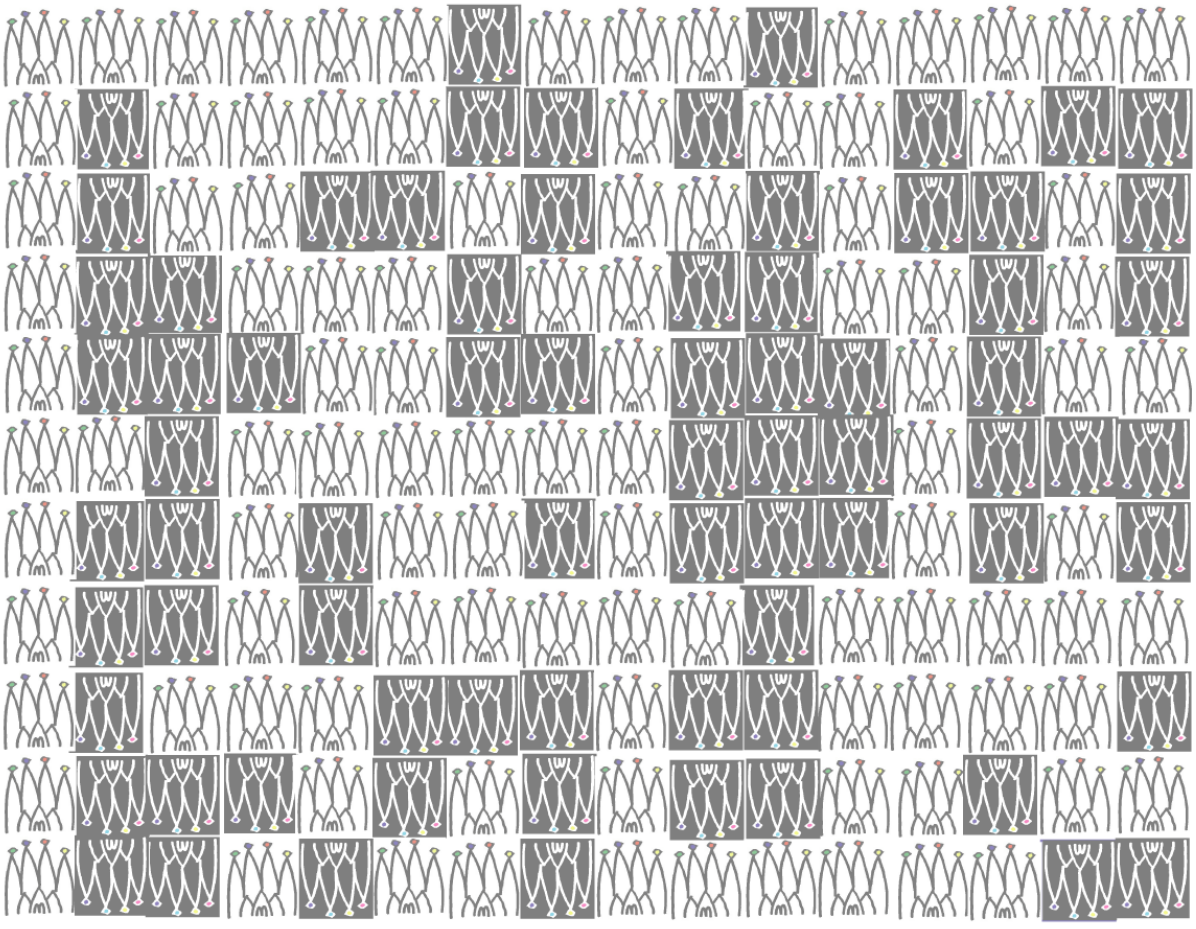
LHCb Computing Technical Design Report

R. Antunes Nobrega, A. Franca Barbosa, I. Bediaga, G. Cernicchiaro, E. Correa de Oliveira, J. Magnin, L. Manhaes de Andrade Filho, J. Marques de Miranda, H. Pessoa Lima Junior, A. Reis, et al.

CERN/LHCC 2005-019
LHCb TDR 11
20 June 2005

LHCb Computing TDR



Technical Design Report

The LHCb Collaboration

Computing TDR Authors List

Last update : 10/06/2005

Brasilian Center for Research in Physics, CBPF, Rio de Janeiro, Brasil
R. Antunes Nobrega, A. Franca Barbosa, I. Bediaga, G. Cernicchiaro, E. Correa de Oliveira, J. Magnin, L. Manhaes de Andrade Filho, J. Marques de Miranda, H. Pessoa Lima Junior, A. Reis

Federal University of Rio de Janeiro, UFRJ, Rio de Janeiro, Brasil
K. Akiba, S. Amato, T. da Silva, J.R.T. de Mello Neto, B. de Paula, L. de Paula, M. Gandelman, J.H. Lopes, B. Marechal, F. Marinho, A. Massafferri, E. Polycarpo

LAPP Annecy, IN2P3-CNRS, Annecy-Le-Vieux, France
D. Boget, A. Bret, P. Delebecque, I. De Bonis, D. Decamp, C. Drancourt, N. Dumont Dayot, C. Girard, B. Lieunard, M.-N. Minard, S. Moreau, B. Pietrzyk, T. Rambure, H. Terrier

LPC Clermont, IN2P3-CNRS and University Blaise Pascal, Clermont-Ferrand, France
Z. Ajaltouni, G. Bohner, D. Borras, C. Carloganu, E. Conte, R. Cornat, E. Delage, O. Deschamps, P. Henrard, J. Lecoq, M. Magne, S. Monteil, P. Perret, G. Reinmuth, C. Rimbault, A. Robert

CPPM Marseille, IN2P3-CNRS and University of Aix-Marseille II, Marseille, France
E. Aslanides, J. Babel, C. Benchouk, J.-P. Cachemiche, J. Cogan, B. Dinkespiler, P.-Y. Duval, R. Le Gac, D. Lunesu, V. Garonne, O. Leroy, P.-L. Liotard, F. Marin, A. Tsaregorodtsev

LAL Orsay, IN2P3-CNRS and University of Paris-Sud, Orsay, France
G. Barrand, C. Beigbeder-Beau, R. Beneyton, D. Breton, O. Callot, D. Charlet, B. D'Almagne, B. Delcourt, O. Duarte, F. Fulda Quenzer, B. Jean-Marie, J. Lefrançois, F. Machefert, P. Robbe, M.-H. Schune, V. Tocut, I. Videau

LPNHE Paris, IN2P3-CNRS and Universities of Paris VI and VII, Paris, France
M. Benayoun, L. Del Buono, P. David, G. Gilles

University of Dortmund, Dortmund, Germany
M. Nedos, B. Spaan

Max-Planck-Institute for Nuclear Physics, Heidelberg, Germany
M. Agari, C. Bauer, J. Blouw, H.P. Fuchs, W. Hofmann, K.T. Knöpfle, S. Löchner, A.-S. Mueller, M. Schmelling, B. Schwingenheuer, N. J. Smale

Physics Institute, University of Heidelberg, Heidelberg, Germany
S. Bachmann, F. Eisele, T. Haas, S. Henneberger, P. Igo-Kemenes, I. Kisel, V. Lindenstruth, U. Stange, U. Trunk, D. Wiedner, U. Uwer

Laboratori Nazionali dell' INFN, Frascati, Italy
M. Alfonsi, G. Bencivenni, C. Bloise, F. Bossi, P. Campana, G. Capon, P. Ciambrone, P. de Simone, G. Felici, C. Forti, G. Lanfranchi, F. Murtas, V. Patera[2)], M. Poli Lener, A. Saputi, A. Sarti, A. Sciubba[2)]

University of Bologna and INFN, Bologna, Italy
G. Avoni, G. Balbi, M. Bargiotti, A. Bertin, D. Bortolotti, M. Bruschi, A. Carbone, S. de Castro, P. Faccioli, L. Fabbri, D. Galli, B. Giacobbe, D. Gregori, F. Grimaldi, I. Lax, U. Marconi, I. Massa, G. Peco, M. Piccinini, N. Semprini-Cesari, R. Spighi, V. Vagnoni, S. Vecchi, M. Villa, A. Vitale, A. Zoccoli

University of Cagliari and INFN, Cagliari, Italy
W. Bonivento, S. Cadeddu, A. Cardini, V. de Leo, C. Deplano, A. Lai, D. Raspino, B. Saitta

University of Ferrara and INFN, Ferrara, Italy
W. Baldini, V. Carassiti, A. Cotta Ramusino, P. Dalpiaz, S. Germani, A. Gianoli, M. Martini, F. Petrucci, M. Savrié

University of Florence and INFN, Florence, Italy
A. Bizzeti, G. Graciani, M. Lenti, M. Lenzi, G. Passaleva, P.G. Pelfer, M. Veltri

University of Genoa and INFN, Genoa, Italy
S. Cuneo, F. Fontanelli, V. Gracco, G. Mini, P. Musico, A. Petrolini, M. Sannino

University of Milano-Bicocca and INFN, Milano, Italy
M. Alemi, C. Arnaboldi, T. Bellunato, M. Calvi, F. Garibaldi, C. Matteuzzi, M. Musy, P. Negri, E. Panzeri, D. Perego, G. Pessina, N. Poukhaeva, L. Trentadue[3)]

University of Rome, "La Sapienza" and INFN, Rome, Italy
G. Auriemma[5)], V. Bocci, C. Bosio, E. Dane[4)], D. Fidanza[5)], A. Frenkel, F. Iacoangeli, G. Martellotti, G. Penso, S. Petrarca, D. Pinci, W. Rinaldi, R. Santacesaria, C. Satriano[5)], A. Satta

University of Rome, "Tor Vergata" and INFN, Rome, Italy
G. Carboni, S. de Capua, R. Messi, E. Santovetti

NIKHEF, The Netherlands
G. van Apeldoorn(i,iii), R. Arink(i), N. van Bakel(i,ii), T.S. Bauer(i), M. van Beuzekom(i), J.F.J. van den Brand(i,ii), E. Bos, H.J. Bulten(i,ii), M. Doets(i), H. Groenstege(i), V. Gromov(i), R. Hierck(i), L. Hommels(i), E. Jans(i), T. Ketel(i,ii), S. Klous(i,ii), M.J. Kraan(i), M. Merk(i), F. Mul(ii), J. Nardulli(i), A. Pellegrino(i), G. Raven(i,ii), E. M. Rodrigues Figueiredo(i), H. Schuijlenburg(i), T. Sluijk(i), N. Tuning, P. Vankov(i), B. Verlaat, J. van Tilburg(i), H. de Vries(i), L. Wiggers(i), G. V. Ybeles Smit, M. Zupan(i)
(i) Foundation of Fundamental Research of Matter in the Netherlands
(ii) Free University Amsterdam
(iii) University of Amsterdam

Research Centre of High Energy Physics, Tsinghua University, Beijing, P.R.C.
M. Bisset, J.P. Cheng, Y.G. Cui, Y. Dai, Y. Gao, H.J. He, C. Huang, C. Jiang, Y.P. Kuang, Q.Li, Y.J. Li, Y. Liao, J.P. Ni, B.B. Shao,J.J. Su, Y.R. Tian, Q. Wang, Q.S. Yan

Henryk Niewodniczanski Institute of Nuclear Physics Polish Academy of Science and University of Science and Technology, Krakow, Poland
K. Ciba, L. Hajduk, A. Kowal, M. Kucharczyk, J. Michalowski, B. Muryn, A. Oblakowska-Mucha, G. Polok, M. Witek

Soltan Institute for Nuclear Studies, Warsaw, Poland
M. Adamus, A. Chlopik, Z. Guzik, A. Nawrot, K. Syryczynski, M. Szczekowski

National Institute for Physics and Nuclear Engineering, IFIN-HH, Bucharest-Magurele, Romania
C. Coca, M. Orlandea, C. Nicorescu, S. Stoica, P.D. Tarta

Institute for Nuclear Research (INR), Moscow, Russia
S. Filippov, J. Gavrilov, E. Guschin, V. Kloubov, L. Kravchuk, V. Kutuzov, S. Laptev, V. Laptev, V. Marin, G. Rybkine, A. Sadovski, I. Semeniouk, V. Strigin

Institute of Theoretical and Experimental Physics (ITEP), Moscow, Russia
I. Alekseev, A. Aref'ev[1)], S. Barsuk[1)], I. Belyaev, B. Bobchenko, G. Charkov, V. Dolgoshein, A. Golutvin, O. Gouchtchine, V. Kiritchenko, V. Kochetkov, I. Korolko, T. Kvaratskheliya, I. Machikhiliyan, S. Malyshev, M. Martemiyanov, E. Mayatskaya, E. Melnikov, A. Morozov, P. Pakhlov, G. Pakhlova, A. Petriaev, P. Pozolov, M. Prokudin, D. Roussinov, V. Rusinov, S. Semenov, S. Shuvalov, A. Soldatov, D. Svirida, E. Tarkovski, K. Voronchev, V. Zaitsev, O. Zhiryakova

Budker Institute for Nuclear Physics (INP), Novosibirsk, Russia
M. Barnyakov, K. Beloborodov, A. Berdiouguine, A. Bondar, A. Bozhenok, A. Buzulutskov, S. Eidelman, V. Golubev, P. Krokovnyi, A. Kuzmin, S. Oreshkin, A. Poluektov, S. Serednyakov, L. Shekhtman, B. Shwartz, Z. Silagadze, A. Sokolov, A. Vasiljev

Institute for High Energy Physics (IHEP-Serpukhov), Protvino, Russia
I. V. Ajnenko, K. Beloous, R.I. Dzhelyadin[1)], Yu.P. Gouz, K. Kachnov, I. Katchaev, V. Khmelnikov, V. Kisselev, A. Kobelev, A.K. Konoplyannikov[1)], A.K. Likhoded, N. Martchikine, V.D. Matveev, V. Novikov, V.F. Obraztsov, A.P. Ostankov, V. Polyakov, V.Romanovski, V.I. Rykalin, M.M. Shapkin, A. Sokolov, M.M. Soldatov, V.V. Talanov, O.P. Yushchenko

Petersburg Nuclear Physics Institute, Gatchina, St. Petersburg, Russia
G. Alkhazov, V. Andreev, B. Botchine, V. Ganja, V. Goloubev, S. Guetz, A. Kashchuk[1)], V. Lazarev, E. Maev, O. Maev, G. Petrov, N. Saguidova, G. Sementchouk, I. Smirnov, V. Souvorov, E. Spiridenkov, A. Vorobyov, An. Vorobyov, N. Voropaev

University of Barcelona, Barcelona, Spain
E. Aguilo, R. Ballabriga[6)], M. Calvo, A. Comerma Montells, S. Ferragut, Ll. Garrido, D. Gascon, C. Gonzales Bano, R. Graciani Diaz, E. Grauges Pous, S. Luengo[7)], D. Peralta, M. Rosello[6)], X. Vilasis[6)], J. Xirgu Aleixandre

University of Santiago de Compostela, Santiago de Compostela, Spain
B. Adeva, D. Esperante Pereira, C. Lois Gomez[7)], A. Pazos, M. Plo, J.J. Saborido, M. Sanchez Garcia, P. Vazquez Regueiro

Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland
A. Bay, J. Borel, B. Carron, P. Fauland, L. Fernandez, R. Frei, G. Haefeli, J.-P. Hertig, J. van Hunen, C. Jacoby, P. Jalocha, S. Jimenez-Otero, F. Legger, L. Locatelli, A. Perrin, T. Schietinger, O. Schneider, M.T. Tran, K. Vervink, S. Villa, H. Voss, N. Zwahlen

University of Zürich, Zürich, Switzerland
R. Bernet, R.P. Bernhard, J. Gassner, S. Koestner, F. Lehner, M. Needham, O. Steinkamp, U. Straumann, A. Vollhardt, D. Volyanskyy, A. Wenger, M. Ziegler[8)]

Institute of Physics and Technologies, Kharkiv, Ukraine
A. Dovbnya, Yu. Ranyuk, I. Shapoval

Institute for Nuclear Research, National Academy of Sciences, Kiev, Ukraine
V. Aushev, Y. Filipchenko, V. Kiva, Yu. Pavlenko, V. Pugatch, D. Volyanskyy

University of Bristol, Bristol, UK
N.H. Brook[1)], R.D. Head, Q. Jia, F. Metlica, A. Muir, A. Phillips, P. Szczypka, F.F. Wilson[9)]

University of Cambridge, Cambridge, UK
A. Buckley, J. Dickens, K. George, V. Gibson, K. Harrison, C.R. Jones, S.G. Katvars, C. Lazzeroni, J. Storey, C.P. Ward, S.A. Wotton

Rutherford Appleton Laboratory, Chilton, UK
C.J. Densham, S. Easo, B. Franek, R.N.J. Halsall, G. Kuznetsov, P. Loveridge, D. Morrow, J.V. Morris, A. Papanestis, G.N. Patrick, M.L. Woodward, Z. Zhang

University of Edinburgh, Edinburgh, UK
R. Chamonal, P. J. Clark, A. Earl, S. Eisenhardt, N. Gilarsi, J. Lawrence, J. McCarron, F. Muheim, S. Playfer, A. Smith, S. Thorn[10)], A. Walker, Y. Xie

University of Glasgow, Glasgow, UK
A.J. Flavell, R. Bates, A. MacGregor, V. O'Shea, C. Parkes, S. Paterson, D. Petrie, A. Pickford, A. Saavedra, F.J.P. Soler[11)], T. Szumlak, S. Viret

University of Liverpool, Liverpool, UK
S. Biagi, T. Bowcock, G. Casse, R. Gamet, M. George, D. Hutchcroft, G. Patel, T. Shears, I. Stavitskiy, M. Tobin, A. Washbrook

Imperial College, London, UK
L. Allebone, G.J. Barber, T. Blake, W. Cameron, D. Clark, P. Dornan, U. Egede, R. Hallam, A. Howard, S. Jolly, R. Plackett, D.R. Price, T. Savidge, G. Vidal Sitjes, D. Websdale

University of Oxford, Oxford, UK
M. Adinolfi, J.H. Bibby, C. Cioffi, V. Gligorov, N. Harnew, F. Harris, I.A. McArthur, R. Muresan, C. Newby, A. Powell, J. Rademacker, R. Senanayake, L. Somerville, A. Soroko, I. Stokes-Rees, P. Sullivan, S. Topp-Jorgensen, G. Wilkinson

CERN, Geneva, Switzerland
L. Abadie, G. Aglieri Rinella, G. Anelli, F. Bal, A. Barczyk, J.C. Batista Lopes, I. Bird, V. Bobillier, A. Braem, L. Brarda, J. Buytaert, L. Camilleri, M. Campbell, M. Cattaneo, Ph. Charpentier, J. Christiansen, M. Clemencic, J. Closier, P. Collins, G. Corti, C. D'Ambrosio, H. Degaudenzi, H. Dijkstra, M. Domenech-Saavedra, J.-P. Dufey, D. Eckstein, L. Eklund, M. Ferro-Luzzi, W. Flegel, F. Formenti, R. Forty, M. Frank, C. Frei, B. Gaidioz, C. Gaspar, P. Gavillet, A. Gouldwell-Bates, J. S. Graulich, A. Guirao Elias, T. Gys, J. Harvey, J.A. Hernando Morata, E. van Herwijnen, H.J. Hilke, R. Jacobsson, C. Joram, B. Jost, N. Kanaya, P. Koppenburg, D. Lacarrère, M. Lammana, T. Lastovicka, R. Lindner, M. Losasso, A. van Lysebetten, A. Maier, P. Mato Vila, M. Moritz[12)], J. Moscicki, M. Muecke, T. Nakada[13)], N. Neufeld, R. Nogueira Fernandes, J. Palacios, M. Patel, M. Pivk, W. Pokorski, S. Ponce, F. Ranjard, S. Roiser, T. Ruf, D. Ruffinoni, H. Ruiz Perez, B. Schmidt, T. Schneider, A. Schopper, A. Smith, F. Teubert, O. Ullaland, P. Vannerem, W. Witzeling, K. Wyllie

[1)] also at CERN, Geneva
[2)] also at Dipartimento di Energetica, University of Rome, "La Sapienza"
[3)] also at Universita degli Studi di Parma
[4)] also at Frascati, Frascati
[5)] also at University of Basilicata, Potenza
[6)] also at departament d'Engineria Electronica La Salle, Universitat Ramon Llull, Barcelona
[7)] now at University of Zürich
[8)] now at University of California, Santa Cruz
[9)] now at Rutherford Appleton Laboratory, Chilton
[10)] now at NESC, Edinburgh
[11)] also at Rutherford Appleton Laboratory, Chilton
[12)] now Finance in Frankfurt
[13)] also at Ecole Polytechnique Fédérale de Lausanne

Technical Associates Institutes
Espoo-Vantaa Institute of Technology, Espoo, Finland
University College, Dublin, Ireland

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

The LHCb experiment [1] [2], illustrated in Figure 1-1, is designed to study CP violation in the b-quark sector at the LHC and expand the current studies underway at the B-factories (Babar, Belle) and at the Tevatron (CDF, D0). The LHC, being a hadron-collider, opens the opportunity to study B-hadrons that cannot be produced at current B-factories, and the energy of 14 TeV, much higher than that of the Tevatron, allows an abundant production of B-particles (10^5 particles/s at the nominal luminosity).

The bb production cross section is 2 orders of magnitude smaller than the total cross section visible in the detector, and the decay modes of the b hadrons that are of interest for CP violation studies all have very low visible branching fractions, typically smaller than 10^{-4}. Hence a very selective and sophisticated trigger is needed. LHCb is planning to operate a 3-level trigger system [3] to select the events of interest. The L0 trigger is a hardware custom-designed trigger requiring high p_T leptons or hadrons. Its output rate is limited to 1.1 10^6 Hz out of the 40 10^6 crossings per second. For L0-selected events, a subset of information from a limited number of sub-detectors is readout into a farm of CPUs that perform a further selection using pure software (L1 trigger). L1 requires that high p_T particles have a large impact parameter with respect to the primary vertex. The rate is further reduced to 40 kHz. For those events that are selected, the full information of all the sub-detectors is readout into the same farm of computers where the High Level Trigger selection (HLT) is applied. As all information is now available, more accurate selections can be applied in the HLT in order to reduce the overall rate to 2 kHz.

.

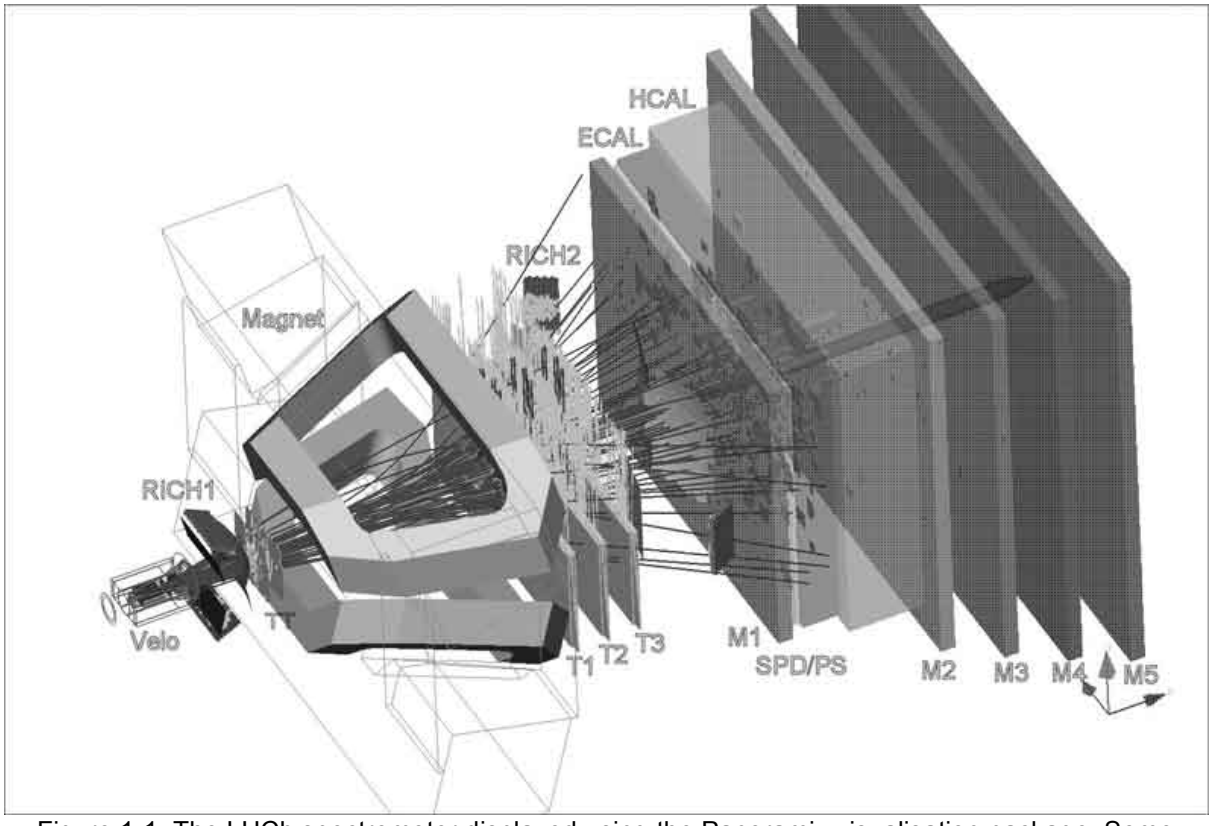


Figure 1-1: The LHCb spectrometer displayed using the Panoramix visualisation package. Some detectors are only shown partially to allow visualization of their measurements.

1.1 Requirements and scope

The Offline Computing system must allow the LHCb physicists to perform an efficient processing of the collected data (about 20 billion events a year), an accurate alignment and calibration of the sub-detectors and an efficient selection of events of interest as well as provide facilities for extracting physics results from the selected samples. The measurements aimed at by LHCb require a very high precision; hence systematic errors must be mastered to a very high degree. Amongst the 2 kHz of HLT-accepted events, a large fraction is dedicated to a very precise calibration and understanding of the detector and its capabilities.

Each group of physicists working on specific decay modes of B-particles will only handle a limited number of events; hence they rely heavily on a full central processing chain from the raw data to very elaborated and pre-selected reconstructed data. It is expected that individual analyses will cope with only a few million pre-selected events while manipulation of larger datasets will be handled centrally by a production team.

The Computing project is responsible for providing the software infrastructure for all software data processing applications (from L1 trigger to event selection and physics analysis). It is also in charge of coordinating the computing resources (processing and storage) as well as providing all the tools needed to manage the large amounts of data and of processing jobs.

In order to develop efficiently the software, for example developing L1 or HLT applications using simulated data, it is beneficial to implement a high level of standardisation in the underlying software infrastructure provided. Algorithms must be able to be executed in very different contexts, from the Online Event Filter Farm to a physicist's laptop. The Core Software sub-project is in charge of providing this software infrastructure.

The large amounts of data and of computing power needs imply that data processing must be performed in a distributed manner, taking best advantage of all resources available throughout the sites that allow the collaboration to use their resources. These resources (CPU and storage) are expected to be accessible through a standard set of services provided to all LHC experiments but also to the larger HEP community and beyond. The LHC Computing Grid project [4] is expected to provide these resources.

The LHCb Collaboration is fully committed to participate in the LCG by utilising and contributing to the common software projects as well as making full use of LCG computing Grid infrastructure. It is expected that LHCb will be able to benefit from the developments made inside LCG or available through LCG. In particular, the offline software uses the software developed by the LCG Applications Area. The distributed computing (data management and job handling) uses the Grid infrastructure deployed by LCG as well as baseline services provided through the LCG.

1.2 Overview

The present TDR describes first the architecture of the LHCb Offline software in Chapter 2. It covers the LHCb Software framework Gaudi as well as the main applications that are built on this framework.

The high-level tools needed for managing the LHCb Distributed Computing are described in Chapter 3. It covers the LHCb-specific services such as bookkeeping and file query, the distributed workload management system, DIRAC, and the end-user interface to Distributed Computing, GANGA.

The software and computing infrastructure described in the preceding chapters are used for data processing following the Computing Model described in Chapter 4.

The requirements of LHCb on the LCG software or services are described in the relevant chapters referred to above. We describe the current experience we had with the LCG-deployed infrastructure LCG-2 in Chapter 5.

Finally Chapter 6 presents the organisation of the project, the sharing of responsibilities and the planning for development and deployment of the described system.

Chapter 2 LHCb Software

2.1 Introduction

The LHCb software development strategy follows an architecture-centric approach as a way of creating a resilient software framework that can withstand changes in requirements and technology over the expected lifetime of the experiment. The software architecture, called Gaudi [5], supports event data processing applications that run in different processing environments ranging from the real-time L1 and high-level triggers in the on-line system to the final physics analysis performed by more than one hundred physicists. Object oriented technologies have been used throughout. The LHCb reconstruction (Brunel), the trigger applications (L1/HLT), the analysis (DaVinci) package, the digitization (Boole) together with the simulation application (Gauss) based on Geant4, and the event and detector visualization program (Panoramix) are all based on the Gaudi framework.

LHCb will produce large amounts of data, of the order of Petabytes per year, which will need to be reconstructed and analyzed to produce the final physics results. In addition, physicists are continuously studying the detector and the physics performance that can be achieved using it. Software for all data processing stages for the various needs of the experiment has been produced and is at different levels of deployment. This software will have to be maintained throughout the lifetime of LHCb, expected to be of the order of 10-20 years; the impact of changes in software requirements and in the technologies used to build software can be minimized by developing flexible and adaptable software that can withstand these changes and can be easily maintained over the long timescale involved.

With these goals in mind we have constructed Gaudi, a general Object Oriented framework designed to provide a common infrastructure and environment for the different software applications of the experiment. The applications, supporting the typical phases of Particle Physics experiments software, from simulation to reconstruction and analysis, are built within the Gaudi framework. Experiment specific software, as for example the Event Model and Detector Description are also provided within the framework as core software components. The framework together with these services and the applications constitutes the complete LHCb software system. The sub-detector software developers, or physicists performing analysis, provide the software algorithms to these applications. Use of the framework in all applications helps to ensure the integrity of the overall software design and results in maximum reuse of the core software components.

Tutorials with hands-on documentation are regularly held to train members of the collaboration. In addition, there are also specialized courses for software developers.

2.2 Gaudi Architecture & Framework

The development process for Gaudi is architecture-centric, requirements-driven, incremental and iterative. This involves identifying components with specific functionality and well-specified interfaces, defining how they interact with each other to provide the whole functionality of the framework. Whereas the architecture is the blueprint of the things to build, the framework is real code implementing the architecture and ensuring its design features are respected. The approach to the final software system is via incremental releases, adding to the functionality at each release according to the feedback and priorities of the

physicists developing the code for the different applications and following the evolution and changes in their needs.

A schematic view of the Gaudi architecture can be seen in the object diagram shown in Figure 2-1. It represents a hypothetical snapshot of the state of the system showing the objects (in this case component instances) and their relationships in terms of ownership and usage. Note that it does not illustrate the structure of the software in terms of class hierarchy. In the following we will outline the major design choices taken in the Gaudi architecture.

Classical Object Oriented programming assumes objects own the required functionality (methods) to transform themselves. Gaudi however considers the algorithmic part of data processing also as a set of OO objects. This decoupling between the objects describing the data and the algorithms allows programmers to concentrate separately on both. It also allows a longer stability for the data objects (the LHCb event model) as algorithms evolve much more rapidly. The Event Model classes only contain enough basic internal functionality for giving algorithms access to their content and derived information. Algorithms and tools perform the actual data transformations.

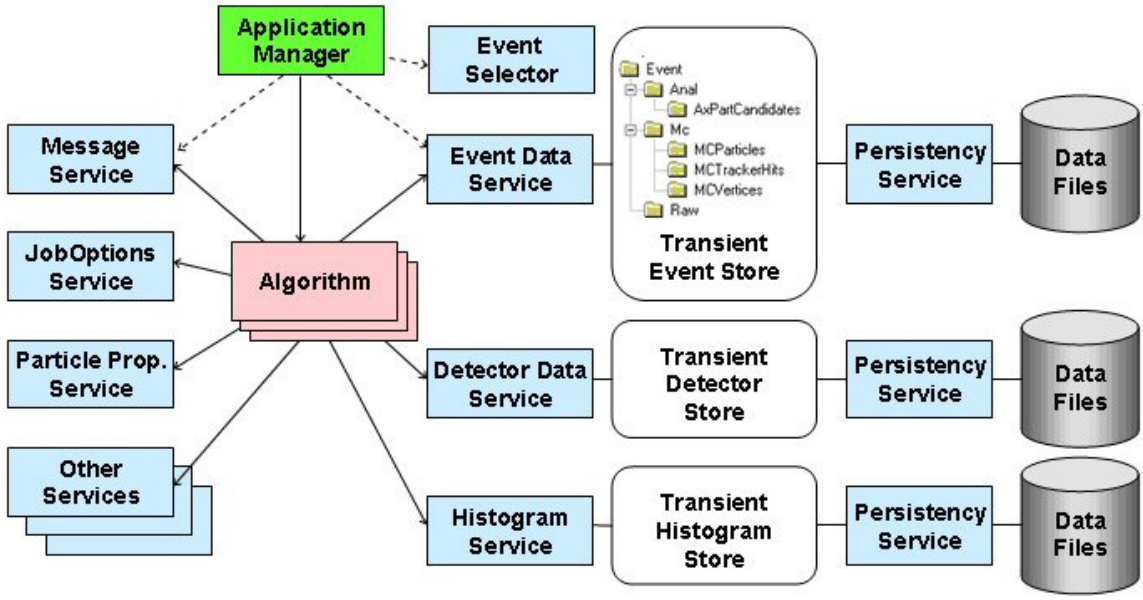


Figure 2-1: Object diagram of the Gaudi architecture

2.2.1. Generic component model with well defined interfaces

Each component of the architecture implements a number of interfaces (pure abstract classes in C++, the main language used in the implementation) for interacting with the other components. The basic idea of Gaudi is to define a set of services that are common to most of the event data processing applications. LHCb defined and developed the interfaces independent of their actual implementation. In order to ease the integration of components we defined an interface model supporting interface versioning, dynamic interface discovery and generic component factories. With these features we were able to implement run-time loading of components (dynamic libraries) allowing us to use a plug-and-play mechanism in the implementation of the data processing applications.

Since all components are essentially decoupled from each other, they can be implemented independently and in a minimal manner, i.e. supplying sufficient functionality to do their job but without the many refinements that can be added later. Components can be developed

using other specialized frameworks or toolkits, for example for data persistency, visualization, simulation, etc.

A specific implementation of a component can be replaced by another one implementing the appropriate interface and providing equivalent functionality. This makes possible a transparent use of *third-party* software. This approach has allowed us to build the LHCb applications by customizing the framework, i.e. by dynamically selecting the most suitable components to perform the different tasks. Due to these features, the Gaudi framework is easily adaptable for use in other experiments: although originally developed for the LHCb experiment it has been adopted and extended by the ATLAS experiment [6] and adopted by other experiments e.g. GLAST and HARP.

2.2.2. Separation between data and algorithms

Broadly speaking, the tasks of event simulation, reconstruction and analysis consist of the manipulation by algorithms of mathematical or physical quantities such as points, vectors, matrices, hits, momenta etc. This kind of task maps naturally onto a procedural language such as Fortran, which makes a clear distinction between *data* and *code*. A priori, there is no reason why using an object-oriented language such as C++ should change the way of doing physics analysis. This is the reason why the Gaudi application framework makes a clear distinction between *DataObject*s (essentially containers of data quantities) and *Algorithm*s and *Tools* that manipulate these data objects, i.e. that have well defined input and output data. Of course, intelligent data objects (e.g. tracks that know how to fit themselves) are possible, but they are discouraged in the Gaudi architecture.

While data objects essentially provide manipulation of internal data members, algorithms will, in general, process data objects of some type and produce new data objects of a different type.

Algorithms and *Tools* are themselves objects based on Gaudi base classes and they implement an extensive set of interface functions such as simple access to data, to all main services and run-time configuration facilities through job options.

2.2.3. Transient and persistent data

An important design choice has been to distinguish between a *transient* and a *persistent* representation of the data objects, for all categories of data. *Algorithms* see only data objects in the transient representation and as a consequence are shielded from the technology chosen to store the persistent data objects. In fact, so far, we have changed from ZEBRA [7] (for legacy data) to ROOT/IO [8] and more recently to POOL [9] without the physics code encapsulated in the *algorithms* being affected. The two representations can be optimized following different criteria (e.g. execution vs. I/O performance) and different technologies can be accessed (e.g. for the different data types).

2.2.4. Transient data stores

The data flow between *algorithms* proceeds via the so-called *Transient Store*. This not only shields them from the persistent technology but also minimizes the coupling between independent algorithms, allowing their development in a fairly autonomous way.

We have distinguished between three categories of data: *event data* obtained from particle collisions (real or simulated) and their successive processing; *detector data* describing the detecting apparatus (geometry, calibration, etc.) and *statistical data* derived from processing a set of events (histograms, Ntuples). They are not only conceptually different types of data,

their access pattern and their "lifetime" during a "job" is also different, hence we have organized them in corresponding separate transient data stores.

- The *Transient Event Store* contains the event data that are valid only for the time it takes to process one event.
- The *Transient Detector Store* contains the data that describe the various aspects of the behaviour of the detector (e.g. alignment) during a period of data taking corresponding to the processing of many events.
- The *Transient Histogram Store* contains statistical data that typically have a lifetime corresponding to a complete job.

Although the stores behave slightly differently, i.e. the clearing of the store is handled at different frequencies in the three cases, their implementation is based on a common transient store component, given the many things they have in common.

We have already mentioned that the data flow between algorithms proceeds via the transient store. In addition, the transient store acts as an intermediate buffer for any type of data that needs to be converted to a different type of data representation, in particular the conversion to persistent or graphical objects. Zero or more persistent or graphical representations of the data can correspond to one transient representation.

The data within the transient store is organized in a "tree-like" structure, similar to a Unix file system, allowing data items that are logically related (for example produced in the same processing stage) to be grouped together into a *data container*. Each node in the tree is the *owner* of everything below it and propagates its deletion to all items in its branches. To map Object Oriented data models onto a tree structure, object associations have been implemented using symbolic links in which ownership of the referenced items is left to the node holding them in the transient store.

2.2.5. Algorithms

Algorithms are the essence of the data processing applications and where the physics and sub-detectors code is encapsulated. Due to the fact that algorithms implement a standard set of generic interfaces they can be called by the framework without knowing the details of their implementation. The *application manager* knows which algorithms to instantiate and when to call them. It is configured by a set of job options.

The algorithms' execution is scheduled explicitly by configuring the application manager or by the execution of the *Data On Demand* service: one can instruct this service to run a specific algorithm when requesting a specific object container that does not exist yet and cannot be retrieved from the persistent store.

Complex algorithms can be implemented by using a set of simpler ones; a more elaborate *sequence* can be configured in the applications in order to support filtering and branches. These can, for example, be combined with multiple output streams to provide event filtering and selections. The different LHCb data processing applications are customized by choosing the appropriate set of algorithms or sequences to be executed.

2.2.6. Tools

Tools are lightweight algorithmic objects whose purpose is to help other components in performing their algorithmic work. They are in essence very similar to algorithms, but can be re-used by several components in order to perform a given task. They contain a piece of code that can be executed with different frequency (only for some events or many times per event);

they are convenient for processing individual transient data objects or data objects that are local to the component.

Different components may wish to share the same algorithmic operation *as-is* or configure it slightly differently (e.g. different event selection algorithms will want to combine reconstructed particles to make vertices). Hence tools can be either generic or owned by a component (an algorithm or another tool).

2.2.7. Services

This category of components offers the services common to most of the applications. They are generally sizable components set up by the framework at the beginning of a job and used by the algorithms as often as needed. This approach avoids the algorithm developers having to write routine software tasks that are typically needed in a physics data application. Some examples of services can be seen in Figure 2-1.

2.2.8. Core Services

The Gaudi framework is decomposed into a number of independent sub-frameworks to provide the basic software tasks typically needed in an application. Many of these services use *third-party* components. This allows LHCb to profit from existing software and helps in minimizing development and maintenance efforts.

The basic kernel of the framework, together with a set of utility services, constitutes the *General Framework Services* amongst which:

- The *Job Options Service*: used to configure the applications at run-time. Components declare at construction time a set of *named Properties* that are associated to data members. The default values of these data can be overwritten by values provided in a set of Job Options files. They are referred to by the instance name of the component and their property name. Basic C++ types are supported for job options.
- The *Message Service*: allows components to produce labelled output. Each message is associated a level that allows run-time filtering of messages.
- The *Event Data Service* allows containers to be retrieved from the Transient Event Store.
- The *Histogram Service* provides a technology neutral handling of histograms.
- The *Random Number Generator Service*: allows a uniform usage of random numbers by all algorithms.
- The *Object Persistency Service:* a technology-neutral service has been developed and interfaced with the framework, given the fact that a single persistency technology may not be optimal in all cases. The persistency mechanism has been designed such that the best-adapted technology can be used for each category of data. The LCG POOL framework [9] is based on a similar architecture allowing the client code to be technology free. POOL has replaced the LHCb ROOT/IO [8] based persistency solution previously in place. This allows LHCb to benefit from the additional functionality provided by POOL such as file catalogues and event collections.
- The *Conversion Service* allows specific *Converters* to be invoked when accessing specific classes. The *Converter* is in charge of instantiating the actual object in the Transient Store. They can eventually perform complex conversions or calculations.
- The *Detector Description Service* allows detector-related information to be available to the physics applications providing a generic description of the structure of the

geometry. The aim is to have a unique description of the detector for all applications (e.g. simulation and reconstruction). The physical and logical description of the LHCb detector as well as sub-detector specific data resides in a Detector Description Database (DDDB) or in the Conditions Database (see section 2.4) that provides the persistent storage of the detector data. Several versions of the DDDB following the evolution of the LHCb detector design have been produced. Reconstruction algorithms access the geometry information through an interface (*DetectorElement*) that is customised to fulfil the need of a specific sub-system. This service is described in more details in section 2.5.

- The *Data Dictionary Service* provides a high level modelling language to define the event object model, independent of the language used in the current implementation (i.e. C++) [10]. The description language chosen is XML, which provides a very strict syntax in addition to being very flexible. A Gaudi parser package (Gaudi Object Description) automatically produces the C++ header files. This approach ensures adherence to coding conventions, consistent sets of member functions, standard ways of cross-referencing objects, and documentation lines in the format required by the code documentation tool (Doxygen [11]). The service also provides runtime introspection information for object persistency and interactive analysis making use of the LCG object dictionary provided by the SEAL project [12].

Definition and implementation of interactive services, graphical interfaces and scripting tools are provided in *User Interaction* services.

Finally, specialized frameworks for simulation, analysis tools (not the tools themselves) and data visualization have been put in place; they are discussed in more detail in sections 2.6 and 2.7.

2.3 The LHCb Event Model

The set of classes (and relationships between classes) that describe the LHCb event data, together with the conventions governing their design and organization, are known as the *LHCb Event Model* [13].

2.3.1. Objects in the Transient Event Store

In the Gaudi architecture, algorithms communicate with each other by exchanging data via transient data stores. In particular, the Transient Event Store (TES) is used to exchange event data inside the event-processing loop; algorithms retrieve their input data on the TES, and publish their output data to the TES. They are not interested in knowing how (by which algorithm) their input data was produced, they just need to find it in a well defined location and in a well defined state. This, of course, imposes some discipline on the use and organization of the TES and requires some conventions.

The Gaudi TES is organized as a tree structure (by analogy with a file system) of nodes (directories) and leaves (files). In the LHCb Event Model, this tree is structured as a number of sub-trees, corresponding to the output of each processing step. Typically each sub-tree has a number of branches and sub-branches, ending with the leaves containing the event data. This hierarchical structure is chosen to simplify and optimize navigation within the TES. Figure 2-2 shows a part of the LHCb Event structure in the TES, highlighting the difference between nodes and leaves.

Figure 2-2: Part of LHCb Event structure in the TES.

An essential feature of the Gaudi TES is that only objects inheriting from a base-class *DataObject* are directly accessible from the store. This can be either a single object (e.g. the Event Header) or more generally a container of objects that cannot be retrieved individually from the TES (e.g. the set of MC Particles). By convention, algorithms may not modify data already on the TES, and may not add new objects to existing containers. This implies that a given container can only be manipulated by the algorithm that publishes it on the TES, but ensures that subsequent algorithms that are interested in this data can be executed in any order, and greatly simplifies the integration of complex data analysis applications.

In the LHCb event model we use a special type of container (*Keyed Container*) that can only contain *Keyed Object*s – i.e. objects that can be identified within their container by means of a unique *Key*. Relationships between *Keyed Objects* can then be implemented as references consisting of a container name (or rather, an index in a table of container names) and a Key that is unique in the container. The container ensures the uniqueness of the Key; the default case is that the Keyed Container assigns a unique Key when the Keyed Object is inserted. In cases where the Key has a physical meaning (for example an electronics channel identifier), it can be defined when creating the Keyed Object, but then the Keyed Container only allows insertion if an object with the same Key does not already exist.

2.3.2. Relationship between objects

Explicit relationships between classes in the data model can occur as data members of the target class (defined as the result of the processing of a source class), but only between classes adjacent in the processing sequence, as shown in Figure 2-3. For example *Tracks* can contain pointers to *Clusters* but neither to *Digits* nor *Particles*.

In the LHCb Event Model there is a clear separation between reconstructed data and the corresponding Monte Carlo Truth data. There are no references in Digits that allow transparent navigation to the corresponding *MC Digits*. This allows using exactly the same classes for reconstructed real data and reconstructed simulated data. The relationship to Monte Carlo is preserved by the fact that the *MC Digits* and the *Digits* use the unique electronics channel identifier as a *Key*; any reconstructed object (such as *Clusters*) can refer to one or more electronics channels via their channel identifier, which is valid for both real and simulated data.

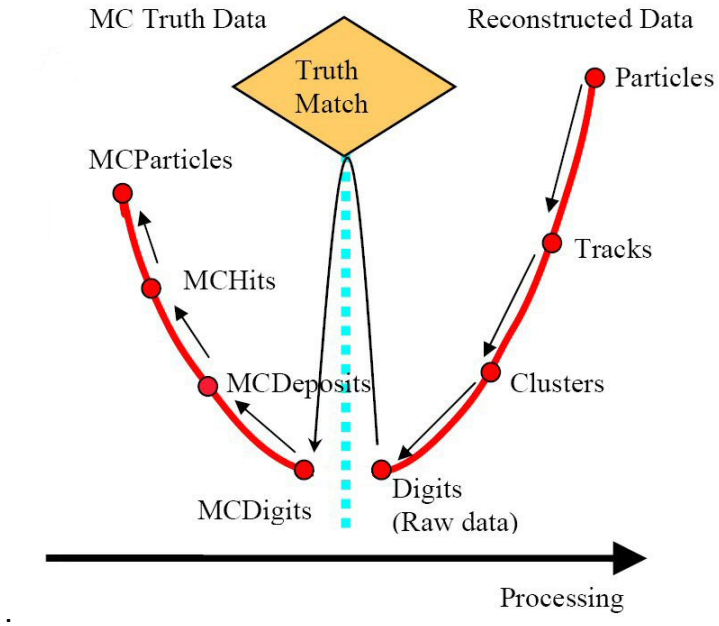


Figure 2-3: MC Truth Relation.

The direction of the direct reference between classes is from the class further in the processing sequence towards the class earlier in the sequence (this is a constraint imposed by the TES convention that already published objects cannot be modified). These relationships, shown as arrows in the figure, are implemented as *SmartRef*s (extended pointers allowing references to objects in other containers, possibly made persistent in another file). They can be de-referenced directly in the code, and used just like C++ pointers.

One may however want to study also the relationships between objects distant in the processing chain (e.g. which MC Particle gave rise to a specific Cluster), or in the direction opposite to the processing chain (e.g. what are all the MC Hits produced by a given MC Particle). It would be very inefficient to do this by following the *SmartRefs*, particularly if one has to navigate through many intermediate objects that may even reside in different physical files. An alternative is to calculate the relationship once only, and store it in a table that is then accessed by the association code. Two implementations of these tables are available [14][15], one of which (*Linkers*) is more appropriate for tables that have to be made persistent, whereas the second (*Relations*) offers additional functionality when used in a purely transient context.

2.3.3. Gaudi Object Description

The event classes are described in XML, using the Gaudi Object Description language (see section 2.2.8). The class header files are automatically generated from the XML, including the inline implementation of all simple methods (e.g. set and get methods for data members, serialization methods for printing), a mnemonic *typedef* for the templated *Keyed Container* of this type, and a static string containing the default location in the TES of objects of this type, which is used by algorithms to publish and retrieve data on the TES. This not only ensures a uniform look-and-feel to the event classes, but also simplifies migration to new versions of the underlying Gaudi and LCG software, because all the implementation-specific dependencies are encapsulated in the code generation tool.

The event model classes have been thoroughly reviewed before being implemented, and changes have to be widely discussed and well justified before being approved. This is particularly important for classes that are stored on persistent datasets, and will become more so in future, to ensure that older data can continue to be read by the newer implementation of the classes. The problem of schema evolution is being addressed both in the context of the LCG persistency solution (POOL [9]), and via the conversion mechanism of the Gaudi data stores, which makes it possible to have different classes in the persistent and transient worlds, with algorithms triggered automatically to convert between the two. It is planned to take advantage of this mechanism to minimize the dependency of persistent classes on external packages such as CLHEP, without imposing such unnecessary restrictions on the corresponding transient classes.

2.3.4. Buffer Tampering

The precision measurements of LHCb require a detailed understanding of systematic effects introduced at different phases of data taking and data analysis due to the applied selection algorithms. The motivation of Buffer Tampering is to determine these effects from real data instead of relying on Monte Carlo simulations. The two main goals are to estimate the biases introduced by the trigger levels and to calculate the acceptance along the flight path of a B-meson.

The implementation is based on a conditional modification of input data at the beginning of the raw data processing i.e. modifications of L1 and Raw Buffers that come directly from the readout chain. In the case of the trigger bias, the raw data related to the reconstructed B-decay chain is removed from the L1 and Raw buffers, while for the lifetime acceptance case, data is added to mimic the same B-decay but wtih a different decay lenghth. The on-line algorithms are emulated in the off-line phase of the data processing. This requires using the same software and the same conditions database as has been used during the on-line trigger execution. The key elements of the implementation are the high level manipulation of the transient event store (TES) and an interaction with the application manager from inside an algorithm. A set of dedicated tools has been developed which allow to:

- Move a sub-tree of the TES to a temporary location of the TES. This leaves only references to the objects at the original location.
- Reload the data in their original form to allow independent manipulation, called "tampering"
- Restore the tree from a temporary location back to the TES and delete temporary space for a new event.
- Send a request to the Application Manager to execute a given sequence of algorithms.

A first version of the Tampering algorithm is currently used successfully in the estimate of the wrong tag fraction for signal events by using calibration channels, which are generally triggered differently. There it is necessary to understand in detail the source of a positive trigger, in order to equalize the phase space of calibration and signal events before estimating the expected wrong tag fraction in signal events.

2.4 Conditions Database Services

The Conditions Database (CondDB) is a database facility that permits the handling of information regarding the running conditions of LHCb sub-systems that may vary with time.

A condition can be any kind of information that an algorithm may need, like the temperature or the pressure inside an element of the RICH as well as the alignment constants of the stations of the VELO. Each condition value has an interval of validity and can be superseded by a newer version (better alignment or re-calibration of probes). A set of conditions can be grouped together under a logical name, referred to as a *tag*. Figure 2-4 shows a schematic view of the 3-dimension space in which conditions live: data item, time and version.

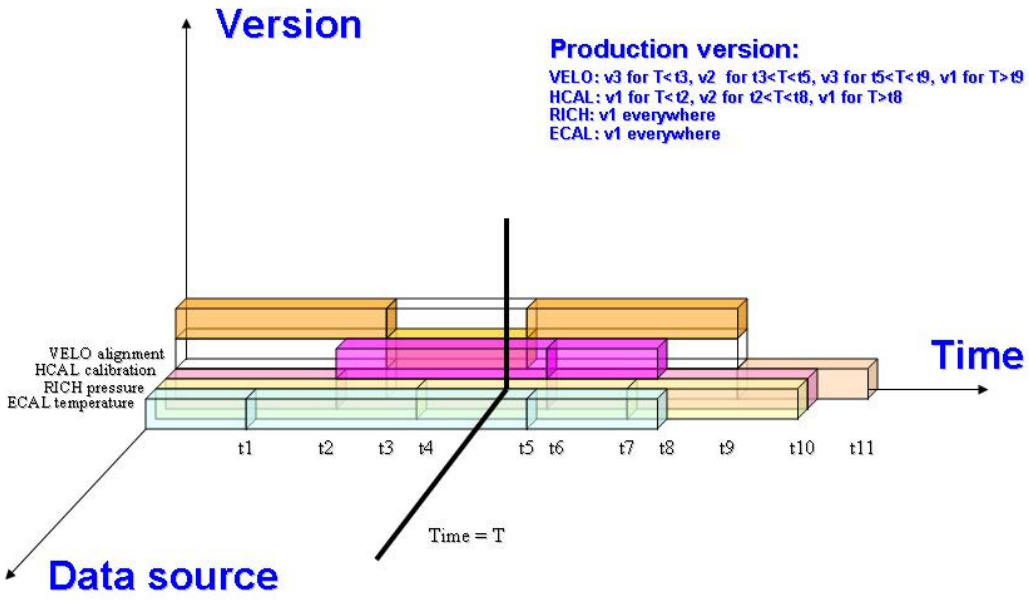


Figure 2-4: The three axes for identifying uniquely each data item in the condition database

The aim of the Gaudi CondDB service is to provide a framework integrated in Gaudi that allows users to use conditions data. Two main issues can be identified: the database access, and the update of the transient objects; they will be discussed in the following sections.

2.4.1. Database access

The access to a specific database implementation is obtained using a project developed by the LHC Computing Grid (LCG), named COOL [16][17](section 2.4.3) The usage of the COOL library is hidden to the general user in order to disentangle as much as possible user algorithms from the technical details of the underlying library. The connection to the specific Relational Database Management System (RDBMS) is encapsulated in a dedicated service, the *CondDB Access Service* that can be configured in order to specify the connection parameters (user name, database host, etc.) and the *tag* name identifying the set of conditions

to be used by the job. The interfacing to COOL is obtained by exploiting the "conversion service" mechanism, which is part of the general Gaudi framework.

Conditions reside in memory in the Transient Detector Store (TDS), as they are usually related to detector information (alignment, calibration, etc.) The TDS is aware of the existence of a specific *CondDB Conversion Service*. This service passes the request to the most appropriate converter for the requested object. The converter accesses the database through the *CondDB Access Service*, which returns a pointer to the object representing the open *CondDB*. Then the converter uses COOL for retrieving the condition data valid at the current event time and converts it to its transient representation.

The GPS event time (recorded in the event raw data) is passed to the TDS for each new processed event and is then used by the conversion service machinery enabling it to find the condition data object with a validity range compatible with the event being processed.

The implementation of detector data description is based on XML files and converters for the condition objects from XML are already available. In order to avoid replication of code, the XML conversion service was adapted to handle not only files, but also strings, thus allowing the storage of XML strings in the CondDB.

The objects retrieved from the CondDB have a validity range and for each event one must ensure the validity of all conditions. Accessing the database for each event in order to be sure that the conditions are still up-to-date is not advisable. Hence, at the beginning of each new event in the event loop, one has to check if all objects are still valid for the event that is going to be processed and, only if they are not, get the valid object from the database. The Transient Store allows users to get normal pointers to objects in the store and guarantees that those pointers are always valid; hence the new value has to be stored at the already existing memory location.

Special consideration applies to the usage of the CondDB by algorithms running on-line. During data taking, the Event Filter Farm (EFF) nodes will not be able to access a real database; hence a special *Online CondDB Access Service* will provide conditions uploaded by the control system, without the intermediate step of a physical database. Newly uploaded conditions will invalidate existing conditions at a predefined validity time; the new value will be cached temporarily and replace the current value when a new event enters the new validity range. This mechanism associated with a slight post-dating of Online conditions changes is essential to allow reproducibility of results obtained in the EFF when repeating them Offline.

2.4.2. Update Mechanism

Simple condition objects are not the only ones that need to be updated. Complex objects in the detector data store, like the *DetectorElement*, will use condition objects like alignment constants. User-defined classes deriving from such complex objects will possibly need to re-compute cached quantities using new values of the conditions.

Other objects that are not in the data store and do not implement the validity interface may also cache quantities that depend on conditions, for example algorithms and tools.

All these objects register their dependencies to an *Update Manager* service. A dependency can only be declared on an object that has the validity interface or that has already been declared to the Update Manager. A method is associated to each dependency that will be called whenever the dependent object is updated. That method will in general be the same that is used for caching information at initialisation time.

All these objects have to be considered invalid and need to be updated, if any of the conditions or other objects they depend on becomes invalid. In order to quickly check if such

a complex object is valid or not, the Update Manager assigns to it an interval of validity calculated as the intersection of the intervals of validity of all the objects it depends on. When the validity of the object does not contain the current event time, it means that at least one of the objects it depends on is not valid for that time. Thus the tree of dependencies can be efficiently navigated from the top level to the far leaves, updating all and only those objects that really need it without having to check the validity of all the objects in the store; when a branch is found to be valid, the recursion stops.

2.4.3. COOL Library

The access to an actual RDBMS implementation and the database schema is achieved using the LCG COOL library, part of the POOL project.

From the user point of view, the CondDB looks like a tree where the leaf nodes can hold condition data objects. The leaf nodes are called *Folders*, while the nodes that contain other nodes are called *FolderSets*. The hierarchical structure allows a logical organization of the conditions, for example one can put all the *Folders* for the conditions needed by a sub-detector in a dedicated *FolderSet*, or all the temperatures measured can go in *Folders* within the "Temperature" *FolderSet*.

The COOL API provides two types of *Folders*: single-version and multi-version. The first type can only store conditions values with Intervals Of Validity (IOVs) that do not overlap, so there is no possibility of superseding them. The second type allows the storage of conditions values with overlapping IOVs. Single-version *Folders* are less flexible than multi-version ones, but have better performance for insertion speed and storage space.

In a multi-version *Folder*, the most recent version of all the conditions values stored is called the HEAD version. At any time it is possible to take the HEAD version and give it a logical name or tag, allowing users to retrieve always a defined set of versions while detector responsible people can produce refined versions of the conditions data.

The actual RDBMS implementation can be chosen between ORACLE [18], MySQL[19] and SQLite[20].

2.5 Geometry Framework Services

The geometry framework serves three defined purposes:

- Providing geometry information to algorithms by combining in a transparent way the values of the nominal alignment with the measured deviations obtained from the conditions database and valid for the current event.
- Providing a mechanism to modify deviations from the nominal alignment without accessing the conditions database. This is required for example during the execution of an iterative alignment procedure. The framework must ensure that the modifications are propagated coherently to the detector geometry description.
- Providing a mechanism to update the deviations in the conditions database.

2.5.1. Detector Description Service

The Gaudi Detector Description Service provides a hierarchical description of the detector elements defined as volumes. Volumes are supported in a hierarchical tree. In order to simplify the description of repetitive volumes, it uses the concept of Logical and Physical volumes. It also contains the description of the material out of which the volumes are made; this information is needed for simulation as well as for track fitting.

A logical volume represents the shape of an object and its composition without a particular position is space. A physical volume consists of the placement in space of a physical volume within a higher-level logical volume. The top-level volume contains the whole LHCb apparatus and part of the cavern.

The hierarchy of volumes is defined using XML as a meta-language. XML files are maintained by individual sub-system groups and placed in a hierarchy of files.

XML files describe an ideal detector or the best known positioning of sub-detectors obtained as a result of a geometrical survey. Fine-tuned alignment constants obtained by running sophisticated offline algorithms are represented as small transformations from this ideal geometry. As they may vary with time and several versions of an alignment valid at a certain time may arise, they are very conveniently stored in the Conditions Database.

The access to geometry information for any algorithm is done via the *DetectorElement* interface. Alignment requirements imply that there exists a *DetectorElement* instance for each "alignable" component of the LHCb detector or that there exist intelligent *DetectorElements* capable of associating the right misalignments to their corresponding daughter elements. *DetectorElements* can be organised in a hierarchical tree describing more and more precise elements of the detector. The granularity needs to be defined by each sub-detector.

The *DetectorElements* as well as the hierarchy of volumes that are attached to it are stored in a dedicated Transient Store: the Transient Detector Store (TDS). They are accessible as in any TS by a path similar to that of a hierarchical file system. Their lifetime is that of the application, contrary to the Transient Event Store that is cleared after each event. Figure 2-5 shows an example of the detector description hierarchy in the TDS.

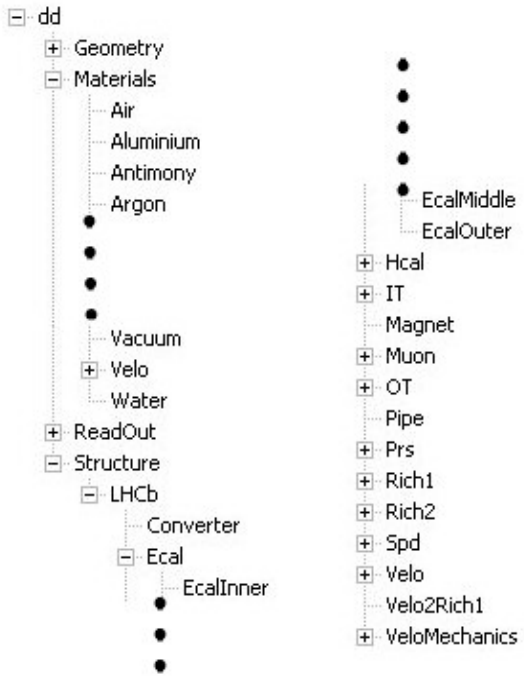


Figure 2-5: Browsing view of the TDS showing hierarchy of Detector Description as well as material description objects.

2.5.2. Misalignment in the Conditions Database

The alignment information itself is encapsulated in the *AlignmentCondition* class, which contains a transformation matrix for each alignable object. The transient store path for each set of alignment parameters is defined in an XML tree and can refer to a location in the conditions database or to an XML file holding the parameters.

Dedicated converters have the role of instantiating each *Alignment Condition* object starting from the alignment parameters. References to these addresses are stored in the XML definition of the corresponding *DetectorElements*. Each *Alignment Condition* object contains the transformation matrix representing the deviation from the ideal alignment between the *DetectorElement* it corresponds to (or daughter physical volume in the case of an intelligent *DetectorElement*) and its parent volume. The bridging between the local detector frame and the global frame is handled by the *Alignment Info* class, which has access to the *Alignment Conditions* of all parent *DetectorElements*, thereby calculating the transformation matrix in the global LHCb frame.

Through the *Alignment Info* object, a *DetectorElement* can perform transformations to and from that frame. These transformations can be combined with those corresponding to the nominal geometry, as defined in the detector description database. The nominal geometry information is available to the *DetectorElement* via the *Geometry Info* class, whose interface allows for transformations to and from global and local reference frames, and allows access to the corresponding transformation matrices. The deviations from the nominal geometry are accessed via the *Alignment Info* class as well as the combination of the two.

2.6 Data Processing Applications

Typical phases of Particle Physics data processing have been encapsulated in the various LHCb applications. Each application is a producer and/or consumer of data for the other stages as shown in Figure 2-6. The applications (including those that run online) are all based on the Gaudi framework, they share and communicate via the LHCb Event model and make use of the LHCb unique Detector Description. This not only ensures consistency between the applications but allows algorithms to migrate from one application to another as necessary. The subdivision between the different applications has been driven by their different scopes (simulation and reconstruction) and convenience (simulation of the events and detector response) as well as CPU consumption and repetitiveness of the tasks performed (reconstruction and analysis).

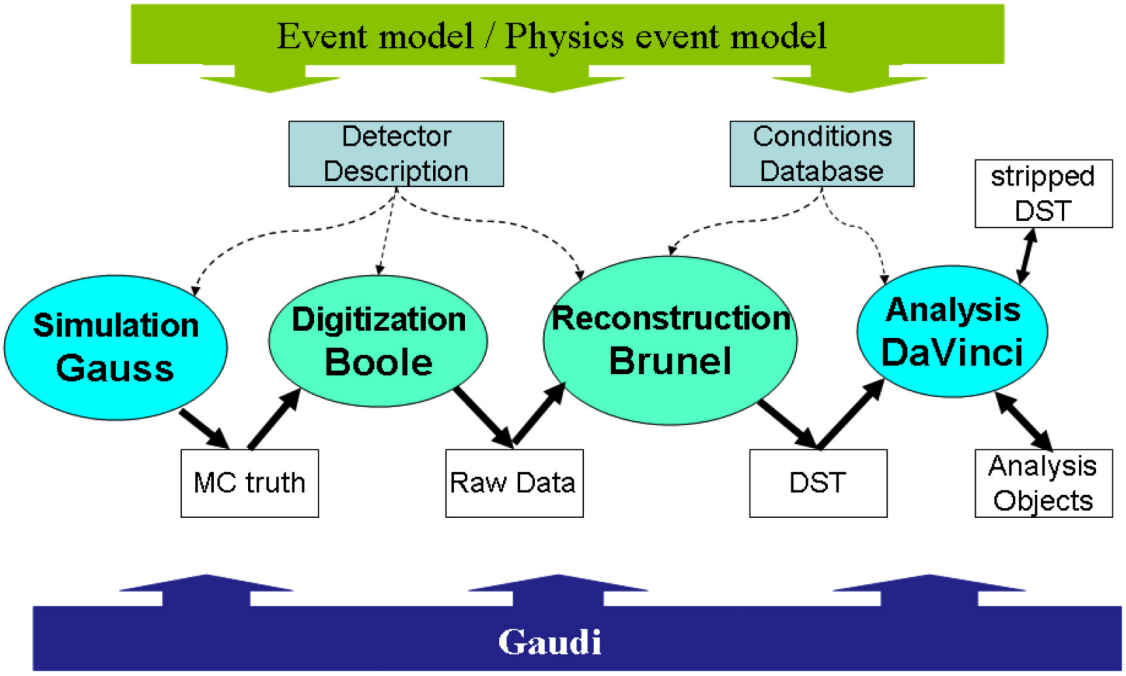


Figure 2-6: The LHCb data processing applications and data flow. Underlying all of the applications is the Gaudi framework and the event model describes the data expected. The arrows represent input/output data.

2.6.1. Gauss, the simulation application

Gauss[21] simulates the behaviour of the spectrometer to allow understanding of the experimental conditions and performance. It integrates two independent phases that can be run together or separately. Normally they are run as a single job. Both phases make use of libraries and toolkits available in the Physics community. A schematic structure of the application phases is shown in Figure 2-7.

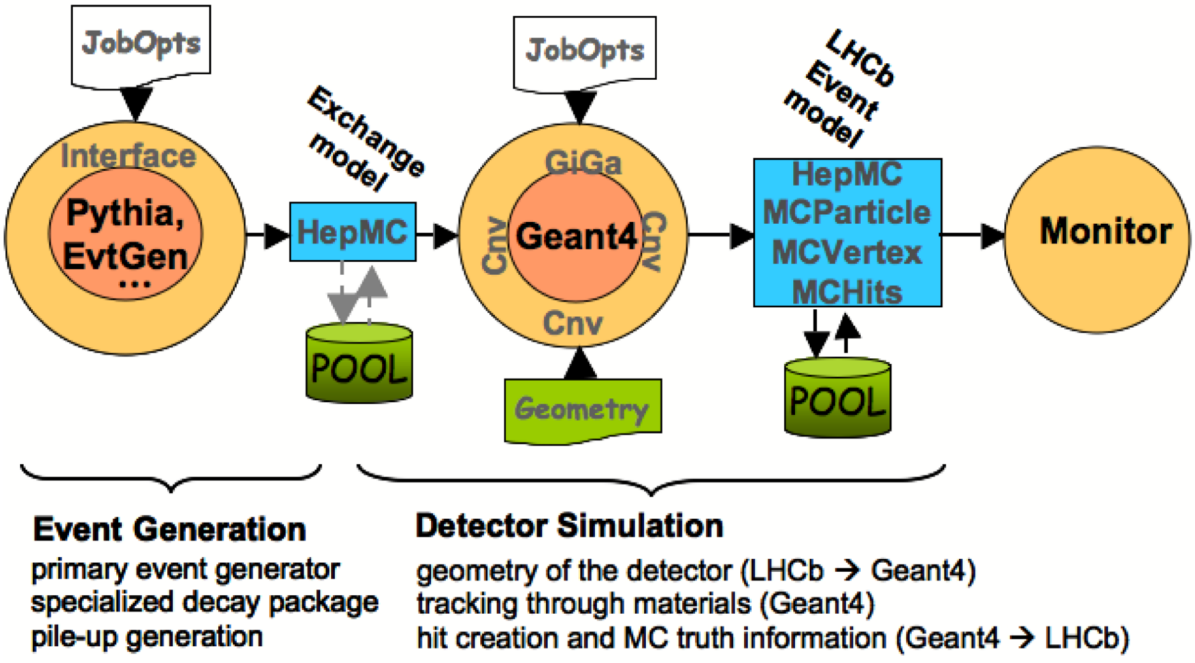


Figure 2-7: Structure of the Gauss application

The first phase consists of the event generation of proton-proton collisions and the decaying of the B-mesons in channels of interest for the LHCb physics program. It is interfaced to Pythia [22] for the event production and to a specialized decay package, EvtGen [23]. Pythia settings were tuned to reproduce the multiplicities at lower energies [24]. EvtGen is a specialized package for B-decays originally designed for the BaBar collaboration to accurately model decays of B^o and B^+ hadrons. A modification was necessary for LHCb to handle incoherent B^o and B^o_s production in contrast to the coherent production at the B-factories. Some B^o decay models provided by EvtGen were extended for B^o_s and decays of excited B-mesons were added to the decay tables. EvtGen with the modifications introduced by LHCb provides the starting point to the EvtGenLHC version now distributed by the LCG Application Area Generator Project [25].

The generator phase of Gauss also handles the simulation of the running conditions, the smearing of the interaction region due to the transverse and longitudinal sizes of the proton bunches and the change of luminosity during a fill due to the finite beam lifetime. Single and multiple pp-collisions are produced according to the chosen running luminosity. Other event generator engines can be interfaced in this phase if required. The implementation of the machine backgrounds is in progress: they can be generated separately or added to physics events with the appropriate weight. The particles produced in the generator phase are stored in the HepMC [26] generic format and can be made persistent if this phase is run in *stand-alone* mode as indicated in Figure 2-7.

The second phase of Gauss consists of the tracking in the LHCb detector of the particles produced by the generator phase. The simulation of the physics processes, which the particles undergo when travelling through the experimental setup, is delegated to the Geant4 toolkit [27]. Geant4 interacts with Gauss using a set of interfaces and converters encapsulated in a Gaudi specialized framework (GiGa [28]). GiGa allows the conversion of the LHCb detector geometry into the Geant4 geometry. It also converts the output of the first phase of Gauss to the Geant4 input format. The output of Geant4 in the form of hits produced in the sensitive detectors as well as the Monte Carlo truth history is then converted back into the LHCb event model. The behaviour of the Geant4 simulation engine in terms of detectors to simulate, physics models to use, details of the Monte Carlo truth to be provided, is controlled at run time via job options configuration.

The geometry description is taken from a specific version of the XML geometry database as specified in the job options. For physics performance studies particular care has been taken to describe the detectors and supports in the LHCb acceptance. Details of the infrastructure are to be added for special studies (e.g. study of radiation in the cavern). An example of the details with which the VELO is described is shown in Figure 2-8.

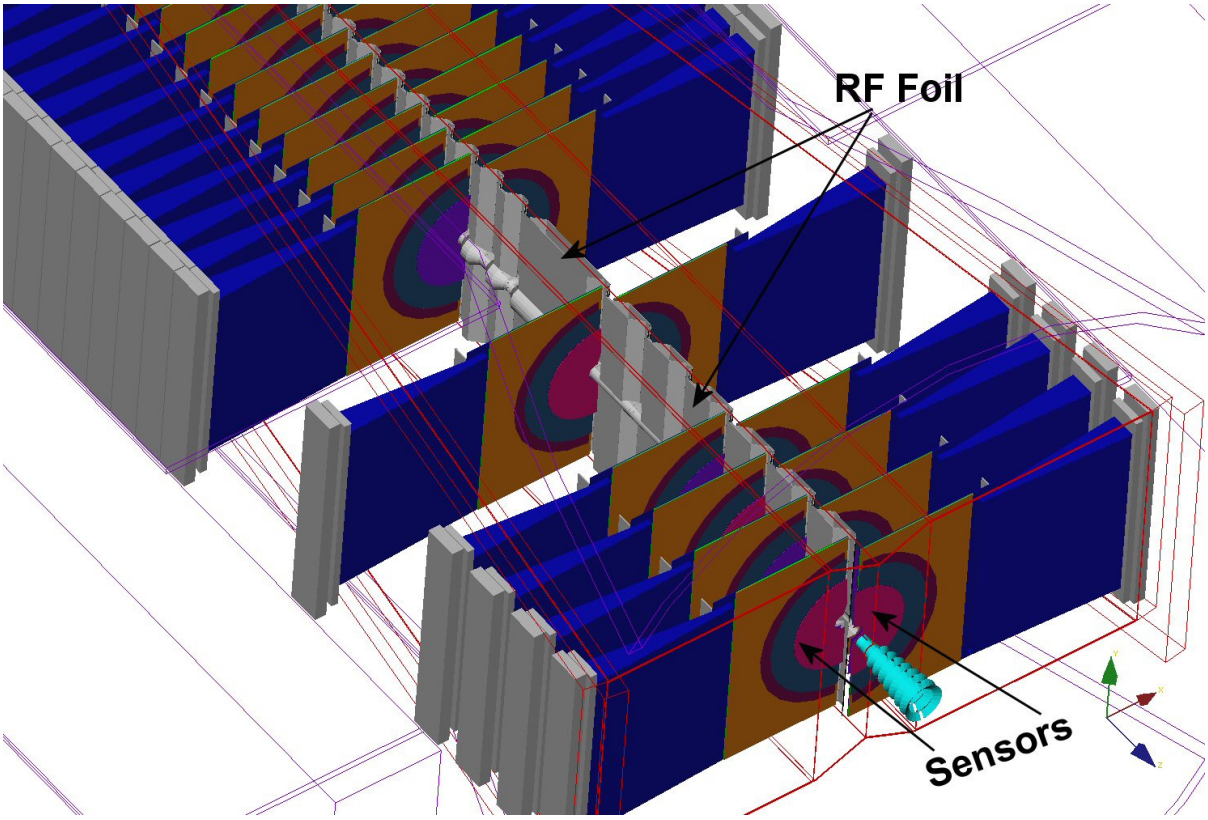


Figure 2-8: Geometry description of the Vertex Locator (VELO)

Gauss replaced the previous FORTRAN-based simulation application at the end of 2003. The Geant4 simulation was adapted to take care of LHCb specialities. For example, the energy deposition in individual Electromagnetic calorimeter cells is corrected for saturation effects according to Birk's law with parameters taken from [29] and the full Electromagnetic calorimeter simulation is tuned with test beam data [30]. Another feature of the LHCb spectrometer is the identification of particles with RICH detectors and the use of an Aerogel radiator for the low energy range. Physics processes specific to the RICHes, e.g. Cerenkov emission and Rayleigh scattering, were studied and validated in the simulation with comparison with test beam data [31]. The simulation of photoelectron productions in the HPDs was implemented in Gauss as a Geant4 user physics process to handle directly Quantum Efficiency values. The details of the response of the tracking detectors are handled in Boole (section 2.6.2).

For MC Particles and MC Vertices, only the necessary information is stored which is needed to find out the truth for the main spectrometer. For the calorimeters, we do not store the full shower history, though it is possible to do it for special studies. In addition it is possible to store more detailed information for example the full Cerenkov and photoelectron history if necessary so as to produce debugging information for the RICH reconstruction (Figure 2-9).

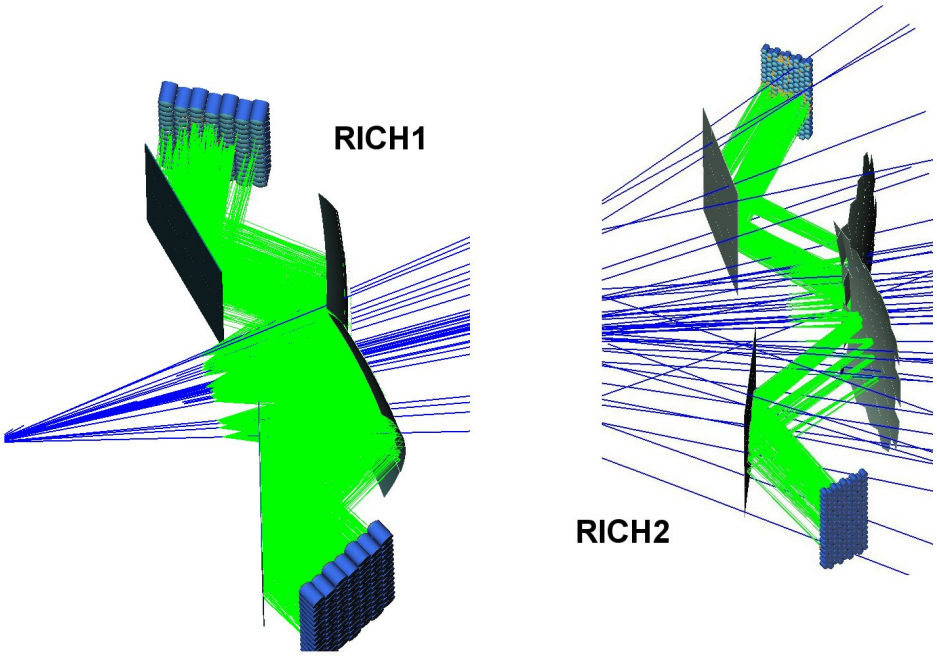


Figure 2-9: Detailed RICH simulation showing the charged particles and the tracing of the emitted Cerenkov photons via the mirrors up to the photon detectors.

After validating Gauss both with test beam data and by comparison with its predecessor, it is now used for massive production of data (DC04 data challenge). The Gauss CPU performance for the version of the code used in DC04 varies depending on the complexity of the events and ranges from ~20 kSI2k sec/event for minimum bias to ~50 kSI2k sec/events for signal events (using gcc 3.2 -O2). Improvements to the simulation both in terms of the application itself, additional details in the descriptions of the detector, new features (e.g. new physics or background generators) are foreseen and are being continuously implemented.

2.6.2. Boole, the digitization application

The Boole digitization program is the final stage of the LHCb detector simulation. Boole applies the detector response to "hits" previously generated in sensitive detectors by the Gauss simulation program. The digitization step includes simulation of the detector response and of the read-out electronics, as well as of the L0 trigger hardware. The output has the same format as the real data coming from the detector.

The program is normally configured to read events from two distinct Gauss files: an *in-time* file containing simulated events for the channel under study (typically these are specific decay channels, or a generic mixture of B events, or a minimum bias mixture of events), and a *spillover* file containing an independent mix of minimum bias events. When initializing the processing of an event, Boole uses the instantaneous luminosity with which the *in-time* event was generated to determine the probability of one or more interactions occurring in the two preceding (-25ns, -50ns) and one following (+25ns) beam crossings. A random number is used to populate these additional beam crossings with events from the *spillover* file according to this probability; these events are then used by the digitization algorithms to simulate *spillover* into the electronics from the out of time signals of other beam crossings.

The program then executes a sequence of algorithms from each of the sub-detectors, to simulate the sub-detector and electronics response including simulation of imperfections such as noise, cross-talk and dead channels. These simulations are continuously improved with the evolving knowledge acquired from test beam data [32][33][34][35]. The output of this phase is a series of *digits* corresponding to the output of the front end electronics which are then fed

into a simulation of the readout partitioning and of the on-line zero suppression and clustering software. This results in a series of *banks* whose format and partitioning is identical to that of the banks injected into the Data Acquisition System (DAQ) by the readout electronics [36]. Both the L1 (readout after L0YES) and Raw (readout after L1YES) bank formats are simulated. The event building stage of the DAQ is simulated by concatenating these banks. The resultant L1 Buffer and Raw Buffer objects have the same format as the event buffers that will be seen respectively by the Level 1 Trigger (L1) and High Level Trigger (HLT) applications running in the on-line event filter farm [37] .

The L0 trigger simulation is then executed, taking the Raw Buffer as input. It is important to prove that the L0 trigger can be simulated in this way, as this will ensure that the Raw Buffer contains sufficient information to reproduce the L0 trigger decision off-line in the real experiment. The result of the simulation is appended to the L1 Buffer and Raw Buffer, in the format expected in the real data.

The L1 Buffer and Raw Buffer are the main output of the Boole application. It is important however to preserve also information about the Monte Carlo truth, e.g. which set of Gauss hits gave rise to a given digit, or whether a digit came from a noise hit or a spillover event. Since the L1 Buffer and Raw Buffer mimic the real data, they cannot contain explicit references to the MC Hits. Instead, an association is made between a given *channelID* (the identifier of an electronics channel encoded in the L1 and Raw buffers) and the corresponding Monte Carlo truth, as described in section 2.3. Sufficient information is stored on the Boole output to allow navigation to the MC hits of the *in-time* event. Any digits that cannot be associated in this way are then due to noise hits or to spillover.

A monitoring phase is available in Boole. In normal production most histograms and printout are turned off. They can be selectively turned on to study the performance of the digitization in specific sub-detectors, in particular to verify new versions of the program against reference output produced with large statistics by a well-tested version.

The event output of Boole can be customized according to the requirements of the analysis. Two types of output are currently possible: full digitization (“Digi” output: L1 Buffer and Raw Buffer plus full MC truth history) and raw data simulation (“L1” and/or “Raw” output: L1 Buffer and/or Raw Buffer only). For both types it is possible to output all events, or just events selected by the L0-trigger. The average event sizes (on disk, after Root compression) are shown in Table 2-1.

Table 2-1: Event sizes of the Boole output.

Event size (kB)	Digi	L1 Buffer	Raw Buffer
Minimum bias events	340	2.2	15.4
L0 selected		3.6	26.3
L0 and L1 selected		4.4	31.7

Note that the numbers given for the Digi format are for unpacked data – a reduction factor of 2-3 should be achievable using the techniques described for the rDST in section 2.6.3. The L1Buffer and RawBuffer include all overheads due to the formatting and partitioning of the data, and conservative estimates of the overheads due to electronics noise. The zero suppression thresholds and the encoding of the subdetectors is still in the phase of being optimized. The aim is to reach 25kbytes per triggered event [38]. The breakdown of the RawBuffer size in memory per sub-detector is shown in Table 2-2. The data compression of ROOT saves about 30% of disk space for raw data.

Table 2-2: Raw data sizes for L0 selected events.

Sub-detector	Raw data size (kB)
Velo	10.4
TT	4.6
Rich	5.3
IT	3.8
OT	7.6
CALO	8.2
Muon	0.9
L0PU	0.3
L0Calo	0.5
Total	41.6

Boole has been used in production in the DC04 data challenge. It processed over 200 million events, with less than 1000 crashes, the vast majority of which were due to a single well identified problem that has since been fixed. The processing time for a signal (minimum bias) event is 0.6 kSI2k.s (0.36 kSI2k.s) using gcc 3.2.3 compiler with –O2 optimization. Memory usage is stable, approximately 300 MB.

2.6.3. Brunel, the reconstruction application

Brunel is the LHCb reconstruction application. It takes as input the Raw Buffer object described in the previous section, from which it produces an rDST (for use in the application for production analysis, stripping see section 4.2.4) or a complete DST (for use in analysis with the *DaVinci* application).

Because it starts from Raw Buffer, Brunel can process identically real data coming from the DAQ or the simulated data resulting from the Boole digitization and as such is independent from simulation. It is intended to run the same application in the on-line event filter farm, for the rDST production, and for full reconstruction of both real and simulated data.

Brunel is organized as a series of independent processing *phases*. In particular, all access to Monte Carlo truth is confined to a dedicated phase that can be switched off when processing real data. This guarantees that exactly the same algorithms will be run on both real and simulated data, and that the reconstruction will not break in the absence of Monte Carlo truth. In normal running mode, the program reads in the same detailed detector geometry and material description as used in the simulation, thus ensuring consistency between the simulated geometry and the geometry used for reconstruction. In addition, a misaligned geometry different from the simulation can be read in for the reconstruction step for use in alignment studies. In both cases, alignment corrections, as measured by the alignment procedure or the detector survey, will be read from the conditions database and applied to the basic geometry description.

The reconstruction phase is completely independent of Monte Carlo truth information. The detailed description of the reconstruction algorithms that was given in [2] is still relevant, although many algorithms have evolved since then, and will continue to do so in the coming years. Here we give only a very brief overview of the program flow. It begins with *clustering* in the tracking detectors. The Raw Buffer information is decoded and off-line clustering algorithms applied to produce the clusters used as input to the tracking pattern recognition. The tracking pattern recognition proceeds in several steps, each step benefiting from the result of the previous steps, the goal being to provide as complete and precise a set of tracks

as possible, while minimizing the number of ghosts. The last stage of the tracking is a full Kalman filter track fit taking into account the detailed material description, followed by a clone-killing step that removes tracks that share too many clusters. The resulting *unique* tracks are passed to the Calorimeter, Rich, and Muon detectors for Particle ID reconstruction. The clusters, unique tracks and Particle ID objects are currently all stored on the DST.

The reconstruction phase is followed by a (Monte Carlo specific) *Relations* phase in which algorithms navigate the event model relationships to associate reconstructed clusters to the MC Particles that gave rise to the hits from which the clusters were built. If all (or more than a predefined fraction of) the clusters that make up a track come from the same MC Particle, the track is said to be associated to that MC Particle, and otherwise it is considered a *ghost*. If more than one track is associated to the same MC Particle, the tracks are classified as *clones*. The association tables between clusters and MC Particles and between tracks and MC Particles are stored on the DST. All other intermediate truth information is dropped, it can only be retrieved by re-accessing the Boole output file – this can be done transparently by the Gaudi framework following the smart references present in the data, but has the additional overhead of a file catalogue lookup and staging of a separate file. This functionality is only needed for detailed debugging of the simulation and reconstruction algorithms and therefore, Boole output files are only kept for a small subset of the data produced.

The event loop finishes with a monitoring phase. Currently much of the monitoring relies on the existence of MC truth information; it is foreseen to split this phase into two, one of which would be independent of the MC truth and which could be executed also on real data. As with Boole, histograms and printout can be selectively switched on to study the performance of the reconstruction in specific sub-systems, and to verify new versions of the program against reference output produced with large statistics by a well tested version.

It is foreseen that, in the on-line environment, Brunel will run in the same application as the HLT. The HLT will be executed as the first phase of the application, with the reconstruction following only for events selected by the HLT exclusive selection. In the current implementation the HLT and reconstruction phases are completely independent of each other, both starting afresh from the Raw Buffer. A more integrated approach, currently under investigation, may be beneficial (for example sharing the decoding of the Raw Buffer information, or using the results of the HLT pattern recognition as a starting point for the full reconstruction).

Most of the currently implemented reconstruction algorithms assume a perfectly aligned detector. In future it will be necessary to tune the algorithms to deal with a detector whose alignment is not perfectly known. In addition, alignment and calibration data will be time-dependent and may differ from event to event. Work is in progress to understand how such *conditions* data will be made available to the algorithms in a consistent way, both in the on-line and off-line (including grid) environments. The development and deployment of a conditions database framework is being carried out in close collaboration with the LCG (see section 2.4.)

The event output of Brunel can be customized according to the production environment. Two types of output are currently foreseen: complete DST for end-user analysis and reduced DST (rDST) for input to the event stripping production step. For both types it will possible to output all events, or just events selected by a selection decision such at the HLT exclusive selection. The average event sizes (on disk, after ROOT compression) are shown in Table 2-3.

Table 2-3: Event sizes of the Brunel output.

Event size (kB)	rDST	DST		
		Raw+L1 Buffer	Reco data	MC Truth
L0+L1 selected events		36.1	198.8	305.5
B signal	7.7	29.1	163.3	255.6

The number given for the rDST format is for packed data, in which all data are packed into 32-bit fields, and where the persistent classes have been optimized to take full advantage of the ROOT serialization and compression mechanisms. This procedure reduces the size of the data required for the stripping step from 22 kB to 7.7 kB. It has been shown that similar reduction factors can be achieved also for the full DST data (and in particular the MC Truth) by applying similar techniques.

The processing time per signal (minimum bias) event is 2.8 kSI2k.s (0.8 kSI2k.s) using gcc 3.2.3 compiler with –O2 optimization, plus 0.6 kSI2k.s (0.3 kSI2k.s) for the MC truth monitoring. The design goal is 2.4 kSI2k.s per event for real data. Table 2-4 shows the time for the different phases and the major contributions to the reconstruction time for signal events.

Table 2-4 Execution times of the major algorithms.

	Time per event (ms) on 1Ghz PIII
Initialization	5
Reconstruction (total)	6981
Tracking (total)	4372
Pattern Recognition	2316
Track Fit	1954
Rich	2431
Relations	595
Monitoring	893

The tracking pattern recognition code is an old implementation dating from 2003. A complete re-implementation is foreseen, integrating many newer, faster, algorithms that have been developed recently. The time taken for the track fit can be reduced by optimizing the use of the detector geometry to determine the material distribution for the Kalman filter (transport service). Ideas for reducing the Rich processing time are under investigation in the context of developments for the HLT.

Brunel has been used in production in the DC03 and DC04 data challenges. In DC04 it processed over 200 million events, with less than 200 crashes, the vast majority of which were due to a single well identified problem that has since been fixed. Memory usage is stable, approximately 350 MB.

2.6.4. Gaudi application executing in the on-line environment

The structure of the Gaudi architecture, with *algorithms* never communicating directly with permanent data storage, makes it also well suited for on-line applications where data comes from the DAQ. Only the Input Service to the Transient Store, the job control and the monitoring components need to be specialized to interface with the DAQ and with the Experiment Control System, while other components can be used identically as in off-line applications. On-line applications differ slightly from off-line and batch oriented applications:

- On-line applications receive the event data from the subfarm controller node (SFC)

- The application must provide run-time monitoring information and
- They must allow for external control.

The use of Gaudi online will be fully tested in the 2006 real time trigger challenge.

Access to event data

The applications executed on-line are trigger applications, L1 and high level triggers, or calibration applications. The typical processing scheme of the trigger applications is shown in Figure 2-10. The difference to off-line like data processing is the interaction with the subfarm controller node (SFC) to which the resulting trigger decision must be reported.

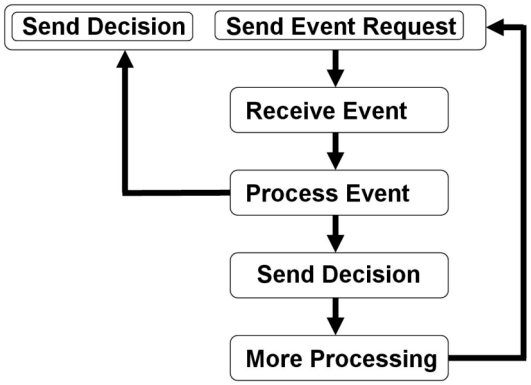


Figure 2-10: The logical processing scheme for trigger applications

Whereas the physics code executed in on-line applications does not need to be aware of the processing environment, the service responsible to bootstrap the access to the event data collected in the experiment requires a customized implementation.

The processing scheme implemented in the Gaudi event loop service to analyse events in the on-line environment includes the following actions:

- The event loop service requests a new event from the *EventSelector*. This *EventSelector* requires a custom implementation to interact with the SFC.
- The *EventSelector* issues a request to the SFC to receive event data and waits until the data have arrived. It encapsulates the received data and passes the data buffer to the event loop service.
- The event loop service bootstraps the transient event data store by passing the data buffer.
- The physics algorithms then pick-up the digitized data from the event store using the standard access mechanism and compute the trigger decision, which itself is stored in the transient data store.
- A specialized algorithm executed last inspects the trigger decision and forwards the result to the SFC.

For a small fraction of the events it is foreseen to optionally monitor the trigger decisions. Hence, before a new event is requested, a sequence of monitoring algorithms may be executed after sending the trigger decision.

Gaucho monitoring and application control

Gaucho (GAUdi Component Helping Online) allows the control and monitoring by the Experiments' Control System (ECS) of the L1 and HLT algorithms written in the Gaudi framework running on the event filter farm.

Gaucho provides a lightweight C++ DIM[39] (Distributed Information Manager, a communication system for mixed environments) library to be used by the algorithms and a library of PVSS [40] (the Process Visualization and Control System used by the ECS) scripts and panels to enable integration into the ECS. Algorithms can use a Monitoring Service that is part of the Gaudi Online project to publish variables, counters and histograms. The published information is passed to PVSS via DIM where they are displayed in real time.

Gaucho steers the Gaudi Application Manager through a Gaudi DIM Controller that calls the Application Manager and takes control of the event loop. Commands can be sent from PVSS via DIM to the Gaudi DIM Controller to configure, start, pause and stop algorithms. Once an algorithm has been configured and started, commands can be sent to it (from PVSS) to explore, publish and visualize histograms on the transient store. Properties of the algorithm can be read and reset. Some real time analysis can be done in PVSS such as adding histograms, calculation of mean quantities and displaying their evolution in real time.

The communication via DIM presents a low CPU load and interferes as little as possible with the event data processing by the algorithms. It generates low network traffic, is scalable with the number of nodes and is compatible with farm partitioning.

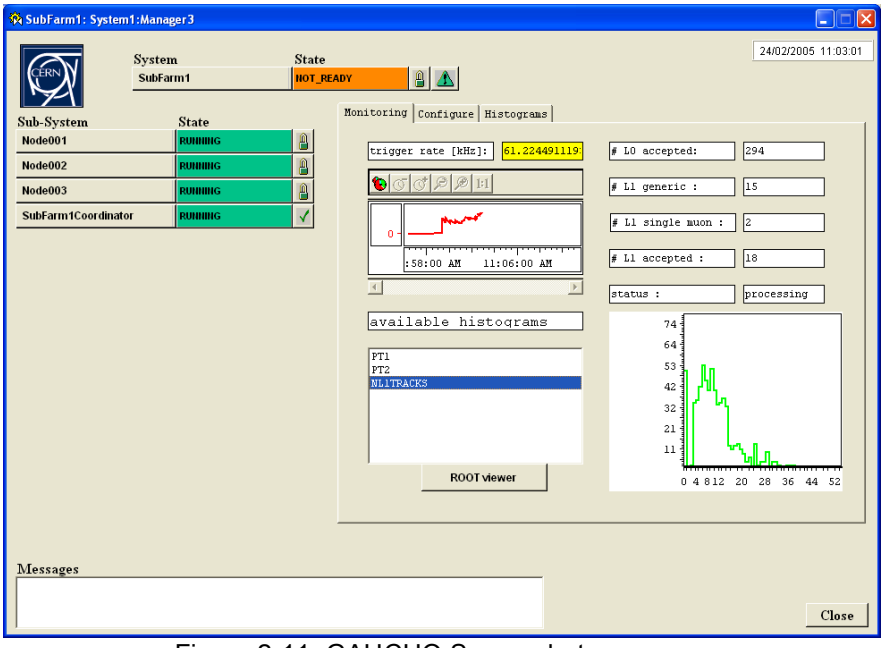


Figure 2-11: GAUCHO Screenshot

2.6.5. L1/HLT, the on-line trigger applications

The L1 and HLT [3][41] applications are running in parallel on every CPU of the event filter farm [42], with the L1 applications having priority. The L1 application receives only the input from the VELO, TT and the L0 subsystems with a rate of about 0.7 kHz/CPU. After a positive L1 decision, the data of all sub-detectors are sent to the event filter farm for processing by the HLT with a rate of about 30 Hz/CPU.

The L1 trigger selection is based on the p_T of reconstructed tracks that have a large impact parameter to the primary vertex, the highest di-muon invariant mass from L0, and the highest L0 photon and electron candidate E_T.

The HLT execution happens in two steps (Figure 2-12). The first step is the HLT generic selection that re-does the L1 decision but with better momentum resolution of the particles, as the tracking uses the information from T1-3 stations. In the generic part the event is partially reconstructed: approximately 1/3 of the total tracks, those that have a large impact parameter and all possible muons. The output rate of the HLT-generic is 12 kHz; out of which 900 Hz correspond to "b-inclusive" events with a muon of high impact parameter and large transverse momentum ($b \rightarrow \mu$); and 600 Hz of events with large di-muon masses, used mainly for systematic studies. These 1.5 kHz of events are directly marked for storage. In the second step, the HLT-specific, all the events that passed the generic algorithm are fully reconstructed and filtered by the exclusive selections. In the initial part of the exclusive selection, intermediate particles created in a standard way from 2 or 3 tracks vertices are combined to form B-decay candidates. In the second part, specific cuts are applied to each selection variable such as invariant masses, impact parameter significance, quality of vertices, etc. The use of the RICH in the HLT is currently under study. The final output rate of the specific selection is 200 Hz, with an additional branch of 300 Hz of D* candidates, to be used for calibration and systematic studies and also for charm physics. The final output rate of the HLT is 2 kHz.

L1 and HLT applications share the same event data classes and algorithms; therefore trigger algorithms are interchangeable between both applications. In order to have a flexible code, the trigger is a sequencer of small algorithms. The algorithms can alternate doing reconstruction or trigger decision. Only the necessary data is processed at each trigger level, for example, in the HLT generic selection, a fraction of the tracks are reconstructed, and only if the event passes the generic selection the rest of the tracks are reconstructed. Therefore the reconstruction algorithms need to be able to perform a "partial" reconstruction on demand. In order to save time, a Gaudi tool is used to manage the memory allocation of the data classes (the tool will be replaced in the future by a Gaudi Data Service). This tool creates a finite number of objects at the beginning of the run to store clusters, tracks and particles. These data objects are then filled and cleared per event, avoiding the time used for memory management in their creation. In a similar way, to avoid losing extra time accessing the detector and calibration data, a table is filled with the relevant information at the initialization of the run, and simple and fast access is provided to this information to the different algorithms.

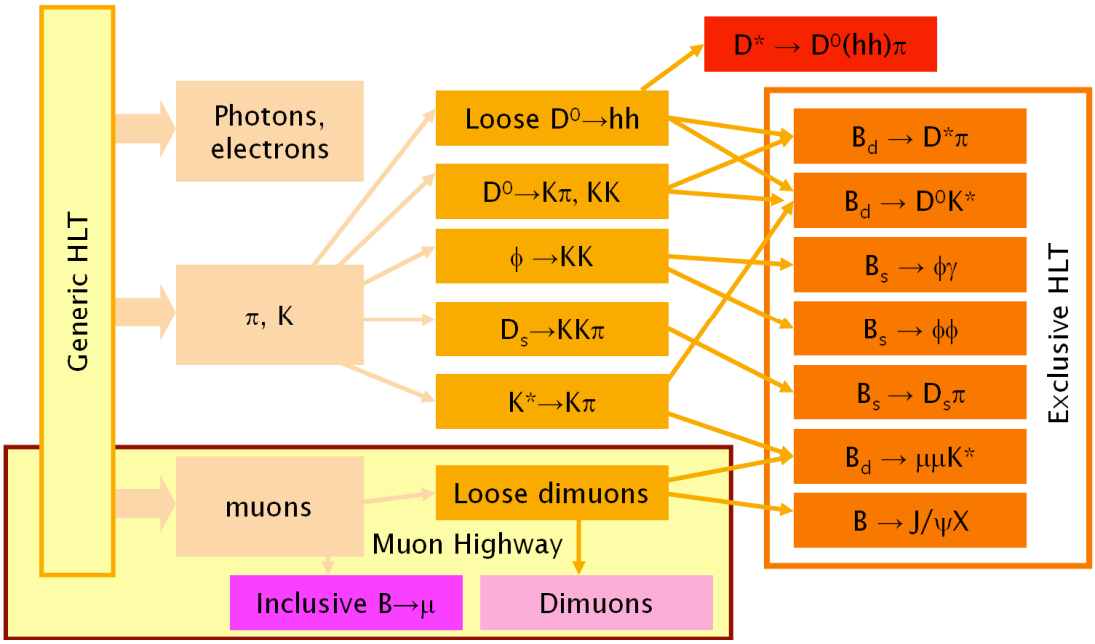


Figure 2-12: Schematic of the high level trigger for benchmark channels

The software trigger algorithms are currently tested and improved in the DaVinci application (see section 2.6.6) to assess their performance [43][44][45]. The main priority is to increase the reconstruction efficiency without losing in speed, and making the code more flexible and robust against misalignments and detector inefficiencies. The full sequence of HLT takes at the moment about 20 SI2k.s (Table 2-5) compared to the budget of 27 kSI2k.s foreseen in 2007.

Table 2-5: HLT CPU needs

	SI2k.sec	
	CPU needs per algorithm	average number of calls per event
VELO Tracking	2.8	1.0
Generic HLT	7.6	1.0
Rest of tracking	6	0.33
PID (mainly RICH)	14	0.33
Shared resonances	4.8	0.33
D* stream	0.4	0.33
Exclusive stream	3.6	0.33
Total		20 SI2k. sec

2.6.6. DaVinci, the analysis framework

The analysis framework supports selection of events and analysis proceeding from the further processing of the DST or rDST data. The output of DaVinci can be purely statistical or event data. Analysis Object Data (or Ntuples) files containing physics objects can be written to allow further processing. The output of DaVinci can also be a reduced DST, where only events satisfying certain conditions are written.

A minimal DaVinci job includes:

- the reconstruction of primary vertices,

- the assignment of one or several particle ID hypotheses to tracks and calorimeter clusters by using the PID information from the RICH, calorimeters, muon detector and VELO,
- and a sequence of selection algorithms.

A dedicated algorithm base-class inheriting from the Gaudi classes is provided for physics selections algorithms. It hides all interactions with the *transient event* (TES) and *histogram stores* from the user, and interfaces several commonly used tools.

Physics analysis tools manipulate physics event objects that are described in terms of "particles" and "vertices". They contain for instance a set of vertex fitters, particle extrapolators or filters (for selection cuts). Wherever possible, tools performing similar tasks inherit from the same abstract interface, to allow the user an easy switching from basic to more sophisticated tools without having to rewrite any code. A typical example is the set of unconstrained and mass or geometry-constrained vertex fitters. DaVinci also contains a standard algorithm that attempts to assign a flavour tag to each reconstructed B meson.

All these physics tools and algorithms are designed to be able to handle both off-line and on-line data in a transparent way, allowing running the same algorithms as offline in the HLT environment [46]. When necessary, a fast but approximate replacement tool is provided for the HLT needs.

DaVinci also provides a set of "generic" *algorithms* allowing performing repetitive tasks, like the reconstruction of a decay to *n* decay products, involving a vertex fit and some selection cuts [47]. These standard algorithms are used both in the HLT and in the stripping stage, ensuring that all successive steps of the selection (HLT, stripping, final physics analysis) of a decay of interest are highly correlated and well understood.

Selection algorithms can be integrated into a complete, dynamically configured DaVinci selection job. This job has been successfully used in the stripping stage of the DC04 data challenge. The DC04 stripping code includes 43 selections and 25 final selections. It consumes an average of 0.65 kSI2k.s per inclusive bb event. The pre-selections have not yet been optimized for speed.

To help the physicists understanding their selection, DaVinci also provides a set of tools and algorithms accessing MC truth and assessing reconstruction and selection efficiencies [48]. These MC utilities are packaged separately to enforce that no MC truth information is used in the selection phase of the analysis, and that the program can run on data not containing any truth information, like the future real data. In addition, a special toolkit named LoKi is provided to facilitate the coding of physics analysis algorithms. It combines the power of the DaVinci tools with physics oriented semantics.

LoKi, an analysis toolkit

LoKi is a C++ toolkit for Physics Analysis that provides a set of high level analysis utilities with physics oriented semantics. The package has been inspired by the success of the Kal program, used for physics analysis by the ARGUS collaboration, and the Pattern[49] package used by the HERA-B collaboration. The ideas from GCombiner[50], Loki[51] and CLHEP[52] libraries are also used.

The current functionality of the package includes

- Set of predefined *function objects* and generic operations

- Selection and filtering of particles and vertices,
- Multi-particle combinatoric loops
- Simple matching of reconstructed objects with Monte Carlo truth information

There is a clear separation between physics analysis code and technical details.

The majority of complicated physics analysis idioms can be expressed by only one line of LoKi code. It has been demonstrated that usage of LoKi results in a drastic reduction of the number of lines of code. In order to make the end-user code even more compact, the concepts of *Patterns* and implicit loops in the spirit of standard template library (STL) algorithms have been introduced.

LoKi-based analysis code is further enhanced by the concept of *locality*, in which the entities are declared and defined only at the place they are used. The "book-on-demand" treatment of histograms and N-Tuples illustrates this important concept.

There are no raw C++ pointer manipulations and explicit memory management in LoKi-based physics analysis code. This fact together with the suppression of explicit and tedious loops makes the code less prone to errors and easy to debug.

The implementation of LoKi heavily exploits the modern technique of generic template meta-programming [51][53]. In general, LoKi code is very efficient due to the templated nature and the fact that most of the code is in-lined. The kernel components of LoKi are loosely coupled with the LHCb Event Model.

2.7 Interactive Analysis

Being able to perform interactive analysis of LHCb events is not only useful for providing an easy way of learning and using the software but also for debugging and developing the software. The event display allows to visualize and to inspect the detector geometry and the event data itself using a graphical user interface. Choosing Python as a scripting language enables direct access to the objects in the C++ world in a much simpler way than writing C++ code and still being able to perform sophisticated physics analysis. Python scripts are also used to perform complex operations behind the graphical user interface of the event display.

2.7.1. Bender, an interactive physics analysis tool

Bender[54], a Python[55] based physics analysis application, combines the Gaudi software architecture with the flexibility of the Python scripting language and provides end-users with a user-friendly physics analysis environment. Bender is based on the generic Python bindings for the Gaudi framework, called GaudiPython, and on the C++ physics analysis toolkit LoKi (section 2.6.6). LoKi in turn uses Tools and Algorithms developed in C++ in the context of the DaVinci analysis framework. The usage of Python, the AIDA [56] abstract interfaces and standard LCG reflection techniques allow an easy integration of Bender's analysis environment with third party products like interactive event display, visualization and statistical analysis tools, like Panoramix (section 2.7.2), ROOT [8] and HippoDraw [57]. It has been demonstrated that Bender facilitates the writing of extremely compact and easy-to-read self-contained code. Interactivity of Bender provides physicists with the possibility to (re)define the algorithms, parameters and configuration in the process of code development from the interactive program prompt. By delegating the time consuming tasks to the C++ background functions, almost no penalty for using a non-compiled computing language has to be paid.

Any piece of code or configuration file developed for usage with the DaVinci analysis framework could be used with no limitation within the Bender analysis environment. The interactive program prompt provides physicists with the natural bridge to the event display application Panoramix.

2.7.2. Panoramix, the visualization application

The graphical display of detector geometry and event data objects is provided by a dedicated application called Panoramix (see Figure 2-13 and Figure 2-14.) It is based on a set of visualization services and converters providing the graphical representation of the LHCb setup as well as of the data. The event data can be read from files or produced on the fly. An interactive user interface allows the user to choose what to display and how it is visualized. The visualization services are based on the OnX [58] package for interactivity and Open Inventor [59] for the graphics. Python is used as the scripting language to control the GUI and to provide the necessary functionality by wrapping LHCb C++ code. Predefined views have been implemented and are available in the GUI as well as the normal zoom and rotation facilities.

Since Panoramix is based on the Gaudi framework it can work with any of the data processing applications described before allowing not only 3D graphical rendering for geometry verification but also providing aid in the development and understanding of the physics algorithms.

Graphical User Interface

The GUI is organized as one compact GUI panel organized around a document area made with a stack of Inventor viewers. At the left of the document area there is a data tree browsing widget, at the top a menu bar and at the bottom a command typing area. Various dialog panels can be mapped through the menu bar items in order to parameterize and trigger an action, like printing or changing parameters. The menu bar items can execute either complicated C++ functions or Python scripts to define what and how to visualize objects. Using Python scripts has the advantage that they can be changed on the fly, no re-compilation of code is necessary.

Connection to the data framework

The data framework (Gaudi) should be understood as the software which manipulates and connects the event and detector data to facilities like storage, graphic, GUI, scripting.

OnX is the interactivity framework. It allows the connection between the GUI (via an XML description), viewers, scene manager (Inventor), renderer (OpenGL) and scripting. The connection between the data framework and interactivity framework is done through the Gaudi OnX service. The various elements of the LHCb event model have a "representation" which is in general a Gaudi converter for the Inventor technology (a SoConverter). A SoConverter builds from a data instance an Inventor scene graph. When built, the scene graph is sent to OnX to be displayed. A visualization request starts from a scripted GUI callback, then a data conversion for an Inventor request is activated for various pieces of selected data. The usage of abstract interfaces permits the use of various different visualization technologies.

The visualization packages

Various converters are needed to create the graphical representations of the objects in the transient event store, the detector geometry, hits, tracks, particles, etc.

The SoDet package builds the Inventor representation of the detector. It is very generic and offers together with the LHCb XML detector description a very flexible way to enter and view geometry. The SoStat package allows histograms in the Gaudi transient store to be presented. A package is provided to visualize HepMC information in conjunction with the LHCb Geant4 simulation program (Gauss). Dedicated converters for most of the reconstructed objects exist.

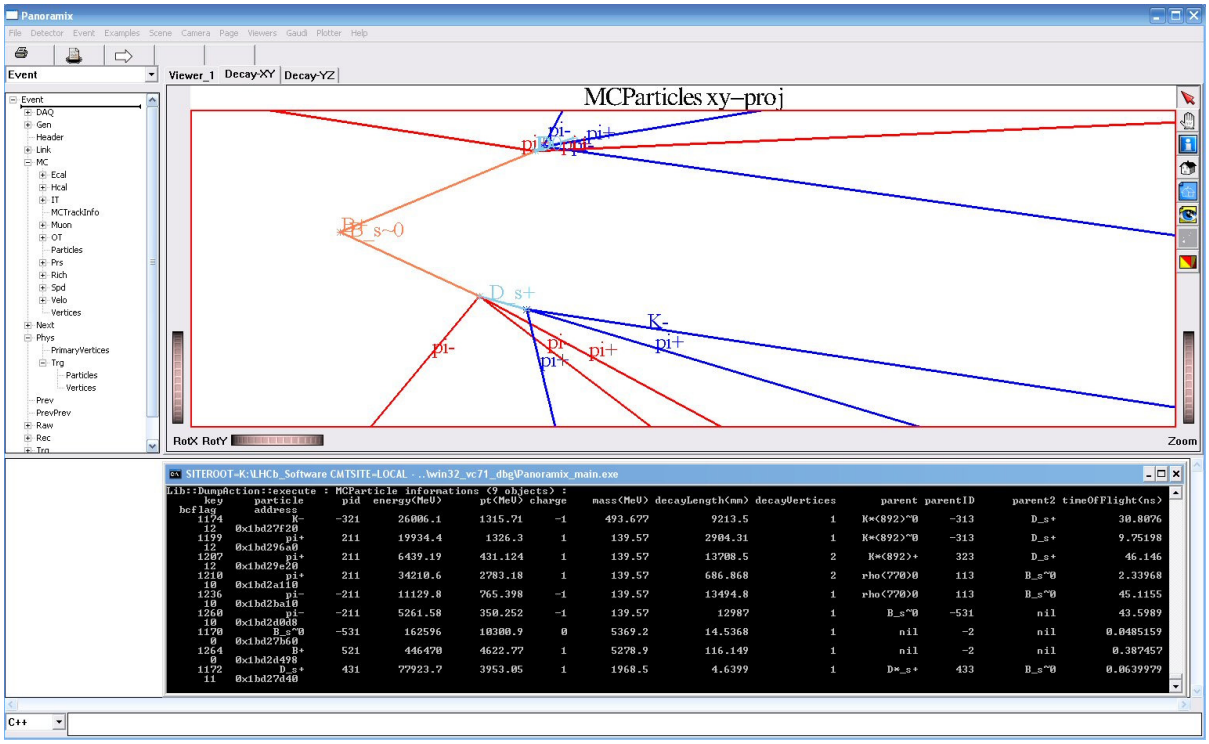


Figure 2-13: Example plot for interactive analysis with Panoramix.

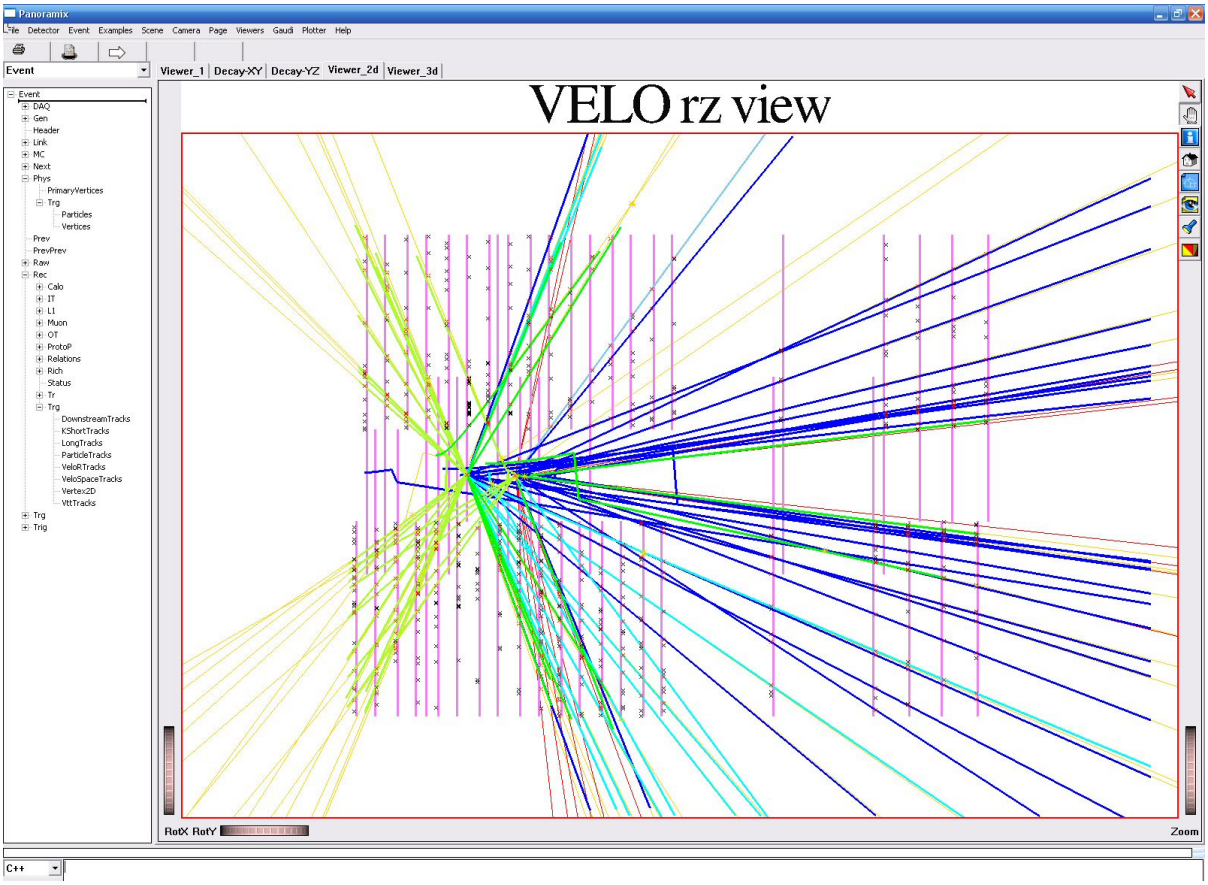


Figure 2-14: Close look at the interaction region showing the reconstructed tracks and their measurements in the Velo overlaid with the original Monte Carlo true tracks.

Chapter 3 Distributed Computing

3.1 Introduction

The resource requirements for computing in LHCb are such that they can only be obtained from a distributed environment. The LHCb Computing Model that justifies this statement is described in the next chapter. We shall describe in this chapter the main activities within LHCb that are related to providing the infrastructure necessary for using this distributed computing.

LHCb will use as much as possible the capabilities provided by the LCG both in terms of computing resources (CPU and storage) and in terms of software components. We expect some basic services to be completely generic and provided by LCG projects and sites while higher level integration and LHCb-specific tools will be provided by the LHCb collaboration.

The developers of the LHCb applications described in the previous chapter as well as physicists developing analysis code use a standard environment and set of tools. These applications also need to be released, packaged, distributed and placed in the appropriate environment before running. This infrastructure is described in section 3.2.

It is expected that LCG will provide a set of baseline services for workload management (job submission and follow-up) and data management (storage, file transfer, etc.) Several higher-level services however are very much experiment-dependent and thus will be provided by LHCb. This is the case for the file and job provenance (Bookkeeping database), for the Workload Management tools (DIRAC) and for the Distributed Analysis tools (GANGA). These high level services are described in the sections 3.3 to 3.5.

The interplay between the LHCb-provided services and the LCG-provided services are outlined in each of the appropriate sections.

3.2 Software environment and distribution

The LHCb software is structured in several *projects*. A *project* consists of a set of *packages* maintained under a unique version number with a well-defined purpose, it can be used by other projects and can use other projects maintained or not by LHCb.

Gaudi as a foundation project uses several LCG projects such as SEAL[12], POOL[9], ROOT[8]. Another project called LHCb is dedicated to handle the LHCb Event Model (see section 2.3), the Detector Description framework and several general use packages on which all applications depend. A set of projects house the actual algorithms that are assembled in order to produce applications:

- Lbcom: contains a few packages shared by Boole, Brunel and DaVinci
- Rec: contains all reconstruction packages
- Phys: contains all physics-related packages
- Online: contains the online services used by applications running on the Event Filter Farm.

On top of these are built all applications projects: Gauss, Boole, Brunel, DaVinci, L1 & HLT and Panoramix. Figure 3-1 shows the interdependency of the LHCb projects.

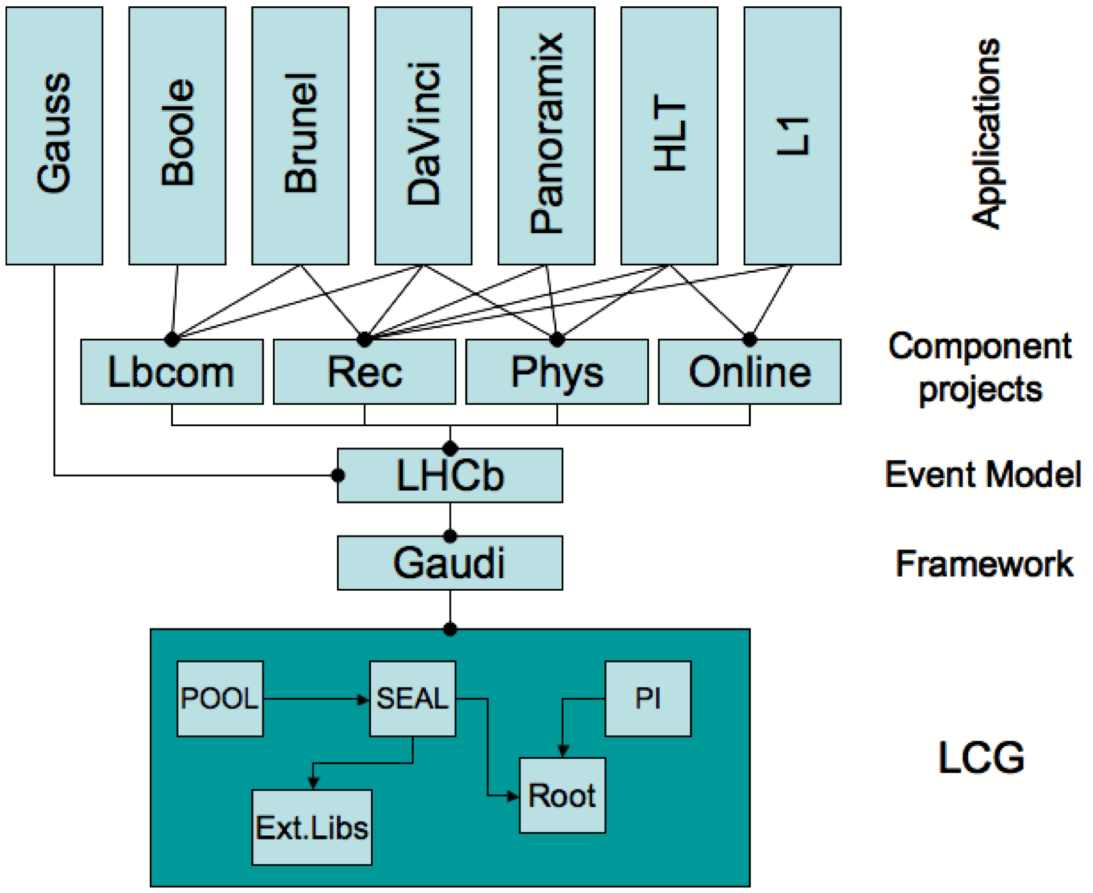


Figure 3-1: The LHCb CMT projects and their dependencies. Also shown are the LCG packages on which LHCb relies

To build any project and to maintain the dependencies between them we use the Configuration Management Tool (CMT) developed and maintained by LAL-Orsay[60]. Relations between packages of a project or between projects as well as parameters required to build libraries, executables, and documentation are described by a CMT meta-language in a requirements file kept within the package. Dependencies and makefiles are automatically rebuilt every time requirements are modified. Specific site/platform/compiler options can be included in requirements files to allow different site/platform/compiler configurations. Similarly dependencies to and within the LCG software use CMT requirements files provided by the LCG-AA project.

The source code is maintained in a CVS repository on centrally maintained servers at CERN.

Builds are made on Linux and Windows platforms starting from Gaudi as a framework, followed by the other projects in reverse order of dependencies. Binaries are kept with the code at CERN on an AFS release area.

We do not produce nightly builds. The frequency of the releases is:

- Gaudi - major release twice a year with minor versions every month or when a new version of LCG software has to be used.
- LHCb event model – releases following Gaudi releases with intermediate versions.
- Component projects and Applications – releases are made when necessary, usually at least once a month.

All projects are built with a unique shell script to book the AFS space, checkout the code, build libraries and executables, create source and binary tar files, dOxygen documentation and web pages. The AFS space occupied by a full release (Linux optimized and debug, Windows debug, dOxygen documentation) is about 9 GB.

The tar files are available from the web. It is possible to download one project/version and all its dependency projects at once with or without binaries. LCG projects tar files are included in the dependencies as well as some compiler run time libraries to be able to run executables on Grid platforms that have a different compiler version. A python script is provided to perform the installation. The use of *pacman[61]* and the developments that were made to use it with CMT is under consideration[62]. The goal is a unique tool that could be used to install the software on any platform: (e.g. laptop or Grid platform.)

3.3 DIRAC

LHCb will have to integrate a coherent system of resources and Grid services to carry out its computing tasks in the distributed environment. Therefore, a project was started which will combine LHCb specific components together with general-purpose components where it proves to be appropriate. This work is being done within the DIRAC project (Distributed Infrastructure with Remote Agents' Control).

DIRAC is conceived as a lightweight system with the following requirements:

- support a rapid development cycle,
- be able to accommodate ever-evolving grid opportunities,
- be easy to deploy on various platforms,
- updates to bring-in bug-fixes and new functionalities should be transparent or possibly automatic.

DIRAC is designed to be highly adaptable to the use of heterogeneous computing resources available to the LHCb Collaboration. These are mainly resource provided by LCG grid. However, other resources provided by sites not participating to the LCG as well as a large number of desktop workstations should be easy to incorporate.

One of the main design goals is the simplicity of installation, configuring and operation of various services. This makes the threshold low for new sites to be incorporated into the system. Once installed and configured, the system should automate most of the management tasks, which allows all the DIRAC resources to be easily managed by a single Production Manager.

The system is designed to be robust and scale well to the computing needs of the LHCb Collaboration. This scale we roughly define for the moment as $\sim 10^4$ concurrent jobs, $\sim 10^5$ jobs in the queue processing, handling $\sim 10^7$ datasets.

3.3.1. DIRAC architecture

DIRAC uses the paradigm of a Services Oriented Architecture (SOA). The services decomposition follows broadly the one proposed by the ARDA LCG/RTAG in 2003 [63] as shown in Figure 3-2.

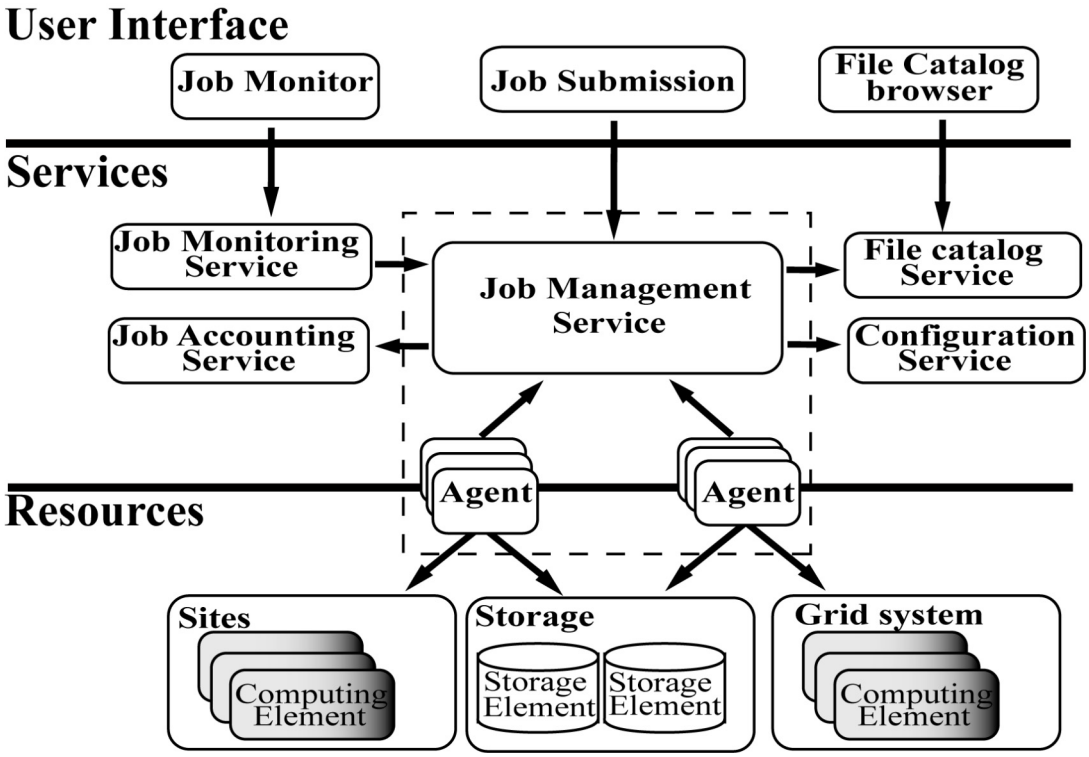


Figure 3-2: General view of the DIRAC architecture. Arrows originate from the initiator of the action (client) to the destination (server.)

The main types of the DIRAC components are Resources, Services and Agents:

- Resources represent Grid Computing and Storage elements and provide access to their capacity and status information.
- Services provide access to the various functionalities of the DIRAC system in a well-controlled way. The users interact with the system via agents.
- Agents are lightweight software components usually running close to the computing and storage resources. These are applications distributed as necessary, which allow the services to carry out their tasks in a distributed computing environment.

The main DIRAC subsystems, Workload Management and Data Management, are combinations of central Services and distributed Agents. This allows an efficient operation of the distributed system with an easy and non-intrusive deployment of its distributed parts. This feature of the DIRAC architecture is essential in the deployment phase of the system. Since the grid environment is intrinsically very dynamic, the efficient deployment is one of the most important characteristics of the system. In the following, the DIRAC services are presented together with an outlook for possible incorporation of components developed within the EGEE/LCG projects.

LHCb considers the approach used for the design of DIRAC as the most suitable for efficient Grid operations. Therefore the necessary infrastructure needed from the LCG to allow the deployment and usage of DIRAC on its supported resources has been requested. The requirements in terms of Baseline Services are summarised in section 5.3.

3.3.2. Computing Resources

The **Computing Element (CE)** in DIRAC is an API abstracting common operations of job manipulation by computing batch systems. It also provides access to the state information of the computing resource such as its capabilities, environment or occupancy. The API is

implemented for various back-end batch systems: PBS, LSF, NQS, BQS, Sun Grid Engine, Condor or standalone PC. One particular case is the access to the LCG grid that is realized as standard DIRAC CE. In the latter case, access to LCG resources both through a Resource Broker scheduling system and direct access to the LCG CE (through GRAM interface) is possible. We are planning also to provide an interface to any standard CE implementation that would be deployed on LCG sites, summarised in section 5.3.3.

The DIRAC **Storage Element (SE)** is a combination of a standard server, like gridftp, and information stored in the DIRAC Configuration Service (see next section) on how to access it. The SE API provides a possibility to dynamically plug-in modules for transport protocols by which the SE is accessible as described in its configuration. Modules for most of the existing protocols are available: `gsiftp, bbftp, sftp, ftp, http, rfio`, direct file access. A special `xml-rpc` protocol allows transfer of relatively small files encapsulated into an XML-RPC message. A variant of the SE compatible with other DIRAC components and accessing standard SRM based storage is being developed (see section 5.3.2.)

3.3.3. Configuration Service

The DIRAC Configuration Service (CS) provides necessary configuration parameters to other services, Agents and Jobs to ensure their collaborative work. The CS contains endpoints of all the DIRAC services and resources, their properties as well as policies for the current production run. This service is vital for the stable operation of the whole system and must be absolutely available despite any outages of the network or server hardware. Therefore special care was taken to provide a reliable and redundant implementation.

The service consists of several servers among which one is a master server accompanied with any number of secondary servers. The master server has interfaces for both updating the configuration information and for serving it to clients. The slave servers have only a read-only interface to provide information to clients. The read-only interfaces of both the master and the slave servers are identical. The secondary servers are periodically updating their copies of the LHCb-wide configuration parameters from the master. Since this information is not changing frequently, the updates are done every minute. The secondary servers are usually running on sites different from the master one in order to eliminate the risk of service interruption due to network cuts.

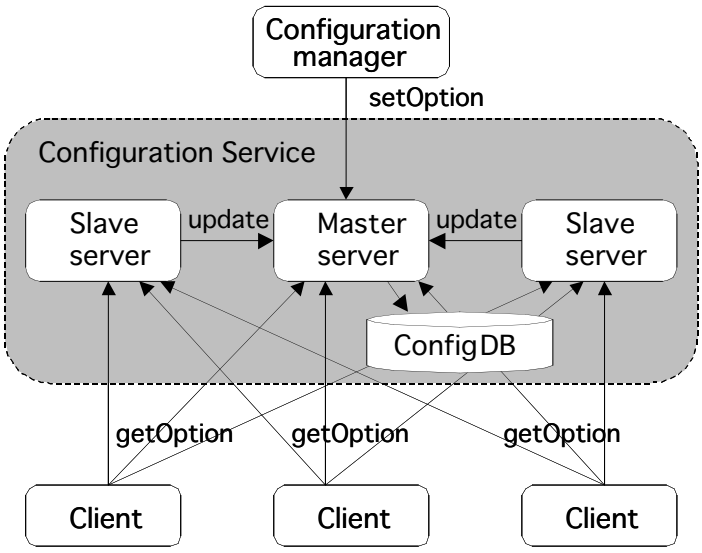


Figure 3-3: Configuration Service architecture. Arrows originate from the initiator of the action (client) to the destination (server.)

Each client has a list of configuration servers to talk to. If one server is not available, the same information can still be obtained from any other server in the list. This ensures the redundancy necessary for the service reliability, as well as for high query rate capacity. The client API provides also the possibility to define configuration parameters in any number of local configuration files. These local settings override the globally provided default values. The whole parameter space is divided in sections providing a single level hierarchical structure. The configuration files are following the syntax of Microsoft .ini files.

3.3.4. Monitoring and Accounting

The Job Monitoring Service receives status information about the running jobs and provides it to the requests of users, for example through a dedicated Web Portal, or to other services. The monitoring information is currently kept within the DIRAC central WMS jobs database. We are considering interfacing the DIRAC agents to the MonaLisa monitoring system [64].

The Accounting Service accumulates statistics on the usage of the computing resources and generates reports that can be used to follow the production progress or to apply policies and quotas while job scheduling.

3.3.5. Workload Management System

The Workload Management System (WMS) consists of three main components: a central Job Management Service (JMS), distributed Agents running close to DIRAC Computing Elements and Job Wrappers which are encapsulating the user job applications.

The JMS is in turn a set of services, highlighted in Figure 3-4:

- Job Receiver Service: provides an interface for users to submit jobs.
- Optimisers: based on the job description provided in standard Job Definition Language (JDL), they sort jobs in task queues
- Matchmaker Service: receives job requests from the job-agents.
- Monitoring Service (not shown): serving job status information.

Agents continuously check the availability of resources in their respective CE, make requests to the matchmaker Service of the central JMS, pull jobs from the JMS and steer job execution on the local computing resource.

Job Wrappers prepare the job execution on the Worker Node, get the job's input sandbox, send job status information to the JMS, and upload the job output sandbox.

The jobs requirements are described using the JDL and their matching to the capabilities of the computing resources is done with the ClassAd library from the Condor project [65].

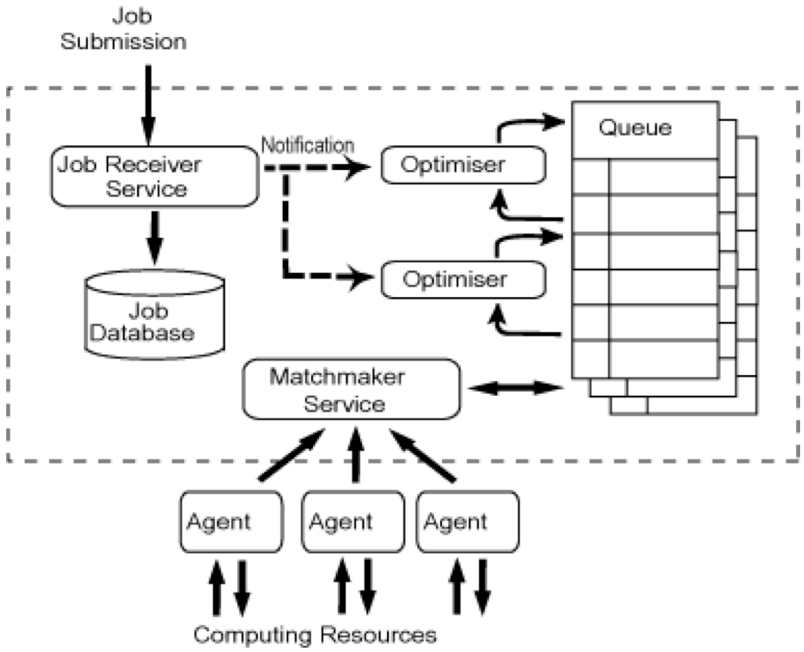


Figure 3-4: The DIRAC Job Management Service architecture. Arrows originate from the initiator of the action (client) to the destination (server.)

One interesting feature of the WMS is that services or users can communicate with Agents and Job Wrappers by means of an Instant Messaging (IM) protocol. In particular, the Jabber/XMPP protocol [66] is used in DIRAC. It provides a reliable asynchronous bidirectional communication channel that can be used to monitor Agents and Jobs or even maintain interactive sessions with running jobs.

3.3.6. Data Management System

The Data Management System (DMS) includes File Catalog Services, which keep track of available data sets and their replicas, as well as tools for data access and replication.

File Catalogs. The LHCb Bookkeeping Database (BKDB) (see section 3.5), which keeps track of the executed jobs and metadata of the available datasets (what is usually called Metadata Catalog and Job Provenance Database) [67], also keeps information about the physical replicas of the files. A service was built as a front-end to this part of the BKDB, which allows usual File Catalog operations (registering files and their replicas, queries for file replicas for a given location, etc). However, this File Catalog implementation has rather limited functionality, and we looked for other solutions that can be imported as a service into DIRAC.

We have tried out the File Catalog which is part of the AliEn project [68] because of its rich functionality and proven robust implementation. This catalog provides almost all the necessary features that we expect:

- hierarchical structure following the file system paradigm,
- access control list (ACL) mechanisms,
- possibility to store metadata associated with files.

A front-end service was developed to provide access to the AliEn File Catalog functionality. This service keeps a connection to the catalog and translates incoming queries into the AliEn UI commands.

The client APIs to access both File Catalog services are identical, so that data management tools can use either of them (or both simultaneously) by just setting the appropriate configuration parameters.

Other File Catalog implementations are being evaluated in the same way. In particular, we consider the LFC File Catalog [69], the FiReMan Catalog [70] and the new generation of the AliEn File Catalog. The final choice will be made based on the thorough assessment of the catalog properties: completeness of the functionality, scalability, responsiveness with high query rates, reliability, etc (see section 5.3.2.)

Reliable File Transfer Service. File transfer is a fragile operation because of potential network and storage hardware failures or errors in the associated software services. It is not unusual to lose the output of a long job because of the failed data transfer that was never retried. Therefore, a Reliable File Transfer Service (RFTS), which allows retries of the failed operations until complete success is a vital part of the DMS.

In DIRAC the RFTS is constructed using the same building blocks as the WMS (Figure 3-5). Each site maintains a Request Database (RDB) of data operation requests. The requested operations can be data transfer, replication or registration in a File Catalog. One request can contain any number of operations. A special module called the Transfer Agent is continuously checking the contents of the RDB for outstanding requests and attempts to execute them. In case of failures, the request stays in the RDB for further retries. Partially accomplished requests are modified to retry only undone operations.

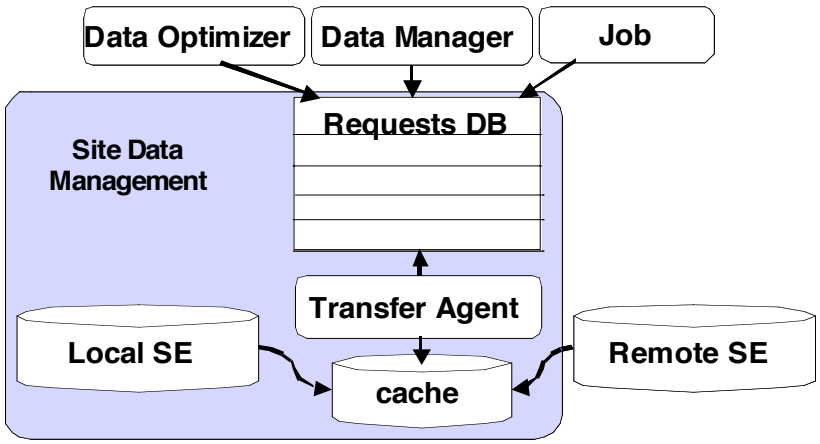


Figure 3-5: On-site data management tools. Arrows originate from the initiator of the action (client) to the destination (server.)

The RDB can be populated either by a regular production job executed on the site or by a special job the only purpose of which is to set a data transfer request. In both cases, the progress of the request execution can be monitored by the standard job monitoring tools provided by the WMS.

The DIRAC RFTS uses basic transfer protocols (such as gridftp) as defined in the Configuration System for transferring files. It is however envisaged to use an underlying centrally provided fts when available (see section 5.3.2.)

3.3.7. Services implementation

All the DIRAC services are written in Python [55] and implemented as XML-RPC servers [71]. The standard Python library provides a complete implementation of the XML-RPC protocol for both the server and client parts.

A significant effort was made to provide fault tolerant services. The crucial services are duplicated to increase their availability. Many requests are repeated in case of failures to overcome network outages or service saturation. All the services are run using the *runit* watchdog tool [72], which ensures restarting in case of failure or on machine reboot. It provides also many other useful features for service manipulation and debugging.

The services should provide secure access to their functionalities based on the de-facto standard GSI security infrastructure adopted on the grid. Several prototypes are being studied. One possibility is to use upgraded XML-RPC servers communicating with clients over the HTTPS protocol enhanced to use grid certificates for authentication. The other possibilities are based on the use of grid service portals ensuring authentication and authorized access to back-end services. In the latter case, an authentication mechanism provided by the GridSite project [73] was evaluated and gave satisfactory results. We are also evaluating the Clarens grid services framework [74] to provide service containers enabled with authentication/authorization mechanisms. As a result of this prototyping work we will have a robust secure grid services infrastructure capable of standing the high load of the LHCb production system.

3.3.8. Interfacing DIRAC to LCG

LCG already in its present state provides a large number of computing resources accessible through the LCG-2 infrastructure. There are several ways to exploit these resources.

The seemingly most straightforward way is to use the standard LCG- provided middleware for job scheduling. However at the time of writing, this approach is not yet reliable enough as demonstrated by the LHCb Data Challenge 2004 [75], so other possibilities had to be explored. An alternative approach consists in sending jobs directly to the LCG CE. This approach was tried out successfully in our DC 2003 [76] to gain access to resources provided by the EDG testbed. However, in the recent Data Challenge 2004 another approach was realized.

This third approach consists of a workload management with reservation of computing resources using pilot-agents. We took advantage of having a light easily deployable “mobile” agent, which is part of the DIRAC native WMS. The jobs that are sent to the LCG-2 Resource Broker (RB) do not contain any particular LHCb job as payload, but are only executing a simple script, which downloads and installs a standard DIRAC agent. Since the only environment necessary for the agent to run is the Python interpreter, this is perfectly possible on all the LCG sites. This pilot-agent is configured to use the hosting Worker Node (WN) as a DIRAC CE. Once this is done, the WN is *reserved* for the DIRAC WMS and is effectively turned into a virtual DIRAC production site for the time of reservation. The pilot-agent can verify the resources available on the WN (local disk space, CPU time limit, etc.) and request to the JMS only jobs corresponding to these resources. The reservation jobs are sent whenever there are waiting jobs in the DIRAC Task queue eligible to run on LCG.

There are many advantages in this approach. The agents running on the WN are ensuring that a valid environment is available before scheduling the real jobs. If the agent fails to start for whatever reason (failure of the RB, site mis-configuration, etc), the DIRAC Task Queue is not affected. This approach allowed LHCb to use both LCG and non-LCG resources in a consistent way. In fact, the LCG RB was used to dynamically deploy the DIRAC infrastructure upon the LCG resources providing a completely homogeneous system. The jobs running on LCG resources were still steered, monitored and accounted for in the same way and by the same services as other DIRAC jobs. This way allowed for efficient use of the LCG resources during the DC 2004 (over 5000 concurrent jobs at peak) with a low effective failure rate, despite the rather high intrinsic failure rate of LCG (about 40%).

The workload management with "resource reservation" by pilot-agents opens interesting opportunities for optimization of job scheduling. While the resource is reserved, it can be used flexibly, for example, running multiple short jobs without repeated scheduling or participating in a coordinated parallel work together with other reserved resources. The latter mode of operation is suitable for running interactive data analysis sessions on the grid.

3.3.9. LHCb Workflow Description

The DIRAC job wrapper that sets the environment and runs the actual application on the WN can accept job scripts, but in order to run complex jobs, a specific workflow description is used. The execution processes as well as the software packages to be used are described using the XML language.

A *workflow* can be defined as a complex diagram of processing phases called *steps*. A step is the smallest unbreakable element in a workflow. While the WMS is entitled to break a workflow into its steps and submit them in parallel if required, steps are always executed within a single job.

Steps in turn are a sequence of *modules* that are themselves most usually scripts (e.g. Python or shell scripts). A library of steps can be used to build up modules that will then be included in a workflow.

Modules in a step and steps in a workflow are connected by their input and output variables (usually temporary files). Several instances of a given step can be used to build a workflow, e.g. several simulation steps used by one or several digitisation steps in order to include spillover events.

The DIRAC Console [77] provides the framework for describing workflows. It contains graphical editors for modules, steps and workflows. Workflows can be instantiated in a Production Request editor to prepare jobs for production. The primary description of workflows uses XML as a description language that is interpreted by the DIRAC job wrapper. A Code Generator can also be used to produce directly executable Python scripts to be submitted to DIRAC or any other WMS.

The DIRAC Console is used successfully in LHCb to prepare production jobs that instantiate complex workflows (such as the stripping jobs). It is very useful, although not mandatory, to create jobs to be submitted to DIRAC.

3.4 GANGA - distributed analysis

A physicist analysing data from LHCb will have to deal with data and computing resources that are distributed across multiple locations and have different access methods. The GANGA application has been developed, in cooperation with ATLAS, to help with this task by providing a uniform high-level interface to the different low-level implementations for the required tasks, ranging from the specification of input data to the retrieval and post-processing of the output.

For LHCb the goal of GANGA is to assist in running jobs based on the Gaudi framework. GANGA is written in Python and presents the user with a single interface rather than a set of different applications. It uses pluggable modules to interact with external tools for operations such as querying metadata catalogues, job configuration and job submission. At start-up, the user is presented with a list of templates for common analysis tasks, and information about ongoing tasks is persisted from one invocation to the next. GANGA can be used either through a command line interface at the Python prompt (CLIP, see Figure 3-6) or through a

GUI (see Figure 3-7). Their behaviour are completely linked allowing easy transition from one to the other.

3.4.1. A typical GANGA session

This section illustrates the use of GANGA through a complete imagined analysis. The physicist will use GANGA as a way to keep track of his/her analysis, much in the same way that we all use our email application to keep track of our communications. For his/her analysis the user wants to analyse 3 large datasets called Data, Monte Carlo and Reference within the DaVinci framework.

```
>>> from GANGA.CLIP import *
>>> dv = DaVinci(optionsfile='myanalysis.opts')
>>> j = Job(name='MyAnalysis, application=dv, backend='DIRAC')
>>> j.submit()
>>> print jobs
Statistics: 2 jobs jobs
----------------
 ID        status              name
# 1    completed    Bd2DstarPi
# 2          new    MyAnalysis
>>> j.submit()
>>> print jobs
Statistics: 2 jobs jobs
----------------
 ID        status              name
# 1    completed    Bd2DstarPi
# 2      running    MyAnalysis
```

Figure 3-6: Use of CLIP from the Python prompt to create and submit a DaVinci job to the DIRAC Workload Management System.

The user starts GANGA and as the first thing selects a small dataset for developing code on in the LHCb Bookkeeping database. The dataset is saved as a local template. The user creates a new job of type DaVinci and develops the C++ code outside GANGA. Using the Job Options Editor, which is a part of GANGA, the job is configured and submitted as a local job. In a series of iterations the user copies the job and resubmits it first as a local job and then to the local batch system for slightly larger datasets.

The user is now ready to perform the analysis. For this he/she creates a set of template jobs, and selects the data to analyse and saves them as local selections. This involves multiple selections in each of the 3 large datasets to deal with differences in running conditions over time. To keep things neat the user divides the analysis up into sub folders corresponding to the 3 different categories.

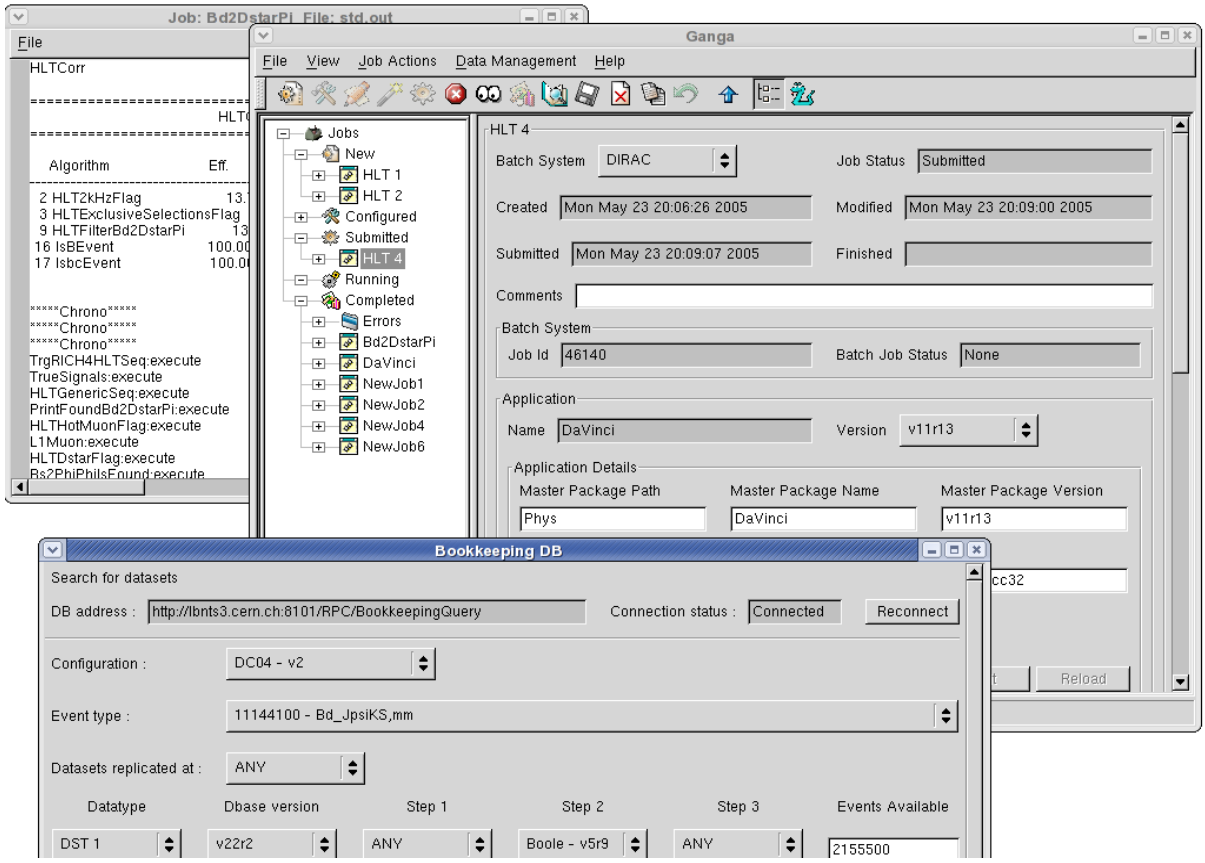


Figure 3-7: Screenshot of a GANGA session using the GUI

The analysis is now submitted by creating a small set of jobs from the templates. The datasets to analyse are specified for the jobs, and a policy for splitting the jobs is defined. The size of the job is such that it will get divided up into the order of 1000 sub jobs. As the jobs will provide rather bulky output it is also decided to specify an alternative location for output files on a scratch disk. The jobs are then submitted to the Grid (via DIRAC.) Days later the user starts a new GANGA session to monitor the progress and looks at some of the output from finished sub jobs to see everything works as expected.

A certain fraction of the sub jobs failed due to a hardware failure. These are resubmitted as identical jobs again. Each sub-job creates a ROOT output file and when all jobs are finished the user merges the output to ease the analysis. Towards the end of the analysis the user cleans up the system by deleting the many jobs that are no longer relevant by selecting them all at a high level and issuing a single delete command.

3.4.2. Implementation

The GANGA project has developed over the last 3 years and the current version 3.0 represents a functional model that is used within the collaboration. However it has several restrictions such as the implementation of new features are difficult due to the design being developed along with the implementation; the central job registry does not scale much beyond 100 jobs; and the existing implementation does not easily allow certain parts of GANGA to be placed on remote servers. It was therefore decided to use the current release as a functional prototype for a complete reimplementation of the core of GANGA. Experience from the current GANGA version was also used to create an updated set of Use Cases for LHCb [78]. This reimplementation, known as GANGA-4 [79] has more or less the same functionality as the existing implementation but without the limitations mentioned above.

The finished product should support all details of the use case outlined in the previous section.

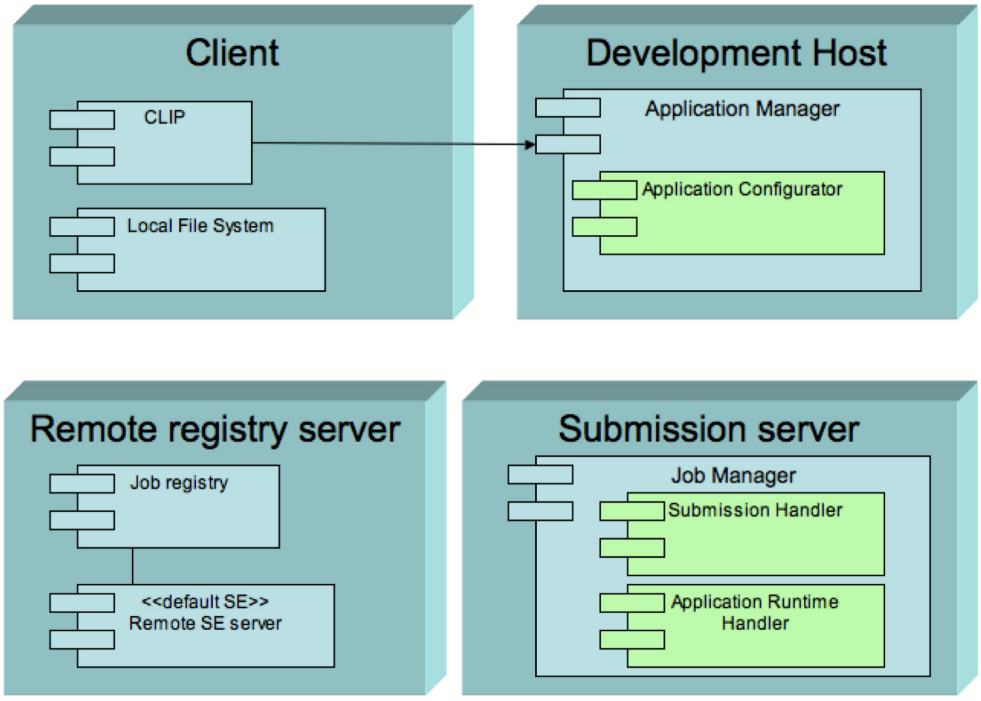


Figure 3-8: The reimplemented Ganga is logically divided into 4 parts that can all be hosted on different client/servers if required

The GANGA client, which the user interacts with directly, is mainly implemented in pure Python. It initially sets up the job and performs client side job splitting if required. If there is server side splitting the client receives a notification from the Job Manager and can then go to the registry to get information on the sub-jobs. Server side sub-jobs are limited in manipulation depending on the features supported by the backend. The client also talks to the bookkeeping or to the file catalog to retrieve data sets. The client communicates with:

- The Application Manager to retrieve available applications and their versions, compiled user code (shared libraries), and pre-processed job-parameters (option files).
- The Application Manager to send application relevant parameters for manipulation (e.g., option files) and user code to be compiled in the context of the chosen application (optional). The client also receives information on pre-processed run-options and compiled user code (shared libraries).
- The Registry Service: several simultaneous registry services are possible, which may be local or remote. The client saves new jobs in the selected remote repository. Once the job is submitted, the Job Manager manages the job status and the client only has read-access to the entry in the registry.
- The Job Manager: the client submits a configured job to the Job Manager. Subsequently it receives notifications of the status from the Job Manager. Commands to restart or kill a job are submitted to the Job Manager, which will take action and subsequently update the registry.

The Application Manager informs the client on available applications it knows about (e.g. generic, DaVinci, general Gaudi, Bender etc.). For all of these it defines sensible defaults to aid the user. The Application Manager processes all user options and the configuration associated with the job itself. It is planned to implement the application manager on a server.

In addition, the Application Manager might also compile user code to generate shared libraries.

The Job Manager accepts jobs from the client. It has knowledge of the supported back-ends (local, batch system, DIRAC etc.) and creates the required wrapper scripts in order to run the job on the chosen back-end. This requires a certain level of matching between back-ends and applications that is taken care of by the Application Runtime Handler. As an example, the method for running jobs within DIRAC is different for a general script and for a Gaudi job that takes advantage of the preinstalled environment. The Job Manager also modifies the job information on the job output depending on what the application and back-end support. As an example the Job Manager may decide to change the output location of output files specified in the output-data to be local and then subsequently copy the data to the final location. After the Client has submitted a job the Job Manager takes ownership on behalf of the client for operations like kill, resubmit or delete. The Job Manager will change the information in the registry each time the job enters into a new status. This includes information on run status, number of subjobs created from server side splitting, location of output data like large ROOT files and the location of the output sandbox. The only information flowing from Job Manager to Client is notifications and information about available back-ends. Following a notification about a job the Client can then update itself from the registry.

Users	Jobs	creation	retrieval	deletion
1	10	0.47	0.1	0.019
1	50	0.29	0.09	0.015
1	100	0.35	0.06	0.01
10	10	0.49	0.08	0.048
10	50	0.31	0.1	0.03
10	100	0.35	0.14	0.028

Table 3-1: Test of multiple users creating, retrieving and deleting a given number of jobs in a remote registry implemented using the ARDA MetaData database. All times are in seconds per job

The registry keeps track of GANGA jobs. It is implemented as a remote registry with a local cache. The remote registry receives job configuration from a client for new jobs. This consists of the job object, the input sandbox and information about the output location (sandbox) and the output-data. For a submitted job the registry receives the job status from the Job Manager. The remote registry is implemented as a dumb storage of information and will not by itself initiate any actions. In Table 3-1 the performance of GANGA is given for the use of multiple clients (i.e. different users) working with several jobs in the remote registry implemented using the ARDA MetaData database[80].

3.4.3. Required Grid services

In order for GANGA to work smoothly for a physics analysis many services are required from the Grid. For LHCb the submission model for distributed analysis jobs is that they will be submitted to the DIRAC WMS and from there go onto the Grid. In this way LHCb will only have to deal with Grid submission from one application. Analysis, opposed to production, presents extra issues with respect to using the Grid. The first one is that analysis is typically done by physicist with a lower computational expertise so requirements on transparency, clear error messages, success rate etc. are higher than for the Grid used for production type tasks. The second one is that analysis jobs are more iterative and individual so the process of changing the executed code and configuration of jobs needs to be much easier than for production jobs.

3.5 Bookkeeping

The processing of data in LHCb is performed iteratively in a chain of programs each executing as a separate processing step, both for real data and for simulated data. Each of these processing steps may require one or several input files and produces one or several output files, which may be data files, tag collections or simply log files. A step is typically running an application that is steered by a set of parameters. In order to ensure reproducibility of these data as well as the possibility to classify the produced data all these parameters need to be recorded and made accessible publicly to the LHCb physicists. The Bookkeeping Database (BKDB) is used for storing all these parameters. Typically a physicist will perform queries to the BKDB in order to select a dataset to be analysed.

A dataset is a set of files logically grouped according to common properties. The atomic dataset is a single file. The published information allows:

- Fast selection of datasets possibly consisting of many individual files according to predefined physics criteria
- Detailed browsing of all parameters characterizing a given file and its processing history (job provenance).

The Bookkeeping database is accessed by two sets of users with different characteristics:

- **Data Production Managers**, who supervise and control major data processing efforts on behalf of the collaboration. The data production on one hand publishes the provenance information to the bookkeeping facility, but also requires access to datasets e.g. in the case of reprocessing. The production manager must also be able to mask faulty files such that they are not selectable for physics analysis.
- **Physicist users**, who develop data analysis algorithms in order to extract physics parameters from the data. Physicists query the bookkeeping system in order to select the dataset they are interested in. Physicist users are interested to obtain a subset of the produced datasets depending on physics parameters related to their subject of work.

Both may need access to the detailed history of a produced dataset e.g. in case of problems.

The different access mechanisms to the datasets are discussed below.

In order to facilitate the development of tools such as graphical user interfaces (GUI) and applications for production managers and users, the information must be accessible using a programming interface (API) and not expose the internal database schema.

3.5.1. The Data Model

The data model needs to be flexible enough to describe different processing steps and the resulting files. The relation between processing steps and files has the constraints that:

- Every file is the output of a unique step but may be the input of many steps. This is valid for data files as well as event tag collections.
- A step may have several files as input and several files as output

The schema for steps and jobs contains the following information:

- Each step is described by
  - The step execution date
  - The configuration tag of the application identified by a name and a version.
  - A set of parameters that characterises the step, given as name/value/type triplets. Examples of parameters are the production site, the run number (if any), the version of the detector geometry used etc.

  - A set of input files and a set of output files (possibly empty)
- Each file is described by
  - A logical name.
  - A file type.
  - A set of quality flags.
  - A set of parameters. Examples of parameters are the file size. If the file is a data file, more parameters may be defined such as the run number, the first and last event numbers, the number of events in the file, the event type etc.
- A File Type is
  - A name (RAW, DST, Log, Tag Collection, etc.)
  - A version number.
  - A set of parameters if more details are required to describe the data type. An example is a short description of the type.
- A Quality flag
  - A group, which is the type of analysis or the group of people concerned by this quality. Examples are “Production Manager” or “Calo”.
  - The actual quality, which can be “Good”, “Bad” and “Not checked”.

Figure 3-9 shows the logical model describing execution steps and the corresponding input- and output files using generic (name, value) tuples. Data quality flags facilitate the exclusion of certain files. Such a data model is flexible enough to host any kind of job provenance information.

In absence of a suitable file catalog in the early stage of the BKDB development, a table has been added to the schema containing replica information for those files that are made persistent. Not all files registered in the file tables do actually correspond to a replicated file, but they need to be present in the BKDB in order to provide history / provenance information for subsequent steps and files.

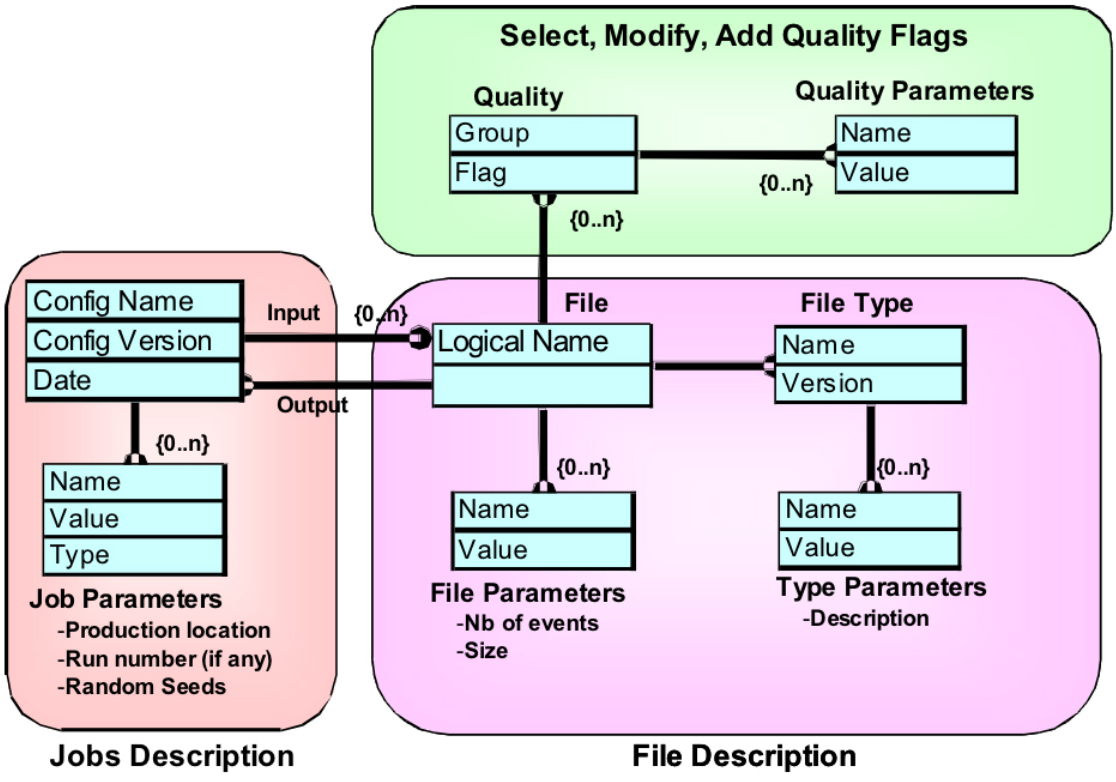


Figure 3-9: The logical data model of the Bookkeeping application

3.5.2. Views to the Bookkeeping

Though very flexible, the above data model cannot satisfy all criteria required by the different clients. A (name, value) based data model is not optimised to allow e.g. the fast selections of many datasets according to many parameters describing either the dataset itself or the parent step(s). On the other hand it is not necessary that such queries always completely reflect the latest update, but one can usually allow for certain latency.

Following known recipes of data warehousing [81] these requirements were implemented using separate views created from the primary data model, which are optimized for the different client applications. An example is the WWW GUI interface (see Figure 3-10) and the browsing interface as it is used by GANGA (see Figure 3-7.)

Another application, which is based on a view of the primary information of the replica table, is the implementation of a read-only file catalog interface.

The views optimized for access by individual applications need to be refreshed regularly depending on the tolerable latency. Such a refresh is implemented either as a complete view recreation or an incremental update if only very few changes need to be reflected.

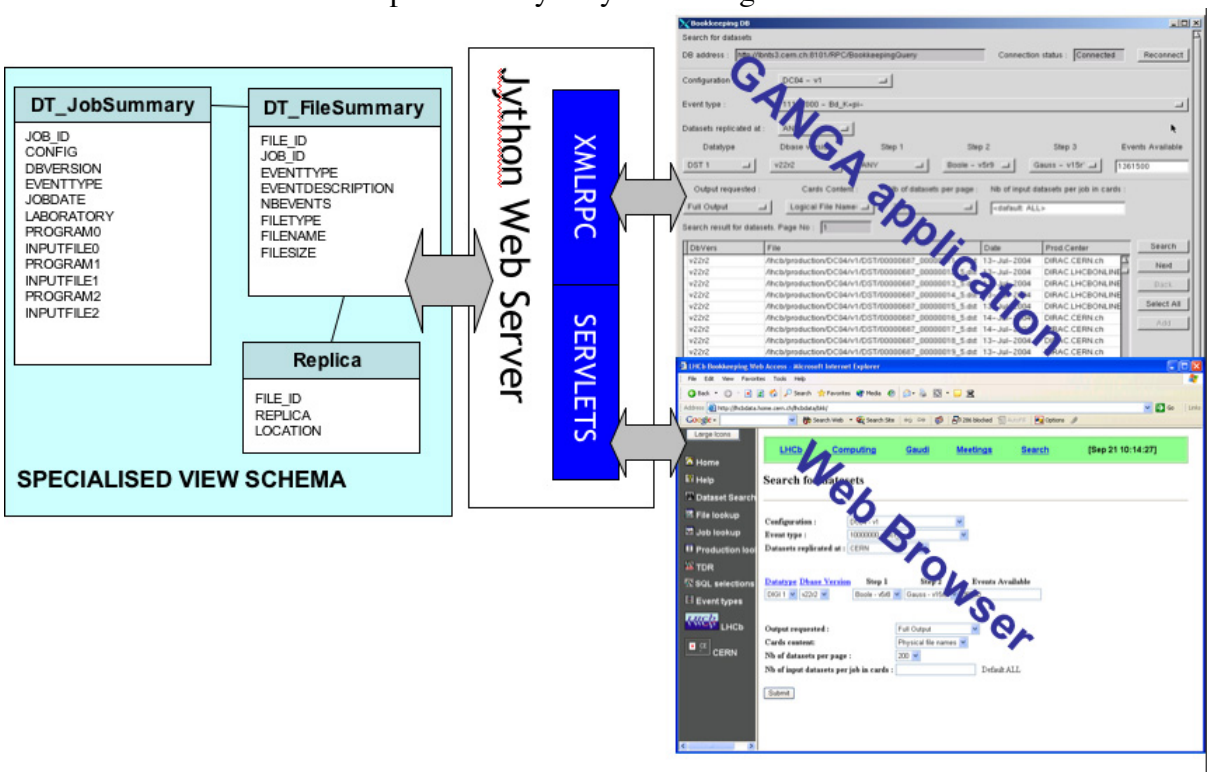


Figure 3-10: The browsing applications to the bookkeeping.

3.5.3. The BKDB Application Programming Interface

Any application accessing the bookkeeping and job history information can use either of two interfaces: an HTTP interface on which e.g. the web GUI is implemented and an application programming interface.

As shown in Figure 3-11, neither the web based data access nor the API implementation of the interfaces depends on the internals of the data model, both rely on a “Bookkeeping service”. The bookkeeping service itself implements two interfaces: a read-only and a read/write interface to emphasize the logical distinction between query and update roles.

The programming interface only requires network access; data are transferred using the XML-RPC [71] protocol. The XML-RPC protocol provides a very flexible and lightweight communication mode with much less overhead than e.g. the SOAP protocol [82]. Any client using this API is entirely independent from the database structure or technology. For dedicated applications like the GANGA GUI or the read-only file catalogue, also the distilled data stored in some of the bookkeeping views can be programmatically accessed in read-only with a special API using the XML-RPC protocol.

The web-based interfaces were implemented using standard servlet technology, which can be hosted by web servers like TomCat [83]. The XML-RPC based API is currently hosted by a standard python web server.

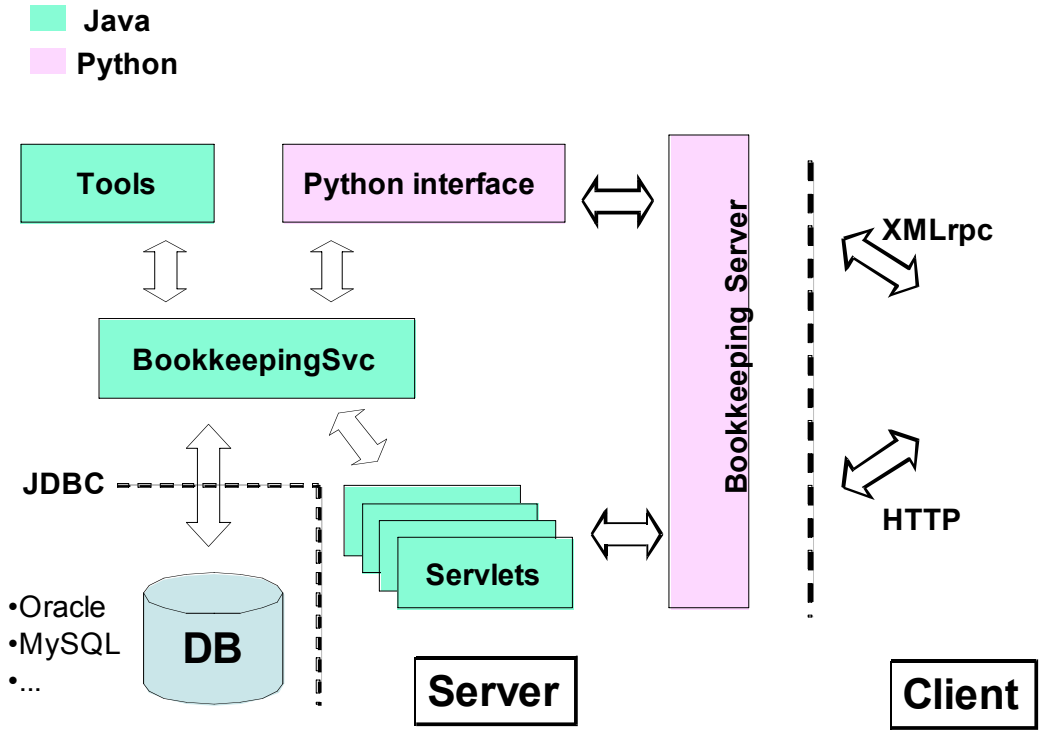


Figure 3-11 The implementation of the various interfaces of the bookkeeping facility.

3.5.4. Experience with the BKDB

LHCb has used the bookkeeping implementation described above since 2003. Currently the bookkeeping contains the characterization of ~8 x 10^6 files described by 26 x 10^6 parameters. These files were produced by ~3.5 x 10^6 processing steps described by 64 x 10^6 parameters.

The complete recreation of all required views takes on average 40 minutes. A daily incremental update typically finishes within a few minutes.

The bookkeeping applications have shown to be rather independent of the database technology. Whereas the application is deployed on ORACLE 10g [18], a prototype is available using mySQL [19].

3.5.5. Alternative Implemetation

After the LHCb bookkeeping applications were operational, the ARDA project started to define an experiment-independent metadata catalogue [80]. We are currently investigating if our data provenance model can be applied using the ARDA metadata catalogue concept.

Clearly a solution shared between several experiments would be favourable for the following reasons:

- The API would be standardized and implementation could be replaced without consequences on applications.
- The maintenance of the implementing software would be shared and the effort for LHCb would be smaller.

The ARDA concept is solely based on name, value pairs attached to files. The main differences between the two approaches are:

- The concept of “Steps” representing executed applications does not exist. The step information must be replicated for each file.
- The ARDA model is based on logical files organized in a file-system like directory structure. Such an approach may have, depending on the implementation, a better scaling behaviour than the solution currently used for the BKDB. However, scanning the entire step or file space, for which our model was designed, is rather costly due to such partitioning concepts.
- The concept of “Steps” representing executed applications has to be implemented using the concept of logical files i.e. a step is described by a special type of file.
- In the ARDA model the schema for the parameters describing a file is shared for all entries in a directory.

First results show that the ARDA model is functionally able to replace the views created from the provenance data. Further tests are ongoing and it is too early for a final decision about moving from the existing solution.

Chapter 4 Workflow and Computing Models

4.1 Introduction

This section describes the dataflow model for all stages in the processing of the real and simulated LHCb events. The CPU and storage, both disk and mass storage (MSS), requirements for 2006-2010 are given based on estimates made from the current software; these estimates are under continuous review. In addition, the trigger rates and selection efficiencies of the various processing steps should be considered as the current best estimates.

The roles of the various Tier centres are discussed and the distribution of the processing load and storage needs are given. Requirements are also presented for the computing infrastructure, both internal (e.g. MSS i/o rates) and external (e.g. data transfer rates) to the Tier centres.

The baseline LHCb computing model is based on a distributed multi-tier regional centre model. It attempts to build in flexibility that will allow effective analysis of the data whether the Grid middleware meets expectations or not. Of course this flexibility comes at the cost of a modest requirement overhead associated with pre-distributing data to the regional centres. Analysis is foreseen at the Tier-1 centres and possibly the larger Tier-2 centres. The LHCb Tier-1 centres are, in general, already familiar in providing such analysis centres for current HEP experiments and the associated infrastructure is already in place or in a mature state of planning.

4.2 Logical Dataflow and Workflow Model

There are several phases in the processing of event data; this section describes the terminology used to define each processing step and the data sets that are produced. The various stages normally follow each other in a sequential manner, but some stages may be repeated a number of times. The workflow reflects the present understanding of how to process the data. A schematic of the logical dataflow is shown in Figure 4-1 and is described in more detail in this section.

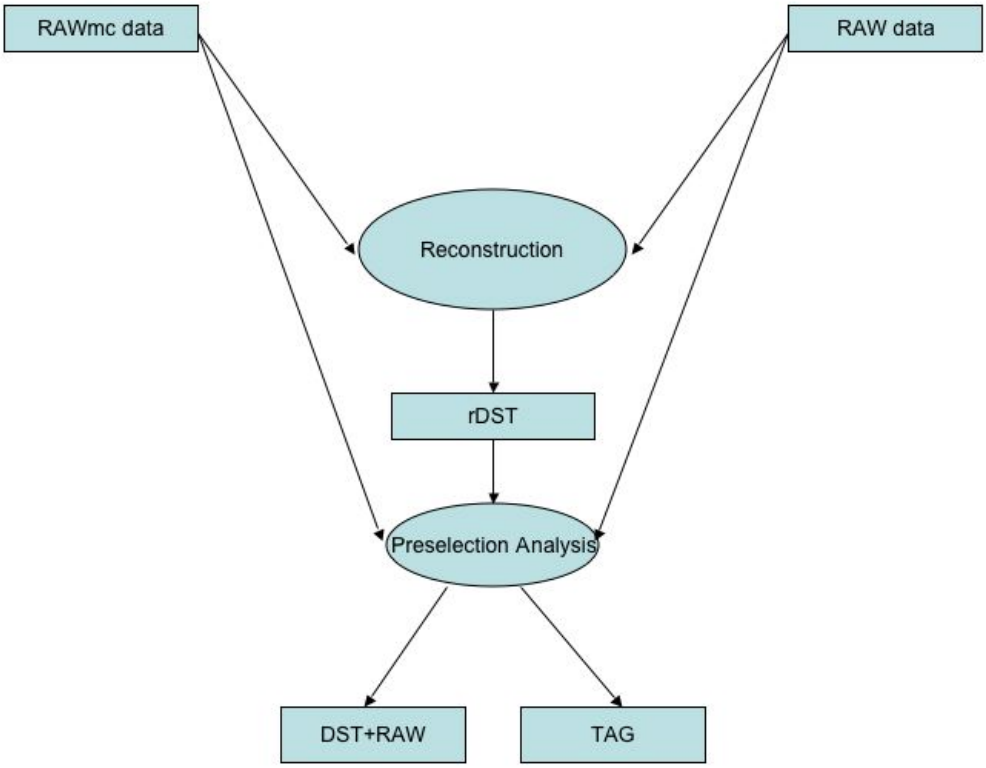


Figure 4-1: The LHCb computing logical dataflow model. The arrows indicate the input/output datasets at each data processing stage.

4.2.1. RAW data

The "real" raw data from the detector is produced via the Event Filter farm of the online system. The first step is to collect data, triggering on events of interest. This procedure involves processing data coming from the sub-systems using sophisticated and highly optimised algorithms in the High Level Triggers. The trigger software will apply calibration corrections during the reconstruction of physical properties of the particles and will apply selections based on physics criteria. The results of this step are the RAW data. For convenience the RAW data can be grouped in several output streams.

The RAW data are transferred to the CERN Tier 0 centre for further processing and archiving. Those data not selected for permanent storage by the trigger are lost forever.

4.2.2. Simulated data

The simulated data are produced from a detailed Monte Carlo model of LHCb that incorporates the current best understanding of the detector response, trigger response and dead material. These RAWmc data sets contain simulated hit information and extra 'truth' information. The truth information is used to record the physics history of the event and the relationships of hits to incident particles. This history is carried through to subsequent steps in the processing so that it can be used during analysis. Simulated raw data sets are thus larger than real raw data. Otherwise the format of the simulated raw data is identical to that of the real data and they are processed using the same reconstruction software. In analogy with the "real" data the RAWmc will, in general, only be stored for events that pass the trigger simulation.

4.2.3. Reconstruction

The RAW data, whether real or simulated, must then be reconstructed in order to provide physical quantities: calorimeter clusters to provide the energy of electromagnetic and hadronic showers, trackers hits to be associated to tracks whose position and momentum are determined. Information about particle identification (electron, photon, π^0, hadron separation, muons) is also reconstructed from the appropriate sub-systems.

The event reconstruction results in the generation of new data, the Data Summary "Tape" (DST). Only enough data will be stored in the DST that is written out during reconstruction to allow the physics pre-selection algorithms to be run at a later stage. This is known as a reduced DST (rDST.)

The pattern recognition algorithms in the reconstruction program make use of calibration and alignment constants to correct for any temporal changes in the response of the detector and its electronics, and in its movement. Calibration and alignment data as well as necessary detector information (detector conditions) will be stored in a distributed database.

The calibration and alignment data will be produced from online monitoring and/or off-line from a pre-processing of the data associated with the sub-detector(s). Detector conditions will be a subset of the Experiment Control System database and will contain only information needed for reconstruction, e.g. information for monitoring the detector will not be included.

It is planned to reprocess the data of a given year once, after the end of data taking for that year, and then periodically as required.

The reconstruction step will be repeated to accommodate improvements in the algorithms and also to make use of improved determinations of the calibration and alignment of the detector in order to regenerate new improved rDST information.

4.2.4. Data stripping

The rDST is analysed in a production-type mode in order to select event streams for individual further analysis.

The rDST information (tracks, energy clusters, particle ID) is analysed to determine the momentum four vectors corresponding to the measured particle tracks, to locate primary and secondary vertices and algorithms applied to identify candidates for composite particles whose four-momentum are reconstructed. Each particular channel of interest will provide such a pre-selection algorithm. The events that pass a physics working group's selection criteria are written out for further analysis. Since these algorithms use tools that are common to many different physics analyses they are run in production-mode as a first step in the analysis process. This is shown schematically in Figure 4-2.

The events that pass the selection criteria will be fully re-reconstructed, recreating the full information associated with an event. The output of the stripping stage will be referred to as the (full) DST and contains more information than the rDST.

Before being stored, the events that pass the selection criteria will have their RAW data added in order to have as detailed event information as needed for the analysis. We note that in the early stages of data taking both the Fermilab and HERA experiments needed access to the RAW data for analysis. It is envisaged the amount of information stored at the output of the stripping stage will reduce as the experiment and the accelerator matures.

An event tag collection will be created for faster reference to selected events. It contains a brief summary of each event's characteristics as well as the results of the pre-selection

algorithms and a reference to the actual DST record. The event tags are stored in files independent of the actual DST files.

It is planned to run this production-analysis phase (stripping) 4 times per year: once with the original data reconstruction; once with the re-processing of the RAW data, and twice more, as the selection cuts and analysis algorithms evolve.

It is expected user physics analysis will primarily be performed from the output of this stage of data processing (DST+RAW and TAG.) During first data taking it is foreseen to have at least 4 output streams from this stripping processing: two associated with physics directly (b-exclusive and b-inclusive selections) and two associated with "calibration" (dimuon and D* selections)[1], discussed in more detail in section 4.3.1.

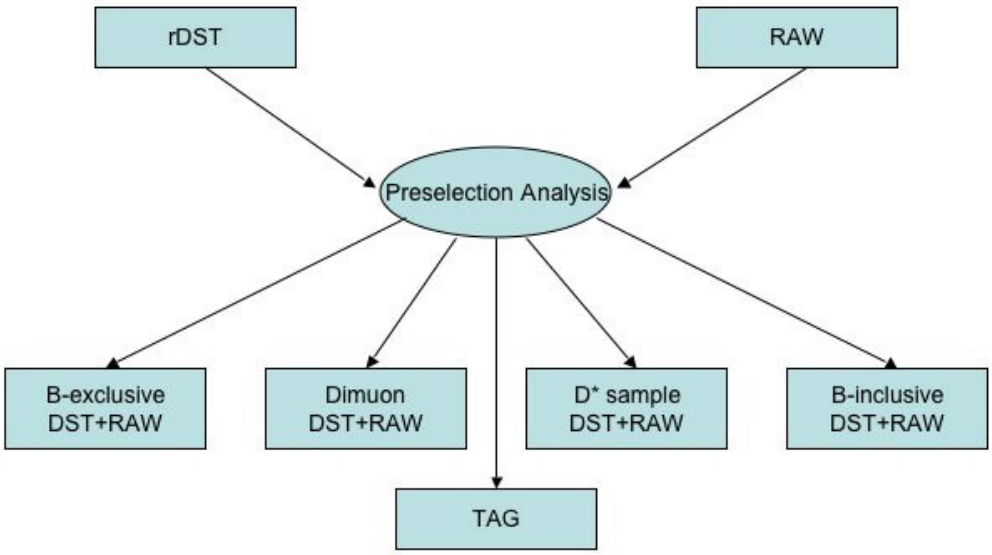


Figure 4-2: Schematic of the logical dataflow for the production analysis phase. The arrows indicate the input/output datasets at each data processing stage.

4.2.5. Analysis

Finally physicists will run their Physics Analysis jobs, illustrated in Figure 4-3. They process the DST output of the stripping on events with physics analysis event tags of interest and run algorithms to reconstruct the B decay channel being studied. Therefore it is important that the

[1] It is quite possible there will be more than 4 output streams, corresponding to subsets of the 4 categories.

output of the stripping process is self-contained. This analysis step generates quasi-private data (e.g. Ntuples or personal DSTs), which are analysed further to produce the final physics results.

Since the number of channels to be studied is very large, we can assume that each physicist (or small group of physicists) is performing a separate analysis on a specific channel. These "Ntuples" could be shared by physicists collaborating across institutes and countries, and therefore should be publicly accessible.

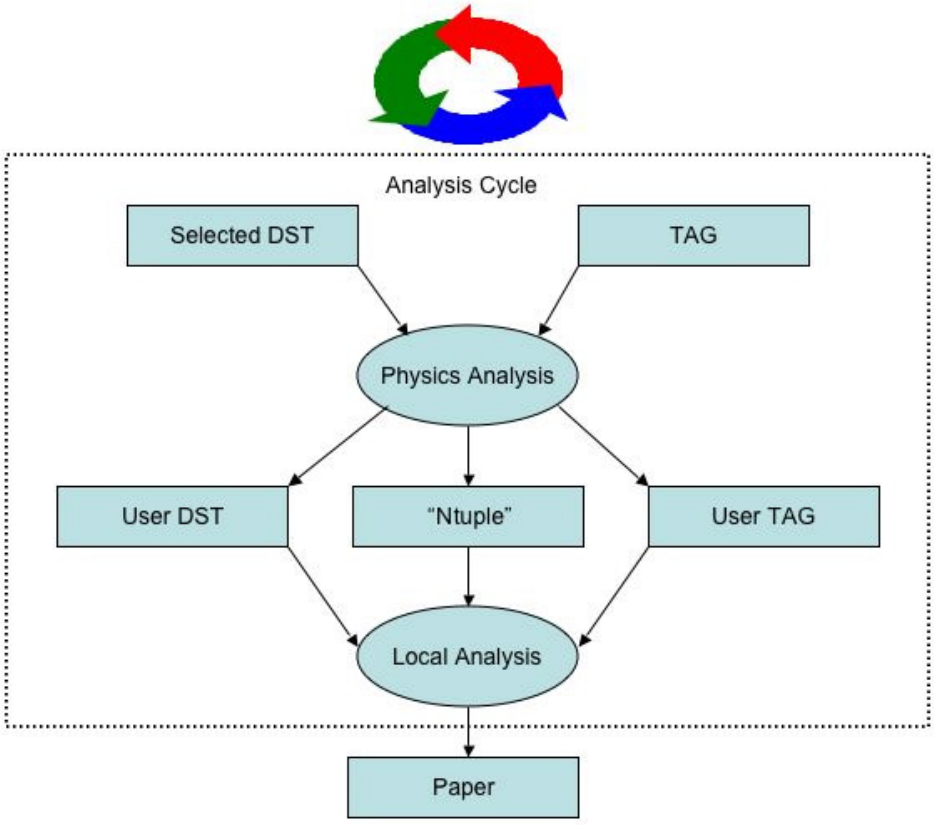


Figure 4-3: LHCb physicist analysis cycle

4.3 Data Processing and Storage Requirements

The frequency of each of the data processing operations, the volume of input and output data, and the amount of computing hardware resources needed to accomplish the tasks must be quantified in order to specify the computing model precisely. A detailed breakdown of the processing and data requirements has been made in terms of each processing stage. The parameters used to estimate these requirements for real data are given in Table 4-1. The expected event sizes listed correspond to the size of data as stored on disk.

Event Size	kB
RAW	25
rDST	25
DST	75
Event processing	**kSI2k.s**
Reconstruction	2.4
Stripping	0.2
Analysis	0.3

Table 4-1: Event parameters for real data

In this section the estimates of the CPU and storage requirements do not assume any inefficiencies.

4.3.1. Online Requirements

A detailed discussion of the online and trigger systems has been presented elsewhere [3][38]. The Event Filter Farm will contain of the order of 1800 CPUs and the Online system will provide about 40 TB of local storage at the experimental pit.

The High Level Trigger (HLT) receives data, at 40 kHz, corresponding to the full event after each positive Level 1 decision. The HLT will then be applied in a series of steps of increasing refinement until the event is either positively accepted or rejected. The events can be thought of as being classified in 4 categories: exclusive b sample, inclusive b sample, dimuon sample and D* sample[2]. The expected trigger rate after the HLT for each of these samples is given in Table 4-2.

The b-exclusive sample will be fully reconstructed on the online farm in real time and it is expected two streams will be transferred to the CERN computing centre: a reconstructed b-exclusive sample at 200Hz (RAW+rDST), the “hotstream”, and the RAW data sample at 2kHz. The RAW event size is expecterd to be 25kB, compared to the current measured value of ~30kB, whilst there is an additional 25kB associated with the rDST. This would correspond to a sustained transfer rate of 60MB/s, if the data is transferred in quasi real-time.

	b-exclusive	dimuon	D*	b-inclusive	Total
HLT rate (Hz)	200	600	300	900	2000

Table 4-2: Working numbers for HLT output rates

[2] It is appreciated that there will be events that satisfy more than 1 selection criteria; for the sake of simplicity this overlap is neglected.

Throughout this document, we assume an effective running period each year of 10^7 seconds over a 7-month period, starting in 2008. We expect to accumulate 2×10^{10} events per year, corresponding to 500TB of RAW data.

4.3.2. Reconstruction Requirements

The CPU time and event size associated with the reconstruction are summarised in Table 4-1. The CPU requirements for the reconstruction programme have been stable for a prolonged period and are not envisaged to change substantially from 2.4 kSI2k.sec per event.

One reconstruction pass of the complete data set of 2×10^{10} events would require computing resources equivalent to ~1.5MSI2k.years. A detailed breakdown of the CPU requirements is given in Table 4-3 and Table 4-4.

	b-exclusive[3]	Dimuon	D*	b-inclusive	Total
Input fraction	0.1	0.3	0.15	0.45	1.0
Number of events	2×10^9	6×10^9	3×10^9	9×10^9	2×10^{10}
MSS storage (TB)	50	150	75	225	500
CPU (MSI2k.yr)	0.15	0.45	0.23	0.68	1.52

Table 4-3: Offline resource requirements for the reconstruction of each stream

	Duration (months)	CPU power (MSI2k)
Reconstruction	7	2.61
Re-processing	2	9.12

Table 4-4: CPU requirements for the reconstruction, excluding the subsequent stripping

The current size of the (full) DST is 125kB/event and since the LHCC review of the computing model a prototype rDST has been implemented that meets the 25 kB/event.

It is anticipated to make use of the CPU capacity of the Event Filter Farm outside of data taking periods for reprocessing of events.

Re-processing of the complete year's data sample will need to be performed at least once during the year of data taking. The CPU capacity that will be available from the Event Filter Farm corresponds to a power of ~5.4MSI2k and it would be available for a minimum period of 2 months. This is a significant computing capacity that we intend to harness to full effect.

[3] The first pass on the b-exclusive stream will be made on the Event Filter Farm immediately after the HLT decision. Another copy of this output will be kept on disk for the duration of that particular year's data taking.

4.3.3. Stripping Requirements

The stripping will take place 4 times; twice in association with the reconstruction of the data and twice outside of these periods.

The total amount of integrated CPU power required for this phase is modest but will be required over a short-time period, therefore peak CPU needs have to be considered. In order to estimate the CPU requirements to strip the data we assume 0.2 kSI2k.s per event is needed to make a decision. This compares to the current 0.65 kSI2k.s. We believe this can be improved significantly, for example by ensuring that particular algorithms are executed once per event rather than once per selection algorithm.

The complete stripped sample is reconstructed, using the latest algorithms and calibrations. Each stripping and the subsequent reconstruction requires a total of ~0.3 MSI2k.years.

Outside of a reconstruction period we would aim to produce the stripped files in a period of a month. In order to meet this requirement CPU power of ~3.4MSI2k (assuming no inefficiency) will be needed for the duration of the stripping.

If the duration were longer than a month it would seem unreasonable to perform 4 stripping passes per year. This frequency seems the maximum time lapse before the full sample becomes available with the latest selections. This is particularly true in the early data-taking period when algorithms are frequently being refined and new algorithms being developed.

There will be *at least* 4 output streams from the stripping associated with the b-exclusive, dimuon, D* and b-inclusive samples. For the b-exclusive and b-inclusive events, the full information of the DST and RAW will be written out and it is expected to need 100 kB/event. For the dimuon and D* streams only a subset of the DST information, associated with objects of interest, will be written out, with the RAW information added; this is estimated to be 50 kB/event. The optimal event size for the dimuon and D* samples is still under study. The stripping reduction factor, number of events, output event size per stripping and CPU requirements of the 4 streams are given in Table 4-5. In addition an event tag file will be produced which will contain information on each event, whether or not it was selected. The size of this tag data will be ~20TB per processing.

	Exclusive-b	Dimuon	D*	Inclusive-b	Total
Input fraction	0.1	0.3	0.15	0.45	1.00
Reduction factor	10	5	5	100	9.57
Event yield per stripping	2×10^{8}	1.2×10^{9}	6.0×10^{8}	9.0×10^{7}	2.09×10^{9}
CPU (MSI2k.year)	0.03	0.13	0.06	0.06	0.29
Storage requirement per stripping (TB)	20	60	30	9	119
TAG (TB)	2	6	3	9	20

Table 4-5: Reduction factors and computing requirements of the stripping stage

4.3.4. Simulation Requirements

Simulation studies are made in order to measure the performance of the detector and of the event selections in particular regions of phase space, and to estimate the efficiency of the full reconstruction and analysis of the B decay channel. The number of simulated events that would have to be generated to determine the detector performance and background levels from simulation, given the sheer number of b-events triggered at LHCb, is too large a computing task. Therefore the general performance of the detector is envisaged to be understood from the large statistics dimuon and D* samples collected; this is based on the experience of b-physics analysis performed, for example, by CDF at the Tevatron.

The simulation strategy is to concentrate on particular needs that will require an inclusive b-sample and the generation of particular decay modes for a channel under study. The inclusive sample numbers are based on the need for the statistics to be sufficient so the total error is not dominated by Monte Carlo statistical error. To that end these requirements can only be best guess estimates.

It is anticipated that 2×10^9 signal events will be generated plus an additional 2×10^9 inclusive events. Of these 4×10^9 simulated events, it is estimated that 4×10^8 events will pass the trigger simulation and will be reconstructed and stored on MSS.

The simulation process involves a number of steps:

- physics generation (e.g. using PYTHIA or other generators), cuts are applied at an early stage to take only those events that are in the acceptance of the detector,
- the tracking through the detector using GEANT4 to produce detector hit information,
- digitisation to simulate the response of the detector and produce the simulated RAW data,
- triggering, to select those events that would pass the LHCb trigger,
- full reconstruction of the triggered event sample.

The first two bullet points are handled by the Gauss application; the next two by the Boole application and the final one by the Brunel application. The breakdown of the CPU requirements for each of these applications is given in Table 4-6.

In summary ~6.5 MSI2k.years will be needed to meet LHCb simulation requirements; this dominates the CPU needs for LHCb.

	Application	Nos. of events	CPU time/evt (kSI2k.s)	Total CPU (kSI2k.year)
Signal	Gauss	2×10^9	50	3171
	Boole	2×10^9	1	63
	Brunel	2×10^8	2.4	15
Inclusive	Gauss	2×10^9	50	3171
	Boole	2×10^9	1	63
	Brunel	2×10^8	2.4	15
Total				6499

Table 4-6: CPU requirements for simulation

It is not anticipated that all the GEANT4 generated hits will be stored with the Monte Carlo RAW data, but some truth information and relationships will be stored to allow the analysis of the simulated data. The current event size of the Monte Carlo DST (with truth information) is approximately 500kB/event. We are confident that this can be decreased to 400kB/event. Again TAG data will be produced to allow quick analysis of the simulated data, with ~1kB/event. The data volumes per year are given in Table 4-7.

The total storage required for the simulated data is ~160TB.

	Output	Nos. of events	Storage/evt (kB)	Total Storage (TB)
Signal	DST	2×10^8	400	80
	TAG	2×10^8	1	0.2
Inclusive	DST	2×10^8	400	80
	TAG	2×10^8	1	0.2
Total				160.4

Table 4-7: Storage requirements for simulation

4.3.5. User Analysis Requirements

The user analysis discussed in this document is performed in batch mode and includes an element of systematic studies of individual sub-detectors. The physicist, starting from the stripped DST, further reduces this sample to focus on one particular analysis channel producing a Ntuple-like object (or perhaps a "private" stripped DST.) This reduced sample maybe used by a single physicist or a small number of collaborators, but it is assumed further iterative cycles on the Ntuple will be performed on resources local to the physicist, hence beyond the scope of this document.

In general it is assumed a physicist will process ~10^6 events per job; this is based on a channel of interest that will have been appropriately tagged. It is recognised some analyses will run over larger event samples, ~10^7 events per job. It is assumed that 0.3 kSI2k.sec per event will be needed and corresponds to current experience.

The total needed CPU power required for user analysis on the real data is based on the assumptions given in Table 4-1 and Table 4-8. The same parameters are used in analysing both the real data and Monte Carlo. The required number of jobs per annum to analyse both Monte Carlo and real data is ~ 30k; each job will analyse on average 2.8×10^6 events.

The CPU resources needed for analysis in 2008 is ~0.8MSI2k.

It is expected this requirement will grow linearly with the available data during the early phase of the experiment i.e. by 2010 the needs will be ~2.4MSI2k.year.

It is recognised that some analyses will run a toy-Monte Carlo model for sensitivity studies; these could require non-negligible CPU resources but have minimal input and output data requirements.

Nos. of physicist performing analysis	140
Nos. of analysis jobs per physicist/week	2
Fraction of jobs analysing 10^6 events	80%
Fraction of jobs analysing 10^7 events	20%
Event size reduction factor after analysis	5
Number of "active" Ntuples	5
2008 CPU needs (MSI2k.years)	0.78
2008 Disk storage (TB)	200

Table 4-8: Estimate of analysis requirements, excluding any efficiencies

The estimated storage requirements for analysis are ~200TB.

This estimate of storage requirements uses the assumptions given in Table 4-8 on event size reduction and the number of "active" Ntuples for data and Monte Carlo. Like the CPU requirements, the estimate is that these storage needs will grow linearly in the early years of data taking.

4.4 Computing Model

4.4.1. Introduction

In this section we will describe a baseline model but we will comment on possible variations where we believe this could introduce additional flexibility. A schematic of the LHCb computing model is given in Figure 4-4. CERN is the central production centre and will be responsible for distributing the RAW data in quasi-real time to the Tier-1 centres. CERN will also take on a role of a Tier-1 centre. This centre at CERN will be essential for accessing the "hotstream" data to understand the behaviour of the detector during data taking. An additional six Tier-1 centres have been identified: CNAF(Italy), FZK(Germany), IN2P3(France), NIKHEF(The Netherlands), PIC(Spain) and RAL(United Kingdom) and an estimated 14 Tier-2 centres. These Tier-2 centres may not all qualify as signatories to the LCG Memorandum of Understanding that is in preparation. CERN and the Tier-1 centres will be responsible for all the production-processing phases associated with the real data. The RAW data will be stored in its entirety at CERN, with another copy distributed across the 6 Tier-1's. The 2nd pass of the full reconstruction of the RAW data will also use the resources of the LHCb online farm. As the production of the stripped DSTs will occur at these computing centres, it is envisaged that the majority of the distributed analysis of the

physicists will be performed at CERN and at the Tier-1's. The current year's stripped DST will be distributed to all centres to ensure load balancing. To meet these requirements there must be adequate networking not only between CERN and the Tier-1's but also between Tier-1's; quantitative estimates will be given later.

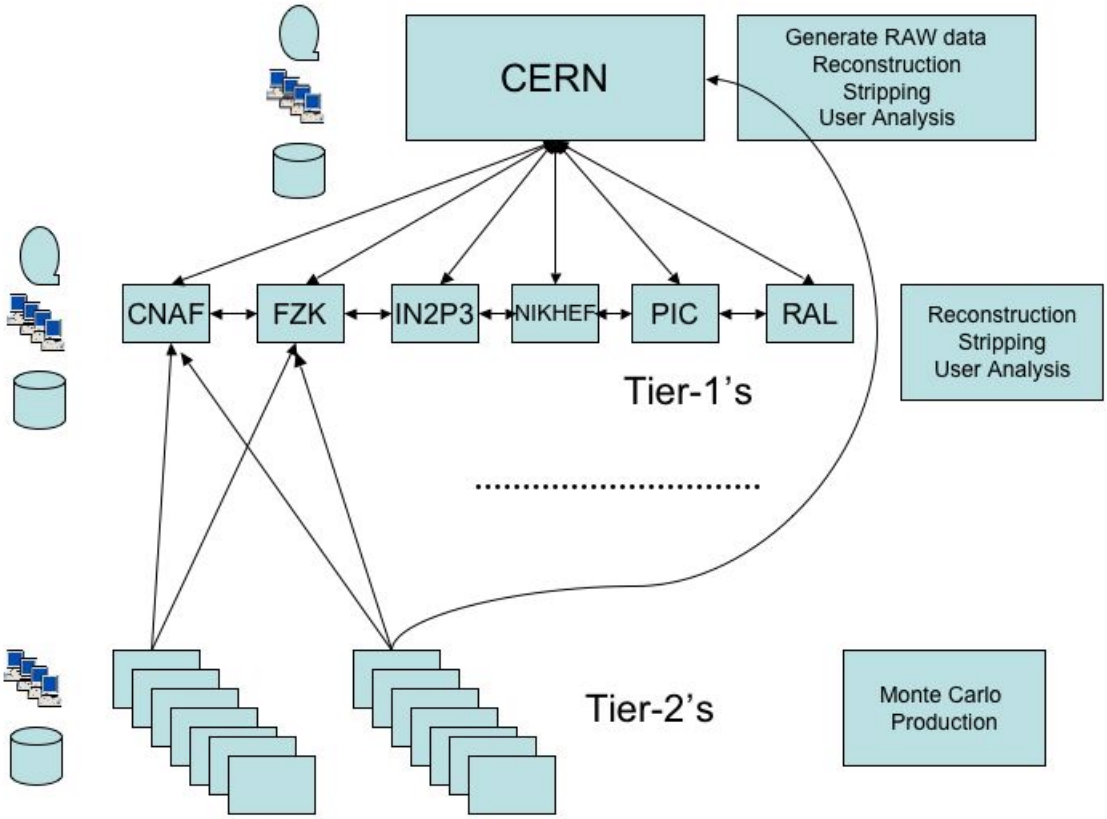


Figure 4-4: Schematic of the LHCb Computing Model

The Tier-2 centres will be primarily Monte Carlo production centres, with both CERN and the Tier-1's acting as the central repositories for the simulated data. This envisages that there will be network traffic between the Tier-2's to both CERN and the Tier-1's. It should be noted that although we do not envisage any analysis at the Tier-2's in the baseline model presented, it should not be proscribed, particularly for the larger Tier-2 centres. Both the Tier-2 network needs and a minimum requirement for a Tier-2 centre to support analysis will be given later.

In the following sub-sections we estimate the resource requirements at CERN, the Tier-1's and the Tier-2's incorporating an efficiency factor into the calculation. These efficiency factors are listed in Table 4-9 and are the same as those that were applied at the time of the April estimation of the resource needs (given to the MoU taskforce.)

	Efficiency factors
Scheduled CPU usage	85%
Chaotic CPU usage	60%
Disk usage	70%
MSS Usage	100%

Table 4-9: Efficiency factors for CPU and storage needs

A "typical" Tier-1 centre is assumed to be one sixth of the total integrated Tier-1 capacity, although it is recognised that not all Tier-1 centres will contribute equally.

4.4.2. Requirements for 2008

For the purpose of this document, 2008 is assumed to be the first full year of data taking corresponding to 10^7 seconds of data taking. LHCb assumes that the delivered luminosity will be 2×10^{32} cm^{-2} s^{-1} with any appropriate de-focussing of the beam. The nominal data taking rate to storage for LHCb is 2 kHz, constituted from the 4 "independent" components given in Table 4-2.

Raw Data

A first reconstruction of the b-exclusive channel will be performed online. It is envisaged there will be two streams of data from the experimental pit to the CERN Tier-0 corresponding to:

- 2kHz RAW data, and
- 200Hz RAW & rDST corresponding to the reconstructed b-exclusive channel.

This in total corresponds to a transfer rate of 60 MB/s, if data is transferred in quasi-real time, during data taking.

The RAW data from the CERN computing centre will then be distributed across the Tier-1 computing centres. It is assumed appropriate disk buffering will be provided at the CERN centre to facilitate this.

Data processing during data taking

It is expected that the reconstruction and the first stripping of the data at CERN and at the Tier-1's will follow the production in quasi real-time, with a maximum delay of a few days. The DST output of the stripping will remain on disk for analysis and be distributed to *all other* Tier-1centres and CERN, whilst the RAW and rDST will then be migrated to the mass storage system.

The RAW data will have to be distributed from the Tier-0 amongst the Tier-1's. The CERN fraction of the stripped DST and TAGs (assuming 1/7) will have to be distributed to all 6 Tier-1's and CERN will receive all DSTs and TAGs produced at the 6 Tier-1's. This will occur over the 7 month data taking period.

This will require a sustained rate over the network at the Tier-0 of 40MB/s. The breakdown of these 40 MB/s is given in Table 4-10.

	RAW (TB) Outbound	RAW (TB) Inbound	DST (TB) Outbound	DST (TB) Inbound	Network bandwidth (MB/s)
@ CERN ↔Tier-1's	500	-	119	119	40
@Tier-1 ↔ CERN	-	83	19.8	19.8	6.7
@ Tier-1 ↔Tier-1's	-	-	102	102	11

Table 4-10: Network transfer needs during experimental data taking.

The traffic at a typical Tier-1 with CERN will be 6.7 MB/s. Table 4-10 also shows the inter-Tier-1 traffic for a typical Tier1. The total traffic at a given Tier-1 amounts to 17.7 MB/s.

Averaged over the period and summed over the CERN and the Tier-1 centres the rate to the MSS is sustained at

- ~98 MB/s (CERN: 40 MB/s; Tier-1's: 58 MB/s)

The computing power needed to perform the reconstruction and stripping over the 7 months is

- ~3.7 MSI2k

including the efficiency factor quoted in Table 4-9. This is in addition to reconstruction of the "hotstream" on the online farm.

The rDST and the RAW associated with the exclusive-b stream will be kept on disk at CERN as will 10% of the data of the other 3 channels.

Re-processing of data

The re-processing of the data will occur over a 2-month period. During this process the RAW data will need to be accessed from the MSS both at CERN and the Tier-1 centres.

The CPU resources available at the pit will allow 42% of the total re-processing and subsequent stripping to be performed there. Hence at CERN there is an additional complication that this data will also have to be transferred to the pit; similarly the produced rDST and stripped DSTs will have to be transferred back to the CERN computing centre and then distributed to the Tier-1 centres. Given the compressed timescale, the transfer rate between the Tier-0 and the pit is estimated to be

- ~90 MB/s,

higher than that required during data taking. It is assumed appropriate disk buffering, associated with the MSS, will be provided to allow this re-processing at CERN and the Tier-1 centres. The data accessed by this re-processing amounts to:

- 500 TB in total for the input RAW,
- 500 TB for the output rDST, and
- 139 TB for the output DST (associated with the stripping.)

The re-processing of the remaining 58% of the data not processed at the pit will be shared by the 6 Tier-1's and CERN, which will bring CERN's contribution to ~50%.

The DST output of the stripping will remain on disk for analysis and will be distributed to *all* other production centres. To enable later stripping it is necessary to distribute a fraction of the rDST produced at CERN during this re-processing to the Tier-1's; this is a consequence of the large contribution from the online farm.

The network requirements between CERN and the Tier-1's, a typical Tier-1 and CERN and between Tier1's during this 2-month period are summarised in Table 4-11.

	rDST (TB) Outbound	rDST (TB) Inbound	DST (TB) Outbound	DST (TB) Inbound	Network bandwidth (MB/s)
@ CERN ↔Tier-1's	181	-	422	68.7	128
@Tier-1 ↔ CERN	-	30.2	11.5	70.3	21
@ Tier-1 ↔Tier-1's	-	-	57.3	57.3	22

Table 4-11: Network transfer needs during re-processing

The computing power needed to perform these tasks is

12.8 MSI2K out of which 5.4 MSI2k are provided by the LHCb online infrastructure, hence only 7.4 MSI2k from external resources.

The average access rate (integrated over the 6 Tier-1's and CERN) to the MSS would be

- ~243 MB/s (CERN: 88 MB/s; all 6 Tier-1's: 155 MB/s.)

Additional Strippings

The (two) stripping productions outside of the reconstruction or of the re-processing of the data will be performed over a one-month period. Both the RAW (500 TB) and the rDST (500 TB) will need to be accessed from the MSS to perform this production. The produced stripped DSTs (139 TB) will be distributed to all production centres.

The network transfer requirements during this period are given in Table 4-12. During this period the network traffic between the Tier-1's will be at its highest.

	DST (TB) Outbound	DST (TB) inbound	Network bandwidth (MB/s)
@ CERN ↔Tier-1's	119	119	91
@Tier-1 ↔ CERN	19.9	19.9	15
@ Tier-1 ↔Tier-1's	99.3	99.3	76

Table 4-12: Network transfer needs during stripping.

The average access rate (integrated over the 6 Tier-1's and CERN) to the MSS would be

- ~486 MB/s (CERN: 107 MB/s; Tier-1's: 379 MB/s)

Again appropriate disk buffering is assumed for both the networking & MSS access.

The CPU power required for this month will be

- ~4.0 MSI2k.

It should be noted that for the stripped DST it is intended to keep only the latest and next-to-latest copy of the current year's b-exclusive and b-inclusive data on disk and the latest copy of the calibration dimuon and D* channel at each processing centre, though the current year's data for all stripping passes will be available on the MSS at CERN and a copy distributed across the MSS's of the Tier-1 centres.

Monte Carlo production

The Monte Carlo production is expected to be an ongoing activity throughout the year and is the mainstay of the Tier-2 centres. The Tier-1 centres and CERN will act as the repository for the produced Monte Carlo data. The whole of the current year's Monte Carlo production DST (~160 TB) will be available on disk at CERN and another 3 copies, on disk, distributed amongst the 6 Tier-1 centres. The transfer rates from a typical Tier-2 to the Tier-1's or CERN are relatively small, ~1.1 MB/s and 0.4 MB/s respectively averaged over the year.

The CPU requirements over the year (including the efficiency factors in Table 4-9) are

- 7.6 MSI2k.

It is assumed a buffer space of 10% of the year's production will be available at the Monte Carlo production centres, which is a modest 23 TB integrated over the Tier-2 centres. The Monte Carlo data will be eventually archived to the MSS with 1 copy at CERN and another copy distributed amongst the Tier 1 centres.

User Analysis

Analysis will be distributed across the data production centres (CERN and the Tier-1's). Due to better access to the RAW data, past copies of the stripped DST and the availability of the Monte Carlo data, we foresee CERN servicing a larger fraction of the analysis, which we estimate at 25%. The mean analysis power averaged over the year is

- 1.3 MSI2k

(including the efficiency factor), though it is recognised there will be peak demand leading up to conferences; experience from BaBar indicates this peak could be as high as that required for reconstruction.

For a Tier-2 to provide a facility for analysis we estimate it should provide a minimum disk storage of ~0.2PB for 1 copy of the latest stripped DST, at least 5% of the CPU requirements for analysis (in addition to the Monte Carlo production) and the networking infrastructure to be able to support the replication of the data to that computing centre. For a Tier-2 to support analysis it should be able to receive the latest version of the stripped DSTs in quasi-real time which corresponds to ~50 MB/s during the 1 month stripping process.

Summary

The CPU requirements for 2008 are summarised in Table 4-13. The total CPU requirements are 12.97 MSI2k.years excluding the 0.9 MSI2k.year contribution from the LHCb online farm for the re-processing. The fractional distribution is 7% CERN, 34% Tier-1's and 59% Tier-2's. The disk requirements are given in Table 4-14. The total is ~3.3 PB with a fractional breakdown between CERN and the Tier-1's being 25% and 75% respectively. The MSS storage requirements are given in Table 4-15. In 2008 it is estimated a similar amount, ~3.4 PB, of mass storage is required as is needed for disk with 40% of the MSS data being at CERN.

	CERN	Tier1's	Tier2's	Total
Stripping	0.17	1.03	0.0	1.20
Full reconstruction	0.40	2.42	0.0	2.82
Monte Carlo	0.0	0.0	7.6	7.6
Analysis	0.32	0.97	0.0	1.29
Total	0.90	4.42	7.65	12.97

Table 4-13: 2008 CPU requirements in MSI2k.years

	CERN	Tier-1's	Tier-2's	Total
RAW	136	0	0	136
rDST	136	0	0	136
Stripped DST	440	1954	23	2417
TAG	45	267	0	312
Analysis	70	210	0	280
Total	826	2432	23	3281

Table 4-14: 2008 disk requirements in TB

	CERN	Tier-1's	Total
RAW	500	500	1000
rDST	143	857	1000
Stripped DST	636	636	1272
TAG	80	80	160
Total	1359	2074	3433

Table 4-15: 2008 MSS requirements in TB

4.4.3. Requirements for 2009

Due to the de-focussing of the beam at the LHCb experimental area the luminosity is still assumed to be 2×10^{32} cm^{-2} s^{-1} in 2009 (and 2010.) The assumption for a data taking year remains 10^7 seconds extended over 7 months.

The processing of the data is essentially identical to the situation described in section 4.4.2 with the following exceptions.

Data Processing

During the re-processing of the data during the shutdown it is envisaged to re-reconstruct the stripped data of 2008 in addition; this has a modest change in the CPU requirements (~0.2MSI2k.years, increasing the need during re-processing to ~8.5 MSI2k compared to ~7.4 MSI2k in 2008.) There is an additional requirement to store this re-processed 2008 data on the MSS (1 copy at CERN and another distributed amongst the Tier-1's.) It is also anticipated that the needs for analysis will double as the amount of data doubles with an additional 1.3 MSI2k.years. Overall this leads to a modest 11% increase in our CPU requirements compared to 2008, see Table 4-16.

	CERN	Tier1's	Tier2's	Total
Stripping	0.20	1.19	0.0	1.39
Full reconstruction	0.40	2.42	0.0	2.82
Monte Carlo	0.0	0.0	7.65	7.65
Analysis	0.64	1.94	0.0	2.59
Total	1.25	5.55	7.65	14.45

Table 4-16: 2009 CPU requirements in MSI2k.years

MSS Cumulative Storage

The major cumulative effect however is how the data from 2008 is stored and its distribution between the centres and the proportion on disk and MSS. It is envisaged to store 1 copy of each stripped DST at CERN and another distributed over the Tier-1 centres on the MSS. This is equivalent to

- ~1.4 PB.

Two copies of the previous year's RAW data (1 at CERN and the other distributed) will remain on MSS and a distributed copy (between CERN and the Tier-1's) of the two reconstruction passes, which in total is

- 2 PB.

The previous year's Monte Carlo will be stored at CERN with another copy distributed across the Tier-1 centres ~321 TB of data in total. The total MSS requirements are listed in Table 4-17. Including the need to store on MSS the re-reconstructed stripped data from 2008, the total requirement has doubled to ~7 PB with the same fractional breakdown as 2008.

	CERN	Tier-1's	Total
RAW	1000	1000	2000
rDST	286	1714	2000
Stripped DST	1391	1391	2782
TAG	181	181	362
Total	2857	4285	7144

Table 4-17: 2009 MSS requirements in TB

Disk Cumulative Storage

On disk, the latest copy of the stripped DST from previous years will be stored at CERN with another two copies of the b-exclusive and the b-inclusive and a single copy of the dimuon and D* distributed over the Tier-1 centres, making 3 copies of the b-exclusive and inclusive in total and 2 copies of the dimuon and D* samples; this corresponds to

- ~400 TB (including the TAG data.)

Another ~300 TB increase in disk derives from the increased analysis requirements. The disk storage needs are summarised in Table 4-18 and are a 22% increase on the requirements in

2008 to ~4 PB, with a fractional breakdown between CERN and the Tier-1's of 27% and 73% respectively.

	CERN	Tier-1's	Tier-2's	Total
RAW	136	0	0	136
rDST	136	0	0	136
Stripped DST	610	2165	23	2798
TAG	74	311	0	385
Analysis	140	420	0	560
Total	1095	2897	23	4015

Table 4-18: 2009 disk requirements in TB

Networking and MSS access

The network requirements in 2009 are similar to those in 2008. The greatest change occurs during the re-processing of the data. The details of these changes are listed in Table 4-19. Similarly the MSS i/o rates are almost identical to 2008 except for the period of re-processing of the data where the requirement is 296 MB/s (CERN: 114 MB/s; Tier-1: 182 MB/s.)

	Transfer rate (MB/s)
CERN ↔Tier-1's	173
Tier-1↔CERN	29
Tier-1↔Tier-1's	60

Table 4-19: Network requirements for re-processing in 2009

4.4.4. Requirements for 2010

The processing of the data is again similar to the situation described in 2008 with the following additional requirements.

Data Processing

During the re-processing of the data during the shutdown it is envisaged to re-reconstruct the stripped data of 2009. In addition it is anticipated to re-reconstruct the 2008 data commencing from the RAW data; this will have to occur in parallel with the data taking period. This amounts to an additional computing power requirement of:

- ~2.1 MSI2k.year

The maximum CPU power requirement will remain at 8.5 MSI2k during the 2-month annual re-processing but with an increased need for reconstruction during data taking of 7.2 MSI2k i.e. double the resources required during 2008 and 2009. The needs of the analysis will continue to grow; it is anticipated that in 2010 they will be three times the requirements of 2008. Overall this leads to a 35% increase in our integrated CPU requirements compared to 2008, see Table 4-20.

	CERN	Tier1's	Tier2's	Total
Stripping	0.25	1.48	0.0	1.73
Full reconstruction	0.66	3.96	0.0	4.62
Monte Carlo	0.0	0.0	7.65	7.65
Analysis	0.97	2.91	0.0	3.88
Total	1.88	8.35	7.65	17.88

Table 4-20: 2010 CPU requirements in MSI2k.years

MSS Cumulative Storage

The major cumulative effect is still dominated by how the data from the previous two years are stored and its distribution between the centres and the proportion on disk and MSS. It is envisaged to continue to store 1 copy of each stripped DST at CERN and another distributed over the Tier-1 centres from both previous years on the MSS. This corresponds to

- ~1.7 PB above the 2009 level.

Two copies of the previous year's RAW data (1 at CERN and the other distributed) will remain on MSS and a distributed copy of the two reconstruction passes, which in total is a

- 2 PB increase compared to the 2009 requirements.

In addition, the re-processed 2008 rDST data will have to be stored on MSS with one copy at CERN and another distributed around the Tier-1 centres; this corresponds to an additional 0.5 PB of storage.

The previous years Monte Carlo will be stored at CERN with another copy distributed across the Tier-1 centres corresponding to ~321 TB of data. The total MSS requirements are listed in Table 4-17. The total requirement has grown to ~12PB, a factor ~3.4 increase compared to 2008, with a similar fractional breakdown as 2008 and 2009.

	CERN	Tier-1's	Total
RAW	1500	1500	3000
rDST	500	3000	3500
Stripped DST	2265	2265	4530
TAG	301	301	602
Total	4566	7066	11632

Table 4-21: 2010 MSS requirements in TB

Disk Cumulative Storage

The disk storage will follow the principles outlined for the 2009 accumulation of data on disk. For 2010 this will see another ~700 TB increase in disk needs (a 45% increase compared to 2008), the detailed breakdown is given in Table 4-22.

	CERN	Tier-1's	Tier-2's	Total
RAW	136	0	0	136
rDST	136	0	0	136
Stripped DST	780	2377	23	3180
TAG	102	376	0	478
Analysis	210	630	0	890
Total	1363	3363	23	4749

Table 4-22: 2010 disk requirements in TB

Network and MSS access

The major change in the estimated MSS i/o rate, compared to 2009, is associated with the data taking period when the 2008 data is being re-processed during data taking. The additional processing results in i/o rate that is estimated to be ~166 MB/s (CERN: 54 MB/s; Tier-1's: 112 MB/s) over the 7-months of data taking. There is also an increase in the network needs, above 2009 needs, during the re-processing and a breakdown is given in Table 4-23.

	Transfer rate (MB/s)
CERN ↔Tier-1's	53
Tier-1↔CERN	9
Tier-1↔Tier-1's	22

Table 4-23: 2010 network requirements during data taking.

4.5 Profiles

As was recognised at the time of the LHCC review of the computing model, the LHCb requirement profiles are not flat at CERN or the Tier-1 centres and are peaked during particular data processing periods. The Tier-1 centres serving LHCb are listed in Table 4-24, along with the other proposed supported experiments. All centres support either all 4 or 3 of the LHC experiments; therefore it is necessary to investigate the peaks in the LHCb requirements in conjunction with, at least, the needs of the other LHC experiments.

Centre	LHCb	ALICE	ATLAS	CMS
CNAF	X	X	X	X
FZK	X	X	X	X
IN2P3	X	X	X	X
NIKHEF	X	X	X	
PIC	X		X	X
RAL	X	X	X	X

Table 4-24: List of LHCb Tier-1 centres and the experiments that are supported

The assumptions made when investigating the resource profiles of LHCb and the other experiments was that data taking commences April and runs to the end of October. The anticipated CPU profiles broken down by month, for CERN, the Tier-1's and the Tier-2's, from the start of 2008 to the end of 2010 are given in Figure 4-5. Overall there are well-defined steps in CPU needs corresponding to the start proton-proton data taking. LHCb requirements even though not flat are small perturbations on the overall requirements for the LHC experiments.

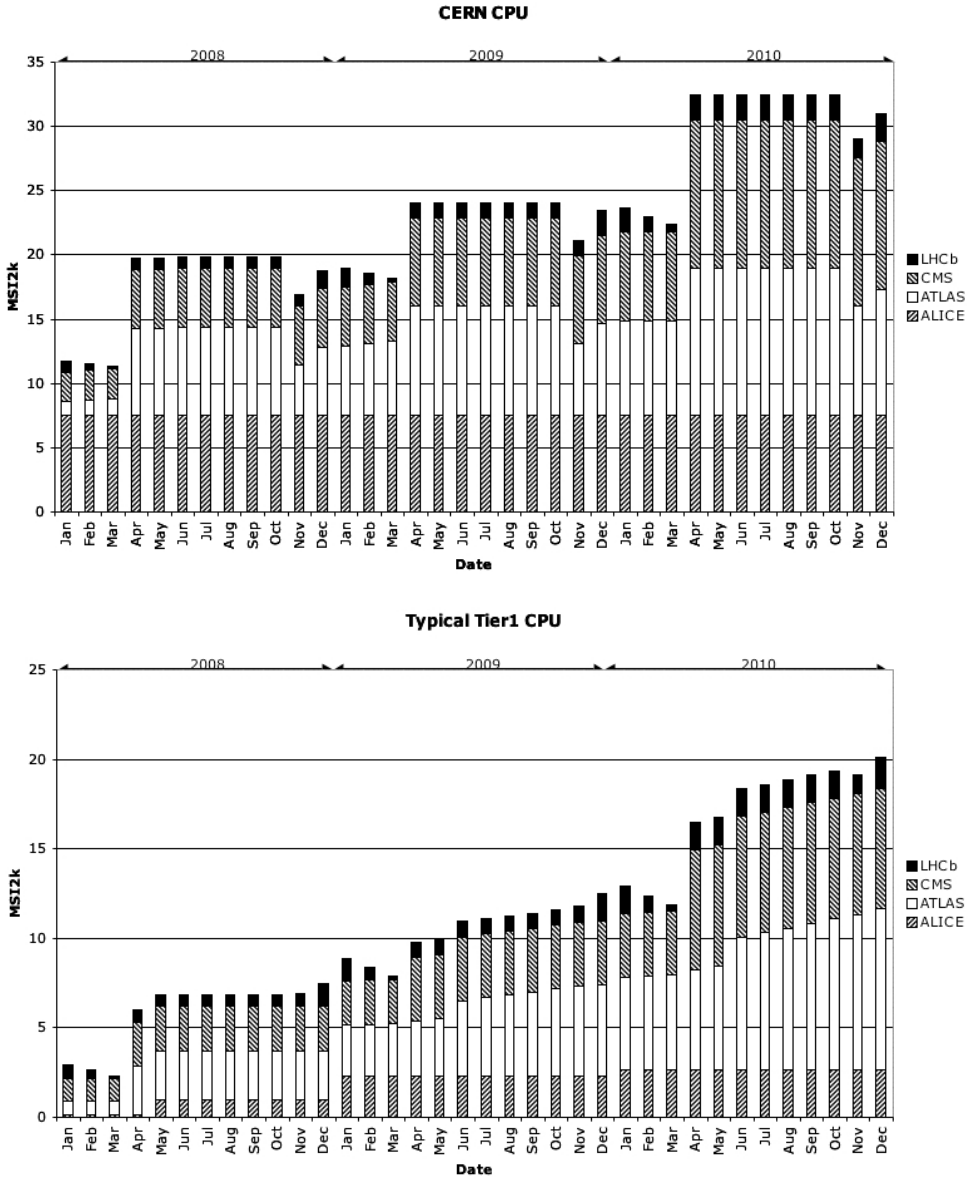


Figure 4-5: CPU profiles for the 4 LHC experiments broken down by month from 2008 to 2010 at CERN and a "typical" Tier-1.

Similarly the profiles of the needs for the MSS i/o and the network requirements are shown in Figure 4-6 broken down by month. Peaks exist in the MSS i/o rate, particularly for CERN, that corresponds to the heavy ion running period. LHCb peak needs fall outside of that period and are small in comparison to ALICE and CMS. The integrated network needs peak during the proton-proton data-taking period whilst in fact the peak need for LHCb falls outside of this period.

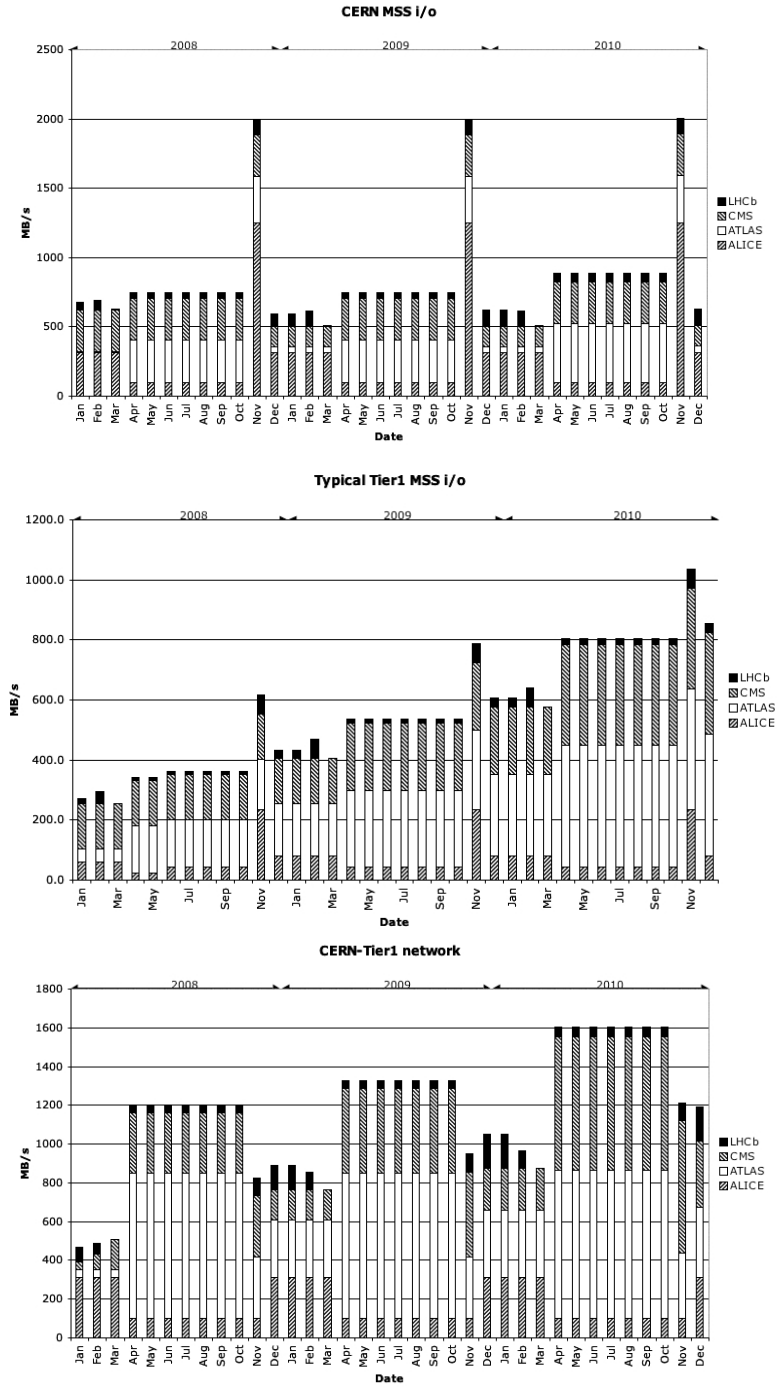


Figure 4-6: MSS i/o and CERN-Tier1 network needs for the 4 LHC experiments broken down by month from 2008 to 2010

In summary the profile of the anticipated resource needs of LHCb through 2008 to 2010 seem to be acceptable when taken in context of the overall requirements of the LHC experiments.

4.6 Summary

It is anticipated that the 2008 requirements to deliver the computing for LHCb are 13.0 MSI2k.years of processing, 3.3 PB of disk and 3.4 PB of storage in the MSS. The CPU requirements will increase by 11% in 2009 and 35% in 2010. Similarly the disk requirements

will increase by 22% in 2009 and 45% in 2010. The largest increase in requirements is associated with the MSS where a factor 2.1 is anticipated in 2009 and a factor 3.4 for 2010, compared to 2008. The requirements are summarised in Table 4-25. The estimates given in 2006 and 2007 reflect the anticipated ramp up of the computing resources to meet the computing requirements need in 2008; this is currently 30% of needs in 2006 and 60% in 2007. This ramp up profile should cover the requirements of any data taken in 2007.

CPU(MSI2k.yr)	2006	2007	2008	2009	2010
CERN	0.27	0.54	0.90	1.25	1.88
Tier-1's	1.33	2.65	4.42	5.55	8.35
Tier-2's	2.29	4.59	7.65	7.65	7.65
Total	3.89	7.78	12.97	14.45	17.88
Disk(TB)					
CERN	248	496	826	1095	1363
Tier-1's	730	1459	2432	2897	3363
Tier-2's	7	14	23	23	23
Total	984	1969	3281	4015	4749
MSS (TB)					
CERN	408	825	1359	2857	4566
Tier-1's	622	1244	2074	4285	7066
Total	1030	2069	3433	7144	11632

Table 4-25: LHCb computing resource estimates 2006-2010

Chapter 5 LHCb & LCG

5.1 Introduction

In this chapter a description of the LHCb use of the LCG Grid during Data Challenge'04 is outlined. The limitations of the LCG at the time and the lessons learnt are highlighted. We also summarise the baseline services that LHCb need in the LCG in order for the data to be processed and analysed in the Grid environment in 2007. The detailed implementation of these services within the LHCb environment is described earlier in this document.

5.2 Use of the LCG Grid

The results described in this section reflect the experiences and the status of the LCG during the LHCb data challenge in 2004 and early 2005. The data challenge was divided into three phases:

- Production: Monte Carlo simulation
- Stripping: Event pre-selection
- Analysis

The main goal of the Data Challenge was to stress test the LHCb production system and to perform distributed analysis of the simulated data. The production phase was carried out with a mixture of LHCb dedicated resources and LCG resources. LHCb managed to achieve their goal of using LCG to provide at least 50% of the total production capacity. The third phase, analysis, has yet to commence.

5.2.1. Production

The DC04 production used the Distributed Infrastructure with Remote Agent Control (DIRAC) system. DIRAC was used to control resources both at DIRAC dedicated sites and those available within the LCG environment.

A number of central services were deployed to serve the Data Challenge. The key services are:

- A production database where all prepared jobs to be run are stored
- A Workload Management System that dispatches jobs to all the sites according to a "pull" paradigm
- Monitoring and Accounting services that are necessary to follow the progress of the Data Challenge and allow the breakdown of resources used
- A Bookkeeping service and the AliEn File Catalog (FC) to keep track of all datasets produced during the Data Challenge.

Before the production commenced the production application software was prepared for shipping. It is an important requirement for the DIRAC system to be able to install new versions of the LHCb production software soon after release by the production manager. All the information describing the production tasks are stored in the production database. In principle the only human intervention during the production by the central manager is to prepare the production tasks for DIRAC. The first step of production is the preparation of a

workflow, which describes the sequence of applications that are to be executed together with the necessary application parameters. Once the workflow is defined, a production run can be instantiated. The production run determines a set of data to be produced under the same conditions. The production run is split into smaller jobs to facilitate the scheduling procedure. Each DIRAC production agent request is served with a single job. When new datasets are produced on the worker nodes they are registered by sending an XML dataset description to the bookkeeping service. The output datasets are then transferred to the associated Tier-1 and the replica is registered in the bookkeeping service.

The technologies used in this production are based on C++ (LHCb software), Python (DIRAC tools), Jabber/XMPP (instant messaging protocol used for reliable communication between components of the central services) and XML-RPC (the protocol used to communicate between jobs and central services). ORACLE and MySQL are the two databases behind all of the services. ORACLE was used for the production and bookkeeping databases, and MySQL for the workload management and AliEn FC systems.

On the LCG, “agent installation” jobs were submitted continuously. These jobs check if the Worker Node (WN) where the LCG job was placed was configured to run a LHCb job. If these checks were in the affirmative, the job installed the DIRAC agent, which then executed as on a non-Grid DIRAC site within the time limit allowed for the job; turning the WN into a virtual DIRAC site. This mode of operation on LCG allowed the deployment of the DIRAC infrastructure on LCG resources and uses them together with other LHCb Data Challenge resources in a consistent way.

A cron script submits DIRAC agents to a number of LCG resource brokers (RB). Once the job starts execution on the WN, and after the initial checks are satisfied, the job first downloads (using http) a DIRAC tarball and deploys a DIRAC agent on the WN. The DIRAC agent is then configured and started. This agent then requests tasks from the DIRAC WMS. If any task is matched the task description is downloaded on the WN and executed. The software is normally pre-installed with the standard LCG software installation procedures [84]. If the job is dispatched to a site where software is not installed, then installation is performed in the current work directory for the duration of the job. All data files as well as logfiles of the job are produced in the current working directory of the job. Typically the amount of space needed is around 2 GB plus an additional 500 MB if the software needs to be installed. The bookkeeping information (data file “metadata”) for all produced files is uploaded for insertion into the LHCb Bookkeeping Database (BKDB) At the end of the reconstruction, the DST file(s) are transferred by GridFTP to the SEs specified for the site, usually an associated Tier1 centre and CERN (as Tier-0.) Once the transfer is successful, the replicas of the DST file(s) are registered into the LHCb-AliEn FC and into the replica table of BKDB. Both catalogues were accessed via the same DIRAC interface and can be used interchangeably.

By the end of the production phase, 3000 jobs were regularly executed concurrently on LCG sites. A total of 211k jobs were submitted to LCG, LHCb cancelled 26k after 24-36 hours in order to avoid the expiration of the proxy. Of the remaining 185k, 113k were regarded as successful by the LCG. This is an efficiency of ~61%. A breakdown of the performance is given in Table 5-1. A further breakdown of these 113k successful jobs was made and is summarised in Table 5-2. The initialisation errors included missing Python on the worker node, failure of DIRAC installation, failure to connect to DIRAC server and failed software installation. If there were no tasks waiting to be processed in the DIRAC WMS that matched the criteria being requested by the agent, then the agent would simply terminate. The application error is a misnomer as it includes errors not only with the LHCb software but also hardware and system problems during the running of the application. The errors while transferring or registering the output data were usually recoverable. In summary, LCG

registered that 69k jobs produced useful output datasets for LHCb but according to the LHCb accounting system there were 81k successful LCG jobs that produced useful data. This is interpreted that some of the LCG aborted jobs did run to completion and some jobs that were marked as not running did actually run unbeknown to the LCG system.

	Jobs(k)	% remaining
Submitted	211	
Cancelled	26	
Remaining	185	100.0
Aborted (not run)	37	20.1
Running	148	79.7
Aborted(run)	34	18.5
Done	113	61.2
Retrieved	113	61.2

Table 5-1: LCG efficiency during LHCb DC'04 production phase

	Jobs(k)	% retrieved
Retrieved	113	100.0
Initialisation error	17	14.9
No job in DIRAC	15	13.1
Application error	2	1.8
Other error	10	9.0
Success	69	61.2
Transfer error	2	1.8
Registration error	1	0.6

Table 5-2: Output sandbox analysis of jobs in status "Done" for LCG

The Data Challenge demonstrated that the concept of light, customizable and simple to deploy DIRAC agents is very effective. Once the agent is installed, it can effectively run as an autonomous operation. The procedure to update or to propagate bug fixes for the DIRAC tools is quick and easy as long as care is taken to ensure the compatibility between DIRAC releases and ongoing operations. During DC04, over 200k DIRAC tasks successfully executed on LCG, corresponding to approximately 60% of the total, with up to 60 different contributing sites and major contributions from CERN and the LHCb proto-Tier1 centres.

To distribute the LHCb software, the installation of the software is triggered by a running job and the distribution contains all the binaries and is independent of the Linux flavour. Nevertheless, new services to keep track of available and obsolete packages and a tool to remove software package should be developed.

The DIRAC system relies on a set of central services. Most of these services were running on the same machine that ended up with a high load and too many processes. With thousands of concurrent jobs running in normal operation, the services are approaching a Denial of Service regime, where you have a slow response and with services stalled.

In the future releases of the DIRAC system, the approach to error handling and reporting to the different services will be improved.

As LCG resources were used for the first time, several areas were identified where improvements should be made. The mechanism for uploading or retrieving OutputSandbox should be improved, in particular to have information about Failed or Aborted jobs. The management of each site should be reviewed to avoid and detect that a misconfigured site becomes a "black-hole". The publication of information about site intervention should be also provided to the Resource Broker or to the Computing Element. In particular, both DIRAC and the LCG need extra protection against external failures, e.g. network or unexpected shutdowns.

The adopted strategy, of submitting resource reservation jobs to LCG that only request a LHCb task once they are successfully running on a WN has proven to be very effective to protect LHCb DIRAC production system against problems with the LCG WMS. This approach allowed effective separation of the resource allocation (that is left to LCG) from the task scheduling (that is handled by DIRAC). Some improvement on the LCG scheduling mechanism has taken place but still further improvements are essential in what concerns CPU and local disk space reservation for the jobs.

Another successful approach has been the inclusion, on the same LCG job, of the simulation task, the upload and the registration (including error recovering and retrial mechanisms) of the produced data. This assures that once the job is finished no further actions are needed. Again this has added extra redundancy against errors on the LCG scheduling (at the retrieval of the OutputSandBox step) that would otherwise have been considered as failed.

Other important lessons are the need for better logging and debugging tools that should allow a more efficient understanding of system misbehaviours, the need for bulk operations for large production activities where thousands of jobs need to be processed everyday, and extreme care on the performance of basic commands that must always return (successfully or not) after a reasonable amount of time (simple edg-job-submit or globus-url-copy commands do, under some circumstances, hang for days until they are killed by the user or system administrator).

Running a production over months has shown that every possible hardware component will eventually fail at some point (from the local disk of a WN to the mirrored load-balanced DB server or a system administrator accidentally hitting a reset button) and all software components must be protected against these problems, retrying on alternate servers when possible or returning meaningful error messages otherwise.

5.2.2. Organised analysis

The stripping process consists in running a DaVinci program that either executes the physics selection for a number of channels or selects events that pass the first two levels of trigger (L0+L1). The former will be run on all signal and background events while the latter will be run on minimum bias events.

The DaVinci applications (including JobOptions files) were packaged as a standard production application such that they can be deployed through the standard DIRAC or LCG software installation procedures. For the handling of the stripping, a database separate from the LHCb Bookkeeping Database (BKDB), called the Processing Database (PDB), was used.

Information was extracted from the BKDB based on queries on the type of data. New files were incrementally added to the PDB, upon the production manager request, and initially marked as "created." This database, is scanned for a given event type with enough data to be stripped. The files are marked as "grouped" and assigned a Group tag. Jobs are then prepared

to run on all files with the same Group tag. The files are then marked as "prepared." The JDL of the job contains the logical file names (LFN) of all selected files and from the list of files a GaudiCatalog, corresponding to those files, was created and was shipped in the jobs' InputSandbox.

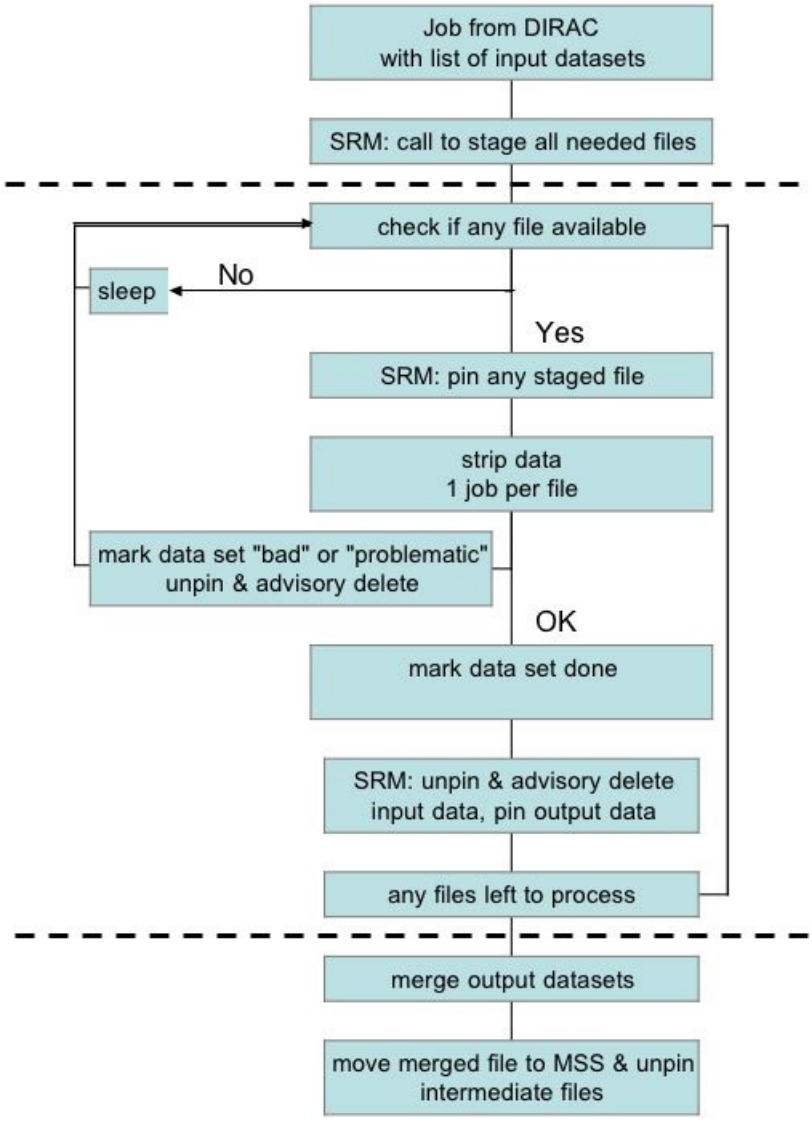


Figure 5-1: Workflow diagram for the staging, stripping and merging process

The stripping process performs the following steps and the workflow is illustrated in Figure 5-1. As the jobs run on a large number of files, a pre-staging takes place in order to take advantage of the optimisation of the underlying staging process. The staging was performed using the technology-neutral SRM interface, and the files should be pinned on the disk pool (see Figure 5-1). The staging happens asynchronously. The checking and stripping steps loop and wait for input files to be available on the staging disk. A DaVinci application is run on the first available file, in a single file processing. Depending on the outcome of DaVinci, the file will be declared "Stripped", "Bad Replica" or "Problematic." The output of the stripping will be a stripped DST file and Event Tag Collection (ETC), all kept on the local disk. A Gaudi job is then run using all stripped DSTs as input and producing a merged stripped DST. This step prepares all necessary BKDB updates as well as PDB updates. It takes care of saving the files on an SE and registering them as replicas into the file catalog(s). The ETCs are also merged, stored and registered.

SRM[85] was used as a technology neutral interface to the mass storage system during this phase of the LHCb data challenge. The original plan was to commence at CERN, CNAF and PIC (CASTOR based sites) before moving to non-CASTOR technologies at other proto-LHCb Tier-1 centres, such as FZK, IN2P3, NIKHEF/SARA and RAL. The SRM interface was installed at CNAF and PIC at the request of LHCb and we were active in aiding debugging these implementations.

The Grid File Access Library (GFAL)[69] APIs were modified for LHCb to allow some of the functionality requirements described above to be available. The motivation of using GFAL was to hide any SRM implementation dependencies, such as the version installed at a site. From these API's LHCb developed a number of simple command line interfaces. In principle the majority of the functionality required by LHCb was described in the SRM (version 1.1) documentation, unfortunately the implementation of the basic SRM interfaces on CASTOR did not match the functional design. Below we describe the missing functionality and a number of ad-hoc solutions used.

The inability to pin/unpin or mark files for garbage collection means it is possible that files for a SRM request are removed from the disk pool before being processed. A number of temporary solutions were considered:

- throttle the rate the jobs were submitted to a site. This would be a large overhead for the production manager and needs detailed knowledge of the implementation of the disk pools at all sites. It also assumes that the pool in use is only available to the production manager; this is not the case. SRM used the default pool assigned to the mapped user in the SRM server.
- Each time a file status is checked, a new SRM request is issued. This protected against a file being "removed" from the disk pool before being processed but it was not clear what the effect had on the staging optimisation. This was the solution adopted.
- use of technology specific commands to (pin and) remove the processed file from disk. This assumes that such commands are available on the worker nodes (not always the case) and an information service that maps a site with a technology.

A problem was found when SRM requested a corrupted (or non-existent) file. Although the stage request was made for all thie files none of the files were returned in a "ready" status. No error was returned by the GFAL/SRM to inform the user there was a problem with the original stage request. This was an implementation problem associated with CASTOR. The only way to avoid this problem is to remove manually every corrupted file as it comes to light or each time a file status is checked issue a new SRM request.

Originally there was no control over the stage pool being used. It is highly desirable to have separate pools for production activities and user analysis jobs to remove any destructive interference. Mapping the production users in a virtual organisation (VO) to a particular user account solved this problem but this required intervention at the LCG system level.

The stripping concept was proven by running on the LXBATCH system at CERN (but with submission directly through DIRAC.) This approach made use of technology (CASTOR) specific stage commands. Over 20 million events were processed through the stripping with over 70 concurrent jobs running on this single site. Work has started to re-use SRM through LCG for this phase.

5.3 Baseline Service Needs

LHCb is contributing to the definition of the Baseline Services to be made available by LCG in readiness for LHC turn on. The LCG Project Execution Board has set up a working group for this purpose. We summarise here the requirements put forward by LHCb in this context.

LHCb expects to leverage from all the developments that were made in the past on its components in distributed computing, in particular DIRAC and GANGA. The baseline for GANGA is that it will use the services provided by DIRAC for job submission. The requirements of GANGA on the DIRAC services are an integral part of the LHCb design. Hence only DIRAC will rely on externally provided Grid services.

5.3.1. Guidelines for services

A distributed computing system relies on several levels of services or components. Depending on the responsibility for setting up the particular services/components, they should or should not be considered as part of the baseline.

At the low level, services will be provided by the site in order to interface to the underlying fabric structure, both storage and CPU. These services are part of the sites local policy e.g. choice of the MSS, of the batch system, VO sharing etc and are not part of the baseline. In the future, network resources might also be managed by the fabric level services provided by the owners of the resources. At a higher level however, VO's need to have the possibility of implementing their specific internal policy e.g. priorities between physics groups, transfer priority for raw data. Again these are not baseline services.

Figure 5-2 shows the breakdown of services proposed by LHCb. Details of these services are described below.

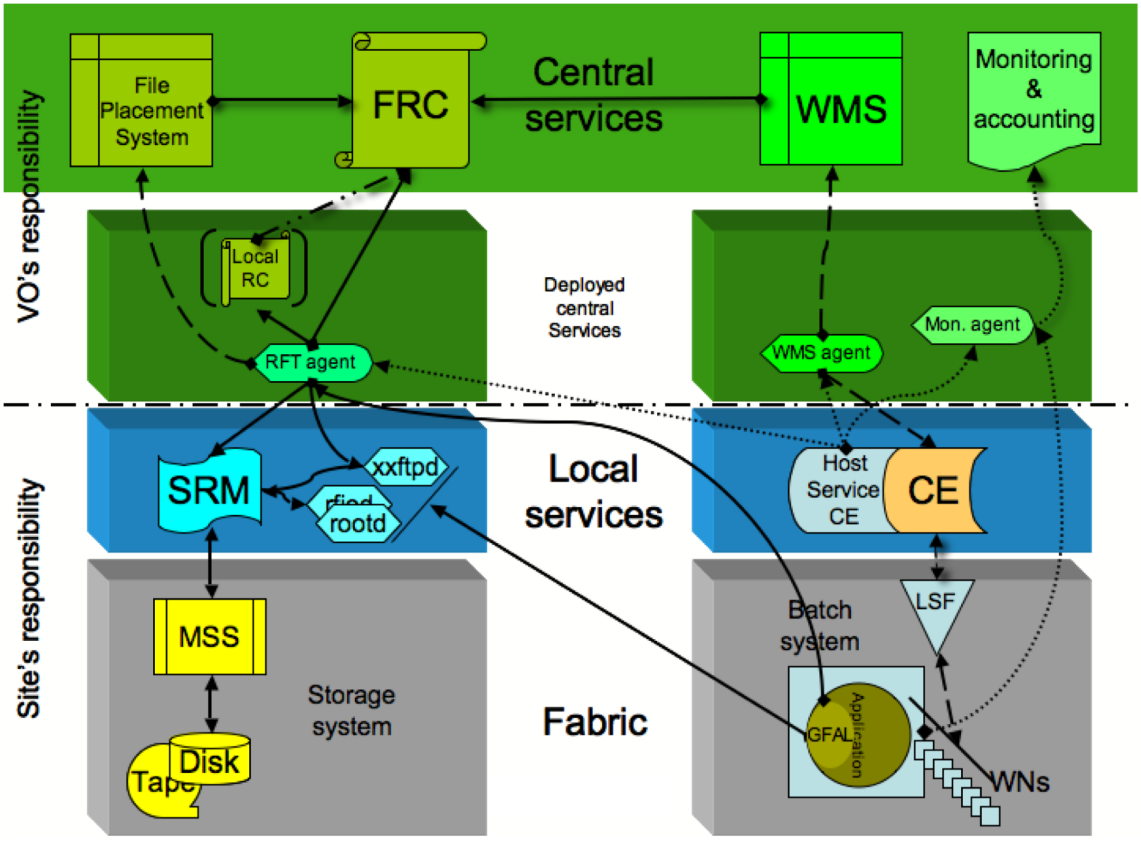


Figure 5-2: Schematic breakdown of services as proposed by LHCb

5.3.2. Data management services

It is necessary to have a standard interface for all storage systems such that jobs can make use of them independently of where they are. We propose that SRM be the standard interface to storage, and hence a Grid Storage Element (SE) should be defined uniquely as an SRM front-end. As a consequence, Physical File Names (PFN) are identified with Site URLs (SURL).

In addition to storage, there is a need for a reliable file transfer system (*fts*). This reliable *fts* will thus permit the transfer between two SRM sites, taking care of the optimisation of the transfer as well as of the recovery in case of failure (e.g. network).

At a higher level, replicas of files need to be registered in a File Catalog (FC). Normal reference to a file by an application is via its Logical File name (LFN). The FC fulfils two main functions:

- Retrieve the SURL of a specific replica of a file at a given SE
- Information provider for the Workload Management System (WMS)

SRM requirements

From the experience of the LHCb DC04, it is clear that the functionality of SRM v1.1 that is currently implemented on most storage systems is not sufficient. Hence we require that the SRM implementations are based on the protocol v2.1. The most urgent features needed in SRM are:

- Directory management
- File management facilities (*get, put…*) with possibilities of define a lifetime for files on disk in case there is a MSS (pinning)
- Space reservation (in particular in case of bulk replication)
- Access control, allowing user files to be stored

These requirements have been submitted to LCG and transmitted to an SRM working group in which LHCb is actively participating.

File Transfer System requirements

As described in section 3.3, the DIRAC system already has capabilities of reliable file transfer. The DIRAC transfer agent uses a local database of transfer requests from a local SE to any external SE(s). In addition it takes care of registration in the LHCb file catalog(s). Currently the DIRAC transfer agent can use several transfer technologies, but Gridftp is the most commonly used. The LCG deployment team has provided a lightweight deployment kit of gridftp in order to use even on non-Grid-aware nodes.

When an *fts* is available and fully operational, LHCb is interested in replacing the current direct use of the gridftp protocol by this *fts*. An implementation with a central request queue as currently implemented in the gLite FTS would be adequate, even if DIRAC keeps the notion of local agents for ensuring file registration.

File Catalogue

The requirements of LHCb in terms of the FC are fulfilled by most current implementations [69],[70]. They all differ by minor details for what concerns the functionality, but we would like to have the opportunity to select the most suitable after appropriate tests of the access patterns implied by our Computing Model. In particular, the scalability properties of the FC services will be carefully studied.

We have developed an LHCb interface that the transfer agent uses and implemented it against several FCs. The aim is to populate all FCs with the few million entries we currently have and continue populating them from the transfer agents. Performance tests will be performed with a central instance of a FC and with local read-only catalog replicas e.g. on Tier-1's. The most efficient and reliable FC will be selected as a first baseline candidate for the LHCb FC.

We do not consider there is a need for standardisation of all VO's on a single implementation provided the FC implements the interfaces needed by the WMS and the transfer service. A good candidate for WMS interface is one of the two currently available in gLite (LFC and FireMan) though only one will be selected.

5.3.3. Workload Management System

A lot of investment has gone into the LHCb production system, as well as into the analysis system (GANGA) for submitting and monitoring jobs through the DIRAC WMS. LHCb would like to keep DIRAC as the baseline for WMS.

The WMS needs interfacing to both the file catalogue (see section 5.3.2) and the Computing Element. The fact that DIRAC needs to interface to the Computing Element implies that some of the agents need to be deployed on the sites. This creates a number of requirements that are described below.

Computing Element requirements

The definition adopted of a CE is that of a service implementing a standard interface to the batch system serving the underlying fabric. Jobs will be submitted, controlled and monitored by the local DIRAC agent through this interface. Hence the following capabilities need to be implemented:

- Job submission and control, including setting CPU time limit.
- Proper authentication/authorisation: the user credentials provided by the DIRAC agent should be used to allow jobs to be submitted with a mapped local userid.
- Batch system query: the DIRAC agent needs to have the possibility to query the batch system about its current load for the specific VO. Depending on the CPU sharing policy defined by the site, this may lead to fuzzy information, that the agent should however use to determine if it is worthwhile requesting a job of a given type to the central WMS queue.

Hosting CE

In order to be able to run local agents on the sites, we need to be able to deploy them on local resources at each site. The deployment is under the LHCb responsibility. Deployed agents will run in user space without any particular privilege. However proper authorisation with a VO administrator role would be required for any action to be taken on the agents (launching, stopping, downloading).

In case agents need a particular infrastructure (e.g. local FC's), this infrastructure needs to be negotiated with the resource providers (e.g. if a specific database service is required). Similarly, the local storage on the node on which agents run will have to be negotiated.

We believe that a specialised instance of a CE limited to specific Virtual Organisation Membership (VOMS) roles and giving access to its local CPU would be adequate provided it can be accessed from outside the site. The deployed agents would run under the VO responsibility and not require any particular intervention from the site besides regular fabric maintenance and survey. The VO would take responsibility for keeping the agents running.

The agents do not require incoming connectivity as they do not provide services to outside the site. The hosting node however needs outgoing connectivity in order to contact the central WMS, file catalogues, monitoring central services etc.

For sites where a hosting CE would not be available, LHCb envisages to use, as it currently does on the LCG, pilot-agents submitted through a third party WMS (e.g. gLite RB) to the sites. This is in particular valid for sites not connected to LHCb formally but which would grant resources to LHCb. It can also be applied to Grids not directly part of the LCG infrastructure. In this specific case, specific issues of authentication/authorisation need to be addressed, in order for the job to be accounted to the actual owner of the job that is running, which could differ from the submitter of the pilot-agent.

Chapter 6 Organisation, Planning & Responsibility

This chapter addresses the organisational structure of the computing within LHCb and the breakdown of responsibilities for the maintenance for some of the core services and packages within the core computing and software, CCS. Manpower to deliver the CCS is party to an addendum to the experiment's memorandum of understanding for maintenance and operation. Information is also presented on the planning schedule. Hardware costs associated with providing the necessary computing resources are not included, as these resources will be provided through the LCG and with the relevant computing centres being party to the LCG memorandum of understanding.

6.1 Organisation

The organisation of computing within the LHCb Collaboration is shown in Figure 6-1. The parts of the organisation concerned with the provision of CCS are indicated in the dashed box. The CCS is seen as the provision of the software framework; tools for distributed computing; coordination of the computing resources; organisation of the event processing of both real and simulated events, and the integration of algorithms (both global and sub-detector specific) in the software framework. The LHCb CCS also provides global reconstruction algorithms that will run in the online & offline environment, as well as the coordination of the sub-detector software.

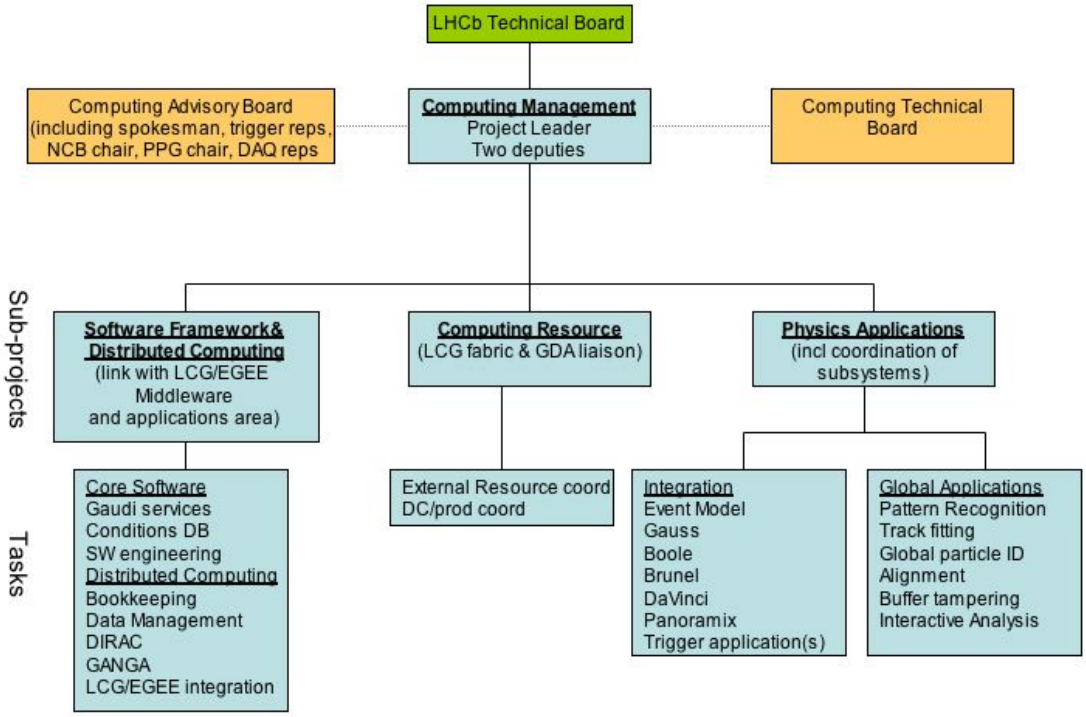


Figure 6-1: Schematic of the organisation of the LHCb computing project

The Computing Advisory Board advises the Computing project and the collaboration management on the priorities and the needs of the LHCb collaboration with respect to computing and software matters. It will address the requirements of the L1/HLT, DAQ/control and computing sub-projects as well as the physics needs for the provision of computing applications and services. The priority of these needs and requirements will be assessed in the board in order to make recommendations to the computing management. It will set the high level deliverables and milestones for the LHCb computing project and receive status reports on progress towards these deliverables. The requirements of the sub-detectors are addressed directly within the computing project.

The Computing Technical Board oversees all aspects of the Computing project, including time schedules, planning and enacting the functionality as requested by the Computing Advisory Board. It serves as an advisory body for the LHCb computing management. The board ensures that there is coordination across the computing sub-projects and to external computing bodies such as the LCG project and will monitor internal milestones. Major technical decisions are discussed in the board. The board advises the computing management on technical decisions but it is the responsibility of the management to make the final decision.

In addition to the boards shown in Figure 6-1, there is the National Computing Board, NCB. The NCB disseminates information with regards computing planning and needs within LHCb. It serves as a forum to discuss common approaches and problems within home institutes/countries. It advises the LHCb collaboration board, LHCb management and computing project on external computing issues, including computing resource requirements and manpower for the CCS. The NCB representatives are responsible for collating and disseminating all relevant information for discussion from and to all institutes they represent.

6.2 Tasks and Institutes

The Core Computing and Software (CCS) is defined as the development and maintenance of the software framework, application integration, global reconstruction software, software infrastructure, visualisation, production and analysis tools, and management and interfacing to the Grid and LCG software. A full list of activities is given in Table 6-1. Two generic areas are identified:

- Development and maintenance of major software projects e.g. Gaudi, DIRAC, which carry long-term responsibility.
- Contribution to generic common support tasks, e.g. webmaster, production manager, software librarian, where the commitment will be provided with individual expertise.

The institutes listed in Table 6-2 are currently actively participating in the activities of the CCS.

Project management

- Computing project leader
- Deputy Computing project leader(s)

Core Software

- Gaudi services
- Conditions DB
- Software Engineering

Distributed Computing

- Bookkeeping
- Data Management
- DIRAC
- GANGA
- LCG/EGEE integration

Production

Integration

- Event model
- Gauss
- Boole
- Brunel
- DaVinci
- Trigger applications
- Panoramix

Global Applications

- Track pattern Recognition
- Track fitting
- Global Particle ID
- Global Alignment
- Buffer Tampering
- Interactive Analysis

Table 6-1: List of high level tasks within the LHCb CCS

Country	Institute
Brazil	Brazilian Centre fo Research in Physics (CBPF), Rio de Janerio
	Universidade Federal do Rio de Janeiro (UFRJ)
CERN	CERN
France	Centre de Physique des Particules de Marseille, CNRS/IN2P3 and Université de la Méditerranée
	LAL-Orsay, IN2P3-CNRS, Université de Paris-Sud
Germany	Max-Planck-Institut für Kernphysik (MPI) Heidelberg
Italy	Univ. of Bologna, INFN
	Univ. of Milano, INFN
	INFN CERN fellows
Netherlands	NIKHEF
Poland	Institute of Nuclear Physics & University of Mining and Metallurgy, Krakow
Russia	Institute for Theoretical and Experimental Physics (ITEP) - Moscow
Spain	Universidad de Barcelona
	Universidad de Santiago de Compostela
Switzerland	Ecole Polytechnique Féderale Lausanne
	Universität Zürich
UK	University of Bristol
	University of Cambridge
	University of Edinburgh
	University of Glasgow
	University of Liverpool
	Imperial College, University of London
	University of Oxford
	Rutherford Appleton Laboratory

Table 6-2: List of institutes involved in the CCS activities

6.3 Milestones and Planning

Milestone	Due date	External Dependencies
2005		
Example of sub-detector alignment	September	
Analysis at all Tier-1's	November	
End of review of reconstruction software, including event model	December	
2006		
Final alignment strategy	March	
Start data processing phase of DC'06 (i) Distribution of RAW data from CERN (ii) Reconstruction/stripping at Tier-1's includind CERN (iii)DST distribution to CERN & other Tier-1's	May	New version of SRM at all Tier-1's LCG services/tools from baseline service working group deployed at all Tier-1's
Alignment/calibration challenge – participation of all Tier-1's (i) Align/calibrate detector (ii) Distribute DB slice – synchronize remote DB's (iii)Reconstruct data	October	DB service to support COOL at all Tier-1's
2007		
Permanent Monte Carlo production mode ready for data taking	January	
Production system and software ready for data taking	April	

Table 6-3: High-level milestones for the computing project

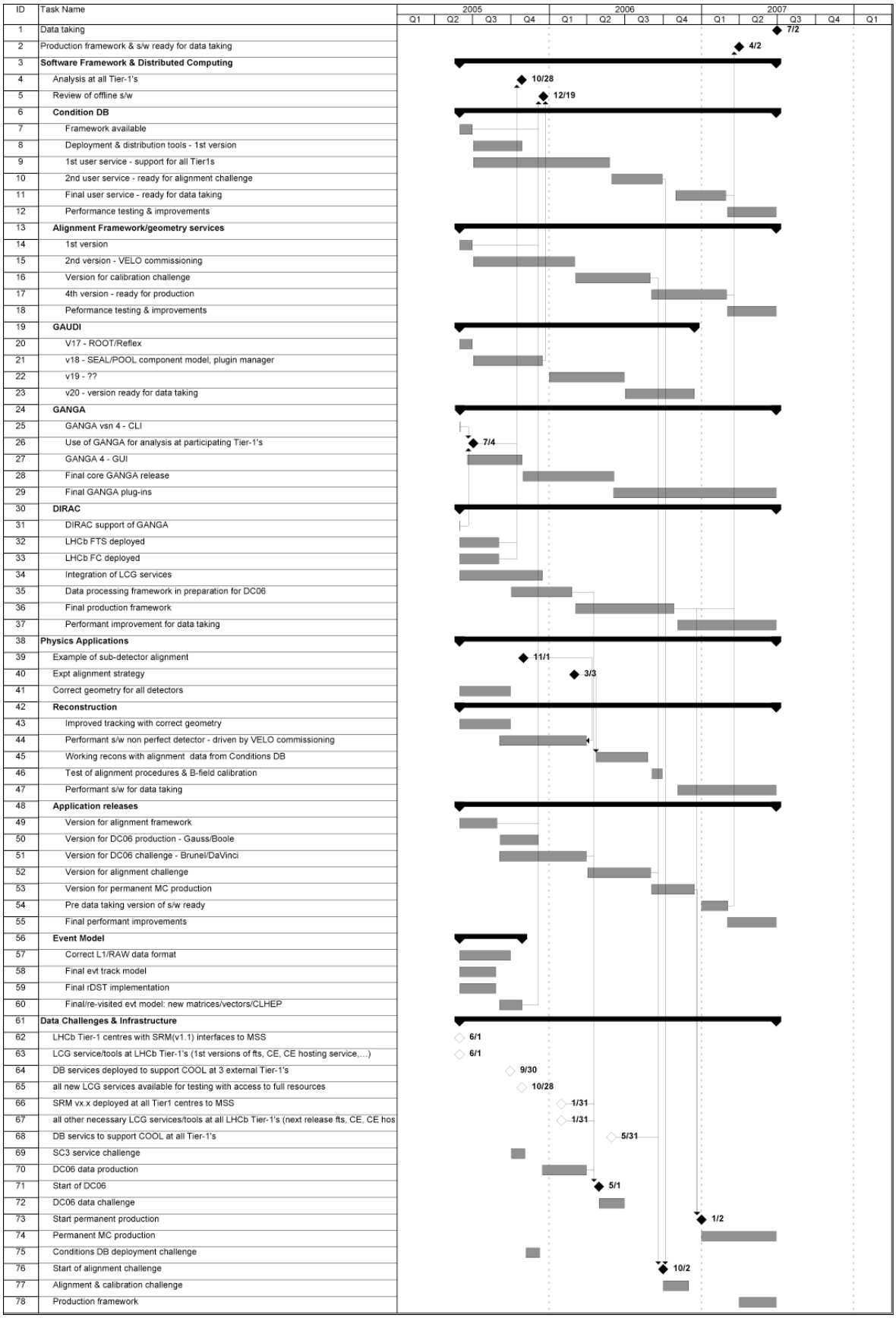


Figure 6-2: Project schedule for the computing project including data challenges. The open diamonds are external milestones

Glossary

AFS	Andrews File System
AIDA	Abstract Interfaces for Data Analysis – part of the LCG applications area
AliEn	Alice Environment
API	A Programming Interface
Bender	A Python based physics analysis application for LHCb
BKDB	BookKeeping DataBase
Boole	LHCb digitisation application
BQS	Batch Queueing System
Brunel	LHCb reconstruction application
CCS	Core Computing and Software
CE	Computing Element
Clarens	Grid-Enabled Web Services Framework
CLHEP	A C++ class library for high energy physics
CLIP	Command Line Interface in Python
CMT	Configuration Management Tool
Condor	A batch system from Univ of Wisconsin
COOL	LCG Conditions Database Project, subproject of POOL
DaVinci	LHCb analysis application
DC03	LHCb data challenge 2003
DC04	LHCb data challenge 2004
DIM	Distributed Information Manager – a communication system for mixed environments
DIRAC	Distributed Infrastructure with Remote Agent Control - LHCb workload management system
DMS	Data Management Service
DST	Data summary tape - output of the Brunel application
ECS	Experiment Control System
EDG	European Data Grid
EGEE	Enabling grids for E-science
ETC	Event TAG Collection
FC	File Catalog
FiReMan	EGEE file catalog
FTS	File Transfer System - EGEE file transfer system
GANGA	Gaudi and AtheNa Grid Alliance – LHCb Grid user interface for analysis
Gaucho	software package to allow the control & monitoring by the ECS of the L1 and HLT algorithms
Gaudi	LHCb Data Processing Applications Framework
Gauss	LHCb detector simulation application
GDA	Grid Deployment Area

GENSER	LCG generator services subproject
GFAL	Grid File Access Library
gLite	Lightweight Middleware for Grid Computing - EGEE middleware
GridSite	A toolkit for Grid credentials, GACL access control lists and HTTP(S) protocol operations
GSI	Grid Security Infrastructure
GUI	Graphical user Interface
HippoDraw	An interactive data analysis environment from SLAC
HLT	High Level Trigger
i/o	input-output
IM	Instant Messaging
IOV	Intervals Of Validity
Jabber	Instant Messaging service
JDL	Job Definition Language
L1	Level-1 trigger
LCG	LHC Computing Grid
LFC	LHC File Catalog
LFN	Logical FileName
LoKi	High level analysis toolkit for LHCb
LSF	A batch system from Platform
MSS	Mass Storage System
NCB	National Computing Board
NQS	Network Queueing System - a batch system
OnX	A software package for interactivity
Open Inventor	A OO 3d toolkit for graphics built on top of OpenGL
OpenGL	OpenGL is an environment for developing portable, interactive 2D and 3D graphics applications.
Pacman	Software package to install, configure and setup software
PBS	Portable Batch System
PFN	Physical FileName
POOL	Pool Of persistent Objects for LHC – LCG persistency framework used by LHCb
PPG	Physics Planning Group
PVSS	Process Visualiisation and Control System used by the ECS
Python	A OO scripting language
RB	Resource Broker
RDB	Request DataBase
RDBMS	Relational DataBase Management System
rDST	Reduced DST
RFTS	Reliable File Transfer
SE	Storage Element
SEAL	LCG Core Libraries and Services Project
SFC	Subfarm controller

SOAP	Simple Object Access Protocol – a lightweight protocol for exchange of information in a decentralized, distributed environment
SQL	Standard Query Language
SRM	Storage Resource Manager
TES	Transient Event Store
TomCat	Open-source implementation of Java Servlet and JavaServer Pages
VO	Virtual Organisation
VOMS	Virtual Organization Membership Service
WMS	Workload Management Service
WN	Worker Node
XML-RPC	A set of implementations that allow software running on disparate operating systems, running in different environments to make procedure calls over the Internet

References

[1] LHCb Collaboration, S.Amato et al., Technical Proposal, CERN-LHCC/98-4.

[2] LHCb Collaboration, R.Antunes Nobrega et al., LHCb Reoptimized Detector Design and Performance TDR, CERN-LHCC/2003-030

[3] LHCb Collaboration, R.Antunes Nobrega et al., LHCb Trigger TDR, CERN-LHCC/2003-031

[4] LHC Computing Grid, TDR, CERN-LHCC/2005-xxx

[5] G. Barrand et al., "Gaudi – a software architecture and framework for building HEP data processing applications", *Comp.Phys.Comm.*140 (2001).

[6] W. Amstrong et al., "ATLAS Technical Proposal", *CERN-LHCC-94-43,* 1994.

[7] R. Brun, "ZEBRA Reference Manual", *Program Library Q100, CERN,* 1991.

[8] R. Brun, F. Rademakers, "ROOT – An object oriented analysis framework", *Nucl.Inst.Meth. in Phys. Res A*, 389 (1997) 81.

[9] D. Duelmann et al., "The LCG POOL development and production experience", *Proceedings of IEEE-NSS Rome2004*, Rome, October 2004.

[10] M. Cattaneo et al., "Event data definition in LHCb", *Proceedings of CHEP03,* San Diego, March 2003.

[11] Doxygen, http://www.doxygen.org

[12] J. Generowicz et al., "SEAL: Common core libraries and services for LHC applications", *Proceedings of CHEP03,* San Diego, March 2003.

[13] M. Cattaneo et al., The new LHCb Event Data Model, LHCb 2001-142.

[14] O. Callot, "A new implementation of the relations: the Linker objects", LHCb-2004-007.
O. Callot, "Usage of the Linker Classes", LHCb-2005-018.

[15] I. Belyaev, "Relations – Generic external relations in Gaudi", LHCb-2005-005.

[16] A. Valassi et al., "LCG Conditions Database Project Overview", *Proceedings of CHEP04*, Interlaken, Switzerland, September 2004.

[17] COOL project, http://lcgapp.cern.ch/project/CondDB/

[18] ORACLE, http://www.ORACLE.com

[19] MySQL, http://www.mysql.com

[20] SQLite, http://www.sqlite.org

[21] I. Belyeav et al., "Simulation application for the LHCb experiment", *Proceedings of CHEP03,* San Diego, March 2003,
http://lhcb-comp.web.cern.ch/lhcb-comp/Simulation/Gauss.pdf

[22] T. Sjostrand et al., "High-Energy physics event generation with Pythia", *Comp.Phys.Comm.*, 135 (2001) 238.

[23] D. Lange et al., "The EvtGen particle decay simulation package", *Nucl.Inst.Meth.A* 462 (2001) 152.

[24] P. Bartalini et al., "Tuning of multiple interactions generated by PYTHIA", LHCb-99-028,
N. Brook et al., "Studies on the tuning of Minimum Bias events at LHCb", LHCb 2005-023,
N. Brook et al., "Generator settings for the LHCb Data Challenge 2004", LHCb 2005-022.

[25] P. Bartalini et al., "LCG Generator", *Proceedings of Computing in High Energy and Nuclear Physics*, CHEP 04, Interlaken, Switzerland 2004.
GENSER LCG project, http://lcgapp.cern.ch/project/simu/generator

[26] M. Dobbs et al., "The HepMC C++ Monte Carlo event record for High Energy Physics", *Comp.Phys.Comm.* 134 (2001) 41.

[27] S.Agostinelli et al., "GEANT4 – a simulation toolkit", *Nucl.Inst.Meth. A* 506 (2003) 250.

[28] I. Belyaev et al., "Integration of Geant4 with the Gaudi framework", *Proceedings of CHEP2001,* Beijing, September 2001.

[29] R. L. Graun and D. L. Smith, NIM 80 (1970), 239.

[30] E. Aguilo at al., "Updated results on the Monte Carlo Simulation of the SPD/PRS Pulseshape", LHCb-2005-017.

[31] T.Bellanuto et al., Nucl.Inst.Meth. A 519 (2004), 493-507.

[32] M. Needham, "New data model, digitization and reconstruction algorithms for the inner tracker", LHCb-2002-030.

[33] M. Adinolfi et al., "A simulation study of the LHCb RICH performance", LHCb-2000-066.

[34] G. Martellotti et al., "Muon System Digitization", LHCb-2004-063.

[35] J. Nardulli and J. Van Tilburg, "Outer Tracker Event Data Model", LHCb-2005-004.

[36] O. Callot et al., "Raw-data Format", EDMS 565851.1

[37] J. H. Lopes, "Software Implementation of the Raw Data Buffers for the L1 and High Level Triggers of LHCb", LHCb-2003-152.

[38] LHCb Online System TDR, CERN-LHCC-2001-040.

[39] C. Gaspar, M. Dönszelmann and Ph. Charpentier, Comp. Phys. Comm. 140 (2001) 102-109.

[40] PVSS [online] http://www.pvss.com/english/index.htm .

[41] M. Witek, "Execution time optimisation of L1 algorithm", LHCb 2003-061.

[42] P.R. Barbosa Marinho et al., "LHCb online TDR", CERN-LHCC-2001-040.

[43] M. Witek, "VELO-TT matching and momentum determination at L1 trigger", LHCb 2003-060.

[44] O. Callot, "Velo Tracking for the High Level Trigger", LHCb 2003-027.

[45] O. Callot, "Online Pattern Recognition", LHCb 2004-094.

[46] P. Koppenburg and L. Fernandez, "Exclusive selections in the High Level Trigger", *LHCb-2005-015.*

[47] P. Koppenburg and L. Fernandez, "DaVinci for busy people, Reference to the generic selection algorithms and tools", *LHCb-2005-016*.

[48] C. Jacoby and P. Igo-Kemenes, "The new efficiency algorithms", *LHCb-2004-080*.

[49] T. Glebe, "Pattern - High Level Tools for Data Analysis", HERA-B-2002-002.

[50] T. Glebe, "GCombiner 1.0", HERA-B-2001-001.

[51] A. Alexandrescu, "Modern C++ Design: Generic Programming and Design Patterns Applied", Addison-Wesley, 2001, ISBN 0-201-70431-5.

[52] CLHEP: http://cern.ch/wwwasd/lhc++/clhep

[53] BOOST: http://www.boost.org

[54] I. Belyaev et al., Python-based Physics Analysis Environment for LHCb, LHCb-2004-089.

[55] Python, http://www.python.org/

[56] AIDA, http://aida.freehep.org/

[57] HippoDraw, http://www.slac.stanford.edu/grp/ek/hippodraw/

[58] http://www.lal.in2p3.fr/SI/OOnX/v10/welcome.html

[59] http://oss.sgi.com/projects/inventor/

[60] Configuration Management Tool – CMT, http://www.cmtsite.org/

[61] Pacman, http://physics.bu.edu/~youssef/pacman/

[62] S. Paterson, "Software Distribution For LHCb Using Pacman", LHCb-2004-066

[63] P. Buncic et al., Report from the LCG ARDA RTAG, CERN-LCG-2003-033

[64] MonALISA project page, http://monalisa.cacr.caltech.edu/

[65] R. Raman, M. Levy and M. Solomon, "Policy Driven Heterogeneous Resource Co-Allocation with Gangmatching" *Proceedings of the Twelfth IEEE International Symposium on High-Performance Distributed Computing*, June, 2003, Seattle, WA. and http://www.cs.wisc.edu/condor/classad/

[66] Jabber Software Foundation, http://www.jabber.org/

[67] C. Cioffi et al, "File–Metadata Managment System For The LHCb Experiment", " *Proceedings of CHEP04*, Interlaken, Switzerland, September 2004

[68] The AliEn project, http://alien.cern.ch

[69] J-P Baud & J. Casey, "Evolution of LCG-2 Data Management" *Proceedings of CHEP04*, Interlaken, Switzerland, September 2004

[70] FiReMan catalog, http://egee-jra1-dm.web.cern.ch/egee-jra1-dm/

[71] XML-RPC, http://www.xml-rpc.org

[72] Runit Service supervision toolkit, http://smarden.org/runit/

[73] A.T. Doyle, S.L. Lloyd and A. McNab, "Gridsite, gacl and slashgrid: Giving Grid Security to Web and File Applications", Proc of UK e-science All Hands Conference 2002, Sept 2002.

[74] C.D. Steenberg *et al.*, "The Clarens Web Servbice Architecture", Proc of CHEP 2003, March 2003.

[75] J. Closier *et al.*, "Results of the LHCb Data Challenge 2004", Proc. Of CHEP2004, Sept 2004.

[76] A. Tsaregorodstev *et al.*, "DIRAC – Distributed Infrastructure with Remote Agent Control", Proc of CHEP2003, March 2003.

[77] G.Kuznetsov, The DIRAC console, LHCb note in preparation.

[78] U.Egede, "LHCb use cases for distributed analysis", LHCb 2005-027

[79] D. Adams et al. "GANGA 4 architecture", http://ganga.web.cern.ch/ganga/documents/pdf/Ganga4Architecture.pdf

[80] ARDA Metadata Catalogue Project, http://project-arda-dev.web.cern.ch/project-arda-dev/metadata

[81] For example, J. Hammer, H. Garcia-Molina, J. Widom, W. J. Labio, Y. Zhuge. "The Stanford Data Warehousing Project." IEEE Data Engineering Bulletin, June 1995 or http://www-db.stanford.edu/pub/papers/whips-overview.ps

[82] Simple Object Access Protocol (SOAP) 1.1, http://www.w3.org/TR/soap

[83] Apache Jakarta Tomcat servlet container, http://jakarta.apache.org/tomcat

[84] F. Donno *et al.*, "Experiment Software Installation on LCG-1", CERN-LCG-GDEIS-412781.

[85] SRM working group, http://sdm.lbl.gov/srm-wg/