Extension of the DIRAC Workload Management System to allow use of distributed Windows resources

Y Y Li[1], K Harrison[1], V Lyutsarev[2], M A Parker[1] and A Tsaregorodtsev[3]

[1] HEP Group, Cavendish Laboratory, University of Cambridge, JJ Thomson Avenue, Cambridge, CB3 0HE, United Kingdom

[2] Microsoft Research Cambridge, Roger Needham Building, 7 JJ Thomson Avenue, Cambridge, CB3 0FB, United Kingdom

[3] Centre de Physique des Particles de Marseille, CNRS-IN2P3, 163, avenue de Luminy Case 902, 13288 Marseille cedex 09, France

Abstract. The DIRAC Workload Management System of the LHCb experiment allows coordinated use of globally distributed computing power and data storage. The system was initially deployed on the Linux platforms, where it has been used very successfully both for collaboration-wide production activities and for single-user physics studies. To increase the resources available to LHCb, DIRAC has been extended so that it also allows use of Microsoft Windows machines. As DIRAC is mostly written in Python, a large part of the code base was already platform independent, but Windows-specific solutions have had to be found in areas such as certificate-based authentication and secure file transfers, where .NetGridFTP has been used. In addition, new code has been written to deal with the way that jobs are run and monitored under Windows, enabling interaction with Microsoft Windows Compute Cluster Server 2003 on sets of machines were this is available. The result is a system that allows users transparent access to Linux and Windows distributed resources. This paper gives details of the Windows-specific developments for DIRAC; outlines the experience gained in deploying the system at a number of sites, and reports on the performance achieved running the LHCb data-processing applications.

1. Introduction

DIRAC (Distributed Infrastructure with Remote Agent Control) is LHCb's Workload and Data Management System (WMS, DMS) that allows coordinated use of globally distributed computing resources (Grid). It will be required to process petabytes per year of raw and simulated LHCb data, once the LHC is in operation. DIRAC's main architectural aim is to be a lightweight robust system, which is able to combine LHCb specific and general-purpose components in a distributed computing environment, is also flexible and easy to deploy on various platforms.

To achieve these goals, DIRAC uses a services oriented architecture [1], exploiting both LHCb and LHC Computing Grid (LCG) services. This is then combined with one of the core components of the system, the DIRAC Agent to pull together the distributed resources. The DIRAC Client, completes the system by providing a user interface to the services. This also provides the support of the GANGA Grid front-end [2] for distributed analysis [3].

2. Modifying DIRAC for use under Microsoft Windows

In order to achieve the goal of allowing Windows machines to be incorporated into the system, only areas of DIRAC visible to the user and the resource need to be considered, with the main services based on Linux servers. These are the two areas of the Client and the Agents. Extension of the Client, the interface between the user and the server allows for job submissions from Windows, while implementing the Agent side allows for the use of Windows CPU resources.

The porting of DIRAC to Windows involves two types of changes, the replacement of Linux specific python code, and providing Windows equivalent solutions where platform independent solutions are not possible, e.g. pre-compiled libraries, secure file transfer protocols. This section focuses on some of the main changes made.

2.1. Replacing Linux-specific Code

Although Python is largely a platform independent language, it does contain functions that allow access to platform dependent commands, for example os.system, os.fork. In order to make the DIRAC code as portable as possible, use of such functions has been avoided wherever a platform-neutral solution was available. Otherwise, Windows specific coding, equivalent to the Linux code, is provided alongside the original.

2.2. DISET Modifications

The DISET (DIRAC Security Transport) module [4] is the underlying security module of DIRAC, providing grid authentication between the DIRAC components. DISET is built on OpenSSL and a modified version of pyOpenSSL, incorporating Grid Security Infrastructure (GSI) support. A modified version of DISET combined with the DIRAC API, provides full Client support under Windows. With the new OpenSSL and pyOpenSSL Windows libraries placed alongside the Linux libraries.

The platform being used is identified during DIRAC installation, allowing the appropriate libraries to be loaded at runtime. The proxy certificate can then be generated with OpenSSL tools using the same command under both platforms, validation checks of the generated proxy has been made though cross-platform submissions.

2.3. Data Transfer

For successful and meaningful distributed computing, the problem of access to data storage must be addressed [5, 6], especially in the case of the data volumes LHCb will produce and analyze. The model for LHCb (and other LHC experiments) is to match jobs to resources where the data is available locally. However in the initial phases of incorporating Windows machines into the system, there is unlikely to be any local data. So analysis jobs will need to rely on the transfer of data to the resource. The same procedure is also required for a secure data transfer of the results back to the storage if requested by the user.

Storage resources, like CPU resources, can also be distributed. Thus the physical data location has to be first obtained (e.g. via a file catalogue) before the secure transfer can take place.

2.3.1. *LCG File Catalogue (LFC).* The LFC [7] acts as a central catalogue for DIRAC that contains a set of listed known data in the distributed storages. Under Linux the LFC is contacted directly using a proprietary security protocol. Once accessed, the catalogue provides a one-to-many mapping of Physical File Names (PFNs) for each Logical File Name (LFN), depending on how many replicas of the file exist. However the LFC protocol is not supported on the Windows platform. A possible solution is to use a DISET portal [4], combined with the Proxy LFC Server. The user's proxy is used in the protocol to contact the Proxy LFC Server. The LFC trusted Proxy LFC Server will then pass the user's proxy credentials to the LFC. Once accessed the user's credentials is used to gain authorization for the LFC operations.

When the replica list of the LFN is obtained, each replica is tried for download until a copy of the file is retrieved to the local resource. A simple local XML based catalogue is created, which contains the mapping between the LFNs required by the job, and the new local PFNs. This can then be used on the worker node to locate the data.

2.3.2. *GridFTP.* GridFTP is a high-performance, secure, reliable data-transfer protocol optimized for high-bandwidth wide-area networks. The protocol is based on the internet File Transfer Protocol (FTP), but extends it with facilities such as multistreamed transfers, and Globus Security Infrastructure (GSI). Two current implementations of the protocol are available under Microsoft Windows, both based on the Microsoft Windows .NET platform, MyCoG.NET [8], and dotNetGridFTP [9]. MyCoG.NET is the implementation of a commodity grid toolkit on .NET. It consists of a Grid Security module supporting GSSAIP and GridFTPClient, GramClient and Proxy classes. With an already existing DISET security module in place for DIRAC, the lighter implementation of a standalone GridFTP client from dotNetGridFTP has been chosen.

The Windows specific code is incorporated into the existing gridFTP script alongside Linux and LCG specific codes.

2.4. Microsoft Windows Compute Cluster Compute Element Backend

DIRAC offers interfaces to various compute backends including Inprocess (standalone machine), LCG, Condor etc.... With the Inprocess backend of Windows machines ported. The extension of the DIRAC system to incorporate the Microsoft Windows Compute Cluster has also been investigated. This section describes the experience gained in the development of this new Compute Element backend.

2.4.1. *Compute Cluster Setup.* The Microsoft Windows Compute Cluster Server 2003 consists of two components, the Microsoft Windows Server 2003, and the Compute Cluster Pack. The latter contains all of the supporting software needed to create and configure the cluster nodes, and manage the infrastructure. The head node of the cluster controls the management services of the cluster and acts as a gateway, with the following services installed:

- Compute Cluster Management Service, which provides the overall monitoring and configuration of the system;
- Compute Cluster Message Passing Interface (MPI);
- Compute Cluster Manager Service, which allows the head node to function also as a worker node;
- Compute Cluster Scheduler Service, which provides the internal cluster job scheduling, node allocations, and job execution.

Services on the worker nodes are limited to the top three services above. The Management Service communicates to the same service on the head node, while the Manager Service communicates with the Scheduler Service on the head node, permitting only the head node to submit jobs.

2.4.2. *Microsoft Windows Compute Cluster Backend.* Following the modular structure of the Compute Element backend scripts in DIRAC, the implementation of the Compute Cluster has been confined to the new module, with the standard interface backend.

Figure 1 shows the overview of the DIRAC architecture on the Windows Compute Cluster. The main DRIAC installation is based on the head node. Agents are initialized from the head node to contact the Job Matcher to retrieve site capacity matched jobs. Once a job is retrieved, it checks and installs, if necessary any required software from the Software Repository. The Job Wrapper is then created, and sent to the internal Compute Cluster Scheduler Service for processing. The shared DIRAC installation and applications are seen on the worker nodes as a vertical drive, allowing easy management of the DIRAC installation.

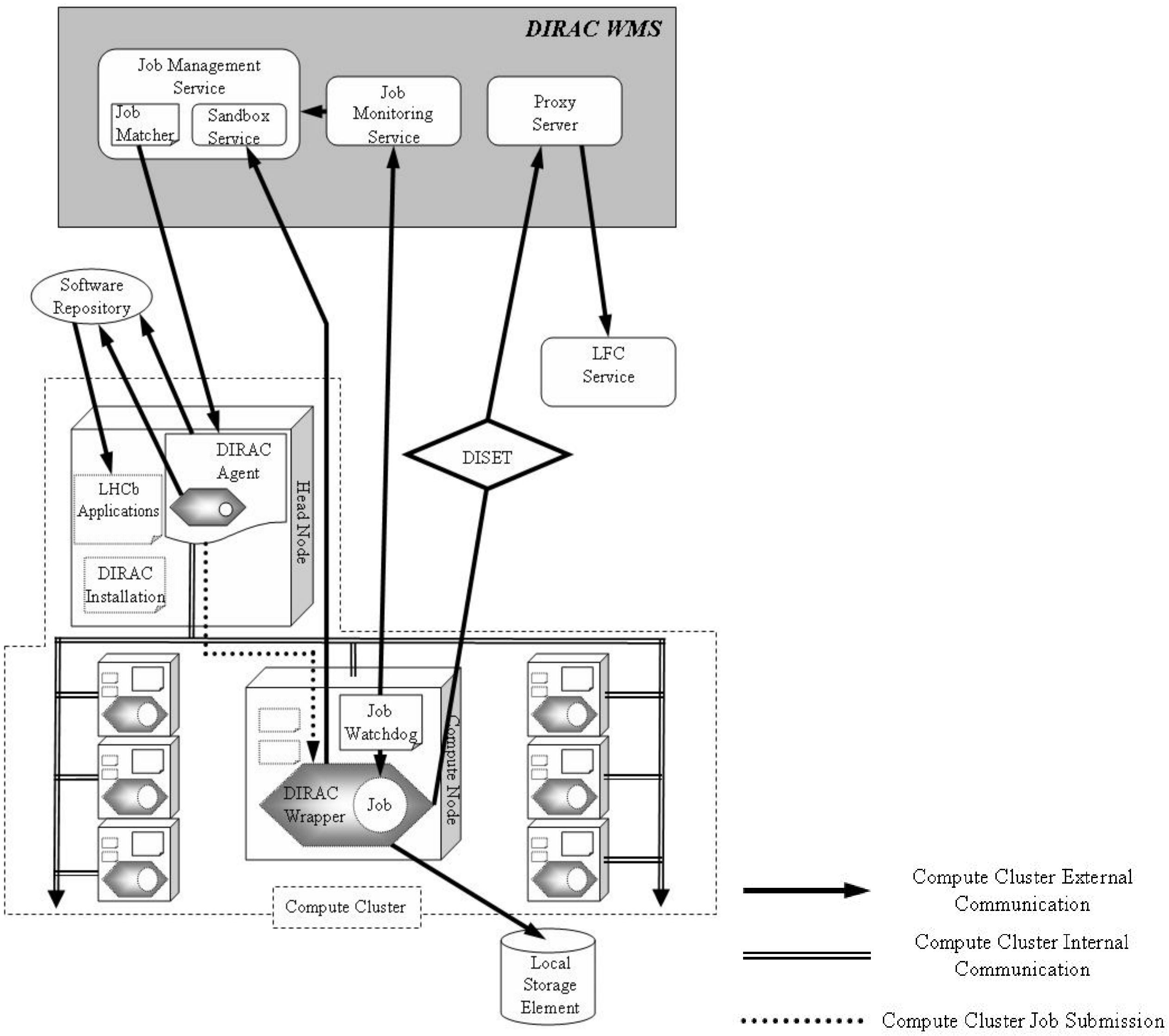

Figure 1. Architecture of Compute Element for compute cluster

The first challenge for sharing the same DIRAC installation arises in the Job Wrapper, which must perform the correct environment setup for the worker node allocated. This is achieved using the Universal Naming Convention (UNC) of Windows, which allows shared directories to be located from each node via a common path. On the worker node, this is assigned to a pre-defined virtual drive letter, where all subsequent tasks are performed. This step is added due to the limitation of UNC to start sub-processes from the command line interface.

The last step of preparation before executing the job is to obtain the input data, either stored locally on the cluster, or on an external storage disk via another virtual drive mapping.

When a job is started on the worker node, a watchdog is also launched to report the job heartbeat to the Monitoring Service. In this case the DIRAC Job Monitoring Service is contacted directly from the worker node. The Compute Cluster backend also provides an interface to the internal Compute Cluster Scheduling Service, which is queried by the Agent running on the head node for a overall resource capacity status report.

2.4.3. *Windows Wrapping.* The code modifications to allow use of DIRAC under Windows can be grouped into three classes;

- Platform specific libraries and binaries – OpenSSL, pyOpenSSL, and .NetGridFTP require Windows libraries, which are included in the DIRAC distribution. The platform specific library import is determined by the initiation of DIRAC through the environment variable *DIRAC_PLATFORM*.
- Additional Windows specific code – This includes the addition of the Compute Cluster backend, and the addition of Windows .bat files to match corresponding Linux shell scripts.
- Minor Python code modifications.

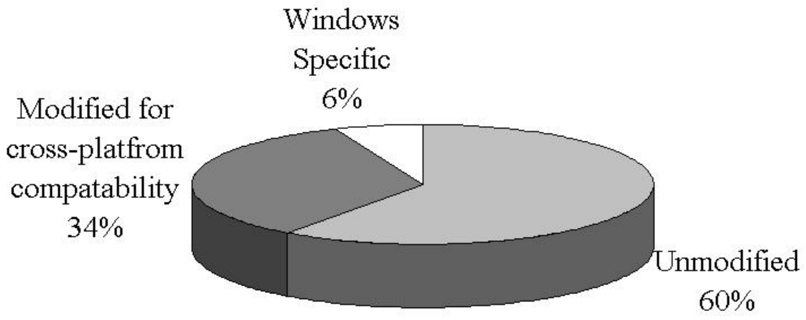

Figure 2. Overview of DIRAC packages after modifications for compatibility with Windows

Seven out of thirteen DIRAC packages are required for the setting up of DIRAC and for running of the Client and Agent in the Inprocess backend mode. Of the required packages 34% of the code has been modified for cross-platform compatibility, with an addition of 6% Windows specific coding. With all Windows libraries and binaries also added, Windows specific proportion of DIRAC accounts for a total of ~11% of the total installation (by disk space). However Linux specific code has also been included in the total calculation.

3. Deployment and Performance

Demonstration of DIRAC under Windows and performance tests were carried out. This section describes the experience gained in deploying at a number of sites, and also using the system to perform a full physics selection study. The study involved running the LHCb analysis application DaVinci (C++ based) [10], testing both the Inprocess and Compute Cluster backends of the installation at Cambridge, and cross-platform submission test were performed with the LHCb interactive analysis application Bender (python based) [11].

3.1. Deployment

The first system used a small Windows Compute Cluster at Cambridge, designed to allow development and testing of DIRAC under Windows. This cluster consists of four Shuttle SN95G5 boxes. The cluster network topology is such that the head node has a public IP address and controls traffic to the other cluster nodes (on a private network) via a 10/100MB Realtek 8139 Ethernet adaptor. All four nodes have been defined as worker nodes, and are able to process user jobs. The cluster software chosen is the Windows Server 2003 Standard x64 Edition, which is installed on all four nodes. The rest of the cluster is then installed remotely from the head node. This remote installation could also be configured to install at the same time DIRAC and all pre-requisite applications, to allow for ease of deployment by the site administrator.

Following the successful testing and development at Cambridge, DIRAC has also been deployed on machines at the Universities of Bristol and Oxford as shown in Table 1. The main goal of the deployments at the Bristol, Cambridge and Oxford sites has been to provide CPU resources. In addition, deployment on a Windows laptop has been undertaken for the client functionality.

Table 1. DIRAC Windows deployment summary.

Location	Platform	Compute Element Backend	Number of CPUs Available
Bristol	Windows XP Professional	Inprocess	4
Cambridge	Windows Server 2003 x64 + Compute Cluster Pack 2006	ComputeCluster	8
Laptop	Windows XP Tablet PC	Inprocess	2
Oxford	Windows Server 2003 x64 + Compute Cluster Pack 2006	ComputeCluster	100
	Windows Server 2003	Inprocess	2

3.2. Performance.

LHCb is designed to investigate the matter-antimatter asymmetry seen in the Universe, concentrating on CP violation in the decay of B mesons. One example of a decay where CP violation effects are present is $B^{\pm}\rightarrow(D^0/\overline{D}^0\rightarrow K_s\pi^+\pi^-)K^{\pm}$. However in order for this calculation to be performed, an efficient selection of the signal events must first be extracted from the data.

Such a selection study is performed on the latest simulated data [12]. The goal is obtain an N-tuple of the selected events, and also the calculated mass squared of the final state products. This can then form the required Dalitz plot of the selection, and to extract the CP violation parameters.

3.2.1. *DIRAC Job Submission Setup.* The Cambridge Compute Cluster was chosen as the testing site, with access to the local data storages with the required data, visible to the cluster. However the GridFTP protocol could have also been used at the other sites to bring the data locally. Both Windows compatible Compute Element backends were tested on the cluster separately, the Inprocess backend on only the head node, and the Compute Cluster backend. For each backend the Agents were left in default 60sec loops.

The total 928,000 signal events are stored in over 500 files, equating to 27.2GB of pre-processed data. 11,000 of these events were used to test the Inprocess backend, while the rest were processed with the Compute Cluster backend.

The Inprocess backend was tested by creating three jobs, the first consisting of the first 1,000 events, the second, consisting of the next 1,500 events, the last job consisting of all 11,000 events, allowing the results of the first two jobs to be checked for consistency by the results of the third job. All three jobs were first submitted from a Windows machine to DIRAC to wait for the incoming Agent requests.

The remaining 917,000 events were split amongst 11 jobs, with varying job sizes of between 120,000-25,000 events per job, to be processed with the Compute Cluster backend. Here all four nodes were able to accept and process jobs, and an Agent configuration of four maximum running jobs, with one job per node.

3.2.2. *Results.* The selection reconstructed and passed 2528 B's. The plots of the reconstructed B mass distribution, and the respective Dalitz plot for the three body decay of the D^0 are shown in Figure 3 and 4 respectively. It can be seen with this sample the characteristics of the D^0 decay resonances can already be seen, and can be fitted to extract the CP violation information of the decay.

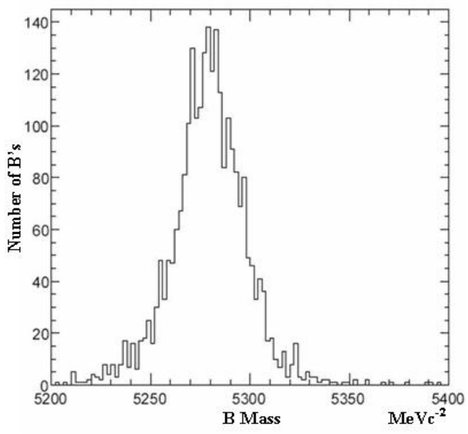

Figure 3. Reconstructed B mass

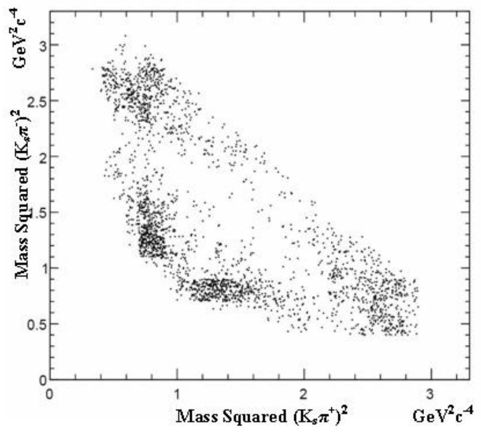

Figure 4. Dalitz y-projection

The Inprocess backend took a total of ~40min to process all three jobs, a total of 13,500 events, with all three jobs completing successfully in a row. This includes the matching times, the 60sec looping of the Agent, ~2sec startup time for the installed DaVinci application, and also the finalization for each job.

The larger jobs were processed by the Compute Cluster backed with a total CPU consumption of 46.8hours to process all 11 jobs (917,000 events). This equates to ~12hours for all four nodes running in parallel.

3.2.3. *Cross-platform Job Processing.* To demonstrate and test the cross-platform analysis on the Grid, Selection of $B^0 \rightarrow J/\psi(\varphi \rightarrow K^+K^-)$, and $D^{*+} \rightarrow (D^0 \rightarrow K^-\pi^+)\pi^+$ was performed using Bender, at a number of sites, most with Linux machines and some with Windows machines. The python based Bender application allows for the full analysis chain to be performed under Windows, with processing under Linux or Windows.

Using the jobs for the $B^\pm \rightarrow (D^0/\overline{D}^0 \rightarrow K_s\pi^+\pi^-)K^\pm$ study with DaVinci, and the D* and B^0 Bender example jobs described above, a comparison was made between the processing rates of DaVinci and Bender under the two operating systems. In order to make the comparison over various machines fair, the times are normalized to a 2.8GHz Xeon machine. The algorithm processing rate under Linux was 0.02sec per event, under Windows this was a factor of 10 higher at 0.2sec per event. This comparison however is biased by the performance difference between the optimized Linux binaries, and the default Windows debug binaries that are provided for the LHCb applications. This difference is unseen in the use of the Bender application tests. However compared with DaVinci the Bender startup times are slightly longer, but the processing times become more comparable as the total event numbers increase.

4. Conclusion

DIRAC has successfully been extended to allow use of distributed Windows resources. The DIRAC code has been made platform independent wherever possible, and Windows specific modules have been added where necessary, following the same structure as for Linux modules. From the point of view of the users there is very little change to the syntax used.

The result is a system that is easily deployed on various Windows platforms (XP and Server 2003), using a python installation script. The system is able to integrate both stand-alone Windows machines and Windows Compute Clusters into the existing Grid of Linux machines used by LHCb. The ability to perform cross-platform jobs submission has been demonstrated, with full analysis job chains completed either fully under Windows with DaVinci (creation, submission, processing and result retrieval under Windows) or cross-platform with Bender (creation, submission and result retrieval under Windows, processing under Linux or Windows).

The current DIRAC v2 system has been deployed and tested under Microsoft Windows at a number of locations, with deployment at additional sites planned for the future. A full Physics analysis will also be continued with the system, enabling further testing and development under the Windows platform.

References

[1] A Tsaregorodtsev et al. DIRAC: A community grid solution (These proceedings), 2007
[2] A Maier et al. Ganga – a job management and optimising tool (These proceedings), 2007
[3] S Paterson and A Maier Distributed Data Analysis in LHCb (These proceedings), 2007
[4] R D Graciani and A R Casajús DIRAC Agents and Services (These proceedings), 2007
[5] M Bargiotti and A C Smith DIRAC Data Management: consistency, integrity and coherence of data (These proceedings), 2007
[6] A C Smith DIRAC: Reliable Data Management for LHCb (These proceedings), 2007
[7] A Frohner et al. Recent Developments in LFC (These proceedings), 2007
[8] A Paventhan and K Takeda MyCoG.NET:Towards a multi-language CoG Toolkit (ACM International conference Proceedings Series; Vol. 117, Proceedings of the 3rd international workshop on Middleware for grid computing), 2005
[9] J Feng et al. Toward Seamless Grid Data Access: Design and Implementaion of GridFTP on .NET (Proceedings of the 2005 Grid Workshop (Associated with Supercomputing 2005) Seattle, WA), 2005
[10] http://lhcb-release-area.web.cern.ch/LHCb-release-area/DOC/davinci/
[11] I Belyaev et al. Python-based physics analysis environment for LHCb (CHEP04 Proceedings), 2004
[12] R Nandakumar The LHCb computing Data Challenge 2006 (These proceedings), 2007