Grid Computing and Particle Identification Performance Study with Untagged $\Lambda \to p^+\pi^-$

Grid Computing und Messung der Teilchenidentifikationseffizienzen am Zerfall $\Lambda \to p^+\pi^-$

Tobias Jenegger

München 2019

Grid Computing and Particle Identification Performance Study with Untagged $\Lambda \to p^+\pi^-$

Grid Computing und Messung der Teilchenidentifikationseffizienzen am Zerfall $\Lambda \to p^+\pi^-$

Tobias Jenegger

Masterarbeit
an der Fakultät für Physik-LMU München
der Ludwig–Maximilians–Universität
München

vorgelegt von
Tobias Jenegger
aus Brixen (I)

München, den 25. Juli 2019

Supervisor: Prof. Dr. Thomas Kuhr

Contents

List of Figures

List of Tables

Acronyms

API Application Programming Interface.

ARDA A Realization of Distributed Analysis for LHC.

ARICH Aerogel Ring-Imaging Cherenkov.

ATLAS A Toroidal LHC ApparatuS.

Basf2 Belle Analysis Software Framework 2.

bash Bourne again shell.

BNL Brookhaven National Laboratory.

CDC Central Drift Chamber.

CentOS Community Enterprise Operating System.

CLI Command Line Interface.

CMS Compact-Muon-Solenoid-Experiment.

CP Charge Parity.

CS Configuration Service.

CVMFS Cern Virtual Machine - File System.

DAQ Data Acquisition Model.

DIRAC Distributed Infrastructure with Remote Agent Control.

DISET DIRAC Secure Transport.

DMS Data Management System.

ECL Electromagnetic Calorimeter.

EGEE Enabling Grids for E-SciencE.

FTS File Transfer Service.

gBasf2 grid Belle Analysis Software Framework 2.

GSI Grid Security Infrastructure.

GUID Grid Unique IDentifier.

HEP High Energy Physics.

HER High-Energy Ring.

IP Interaction Point.

IR Interaction Region.

JMS Job Management System.

KLM K_L^0 and μ detector.

LER Low-Energy Ring.

LFC LCG File Catalogue.

LFN Logical File Name.

LHC Large Hadron Collider.

LHCb Large Hadron Collider beauty.

MC Monte Carlo.

mdst micro data summary table.

MySQL My Structured Query Language.

NP New Physics.

PFN Physical File Name.

PID Particle Identification.

PKI Public Key Infrastructure.

POSIX Portable Operating System Interface.

PXD Pixel Detector.

QCD Quantum Chromodynamics.

QED Quantum Electrodynamics.

RPC Remote Procedure Call.

SL 6 Scientific Linux 6.

SLAC Standford Linear Accelerator Center.

SM Standard Model.

SSH Secure Shell.

SSO Single Sign-On.

SVD Silicon Vertex Detector.

TCP Transmission Control Protocol.

TDD Test Driven Development.

TOP Time of Propagation.

UDP User Datagram Protocol.

VO Virtual Organization.

VOMS Virtual Organization Membership Service.

WIMPS Weakly Interacting Massive Particles.

WLCG Worldwide LHC Computing Grid.

WMS Workload Management System.

Abstract

This thesis presents my own developed ‘Minimal BelleDIRAC Grid System‘ (MBGS), a fully functional miniature of the Belle II Grid Computing System based on the Docker virtualization software. The MBGS can be installed locally and as an isolated system it is a convenient testing environment for Grid related software development at Belle II.
As such it was also used in the development of my own client API, called ‘Gbasf2ManagementSystem‘, which is also introduced in this thesis. It transfers the logic of job submission from user to server side with the goal to make the access to the Grid independent of the user's environment.
Furthermore, this thesis presents a study about particle identification at Belle II. For this purpose the $\Lambda \to p\,\pi$ decay mode is used, where the interest is focussed on the identification of the proton, the daughter particle of the Λ - meson. As a measure for the goodness of the proton identification algorithm the proton identification efficiency for correctly identified protons and the pion fake rate for misidentified pions as protons are introduced and the results from both simulated and real data samples are compared.

Chapter 1

Motivation and objectives

The Belle II project as successor of the Belle experiment is the most prominent $\mathcal{B}$-factory on the world. With the observation of the direct Charge Parity (CP) violation (from decays of B mesons) in cooperation with the BaBar experiment at the Stanford Linear Accelerator Center (SLAC) in 2001 [1] the Belle experiment made a significant contribution for the experimental verification of the CP violation and integrating it as intrinsic asymmetry of the Standard Model (SM).
With the main research center Ko Enerugi Kasokuki Kenkyu Kiko (KEK)(高エネルギー加速器研究機構) and about 750 researchers from about 100 universities/laboratories in 25 countries (as of 2018) [2] it is one of the major international physics collaborations and provided the experimental foundation for the Nobel Prize in Physics for Makoto Kobayashi and Toshihide Maskawa and their hypothesis for the origin of the CP violation.
Behind the astonishing discoveries in experimental particle physics huge effort has to be made to store, process and transfer the enormous amount of data which are retrieved from the Belle II detector and the physics simulations. This requires a large amount of computing resources (CPU power) as well as standardized and reliable storage and data transfer systems. For this purpose a Grid system, called Belle II Grid, has been developed to satisfy the demands of the Belle II collaboration group. The Belle II Grid system consists of several computing centers spread all around the world and makes also use of cloud computing (for simulations). The software framework DIRAC (**D**istributed **I**nfrastructure with **R**emote **A**gent **C**ontrol), which has already been used for the LHCb experiment at CERN, provides the adequate solution for the access and management of the partly heterogeneous computing resources [3].
To access and make use of the Grid resources the user only needs a valid Grid user certificate and a Scientific Linux 6 (or 7) environment with (pre)installed gBasf2 (**g**rid **B**elle II **a**nalysis **s**oftware **f**ramework) [4].
All the processes that run in the background are hidden from the user with the advantage to make the submission of grid-based basf2 jobs as simple as possible. The disadvantage is that the user does not have great insight into the configuration of the Grid system and the job submission workflow. In addition there doesn't exist an easy way for developers to test their tools or make analysis workflows on the Grid without partially interfering or

corrupting the whole Grid system which consequently can in the worst case cause sensitive outages of the computing system. These are some of the reasons for developing the " Minimal BelleDIRAC Grid System " which operates as a docker based network on an isolated environment. This light-weighted system can be used as a testing environment or for instructional purpose.
Another problem that arises from the current usage of the Belle II Grid system as a user is the fact that the functionality of the gBasf2 Client is limited to Scientific Linux 6 (or 7) operating systems. This forces a large part of users to switch to virtual machines, docker/singularity containers or other way around to access the benefits of Grid computing. For this reasons the development of a "Gbasf2ManagementSystem"-API was invoked. It should guide to the progression toward a paradigm shift for a system independent Grid access.

Chapter 2

Physics at the Belle II experiment

This chapter gives a brief overview of the standard model and the physics principles associated with it. In this context it will focus on the physics of $\mathcal{B}$-factories like Belle II.
The second part of this chapter will give a short introduction into the setup of the Belle II detector and show its role in the search for new physics.

2.1 The Standard Model (SM) of particle physics

The perception that matter is composed of elementary particles is already dated back to the ancient Greeks around the 6^{th} century BC [5]. The polymath Democrit made the first attempt to reduce matter to an invariant state which based on the widespread doctrine of atomism at that time [5]. In the following centuries the study of matter and its phenomena was preferably seen as part of "Natural Philosophy" and its approach was much influenced by the contemporary philosophical and religious movements and was adapted accordingly. Only by the 19^{th} century physics became a distinct discipline, separated from philosophy and other sciences, and the accurate scientific method for testing implications of the theories were on the advance [5] [6] and at the end of the century it was known that all matter consists of atoms. The discovery of the existence of about hundred elements with periodically recurring characteristics gave hints to for an underlying structure beyond the atom model [6].
In the early 20^{th} century with the discovery of the proton, neutron and the phenomena of radioactive decays a modern model of the atom was constructed [7]. From the measurements of particle accelerators and colliders in the 50s and 60s [8] the hypothesis of the inner structure of atoms was emphasized and a large number of these subatomic particles were found and assembled to a big "particle zoo". Upon experimental confirmation of the existence of the quark in the mid-1970s and the vast amount of newly proposed theoretical explanatory models as the SU(3) flavour symmetry for hadrons, the non-Abelian gauge theories by Yang and Mills and the spontaneous symmetry breaking came up. These theories were merged to the Standard Model as general theory for describing the electroweak and strong interactions in the universe [9].

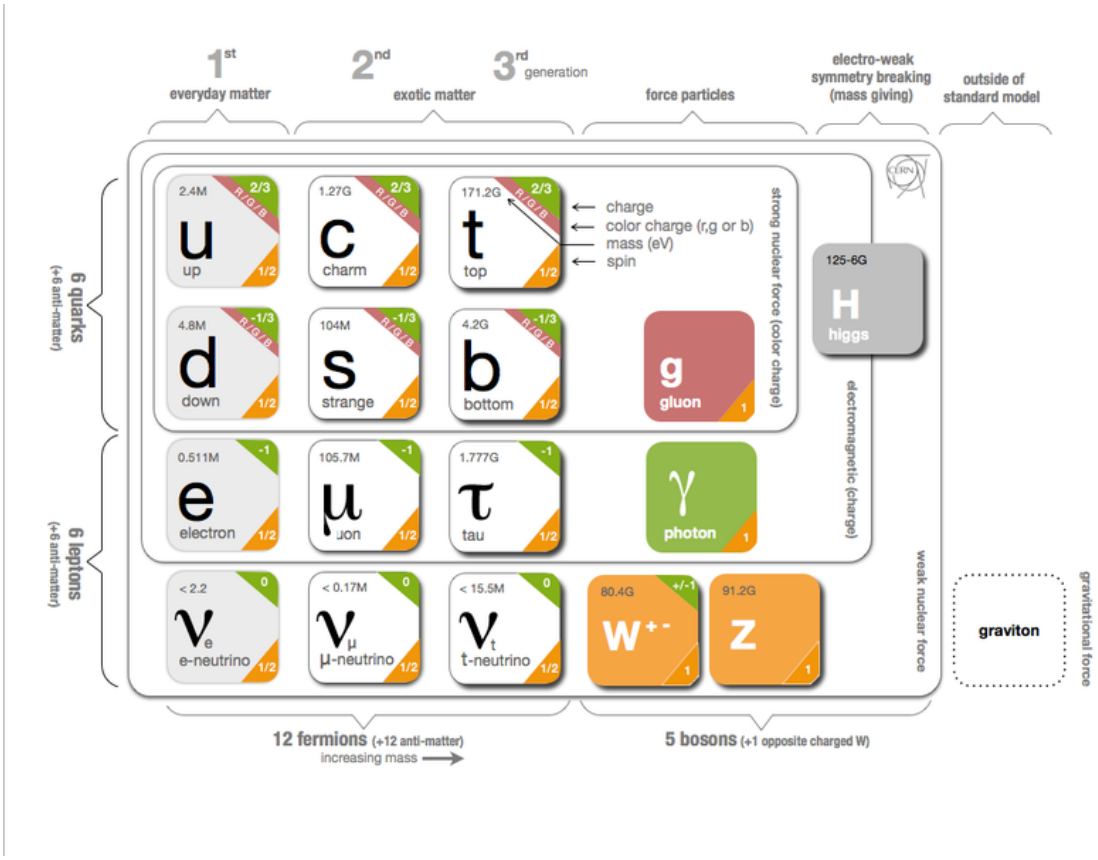


Figure 2.1: Standard Model of Elementary Particles [10].

2.1.1 Particle content and properties

The Standard Model is based on a fixed set of elementary particles (see Figure 2.1), which can be divided into two groups according to their spin: Fermions (particles with half-integer spin) and Bosons (particles with integer spin). Each elementary particle has its own anti-particle that has the same rest mass, but has the opposite charge-like properties.

Fermions

There are 12 different types of fermions, each of it called a "flavor".
Fermions can be subclassified into:

- ***Quarks***:
  Quarks are the smallest particles that assemble the atomic nucleus. Their spin is 1/2. There exist 6 types of quarks: up, down, strange, charm, top and bottom. Each quark has a fractional electric charge value - either $(-1/3)$ or $(+2/3)$. More complex particles composed of two quarks are called *Mesons* (e.g. B-Mesons) with three quarks *Baryons* (i.e protons or neutrons) respectively. All these quark composed

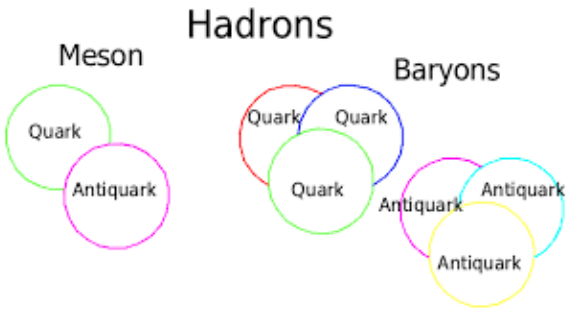


Figure 2.2: Overview of the classification of hadrons [11].

particles are summarized under the name of *Hadrons* (see Figure 2.2). Furthermore the quarks are split into three tuples of families:

$$\begin{pmatrix} \text{u(p)} \\ \text{d(own)} \end{pmatrix}, \begin{pmatrix} \text{c(harmed)} \\ \text{s(trange)} \end{pmatrix}, \begin{pmatrix} \text{t(op)} \\ \text{b(ottom)} \end{pmatrix}$$

Quarks have the additional property, called color charge. Quarks can take three values of colors: red, green and blue (antiquarks accordingly: anti-red, anti-green and anti-blue). All out of quarks composed particles (hadrons) are colorless. This phenomenon is also called " *confinement*".

- ***Leptons***:
  As for quarks there are also 6 leptons (and their antileptons respectively). The most prominent representative of this particle group is the electron. Others are: muon, tau, electron neutrino, muon neutrino and tau neutrino. While the muon, tau and electron have a electric charge value of (-1) the corresponding neutrinos have no electric charge. Same as the quarks also leptons are split into three tuples of families:

$$\begin{pmatrix} \text{e(lectron)} \\ \nu_e \end{pmatrix}, \begin{pmatrix} \mu\text{(muon)} \\ \nu_\mu \end{pmatrix}, \begin{pmatrix} \tau\text{(au)} \\ \nu_\tau \end{pmatrix}$$

Bosons

The gauge bosons are the force carriers in the SM (see Figure 2.1). Gluons, photons, Z/W bosons are spin 1 particles (i.e. vector bosons), the higgs boson belongs to the spin 0 bosons (i.e. scalar bosons). As they are particles with integer spin the Pauli exclusion principle doesn't affect them.

Forces

As far as we know there are just four fundamental forces in nature: the *strong, electromagnetic, weak* and *gravitational interaction* [12]. In Table 2.1 they are listed in order of

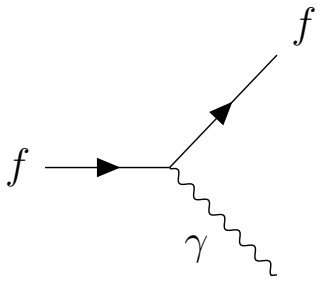


Figure 2.3: Elementary QED-interaction from which all electromagnetic phenomena emerge. The constituent parts are two fermions f that interact with the photon γ. (In quantum chromodynamics (QCD) the γ also couples to W^+ and W^-)

decreasing strength.

Force	Strength	Theory	Mediator
Strong	10	Chromodynamics	Gluon
Electromagnetic	10^{-2}	Electrodynamics	Photon
Weak	10^{-13}	Flavordynamics	W and Z
Gravitational	10^{-42}	Geometrodynamics	Graviton

Table 2.1: The four elementary forces [12].

- ***Strong Force***:
  The strong force is responsible for holding the quarks within the nucleus (protons and neutrons) together. It is the strongest of the four fundamental forces, described by the theory of quantum chromodynamics (QCD), a non-Abelian gauge theory. Its mediator particles are the gluons, which are as quarks colorful particles (carrying a color and an anticolor charge). The strong forces coupling "constant" α_s, which is highly dependent on the distance between the interacting particles, induces another phenomenon: the *asymptotic freedom* [12]. It means: at relative large distance (in the range of nuclear physics) its value increases, at very short distances it becomes small, leading to the confinement of quarks and gluons within their hadrons.

- ***Electromagnetic Force***:

  The dynamics of the electromagnetic force with its well known mediator particle, the photon, was already formulated in Maxwell's equations over one hundred years ago [12]. With the quantum theory a more general description was sought: quantum electrodynamics (QED). The electromagnetic force can be either attractive or repulsive. Its range is infinite and obeys Coulomb's law ($1/r^2$ dependence). Due to the $U(1)$

gauge invariant electromagnetic field the mass of the photon has to vanish. Figure 2.3 shows the elementary QED-interaction from which all electromagnetic phenomena emerge.

- ***Weak Force***: Weak interactions can be observed in β^- decays where a neutron decays into a proton (and an electron and electron-antineutrino). It is an example of a charged weak interaction where the charge of the involved quarks or lepton is changed by $+1e$ or -1e. There are three processes of charged weak interactions: leptonic, semileptonic and non-leptonic (see Figure 2.4). The mediators for these types of weak interactions are the W^+/W^- bosons.

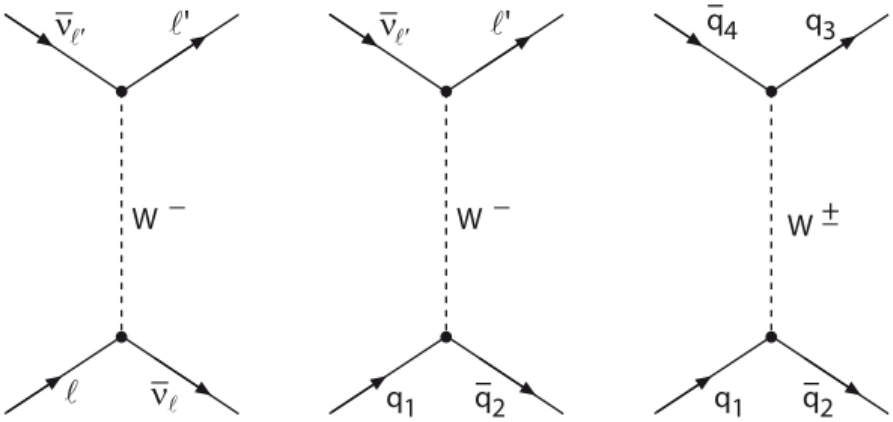


Figure 2.4: The three types of charged weak interactions: leptonic process (*left*), semileptonic process (*center*) and nonleptonic process (*right*) (the time arrow points upwards) [6].

The neutral weak interactions are carried by the Z^0 boson (see Figure 2.5). Notice that any process mediated by the photon could also be mediated through the Z^0 [12]. Due to its high mass of 91.1876 ± 0.0021 GeV/c^2 [13] it was not directly discovered before the end of the 1970s [14].

- ***Gravitational Force***: The gravitational force is the weakest of the four fundamental forces but it has an infinite range. Although not yet found, its hypothetical interaction particle is the 'graviton'. The SM includes the electromagnetic, strong and weak forces and can precisely explain how these forces interact with matter. Even though gravity is the most familiar and intuitive force for the human being, it is not part of the SM. Due to its weak coupling constant G, the effect of gravity is luckily so weak as to be negligible for the analyses of particle physicists, who work in the regime of MeV to few GeV [15].

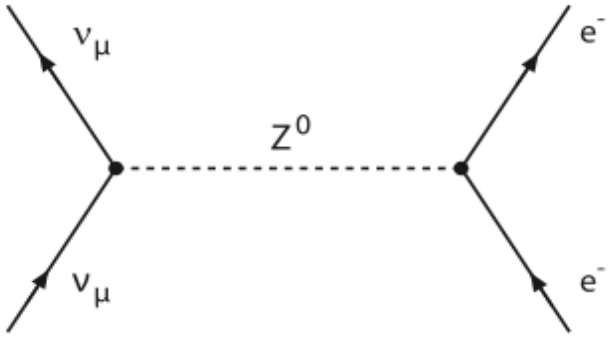


Figure 2.5: Example for a neutral weak interaction: the scattering of muon neutrino to an electron, $\nu_\mu e^- \rightarrow \nu_\mu e^-$ with the Z^0 boson as interaction particle (the time arrow points upwards) [6].

2.1.2 Additional particle properties

To describe particles and their composites several conserved quantities are used. To mention the angular momentum **L** and spin **S** of a particle. Former is nothing else than the classical orbital angular momentum while the spin is simply an intrinsic property of the particle itself. Added together they result in the total angular momentum **J**. For example an electron has an orbital angular momentum of l and a spin value of $1/2$ which results in a total angular momentum of:

$$j = l \pm 1/2 \text{ (parallel/anti-parallel addition)}$$

To explain better the concept of flavour symmetries in conformity with the formalism of spin and angular momentum the *isospin* was introduced. It is not a vector in ordinary space but in the abstract 'isospin space' [12]. Up and down quarks have a value of $1/2$. The third component of the isospin I_3 is defined as $+1/2$ for up quarks, $-1/2$ for down quarks and 0 for all the other quark types. For hadrons we have:

$$I_3 = \frac{1}{2}(n_u - n_d)$$

where n_u is the number of containing up and n_d of down quarks respectively.

Higgs mechanism

The actual SM without Higgs fields would predict massless fermions as well as massless gauge bosons. From experiment we know that the W and Z bosons are massive instead and would therefore contradict to the Glashow-Weinberg-Salam model [16]. Therefore the Higgs-mechanism with its *Spontaneous Symmetry Breaking* was postulated in the 1960s by the theorists Robert Brout, François Englert, T. W. B. Kibble, Carl R. Hagen, Gerald

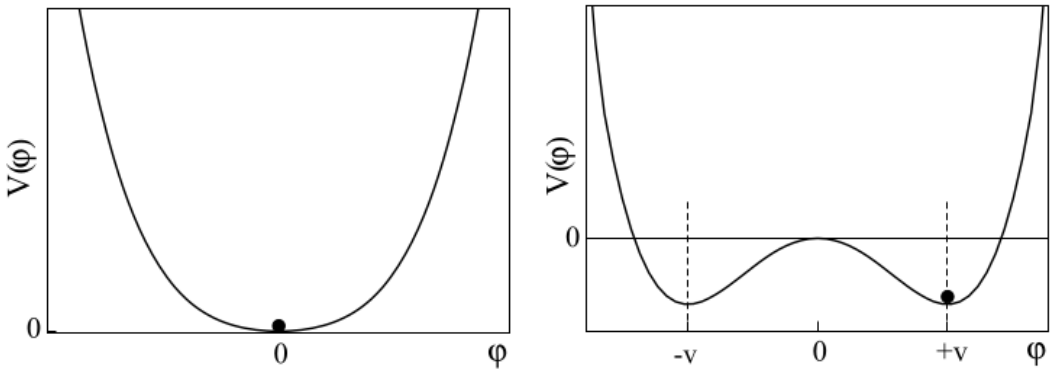


Figure 2.6: One dimensional higgs field. For the case $\mu^2 > 0$ and $\lambda > 0$ the minimum of φ lies at 0 (*left*). For $\mu^2 < 0$ and $\lambda > 0$ the minimum moves to $v = \sqrt{-\mu^2/(2\lambda)}$ (*right*) [6].

Guralnik and Peter Higgs [17].
To illustrate the phenomenon of spontaneous symmetry breaking we take the one dimensional real case (in the SM the Higgs field has four degrees of freedom) where the corresponding Higgs potential is:

$$V(\varphi) = \mu^2\varphi^2 + \lambda\varphi^4.$$

For the case of positive μ^2 and λ the energetic preferable state is $\varphi = 0$ (see Figure 2.6). Spontaneous symmetry breaking takes place when $\lambda > 0$ and $\mu^2 < 0$. This results in a shift of the ground state to

$$\varphi_0 \equiv v = \pm\sqrt{\frac{-\mu^2}{2\lambda}}.$$

The v is called the vacuum expectation value of the Higgs field. This ground state isn't symmetric in respect of reflection symmetry, it's so called 'spontaneously broken'. This is an analogy to the phenomenon of condensates in superconductors and the charged Cooper pairs, called Meissner-Ochsenfeld effect. In the case of the Higgs field it is a condensate in the space of *weak hypercharge*. This condensate with indefinite number of 'weak hypercharge units' interacts with all fermions and gives thereby their mass. The spontaneous symmetry breaking of the four dimensional Higgs field gives additionally masses to the W^+, W^- and Z boson as they all absorb one degree or freedom in this breaking process. The last degree of freedom goes to the physical scalar Higgs boson [6]. This can also be interpreted as an excitation of the complex Higgs field in the radial direction of the 'mexican hat' potential (see Figure 2.6). The Higgs boson with a mass of approximately 125 GeV was discovered by the ATLAS and CMS experiments at the Large Hadron Collider (LHC) at CERN in 2012 [18]. The discovery of the existence of the Higgs boson was a milestone for the interpretation of modern particle physics.

CP violation

The study of symmetries in relation to the laws of physics is a fundamental question and highly connected to the question why the universe is how it is. According to the big bang theory, equal amounts of matter and anti-matter were initially created. Most of it annihilated. One out of every billion quarks survived this annihilation process and laid the foundations for the massive universe we have. But why could some matter survive the annihilation process and why do we have an excess of matter? Is there some other effect which pushed the annihilation of anti-matter?
These are just some of the questions which moved physicists to concern themselves with the combined CP symmetry.
CP-symmetry stands for **C**harge conjugation **P**arity symmetry and is a combination of:

- **C**harge swapping: switches all particles with their corresponding antiparticles (and vice versa) by changing the signs of all charges.

- **P**arity swapping: flips the sign of all spatial coordinates of the particle framework.

It has been experimentally shown that both electromagnetic and strong interactions are C and P invariant, therefore they must also be symmetric under the product CP [19]. This means that these two fundamental forces act exactly the same way on mirrored 'anti-charged' particles. The weak force on the other hand violates both C and P symmetries, as experimentally confirmed by Chien-Shiung Wu et al. in 1957 [20].
In 1964 Cronin and Fitch reported that also the CP invariance for the weak force was violated in their study of the decays of neutral kaons. They observed that the neutral kaon (formed by strange and down anti-quark) can turn into its antiparticle and form a linear combination of the two. The two possible linear (normalized) combinations are eigenstates of CP:
$|K_1\rangle = (1/\sqrt{2}))(|K^0\rangle - |\bar{K^0}\rangle)$ and $|K_2\rangle = (1/\sqrt{2}))(|K^0\rangle + |\bar{K^0}\rangle)$
with
$CP\,|K_1\rangle = |K_1\rangle$ and $CP\,|K_2\rangle = -\,|K_2\rangle$

From this follows, assuming CP conservation, that $|K_1\rangle$ decays into states with $CP = +1$ and $|K_2\rangle$ into $CP = -1$ combinations. Typically neutral kaons decay into two (CP $= +1$) or three pions ($CP = -1$) which means that $|K_1\rangle$ decays into two pions and $|K_2\rangle$ into three pions [12].To their surprise Cronin and Fitch observed a small number of $|K_2\rangle$ decaying into two pion and thereby an unmistakable evidence of CP violation. The Kobayashi-Maskawa mechanism provides an elegant and simple explanation of CP violation [19]. In the form of a 3x3 matrix it describes the probability of a transition from one quark type to another as flavour-changing weak interaction. Despite the success of this theory explaining the small deviations of the CP invariance in the experiments it is not able to give an answer for the large asymmetry between matter and antimatter in the universe.

Shortcomings of the SM and New Physics

The Standard Model has been a genuine success since its beginnings, just to mention the experimentally proven predictions of the Higgs boson, the W and Z bosons and the anomalous magnetic dipole moment. On the other side the list of failures of the model is quite long too, just to mention a few:

- It doesn't implement the gravitational force.
- 'Hierarchy Problem'. The SM cannot explain the large differences in the coupling constants of forces at low energy scales.
- The SM predicts massless neutrinos. The observation of so called '*neutrino oscillations*' (which are also SM-forbidden) imply massive neutrinos. Extensions of the SM are aiming to explain neutrino masses by either defining neutrinos as Dirac particles or Majorana particles. Both models come along with implications which are problematic to explain from the SM [21].
- The matter we know and which the SM tries to describe just makes up 5% of the content of the universe. Dark matter (DM) and dark energy are accounting for the remaining 95% [22]. The SM does not make any predictions about the nature of dark matter and dark energy.
- The Kobayashi-Maskawa mechanism manages to explain the CP-violation measurements obtained in the particle physics experiment but nevertheless fails to explain the observed cosmological CP violation which is several orders of magnitude higher [19].

These drawbacks lead to numerous SM-extensions and complete new models in the field of New Physics (NP). To explain the crucial experimental problem of the nature of DM a vast number of solutions exist from WIMPS (Weakly Interacting Massive Particles) to axions or to sterile neutrinos with masses of several keV [23]. With the help of the Belle II detector hundreds of scientists all around the world are seeking to answer several of the open questions in particle physics.

2.2 The Belle project

The foundation of the High Energy Accelarator Research Organisation (KEK) in 1997 in Tsukuba, Japan, was the starting shot for the Belle project [24]. The core of the project is the Belle detector which was first put into operation in 1998 at the interaction point of the KEKB, a 3 km circumference asymmetric electron-positron collider, located at the KEK. The construction of the collider with two dedicated rings where beams of electrons and positrons run with asymmetric energies giving brilliant Lorentz boost to the the bunch of colliding particles sufficient for observing the time evolution of B decays (see Figure 2.8). These and other technical advantages equipped the Belle detector optimally for

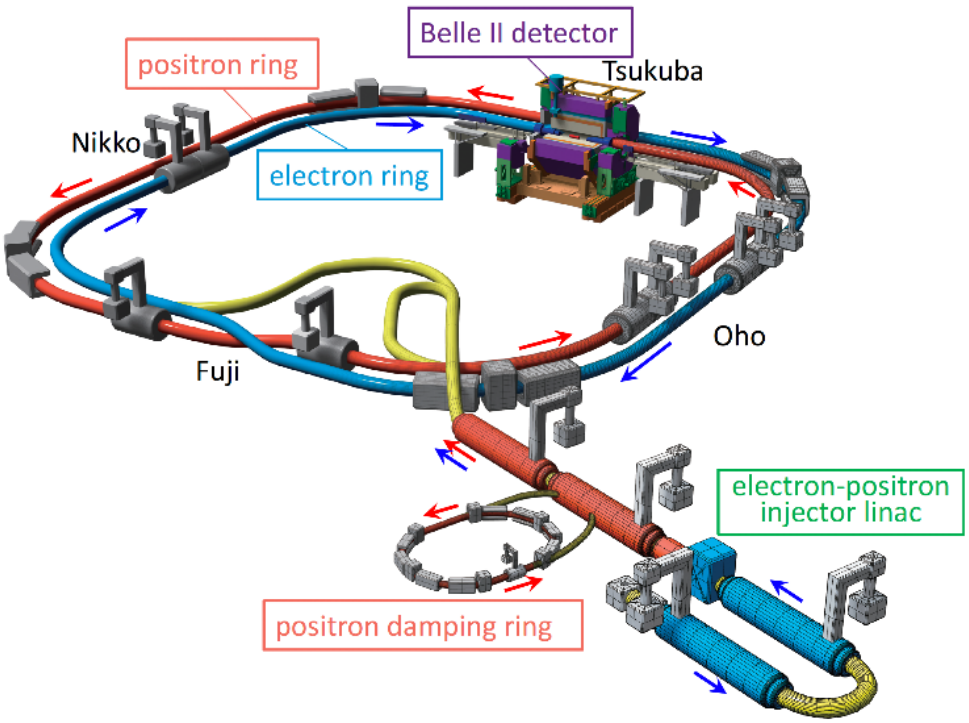


Figure 2.7: Schematic view of SuperKEKB. The electron and positron rings have four straight sections named Tsukuba, Oho, Fuji and Nikko. The BelleII is right at the center of the collision point [26].

precise CP-violation measurements and has brought important insight into the structure of elementary particles. These measurements culminated in the 2008 Nobel Prize for physics awarded to M. Kobayashi and T. Maskawa for their theory of CP violation [25]. The Belle experiment ran for 12 years from 1998 until 2010. In 2011 the construction of SuperKEKB, the successor of the KEKB, started. In 2016 the first beam circulations were achieved and first collisions were scheduled in the mid of 2018 [26]. The SuperKEKB targets a luminosity 40 times higher ($8\text{x}10^{35}$ $\text{cm}^{-2}\text{s}^{-1}$) than the luminosity at KEKB. Along the way the Belle detector was upgraded to Belle II. On the 11th of April 2017, the Belle II detector at the KEK laboratory in Japan was successfully "rolled in" to the collision point of the upgraded SuperKEKB accelerator [27]. Data taking at Belle II started in early 2018.

At the Belle II detector bunches of electrons with a beam energy of 7 GeV collide with 4 GeV positrons [25]. This corresponds to a centre-of-mass energy close to the $\Upsilon(4S)$ resonance that predominantly decays into B-mesons. Therefore Belle II is often called a ***B***-factory.
Besides the study of CP violation in the various decay channels of B mesons tau lepton

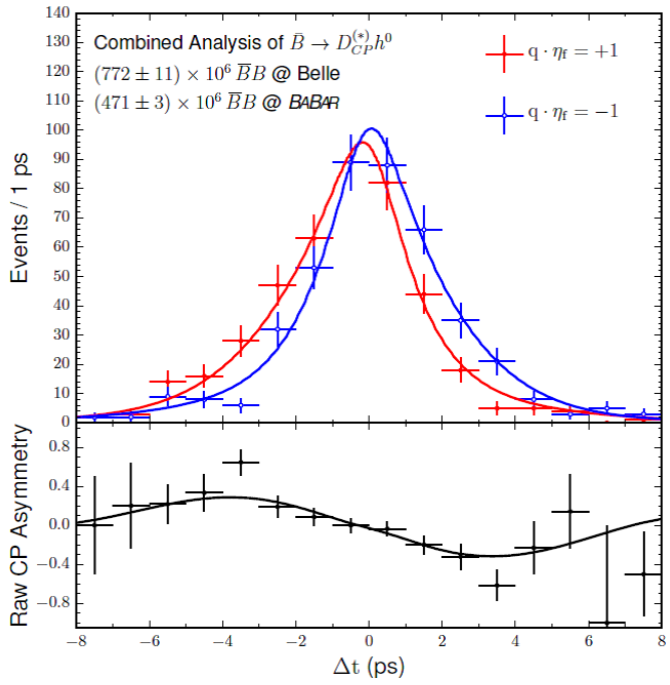


Figure 2.8: Plot of the observed B/$\bar{B}$ for given time difference Δt. The difference between the red and blue lines shows the difference in how a particle and its antiparticle behave. This indicates the expected CP violation [28].

flavour violation, D^0 mixing CP-violations and indirect dark matter analyses are very exciting and promising research areas of the Belle II collaboration towards new physics contributions and SM testing.

2.2.1 The Belle II detector

The upgrade of the Belle detector to the Belle II detector was necessary to cope with the higher instantaneous luminosity provided by the SuperKEKB accelerator. The Belle II detector is build up of following sub-detectors:

- **Interaction Region/Chamber:**
  The Belle II detector is set at the interaction region (IR) where electrons from the high-energy ring (HER) collide with the positrons from the low-energy ring (LER) with a crossing angle of 83 mrad. To focus the beams at the interaction point (IP) various magnetic focusing systems are used. The IP-chamber envelopes the collision point. It consists of a vacuumized double-walled beryllium pipe with a normal paraffin liquid as a surround coolant.

- **Pixel Detector (PXD):**
  The two layers of pixelated silicon sensors (PXD) are responsible for gathering the hit

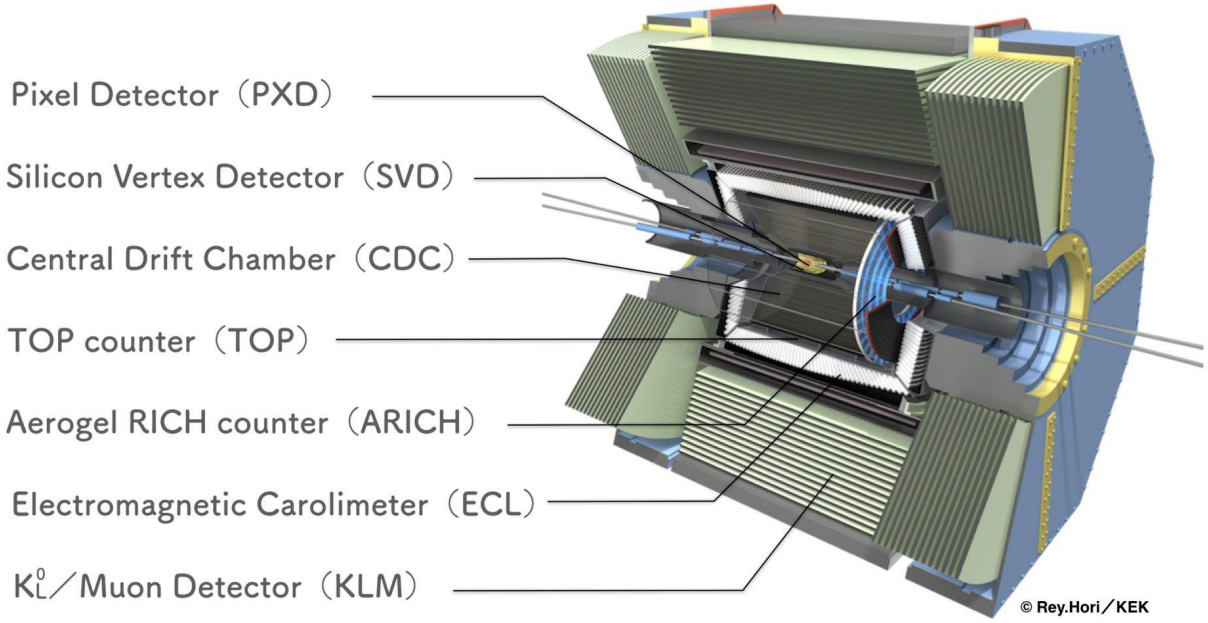


Figure 2.9: The Belle II detector with its sub-detectors [29].

information near the beam interaction region allowing reconstruction of secondary vertices. The PXD is based on the DEPFET (DEPleted Field Effect Transistor) technology which allows for very thin (50 μm), energy saving and low scattering sensors [25].

- **Silicon Vertex Detector (SVD):**
  The silicon vertex detector together with the inner PXD is in control of the measurement of the two B decay vertices (from the $\Upsilon(4S)$ decay). In addition, the SVD measures vertex information in other decay channels involving D-meson and τ-lepton decays [25]. It is made out of four layers of double sided strip detectors and together with the PXD the impact parameter resolution goes down to 20 μm [30].

- **Central Drift Chamber (CDC):**
  The central drift chamber carries out three main tasks: it measures trajectories, extracts particle information from measurements of energy loss and works as trigger for charged particles [25]. The drift chamber is filled with a gas mixture of helium (He) and ethane (C_2H_6) with an inner and outer radius of 160 mm and 1130 mm, respectively. The detector's polar angular acceptance lies between 17° and 150°. 14,336 wires with a diameter of 30 μm keep track of the passing charged particles [25].

- **TOP and ARICH detector:**
  A barrel shaped Time of Propagation (TOP) counter consisting of 16 quartz modules, each 2.5 m long and 2 cm thick enwrap the CDC detector. The TOP counters en-

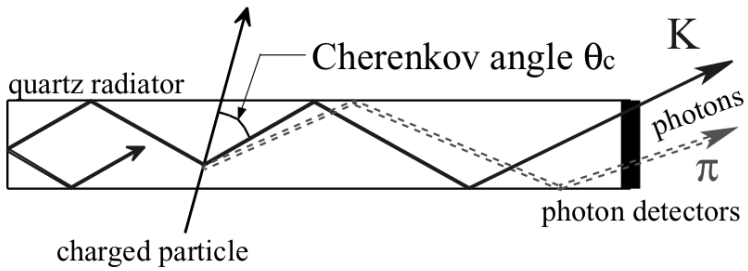


Figure 2.10: Schematic side-view of TOP counter and internal reflecting Cherenkov photons [25].

able K/π separation for a wide range of momenta (from ~ 0.6 GeV/c to ~ 4 GeV/c). Charged particles that pass through the TOP detector create Cherenkov photons. Their time of flight is measured (with a resolution of ~ 100 ps) inside a quartz radiator. Figure 2.10 shows the schematic measurement method: A charged particle with a velocity of v intersects the detector. Depending on its velocity Cherenkov photons are emitted along the Cherenkov angle θ_C. It applies

$$cos(\theta_C) = \frac{1}{n\beta}$$

with n being the refractive index of the medium and β the ratio between the speed of the incoming particle and the speed of light ($\beta = v_P/c$).
To separate kaons from pions over most of their momentum spectrum in the forward direction of the asymmetric Belle II detector a proximity-focusing Aerogel Ring-Imaging Cherenkov (ARICH) detector was designed [25]. Similar to the TOP detector it exploits the Cherenkov radiation to discriminate between incoming charged particles.

- **Calorimeter(ECL):**
  The electromagnetic calorimeter (ECL) comprised of scintillation crystals is located inside a superconducting solenoid coil that provides a 1.5 Tesla magnetic field [31]. The ECL provides a precise determination of the photon energy and angular coordinates as well as electron identification since many of the final states of interest (e.g. $B^0 \to J/\Psi K_S^0$, where J/ψ decays to e^+e^-) require to measure the energy of both charged and neutral particles [32] [25]. Other tasks of the ECL are on- and offline luminosity measurement, triggering and K_L^0 detection together wit the KLM.
  When signal particles pass the detector the CsI(Tl) crystals are activated and the signal is amplified by the preamplifiers. As a next step the signal is digitalized at a 2-MHz clock frequency and is filtered by various filtering algorithms. The data is only processed if triggered by a signal with corresponding requirements [25].

- **K_L^0 and μ detector (KLM):**
  The KLM detector identifies K_L^0 and μ above 600 MeV/c with high efficiency [32]. It consists of an alternating sandwich of 4.7 cm thick iron plates and active detector elements. The iron plates serve as the magnetic flux return for the inner superconducting solenoid coil. The active elements are double-gap gas electrode RPCs. The K_L^0 mesons interact in the ECL or the iron plates and create a clear signature while muons penetrate several metres of iron without interacting. When a charged particle like a muon traverses the KLM detector a discharge in the gas-electrode RPCs induces signals in both of the orthogonal external copper strip planes. Due to the external magnetic field of the solenoid coil the particles, depending on the sign of their electric charge, follow tracks with negative or positive curvature.

Chapter 3

Grid Computing

Grid Computing, as a visionary model of freely sharing computer power and data storage, evolved in the early 1990s. Throughout the scientific world the size of the data output of the experiments and the complexity of the problems increased rapidly and with it the demand for computational power. Large computer centres could provide the required resources but for many small collaborations such kind of infrastructures is too expensive and service intensive in relation to the actual benefit. On the other side large computer centres, if just used from individual science groups, are badly load balanced due to the fact that most scientific projects run their computing jobs within short campaigns leaving the computing processors of those powerful computer platforms in an idle state most of the time. Grid computing tries to bypass these issues by providing its users access to high end computational capabilities and thereby utilizing at maximum the existing resources across a network. The high efficient utilization of the resources makes grid computing a cost-effective infrastructure that supports innovation and collaboration in the world of science.
In analogy to the electric power grid, the idea of grid computing is to provide the customer instantaneous computing power, data storage and accessing capacity through a user friendly and standardized interface to make grid computing behave like the electric grid of the twenty-first century.

3.1 What is the Grid?

With the rise of interest in the grid technology, not just in the academic circle but also in the commercial sector, the ambiguity of the terminology 'grid (computing)' increased. The first attempt to describe this new computing architecture was made by Carl Kesselman and Ian Foster, back in 1998 [33]:
"A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities."
As the grid technology evolved and with it the conceptual understanding of it, Ian Foster suggested a three point checklist on which grid computing systems, for be calling so, have

to agree [33]:

1. *'Coordination of resources that are not subject to centralized control'*:
   This point clearly differentiates grid systems from large computing farms where the system is centrally controlled, even if quite diverse processes are running on parallel on the system. Contrary to intuition, also the Web isn't a grid: it is open and uses general-purpose protocols (see Point 2) but it doesn't offer services to coordinate the available resources and their workloads.

2. *'Usage of standard, open, general-purpose protocols and interfaces'*:
   To give the user a reliable and comfortable access to the grid resources, it has to be standardized and transparent. Similar to the electrical power grid, where the user can utilize the current for its drilling machine as well as for the washing machine, it should be independent of the users application field.

3. *'Delivery of nontrivial qualities of service'*:
   This requirement demands smart services and management tools. It points to the general aim of grid computing which is to provide a efficient and economic solution for complex user demands without disregarding high security standards.

3.2 Grid Architecture - An Overview

The grid and its architecture, as depicted from section 3.1 has a multiple-owner structure. That means it is based on resource sharing. Its complex and widespread structure can be grouped into two blocks: the hardware and software grid components.

3.2.1 Grid Hardware

Big computing centres, (commercial) cloud computing systems as well as singular local personal computers constitute the grid resources and provide the users the requested CPU power and data access (or storage place). These resources, called nodes, can be truly heterogeneous computing systems in different control domains as well as unusual data resources, such as printers.
The resources are interconnected by a network layer which form an aggregation of network-connected computers to a large-scale distributed system (see Figure 3.1). These networks can show widely varying performance characteristics and control policies [34]. Certainly favoured are fast and stable connections with low latency and maximum network bandwidth for huge data transfers. But grid computing means also to be able to handle unstable network connections with appropriate software solutions. Even if sophisticated software

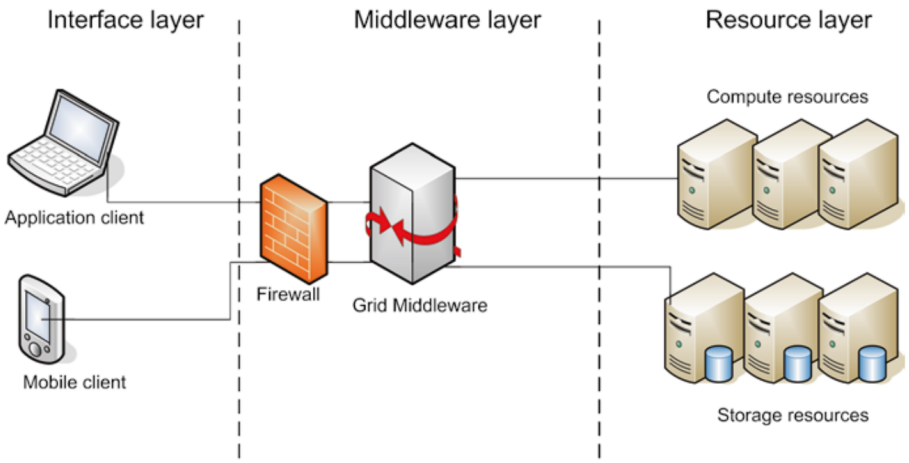


Figure 3.1: Example of a three-layered grid architecture [34].

technologies come to terms with the geographically large distance networking it becomes clear that applications which demand high back and forth communication traffic between user and working nodes and at the same time consume small computing resources (in the sense of CPUs) are not suited for the use of the grid.

3.2.2 Grid Network Infrastructure

Beyond the hardware part, the grid infrastructure needs well synchronized software solutions. Firstly it has to deal with the interconnection between the different types of resources and the users. Speed and robustness of an underlying network is a necessary condition for functionality of a grid [35]. General standards are needed and network architectures and infrastructures are on their way toward Internet Protocol (IP) convergence [34]. This technology isn't appropriate for several grid applications which require high troughput of data with fast data transfer (several Gbit/s) [36]. *Hybrid networks* seem to be the solution for different data transfer types at the same time. This new technology, often called *lambda networking*, allows the use of multiple optical wavelengths and provides independent transfer channels along a strand of fiber optic cable [37]. Also on the higher stage, the transport layer, different protocols are used: the User Datagram Protocol (UDP) seems to be preferred over the Transmission Control Protocol (TCP) due its large data losses. An average usage of a 10 Gbit/s network provides only about 7.5 Gbit/s when TCP is used [35].

3.2.3 Grid Middleware

The middleware, which enables sharing and manages the grid components based on user requirements and resource attributes (e.g. capacity, availability), is the core of the grid

software. The middleware as software infrastructure has to orchestrate and manage the grid interconnection between the wide range of resource infrastructure and the user clients needs to support a variety of technologies in order to avoid interoperability issues and to accommodate a wide field of uses at once. To make grid computing userfriendly the middleware should infold the complex grid network and provide well controllable APIs (Application programming interface) for the application and service layer. The top of the grid software framework is reserved for the actual application tools of the various user groups/research collaborations. These servicewares are specific to the user requirements, nevertheless they have one thing in common: the usage of the middleware APIs which connects them over the grid to the demanded resources. Based on [38], a convenient grid middleware architecture should be motivated from three aspects:

(a) Arrangement in layers: this allows to bootstrap the infrastructure from low to high level.

(b) Role-based: this should abstract the grid functionalities. Based on the users privileges, resources are interfaced and access is allowed.

(c) Service oriented: the middleware model should be abstracted in a way that others can easily deploy, create, and contribute to. It should be constructed in a way that it is accessible for multiple user applications.

(d) Low-maintenance: When developing middleware, maintenance, management and policy issues must be taken into account. An example would be the regular check of versioning compatibility of the applied software components, which isn't an easy task for state of the art grid middlewares. A middleware achieves high acceptance within a large community only under reliable software maintenance.

Apart from these general infrastructure concerns, the middleware should also focus on the services it delivers. Following elementary services/functionalities are delivered by grid middleware [38]:

- ***Job execution services***
- ***Security services***
- ***Information services***
- ***File transfer services and data management***

Job execution services

The grid middleware has to be able to overlay a huge number of heterogeneous resources to resolve the appropriate resource for the specific job considering its actual load. Therefore basic resource management functionalities should be supplied [36]:

- resource discovery and inventories
- fault isolation
- resource monitoring
- variety of autonomic capabilities (e.g. self-configuration tools)
- service-level management activities

Important for the job submission are services that regularly check its status. All these tools are progressively designed towards self-directed software to keep the maintenance as low as possible. To keep track of the submitted jobs on the vast grid landscape, jobs must have a unique identifier called jobID.

Security services

Appropriate security mechanisms have to be applied so that just authenticated and authorized users can use the requested resources. Even if there are very specific security issues and policies in the different grid domains, the appropriate security services should allow the users to perform a single sign-on (SSO) to access the grid. This means the user has to declare its identity just once. The authentication services create authentication attributes, which are callable on the whole grid, and have a fixed validity period. If the session key validity has expired, the user has to sign in again to refresh his authentication. These elementary security mechanisms and encryption make generally use of commodity technologies such as Public Key Infrastructure (PKI) or Kerberos [38].
At the same time of authentication also authorization takes place. The users privileges depend on its virtual organization (see Section 3.3) and its role within it.

Information services

Information services are needed to query important information about (used as well as unused) resources, participating services and users on the grid. This information is send to the job execution services to allocate appropriate resources for the respective jobs. Elementary information about the users is already provided by the affiliated virtual organization (see Section 3.3) and is resumed by the grid information services.
When designing information services developers should be aware of the yield of the information queries and scale their frequencies with respect to the requirements. For example services for querying the amount of free memory for a compute resource may pose immediately problems if the information service is not properly implemented. The information service for example queries the amount of memory every millisecond, but the memory change is updated only every second on the remote computer. The consequences are wasted resources on the information service server and an overloaded network. Hence protocols and standard mechanisms are asked to avoid such a scenario. Resource intensive services should be substituted by smart grid software solutions with high throughput capabilities.

File transfer services and data management

Next to the allocation of computing resources the grid allows to store and manage a huge amount of data. File transfer and data management systems should provide software solutions for fast data access. Such technologies should be able to store and replicate dynamically data depending on the demands of the users. That means if users in a certain geographic region often demand specific data packets those should be preferably stored on the nearest data center to speed up the data access and reduce latencies. Up to now most of these kind of data transfers is done manually. In grid communities it is generally agreed that this process should be mostly taken over by some smart services that can exactly register data retrievals which drive to data transfers for optimal distributed resources.
The huge data storage capacity makes it also necessary to offer effective data searching environments. A certain metadata system is used for this reason. The metadata system catalogues the data according to some attributes (e.g. location, contents, formats, etc.). This system contributes to a visualization of the overall storage and hides the complexity of the overlayed storing network from the user.

3.3 Virtual Organization (VO)

Virtual organizations (VO) play a major role in grid computing. Similar to real organizations they represent pressure groups with special computing and data demands (and restrictions). Such virtual organizations could be a cluster of universities, research groups or other institutions. Every user on the grid system has to be registered for at least one of the accredited VOs of the grid system. As part of a VO the user has a well defined role with clearly defined privileges and restrictions.
The grid middleware service that provides functionality for the VOs is called Virtual Organization Membership Service (VOMS). It supports the authorization management on the grid and provides a database of user roles and capabilities as well as generation tools of local proxy credentials [39].
Figure 3.2 shows how user authorization with VOMS works: The VO admin has set the role of the user and sends this information to the dedicated VOMS server (1). When the user requests a grid certificate from the local certification authority (CA) the role attributes are integrated in the acquired certificate (2). The user has now access to the grid services with the authorization policies encoded in his grid certificate (3).

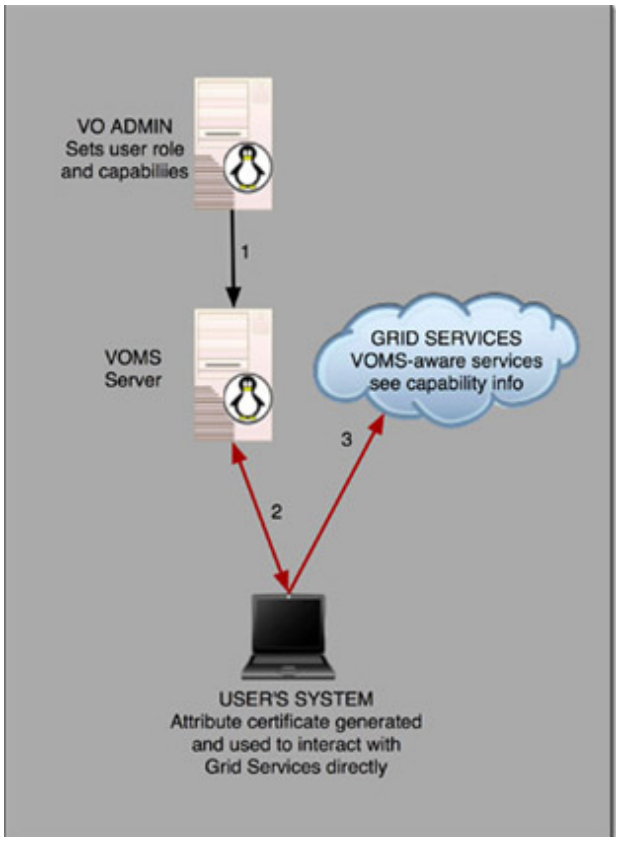


Figure 3.2: Schematic VOMS workflow [39].

Chapter 4

Grid Computing at Belle II

Belle II as successor of the Belle detector, and its upgraded accelerator SuperKEKB are designed for a data rate 40 times higher than Belle [40]. The accumulated raw data is expected to reach at 200 PB by the end of the experiment (2022) [41] which corresponds to a data sample of about 50 ab^{-1}. While the computing system of the Belle experiment was managed by a centralised computing facility at KEK over its ten-year operation [25], Belle II has decided to adopt a distributed computing model. Due to the similar requirements as for the LHC experiments the Belle II computing model selected a similar hierarchical structure to the Worldwide LHC Computing Grid (WLCG), just with a lower complexity. The following sections will give an overview of the hardware and software components involved in the Belle II computing model from raw data processing to the final Belle II specific physics analysis tools.

4.1 From particle collisions to physics analysis

At Belle II positrons (e^+) and electrons (e^-) collide at the interaction point (IP). New particles are produced (mainly B mesons) which further on decay into long-lived and stable particles. The Belle II sub-detectors keep track of these events and translate the taken measurements into feasible data using a sophisticated data acquisition model (DAQ). The raw data is stored in the ROOT [1] format on tape at KEK and copied only to one other site, BNL (Brookhaven National Laboratory). This provides a backup copy and the possibility to distribute reprocessing tasks over both sides [40].
The second layer of the computing system are the grid sites which are distributed all around the world. These can be of very heterogeneous nature. Also some cloud clusters from commercial vendors can be applied, especially for Monte Carlo (MC) production when all grid sites are fully loaded. But the aspect of buying resources from a company can raise the potential danger of vendor lock-in, which has to be avoided.
The raw data which is processed at the main computing center (KEK) is copied to the grid sites in the mdst (micro data summary table) format, which contain all relevant information

[1] A machine-independent compressed binary file format [42].

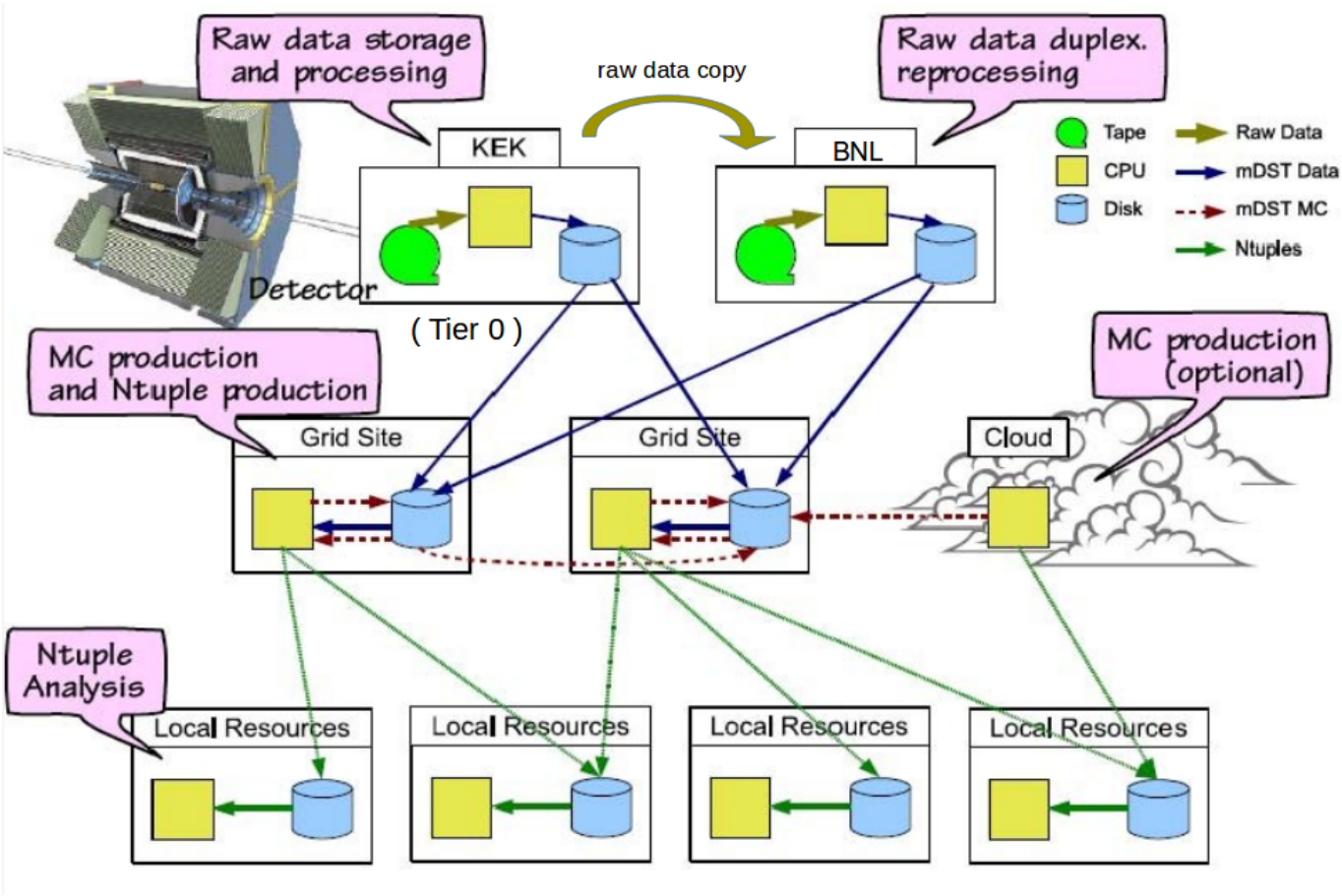


Figure 4.1: Schematic model of the Belle II grid system from data acquisition to local scientific analyses [25].

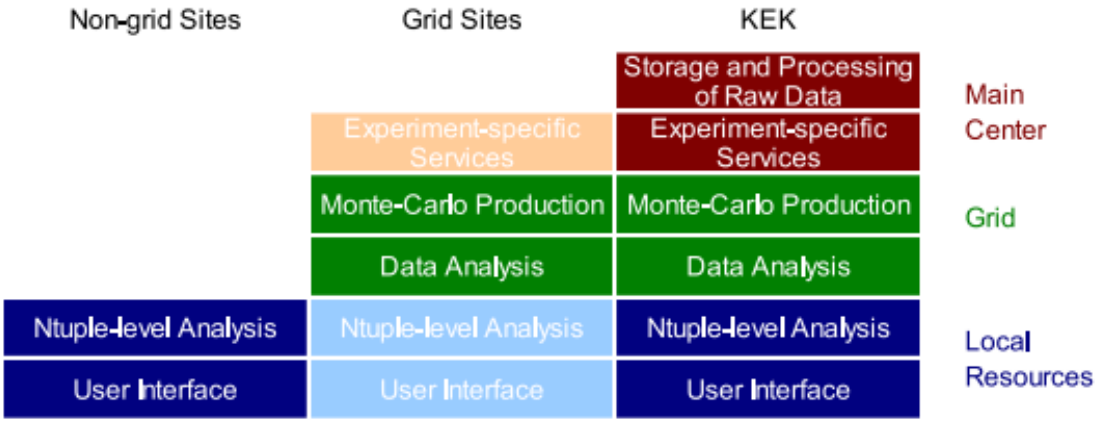


Figure 4.2: Tasks of the computing facilities. As ntuple-level analyses are processed frequently they need a fast turn-around time and are therefore preferably operated on local (grid-enabled) sites [25].

for physics analyses. Through the grid middleware the users have access to the resources and data available on the grid sites. Also all the simulated data samples (MC data) are available in mdst format on the grid sites. Physicists usually process the large mdst files on the grid sites using appropriate grid software with the output, ntuples, transferred to the local sites. The compact size of the received ntuples allow to process them with the local resources. Here the final analysis takes place (see Figure 4.1).
The computing system, even if largely designed as a distributed system, has dedicated sites for special tasks. The KEK computing center, located next to the Belle II detector, is the main computing facility and with the processing and storing of raw data it plays a major role in the architecture. The replication of the created mdst files is managed manually by the Distributed Computing groups and highly depends on the size of the site and the physics topics the users at that center are interested in. A structured overview of the tasks assigned to the individual components of the computing infrastructure is given by figure 4.2.

4.2 (Belle)DIRAC

DIRAC (**D**istributed **I**nfrastructure with **R**emote **A**gent **C**ontrol) is a software framework that provides a common interface and complete solution among heterogeneous computing resources for one or more user communities requiring access to distributed resources. It overlays complex interoperability structures between diverse computing systems and enables an optimized, transparent and reliable usage of the resources [3].
DIRAC was originally developed for the LHCb experiment at CERN to manage all its computing operations on the grid. Its modular architecture allows to independently extend the software packet in a way to adjust it to the requirements of the communities using the

grid. In case of Belle II, the extension module is called BelleDIRAC.

4.2.1 Architecture Overview and design principles

Although DIRAC was developed for the LHCb experiment, it is designed to be a generic light grid system which allows to orchestrate grids of up to several tens of thousands processors with the help of its integrated Workload Management System. The DIRAC project consists of a large number of components (the most important ones will be described at a later point in time). They can be grouped in the following four categorie [43]:

- **Resources:**
  Possible resources can be individual PCs, computing farms with various batch systems and computing elements in the EGEE (Enabling Grids for E-SciencE) grid. DIRAC doesn't have any complex Storage Element service capable of managing multiple disk pools or tertiary storage systems (except for disk storage managed by a POSIX compliant file system). Some storage systems are accessible through different protocols which adds redundancy to the infrastructure. DIRAC data access tools will choose the optimal protocol [44].

- **Services:**
  DIRAC is build upon loosely coupled services. They keep the system state and help to carry out workload and data management tasks. Services are passive components which are only reacting to the requests of their clients. All services and clients are built in the DISET framework (see Section 4.2.2) and each service has typically a MySQL database back-end to store the state information. Services are installed on a limited number of sites with a well-controlled environment which ensures a high availability.

- **Agents:**
  Agents are light and easy to deploy software components which run as independent processes to fulfil one or several system functions [44]. They run on different environments and either close to the corresponding services and watch for changes in the service states and react accordingly or as part of a job executed on a worker node.
  Worth mentioning are the so called '*Pilot Agents*'. The pilot agents are submitted automatically to the grid sites by an Agent Manager (which works close to Workload Management System, see Section4.2.3). The pilots check the execution environment on the nodes where they are running and send the resource description to the Task Manager (Matcher service). The Task Manager receives also the tasks from the users and controls the queue of user tasks. Based on the resource description it chooses the appropriate user job/task for the working node [45]. This is the so called "pull" scheduling paradigm which is a central principle of the DIRAC middleware. The use of the job pilot model is now widely accepted as a way to hide the fragility of the underlying distributed resources and increases the efficiency of their usage [44].

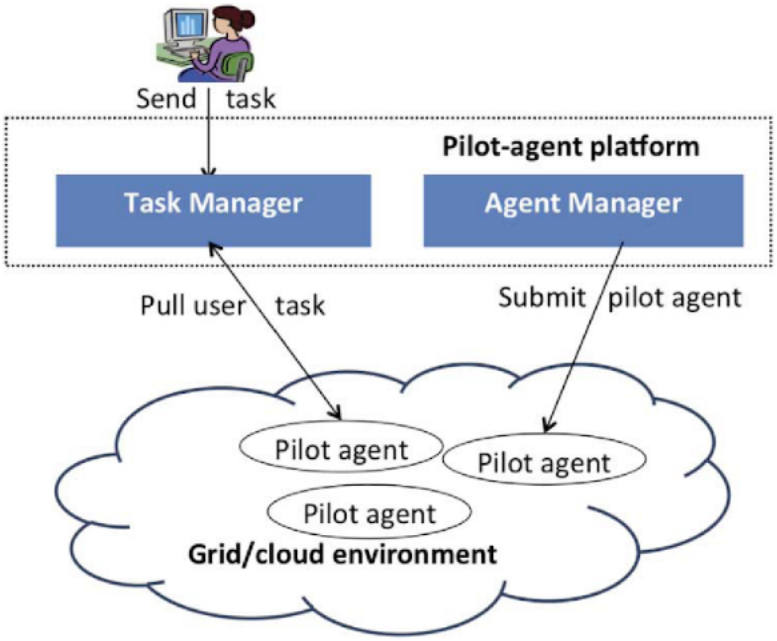


Figure 4.3: Schematic design of a pilot agent platform as part of the "pull" scheduling paradigm [45].

- **Interfaces:**
  DIRAC provides several python based programming interfaces (APIs) on the top of which user specific applications can be developed to give users access to the grid according to their modalities. Examples for such high level specialized applications are gBasf2 (for the Belle II community, see Figure 4.7) and the job management tool GANGA (widely used in ATLAS, LHCb and other communities) [46].
  DIRAC provides also Web interfaces for users and system managers to monitor the system behaviour and to control the ongoing tasks. To secure the acces to the DIRAC Web Portal the X.509 public key infrastructure is used [43].

The DIRAC middlerware established also some design principles which should be followed by its components and by the system itself.
DIRAC pursues the paradigm of a Service Oriented Architecture (SOA). That means the software model should integrate a collection of services most suitable for loosely coupled distributed applications. These services communicate with each other to offer interoperability between distributed systems, therefore enabling interoperability between diverse systems and reducing the complexity of administering heterogeneous systems [34].
For a distributed computing environment, which is intrinsically unstable, redundancy is necessary. In case of temporarily unavailable services, data transfer failures or storage problems redundancy should avoid to loose results obtained after consumption of a large amount of computing resources. In DIRAC redundancy is achieved in several ways [44]. First, the essential information of a submitted job is duplicated at several services, so that

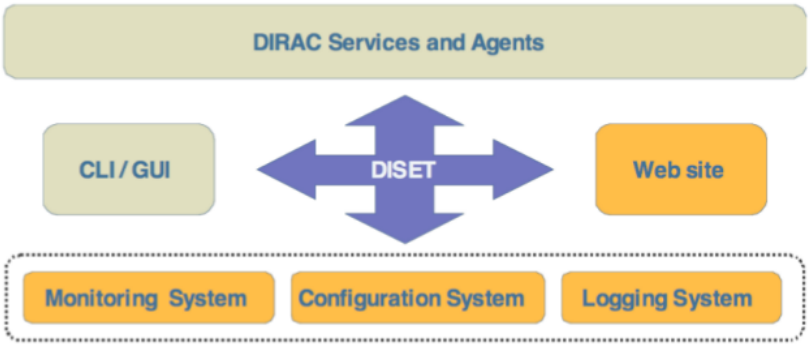


Figure 4.4: DIRAC framework components [44]. The framed components (*Monitoring, Configuration and Logging System*) are primarily responsible for gathering internal information from agents and services. The DIRAC Web site enables the user to utilize services over a web browser, platform independently.

at least one backup copy is available to client requests. Second, the most important service operations where a failure would mean a crash of the whole process/job have a fail-over recovery mechanism. In case of failure these mechanisms relaunch the operations. Third, in case of data transfer failure (e.g. data file upload) data management methods store the affected files temporarily in some spare storage element.
Another important design feature of DIRAC is the strict separation of static and dynamic system state information. The static configuration data is accessible for all clients over the Configuration Service (CS) and gives long-lived information about the grid components (e.g. software versions, names and software configuration of the working nodes etc.). The dynamic information is in most cases looked for at its source (directly from the batch system or the computing element) and describes the current availability and status of the source.
On the other hand DIRAC puts very low requirements on the sites asking for no special support from the site managers chasing its principle of a light-weighted grid system. As a consequence this allows the exploitation of numerous sites providing in conjunction an immense resource pool.

4.2.2 The DIRAC Framework

The DIRAC framework has been built with intention to be a generic system for various communities. It is designed in a way that it can be easily extended. Figure 4.4 gives an overview of the essential components of the DIRAC framework. The communication between these components is controlled by the DISET protocol which enables secure and standardized information transfer.
The five fundamental components of the framework are:

- **DISET (DIRAC Secure Transport):**
  The original DISET is based on the XML-RPC protocol [44] and was enhanced with

GSI(Grid Security Infrastructure) authentication mechanism. Due to intensive use of the system a proprietary DISET protocol was introduced. The *proprietary* DISET, as successor of the original framework, shows an increases in the efficiency of the client/service interactions and allows to change the configuration of the authorization rules which can be specified for each service. DISET provides a complete software solution for creating services on DIRAC. It has a build-in support for multiple threads and automatic logging of the history of the requests [44]. It does not just provide RPC (Remote Procedure Call) functionality but also allows to transfer files in the same interface. For developers it allows to build portals that act as single access points for multiple service availability, just using one single authentication step.

- **Web Portal Framework:**
  The DIRAC Web portal framework is the base layer for building Web interfaces to DIRAC services. The authentication works through user grid credentials which are extracted from the grid certificate uploaded on the user's browser. The DISET framework is used as secure transport layer in the background to redirect client requests and collect service responses. Web interfaces allow job, site and resource monitoring in a uniform way, independent of the user's environments and platforms.

- **Configuration System (CS):**
  The static configuration data is made available to all the clients via the Configuration Service (CS). It is also based on the DISET framework. As it is the backbone of the whole system information it needs exceptional reliability. It is structured in a single master service which takes care of all updates and multiple read-only slave services which are distributed geographically. When users make a request the queries are managed in a load balancing way to provide data consistency and good scalability properties [44].

- **Logging and Monitoring Services:**
  The DIRAC logging system accounts for the state information of the distributed components, including third party services provided by the sites, with its possibility to report where it encountered system failures. The logging output can be done over standard output, log files or an external service.
  The Monitoring Services collects activity reports from all the DIRAC services and some agents [44]. This information is presented in a variety of ways (eg. historical plots, summary reports, etc.) [36]. Together with the Logging Service it provides a powerful sanity checking tool for the whole system.

4.2.3 Workload Management System (WMS)

The Workload Management System is the core component of the DIRAC interware. It has to redirect the submitted jobs to the appropriate sites. In contrast to the push model which attempts to centrally optimize the allocation of jobs to resources, in the pull model, as used for the DIRAC WMS, the computing resources request jobs from a large job pool

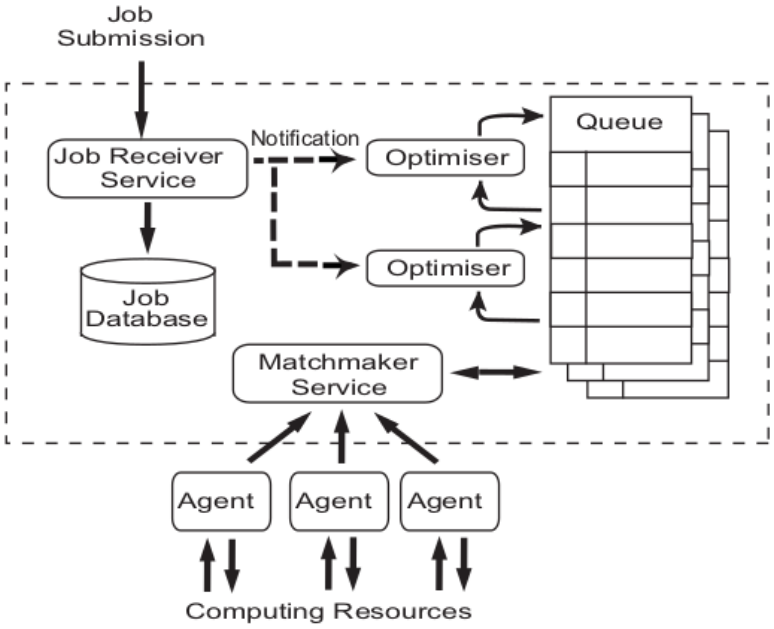


Figure 4.5: Job Management Services [47].

[47].
The WMS can be divided in two distinct parts: First, the Job Management Service (JMS) as part of the central services and second, a number of agents which run close to the available computing resources.

Job Management Service (JMS)

The JMS consists of the following deployed components (see Figure 4.5) [47]:

- **Job Receiver:** gets the job submissions from the client and registers them to the Job Database. At the same time it informs one of the Optimizers about the incoming jobs.

- **Job Database:** it contains the basic job information.

- **Optimizers:** it sorts the jobs into queues depending on the queue states, job load, resource availability using a range of techniques. Optimizers are highly customizable and multiple optimizers can coexist, offering each user community the best job scheduling algorithm.

- **Matchmaker:** it finally matches the jobs to the resources from the global job queues which have been sorted by the optimizers.

WMS Agents

As part of the DIRAC WMS the agents are light daemons running close to the computing resources. They check the local state of the resources, the available software packages, store job parameters in a local database and query jobs from the Matchmaker. All the information which is important for job tracking and job sanity checking is sent to the Job Monitoring Service, a component of the JMS.

The JMS and the agents contribute to the DIRAC pull paradigm for a successful and reliable job submission which overlay the complex and fragile grid network. The DIRAC pull strategy has the following phases [47]:

1. Agent (acting on the local nodes) detects free computing resource
2. Agent requests job from Matchmaker
3. Matchmaker checks queues for appropriate match
4. Matchmaker commits best matching job to Agent

As next step on the working node a wrapper script prepares the execution environment and downloads necessary data. The so called 'Watchdog process' periodically checks the state of the job and sends the gathered information to the central Job Monitoring Service.

4.2.4 Data Management System (DMS)

An essential criterion for an efficiently working grid system is the high availability of the large data volumes stored on distributed and heterogeneous storing elements. Therefore a reliable Data Management System (DMS) is needed. The highest level DMS components are responsible for the automatic data distribution system and data integrity checking system. The data distribution system selects the optimal destination storages where the (replicated) data should be sent. The DIRAC data integrity checking system provides corresponding tools centred around the integrity database which accumulates reports of inconsistencies or failures on the storages or file catalogues. In case of an issue the reports are sent to different DIRAC tools which try to repair the failure or at least to circumvent the problem and pass the control to the human Data Manager.
The actual file transfer is managed by the Replica Manager which is in charge of the basic file management operations: uploading, replication and registration. For the file transfer the FTS (File Transfer Service) is used or some specialized DIRAC agents. The Replica Manager masks the diversities of different storage systems and is able to handle different file catalogues at the same time [44].
The clients of various kinds of storage and file catalogues are called the low level tools. The main file catalogue used for Belle II data is LCG File Catalogue (LFC) (see Section 4.3). DIRAC creates an uniform API for access to various types of storage and catalogues and provides one simple storage element service for data transfer [36].

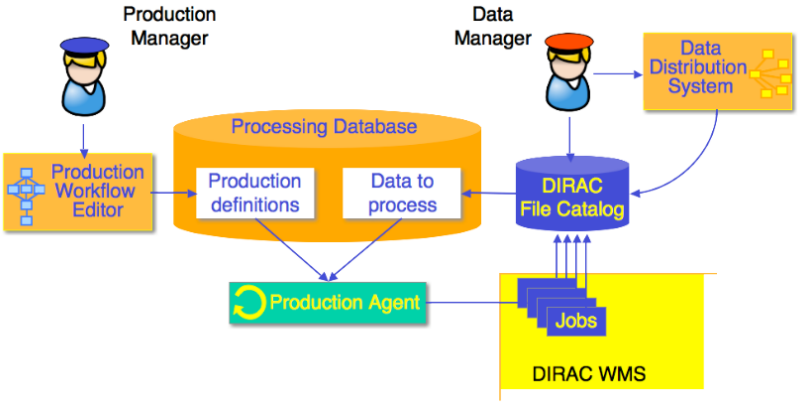


Figure 4.6: Production Management System components [44].

4.2.5 Production Management System

For higher level applications, e.g. making the second replica of the raw data in the main computing centres, reconstruction of raw data and distribution of the resulting mdst data to other Tier-1 centres, MC production campaigns, the DIRAC system provides the Production Management System (see Figure 4.6).
As a first step the workflow is prepared. The workflow contains the exact sequence of operations which should be processed and all the production definitions and data information which are required for a successful job submission. This information is stored on the Processing Database and the appropriate Production Agent generates and submits the jobs to the WMS which on its parts calls the DMS for the control of the large data stream.

4.3 LFC

To be able to access the distributed data a file catalogue is needed to map the logical file names (LFN)[2] to the physical file names (PFN)[3]. For the BelleDIRAC project the well-proven LCG File Catalog (LFC) is used. It allows to overlay all namespaces of the grid storage elements to a big one and uses a classical relational database as back-end. In the LFC, a given file is represented by a Grid Unique IDentifier (GUID). Thanks to the GUID a file can be replicated at different sites but appears as an unique logical entry in the LFC [48].
The LFC is used as centralized catalogue containing all managed files which simplifies the operations required to obtain replica information. The major drawback of this centralized architecture, the single point of failure, is circumvented by adding to the single master write accessible instance multiple read-only mirrors to provide redundancy in the availability of file information [49].

[2]Platform-independent descriptive name(s) of a file.
[3]Platform-dependent name(s) of a file exclusively assigned to one LFN.

The DMS integrity checking system (see Section 4.2.4) marks the files which no longer exist so that they are no longer 'visible' in the LFC and are eventually deleted from the catalogue. DMS agents periodically check the consistency of the LFC contents and remove orphan physical files on the storage resources to efficiently use the available storage elements.
The LFC provides a Unix-like command line client as well as LFC C APIs (python wrapper provided) which allows to integrate it with the DIRAC WMS and other data processing components.

4.4 AMGA

The ARDA (A Realization of Distributed Analysis for LHC) Metadata Grid Application (AMGA), primarily developed at CERN [50], is the employed metadata catalogue for effective file and file content search for the Belle II project. Metadata is 'data about data' and as such used to find files or datasets with the desired attribute values. AMGA uses a relational database back-end to store metadata and supports several database products (MySQL, ORACLE, PostgeSQL, SQLite) [36]. The metadata in AMGA is structured in lists of attributes associated with entries. Each entry stands for a file/dataset [51]. Reasonable attributes have to be selected to keep the AMGA server workload well scalable and the content search as fast as possible. For the BelleDIRAC following attributes are used [36]:

- GUID (file Globally Unique Identifier)
- LFN (logical file name)
- file status
- number of events
- experiment number
- lowest run and event number
- IDs of parent files
- date and time of creation
- ID of site where the file was created
- (Belle)DIRAC job ID

The implementation of AMGA in BelleDIRAC uses the AMGA Python APIs, which were cooperatively developed by the AMGA and Belle II group.
For an AMGA metadata query the user simply has to ask for the desired attribute values eg. experiment number, number of events. AMGA also allows to make queries on a subset of runs or experiments which accelerates the searching process.

4.5 gBasf2 Client

gBasf2 (grid Basf2, see more about Basf2 in Section 4.6) is the BelleDIRAC command line client for job management at Belle II (submits, monitors, deletes jobs) data management (manages data and metadata produced by jobs) and other utilities (proxy management, information about grid, etc.) [52]. The gBasf2 client is built on the (Belle)DIRAC API and the AMGA API. Everything that concerns the metadata management is based on the AMGA API functionalities while the DIRAC API provides the base for job and dataset management.
The gBasf2 as a software package can be grouped into two different script types. The first is a set of bash (Bourne again shell) scripts. They are responsible for the successful environment setup. The second type, which predominates, are python scripts. The set of executable python scripts can be called directly from the command line and therefore act as a command line interface allowing the user to set different options and flags to the tasks. They are stored in the */bin* directory.
All scripts that execute the core functionalities and the business logic are assembled to python modules using the DIRAC and/or the AMGA APIs. The python scripts are stored in the */controllers* and */helpers* directories.

The gBasf2 Client is the entrance door for the users to access the grid resources and shields users from the underlying complexity of the Belle II Grid. The user prerequisites are kept low: a SL6/SLC6 (Scientific Linux) operation system, a valid grid certificate and a Belle VO membership registration are required for submitting grid-based Basf2 jobs.

4.5.1 Structure of gBasf2

The gBasf2 client is structured (apart from the grouping in bash and python scripts) according to its main packages:

1. **BelleDIRAC/Client/bin/**
2. **BelleDIRAC/Client/controllers/**
3. **BelleDIRAC/Client/helpers/**
4. **BelleDIRAC/gbasf2/lib/**

BelleDIRAC/Client/bin/

This directory contains the code for the gBasf2 command line interface (CLI). The user calls these modules with the necessary arguments and the modules in *BelleDIRAC/Client/bin/* parse the command line input and pass the result to the terminal controller. For all of these commands help instructions are available (by adding *–help* to the command line). From the functionality perspective the gBasf2 modules can be divided into three groups:

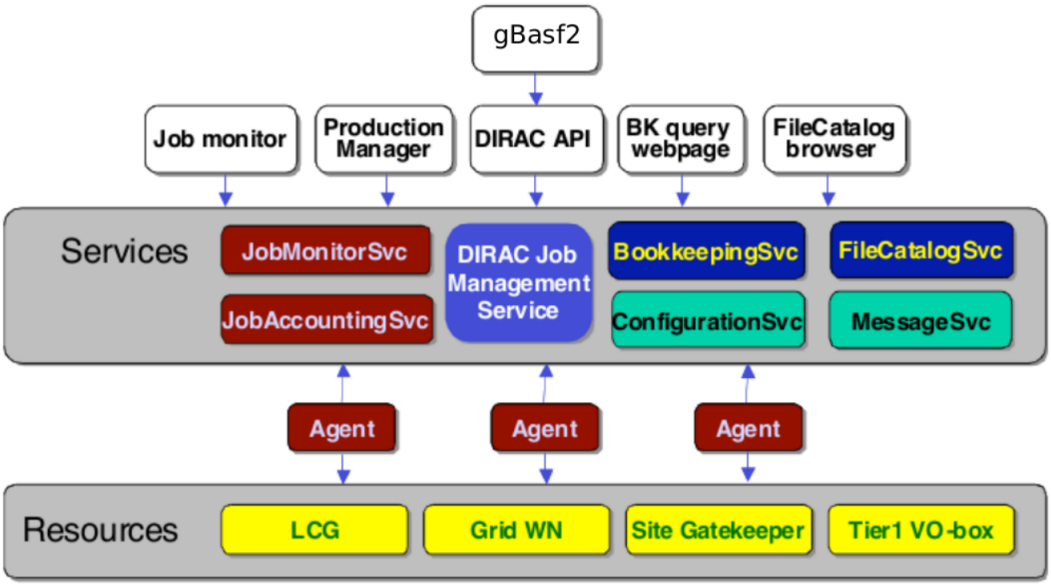


Figure 4.7: DIRAC architecture overview for Belle II. The DIRAC APIs allow to use the grid system for the specific needs of the grid communities (e.g. gBasf2 Client for the Belle II community) [44]

job components, dataset components and others (e.g. scripts for authentication, determine site status, etc.).

BelleDIRAC/Client/controllers/

In this directory the terminal controllers are located. Their main responsibility is to communicate with the user (print output, retrieve output) and pass the input to the core functionality. There are three kinds of controllers [52]:

- *datasetCLController.py* - contains all communication related with datasets
- *projectCLController.py* - contains all communication related with jobs and projects
- *utilCLController.py* - contains all other communication

BelleDIRAC/Client/helpers/

The functions used by the controllers and by the command line scripts (see Section 4.5.1) are placed here [52]:

- *auth.py* - authentication functionalities
- *optutil.py* - argument parsing modules

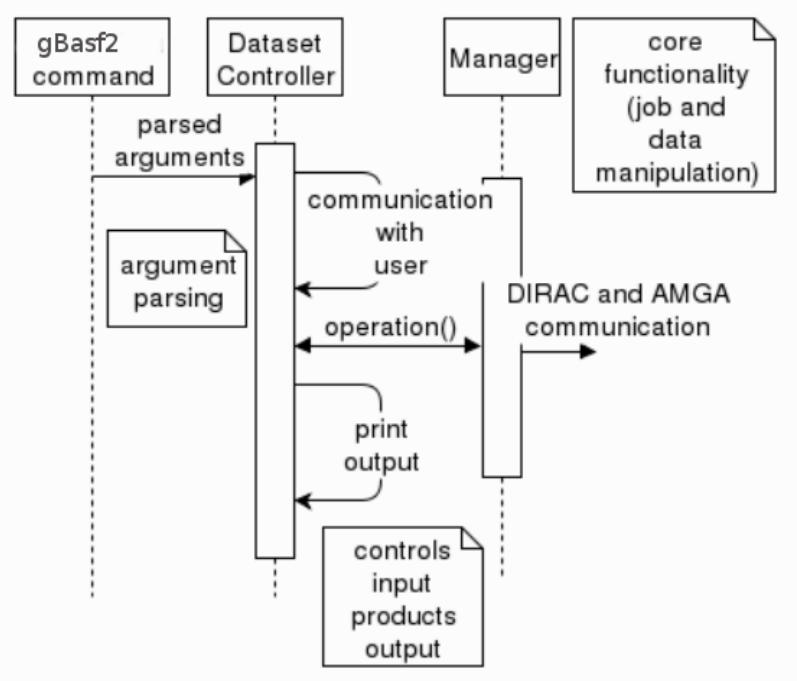


Figure 4.8: Example of the interaction of the various modules when calling a gBasf2 command [52].

- *terminalDisplay.py* - functionality for terminal output creating (tables, etc.)
- *texttable.py* - external module for table creation

BelleDIRAC/gbasf2/lib/

The core functionalities, bundled to managers, are located in the *BelleDIRAC/gbasf2/lib/* directory. These managers interact both with DIRAC and AMGA through the respective APIs. There are three different type of managers [52]:

- *gbasf2/lib/job/manager.py* - contains core functionalities for job and project management (delete, kill, etc.)
- *gbasf2/lib/job/information_collector.py* - contains functionalities to collect all information about jobs and projects
- *gbasf2/lib/ds/manager.py* - contains core functionalities for metadata and dataset management (get, remove, etc.)

4.5.2 Job Submission via gBasf2

The most prominent gBasf2 client command is the *gbasf2* tool which allows the user to submit jobs to the grid. For the submission workflow see Figure 4.8. Given the prerequisites discussed in Section 4.5 the user has to execute the following commands before having access to the *gbasf2* tool:

`source path_to_your_gbasf2_installation_/BelleDIRAC/gbasf2/tools/setup`
for setting up the appropriate environment

`gb2_proxy_init -g belle`
for creating the proxy certificate from the valid grid user certificate

For the actual job submission the user types "*gbasf2*" followed by the mandatory arguments:

- the relative path to the steering file
- `-p ‘projectname‘`
- `-s ‘release‘` (the Basf2 release which should be used when processing the steering file)

Important to mention is the ‘project‘ concept of gBasf2. A project is a basic unit for all modules that encapsulates several logically interconnected jobs and files [36]. Each job has a project it belongs to and each time a bunch of jobs is submitted via the gBasf2 client a new project is created. Depending on if the project is shared or private it is associated to multiple or single users.

There is also a list of optional arguments. The most relevant for gBasf2 client user are:

`--priority`; Job priority: 0 (default)-10

`-i DSPATH, --input_ds DSPATH`; Path to input dataset

`--input_dslist FILENAME`; Input dataset list from file

`-f FILES [FILES ...], --input_sandboxfiles FILES [FILES ...]`; Supplemental files to be included in input sandbox

Instead of defining all options in the command line it is possible to store them in the steering file itself (that would reduce the command line to: ‘`gbasf2 steeringfile.py`‘). However, if command line arguments are given too, the options set in the steering file are overwritten.

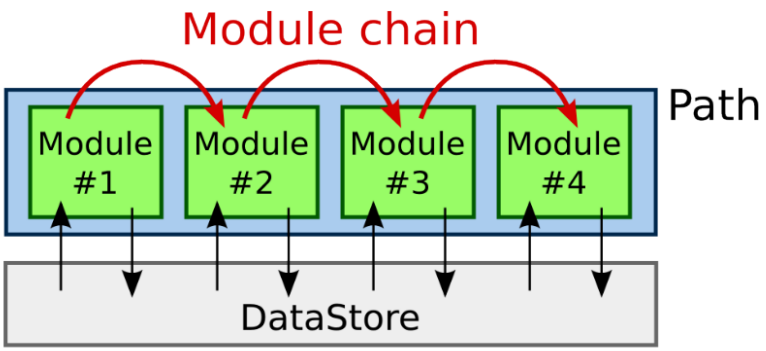


Figure 4.9: Schematic view of the processing flow in the Belle II software [53].

4.6 Basf2 analysis tool

The Basf2 (Belle II analysis software framework) is the last link in the Belle II software infrastructure and at the same time the backbone for physics analyses. With the appropriate Belle II software installed it works both on local computers and with the help of the gBasf2 Client (see Section 4.5) on all dedicated grid sites. Basf2 as simulation tool based on several simulation frameworks (e.g. Geant4) is responsible for the creation of Monte Carlo (MC) simulated data. For simulated data (MC) it offers the convenient reconstruction software by simulating the whole Belle II detector and physics processes that take place. The Basf2 software is organised into 'packages', each of them providing one precise functionality. A typical data processing chain consists of a linear arrangement of smaller processing blocks, called 'Modules'. This linear arrangement is called 'Path'. When data from the DataStore is processed, the framework starts by the first module in the path and proceeds with the module next to it. The modules are executed one at a time, exactly in the order they were placed into the Path [53] (see Figure 4.9). The file in which the path is processed is called *steering file* (see example in Figure 4.10).

The Basf2 software is written in C++17 "under the hood" [54] but has an extensive interface to the python 3 scripting language: All configuration and steering (see steering file in Figure 4.10) is done via python and in principle also simple algorithms can be implemented directly in python [53]. The DataStore is the common storage where all the data processed by the modules of the path is stored. Each module has read and write access to the DataStore [53].

Basf2 can read and/or create following file types [54]:

```
import sys, os
import basf2 as b2

path = b2.create_path()
# set the number of generated events to 10
setter = path.add_module("EventInfoSetter", evtNumList=[10])
# set the decayfile for generating events
generator = \
path.add_module("EvtGenInput",userDECFile=Belle2.FileSystem.findFile('decayfile.dec'))

from simulation import add_simulation
# propagate the generated stable Monte Carlo particles through the detector
# and simulate their effects in the material
add_simulation(path)

from reconstruction import add_reconstruction
# reconstruct tracks, clusters and PID information form the simulated hits
# and energy depositions in the detector using the simulated data
add_reconstruction(path)

from reconstruction import add_mdst_output
# save the generated, simulated and reconstructed events
add_mdst_output(path, filename='some_events.mdst.root')

# process the path defined above
b2.process(path)
```

Figure 4.10: Steering file which generates 10 events, simulates, reconstructs and finally saves them in the mdst format. The path base is created by the calling method `create_path()` followed by the necessary modules added to the path.

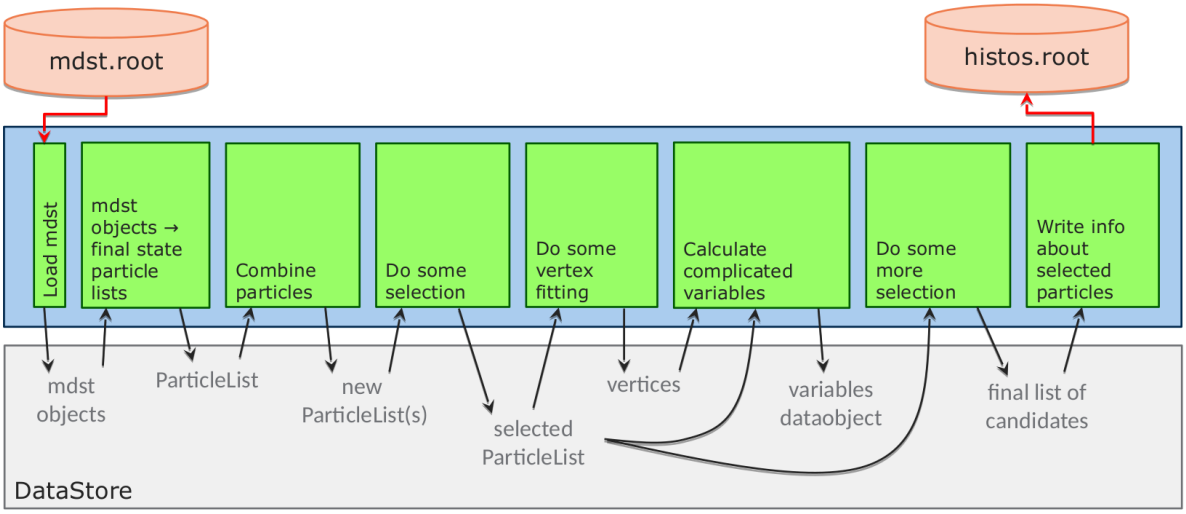


Figure 4.11: Typical path for a Basf2 analysis job [54].

- **dst** (**d**ata **s**ummary **t**able, basically a special ROOT [4] file) files which contain Basf2 objects and populate the DataStore
- **mdst** (**m**icro **d**ata **s**ummary **t**able) files, a compressed version of **dst** files. They are the most common input files for user analysis package scripts.
- The output files in final analyses are mostly in the 'normal' ROOT file format containing a TTree, TNtuple or histograms. As final stage of real/simulated data process the information from files of this filetype is plotted in the form of histograms, 2D-plots, etc., using numerous plotting and fitting tools. At the end these graphical visualizations show whether the applied physical model coincide with the actual data or not.

Figure 4.11 shows a typical Basf2 path for an analysis job with a selected mdst ROOT file as input file. In the DataStore all mdst objects (e.g. each particle track) are read from the ROOT file. These mdst objects are the main ingredient for data processing by the Basf2 modules. The method `VariablesToNtuple` finally writes the result from the analysis process in the form of NTuples to the '*histos.root*' output file. The same workflow structure found in figure 4.11 is implemented in the sample steering file in Figure 4.10 in form of python code.

[4] A machine-independent compressed binary file format [42].

Chapter 5

Minimal BelleDIRAC Grid System (MBGS)

The BelleDIRAC interware with the gBasf2 command line client for submitting grid-based Basf2 jobs is a great solution specific to the Belle II user community requirements. But due to its complexity testing DIRAC extensions like gBasf2 becomes really challenging. Writing tests for distributed application such as gBasf2 is challenging, because due to its distributiveness, many of the gBasf2 functionalities cannot be tested as a standalone module. Many errors could occur during running and it could be very hard to find their source. As the BelleDIRAC interware is deployed on numerous grid sites occurring errors during testing could lead to sensitive breakdowns and enhance the fragility of the grid system. To avoid such large scale interferences the so called Test Driven Development (TDD) process was used to implement several parts of gBasf2 functionality [55].
This kind of programming approach relies on the repetition of a very short development cycle: the developer writes as first step a test for the desired new function which hasn't been implemented yet. This leads to an obvious failure of the test. As next step the developer produces a minimum amount of code to pass that test, and finally refractors the new code[1]. However the TDD programming approach does not provide sufficient testing in situations where the implications of newly developed tools on the grid middleware could be analysed. Therefore a simple and time-saving solution is sought.
The 'Minimal BelleDIRAC Grid System' presented in this chapter could be one of the solutions for making tests of new BelleDIRAC components easier and more flexible. As the MBGS is based on the Docker [56] software which uses operating-system-level virtualization it allows to run even large software packages as the BelleDIRAC interware on an isolated environment. This restricted environment could give the developers the assurance to not corrupt any other systems or applications albeit offering exactly the same features and tools available on the large grid system.
It can be installed on a single local computer and in addition to the prerequisites for the

[1]For further information see: `https://confluence.desy.de/display/BI/Computing+Gbasf2TestingSystem`

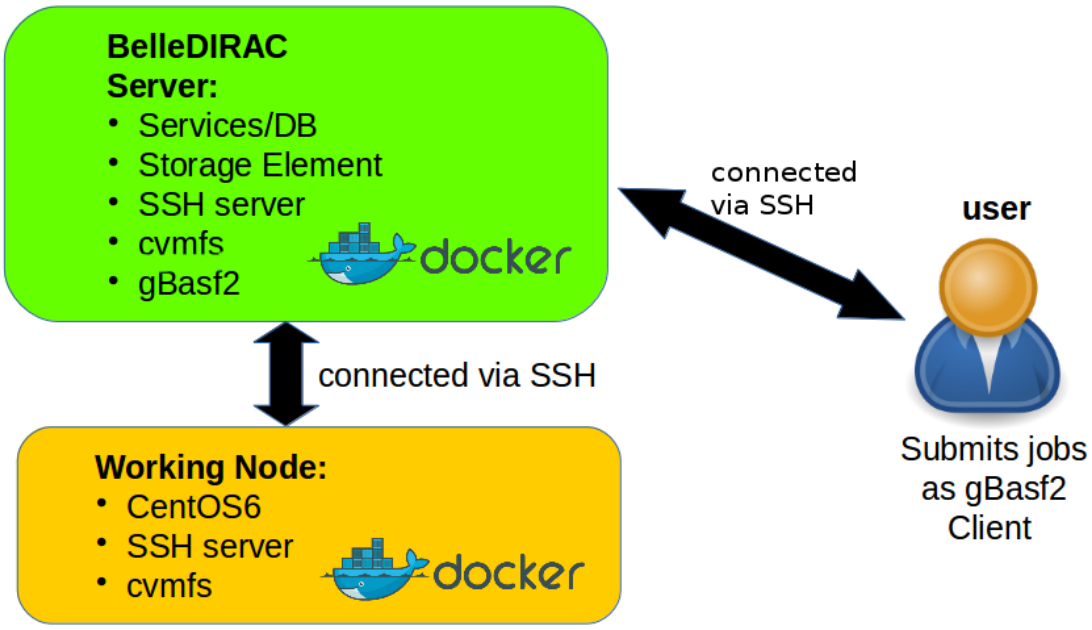


Figure 5.1: Schematic view of 'Minimal BelleDIRAC Grid System'.

gBasf2 Client installation it only requires the Docker software (version $\geq$ 18.06.1) to be installed beforehand.
An important feature which the MBGS can offer to the user/developer is the free access and insight to the (Belle)DIRAC Configuration Service where static system state information is managed. Up to now this is only possible for dedicated DIRAC administrators which have direct access to the central (Belle)DIRAC configuration servers at KEK. The MBGS could therefore be an excellent instruction tool for better understanding on how the whole grid system for Belle II is configured. And of course a playing field for all developers working on extensions regarding the configuration system and changes on the base level of the BelleDIRAC software.

5.1 General Setup and Workflow

Figure 5.1 shows the general workflow for job submission on the 'Minimal BelleDIRAC Grid System'.
The core of the system is the 'BelleDIRAC' server. As mentioned it is build upon a docker container with a CentOS 6 operating system running on it. All the BelleDIRAC core services and databases are running on it and it acts as an SSH server too (this is needed for the communication with the working node). The CernVM File System (cvmfs) as software distribution service is also bind mounted on the 'BelleDIRAC' server (see more in Section 5.2).

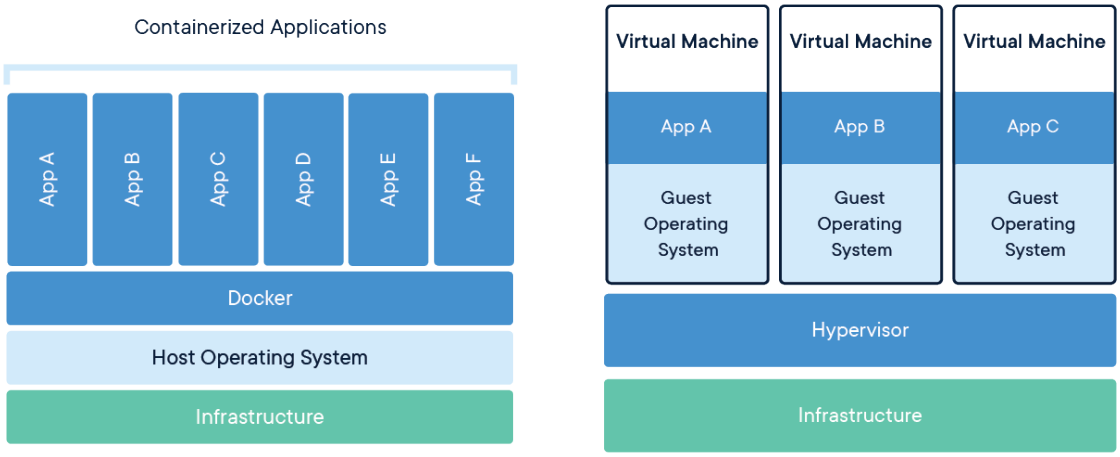


Figure 5.2: In contrast to usual virtual machines Docker containers share the machine's OS system kernel and therefore do not require an OS per application, driving higher server efficiencies [57].

The working node consists of a minimal CentOS 6 docker container with cvmfs bind mounted and a SSH server running on it.
The last component of this system is the user: For simplicity we use the 'BelleDIRAC' server for it. The user just has to build up a SSH connection to the core server, source the gBasf2 environment and initialize a user proxy. With this emulation the 'BelleDIRAC' server acts both as main administrative component and as gBasf2 Client environment.
When the user finally submits a job (using the '`gbasf2 sterringfile.py -p project -s release + further options`' command) the Workload Management System on the 'BelleDIRAC' server checks the availability of the requested resources on the working node (through the pilot agents). If the requirements are fulfilled the job is sent to the working node where a Wrapper script prepares the execution environment and installs any missing software packages if necessary. The job is then ready for execution.
After successful execution the user can download the remote output files of the job by means of the '`gb2_ds_get projectname`' command.

5.2 Tools and Methods

5.2.1 Docker container

The method of containerization is the major tool on which the whole MBGS is built on. The Docker OS-level virtualization software allows to create multiple isolated user-space instances, called containers. As these containers run by a single operating-system kernel on the top of the host operating system they are more lightweight than virtual machines (see Figure 5.2). Besides Docker other software mechanisms like Singularity offer OS-level virtualization implementation. Decisive for the use of Docker was its good documentation,

high flexibilitiy and its wide range of features. The most important ones for the setup of MBGS are:

- **Docker network:**
  Docker allows to create virtual networks that can be configued to provide complete isolation for containers. This is very useful for the MBGS which operates on various containers linked through a network which guarantees safe interoperability.

- **Option:** `--add-host`
  This optional feature allows to add custom host-to-IP mapping in the sense that for the requested host an entity is automatically added in the container's /etc/hosts file. For the MBGS this option allows that the main BelleDIRAC container and the working node container recognize each other.

- **Option:** `-h` *`hostname`*
  With this flag a specific container hostname can be set. Without this flag the container hostname corresponds to the container ID, a twelve characters long string. This option prevents the user to confuse the BelleDIRAC container with the working node during the installation of MBGS (see more in Section 5.3).

- **Bind mounts:**
  With this feature it is possible to mount a directory or file on the host machine into a container. The file or directory does not need to exist on the Docker host already. It is created on demand if it does not yet exist [58]. For the MBGS setup the bind mount option is necessary to connect with CVMFS (see more in Subsection 5.2.4) both for the BelleDIRAC server and the working node.

5.2.2 CentOS 6

CentOS (**C**ommunity **E**nterprise **O**perating **S**ystem) is a Linux distribution which delivers a robust open source ecosystem. CentOS is free software. Supplementary it is compatible to SL6 (**S**cientific **L**inux **6**), an operating system developed specifically for and by large particle physics communities like Fermilab and CERN and also supported by the Belle II computing system. Due to these advantageous features CentOS 6 was chosen as base operating system for the MBGS (since also the gBasf2 command line client still relies on SL6). As both CentOS 6 and SL6 are reaching their end of support (end of 2020 [59]) the MBGS should be moved to the more actual version CentOS 7 in near future.

5.2.3 Self signed certificates

To have access to the MBGS BelleDIRAC Web App both as administrator and standard user a valid certificate is required. For that reason a self signed certificate is created which is the optimal and free of charge solution for non-production applications like the MBGS. To create the self signed certificate the OpenSSL software tool is used. In the Dockerfile

included in the software package for the MBGS installation (see Section 5.3) the production of a self signed certificate can be achieved by the following command:

```
RUN openssl req -x509 -sha256 -newkey rsa:2048 -keyout hostkey.pem -out
hostcert.pem -days 1024 -nodes -subj '/C=DE/O=TUM/CN=host'${SERVERNAME}
```

(with '/C=DE/O=TUM/CN=host'${SERVERNAME} being the custom Name information needed for a SSL Certificate.)

The self signed certificate is automatically created when building the Docker container for the BelleDIRAC server (see Section 5.3). After successful setup of the MBGS the user can access the BelleDIRAC Web App from the web browser by calling up `https://localhost:8443/`. The BelleDIRAC Web App gives the user the possibility to control the running jobs (reschedule, kill, delete, etc.) and as 'dirac_admin´ to manage (start, stop and restart) the various BelleDIRAC components and to modify the predefined configurations of the MBGS BelleDIRAC server.

5.2.4 Cern Virtual Machine - File System

The Cern Virtual Machine - File System (CernVM-FS or CVMFS)[2] provides a scalable, reliable and low-maintenance software distribution service [60]. It is heavily used in the HEP (High Engergy Physics) community. As already mentioned in subsection 5.2 it is bind mounted both on the BelleDIRAC server and the working node container. It acts as a read-only file system in user space. For the MBGS it allows to deploy and install software artefacts (e.g. binaries, libraries, slowly-changing configuration files) for job submission (on the BelleDIRAC server) and for job execution (on the working node). The MBGS setup benefits from the bind mounted CVMFS since it keeps the container size manageable and software deployment always up to date.

5.2.5 SSH Deployment

The Secure Shell (SSH) network protocol is intended to provide secure encrypted communication between two parties over an insecure network [61]. For the MBGS SSH is used for multiple implementations and plays a major role for the working node since it is defined as SSH computing element (this information can be gathered from the config.cfg file included in the software package for the MBGS installation (see Section 5.3)). SSH daemons (sshd) are both deployed on the BelleDIRAC server and the working node and listen for connections from clients enabling secure communication between the two containers.
The BelleDIRAC server authenticates the working node by the use of a manually generated

[2]For further information see: `https://buildmedia.readthedocs.org/media/pdf/cvmfs/latest/cvmfs.pdf`

public-private key pair which allows to log in without having to specify a password. This password free SSH authentification is necessary for successful automatic job submission where neither user nor administrator is prompted to type site specific passwords.
Another helpful use case of SSH is the logging into the BelleDIRAC server as gBasf2 Client for job submission. This makes it possible to use one and the same container for two different applications, namely as BelleDIRAC server (with administrative rights) and as gBasf2 Client environment (with standard user rights).

5.3 Installation of the MBGS

Requirements for the MBGS are the Docker software (version $\geq$ 18.06.1), a valid grid user certificate, access to the CVMFS file distribution service and a Belle VO membership.
The step-by-step installation instructions for the MBGS can be found in the appendix A and on the web:
`https://confluence.desy.de/display/BI/Minimal+BelleDIRAC+Grid+System`

5.4 Improvement Opportunities and Outlook

The MBGS is the first successful attempt to virtualize the whole BelleDIRAC system using Docker container software and offers great features for developers as well as for interested users. For developers it can be used as testing environment which is completely equivalent to the large grid system with the advantage that nothing can be corrupted since the MBGS works as an isolated system. Especially when testing new service blocks which involve various components of the BelleDIRAC system and are therefore prone to cause damages or failures the MBGS could be of great help. It was actually used for the implementation of the new service called 'Gbasf2ManagementSystem', which will be discussed in the next chapter, and has proven to be very convenient also in the sense as error tracking/logging tool.
Also for interested standard users the MBGS provides a great interface for acquiring basic knowledge about the grid system and the configuration of it. With the MBGS the user gets the unique possibility to have administrative rights over a BelleDIRAC server (as 'dirac_admin') which enables for example access to the Configuration Client and System Administrator. The configuration can be customized and again, as the MBGS is working as a standalone application, no file or system corruptions are expected. Eventually the MBGS is as flexible and developable as the large grid system and its utilization depends on the demands of its users with no restrictions set.

The MBGS is the first successful attempt to construct a fully functional miniature of the large Belle II Grid system that runs on containers. As the first of its kind it still has some weak spots and cumbersome workarounds which make it rather unstable and the installation relatively clumsy. Some recommendations and thought-provoking impulses

should be made here to ensure the stability of the system in the future and to retain its high feasibility.
One of these improvements could be done by changing the underlying base operating system of the container complex from CentOS 6 to CentOS 7 (see Section 5.2.2). This change shouldn't cause problems but probably demands to make some simple consistency checks.

Another issue is the extensive use of Docker software for containerization. Since the use of Docker requires root access it can raise various security issues. The migration of the MBGS logic to containerization software which doesn't demand root access, for example Singularity, would fix these security issues. It should be checked beforehand if these container solutions are equipped with the methods and tools required for the MBGS setup (see Section 5.2). If not, feasible detours have to be found.

Software maintenance is another point of concern for the MBGS. Currently the configuration of the MBGS is not synchronized to the large BelleDIRAC grid system and versioning conflicts of the individual components and configurations can occur. The present state of the art of the MBGS configuration setup uses predefined system state information, which could be out-of-date. On the other hand dynamic system state information is added to the MBGS at setup time from the current BelleDIRAC servers. A useful improvement would be to merge the intrinsic MBGS configuration with the actual BelleDIRAC servers to decrease the maintenance costs for the MBGS and keep it synchronized with the large system. This could involve larger changes on the MBGS configuration setup and also conceptual adjustments.
Despite all mentioned improvement opportunities and weak points in the design the MBGS is the first step to guide both standard users and developers of the Belle II community to the new world of grid computing. Depending on the demands of its users it can be expanded with numerous features and tools and be both a high level educational application and an appropriate environment for software development.

Chapter 6

Gbasf2ManagementSystem

Job submission on the BelleDIRAC Grid System is executed via the gBasf2 command line client. Having the appropriate environment this software package gives access to the data stored on the grid and its computing capabilities. The software of the gBasf2 client contains most of the logic for the submission of Belle II specific jobs and makes rough compatibility and consistency checks on the submitting jobs. If the jobs asking for submission follow the gBasf2 syntax and successfully persist these checks the DIRAC API interface 'submitJob' is called which executes the submission.
The client oriented routine from the gBasf2 command line call to the final job submission via DIRAC API has several drawbacks and contradicts in some way to the trend towards service-driven computing with APIs. The Gbasf2ManagementSystem (GMS) as independ BelleDIRAC service block aims to transfer the gBasf2 client logic to server side giving the users access to it over according GMS APIs. With this paradigm shift the BelleDIRAC servers regain control over final job submission. Whereas the gBasf2 command line client can be (whether consciously or not) manipulated by the users, the GMS APIs strictly omit code corruption through user's interventions.
In a further stage the GMS is targeted on a more environment independent grid access. Currently the gBasf2 command line client only works on SL 6/7 or CentOS 6/7 operating systems. This is a major restriction for the grid access and a barrier for the widespread application of grid computing. The GMS tends to a more system and environment independent grid access by the transfer of the logic to the BelleDIRAC servers meanwhile providing APIs which can be easily implemented in various environments.
The GMS points towards the final goal to create a BelleDIRAC Web Application which enables the user to submit jobs from the web browser, completely independent from the user's underlying operating system and computing environment. The GMS can be seen as first attempt on the road to this objective and as motivation to follow this paradigm shift towards API oriented computing software.

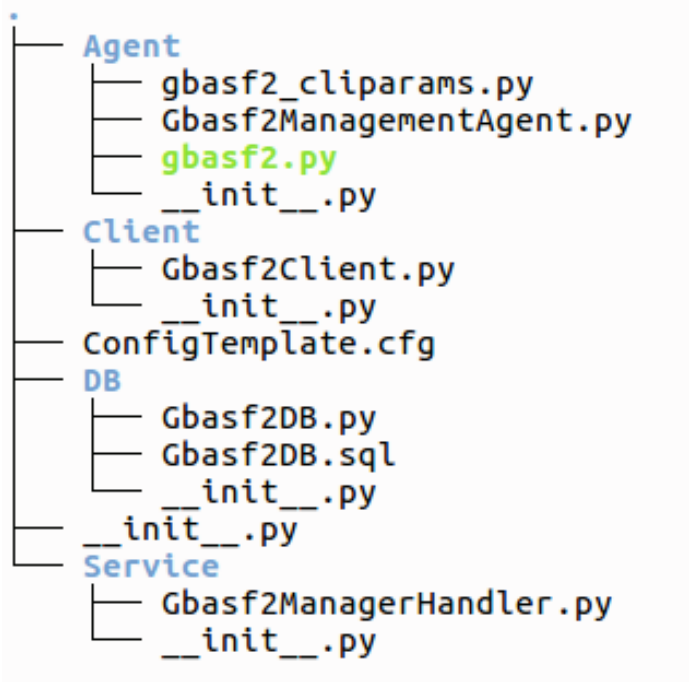


Figure 6.1: General Structure of the Gbasf2ManagementSystem.

6.1 Structure and General Workflow

The GMS as collection of various components can be roughly grouped into four parts (see tree structure in Figure 6.1) all written in Python:

- Agent
- Client
- DB
- Service

6.1.1 Agent

The Gbasf2ManagementAgent works close to the service 'Gbasf2ManagerHandler' (see Subsection 6.1.4). It checks the entries in the Gbasf2DB (the GMS database) with a polling time of 60 seconds and finally submits the jobs for the user with the help of a shifter proxy. The agent run is structured in three parts:

- `checkProjectReadiness`:
  This function checks all the projects which are registered in the database Gbasf2DB.

If the status of the project is ‘Registered‘ (and input files are properly uploaded) it will be converted to ‘Approved‘.

- `submitGbasf2Jobs`:
  The database Gbasf2DB is again checked for projects with status ‘Approved‘. The chosen projects will be submitted after the according steering files are analysed and all the contained gBasf2 options (also known by the gBasf2 command line client) are collected. Thereto the ‘gbasf2_cliparams.py‘ script (located in the same directory) is used. The actual submission is triggered by the ‘submit‘ function, defined in the ‘gbasf2.py‘ script (also located in the ‘Agent‘ directory). Both ‘gbasf2_cliparams.py‘ and ‘gbasf2.py‘ scripts are derived with slight deviations from the correspondent scripts of the gBasf2 command line client package. For example the confirmation step in job submission is automated so that the agent can submit the jobs without command line interactions.

- `finalizeSubmission`:
  This is the last check in the agent running loop. Again it checks the status of the projects in the database Gbasf2DB. If the status of the project is ‘Done‘, the associated steering file and input sandboxfiles will be removed from the server by this function. With this action the agent terminates its run.

6.1.2 Client

The Client class acts as API mediator for the client (user). It is a special wrapper for the Service Handler ‘Gbasf2ManagerHandler‘. All the APIs defined in the Service Handler can be transparently accessed by the Client. In addition the Client can define own logic and methods.
For now it only consists of the function ‘uploadFiles‘. As argument it takes the file name of the steering file which should be uploaded. When this function is called on server side with the desired steering file as input argument the steering file is sent to the server and stored under the location:
‘/opt/dirac/runit/Gbasf2Management/Gbasf2Manager‘.
If local input sandbox files are noted in the steering file they will be merged to a tarball and sent to the server where they will be unpacked afterwards. After successful file transfer the project and all its parameters are added to the Gbasf2DB database.

6.1.3 DB

The Gbasf2DB database and its corresponding script are the core of the GMS. The Gbasf2DB script contains the whole logic for the management of the database: it initializes the tables, adds new entries and modifies or deletes attributes, depending on the agent and client requests. The most important functions are:

- `addProject`: adds a new entry with the corresponding attributes to the database given the file name of the steering file.
- `getProject`: returns all attributes belonging to the queried project
- `setIDprojectParameter`: adds a parameter (attribute) for the supplied project using the project ID for the query
- `setProjectParameter`: adds a parameter for the supplied project using the project name for the query
- `getProjectParameters`: returns the parameter value(s) for the requested project
- `getProjectName`: returns the project name for the given project ID
- `addSteeringfile`: adds the steering file name (and other attributes) to the database
- `addInputsandboxfiles`: adds the file names of the input sandboxfiles (and other attributes) to the database

Besides the function definitions the Gbasf2DB script also defines the structure of the database. As backend the MySQL relational database is used. For proper database initialization the 'Gbasf2DB.sql', containing the line '`USE Gbasf2DB;`', is needed. The Gbasf2DB database consists of five independent tables and field entries:

- `gb2_Files` with fields:
  FileID, ForSteeringfile, FileName, FileType, FileBody, FileStatus, CreationDate, OwnerDN.
- `gb2_Submission` with fields:
  ProjectID, ProjectName, Status, OwnerDN, CreationDate, Release, Steeringfile, InputSandBoxFile.
- `gb2_SubmissionLog` with fields:
  ProjectID, Message, Author, MessageDate.
- `gb2_FileLog` with fields:
  FileID, Message, Author, MessageDate.
- `gb2_AdditionalParameter` with fields:
  ProjectID, ParameterName, ParameterValue, ParameterType.

It should be mentioned that each project and (steering- and input sandbox) file which is added to one of the database tables gets a unique and over the tables consistent identification number which can be used for project/file attribute requests.

6.1.4 Service

The Service (or often called Service Handler) 'Gbasf2ManagerHandler' defines the methods and functions (described in the Gbasf2DB script) which should be available for the client as APIs:

- `getProjects`
- `getProjectParameters`
- `setProjectParameter`
- `addProject`
- `addSteeringfile`
- `addInputSandboxfiles`
- `getProjectFileParameters`
- `setProjectFileParameter`
- `getinputfilepath`
- `transfer_fromClient`
- `transfer_toClient`

When the Service receives a query/request from the client, an instance of this Service Handler is created. Of special interest are the APIs `transfer_fromClient` and `transfer_toClient`. They are responsible for the secure data transfer between client and server and inherit the base Request Handler class from the DISET framework (see Section 4.2.2).

6.1.5 General Workflow

To call the GMS APIs the user needs a specific script which takes the steering file name as exclusive input argument (e.g. here 'example.py'):

```
  # !/usr/bin/env python
from DIRAC.Core.DISET.RPCClient import RPCClient
from DIRAC.Core.Base import Script
from helpers.gbasf2_cliparams import CLIParams, CLIParams_parser
import DIRAC
from BelleDIRAC.gbasf2.lib.gbasf2_params import Params
from BelleDIRAC.gbasf2.lib.job.gbasf2helper import makeSteeringFile
from DIRAC.Core.Base.Client import Client
```

```
from BelleDIRAC.Gbasf2ManagementSystem.Client.Gbasf2Client import *
Script.parseCommandLine()
simpleSubmissionService = Gbasf2Client()
simpleSubmissionService.serverURL = 'Gbasf2Management/Gbasf2Manager'
result = simpleSubmissionService.uploadFiles('example.py')
if not result['OK']:
print 'Error while calling the service', result['Message']
else:
print result['Value']
```

This Python script calls the ‘uploadFiles‘ function from the GMS Client (see Section 6.1.2) which in turn calls the ‘transfer_fromClient‘, ‘addInputSandboxfiles‘ and ‘addProject‘ APIs. The ‘transfer_fromClient‘ client request transfers the steering file and input sandbox files (if specified) to the server side. ‘addInputSandboxfiles‘ and ‘addProject‘ create new entries in the Gbasf2DB database for the acquired project and dedicated files. The assigned project status is set to ‘Registered‘. When the Gbasf2ManagementAgent enters in operation (each 60 seconds) the project status will be changed to ‘Approved‘ and the ‘submitGbasf2Jobs‘ function will be called which finally submits the job from server side calling underlying DIRAC APIs. The Gbasf2ManagementAgent checks the status of the submitted job in a loop and when the WorkloadManagement Service returns the status ‘Done‘ the function ‘finalizeSubmission‘ on the agent side is called. This function call consequently removes the steering file on server side (and if specified input sandbox files) related to the successfully finalized project. The file status of the deleted files in the ‘gb2_Files‘ table is set to ‘Deleted after successful job submission‘. With this step the GMS ends its service for the dedicated project. The whole workflow is summarized in Figure 6.1.

All the options needed for job submission such as project name, release and optional flags should be implemented in the steering file beforehand. A script for this task can be found in Appendix C.

6.2 Installation of the GMS

The step-by-step installation guide for the Gbasf2ManagementSystem can be found in the Appendix D. The presented installation procedure was tested on the MBGS (see Chapter 5) but should be applicable to all generic BelleDIRAC servers.

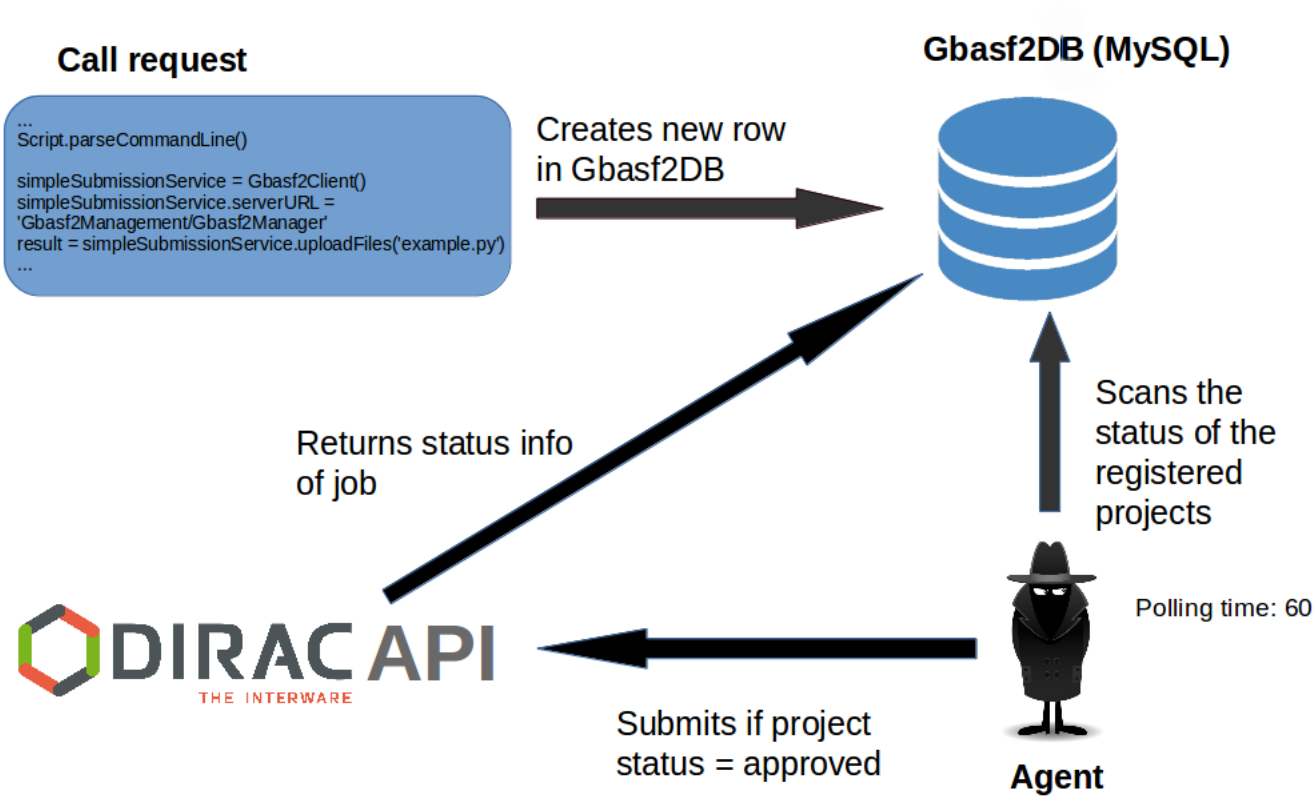


Figure 6.2: General Workflow of the Gbasf2ManagementSystem.

6.3 Improvement Opportunities and Outlook

Before concluding on the status of the GMS the great benefit of the MBGS for the development of the GMS should be mentioned. The MBGS made it possible to test the GMS on a local instance as isolated system and at the same time providing the same features and behaviour as the large scaled BelleDIRAC grid system. Therefore the MBGS can be seen as eligible software environment for the development of (Belle)DIRAC components.

The GMS successfully demonstrates the potential possibility of API based job submissions via the BelleDIRAC server. In future, several improvements could be made.
First of all the GMS interface should become more user-friendly. Currently the user has to format the steering file in a specific style and execute a script to make a GMS call request. In the future these steps should be combined and simplified and preferably some command line tool should be made available for a fast and transparent service request.
As the full project is uploaded to the BelleDIRAC server before final job submission the GMS offers the great chance to check the user files beforehand and therefore refusing the access to malicious and corrupted files. Since no such security or consistency checks have been implemented yet, the structure of the GMS database (Gbasf2DB) and the defined functions and methods have been kept largely general. In this context the whole logic of the GMS should be revised and there are a lot of details (e.g: Should the polling time of the Gbasf2ManagementAgent be stretched to reduce the GMS workload? Which attributes of the Gbasf2DB tables are useful? Which not? Should all the submission options be stored on the database for higher transparency?...) which should be discussed with the aim of making the GMS as transparent and lightweight as possible.
For a prosperous continuance and advancement of the GMS a detailed documentation of the software should be provided to enhance the readability of the code. The description of the GMS service in this chapter just gives a coarse overview of the GMS structure without proper code documentation. First steps have been made applying the python documentation generator ‘sphinx‘ but it isn’t in a mature stage yet.
In summary it can be said that the GMS development points in the right direction towards a web application for job submission and at the same time unveils problems and challenges which will be encountered when taking up this path. Hence an open and constructive discussion about the future proceedings on the road to web based job submission in the Belle II computing group should be advanced in order to offer a stable and comfortable grid access to the users.

Chapter 7

Evaluation of proton-ID performance with untagged $\Lambda \to p\pi$

The Belle II detector is an important experiment to check the results from theoretical flavor physics. This means that the identity of the charged particles, stable enough to be detected, has to be identified. At Belle II particle identification (PID) is based on likelihood ratios (see more in Section 7.1). The particle identification capabilities promise to be a considerable tool to select a purer signal sample with less background. This improves mass peak positions and resolutions for narrow resonances which can be reconstructed by charged particles only [62]. A good particle identification (PID) of protons will improve many analyses which select protons. For example analyses for the verification of the coalescence model which try to describe the fusion of protons and neutrons to form deuterium, also known as heavy hydrogen.

7.1 Introduction

This analysis evaluates the proton identification (proton-ID) performance in terms of the proton-ID efficiency and the pion fake rate by using the $\Lambda \to p\pi$ decay channel. As already mentioned, the particle identification (in this study the proton identification, proton-ID) is based on likelihood ratios. This study is strongly related to the analysis from H. Hirata reported in the Belle II Notes [63]. Since the particle identification (PID) is a promising cut-variable it is reasonable to launch further investigations into it. The purpose of this study is to better understand the differences in proton-ID between data and MC, to either correct these differences and/or to assign a systematic uncertainty. Furthermore, if the proton-ID results as reliable cutting variable, it can be presumed that the deuteron-ID variable could be applied with the same confidence on deuteron particle searches, respectively. In the search of deuteron anti-deuteron couples, which play a major role in the dark matter investigation, these analyses could be of high importance.
The $\Lambda \to p\pi$ decay provides clean proton pion samples, due to the fact that Λ - mesons have a long lifetime (mean life of $\sim 2.6 * 10^{-10}$ s [64]) and the decay products (protons and

pions) can be easily identified by using their vertex information. No proton-ID is needed for it[1]. This makes the $\Lambda \to p\pi$ decay to a convenient channel for testing the proton-ID performance (and indirectly the general particle-ID performance).
Important performance parameters are the proton-ID efficiency and the pion fake rate defined in equation 7.1 and 7.2, respectively:

$$\epsilon_p = \frac{\text{Numbers of protons with proton ID requirement}}{\text{Numbers of protons without proton ID requirement}} \tag{7.1}$$

$$f_\pi = \frac{\text{Number of } \pi \text{ with proton ID requirement}}{\text{Number of } \pi \text{ without proton ID requirement}} \tag{7.2}$$

('without' means that both proton-ID tagged and untagged protons (pions) are considered)
The variable that is in foreground in this analysis, namely the protonID, is defined as following likelihood ratio:

$$\text{protonID} = \mathcal{L}_p/(\mathcal{L}_e + \mathcal{L}_\mu + \mathcal{L}_\pi + \mathcal{L}_K + \mathcal{L}_p + \mathcal{L}_d) \text{ [65]} \tag{7.3}$$

For hadrons (π, K, p, d) the likelihoods $\mathcal{L}_{\pi/K/p/d}$ are calculated based on dE/dx information form the CDC detector (see Subsection 2.2.1), time of flight from the TOF (Time of Flight) sub-detector and the number of photons from the Aereogel Cherenkov counter (see Subsection 2.2.1). For the combined electron likelihood $\mathcal{L}_e$, in addition to the likelihood from the CDC detector and the Aereogel Cherenkov counter, information from the ECL (see Subsection 2.2.1) is used. For the muon likelihood $\mathcal{L}_\mu$ additional information from the KLM detector (see Subsection 2.2.1) is integrated in the identification algorithm.

7.2 Analysed Samples

This section gives an overview of the selected MC and data samples for this study. In Appendix E the Belle II specific file names/locations for both MC and data samples can be found.

7.2.1 Data

For this analysis the data sample from phase 2 of the Belle II detector (located in Tsukuba, Japan) at the peak of the $\Upsilon(4S)$ resonance was used. This data sample was taken in the time period between April and July 2018 and is the first data sample of the Belle II experiment. Its integrated luminosity corresponds to about 500 pb^{-1}. Due to the fact of misconfigurations of the TOP sub-detector (see Subsection 2.2.1) the integrated luminosity (for good runs) corresponds to $\sim$ 135 pb^{-1}. For more information see Appendix E.

[1]They are therefore defined as Λ - mesons with (proton-ID) untagged daughter particles (protons or pions).

7.2.2 Simulation

For this analyis the simulated data (with simulated beam background) from the Belle II MC campaign 10 with an integrated luminosity of 50 fb^{-1} was used. For the event generation the EvtGen 1.3 generator was used. This event generation includes following processes:

- $e^+e^- \rightarrow e^+e^-$
- $e^+e^- \rightarrow$ q(ark) $\bar{q}$ (antiquark) (top quark excluded)
- $e^+e^- \rightarrow B^+B^-$
- $e^+e^- \rightarrow B^0\bar{B}^0$

The full detector simulation for propagating and interacting particles the GEANT4 software package is used which also provides the simulation for the electric noise and beam induced background.

7.3 Analysis

7.3.1 Reconstruction

The Belle II Analysis Software Framework (basf2) [66] with release-02-00-02 is used as reconstruction tool. In this analysis the Λ is reconstructed (after a vertex fit has been performed) using two distinct modules:

- *ReconstructDecay*: It tries to reconstruct the decay string from the observed hits in the various sub-detectors. All different particle combinations are created and combinations that pass the specified selection criteria (see Subsection 7.3.2) are saved to a newly created particle list. For tracks originating from particle decays within the detector this procedure is problematic, because they also get extrapolated to the point of interaction. This introduces an overestimation of the energy loss due to material effects. This method was used in H. Hirata's analysis [63]. Since no cuts on the vertex position are taken, the efficiency of the *ReconstructDecay* lists is high.
- *V0FinderModule*: Given the list of observed tracks, the *V0FinderModule* takes one positive and one negative track per time, all combinations, and performs a fit. If the fit doesn't fail, a pair of tracks (positive and negative) is saved as a V0 object. The V0 objects store the full information about the reconstructed tracks but take a cut on the vertex position (no couples with the vertex smaller than 1 cm are saved, so that the energy loss correction is accounted for properly in the V0). Respectively the resolution of the V0 lists is high.

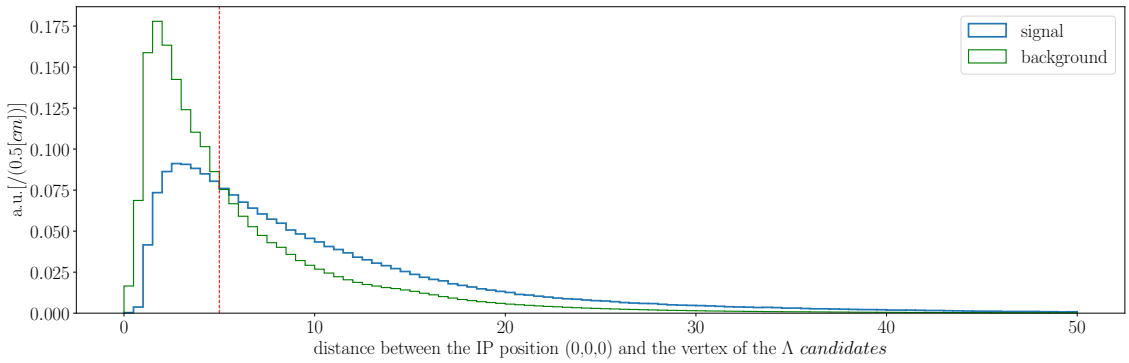


Figure 7.1: *V0ModuleFinder*. The red vertical line indicates the 'distance cut' for the sample. Here the MC simulated signal versus MC simulated background is plotted.

Both modules reconstruct a Λ from a pair of tracks with opposite charges assuming pion and proton masses. While the *V0FinderModule* saves the couples of tracks with opposite charge, keeping always as first the positive (proton) and as second the negative (pion) one, in *ReconstructDecay* the order is decided by the user.

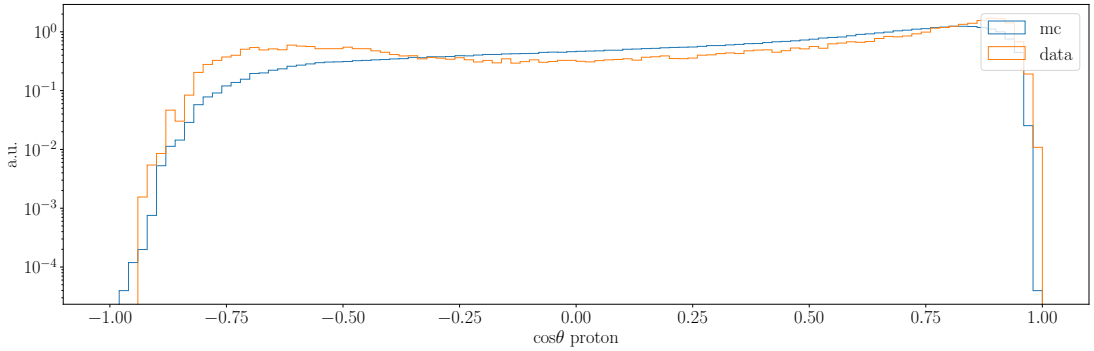
7.3.2 Λ selection criteria

The Λ selection criteria are the same as in H. Hirata's analysis [63] and are based on the kinematics and vertex information of the Λ candidates:

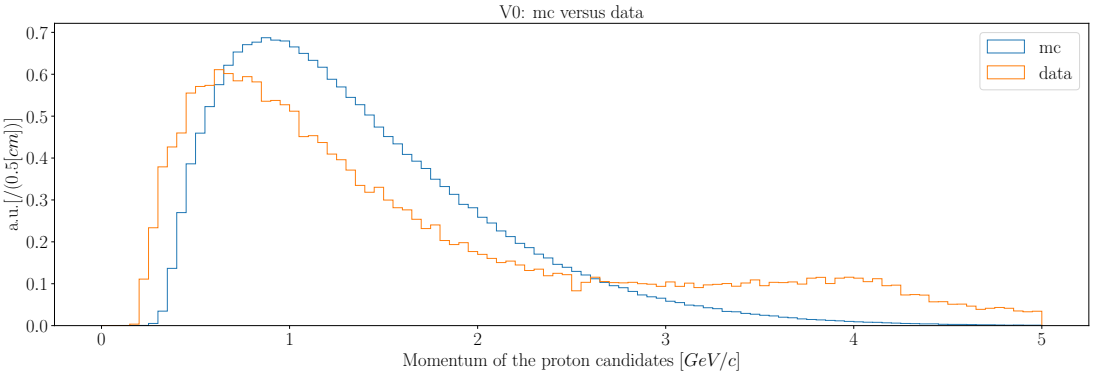
- Kinematics:
  - $M_{p\pi} < 1.2$ [GeV/c^2]
  - $P_\Lambda > 0.5$ [GeV/c]
- Vertex information:
  - $L > 0.5$ [cm] (see Figure 7.1)
  - cos$\alpha > 0.995$

Here, $M_{p\pi}$ is the mass of the Λ candidates, P_Λ the momentum of the Λ candidates in the laboratory frame. L is the distance between the IP position (0,0,0) and the vertex of the Λ candidates and α defines the angle between the decay-vertex vector and the momentum vector of the Λ candidates.

These are the generic cuts for the large MC and data samples on the grid. For the fitting procedure (see Section 7.4) the cuts are getting tighter.

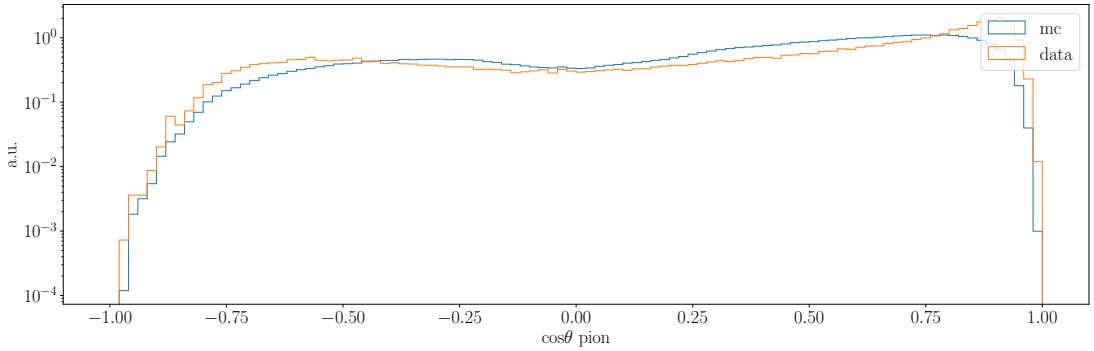


(a) cosθ distribution for MC and data

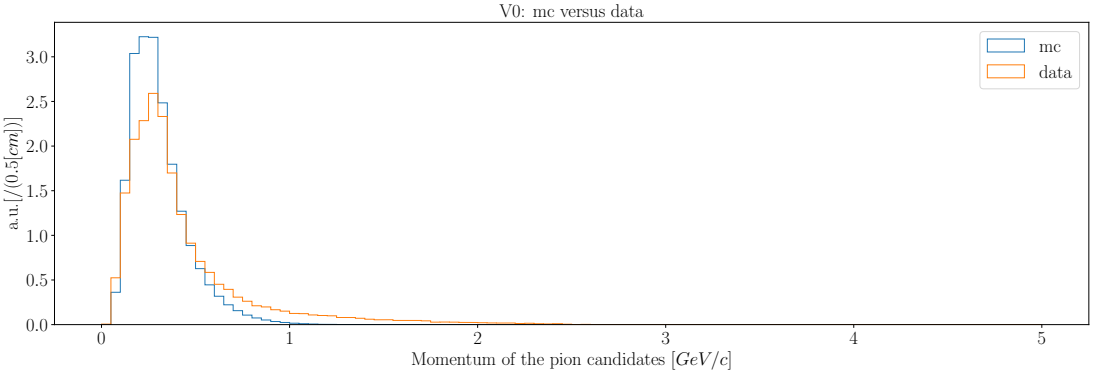


(b) Momentum distribution for MC and data

Figure 7.2: V0ModuleFinder: cosθ and momentum distributions for the proton.



(a) cosθ distributionfor MC and data



(b) Momentum distribution for MC and data.

Figure 7.3: V0ModuleFinder: cosθ and momentum distributions for the pion.

7.3.3 Binning

To analyse the dependence of the PID performance on the momentum of the daughter particles of the Λ and the cosθ of the charged track a suitable binning for these parameter spaces is needed. It allows to extract the according PID performance for each individual bin. Figure 7.2 and 7.3 show the proton and pion momenta and the cosθ distributions of the proton and pion sample with the Λ selection criteria. From consideration of the shape and range of the distributions the binning size was chosen. The binning setup is summarized in Figure 7.4 and 7.5. It covers kind of the same range as in ref [63].

Bin	Pion momentum range [GeV/c]	# events MC	# events data
1	$0.0 < P_\pi < 0.6$	794080	9014
2	$0.6 < P_\pi < 1.2$	846879	10112

(a) Pion bins.

Bin	Proton momentum range [GeV/c]	# events MC	# events data
1	$0.0 < P_p < 0.6$	54449	1046
2	$0.6 < P_p < 1.2$	300266	2887
3	$1.2 < P_p < 1.8$	254977	2461
4	$1.8 < P_p < 2.4$	140322	1466
5	$2.4 < P_p < 3.0$	62285	784

(b) Proton bins.

Figure 7.4: The bins of proton momentum and pion momentum when using the '*V0ModuleFinder*' method.

Bin	cosθ range	region
1	$-0.8660 < \cos\theta < -0.6157$	Backward region
2	$-0.6157 < \cos\theta < -0.2468$	Barrel region (TOP)
3	$-0.2468 < \cos\theta < 0.1222$	
4	$0.1222 < \cos\theta < 0.4911$	
5	$0.4911 < \cos\theta < 0.67555$	
6	$0.67555 < \cos\theta < 0.8600$	
7	$0.8600 < \cos\theta < 0.9563$	Forward region (ARICH)

Figure 7.5: The bins of cosθ for both the proton and pion sample.

Bin	# events MC	# events data
1	26013	852
2	98661	1536
3	130527	1137
4	178095	1452
5	133522	1258
6	192146	2383
7	86939	1915

(a) cosθ proton bins.

Bin	# events MC	# events data
1	29338	892
2	121328	1683
3	114053	1149
4	192430	1604
5	152647	1476
6	180394	2144
7	55651	1569

(b) cosθ pion bins.

Figure 7.6: V0ModuleFinder: Number of events in the MC and data sample for the cosθ proton (a) and cosθ pion bins (b).

Bin	# events MC	# events data
1	967636	51755
2	2533125	77242
3	3243997	68757
4	4274786	111175
5	3155776	100533
6	4264865	167294
7	2169245	97676

(a) cosθ proton bins.

Bin	# events MC	# events data
1	805806	46892
2	2660022	90399
3	5066167	100736
4	4717178	137473
5	3023863	112818
6	3347669	138076
7	1114509	53665

(b) cosθ pion bins.

Figure 7.7: ReconstructDecay: Number of events in the MC and data sample for cosθ proton (a) and cosθ pion bins(b).

7.4 Fitting Procedure

To calculate the proton-ID efficiency and pion fake rate the $p\pi$ invariant mass (in the range of 1.1 to 1.13 GeV/c^2) with and without the proton-ID requirements is fitted to extract the Λ signal yield. In this analysis the RooFit (version 6.18) toolkit for data modelling was used to perform unbinned maximum likelihood fits. To get the actual number of background and/or signal events from the fit the extended likelihood formalism is used [67]. Since two different reconstruction modules (‘*V0ModuleFinder*‘ and ‘*ReconstructDecay*‘) are used also the fitting procedure varies respectively:

ReconstructDecay - Fitting

In the fit, the signal probability density function (p.d.f.) is sum of a Gaussian and Crystal Ball function [2]. For both functions common means are used and the ratio of the Gaussian sigma and the Crystal Ball sigma is fixed by the result for MC fit. As background p.d.f. the 2nd Chebychev polynomial is used. This is similar as in reference [63].
Since peak position and resolution of the $p\pi$ invariant mass distribution depends on the cosθ and momentum the parameters of the signal p.d.f have to be determined for each bin in the cosθ and the momentum space. For this reason the following procedure is used for each bin to figure out the proton-ID efficiency and the pion fake rate:

1. The first step is to determine the background p.d.f with proton-ID requirements on the assumed proton (pion). For MC data the true information of all events is available and can be called by the truth matching module. This allows to select the true background.

2. To determine the signal yield with proton-ID requirements on protons (pions) fulfilled the full MC data is fitted using as background p.d.f. the parameters determined in step 1. The signal p.d.f. parameters are float. The ratio of the Gaussian sigma and the Crystal Ball sigma is determined (and later used for real data fit). Figure 7.8 shows an example for this fitting step.

3. To determine the signal yield without proton-ID requirements on any particles the signal parameters are fixed as determined in step 2 from the signal fit with proton-ID requirements on the protons. The background p.d.f. parameters are taken from 1. Figure 7.9 shows an example for this fitting step.

4. The proton-ID efficiency and the pion fake rate are computed by n/m, where n is the signal yield with proton-ID requirement on proton (pion) and m is the yield of signal without proton-ID requirement on any daughter particle.

5. For real data analysis the signal yield with proton-ID requirements on protons (pions) is determined by fitting to it the signal p.d.f and the background p.d.f. where all parameters are float, just the ratio of the Gaussian sigma and the Crystal Ball sigma is fixed by the one obtained in step 2 (and using common means for the signal p.d.f. function as described above) for the MC fit.

6. The signal yield without proton-ID requirements on any particles is determined in the same way as for MC data (see step 3) using fixed parameters for the signal p.d.f. the ones from the fit in step 5. The background p.d.f. parameters are floating.

7. Again, same as in step 4 the proton-ID efficiency and the pion fake rate are computed by n/m, where n is the signal yield with proton-ID requirement on proton (pion) and

[2]The definition of the Crystal Ball function can be found in reference [68], Appendix E.

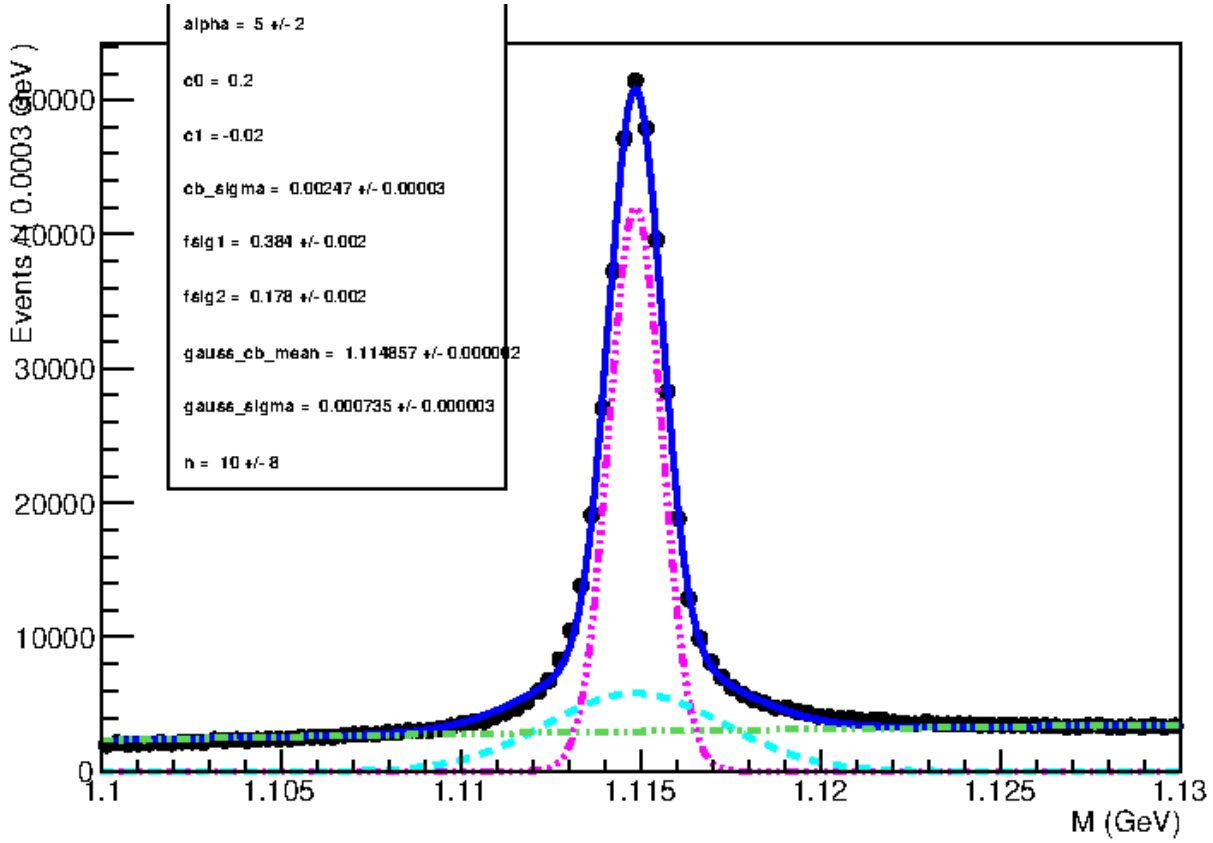


Figure 7.8: '*Reconstructdecay*': Example of fitted MC sample for the the proton bin $0.1222 < \cos_{\text{Proton}}\theta < 0.4911$ with proton-ID requirements on the proton. Pink: Crystal Ball, bright blue: Gauss, green: Chebychev polynomial, dark blue: overall fit.

m is yield of signal without proton-ID requirement on any daughter particle. The results from step 5 and 6 are used for the yields.

V0ModuleFinder - Fitting

For data with events reconstructed with the '*V0ModuleFinder*' the fitting procedure is the same as in reference [63]. In contrast to the fitting steps for MC data obtained by the'*ReconstructDecay*' module 7.4, in step 2 the background p.d.f. parameters are floating (step 1 is omitted). That means that the $p\pi$ invariant mass plot is fitted with signal p.d.f. and background p.d.f., where all parameters are float.
The reason why the fitting procedure for data with events reconstructed with '*Reconstruct-Decay*' module was changed from the reference [63] was the instability of the fits since the number of floating parameters is large.

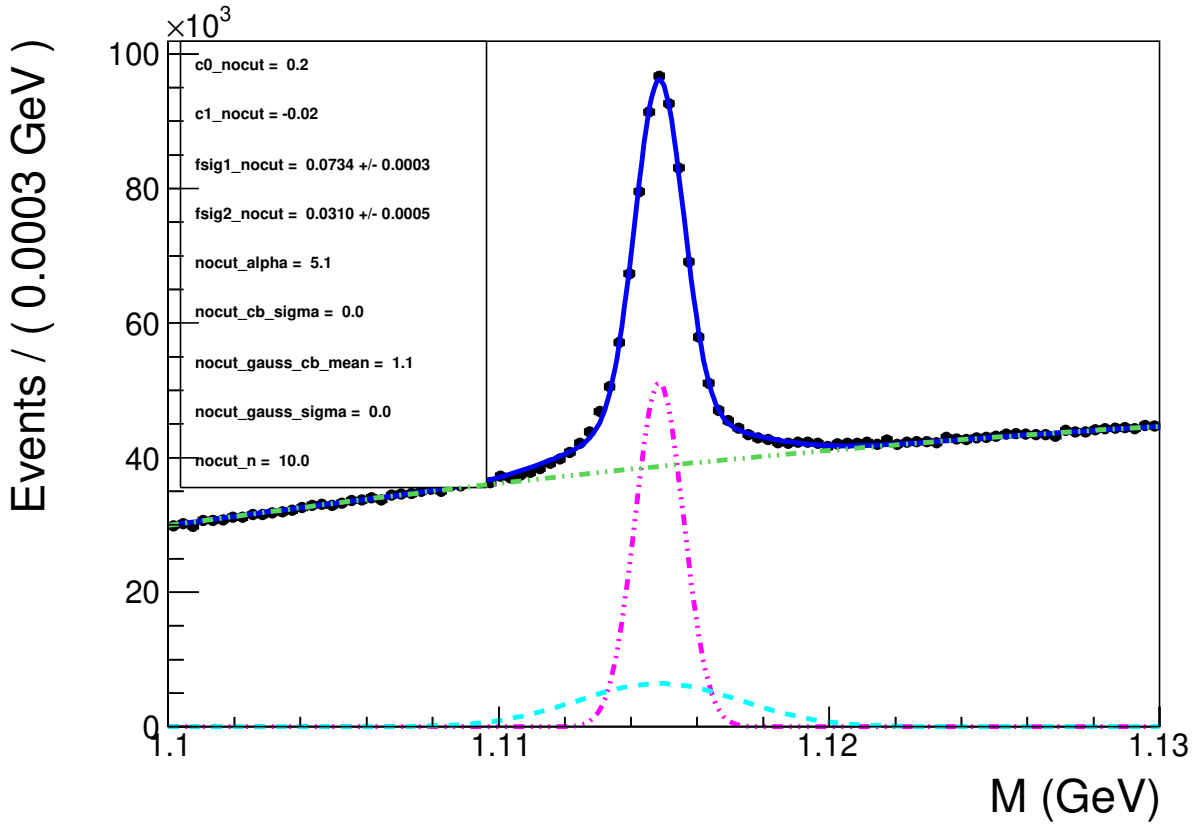


Figure 7.9: '*Reconstructdecay*': Example of fitted MC sample for the the proton bin $0.1222 < \cos_{\text{Proton}}\theta < 0.4911$ without any proton-ID requirements. Pink: Crystal Ball, bright blue: Gauss, green: Chebychev polynomial, dark blue: overall fit.

7.4.1 Statistical Uncertainty

The statistical error $\delta\epsilon$ is calculated as follows [63]:

$$\delta\epsilon = \frac{1}{m}\left[\sqrt{\left|n\left(1-\frac{n}{m}\right)\right|} \bigoplus \sqrt{\left|((\delta n)^2 - n)\left(1 - \frac{(\delta n)^2 - n}{(\delta m)^2 - m}\right)\right|}\right] \tag{7.4}$$

where δn is the fitting error on n and δm is the fitting error on m, and $\bigoplus$ is the quadratic sum.

7.5 Results

7.5.1 V0ModuleFinder

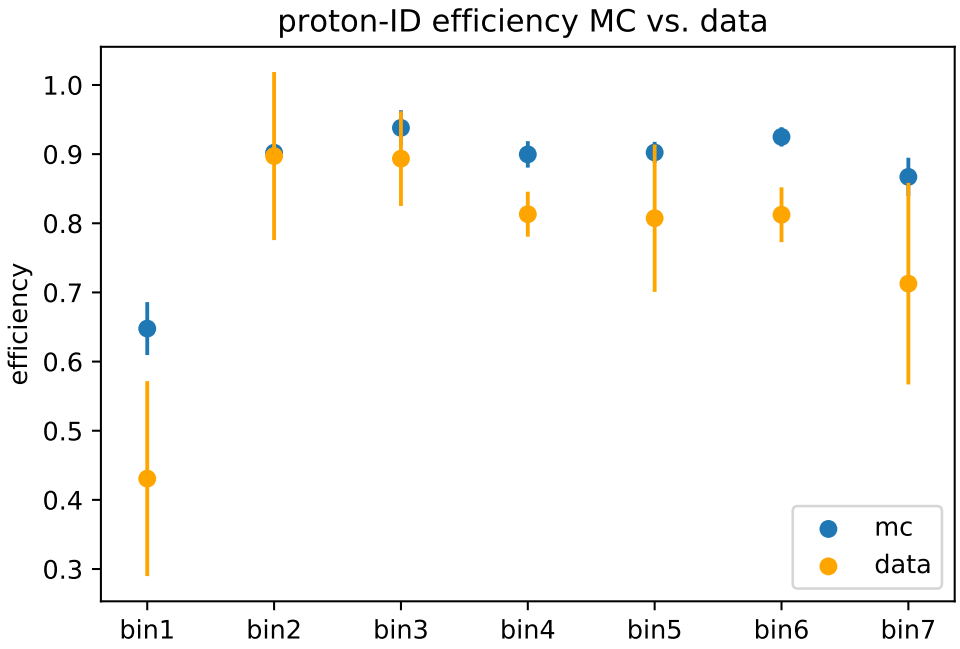
Figure 7.10 and 7.11 show the results for the efficiency of the $p\pi$ invariant mass for the different cosθ and momenta regions when using the '*V0ModuleFinder*'. The efficiency and fake rate is determined by fitting the $p\pi$ invariant mass after a track assumed proton (pion) required proton-ID, as described in the fitting procedure (see Section 7.4). Here the proton-ID requirements are $\mathcal{L}_p > 0.6$. The average proton-ID efficiency in the cosθ range is about 87 % for MC and 77 % for data. For the pion fake rate in the cosθ range an average value of about 0.28 % for MC and 6.68 % for data is obtained.

7.5.2 ReconstructDecay

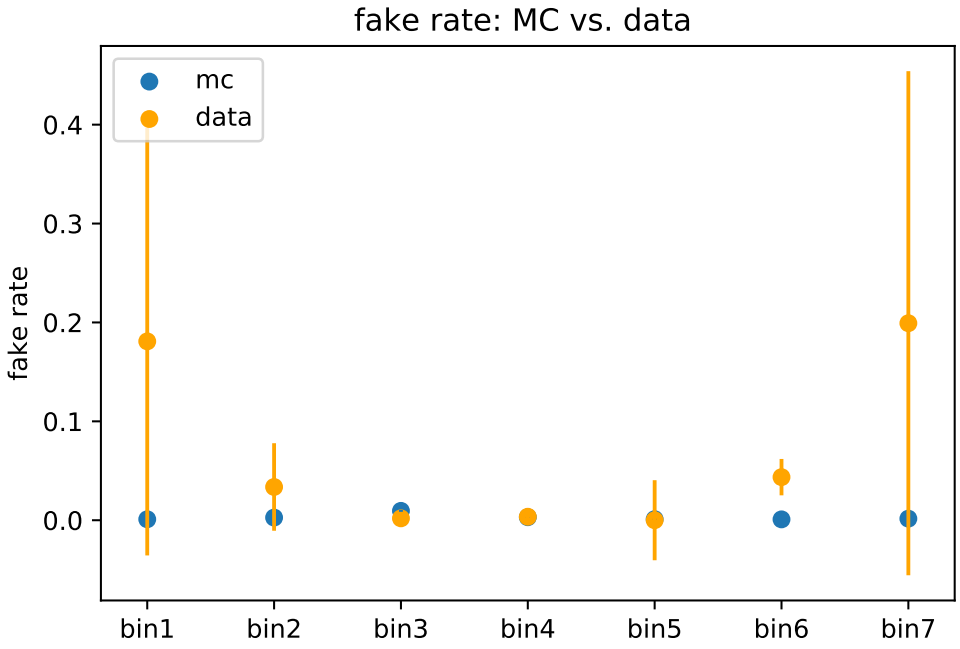
Figure 7.12 shows the results from the fitting of the $p\pi$ invariant mass for the different cosθ regions when using the '*ReconstructDecay*' module. Same as when using the V0ModuleFinder for reconstruction (see Subsection 7.5.1) the proton-ID requirements are set to $\mathcal{L}_p > 0.6$. From the fit result the average proton-ID efficiency in the cosθ range is about 82 % for MC and 89 % for data. For the pion fake rate in the cosθ range an average value of about 0.32 % for MC and 0.98 % for data is obtained.

7.6 Conclusion and Outlook

Comparing with the result from the analysis of H. Hirata [63], where a total proton-ID efficiency on data (with '*ReconstructDecay*') of 85.1 ± 0.7 % (stat. uncertainty) is obtained, in this analysis an efficiency of 89 ± 2.7 % (stat. uncertainty) is reached. This seems to be a respectable improvement. Nevertheless there are discrepancies and open questions that rise from this analysis:
From the results it turns out that the differences between MC and data are not negligible, especially for the efficiency values in the proton momentum bins (see Figure 7.11). The differences seem to diverge when going towards high proton momenta. Therefore further investigations on the sub-detectors are needed since the overall proton-ID is calculated by

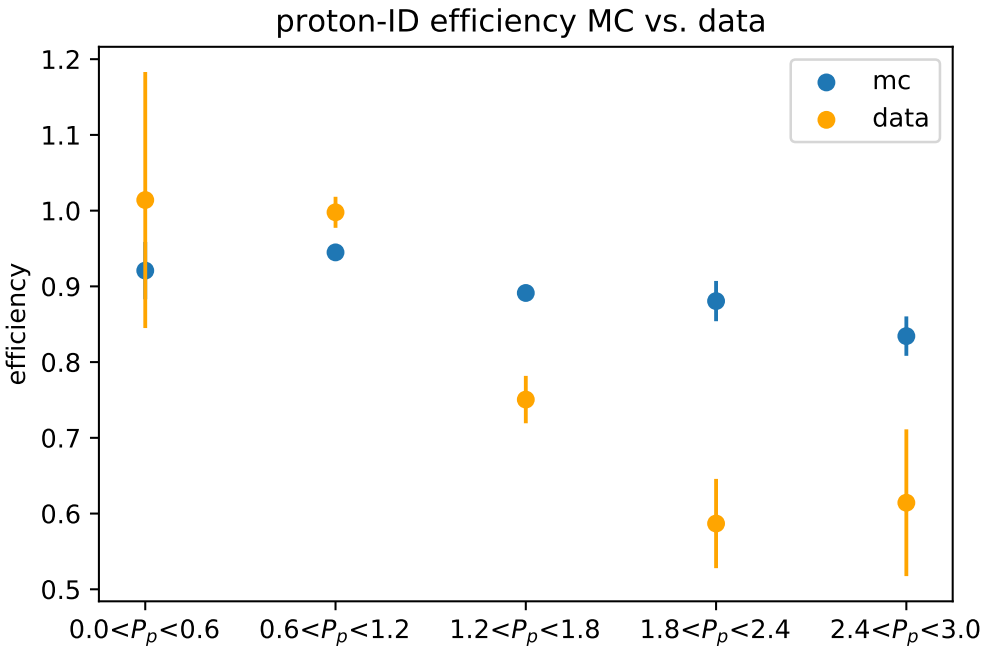


(a) The proton-ID efficiency as a function of cosθ for MC and data.

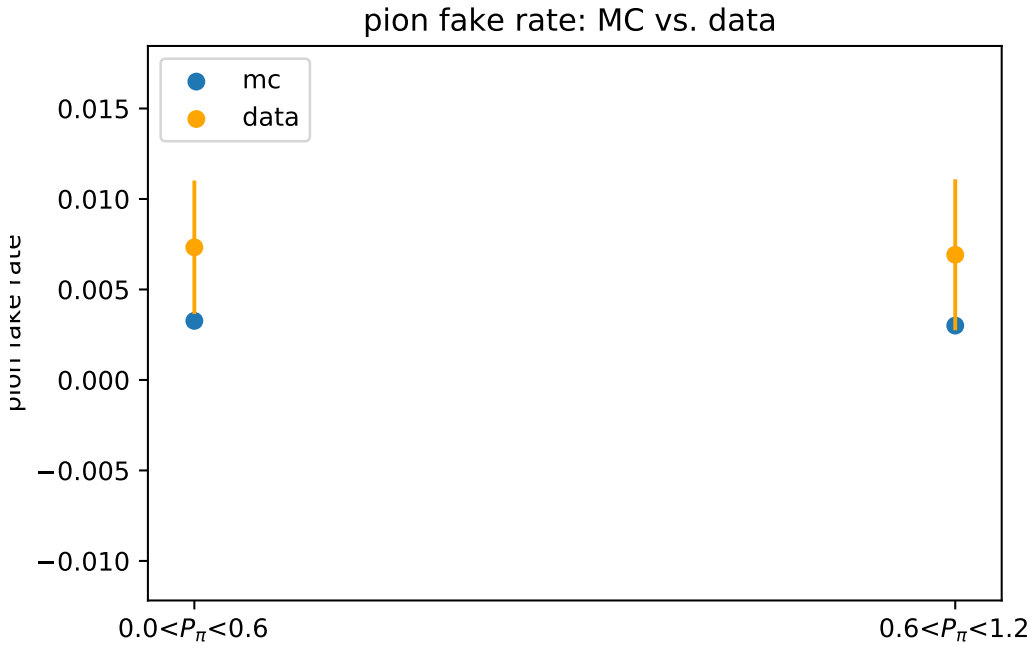


(b) Pion fake rate as a function of cosθ for MC and data

Figure 7.10: V0ModuleFinder. The blue dots show the MC values, the orange the data values, respectively. The bars stand for the statistical uncertainty.

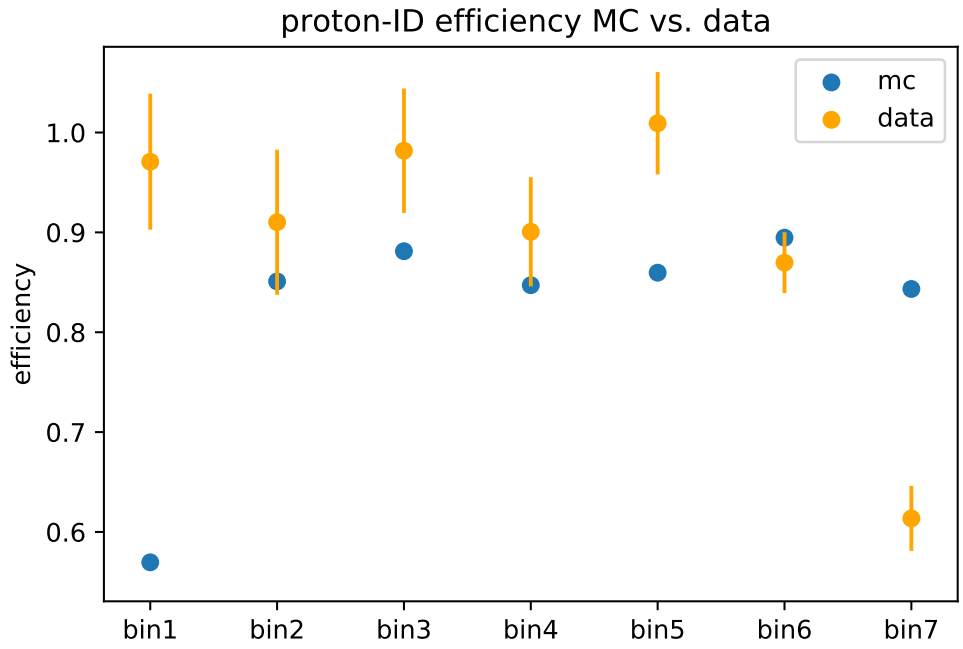


(a) The proton-ID efficiency as a function of the proton momenta for MC and data.

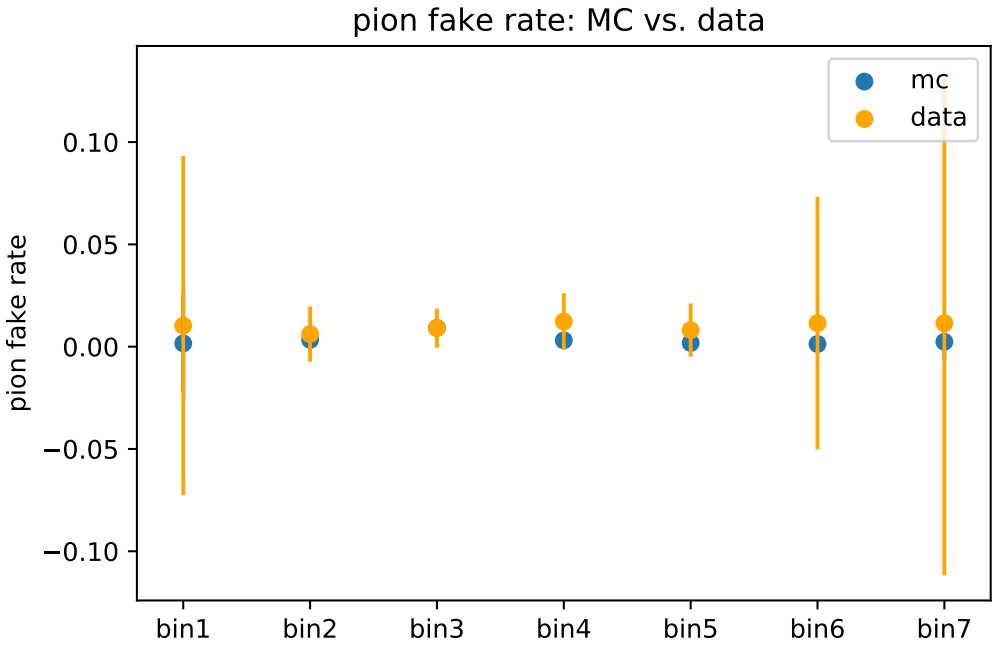


(b) The pion fake rate as a function of the pion momenta for MC and data.

Figure 7.11: V0ModuleFinder. The blue dots show the MC values, the orange the data values, respectively. The bars stand for the statistical uncertainty.



(a) The proton-ID efficiency as a function of cosθ for MC and data.



(b) Pion fake rate as a function of cosθ for MC and data.

Figure 7.12: ReconstructDecay. The blue dots show the MC values, the orange the data values, respectively. The bars stand for the statistical uncertainty.

multiplication of the likelihoods from the various sub-detectors. Particular consideration should be given to the CDC detector (for data) in the forward and backward regions where tracking observes problems due cross talk, which is not modelled in the MC.
Further investigations should also be carried out to explain the differences in the results when using '*V0ModuleFinder*' and '*ReconstructDecay´* module, respectively. The only difference should come from systematic uncertainties as the number of selected background events are different but Figure 7.11 and 7.12 show large discrepancies between the usage of the two modules.
Besides the statistical uncertainties (see Subsection 7.4.1) it should be discussed which systematic uncertainties could affect this analysis. First of all the process of data taking is sometimes affected by shut-downs of various sub-detectors. For the used data sample there were some problems with the TOP sub-detector. As the TOP sub-detector is one of the PID detectors, one possible improvement of the analysis is to repeat the study for runs with and without TOP to see if there are differences.
Systematic uncertainties from fitting arise from the choice of the shapes for signal and background. For this analysis two Gaussians instead of a Gaussian plus a Crystal Ball function for the signal p.d.f. could be used.
Also the differences in the background levels between MC and data has to be included in the discussion concerning the systematic uncertainties. The MC underestimates the background, which has impact on several aspects.
Since simulation of MC events does not perfectly reflect the real physics phenomena and the slight misconfiguration of various sub-detectors that can occur for long running times, differences in the distributions of physical variables between MC and data should be further evaluated. Especially the reconstructed proton the momenta distributions (for '*V0ModuleFinder*') for MC and data show remarkable deviations. Therefore the MC events should be reweighed (and the simulation itself has to be improved) so that the MC distributions converge to the data distributions. This check and reweighing procedure should be done for diverse variables and integrated in the new calculation of the efficiencies and fake rates for MC. It is expected that the discrepancies between MC and data are reduced.
Further on the efficiencies and fake rates for MC and data events reconstructed with the '*ReconstructDecay*' module in the proton and pion bins should be evaluated, as it hasn't been done in this thesis for lack of time. Moreover, as in this study the proton-ID efficiencies and pion fake rates are considered in the $\cos\theta$ and momentum space separately, it has to be checked if there are correlations between the momenta and the $\cos\theta$ for protons and/or pions. If so, this would make the interpretation of the results more difficult.

Appendix A

Step-by-step installation guide for the MBGS

1. **Setting up the docker grid network:**
   Run on terminal:

   ```
   docker network create --subnet 172.99.0.0/16 --driver bridge
   gridnetwork
   ```

   This creates the subnet 172.99.0.0/16 which will be used as grid network in this setup. It connects the BelleDIRAC Server with the working node.

2. **Installation and configuration of the BelleDIRAC Server**

   The file `belledirac_server.zip` has to be downloaded from `https://confluence.desy.de/display/BI/Minimal+BelleDIRAC+Grid+System`. After unzipping the downloaded directory the usercert.pem and userkey.pem have to be copied into the unzipped directory (./dirac_server). Those two files should have the permissions of 640 and the userkey.pem has to be password secured!
   The next step is to set some environmental variables in the Dockerfile (you should find it in the previously unzipped working directory). Follow the explanations on the Dockerfile. Save and exit the Dockerfile. Then run following commands (in the downloaded directory):

   ```
   docker build -t imagename .
   ```

   (Insert for imagename your preferred imagename for the docker image, e.g. dirac_image.

Don't miss the dot at the end of the command out!). This will take several minutes.

```
docker run -it -p 8443:8443 -p 52022:22 --network gridnetwork
--mount
type=bind,source=cvmfs,target=/cvmfs --ip 172.99.0.10
--add-host
'slave:172.99.0.5' --name containername -h servername
imagename /bin/bash
```

(Insert for imagename the previously chosen imagename, for containername a preferred name for the container, e.g. dirac_server,for servername the name set in the Dockerfile and for cvmfs the path to your local cvmfs directory)

Now you should be inside the docker container (on your working directory /home-/dirac) which will be used as BelleDIRAC Server. Inside the container you should run following commands:

```
export DIRAC_ADMINUSERDN=‘openssl x509 -noout -subject
-in /home/dirac/.globus/usercert.pem|cut -d '=' -f2,3,4,5,6‘

export DIRAC_HOSTDN=‘openssl x509 -noout -subject -in
/opt/dirac/etc/grid-security/hostcert.pem|cut -d '='
-f2,3,4,5‘

sudo fetch-crl

envsubst<"/home/dirac/gbasf2KEK/config.cfg">"/home/dirac
/gbasf2KEK/myconfig.cfg"

envsubst <"/home/dirac/install.cfg"> "/home/dirac/myinstall.cfg"

envsubst<"/home/dirac/gbasf2KEK/defaults-Belle-DOCKER.cfg">
"/home/dirac/gbasf2KEK/mydefaults-Belle-DOCKER.cfg"
```

Then type:

```
./install_site.sh myinstall.cfg
```

This will take several minutes. At the end a table of the installed components will be displayed. Then use the following commands to shunt the process into the background:

press the keys: 'Control+z' followed by 'bg'

The next step is to install and configure the BelleDIRAC specific components and the gbasf2 Client:
The gbasf2 Client installation is necessary to get the necessary configuration informations from the main server (https://dirac.cc.kek.jp:8443) For this you need to run following commands:

```
cd gbasf2KEK
wget -N http://belle2.kek.jp/~dirac/dirac-install.py
python dirac-install.py -V Belle-KEK
source bashrc && dirac-proxy-init -x
```

(For the last command you will be prompted to insert the password of your userkey.pem)

```
dirac-configure defaults-Belle-KEK.cfg
cat getResources.txt | dirac-configuration-cli
```

Now the additional information is saved under dataChanges.cfg
To switch to the gbasf2 server environment the current gbasf2 Client environment has to be removed. For that run:

```
rm /home/dirac/gbasf2KEK/etc/dirac.cfg
source /home/dirac/unset
```

Now start with the gbasf2 server installation:

```
source bashrc && dirac-proxy-init -x
dirac-configure mydefaults-Belle-DOCKER.cfg
source BelleDIRAC/gbasf2/tools/dev_setup.sh
```

(With the last command you source as dirac_admin)

```
dirac-proxy-init -g dirac_admin
```

(Proxy initialization as dirac_admin)

Now you will upload all the configuration informations saved in the files /home-/dirac/gbasf2KEK/myconfig.cfg and /home/dirac/gbasf2KEK/dataChanges.cfg to serverside. For that you have to enter the System Configuration Console:

```
dirac-configuration-cli
```

Now you are on the interactive shell. You type:

```
mergeFromFile dataChanges.cfg
mergeFromFile myconfig.cfg
writeToServer
```

(Be aware of the order of the merge: dataChanges.cfg has to be merged before myconfig.cfg!)
Then exit by typing: `'exit'`
As next step you have to add your working node (which you will create later on) to the site mask. Just type:

```
dirac-admin-allow-site your_site 'some comment'
```

(Replace your_site with the name of your DIRAC_SITE declared in the Dockerfile. If you encounter some error message like 'unauthorized query....', type 'dirac-proxy-init -g dirac_admin')
Then:

```
dirac-populate-component-db
```

(If you encounter some error message like 'unauthorized query....', type 'dirac-proxy-init -g dirac_admin')
Now you are almost done. Your BelleDIRAC Server is set up properly.

3. **Installation and configuration of the working node**
(Make sure that /cvmfs is mounted!)

Now it's time to install and configure the working node. For that you have to build the appropriate docker image. Open a new terminal and go to your working node directory:

```
cd your_downloaded_directory/slave
```

Then run inside the directory your_downloaded_directory/slave the command:

```
docker build -t slave .
```

To run the docker container for the working node:

```
docker run -dit --mount type=bind,source=cvmfs,target=/cvmfs
--network
gridnetwork --ip 172.99.0.5 --add-host
'servername:172.99.0.10' --name
slave -p 2204:22 slave
```

(Replace servername with the SERVERNAME you declared in your BelleDIRAC Server Dockerfile and cvmfs with the path of your cvmfs directory)

and

```
docker exec -it slave /bin/bash
```

Now you should be inside the docker container for the working node.
The grid system needs ssh connection between the BelleDIRAC Server and the working node(s). In your working node container the openssh-server is already installed, you just have to start it:

```
service sshd start
```

Set a password for your 'root' user for later ssh connection:

```
passwd root
```

Now your working node is set up.

4. **Final preparations for first job submission**
Go to your interactive shell of your BelleDIRAC Server. Set also here a password for your 'dirac' user:

```
sudo passwd dirac
```

and start the openssh-server:

```
sudo service sshd start
```

For the job submission the BelleDIRAC Server has to connect to the working node via ssh. You want to automate this connection without using any password. For that you have to type (on server side):

```
ssh-keygen -t rsa
```

Follow the steps and leave the input field empty for no passphrase.Then go to the directory where your ssh key is saved (default: /home/dirac/.ssh). Copy the public key (id_rsa.pub) to the working node side and paste it into the file /root/.ssh/auhorized_keys. Save it and make sure that the permissions of /root/.ssh/authorized_keys are set to 640:

```
chmod 640 /root/.ssh/authorized_keys
```

Now everything is set up and you can try to connect to the working node via ssh from server side by typing (on server side):

```
ssh root@slave
```

(If you want to disconnect, just type 'exit' and you will be back on server side) As last step before running our first gbasf2 job we have to initialize and upload the proxies for the user called 'belle' and the belle-pilot (on the server side):

```
dirac-proxy-init -g belle -P -M -U
```

Now we are prepared to submit the first job. For that we want to access the BelleDIRAC Server as gbasf2 client using ssh. Open a new terminal and type:

```
ssh -p 52022 dirac@localhost
```

You should be inside the BelleDIRAC Server. Then change directory to gbasf2KEK:

```
cd gbasf2KEK
```

Now you have to setup the gbasf2 Client environment:

```
source BelleDIRAC/gbasf2/tools/setup.sh

gb2_proxy_init -g belle
```

Finally the first job submission: On the directory /home/dirac/gbasf2KEK you will find the steering file 'example.py'. To submit your first gbasf2 job you just have to type:

```
gbasf2 /home/dirac/gbasf2KEK/example.py -p MyFirstProject
-s release-02-00-02
```

For more information about gbasf2 type:

```
gbasf2 --help
```

5. **How to restart the Minimal BelleDIRAC Grid System?**

By using the docker commit command the previous steps and the server and working node configurations can be saved. Open a terminal and type:

```
docker commit slave slave

docker commit dirac_server dirac_image
```

After this the two containers can be stopped and removed:

```
docker stop slave dirac_server && docker rm slave dirac_server
```

To rerun the two containers you just have to type following commands in two separated shells:

```
docker run -it -p 8443:8443 -p 52022:22 --network gridnetwork
--mount type=bind,source=cvmfs,target=/cvmfs --ip 172.99.0.10
--add-host
'slave:172.99.0.5' --name containername -h servername
imagename /bin/bash
```

and

```
docker run -dit --mount type=bind,source=cvmfs,target=/cvmfs
--network
gridnetwork --ip 172.99.0.5 --add-host
'servername:172.99.0.10' --name
slave -p 2204:22 slave
```

Now you are in both shells inside the docker container. On the server side container you have to reset the environmental variables and to restart the BelleDIRAC and openssh-server:

```
export DIRAC_ADMINUSERDN=‘openssl x509 -noout -subject
-in /home/ dirac/.globus/usercert.pem|cut -d '=' -f2,3,4,5,6‘

export DIRAC_HOSTDN=‘openssl x509 -noout -subject
-in /opt/dirac/etc/grid-security/hostcert.pem|cut -d '='
-f2,3,4,5‘

sudo service sshd start

cd /opt/dirac/sbin

sudo ./start
```

Then use the following commands to shunt the process into the background:
press the keys:`'Control + z'` and type `bg` Also on the working node container the

opsnssh-server has to be restarted. Type there:

```
service sshd start
```

To check the conection between the BelleDIRAC server and woring node, type on server side:

```
ssh root@slave
```

(If you want to disconnect, just type `'exit'` and you will be back on server side) The server is on. The container acting as working node is already rightly configured and no changes have to be made. To submit jobs, access the BelleDIRAC Server as gbasf2 client using ssh. Open a new terminal and type:

```
ssh -p 52022 dirac@localhost
```

You should be inside the BelleDIRAC Server. Then change directory to gbasf2KEK:

```
cd gbasf2KEK
```

Now you have to setup the gbasf2 Client environment:

```
source BelleDIRAC/gbasf2/tools/setup.sh

gb2_proxy_init -g belle
```

For job submission you type for instance:

```
gbasf2 /home/dirac/gbasf2KEK/example.py -p MySecondProject -s
release-02-00-02
```

Appendix B

Installation instructions of independent Docker container for job submission (with GMS from chapter 6)

For a containter in control of job submission the docker image of the working node container can be reused:

```
docker run -it --mount type=bind,source=cvmfs,target=/cvmfs
--network gridnetwork --ip 172.99.0.12 --add-host
'servername:172.99.0.10' --name
submitter slave /bin/bash
```

(Replace servername with the SERVERNAME you declared in your BelleDIRAC Server Dockerfile and cvmfs with the path of your cvmfs directory)
Now you are inside the 'submitter' container. Firstly the user certificate and key has to be uploaded to the container and stored in the directory '`/root/.globus`'. (If it doesn't exist, create it). Be sure that the userkey.pem has the '*rw*' permissions only for the owner and no permission to the others.
As next step the gBasf2 Client has to be downloaded and installed [1]:

[1]For further information see: `https://confluence.desy.de/display/BI/Computing+GBasf2`

```
mkdir gbasf2KEK && cd gbasf2KEK
wget -N http://belle2.kek.jp/~dirac/dirac-install.py
python dirac-install.py -V Belle-KEK
source bashrc && dirac-proxy-init -x
dirac-configure defaults-Belle-KEK.cfg
```

This gBasf2 Client installation already includes a predefined configuration ofthe official BelleDIRAC servers. That means this docker container is now qualified to submit ordinary grid jobs to the large grid system. For job submission on the MBGS the preset configuration has to be changed. Therefore the content of the file '/home/dirac/gbasf2KEK/etc/dirac.cfg' has to be replaced by:

```
LocalInstallation
{
  GenericMcPath = /belle/MC/generic/
  UserPath = /belle/user/
  VirtualOrganization = belle
  RawPath = /belle/raw
  SignalMcPath = /belle/MC/signal/
  Setup = $DIRAC_SETUP
  DataPath = /belle/data/
  PythonVersion = 27
  Project = Belle
  InstallType = client
  SkipCAChecks = True
  Release = v4r6
  ConfigurationServer = dips://$SERVERNAME:9135/Configuration/Server
  LcgVer = v13r0
  SkipCADownload = No
}
DIRAC
{
  Setup = $DIRAC_SETUP
  Extensions = Belle
  Configuration
  {
    Servers = dips://$SERVERNAME:9135/Configuration/Server
  }
  VirtualOrganization = belle
  Security
  {
    UseServerCertificate = no
```

```
    SkipCAChecks = yes
  }
}
```

(Replace $DIRAC_SETUP and $SERVERNAME by the corresponding environmental variables set in the Dockerfile).

Once the above installation is done, you just need to execute the following two commands for the usage of the gBasf2 Client on the MBGS:

```
source /home/dirac/gbasf2KEK/BelleDIRAC/gbasf2/tools/setup
gb2_proxy_init -g belle
```

To use the GMS Service as Client on the newly created container the GMS package has to be downloaded from `https://stash.desy.de/users/jenegger/repos/gms/browse` and uploaded to the dedicated docker container. To copy the local GMS repository to the container, type:

```
docker cp Gbasf2ManagementSystem submitter:/home/dirac
/gbasf2KEK/BelleDIRAC
```

Now the GMS Client can be used. To submit jobs on the MBGS with this client the user simply executes the script presented in subsection 6.1.5 (supposed the specified steeringfile exists).

Appendix C

Creation of GMS conform steeringfile

For successful job submission via GMS Client all options have to be implemented in the steeringfile before actual client call. To achieve this without inconvenient manual input the following script can be used:

```
#!/usr/bin/env python

from helpers.gbasf2_cliparams import CLIParams, CLIParams_parser
import DIRAC

from BelleDIRAC.gbasf2.lib.gbasf2_params import Params
from BelleDIRAC.gbasf2.lib.job.gbasf2helper import makeSteeringFile

cliParams = CLIParams()
parser = CLIParams_parser(cliParams)
parser.fill()

#print cliParams.getProject()
#print cliParams.getSteeringFile()
#print cliParams.getSetuprel()

pm = Params()
pm.cliParams = cliParams

makeSteeringFile(pm)
print 'this is the steeringfile:', makeSteeringFile(pm)['Value']
```

To transform the steeringfile into a GMS suitable format convert the above script to an executable (`chmod +x steering_script.py`) and type:

```
./steering_script your_steeringfile -p project_name -s release `+ additional
flags`
```

The created steeringfile gets the name ‘steeringfile-project_name.py‘. This has to be inserted in the submission script of section 6.1.5. With the execution of the submission script the job gets finally submitted via the GMS Client.

Appendix D

Installation guide for the Gbasf2ManagementSystem

1. **GMS download:**

   The GMS can be downloaded from `https://stash.desy.de/users/jenegger/repos/gms/browse` As next step the GMS package has to be copied to the following directories:

   - /opt/dirac/pro/BelleDIRAC
   - $HOME/gbasf2KEK/BelleDIRAC (possibly MBGS specific)

   Note:

   - check the owner of the GMS on both locations. It has to be 'dirac' (and not 'users'). To change file owner type:

     ```
     sudo chown -R dirac.dirac /opt/dirac/pro/BelleDIRAC
     ```
     and
     ```
     sudo chown -R dirac.dirac $HOME/gbasf2KEK/BelleDIRAC
     ```
   - check the file permission of the GMS package on both locations. It has to be '775'.

2. **GMS configuration on BelleDIRAC server:**

   The implementation of the GMS is done via the DIRAC Configuration Client. This service can only be accessed as (Belle)DIRAC administrator. Therefore the appropriate environment for the so called 'dirac_admin' has to be set up:

```
source $HOME/gbasf2KEK/BelleDIRAC/gbasf2/tools/dev_setup.sh
```

And proxy initialization as dirac_admin:

```
dirac-proxy-init -g dirac_admin
```

To access the DIRAC Configuration Client type:

```
dirac-admin-sysadmin -H $HOSTNAME
```

Now you are inside the DIRAC Configuration Client. Foremost the GMS as overall system has to be added to the BelleDIRAC configuration:

```
add system Gbasf2Management Development
```

Followed by the installation of the GMS components:

```
install service Gbasf2Management Gbasf2Manager
install db Gbasf2DB
install agent Gbasf2Management Gbasf2ManagementAgent
```

Then exit the DIRAC Configuration Client by typing ‘`exit`‘. Now the GMS is successfully installed.

As final step the shifter proxy, which is used to specify the user certificate to manage the submission over the Gbasf2ManagementAgent component, has to be defined. By default (Belle)DIRAC components run with the host certificate, but sometimes (e.g. for the Gbasf2ManagementAgent job submission) a valid user certificate is needed to access grid services (e.g. the file catalogue).
The shifter proxy for the GMS has to be defined in the configuration file ‘/opt/dirac/etc/Belle.cfg‘ under:

```
Operations
{
        DIRAC_SETUP
        {
```

```
        Shifter
        {
        Gbasf2Manager
        {
                User = DIRAC_ADMINUSERNAME
                Group = belle
        }
        }
        }
}
```

For DIRAC_SETUP and DIRAC_ADMINUSERNAME insert name of the (Belle)DIRAC setup and the username of the administrator of the (Belle)DIRAC server respectively. If the MBGS from chapter 5 is used, both environment variables can be obtained from the dedicated Dockerfile.

3. **Final job submission with GMS:**

Jobs can be submitted through the GMS using the BelleDIRAC server of the MBGS by connecting to it via '`ssh -p 52022 dirac@localhost` ' and by user proxy initialization (see section 5.3) or by creating an independent docker container for job submission. How this can be achieved is explained in detail in the appendix B. After setting up the BelleDIRAC user environment (`source _path_to_your_gbasf2_installation_/BelleDIRAC/gbasf2/tools/setup`) and successful user proxy initialization (`gb2_proxy_init -g belle`) the GMS Service can be called straightforwardly by executing the script presented in subsection 6.1.5 (supposed the specified steeringfile exists and was successfully tested as local Basf2 job beforehand).
The log of the GMS service (on server side) can be traced by typing:

```
tail -f /opt/dirac/runit/Gbasf2Management/Gbasf2Manager/
log/current
```

same wise the log of the GMS agent:

```
tail -f /opt/dirac/runit/Gbasf2Management/Gbasf2ManagementAgent/
log/current
```

The status of the submitted job can of course be traced over the BelleDIRAC Web

App from the web browser by calling up `https://localhost:8443/` as it is also done when submitting jobs over the gBasf2 Client.

Appendix E

$\Lambda \to p\pi$ decay: analyzed samples

E.1 Data

For this analysis data from `phase 2`, `exp3` is used with LFN:

- `/belle/Data/release-03-00-03/DB00000528/proc8b/prod00006967/e0003/4S/r<RUN>/mdst/sub00/`
- `/belle/Data/release-03-00-03/DB00000528/proc8b/prod00006968/e0003/4S/r<RUN>/mdst/sub00/`

The global tag is '`GT528`'. The integrated luminosity of this sample is ~ 135 pb^{-1}[1].

E.2 Simulation

For this analysis the official MC10 samples with `phase2` geometry and beam background are used [2].

[1] For further information see: `https://confluence.desy.de/display/BI/Experiment+3`

[2] For further information see: `https://confluence.desy.de/display/BI/Data+Production+MC10#DataProductionMC10-PhaseIIY(4S)generic(50fb-1)`

Bibliography

[1] Abe K. et al. Observation of large cp violation in the neutral b meson system. *Physical Review Letters*, 87(9):091802, 2001.

[2] First collisions at belle 2. `https://www.symmetrymagazine.org/article/first-collisions-at-belle-ii`. "[Online; accessed 7-May-2019]".

[3] Dirac - the interware. `http://diracgrid.org/`. "[Online; accessed 7-May-2019]".

[4] Computing gbasf2. `https://confluence.desy.de/display/BI/Computing+GBasf2`. "[Online; accessed 7-May-2019]".

[5] I.C. Baianu et al. Fundamentals of physicsand nuclear physics. `https://web.archive.org/web/20121002214053/http://novelresearchinstitute.org/library/PhysNuclphys196p.pdf`. "[Online; accessed 9-May-2019]".

[6] Bogdan Povh, Klaus Rith, Christoph Scholz, Frank Zetsche, and Werner Rodejohann. *Teilchen und Kerne: eine Einführung in die physikalischen Konzepte*. Springer-Verlag, 2013.

[7] Jean-Louis Basdevant, James Rich, and Michael Spiro. *Fundamentals in nuclear physics: From nuclear structure to cosmology*. Springer Science & Business Media, 2005.

[8] Short history of particle accelerators. `https://cas.web.cern.ch/sites/cas.web.cern.ch/files/lectures/zakopane-2006/tazzari-history.pdf`. "[Online; accessed 9-May-2019]".

[9] Wilfried Buchmüller and Christoph Lüdeling. Field theory and standard model. *arXiv preprint hep-ph/0609174*, 2006.

[10] `https://stfc.ukri.org/research/particle-physics-and-particle-astrophysics/peter-higgs-a-truly-british-scientist/why-is-the-higgs-discovery-so-significant/`. "[Online; accessed 9-May-2019]".

[11] The large hadron collider-research 2. `https://rampages.us/almahmouda/2014/07/10/the-large-hadron-collider-research-2/`. "[Online; accessed 2-July-2019]".

[12] David Griffiths. *Introduction to elementary particles*. John Wiley & Sons, 2008.

[13] Tanabashi et al. Review of particle physics. *Physical Review D*, 98(3):030001, 2018.

[14] Luigi Di Lella and Carlo Rubbia. The discovery of the w and z particles. In *60 Years of CERN Experiments and Discoveries*, pages 137–163. World Scientific, 2015.

[15] `https://home.cern/science/physics/standard-model`. "[Online; accessed 12-May-2019]".

[16] Glashow-weinberg-salam model: An example of electroweak symmetry breaking. `http://guava.physics.uiuc.edu/~nigel/courses/569/Essays_Fall2007/files/xianhao_xin.pdf`. "[Online; accessed 13-May-2019]".

[17] The brout-englert-higgs mechanism. `https://home.cern/science/physics/higgs-boson`. "[Online; accessed 13-May-2019]".

[18] A L Read. The discovery and measurements of a higgs boson. *Physica Scripta*, T158:014009, dec 2013.

[19] The mystery of cp violation. `http://web.mit.edu/physics/news/physicsatmit/physicsatmit_06_sciollafeature.pdf`. "[Online; accessed 13-May-2019]".

[20] Chien-Shiung Wu, Ernest Ambler, RW Hayward, DD Hoppes, and Ralph Percy Hudson. Experimental test of parity conservation in beta decay. *Physical review*, 105(4):1413, 1957.

[21] Stuart Raby and Richard Slansky. Neutrino masses: How to add them to the standard model. *Los Alamos Sci.*, 25:64–71, 1997.

[22] Dark matter. `https://home.cern/science/physics/dark-matter`. "[Online; accessed 14-May-2019]".

[23] Guido Altarelli. The higgs and the excessive success of the standard model. *arXiv preprint arXiv:1407.2122*, 2014.

[24] Kek history. `https://www.kek.jp/en/About/History/`. "[Online; accessed 14-May-2019]".

[25] Abe K. et al. Belle ii technical design report. *arXiv preprint arXiv:1011.0352*, 2010.

[26] Kazunori Akai, Kazuro Furukawa, Haruyo Koiso, et al. Superkekb collider. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 907:188–199, 2018.

[27] Belle ii rolls in. `https://cerncourier.com/belle-ii-rolls-in/`. "[Online; accessed 14-May-2019]".

[28] Pioneering babar and belle joint analysis. `https://www.kek.jp/en/NewsRoom/Release/20150818160000/`. "[Online; accessed 22-May-2019]".

[29] Electrons and positrons collide for the first time in the superkekb accelerator. `https://www.kek.jp/en/newsroom/2018/04/26/0700/`. "[Online; accessed 22-May-2019]".

[30] Gagan B Mohanty. Belle ii silicon vertex detector. *arXiv preprint arXiv:1511.06197*, 2015.

[31] Detector. `https://www.belle2.org/research/detector/`. "[Online; accessed 15-May-2019]".

[32] AJ Bevan, Boštjan Golob, Th Mannel, S Prell, BD Yabsley, H Aihara, F Anulli, N Arnaud, T Aushev, M Beneke, et al. The physics of the b factories. *The European Physical Journal C*, 74(11):3026, 2014.

[33] Ian Foster. What is the grid? a three point checklist. *GRID today*, 1:32–36, 01 2002.

[34] Nikolaos P Preve. *Grid computing: towards a global interconnected infrastructure*. Springer Science & Business Media, 2011.

[35] Efficient testing of simulations on the belle ii grid infrastructure. `http://ipnp.cz/kodys/works/uceni/en/DP_Ludacka_Radek_2013.pdf`.

[36] Belle II Grid Infrastructure. *E cient testing of simulations on the*. PhD thesis, Institute of Particle and Nuclear Physics, Faculty of Mathematics and ..., 2013.

[37] Steven Wallace. "lambda networking". 05 2019.

[38] Gregor von Laszewski and Kaizar Amin. Grid middleware. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.462.3579&rep=rep1&type=pdf`. "[Online; accessed 20-May-2019]".

[39] Voms: Virtual organization membership service. `http://toolkit.globus.org/grid_software/security/voms.php`. "[Online; accessed 21-May-2019]".

[40] Thomas Kuhr et al. First production with the belle ii distributed computing system. In *Journal of Physics: Conference Series*, volume 513, page 032050. IOP Publishing, 2014.

[41] Hideki Miyake, Rafal Grzymkowski, Radek Ludacka, and Malachi Schram. Belle ii production system. In *Journal of Physics: Conference Series*, volume 664, page 052028. IOP Publishing, 2015.

[42] Rene Brun and Fons Rademakers. Root—an object oriented data analysis framework. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 389(1-2):81–86, 1997.

[43] Architecture overview. https://dirac.readthedocs.io/en/latest/DeveloperGuide/Overview/index.html. "[Online; accessed 23-May-2019]".

[44] Andrei Tsaregorodtsev, M Bargiotti, N Brook, A Casajus Ramo, G Castellani, Ph Charpentier, C Cioffi, J Closier, R Graciani Diaz, G Kuznetsov, et al. Dirac: a community grid solution. In *Journal of Physics: Conference Series*, volume 119, page 062048. IOP Publishing, 2008.

[45] Federico Carminati, Latchezar Betev, and Alina Grigoras. *Grid and Cloud Computing: Concepts and Practical Applications*, volume 192. Ios Press, 2016.

[46] D C Vanderster, F Brochu, G Cowan, U Egede, J Elmsheuser, B Gaidoz, K Harrison, H C Lee, D Liko, A Maier, J T Mościcki, A Muraru, K Pajchel, W Reece, B Samset, M Slater, A Soroko, C L Tan, and M Williams. Ganga: User-friendly grid job submission and management tool for LHC and beyond. *Journal of Physics: Conference Series*, 219(7):072022, apr 2010.

[47] Vincent Garonne, A Yu Tsaregorodtsev, and I Stokes-Rees. Dirac: Workload management system. Technical report, CERN, 2004.

[48] Lfc. https://wiki.scc.kit.edu/gridkaschool/index.php/LFC. "[Online; accessed 31-May-2019]".

[49] Andrew C Smith and Andrei Tsaregorodtsev. Dirac: reliable data management for lhcb. In *Journal of Physics: Conference Series*, volume 119, page 062045. IOP Publishing, 2008.

[50] Massimo Lamanna. Arda experience in collaborating with the lhc experiments. Technical report, 2006.

[51] The amga metadata catalog introduction and hands-on exercises. https://indico.cern.ch/event/431185/contributions/1910378/attachments/933086/1321707/nsantos_AMGA_metadata_catalog_hands_on.pdf. "[Online; accessed 31-May-2019]".

[52] Computing cli2gbasf2. https://confluence.desy.de/display/BI/Computing+CLI2GBASF2. "[Online; accessed 03-June-2019]".

[53] Belle ii python interface. https://b2-master.belle2.org/software/development/sphinx/framework/doc/index-03-framework.html#modules-and-paths. "[Online; accessed 04-June-2019]".

[54] Introduction to the analysis package. https://confluence.desy.de/display/BI/B2-StarterKit+Jun-2019+Agenda?preview=%2F122984628%2F122984636%2FIntroductionToTheAnalysisPackage.pdf. "[Online; accessed 04-June-2019]".

[55] Computing gbasf2testingsystem. https://confluence.desy.de/display/BI/Computing+Gbasf2TestingSystem. "[Online; accessed 05-June-2019]".

[56] Docker. https://www.docker.com/. "[Online; accessed 05-June-2019]".

[57] What is a container? https://www.docker.com/resources/what-container. "[Online; accessed 22-June-2019]".

[58] Use bind mounts. https://docs.docker.com/storage/bind-mounts/. "[Online; accessed 21-June-2019]".

[59] Download centos linux iso images. https://wiki.centos.org/Download. "[Online; accessed 22-June-2019]".

[60] Cernvm-fs documentation-release 2.6.0. https://buildmedia.readthedocs.org/media/pdf/cvmfs/latest/cvmfs.pdf. "[Online; accessed 22-June-2019]".

[61] Openbsd manual page server. https://man.openbsd.org/ssh. "[Online; accessed 22-June-2019]".

[62] T. Matsuda M. Niiyama Y. Kato, H. Hirata and M. Takizawa. Comprehensive study of peak positions and resolutions for various narrow hadrons. 2018.

[63] H.Hirata. *Evaluation of proton-ID performance with untagged $\rightarrow p$ sample in Prod5 data*. BELLE2-NOTE-PH-2018-036, 2018.

[64] Lambda. http://pdg.lbl.gov/2019/listings/rpp2019-list-lambda.pdf. "[Online; accessed 2-July-2019]".

[65] release-03-02-02, variables. https://software.belle2.org/sphinx/release-03-02-02/analysis/doc/Variables.html?highlight=particleid#variable-particleID. "[Online; accessed 2-July-2019]".

[66] T. Kuhr, C. Pulvermacher, M. Ritter, T. Hauth, and N. Braun. The belle ii core software. *Computing and Software for Big Science*, 2018.

[67] Kyle Cranmer, Akira Shibata, Wouter Verkerke, Lorenzo Moneta, and George Lewis. Histfactory: A tool for creating statistical models for use with roofit and roostats. Technical report, 2012.

[68] Tomasz Skwarnicki. A study of the radiative cascade transitions between the upsilon-prime and upsilon resonances. Technical report, 1986.

Acknowledgements

At first I want to thank Prof. Dr. Thomas Kuhr for giving me the opportunity to write my master thesis about these interesting topics and giving me the freedom to work independently on the various subjects.
Special thanks goes to Hideki Miyake who patiently advised me in many computing topics and gave me great tips for solution approaches in regard to my projects. Without his help this work would not have been possible.
Furthermore I want to thank Dr. Thomas Lueck, Dr. Nikolai Hartmann and Kilian Lieret for proofreading my thesis, also giving me new impetus and really detailed feedback.
In addition, great thanks goes to all group members under the direction of Prof. Dr. Thomas Kuhr who contributed to the comfortable and respectful working atmosphere which I truly appreciated.

Eigenständigkeitserklärung

Hiermit erkläre ich, die vorliegende Arbeit mit dem Titel

Grid Computing and Particle Identification Performance Study with Untagged $\Lambda \to p^+\pi^-$

Grid Computing und Messung der Teilchenidentifikationseffizienzen am Zerfall $\Lambda \to p^+\pi^-$

selbstständig verfasst zu haben und keine anderen als die in der Arbeit angegebenen Quellen oder Hilfsmittel benutzt zu haben.

Tobias Jenegger

München, den 25. Juli 2019