Agent-Based Query Optimisation in a Grid Environment[+]

Luciano Serafini [1], Heinz Stockinger [2,3] Kurt Stockinger [2,3], Floriano Zini [1]

1) ITC-IRST, Trento, Italy
serafini@itc.it, zini@itc.it

2) CERN, European Organization for Nuclear Research, Geneva, Switzerland
Heinz.Stockinger@cern.ch, Kurt.Stockinger@cern.ch

3) Institute for Computer Science and Business Informatics, University of Vienna, Austria

Abstract

The next generation experiments in High Energy Physics are the driving force for setting up an International Data Grid at CERN, the European Organization for Nuclear Research. Hundreds of Petabytes of data will be distributed and replicated all over the globe starting from 2005. In order to analyse this massive set of distributed data efficiently, we propose a hierarchical query optimisation architecture based on multi-agent technology. The architecture is optimised for the High Energy Physics community but is representative also for other data intensive scientific applications that use distributed data stores and mass storage systems.

Keywords:
query optimisation, distributed computing, agent technology

1 Introduction

The idea of building an International Data Grid [1,2] at CERN, the European Organization for Nuclear Research, was driven by the needs of the next generation accelerator, the Large Hadron Collider (LHC), which is scheduled to be in operation in 2005. Several Petabytes of data will be produced over a life time of 15 to 20 years which will be analysed by physicists all over the globe.

Current data analysis is based on local data stores which correspond to work stations of physicists and are mainly "private copies" of the basic "analysis data" generated at the experiments. Furthermore, the size of the local or "private" analysis data is mostly restricted by the capacity of the local hard disks. These data are derived from a central data store having a mass storage system (MSS), composed of a robotic tape system and disk pools, where data can be cached and loaded from tapes [3]. With the Data Grid, physicists will be able to access data within a fully distributed environment all over the globe, thus allowing more accurate data analysis due to the higher data volume available to each single physicist who can access data in distributed disk pools or MSSs even though there might not exist a big pool or MSS in his/her local institute.

In order for such a distributed system to work effectively, sophisticated query optimisation techniques [4,5,6] have to be applied. For example, to retrieve relevant data for multiple queries from several users from the MSS, it is necessary to maximise the exploitation of already cached data, therefore choose an execution order that performs in sequence queries that access the same (or almost the same) set of data. Conversely, in order to minimise the traffic of sending unnecessary data over the network, multiple queries on the same set of data might be better solved by duplicating a super set of the "analysis data" in the local site (this is known as "replication").

In this paper, we propose an architecture for a query optimisation system based on multi-agent technology [7] for a typical data intensive analysis scenario of High Energy Physics (HEP). However, the findings from this study can also be applied to other data intensive fields like earth observation, bio-informatics or climate modelling. The proposed architecture is based on a set of intelligent cooperative agents [8], autonomous software entities that can flexibly execute actions in their operative environment. This technology naturally allows us to model task-based organisations, where different tasks are identified and allocated to various agents [9]. In our architecture we support different sub-systems like the indexing component, the query scheduler, or the caching sub-system, as well as user queries. An agent carries out operations on behalf of a user or another program, and in this process represents or has knowledge of the user's goals and wishes. The main goal of the agents is to schedule queries from concurrent users to the appropriate resources in order to maximise query throughput [10], by either minimising access to the disk pools or the number of staging requests for different MSSs.

The architecture exploits current query optimisation technologies from distributed database systems and data warehouses. We discuss the possible use of some current technologies under the specific requirements of query optimisation for a Grid environment. Here, optimisation

[+] IASTED International Conference on Applied Informatics, Innsbruck, Austria, February 2001

has to take into account information about different resources (like the network bandwidth, the CPU load of the distributed machines, or the current state of the cache of the MSSs and their disk pools) and also the number of concurrent queries currently being executed or waiting to be executed.

2 Use Case: High Energy Physics Data Analysis

We guide our query optimisation by a real world application, namely data analysis in typical High Energy Physics (HEP) experiments. The features of such a data analysis are the following. Large amounts of mostly read-only data on the order of several Petabytes will be available at multiple sites distributed and partially replicated all around the globe. The actual analysis of data will also be done in a distributed way.

In general, data are written by the experiment, stored at very high data rates (from 100 MB/sec to 1GB/sec) and are normally not modified afterwards. This is true for about 90% of the total amount of data. Furthermore, since CERN experiments are collaborations of over a thousand physicists from many different universities and institutes, the experiments' data are not only stored locally at CERN. Parts of the data are stored at world-wide distributed sites, in so-called Regional Centres (RCs) and also in some institutes and universities. This is what creates the Grid.

In the HEP community, a single data set is a unit of related data objects and is called an "event". The current data model includes different types of data to represent events. Basically, there is a hierarchy of objects to be stored in a large object store. Initially, data which are produced by the detector are filtered by dedicated hardware and software. This has to be done in order to reduce the amount of data and only store useful parts of data (*raw data* or RAW). In the next step of a physics analysis task, a reconstruction function is used to bring a structure into the raw data. A reconstruction function takes raw data as input, produces new objects called *reconstructed data* objects and puts the newly created objects back into the object store. At a later stage, a further operation on a significant fraction or on all reconstructed objects is done which represents another refinement of objects. This operation can again be considered as a reconstruction process. The objects produced can also be called *event summary data* or *analysis object data*. The smallest data type we can distinguish is the so called *tag data* (TAG) that stores summary information events. Table 1 lists the four general data types, their storage amount per unit and the expected storage amount per year for a typical HEP experiment like CMS. The experiment is supposed to run and to produce data for about 100 days a year.

TAGs are supposed to be stored on disks only, whereas all the other data types are stored in a MSS and are temporarily cached in disk pools when they are used during the analysis.

Data type	Event size	Storage per year
RAW	1 MB	1 PB
Reconstructed data	200 KB	200 TB
Analysis data	10 KB	10 TB
TAG	100 B	1 TB

Table 1: Data types and data sizes for a HEP experiment.

TAGs that are used for physics analysis summarise the most important physics parameters (attributes) of particular events. They can be regarded as replicas of subsets of the whole data store. In fact, TAGs may have references to the other three levels of event representation that contain additional physics parameters. Whenever a query requests the analysis of these parameters, it is possible to access them following the reference included in the event TAG. Another possibility is reconstruction on the fly which means that some physics properties, in particular the particle tracks, are reconstructed from the RAW data during the end-user analysis job.

A typical physics analysis job starts by selecting a large initial collection of data sets which fulfil certain physics properties. These events are independent from each other which means that the physics result yielded by processing the collection of events (data sets) is independent of the sequence of processing each event. In other words, the processing order of the events could be changed according to some optimisation technique. In an analysis job a physicist applies some "cut predicates" (queries) on the data and thereby reduces the number of events in the event collection. The most common cuts can be regarded as multidimensional range queries where the potential search space consists of hundreds of independent dimensions. Queries typically cover 10 to 100 dimensions and are expressed by query predicates, for example:

$$(Energy > 100.5) \text{ AND } (pT1 < 83.7) \text{ AND } (pT2 \geq 92.6)$$

In a typical analysis effort, the number of resulting events, i.e. the found set, is iteratively reduced by applying cuts with smaller ranges or by adding more attributes to the cuts which results in a higher dimensional query. The results yielded by the cuts are mostly stored in histograms and plotted afterwards for analysing the physics properties. During other typical analysis efforts, arbitrary mathematical expressions of attributes can be used during the selection. A typical example looks like:

$$\sin(Energy) > 0.7 \text{ OR } \sin(Energy) \leq 0.3$$

Some attributes in a query predicate can be presented at the TAG level whereas other attributes can belong to lower data levels. If the latter is the case, a possible optimisation strategy is to select first the events on the basis of the TAG level parameters, and then refine the obtained result on the basis of lower level attributes. This strategy reduces expensive loading of data from tape.

3 Architecture

Let us now outline an architecture for optimising queries in a Grid environment with several distributed MSSs. At the abstract level, the architecture (see Figure 1) consists of the following components:

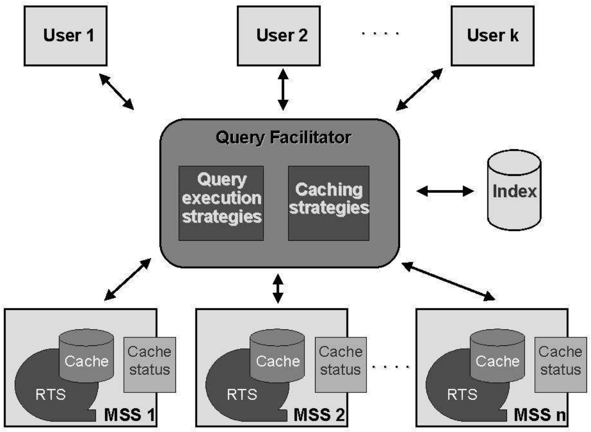

Figure 1: Abstract architecture of the system.

1. A set of users. Each user (U) may either be a single physicist or a group of physicists. In the first case, U represents the application that allows the user to submit his/her queries. In the second case, U represents the proxy server of the group of users that allows coordinated submission of queries by a group of users. Users are supposed to submit queries whenever they need data to examine. Also, they can discard an executed query at any time, for example after seeing the first few "unsatisfactory" results.

2. Query facilitator. The query facilitator (QF) is the heart of our architecture. Its activities can be described as follows: 1) It receives queries (expressed as query predicates) from the users. 2) It interacts with the indexing system to retrieve (an estimation of) the set of files that are needed to execute the queries. 3) It interacts with the set of MSSs to acquire information about their status, including information about the content of the cache, which files they store, their workload, and possibly other relevant information. 4) It performs optimisation of the execution of the set of queries that is based on information obtained from step 4. 5) It executes the optimised queries, distributes them (or sub-parts of them) among the various MSSs. Additionally, the query facilitator can receive from the MSSs the various sets of objects that correspond to (part of) the query, properly compose and send them back to the proper physicist. The users can also directly interact with the MSSs to retrieve data they need, once notified that the corresponding files are available. Further details on query optimisation are given in Section 5.

3. Indexing System. Query predicates are sent to the indexing system (IS) which gives a rough estimate about the retrieval time of the amount of data (the events) that are qualified by the queries. This estimation is passed back to the user by the query facilitator, and the user decides whether or not to proceed in query execution. In addition, the indexing system maps the logical events to physical files. If the user wants to proceed, the IS is activated again, in order to select precisely which events and which files are needed by the query.

4. Mass Storage Systems. A MSS is a hierarchical storage system with the following storage levels: main memory, disk pool, and tapes. It has a mechanised system (Robotics Tape System, or RTS) to mount and load tapes into the cache and a query executor that is responsible for retrieving the event-files needed to satisfy a query it receives as input from the QF. Actually, at this level, a query is expressed in terms of a set of files that have to be put in the cache of the MSS. These files are certainly stored on tapes, but they may already have been loaded in the cache memory when the query is processed. If this is the case, files are immediately accessible and the needed events can be fetched. Otherwise, the tapes containing the required files have to be mounted and loaded before performing the fetching of data.

The query facilitator should be given at least the following capabilities: 1) some strategies to be used for execution of queries; 2) some strategies to be used for caching files from tapes into the disk pool. For example, query execution strategies are used by the facilitator to arrange the order and the place of execution of a set of queries, on the basis of the replication of data, the current cache status of each MSS, and the set of attributes that queries need. Caching strategies are instead related to the mechanisms that are used to load or discard a file from a MSS cache. These strategies can include keeping in cache files that are most frequently used (and discarding those that are less used), or the files that are needed by the greatest number of queries.

One candidate for the indexing system are bitmap [11, 12] indices which give very fast answers to highly multi-dimensional query requests typical for this kind of physics analysis.

4 Agent Based Query Facilitator

An intelligent agent is a software entity that has some degree of autonomy. It carries out operations on behalf of a user or another program, and in this process represents or has knowledge of the user's goals and wishes. The concept of intelligent agents is used to describe a broad range of software functions. All of them are systems with behaviour which is:

- *situated*: the system has sensors and effectors to the environment

- *pro-active*: it takes initiative; goal directed behaviour
- *autonomous*: it decides what to do without external human intervention
- *timely:* it does not spend forever deciding what to do next
- *persistent*: a “lifetime” of extended activity sequences
- *social*: it interacts with other agents.

Mobility (i.e., the capability of moving from site to site for remote execution) is another feature of agents that is often considered. In this paper we do not discuss the adoption of mobile agents, because this would imply the discussion of several security issues, currently not yet addressed by the Data Grid community.

In a distributed setting such as that of a Data Grid one cannot think that the query facilitator is a monolithic architecture, which resides in a single point of the network. Furthermore, the query facilitator should be flexible enough to be able to cope with a very dynamic and asynchronous environment: users can appear and disappear, MSSs can become available or not available without any central control. In addition, managing a query is not a task that can be executed in a batch mode. There should be sufficient interaction with the users, the indexing system and the MSSs. For these reasons it seems that the agent technology is a promising one for the implementation of a robust, effective, and efficient query facilitator. Figure 2 shows a first attempt for an agent-based architecture for the query facilitator.

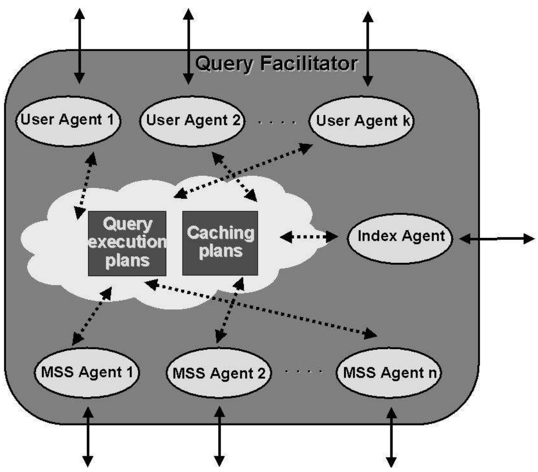

Figure 2: Agent architecture of the query facilitator.

We identify 4 types of agents: user agents, index agents, MSS agents and “internal” agents. The first three types are “wrappers” that allows us to integrate external heterogeneous autonomous systems into an interoperable architecture. The main function of the wrappers is to represent the systems they wrap in the whole architecture. For example, the main capabilities of user agents are to submit requests for query evaluation, execution, or interruption. Internal agents (not explicitly represented in Figure 2) should implement a hierarchical structure of agents that perform query optimisation at different levels of detail (e.g. based on knowledge about geographical distribution of MSSs, or historical logging of executed queries). This structure may range from a centralised query optimiser agent to a fully distributed network of agents. The adoption of a particular internal structure has to be carefully investigated and we have not commited yet to any configuration of internal agents.

The Belief, Desire and Intention (BDI) model [13] is an internal architecture for agents which allows to implement an agent in terms of its *beliefs*, i.e. the perceived understanding of the environment, its *desires* or *goals*, i.e. the objectives that it has to pursue, and the plans it uses to pursue such goals, and its *intentions*, i.e. the currently executing plans. We believe that an architecture based on BDI agents for a query facilitator is appropriate for the following reasons:

1. In the proposed architecture, there is no central control. The behaviour of the whole system is the result of the cooperative behaviour of its components. Indeed, each module, located at some point of the Grid, autonomously decides its behaviour on the basis of its internal state, the interaction with the other components, and the state of the environment. These components must be able to cope with asynchronous and inconsistent requests. They have to autonomously decide how to satisfy the requests coming from other modules. Agents can support very easily this kind of architecture.
2. Each architectural component has a reason, a goal, for existing and computing as a simple stand-alone system. Certain components (like for instance the mass storage system and the indexing sub-system) are systems already implemented that can be integrated in the whole architecture by means of wrapping agents.
3. The proposed architecture must be open. This means that it can scale up when new components participate in the scenario, for instance, new users, new mass storage systems, etc. Furthermore, the architecture can be enlarged with different components that will be available in the Grid. Agents allow the development of such open architectures.
4. This architecture must be able to cope with multiple users who submit asynchronous queries. The query facilitator is therefore supposed to be able to cope with parallel asynchronous solution and optimisation of queries. It is clear that a paradigm that supports parallel plan execution is very important and useful in such a setting.
5. Finally, different optimisation techniques, the scheduling mechanism and the caching policies can be easily modelled as so-called “plans” of the agents, which can interact with each other by message passing in order to handle the optimisation problem in an autonomous way depending on the dynamic parameters of the environment even though some information about the different sites is not available or uncertain.

To better understand what a *plan* is, let us give an example. Suppose that the internal structure of the query facilitator is composed only by a query manager agent that performs a centralised query optimisation. Whenever the query manager receives a query from a user, a plan for its execution is activated. A standard plan for query execution could consist of three high-level activities:

- interacting with the indexing system in order to retrieve references to a set of objects
- negotiating with agents that wrap the MSSs to determine which files to retrieve from each MSS
- requesting each MSS agent to make available a subset of the objects needed for a certain duration.

It is important to note that plans allow structuring the procedural knowledge of agents in quite a flexible way. For instance, the three activities listed above may easily be detailed in terms of sub-plans, and negotiation can be refined either in a centralised or distributed way. In the first case, the query manager might retrieve relevant information from the MSS agents and decide about query scheduling. In the second case, the query manager could start an auction among MSS agents, in which every MSS agent participates in trying to “sell” the objects that it has. Another important feature of agent plans is that their activation is context dependent. Suppose that the query manager has a memory to store queries, objects and files that have been requested. If the current query being processed by the query manager partially or totally overlaps another query being answered, the plan outlined above is not invoked. Instead, the query answer is simply built by fetching the information required directly from memory, without further negotiation or messaging.

5 Query Optimisation Techniques

We have already pointed out that the query facilitator is the heart of our architecture which needs to have the basic functionalities of a query optimiser. It is not our intention to implement query optimisation strategies from scratch. We rather try to “agentify” the most important and convenient existing strategies. These are: minimise the query response time; maximise the query throughput; minimise the optimisation costs for achieving this.

Typical static optimisation techniques use index data structures which are based on certain key values. Since we can distinguish between clustering and non-clustering indices, we identify already a further static optimisation approach, namely clustering of the original physical data layout on the secondary storage. Based on discovered access patterns in the query history, the data can be dynamically re-clustered in order to reduce the number of page I/Os for a query against the secondary storage system. However, since we are dealing with huge amounts of partially replicated data in a wide area network, this optimisation technique must be applied very carefully.

Prefetching and caching technologies are widely studied in the database community but also recently in the field of parallel I/O [14]. The main idea is, in the first case, to "prepare" the data for further computation which is predicted to be needed in the subsequent program execution. This approach shows its strength in performing a few large I/O operations, rather than many small ones, and thus yields a more or less sequential data access pattern rather than a random access one. Currently, the throughput of sequential disk access is between 5 to 10 times faster than the one for random access, which clearly shows that nearly sequential access should be done whenever possible [15]. Caching, on the other hand, tries to find out certain query containments and keeps the most frequently used data in cache. This technique can be regarded as the opposite optimisation strategy to prefetching.

Other well known dynamic optimisation approaches in database and data warehouse (DW) systems are so called materialised views. In other words, pre-computed query results are stored persistently and are re-used for subsequent queries with a similar pattern. We could also regard this technique as “persistent caches”. Current trends in maintaining materialised views in DW applications deal with optimising aggregation queries with expensive joins on two or more tables (for example, the sum of dollar income in the first quarter of the year of all affiliates of company X in region Y). Moreover, it is mostly the user who decides which query to materialise.

In our case we would need a slightly different usage of materialised views. Since the TAG-data of our data model only comprises one large table with several columns, there is no need for joins. On the other hand, we are hardly interested in aggregation functions like the total energy values of all particles of a certain physics property but more in results like: “give me all particles with an energy value between 90 and 150 GeV”. The system should thus be able to decide automatically when to materialise some query results. This obviously leads to a more complex management of materialised views than in traditional systems which need to be handled by a special data structure. In addition to the dynamic optimisation techniques discussed so far, “hot spots” could be efficiently tackled by replicating this kind of data, thus allowing for an inherent parallel access technique.

In traditional parallel and distributed database systems query optimisation for a single user is done by splitting the query into subqueries and evaluating them on the available resources where parts of the data reside. In a Grid environment with replicated data the sub-queries could be theoretically executed on all sites. However, the following factors need to be considered for an optimal response time: - find the optimal query plan with the lowest cost - find the “best” replica with respect to network traffic and processor load (hot spot detection) - find the processor with the best cache state, i.e. when the data resides already in the cache (disk pool of MSS) then no additional I/O needs to be done - find the processor with the access rights for fulfilling this task.

Since our real case scenario deals with concurrent users, we also want to introduce concurrency into our considerations. In the widest sense, the new challenge can be partially answered by prefetching and caching for a single user environment. However, we need a mechanism to insure some fairness among the concurrent users such that the system does not get overloaded by a “power user” who consumes all available resources. What is more, an authentication and authorisation mechanism needs to be introduced since different sites in the Grid may have different local security policies.

6 Future work and conclusions

We believe that the outlined architectural design is a first step towards solving the complex task of query optimisation within the Data Grid and also provides clean interfaces for interacting with other parts like Grid monitoring or Grid workload management.

Our first goal is to implement an agent prototype for the architecture we have presented. The target platform for implementation will be JACK Intelligent Agents™ by Agent Oriented Software [9]. JACK is an environment that extends Java for building, running and integrating commercial-grade multi-agent systems using a component-based approach. Since the main bottleneck of our architecture is the access to the tape drive, the prototype will initially be designed for minimising the amount of disk staging operations and then extended.

Acknowledgment

We want to thank Paolo Busetta who has given many useful suggestions about how the agent architecture could be realised as well as our colleagues in WP2 (Data Management) of the DataGrid project for useful discussions.

References

[1] W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger, K. Stockinger. Data Management in an International Data Grid Project. *1st IEEE/ACM Int. Workshop on Grid Computing (Grid'2000)*, Bangalore, India, December 2000, IEEE Computer Society Press.

[2] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, S. Tuecke, The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. To be published in the *Journal of Network and Computer Applications*.

[3] L. M. Bernardo, A. Shoshani, A. Sim, H. Nordberg. Access Coordination of Tertiary Storage for High Energy Physics Applications. *IEEE Symposium on Mass Storage Systems*, March 2000, College Park, MD, USA.

[4] S. Chaudhuri, An Overview of Query Optimization in Relational Systems, *Symposium on Principles of Database Systems*, June 1998, Seattle, Washington, USA, ACM Press.

[5] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, K. Shim.Optimizing Queries With Materialized Views. *Int. Conf. on Database Engineering*, March 1995, Taipei, Taiwan. IEEE Computer Society.

[6] D. Suci. Query Decomposition and View Maintenance for Query Languages for Unstructured Data. *Int. Conf. on Very Large Data Bases*, Sept. 1996, Mumbai (Bombay), India. Morgan Kaufmann.

[7] N.R. Jennings and K. Sycara and M. Wooldridge, A Roadmap of Agent Research and Development Autonomopus Agents and Multi-Agent Systems, *Kluwer Academic Publisher*, Volume 1, pages 7-38, 1998.

[8] H. Nwana and L. Lee and N. Jennings, Coordination in Software Agent Systems, *BT Technology Journal*, Volume 14-4, pages 79-88, 1996.

[9] P. Busetta and N. Howden and R. Rönnquist and A. Hodgson, Structuring BDI Agents in Functional Clusters, Intelligent Agents VI, ed. N.R. Jennings and Y. Lesperance, *Lecture Notes in Artificial Intelligence 1757*, Springer-Verlag, 2000.

[10] L. Serafini and C. Ghidini, Using wrapper agents to answer queries in distributed information Systems, *Proceedings of the First Biennial Int. Conf. on Advances in Information Systems (ADVIS2000)*, 2000.

[11] K. Stockinger, D. Duellmann, W. Hoschek, E. Schikuta. Improving the Performance of High Energy Physics Analysis through Bitmap Indices. *Int. Conf. on Database and Expert Systems Applications*, London - Greenwich, UK, Sept. 2000. Springer-Verlag.

[12] A. Shoshani, L.M. Bernardo, H. Nordberg, D. Rotem, A. Sim, Multidimensional Indexing and Query Coordination for Tertiary Storage Management, *11th Int. Conf. on Scientific and Statistical Database Management*, Cleveland, Ohio, USA, July 1999

[13] A. Rao and R. Georgeff. Modeling Rational Agents within a BDI-Architecture. In R. Fikes and E. Sandewall, editors, *Proc. of Knowledge Representation and Reasoning (KR&R-91)*, pages 473-484, San Mateo, CA, 1991. Morgan Kaufmann Publishers.

[14] J. No, R. Thakur, A. Choudhary, Integrating Parallel File I/O and Database Support for High-Performance Scientific Data Management, *Supercomputing Conference*, Dallas, Texas, November 2000.

[15] Koen Holtman, Prototyping of CMS Storage Management, Ph.D. thesis (proefontwerp), May 2000, Eindhoven University of Technology