. Rotem, A. Shoshani. Efficient Organisation and Access of Multi-Dimensional Datasets on Tertiary Storage Systems. Information-Systems, vol.20, no.2, p.155–183, Apr. 1995.

[96] R. Grossman, D. Hanley, X. Qin. Caching and Migration for Multilevel Persistent Object Stores. Proceedings of 14th IEEE Symposium on Mass Storage Systems 127-135 (1995).

[97] L. Lueking. Managing and Serving a Multiterabyte Data Set at the Fermilab DØ Experiment. Proceedings of 14th IEEE Symposium on Mass Storage Systems (1995).

[98] L. Bernardo, H. Nordberg, D. Rotem, A. Shoshani. Determining the Optimal File Size on Tertiary Storage Systems Based on the Distribution of Query Sizes. Proceedings 10th Int. Conf. on Scientific and Statistical Database Management, Capri, Italy, 1998.

[99] J. Yu, D. DeWitt. Query pre-execution and batching in Paradise: A two-pronged approach to the efficient processing of queries in tape-resident data sets. Proceedings of 9th Int. Conf. on Scientific and Statistical Database Management, Olympia, Washington (1997).

[100] K. Holtman, P. van der Stok, I. Willers. A Cache Filtering Optimisation for Queries to Massive Datasets on Tertiary Storage. Proceedings of DOLAP'99, Kansas City, USA, November 6, 1999, p. 94–100, ACM press.

[101] K. Holtman, P. van der Stok, I. Willers. Towards Mass Storage Systems with Object Granularity. Proceedings of the Eighth NASA Goddard Conference on Mass Storage Systems and Technologies, Maryland, USA, March 27-30, 2000. (To be published)

[102] StorHouse, the Atomic Data Store. Vendor homepage: http://www.filetek.com/

[103] G. Booch. Object Solutions: managing the object-oriented project. Addison-Wesley, 1995.

[104] E. Argante. CoCa: a model for parallelization of high energy physics software. Ph.D. thesis, Eindhoven University of Technology, 1998.

[105] The RD45 collaboration. Risk Analysis Update: A Discussion of the Action Items Proposed in CERN/RD45/98/08. Internal CERN risk analysis report, 1999.

[106] K. Holtman, H. Stockinger. Building a Large Location Table to Find Replicas of Physics Objects. Proceedings of CHEP 2000, Padova, Italy (to be published).

[107] K. Holtman. CPU requirements for 100 MB/s writing with Objectivity. MONARC note 98/2, November 26, 1998.

[108] K. Holtman. Notes on the Objectivity/DB AMS protocol. MONARC note 98/5, June 24, 1999.

[109] K. Holtman. Simulation of physics analysis workloads on a non-distributed system. Talk at the MONARC Simulation Workgroup meeting, November 16,1998.

[110] F. Brooks. The mythical man-month: essays on software engineering. 1995 edition, Addisson Wesley Longman, Inc, 1995.

[111] K. Holtman. Using IRIS Explorer as an offline software component. Talk at the CMS software/computing meeting, May 1997.

[112] K. Holtman, J. Bunn. 100 MB/s milestone. Talk at the CMS software/computing meeting, June 1999.

[113] K. Holtman. The CMS Persistent Storage Manager, Project Proposal for a Designer's Ph.D. CERN, ECP division, August 21, 1997.

Stellingen behorende bij het proefontwerp

Prototyping of CMS Storage Management

door Koen Holtman.

1. Het ontbreken van gedetailleerde kennis over de toekomstige access patterns van het data-analyse systeem voor CMS maakt het ontwerpen van optimalisaties ervoor moeilijker, maar niet onmogelijk.
   Zie secties 3.1 en 8.2.2 van dit proefontwerp.

2. Het reclusteren van data op disk is essentieel voor een goede performance van het data-analyse systeem voor CMS.
   Zie sectie 8.2.2 van dit proefontwerp.

3. Als het om I/O performance gaat, is een goed begrip van de access patterns van de toepassing en van de hardware performance karakteristieken veel belangrijker dan de keuze voor een specifieke middleware laag.
   Zie hoofdstuk 4 van dit proefontwerp.

4. Bij het optimaliseren van I/O op moderne harddisks zijn sommige traditionele disk parameters, zoals de sector grootte, het aantal sectoren per track, en de sector interleaving, volstrekt onbelangrijk.
   Zie sectie 4.4.1 van dit proefontwerp.

5. Bij het doen van toekomstvoorspellingen worden de korte termijn effecten vaak overschat, en de lange termijn effecten onderschat. In 2005 zullen harddisks nog steeds een centrale rol spelen in computersystemen voor de analyse van datasets met een grootteorde van Petabytes. In 2015 niet meer.
   Zie sectie 4.4.7 van dit proefontwerp.

6. HTTP/1.1 is geen mooi protocol, maar het is goed genoeg.
   Zie ook: Jeffrey Mogul, What's Wrong with HTTP And Why It Doesn't Matter.

7. De naam van de `If-Modified-Since` header in het HTTP protocol is misleidend.

8. In een groot software project is het bedrijven van politiek een belangrijk deel van het takenpakket van de software ontwerper.

9. Bij het werken met informele specificaties kan een achtergrond in formele methoden voor programmaconstructie heel waardevol zijn, omdat vanuit die achtergrond de eventuele 'gaten' in een informele specificatie veel duidelijker zichtbaar zullen zijn.

10. Bij het opzetten van een groot internationaal software project moet er vooral naar gestreefd worden om de benodigde hoeveelheid communicatie tussen de deelnemers te reduceren. Het optimaliseren van die communicatie is een probleem van de tweede orde.

11. De term 'browsermarkt' is misleidend omdat de meeste browsers helemaal niet verkocht worden.

12. Je merkt pas echt dat je naar het buitenland verhuisd bent als je een paar weken nodig hebt om uit te vinden waar je 'normaal' brood kunt kopen.