LUND UNIVERSITY
Faculty of Science

Monte Carlo production of proton-proton collision events using the ATLAS@Home framework

Dimitrios Sidiropoulos – Kontos

Thesis submitted for the degree of Master of Science

Project duration: 12 months

Supervised by Oxana Smirnova

Department of Physics
Division of Particle Physics
January 2018

Abstract

The ATLAS@Home project is a volunteer computing project, part of the larger LHC@Home project, aimed at using the computational power of personal computers from volunteers around the globe, who are interested in helping with the particle physics research taking place at the ATLAS experiment at LHC. Up this point it runs only ATLAS detector simulation tasks. This thesis explores the possibility of having the full generation of large Monte Carlo samples performed on this platform after laying the theoretical physics groundwork and introducing the concepts and elements used by this platform. Such a task has not been attempted so far but with the computational resources increasingly limited for such tasks, due to the large amounts of data the LHC produces nowadays, the need for additional non-dedicated resources, such as those offered by ATLAS@Home, is increasing. The study explores that possibility using reference Monte Carlo samples and tests whether their generation can be reliably reproduced on the virtual machine used by the project and other environments such as the Grid and a local cluster. It also tests whether the generation and derivation of the simulation data in an, appropriate and readable by commonly used analysis software, file format can occur in a single Grid task submission (as ATLAS@Home should operate as a Grid site as well from where the tasks are picked up and where the output files return upon generation), without storing and transferring heavy, intermediate generation files during this process. The study succeeds in meeting those conditions for the samples tested and proceeds to the succesful submission of a number of such tasks to a test project, running parallel to ATLAS@Home for such new kinds of submissions, with plans, in the near future, to have such tasks running on the main application.

Abbreviations and Acronyms Guide

ALICE: A Large Ion Collider Experiment
AMD: Advanced Micro Devices
ARC CE: Advanced Resource Connector Computer Element
ATLAS: A Toroidal LHC ApparatuS
BEH mechanism: Brout - Englert - Higgs mechanism
BOINC: Berkeley Open Infrastructure for Network Computing
CERN: Conseil Européen pour la Recherche Nucléaire (European Organization for Nuclear Research
CernVM: CERN Virtual Machine
CMS: Compact Muon Solenoid
CPU: Central Processing Unit
CVMFS/CernVM-FS: CERN Virtual Machine File System
DAOD: Derived Analysis Object Data
DID: Data IDentifier
FCal: Forward Calorimeter
HEC: Hadronic End Cap
HTTP: Hypertext Transfer Protocol
IP: Internet Protocol
IRC safe: Infra-Red Collinear safe
ISO: International Organization for Standardization
LAr calorimeter: Liquid Argon calorimeter
LHC: Large Hadron Collider
LHCb: Large Hadron Collider beauty
LIGO: Laser Interferometer Gravitational-wave Observatory
MC: Monte Carlo
PanDA: Production and Distributed Analysis
PIN: Personal Identification Number
QCD: Quantum ChromoDynamics
RAM: Random-Access Memory
SM: Standard Model
TileCal: Tile Calorimeter
UA2: Underground Area 2
UC Berkeley: University of California Berkeley
VDI: VirtualDisk Image
VM: Virtual Machine
WIMP: Weakly Interacting Massive Particle

Acknowledgements

This work has been the first major project of that scale and importance I have taken on. Many times during the last year the pressure of responsibility towards the people expecting a positive result in this project, the consistent failures, the lack of knowledge that had to be acquired so fast and the high demands I had for myself really took their toll on me psychologically and how I proceeded with this work. The moments I felt I would not be able to finish the work you are currently reading have been greater than the number of pages in this report. It is not without the input of some special people that I have reached the end of this year-long journey successfully.

First of all I want to thank Oxana Smirnova, my supervisor, for her unlimited patience with my inconsistent working times, for always being available to answer my constant and many times naive questions, my insecurities about each step of the way as the project proceeded and for being a voice of reason whenever I felt overwhelmed.

I owe a lot to David Cameron, of the ATLAS@Home project, for guiding me through the whole way to the final successful submissions on the platform, for his detailed answers to all my, many times also simplistic and basic, questions, for his valuable input every time I felt like I was hitting a wall of non-progress.

A big thank you to Caterina Doglioni for providing me with the reference Monte Carlo generation scripts I used and for helping me understand them and work with them, given I had no previous experience with such generations before.

To professor Charikleia Petridou, who was the first person to introduce me to the world of particle physics, I owe a a lot and a big thank you as well.

Yosse, Simon, Zhiying, my fellow master's student colleagues who I worked in the office with, thank you for putting up with me in the good and the bad days, for the brainstorming whenever help was needed and for the casual conversations that kept me going through the day whenever I needed to let pressure off.

To Konstantinos, my best friend, thank you for reminding me of my goals and re-igniting my fire whenever I felt defeated and unmotivated.

To Ms. Maria Zervaki, thank you for being the kind of physicist I strive to be.

Finally I owe the biggest debt of gratitude and I want to dedicate this Thesis to my mother, Sofia. If it wasn't for her and the enormous sacrifices she made so that I am able to chase my dream and study at Lund University in the particle physics program, this project would never come to fruition. I will never be able to repay what you have done for me mother. I hope this work will at least vindicate that to some degree.

Contents

Chapter 1

Introduction

The field of particle physics has been in the forefront of the expansion of scientific knowledge for years, mainly thanks to the contributions of the Large Hadron Collider (LHC) at CERN. Vast amounts of data are produced within it, from collisions that aim to simulate the conditions that were present during the early stages of the universe. The first run of the LHC provided significant amounts of data and led, most prominently, to the discovery of the Higgs boson, validating theoretical hypotheses on how elementary particles acquire mass. The second run has seen the LHC producing data at even higher rates to probe into even higher energies and investigate possible new phenomena outside the bounds of the Standard Model. During this run the LHC accelerator performs at energy scales of the order of 13 TeV center-of-mass energy per collision with the data rates constantly increasing (the total data flow from the 4 major experiments during Run 2 was expected to be about 25 GB/s and for ATLAS in particular a production rate of data was reported at the range of 800 MB/s - 1 GB/s). All this produced raw data cannot be probed as it is. It needs to be processed, tracks need to be reconstructed, actual physical objects that were produced have to be retrieved from a chaotic number of daughter particles and particle jets. This processing needs computational power, a lot of computational power.

In order to validate a theory or reject it, the comparison of actual data produced by collisions in the LHC with a large amount of simulated events, known as Monte Carlo (MC) samples, also needs to be done. Especially for research groups that look into possibly detecting particles that would validate theories beyond the Standard Model, such as candidate Dark Matter particles, the need for large amounts of Monte Carlo simulations, much larger than the actual data produced, is great. That is because larger background simulations provide a better accuracy of the interpretation of actual data when they are fitted to that data. When one wants to probe into particles that are assumed to interact very weakly with normal matter, this kind of accuracy is essential.

Unfortunately the computational facilities available to LHC are struggling increasingly to cope with processing both the large amounts of data produced and the even larger amounts of Monte Carlo simulations needed to probe into this data for possible new discoveries. Thus the need to explore alternative computational resources becomes greater.

This is what this project inspects. Focused on research performed at the ATLAS experiment at CERN, the goal is to inspect whether the volunteer computing platform ATLAS@Home, that brings together the processing power of personal computers contributed by volunteers around the globe, can provide a suitable alternative of non-dedicated resources to process large amounts of Monte Carlo simulations. Such simulations are needed

especially by researchers who try to confirm the existence of Dark Matter candidate particles, such as Weakly Interacting Massive Particles (WIMPs) or Axions. In order to achieve this, different Monte Carlo samples must be created using various ATLAS specific event generators, such as PYTHIA. Thereupon, the created Monte Carlo samples will be submitted as work units to the ATLAS@Home framework, where their generation will be attempted. The LHC Computing Grid will also be used to process the reference samples. A successful generation on the Grid will imply that the same process can be replicated on the ATLAS@Home platform.

If the project comes to fruition and the ATLAS@Home framework proves capable of processing the created Monte Carlo samples,the availability of computer resources for Monte Carlo production at ATLAS will significantly increase. A significant avenue will be provided to ATLAS researchers to produce much needed background simulations, leading to more accurate interpretation of ATLAS results and the possible expansion of the knowledge in the field through new discoveries, possibly beyond the Standard Model.

Chapter 2 will lay the theoretical foundations of modern particle physics in the form of the Standard Model, as well as theories beyond that in the form of Dark Matter physics. It will be followed by a general description of the Large Hadron Collider at CERN with a focus on the ATLAS experiment and the ATLAS detector. It will also briefly present the concept of Monte Carlo generation and a brief description of jets in the context of particle physics.

In Chapter 3, the concept of virtualization will be first presented in order to then proceed to talk about voulunteer computing in general, the essential BOINC application for volunteer computing, and finally the LHC@Home project and ATLAS@Home application in particular.

Chapter 4 is dedicated to the description of the MC generation scripts provided to work with for this project and in particular the generation of reference samples on two different processing facilities, a local one, Iridium and the LHC computing Grid as well.

In Chapter 5, the setup of the virtual machine used by ATLAS@Home will be presented alongside the concept of contextualization that is relevant to this work.

Chapter 6 will present the processing of the MC generation scripts used previously on this, specially contextualized for ATLAS@Home, virtual machine, and a comparison with the reference samples previously produced will be made. It will also discuss the submission of tasks to the ATLAS@Home project.

Finally, in Chapter 7 a summary of the work that has been done and the progress on the researched matter will be given, while the potential future outlook of such event generations on ATLAS@Home will also be presented.

Chapter 2

The Standard Model, the LHC and the ATLAS experiment

2.1 The Standard Model of High Energy Physics - and beyond

The Standard Model (SM) of particle physics ([1] - [4]) is, essentially, the widely accepted, and so far experimentally confirmed, theoretical framework that describes the fundamental interactions of the elementary particles it presumes all matter to consist of.

It encompasses the *electroweak theory*, proposed by Glashow, Salam and Weinberg [1] that describes the electromagnetic and weak interactions between quarks and leptons (together called *fermions*) [5] as well as the theory of *Quantum Chromodynamics (QCD)*, used to describe the strong interactions between quarks. These theories combined explain the nature of 3 of the 4 forces of nature (electromagnetic, weak and strong), excluding the elusive gravitational force, whose mechanisms of propagation are still a mystery (one that recent discoveries by the LIGO collaboration [6] shed some light into).

In the Standard Model framework, the aforementioned interactions occur through fields to which fermions *couple* with a strength that is described by a quantity known as the *charge* of the particle with respect to a certain field. A particle can carry a charge under several fields and interact accordingly with several forces. Essentially, it can be said that the Standard Model is a *theory of interacting fields* [7], where each one of the fundamental forces corresponds to its own field.

2.1.1 Gauge bosons and interactions

But how is the force of a field on a fermion communicated and how do two fermions interact with respect to that field ? The interaction occurs through the exchange of the "carrier" particle of the field. Those carriers are characteristic particles for each type of field and force and are known as *gauge bosons*, or *field quanta*. In every interaction that occurs, according to the Standard Model, such a field quantum is exchanged. For example, the field quantum of electromagnetic interactions is the, massless and neutral in terms of electric charge, *photon*, and it is exchanged between electrically charged fermions. The charge of the electromagnetic force is the, well known from everyday life, electric charge, and comes with two polarizations, positive (+) and negative (-). The quanta responsible for the propagation of the weak force are the electrically charged $W^{\pm}$ bosons

and the neutral Z boson. The charge related with the weak interaction is called the *weak isospin*. Finally, the field quanta of the strong force are called *gluons* and they are massless, same as the photons. The charge of the strong force is called *colour* (there are three colour charges appropriately named red, green and blue - for each colour charge there is anti colour charge as well called anti-red,anti-green and anti-blue respectively). A fundamental difference between photons and gluons is that the second actually carries color charge during the strong interactions of the quarks inside the nucleons, while the photon is electrically neutral. A quark emitting a gluon can change its colour and for the colour charge to be preserved in the system the gluon must carry a colour (of the initial colour of the quark)-anticolour(of the final colour of the quark) charge itself (e.g. red anti-blue or, if no colour change has occurred, a blue anti-blue, for instance). This means that gluons can interact with each other as well as they are colour charged.

One other property of those gauge bosons that should be discussed is their *range* and thus the range of the forces and fields they correspond to. From Heisenberg's uncertainty principle, expressed in terms of energy and time intervals $\Delta E \Delta t = \hbar$, it is known that if a system is observed for a time Δt there is no way that we can know the energy of the system better than within an uncertainty ΔE. So a photon can have an energy ΔE for a time interval that is given by the uncertainty principle as $\Delta t \approx \frac{\hbar}{\Delta E}$. Since a photon's wavelength can be arbitrarily large or small, its energy can take arbitrarily small or large values respectively and thus it may approach zero leading to $\Delta t \to \infty$, meaning that, essentially, the fact that the photon is a massless particle means it has an infinite range. Likewise, the massive bosons of the weak interaction can be at an intermediate state of energy ΔE for a time $\Delta t = \frac{\hbar}{Mc^2}$, where M is the particle mass, and thus because these bosons have finite masses they have a finite range as well. Given the masses of the W and Z bosons ($M_W \approx 80 GeV/c^2$ and $M_Z \approx 90 GeV/c^2$) the range of the weak force is found to be rather short, of the order of 10^{-3} fm.

The range of the strong force is a more complicated matter. While the gluons are massless and one would expect their range to be infinite, the strong force has a short range and its main effect is to keep the atomic nuclei together. This occurs because, unlike the electromagnetic field, the strong gluon fields are *confining*, meaning that the QCD framework confines all free colour charges at long distances so that every observed state is *colourless*, thus has zero overall colour charge. Essentially the confinement principle states that *the strength of the gauge coupling grows stronger as the distance between the interacting particles grows bigger*, a phenomenon completely opposite to what is observed for other forces and fields, effectively prohibiting free coloured states from occurring and explaining why free quarks cannot be observed in nature - each quark carries a different colour. This occurs, for example, within a proton and thus overall a proton is colourless (a combination of red, green and blue gives a colourless state overall). Inside a hadron (in general hadrons are the composite particles consisting of quarks held together by the strong force) the quarks engage in a constant exchange of gluons.

In order to remove one quark from a hadron due to the confinement principle an extremely high amount of energy must be provided. After a certain point the creation of a quark-antiquark pair is favored. This essentially limits the range of the strong force to that "breaking point" making it very short-ranged overall. Table 2.1 provides the relative strength and range of all 4 fundamental forces.

There is another intrinsic quantity, apart from the charges, that particles carry and that is essential to separate fermions (interacting particles) from bosons (force quanta), called the *spin*. This intrinsic quantum number takes half integer values (e.g. 1/2) for

Force	relative strength	range (m)
Strong	1	10^{-15}
Electromagnetic	$\frac{1}{137}$	∞
Weak	10^{-6}	10^{-18}
Gravity	10^{-39}	∞

Table 2.1: The fundamental nature forces, their relative strengths and their range (in m). [8]

fermions and integer values (including zero) for bosons. The spin of a particle, along with the charges it carries, that indicate how strongly it couples with a field, are the quantities that fully characterize a particle and aide the process of particle identification.

2.1.2 Leptons and Quarks - the matter particles

The fermions (the particles that are the constituents of matter) are divided into two main categories (as it has already been mentioned) - *leptons* and *quarks*. All fermions are spin 1/2 Dirac fermions and obey the *Fermi-Dirac statistics* (in comparison to the gauge bosons that obey the *Bose-Einstein statistics*).

Leptons are a category of particles consisting of 3 charged particles (e^-,μ^-,τ^-) and 3 neutral ones (ν_e,ν_μ and ν_τ), known as neutrinos. The 3 charged particles, with the most fundamental one being the electron (e^-) and the other two being the muon (μ^-) and the tau (τ^-), carry unit electric charge, which is the electric charge the electron carries. The only differences between the 3 charged leptons are their masses (with the electron being the lightest and the tau being the heaviest), their lifetimes (only the electron is a stable particle out of the charged leptons with the muon and the tau having short lifetimes) and another characteristic quantum number called the *lepton quantum number* - each of the three particles has a different lepton quantum number that characterizes it, that must be conserved under electromagnetic interactions (leptons have a +1 lepton quantum number and their anti-particles have -1). Each lepton has an *antiparticle* (predicted by the Dirac equation) that has a positive electric charge (for charged leptons) and a magnetic moment whose direction differs relative to the spin of the lepton (e.g. e^+ is the antiparticle of the electron called the *positron* while $\bar{\nu_e}$ is the antiparticle of the electron neutrino). Charged leptons participate in electromagnetic and weak interactions. Each of the 3 charged leptons forms a *flavour doublet* with a neutrino that carries the same lepton quantum number:

$$\begin{pmatrix} e \\ \nu_e \end{pmatrix} \quad \begin{pmatrix} \mu \\ \nu_\mu \end{pmatrix} \quad \begin{pmatrix} \tau \\ \nu_\tau \end{pmatrix} \tag{2.1}$$

The neutrinos were considered to be massless but they actually have a very small mass, something that has been established through the observation of the *neutrino oscillation* phenomenon during which a neutrino changes its flavour. Neutrinos carry no electric or colour charge and thus they can only interact with matter through the weak interaction. This makes them very unlikely to interact at all and they can travel even through whole celestial bodies undisturbed.

Quarks are the constituent particles of larger particle compounds known as *hadrons*. Due to confinement there are two possible hadronic configurations:

- **Baryons** : They consist of 3 quarks of different colours (so that they are colorless overall and can exist as free particles). The most common baryons, that are the building materials of the atomic nuclei, are the *proton* p (with a uud quark content) and the *neutron* n (with a udd quark content). Baryons are also fermions with proton being the only stable one.

- **Mesons** : They consist of a quark and an antiquark that carries the anti-colour of the quark (and thus they are also colorless particles). The lightest, and most common in interactions, mesons are the *pions* (π^+ with a $u\bar{d}$ configuration, π^- with a $d\bar{u}$ configuration and the π^0 with a $u\bar{u}$ or $d\bar{d}$ configuration). Mesons are bosons (e.g. the pions have zero spin - an integer value - and they mediate the interactions between two nucleons). All known mesons are unstable.

Confinement principles are also the reason free quarks cannot be observed in nature as they carry colour charge, meaning they interact through the strong force within hadrons - being the only fermions to undergo such interactions. They also carry electric charge but at *non-integer* values, while they can also participate in weak interactions as carriers of weak isospin.

A total of 6 quarks have been discovered (up,down,charm,strange, bottom and top) and, like the leptons, they are separated in three doublets representing three generations as follows :

$$\begin{pmatrix} u \\ d \end{pmatrix} \quad \begin{pmatrix} c \\ s \end{pmatrix} \quad \begin{pmatrix} t \\ b \end{pmatrix} \tag{2.2}$$

in increasing mass order from left to right. The positively charged quarks u,c and t carry a charge of +2e/3 while the negatively charged ones carry a charge of -1e/3. It can easily be deduced that the total charge of the proton, for example, is indeed +e and, similarly, for the neutron that it is zero. Each quark also carries a *flavour quantum number* (e.g. strangeness for the s quark) that is generally not conserved under weak interactions. The top quark (the most recently discovered) displays a mass considerably bigger than that of the rest of the known quarks (172.44 GeV/c^2), a discrepancy that is still investigated.

2.1.3 The Brout-Englert-Higgs (BEH) mechanism and the Higgs boson

It is essential for a contemporary presentation of the Standard Model to include a description of the BEH mechanism ([9] - [13]) and the most prominent discovery so far of the ATLAS - and CMS - experiments at LHC, the scalar Higgs boson. The Higgs mechanism is essentially invoked to break the electroweak symmetry giving mass to massive elementary particles, and it implies the existence of a neutral scalar particle, the *Higgs boson* [14]. In practice, breaking the electroweak symmetry implies a splitting of the massless gauge bosons of the underlying symmetry into the massless photon of the electromagnetic interaction and the massive W and Z gauge bosons of the weak interaction, thus splitting the, up to this point unified, electroweak theory to the electromagnetic and weak interaction.

Absent the field related to this mechanism, the *Higgs Field*, it is assumed that all particles would be massless. It is the universal interaction of the massive particles with this field that provides them with their observed masses, with heavier particles having a greater coupling strength to the Higgs field and thus acquiring larger mass. It can be

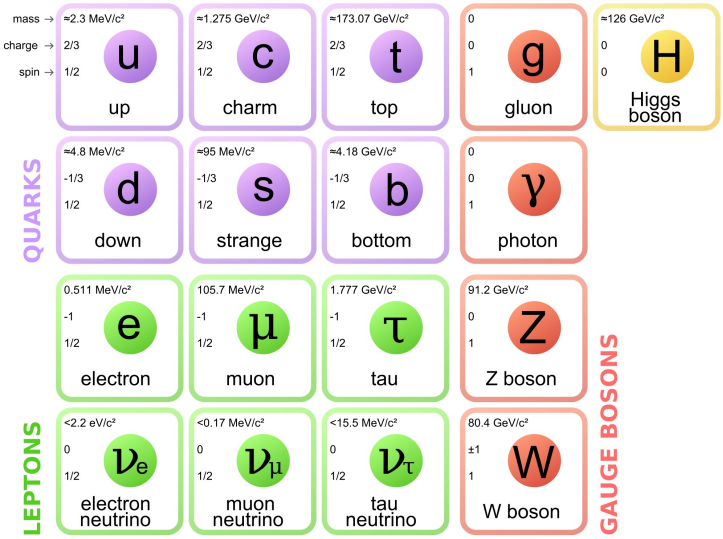

Figure 2.1: The components of the Standard Model as established to this day.[15]

deduced that massless particles, such a photons and gluons, do not interact with the Higgs field at all while neutrinos interact very little given their very small masses.

The *Higgs boson*, a scalar particle, is the mediator of this field, and its existence was confirmed in 2012 by the ATLAS and CMS collaborations [16, 17].The discovery was based primarily on mass peaks observed in the $\gamma\gamma$ and $H \to ZZ \to 4\ell$ decay channels of the boson that offer the best resolution. The latest estimate for its mass, with data combined from both experiments (see [18]) is $m_H = 125.09 \pm 0.21(stat) \pm 0.11(syst)GeV$.

2.1.4 Dark Matter

When the term "Dark Matter" is considered, exotic and mysterious forms of matter come to mind, matter beyond what human senses can observe. Dark Matter refers to a, still hypothesized, form of matter that does not interact with electromagnetic forces and thus does not emit light. Consequentially, it does not interact with what is considered "normal" luminous matter, making it practically undetectable by conventional means humans can develop. The existence of such a non-interacting form of matter can be inferred by a number of observations made :

- The discrepancy in the mass of galaxies and galaxies' clusters between their mass as inferred by gravitational effects and their observed mass as calculated by their emitted light distribution. The observed luminous mass would not be enough to hold a galaxy cluster system or a lone galaxy bound together.

- The rotational curve of a galaxy fails to follow Newton's laws beyond the luminous disk, where the rotational speed should fall as $V_c = r^{-1/2}$, however, it is observed to remain constant at a radius of 50 kpc (the luminous disk radius extends to a distance of 10 kpc with our Solar System at about 8.5 kpc from the galaxy center), so the existence of an extended Dark Matter halo beyond the luminous disk is suspected.

- Observed anisotropies in the cosmic background radiation.

LHC and in particular the ATLAS experiment are actively searching for Dark Matter candidate particles [19], such as the Weakly Interacting Massive Particles (WIMPs) -

hypothetical particles thought to be the Dark Matter constituents that may arise in sypersymmetric extensions of the Standard Model and may couple to the known matter through a kind of weak interaction, either generic, known or a new kind. The ATLAS collaboration probes into event topologies that have a single jet characterized by a large transverse energy and also large missing transverse momentum - these jets are known as *monojets* and, for example, searches for an excess to such monojet events beyond SM expectations (where a monojet can be the product of a Z boson-jet production where the boson decays into two neutrinos that escape detection) can potentially lead to the discovery of such Dark Matter candidate particles. That is because the WIMPs fail to interact with the materials of the detector and thus the fact that they escape the detector undetected leads to missing transverse momentum p_T^{miss} and thus missing energy E_T^{miss}.

2.2 The LHC and the ATLAS experiment

The Large Hadron Collider (LHC) is the centerpiece of research currently at the accelerator complex of the European Organization for Nuclear Research, commonly known as CERN, located on the border between France and Switzerland. With a circumference of 27 km the ring-shaped accelerator is the largest particle accelerator ever built. It has contributed (and continues to contribute) to the expansion of the frontiers of particle physics with the unprecedented high energies and luminosities it operates at. Inside the accelerator's beam pipes (kept at ultrahigh vacuum) two high energy particle beams collide at dedicated collision points (guided, bent and focused by strong magnetic fields generated by various multipole superconducting magnets, such as dipole and quadrupole magnets), around which the detectors of the various experiments are built (the 4 major ones being ATLAS, ALICE, CMS and LHCb) that study those collisions and the particles produced from them.

ATLAS (which will be presented in more detail in Chapter 2.2.1) and CMS function as more general purpose experiments, probing into a wide range of physics - their independent operation allows for reliable confirmation of important results and discoveries, such as the independent confirmation of the discovery of the Higgs boson. ALICE and LHCb focus on more specific phenomena such as heavy ion collisions and b-physics.

A major consideration in experiments, such as those that aim to observe rare phenomena and expand the current knowledge in the physics field, is the rate of collisions, the rate at which events are observed, and thus, ultimately, the total number of events recorded. A higher number of recorded events makes it more likely for rare phenomena to be observed and a discovery to be made. This is where the concept of *luminosity* (and more specifically *instantaneous luminosity*) L is introduced. It indicates the number of potential collisions per second, while *integrated luminosity* indicates the cumulative number of collisions over a time period Δt. A higher value of integrated luminosity makes more likely the discovery of new phenomena. The actual probability of an interaction to occur is generally expressed in particle physics experiments by the *cross section* σ of the reaction.

A major quantity of interest for all LHC experiments is the *transverse momentum* p_T of the, produced from the collision of LHC beams, outgoing particles. Since it is not possible to know the initial momentum of the colliding bunches and given that the movement along the beam axis of the initial particles allows the consideration of the initial total transverse momentum as 0, any p_T final particles have is due the collision

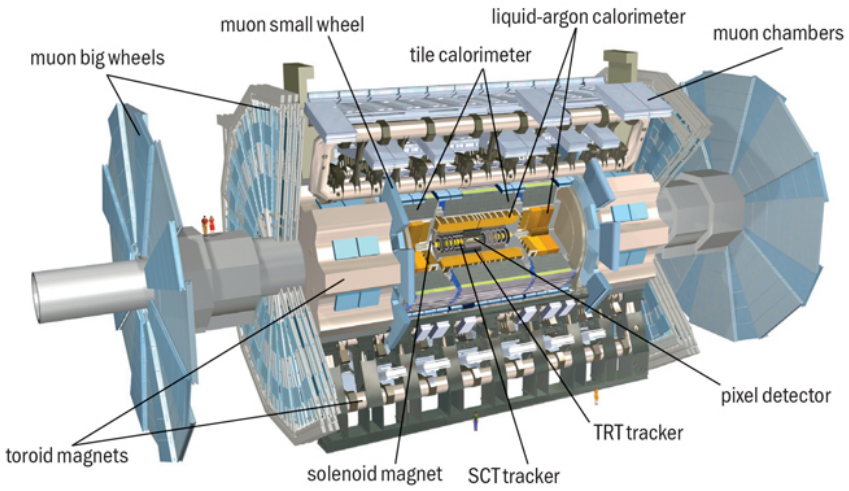

Figure 2.2: The layout of the ATLAS Detector with all its major sub-elements indicated.[22]

itself. Momentum preservation imposes limitations on the transverse momenta of the final particles as the total p_T must be zero. These conditions make p_T a very useful quality to use in order to probe the physics of the collision and its aftermath and calculate other quantities such as the invariant masses of particles and particle jets.

For the majority of its time of operation the particle beams colliding are beams of *proton bunches*. While the collision of proton beams dominates most of the operation time of LHC, lead ion beams also collide within the LHC with the properties of such collisions (such as the creation of the *Quark Gluon Plasma*) studied primarily by the ALICE experiment. During its first period of operation, known as Run 1, from 2009 to 2013 the center-of-mass energy for the proton beams started at 7 TeV to reach eventually the 8 TeV. After a shutdown period, during which the LHC was upgraded, the ongoing run, known as Run 2, started in 2015. During this run, the center-of-mass energy for proton-proton collisions has considerably increased, reaching the 13 TeV, with beam intensity also increasing. At the end of the 2017 run, the accelerator achieved record luminosities with the two major experiments, ATLAS and CMS, reporting a total *integrated luminosity* of 50 fb^{-1} of data, corresponding to about $5 \cdot 10^{21}$ events, about 60 collisions per bunch crossing and a record value of instantaneous luminosity of $L = 2.06 \cdot 10^{34}\ cm^{-2}s^{-1}$.

2.2.1 The ATLAS experiment

The ATLAS detector [20] is a multipurpose particle physics apparatus. It has a length of 46 m, a height of 25 m and a forward-backward symmetric (with respect to the interaction point) cylindrical geometry. It is located 100 m underground, built around one of the beam collision points inside the LHC.

The ATLAS experiment aims to explore mass scales of the order of TeV. The high luminosity and increased cross-sections at the LHC allow for additional tests that probe into QCD, electroweak interactions and flavour physics with enhanced precision. ATLAS focuses on the investigation of electroweak symmetry breaking, which is inextricably linked to the discovery of the Higgs boson, as well as the search for physics beyond the Standard Model, such as WIMPs and other Dark Matter candidate particles.

The goal of a particle physics experiment that hopes to probe into new phenomena

and make discoveries is to track every particle produced by a collision. To that extent the ATLAS detector itself is a many-layered instrument designed to track as many of the outgoing particles produced, for example, from a proton-proton collision, as possible through their interaction with the different sub-detectors it consist of.

The major components of the detector are [20, 21]:

- **The Inner Detector**: It is the part of the detector closest to the interaction point where the collision takes place. At this area the track density is still extremely high and thus this detector is very compact and has a high sensitivity in order to provide excellent spatial resolution of the signals for purposes of momentum calculation and vertex reconstruction. The reconstruction of the primary vertex with high precision is the central task of the inner detector.

- **The Calorimeters**: They are constructed to measure the energy losses of particles passing through them and their main task is to contain and measure the whole energy of a traversing particle, essentially stopping them and "absorbing" them. Today, calorimeters can practically stop all known particles except from the very energetic muons and the rarely interacting neutrinos.

  The system of calorimeters at ATLAS consists of two main components: the *Tile Calorimeter (TileCal)* and the *Liquid Argon (LAr) Calorimeter*. All calorimeters used at ATLAS are *sampling calorimeters*, meaning that they consist of a series of interleaved *passive layers*, made of an absorbing material with high atomic number Z that causes considerable energy loss, and *active layers* consisting usually of solid lead-glass or liquid noble gases (such as Argon) that detect this energy loss through ionization (for noble gases) or scintillation (for the solid materials). According to the kind of particles they identify the calorimeters are divided into *electromagnetic* (LAr electromagnetic calorimeter) and *hadronic* (Tile Calorimeter, Hadronic End-Cap (HEC) and Forward Calorimeter (FCal). Their functioning principle is based on the measurement of the energy of leptons and photons or hadrons respectively that is deposited in the form of showers.

- **The Muon Spectrometer**: They comprise the outermost layer of the ATLAS detector and are used to measure the momenta of muons that are so energetic that pass through the inner detector and the calorimeters without depositing their energy. A group of 3 toroid magnets, described in the next section, provide the force that bends the tracks in order for the particle momenta to be calculated from the curvature.Identification with the spectrometer is easier as particles other than muons are not expected to make it to those detectors.

- **The Magnet System**: It consists of a central solenoid magnet that is thin and superconducting that surrounds the inner detector as well as three large, superconducting toroids - a barrel toroid and two end-cap toroids perpendicular to the collision axis. The field-induced curvature of the transversing particle tracks is used to determine the transverse momentum of the charged particles. The toroids are used to determine the momenta of very energetic muons detected at high pseudorapidities η the solenoid may miss. Pseudorapidity is a spatial coordinate used in particle physics mainly to describe the angle of the track of a particle relative to the beam axis and it is defined as $\eta = -ln[tan(\frac{\theta}{2})]$, where θ describes the (polar) angle formed between the 3-momentum vector of the particle and the positive direction of the beam axis.

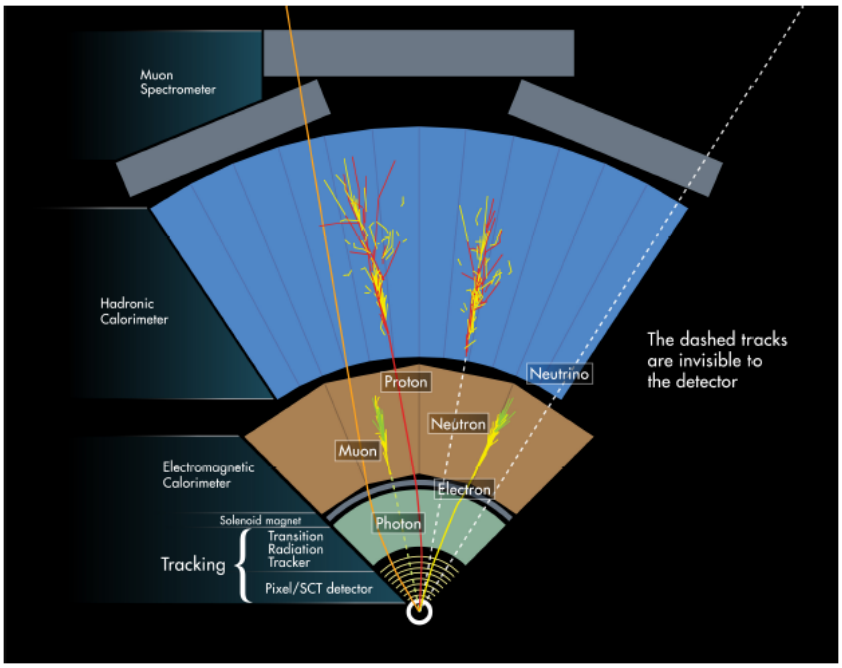

Figure 2.3: Cross-sectional wedge view of the ATLAS detector depicting the different layers of sub-detectors radially distributed around the interaction point and the interaction of different particles with different layers.[8]

Figure 2.2 depicts the layout of the detector along with its sub-detectors and other elements contained within it. Figure 2.3 shows a cross-sectional view of the sub-detector layout along with the identification principle through particle interactions with the different layers.

After the particles have been detected and their transverse momenta defined, a three-level trigger system is used to select those events with distinguishing characteristics that make them interesting for further physics analyses and subsequent off-line analysis, with only data for which all subsystems described above were operational used. Then the ATLAS computing system calibrates, reconstructs and distributes the data to all the members of the ATLAS collaboration around the world, to which they have access through the *Worldwide LHC computing Grid*.

Currently the final data format used in results' analysis is what is called the *Derived Analysis Object Data, or DAOD*, and it includes solely physics objects, such as jets and electrons, that are of interest in a particular analysis. Such objects can still have considerable size - even though the data size has been significantly reduced, since the raw detector data was delivered. That occurred through their transformation to event level data, where only some track parameters are stored instead of every coordinate of every hit. The energy deposits in the calorimeter are then combined to form the aforementioned physical objects. Thus it is recommended for the analysis to be done on distributed computing facilities rather than local computer units by sending the analysis code to where the data is stored instead of the reverse.

2.3 The concept of jets in particle physics

Now that both the Standard Model of particle physics and the ATLAS experiment have been described it is essential for part of the work done during this project to briefly explain the concept of *jets* in particle physics experiments. A jet is *a collimated spray (or spread) of particles that have partonic origins* (they are created during the process of hadronization of quarks and gluons), whose emergence after a particle collision can help the identification of hadronic final states in experiments such as those performed at the LHC, including ATLAS. The experimental analysis of the energy and angular distributions of jets can provide valuable insight to the properties of the basic constituents of matter and the strong force that mediates their interactions. Consequently, their observation was crucial in the establishment and validation of Quantum Chromodynamics (QCD) as the theory that described strong interactions within the bounds of the Standard Model. Such hadronic jets constitute the main contributors to processes that generate high transverse momentum (p_T) particles and have been observed since the early experimentation phases at CERN, before the LHC was even created, at the days of experiments such as the UA2 [23].

Figure 2.4 gives a visual representation of the development of a jet after a partonic collision. Directly after the collision (during which it is quarks and gluons inside the colliding protons that actually interact through hard scattering), high energy quarks and gluons move as quasi-free particles at short distances, developing a cascade of bremsstrahlung radiation that includes a collection of narrowly collimated gluons as well as quark-antiquark pairs (*parton-level jets*). These lead to a gradual decrease of the energy of those partons and, at the same time, an increase in their number. At a certain point during this process the parton energy level decreases to the point where these quasi-free partons cannot escape confinement anymore and are forced to hadronize into mostly mesons, equally well collimated (*hadron-level jets*), that reach the detector calorimeter and deposit large amounts of energy, a "shower" of energy, to a certain direction within the calorimeter. These are what the detector observes and constitute what is called *calorimeter-level jets*. What the detector receives is the signature of a quark or a gluon that has mapped itself, from a small distance, to a jet of hadrons at large distances. The configurations of the initial high-energy partons are reflected in the properties of the final-state jets that are observed, thus making jet identification and analysis extremely important.

The identification of jets constitutes one of the most crucial steps in the process of particle identification in ATLAS, and other experiments. Nowadays, such identification proceeds through the use of *algorithms* that are required to reproduce the calculable results QCD provides. These algorithms need to be well-defined and be able to relate calorimeter-level, hadron-level and parton-level jets to achieve proper reconstruction and identification of the high energy partons that are the origins of a particular jet. The dominant algorithms used presently are part of a family called the *sequential recombination algorithms* [24]. Those algorithms fulfill two very important QCD-based requirements: They are *infra-red* ***and*** *collinear safe (IRC safe)*. Infra-red safety ensures that a soft emission (of a gluon, for example) does not influence the conclusion of the algorithm (does not change the jet), while collinear safety requires that a small-angle splitting in the jet cone cannot change the algorithm conclusion either. One such algorithm (anti-k_t) is used in the samples studied during this project and some more information on it will be provided in Chapter 4.2.1. Another family of algorithms, known as *cone-type algorithms*, do not offer IRC safety and, while they tended to be favoured at hadron colliders due to some computational

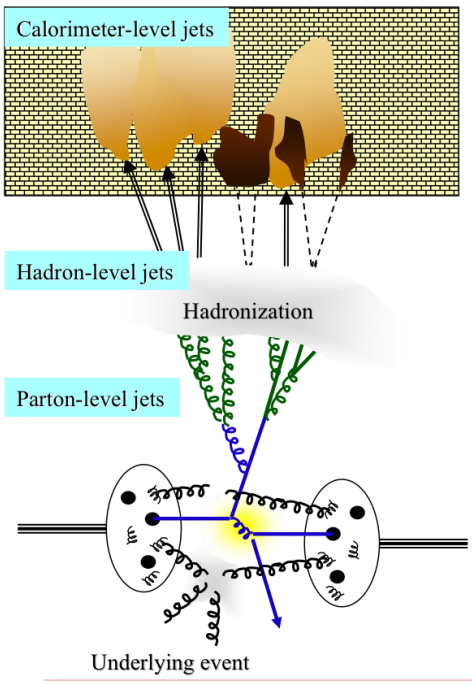

Figure 2.4: The creation and development of a hadronic jet.[25]

performance benefits, today, recombination algorithms are dominating the field both in theoretical simulations and experimental jet identifications in data [26]. For experimental purposes a jet is only the collection of objects that occur as the outcome of a jet finding algorithm - nothing else is identified as a jet.

2.4 Monte Carlo generators and samples

Monte Carlo simulations consist a central concept in this project. Thus the significance of the contribution of Monte Carlo generators to particle physics, as well as their actual function, is also introduced.

Monte Carlo simulations have long been used in particle physics to test the validity of theoretical models and predictions by accurately interpreting data produced by collision experiments, such as the proton-proton collisions (pp collisions) taking place at the LHC. A Monte Carlo generator (also known as an event generator), in the context used in particle physics, is essentially a computer program used to create a simulation of the studied process (in the case of the ATLAS experiment mostly p-p collisions) based on the theoretical predictions of a tested model (in the case of ATLAS and LHC this is the Standard Model of particle physics). The randomly generated events simulate those produced in particle accelerators, where particle collisions at high energy occur, but also events that occurred during the early stages of the creation of the universe. Monte Carlo simulations are, in general, based on the concept of *random sampling of the possible outputs* of a, probabilistic in nature, process, dependent on many variable factors. A distribution of those values should converge to a statistical sample that accurately describes the expected results of the studied process. Then this distribution can be used to be compared with actual data, gathered by detectors such as those of the ATLAS experiment. Such a comparison can lead to either a confirmation of the model, the need for its modification or its rejection. The ATLAS experiment relies heavily on such simulated collisions to make

its SM-based predictions. Commonly used Monte Carlo event generators at ATLAS include PYTHIA [27], Herwig [28] and Sherpa [29]. Those generators do not take under consideration any detector effects, which are always present in real data and are related to effects such as energy loss at the detector material. This is corrected with the use of the GEANT4 [30] detector simulator that simulates the passage of the various produced particles through the detector material in detail based on the geometry of a detector.

During this project, the PYTHIA generator will be mainly used, in particular the latest version of that generator, PYTHIA8. It is a reliable tool for generating high-energy collisions of a wide range of models that include processes involving few interacting bodies to complex multi-hadronic final states. This tool provides a library that contains hard processes and models for both initial and final-state parton showers, parton-parton interactions, particle decays, beam remnants, beam jets and even string fragmentation models. The current version of PYTHIA is based on the C++ programming language.

PYTHIA, as well as other event generators, run within a C++ control framework called Athena [31], in which data processing and analysis is performed. The function of this framework is based on the idea of the execution of algorithms that produce or take as input data objects. These objects may be particle clusters or isolated particles, such as muons or electrons.

Chapter 3

The ATLAS@home Project

3.1 The Virtual Machine Concept

The Monte Carlo generation, that consists the goal of this project, is done using a software called BOINC [32] that distributes tasks to the various machines and spawns a virtual machine on each of them to process those tasks.

A *Virtual Machine (VM)* is what could be called a "software computer", a software emulation of a hardware machine [33]. Like a physical computer, it can run an operating system and applications within it and it contains a number of virtual devices that provide a functionality similar to that of a physical computer. A virtual computer, usually called a "guest", is run inside a physical computer, usually called the "host". The virtual machine consists of a collection of configuration and specification files that setup the environment for the operating system and its applications to run using the physical resources provided by the host machine.

A *virtualization software* is a kind of software that creates the virtual machine environment. To achieve that it gains access to a number of hardware components and certain features of the host machine. Such a software performs what is known as *full virtualization*, a kind of virtualization that allows a completely unmodified operating system, along with all its installed software, to run on the virtual environment especially created for that purpose. Almost all the guest code, belonging to the virtualization software, is run on the host machine with the guest operating system "thinking" it runs on a real machine, it "sees" all the software virtualized hardware as if it was real hardware.

It should be noted that this virtualization process differs significantly from an emulation process, showcasing considerable benefits. During an emulation process the machine instructions and configuration of the guest system are "translated" (emulated) and the guest code does not run on the host directly. An emulation software (such as BOCHS [34]) allows a type of software written for a certain type of hardware to be run on another hardware type (for example a program written for a 64-bit machine to run on a 32-bit machine) but the performance in terms of speed is poor. A virtualization software can run code that was written targeting similar hardware (for example a 32 bit Windows machine can run a 32-bit Linux environment) but the benefit in this case is a performance comparable to the performance the software would have if it was run on its native hardware.

In this project the virtualization software used is Oracle VirtualBox, a "general-purpose full virtualizer for x86 hardware, targeted at server, desktop and embedded use." [35]

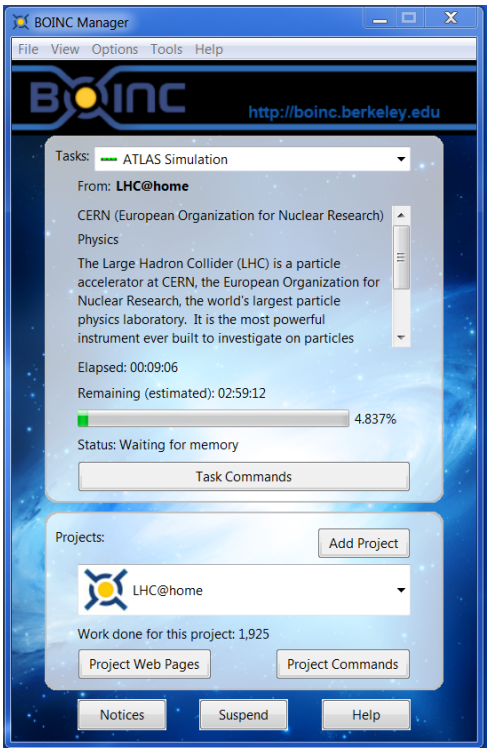

Figure 3.1: Screenshot of the BOINC application environment while running an ATLAS task for the LHC@Home project showcasing the progress that has been made, the time elapsed and the estimate remaining time for the processing of the task to finish.

3.2 The BOINC Volunteer Computing Platform

BOINC is the open-source software developed at UC Berkeley the ATLAS@Home [36] project utilizes to communicate its tasks to the users. It is the software used by the majority of *volunteer computing projects* around the globe. Such projects take advantage of the spare computing cycles of personal computers when they are not used in order to perform a computational task someone else needs (usually a researcher or a research team). The stimulus for people to participate in volunteer computing projects that are commonly related to scientific work is the sense such a contribution can offer to people outside the scientific community that they are contributing to the expansion of scientific knowledge and thus to the greater good.

SETI@Home [37], a project developed at UC Berkeley that aimed to discover evidence of extra-terrestrial life by checking radio signals provided by telescopes, is considered the first major example of a large scale volunteer computing project and the reason why BOINC was initially developed. The participants in the project run a free program that downloads and analyzes such radio telescope data by using the computational power provided by their personal computers. To this day the number of major scientific projects in need of such computational power that have turned to volunteer computing using BOINC has increased and as of January 2018 it includes 37 active projects [38] (including a diverse range of subjects from mathematics to linguistics and medicine and from astronomy to molecular biology) with the number of volunteers also increasing.

But how exactly does this software function ? The basic idea is the establishment of a reciprocal relation between a server and the user's computer. The serves functions as a host to numerous *tasks or work units* related to specific projects that the user can

download on demand to process on their computer. Due to the fact that every personal computer has different operational characteristics and computational capabilities, the BOINC software makes use of virtualization in order to realize the suitable environment for the available tasks on the server to be executed. This means that the relevant software must run through a virtual machine. Thus when the user chooses to contribute to a certain process, a virtual machine is spawned, via an appropriate virtualization software like VirtualBox, on the user's computer. A piece of software called `vboxwrapper` that runs on the BOINC client side controls the creation of the virtual machines. The work unit is then downloaded and processed on that virtual machine. Once the processing of the unit has been successfully performed, the result returns to the server where it is stored after it is validated. A successful validation awards the user with a certain amount of credits. Those credits have no real value, monetary or otherwise, but serve as a measure of the processing that has been done on a certain computer for a certain project and it is supposed to work as some sort of motivation for the volunteer.

The field of particle physics could not remain of course uninterested in such a concept. The use of BOINC by CERN started all the way back in 2004 with the establishment of the LHC@Home [39] project. 10 years later the ATLAS experiment created its own volunteer computing project called ATLAS@Home, now an application that is part of the LHC@Home project that functions as an umbrella project for all LHC-related volunteer computing projects such as ATLAS@Home, CMS@Home, SixTrack and others.

3.3 The LHC@home platform and the ATLAS@home project

The ATLAS@Home project is using the BOINC software to distribute ATLAS-related computing tasks, capitalizing essentially on the high public interest in the LHC physics research. The general LHC@Home umbrella project gives the capability to the user to choose according to his preference which application they want to run (for example a user interested solely in ATLAS research can choose to process only tasks from the ATLAS@Home application within the LHC@Home project). The project has two main purposes, aimed at benefiting both the performed research, by providing extra computational resources, as well as the general public (with the project having a constantly expanding base over the last 2 years of its operation) by involving them in a direct way to the ATLAS research. With a constantly expanding volunteer base, ATLAS@Home now produces about 5-7 % of the total number of ATLAS simulation events with a cost negligible compared to what that would be to run an equivalent Grid site.

In the case of the ATLAS@Home project, each virtual machine BOINC spawns uses 10 GB of space within the drive of the user's computer and utilizes the *CERNVM File System* (commonly known as CVMFS) [41] to distribute the relevant ATLAS software needed to process a work unit of the task the user has chosen. The jobs themselves are taken from *PanDA* [42], the ATLAS job management system, and they are then submitted to the BOINC server through the *Advanced Resource Connector Compute Element (ARC CE)* [43], a Grid front-end on top of a conventional computing resource such as a Linux cluster or a standalone workstation, handling mainly Grid user authentication issues and Grid file managing and user access to Grid files.

An overview of the architecture of ATLAS@Home is shown in Figure 3.2.

Over the last 2 years ATLAS@Home has undergone a number of improvements and

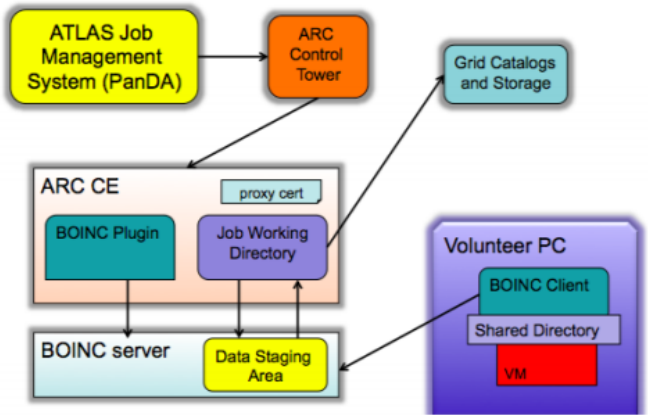

Figure 3.2: A graphical overview of how the ATLAS@Home application works, from taking the job from PanDA and eventually delivering it to the volunteer's PC. [40]

changes with the purpose of improving the performance yield, as well as to make the platform more attractive for the volunteers. The major three changes towards this direction were:

- The implementation of a ***multi-core version*** of the application, to make sure full usage of the CPU cores a volunteer offers is achieved, while gradually ending support for the single-core application.
- The implementation of a ***graphical interface***, used to provide understandable information about the generated tasks as well as a general background about the physics studied at the ATLAS experiment.
- An undergoing effort to integrate the ***ATLAS 3D event display, VP1*** [44] to the application, in order to provide even more information to the volunteers in the form of detailed visualizations of the processed data. Those visualizations are produced within the virtual machine (and are thus task-specific) and displayed through the aforementioned graphic interface.

The above improvements and changes are described in [36].

Chapter 4

Monte Carlo samples generation

4.1 Nature of the studied Monte Carlo samples

In order to achieve the goal of the project, there is a need to have some *reference samples* of generation tasks. For this particular project, a script that generates some generic QCD Monte Carlo samples has been provided.

The script that consisted the main body of this work, for the most part, is wholly available at Appendix A.1. We will provide a step-by-step description of what this script actually achieves.

First of all the basic directories where the generated files will be stored are created along with certain simulation parameters such as the center of mass energy of the collision, the total number of events and the total number of jobs that need to be produced - for the total number of jobs, it is chosen to have it provided on the command line upon execution of the Python script, but a standard number of jobs can also be set. An offset value that will aid in the seed generation of the Monte Carlo production (as it will be seen later) is also provided. It is optimal for such a script to provide a random offset value upon execution and thus why the `random` Python package and its functions are used here to generate such a number:

```
offset = random.randint(100000000,999999999)
```

For the production of the reference samples a constant offset value will be used to provide the same MC generation seed in order to enable comparisons across different processing environments. Next, a bash executable submission file is created that will execute all the generated job scripts.

A fundamental file that needs to be generated in order to progress further is what is known as the *Job Options*. To explain what the Job Options are first it needs to be made clear that the jobs that need to be generated and run are essentially *Athena jobs*, the event loop framework that comes with all ATLAS software releases. It manages almost all ATLAS production workflows including event generation, simulation, reconstruction and derivation production. In order to use Athena a python configuration file must be created that specifies exactly what the user wants to happen during this event loop. This file is what is called the *Job Options* and its main purpose in this script is to define jet QCD processes running with PYTHIA and set some minimum bias selection parameters.

After the Job Options have been generated, it's time to generate the job scripts that will be used for Monte Carlo validation.

A `for` loop is used to generate the number of `bash` executable job scripts requested (dependent on the `numberOfJobs` variable) and assign them a name. Then the job scripts are being written, starting with a basic tracking of the running shell. A temporary directory is created (`tmpDir=‘mktemp -d‘`), in which the event generation will take place and the output files will be created, before transferring any output files that are wanted to actual directories.

After a check is ran to ensure the directory in which job options were stored exists, the Job Options file is copied from the directory where it was originally placed in the temporary directory. The relevant ATLAS software and Athena release needs to be installed:

```
#module load enableATLAS
#setupATLAS
#asetup 19.2.4.16,AtlasProduction
source $ATLAS_LOCAL_ROOT_BASE/user/atlasLocalSetup.sh
source $AtlasSetup/scripts/asetup.sh 19.2.4.16,AtlasProduction
cd ${tmpDir}
```

The first three lines are to be used when ATLAS software source code already exists on the machine the script is running (for example Iridium, a local cluster presented in more detail in Chapter 4.2.1, has an ATLAS software module the first line of code activates). If not, the source must be loaded and executed with the 4th line of code and then the appropriate Athena framework release must be setup (`asetup`) for the project that is supposed to be loaded for. A project is a subset of the code of the Athena framework that is specialized in running certain tasks - for example `AtlasProduction` is an ATLAS-specific project used for event generation - projects will be discussed in more detail in Chapter 4.2.2.1. The code ensures the working directory is the temporary directory previously created and then the event generation using the PYTHIA generator is performed through the `Generate_tf.py` command below:

```
Generate_tf.py --ecmEnergy="""+centerOfMass+"""
--runNumber="""+str(channelNumber)+"""
--firstEvent=0 --maxEvents="""+numberOfEvents+"""
--randomSeed="""+str(jobNumber+offset)+"""
--jobConfig="""+outputJOFileName+""" --outputEVNTFile=event.root
```

The above command generates a proton-proton collision using the PYTHIA event generator for a center of mass energy of 13 TeV, defining a number of the run value, and then defining the range of events starting from event zero and going up to a max number of events (in the scripts running for this project this remains standard at 10000 event per simulation). Then, the important value of the seed the generation will use is defined. Each seed number will generate a different simulated collision and distribution of particles in every event, thus different measurable quantities such as p_T and jet distributions.

It is important in Monte Carlo simulation to generate many different jobs with different seeds and thus different distributions. This is what is done here as the seed value is starting from the offset value for the first generated job and incremented by 1 for each subsequent Monte Carlo generation script created, thus assigning a different seed to each job. The Job Options provide the necessary job configuration, tailoring the generation based on the information provided in that file, and finally the name of the output file is specified.

Event generation jobs in ATLAS produce an output file format called an *EVNT file*. This file contains the *truth record* in HepMC format ("an object-oriented event record written in C++ for High Energy Physics Monte Carlo Generators"[45]) and wraps it in a form that is readable by Athena. The problem with this file format is that it cannot be read as it is by the ROOT[46] analysis framework that is going to be used. Many analysis objects, that are needed for this analysis as well, such as jets, can be constructed from other particles in the truth record but they are not immediately available from the HepMC format record. This is why there has been software developed to actually convert those EVNT files to forms readable from ATLAS-related software, as well as the more generally applicable ROOT framework (which will be used in this project as well for the upcoming analysis) - the readable forms are what is called truth *xAOD* formats, a term that includes a variety of formats such as the DAOD files previously introduced in Chapter 2.2.1, that include all the (analysis-ready) information needed.

That is the next step taken: running a command to generate a truth DAOD from the event file the event generation created. In order to do that a new Athena project must be set up:

```
source $ATLAS_LOCAL_ROOT_BASE/user/atlasLocalSetup.sh
#asetup 20.1.8.3,AtlasDerivation
source $AtlasSetup/scripts/asetup.sh 20.1.8.3,AtlasDerivation
```

As it can be seen, the project loaded now is `AtlasDerivation`, an ATLAS-specific code subset used for DAOD derivation. Then the relevant command (`Reco_tf.py`) is executed:

```
Reco_tf.py --inputEVNTFile=event.root --outputDAODFile=output.root
--reductionConf TRUTH3  --loglevel FATAL
```

What this command does is take the EVNT file just generated in the temporary working directory all the processes take place and produce a DAOD file called `output.root` using a `TRUTH3` truth format (explained below) to do so.

The concept of truth describes what actually has transpired during a collision and after it when we discuss simulations. A *truth record* is the record of particles that really existed in the generated events. In ATLAS truth takes several forms. There is the *generation truth*, that is a representation of the particle interactions in Monte Carlo, essentially a snapshot of the event as it was produced by the generator - the "physical" event. Then there is the *detector simulation truth* - the detector simulation appends new particles to the generation truth due to the simulated interactions with the material that produce these new particles. The generation truth serves as the only source that the detector simulation uses as input. In order to properly represent truth certain concepts related to particle interactions have to be made clear. Every particle interaction that occurs has to be described by:

- A list of the *incoming* particles.
- The *interaction vertex*
- A list of the *outgoing* particles.

All particles, incoming and out going, have kinematic properties and flavours (types), while each vertex has temporal and spatial properties describing when and where an interaction happened. This simple image is reproduced many times during an event, with the outgoing particles for one vertex (their *production vertex*) being the incoming particles for another vertex (their *ingoing vertex*) and the event actually containing numerous such vertices and particles connected in long interaction chains. Most of the times the users are interested only in the particles, not the interactions - nevertheless a basic truth requirement is that the vertices must correspond to actual physical processes. It is essential during this process for the initial information - the generation truth - to be preserved through the reconstruction of particle tracks back to the original vertices and matched to reconstructed objects - the truth event must be connected throughout the simulation software chain.

There are multiple truth formats to configure the reconstruction that produces the truth DAOD file here from the EVNT file generated by the original event generation command. The one used is called `TRUTH3`, which is the main format for truth analysis that is used in any occasion the user is not sure what exactly may be needed. The basic idea behind `TRUTH3` is that any truth information that may be needed by parsing the truth record is saved for the user. The main truth record is actually removed from this format and the various particles contained in the record are copied into their own classes. This reduces considerably the size of the final derivation output and helps to standardize some of the truth information as well. For more information on truth formats one can consult the relevant ATLAS software tutorials on TruthDAOD production [47].

After the DAOD file is created, a specific directory to store it is created, if it does not exist already, and the truth DAOD file is copied in there and renamed appropriately and uniquely in accordance to the offset and job number used, along with the log file of the derivation process, while everything less in the temporary directory is deleted with the `rm*` command.

```
cp DAOD_TRUTH3.output.root """+outputDir+"""/OutputNtuples/SplitQCD/"""
+outputJOFileName+"""_"""+str(jobNumber)+"""_"""+str(offset)+""".root
cp output.log """+outputDir+"""/OutputNtuples/SplitQCD/"""+
outputJOFileName+"""_"""+str(jobNumber)+"""_"""+str(offset)+""".log
ls
rm *
```

Finally the job files are made executable and a line that executes them is added to the submission file created at the beginning, that helps to submit all the created jobs with the execution of that file only (depending on where the file is executed a different command is used to run the `bash` executable job scripts, something that will be analyzed in the coming chapters).

4.2 Reference Samples Production and Analysis

Two kinds of reference samples have been produced during this project. One was produced on a local cluster of the Lund University called *Iridium*. A second set of reference samples was produced by submitting the scripts to the *Worldwide LHC computing Grid* (called the Grid from now on) using the PanDA workflow framework of ATLAS. In both cases the test scripts were processed and produced 2 jobs with the same generation seed and

the same number of events in both cases (later the very same process will be done on the VM itself for the exact same jobs for comparison).

A decision on what kind of quantities should be used as samples had to be made. It was ultimately decided that two characteristic quantities widely used in particle physics research - *the jet transverse momentum p_T and the invariable dijet mass m_{jj}* - would be used.

4.2.1 Iridium samples

The Iridium Cluster [48] is a computing facility aimed to help the researchers in the fields of particle, nuclear and theoretical Physics located at Lund University. A different user group is defined for different divisions and research interests. It consists of 10 nodes, that can be directly accessed, with each node consisting of 16 cores and 64 GB of RAM. Iridium offers a personal storage service to each user where files can be stored that can be picked up directly by any node of the cluster. Finally, a batch processing interface allows many jobs to run in parallel on different nodes and cores, a feature the submission of the samples here will take some advantage of.

The first step towards generating the reference samples is the execution of the *Python master script* in order to produce the Job Options and the `bash` executable job scripts. As it was previously mentioned, a non-random offset value is chosen in order to produce jobs with known generation seeds and the number of created job scripts is set to 2. Although one sample would be enough for the purpose it is needed, we opt to produce a second sample for additional statistics. The two job scripts will from now on be conveniently named "Job 0" and "Job 1" and this convention will be kept throughout the comparison process for all environments on which the reference samples will be produced.

After the generation of the job scripts, the additional file `submitAllSubmissions8.sh` will be executed on Iridium but using the batch interface of Iridium by substituting the `bash` command, used to execute a bash script, with `sbatch`. This way the `bash` executable job scripts will be submitted to one or more of the nodes on Iridium and processed in parallel thus making the generation on Iridium quite time efficient compared to a simple bash execution in a single shell.

After processing has finished, the DAOD files needed are ready to be studied. In order to extract the quantities that will be studied and do the appropriate comparisons, the *ROOT data analysis framework* will be used. This framework provides all the needed functionalities to handle big data processing, statistical, analysis, visualization of results (mainly and in the current project in the form of histograms) as well as storage. It is a software mainly written in C++.

Once ROOT is launched on Iridium and a DAOD file is loaded in it, its contents can be viewed by opening a `TBrowser`. The DAOD contains its information in a *TTree structure*, an object that can store large volumes of same type data in an efficient way, reducing needed storage space and ensuring faster access to that data. A `TTree` can hold various kinds of data, from simple numerical values to objects and arrays. An object of that class consists of a number of subdirectories called *branches* and those branches include *leafs* that are filled with data. Usually a single branch includes variables that are related to each other and are expected to be used together - these variables are the leafs. Such an organization of branches optimizes the data for upcoming use and creates a more efficient `TTree` structure. In that DAOD file we are interested to the `CollectionTree` `TTree` object that contains the generated physics simulation data and in

particular, for the simple reference study that will be done, the branch of `CollectionTree`, `AntiKt4truthjetsAux`, within which the 4 major `TTree` leaves include the jet transverse momenta p_T, the pseudorapidity η, the azimuthial angle ϕ and the jet mass m distributions (according to the coordinate system used at the ATLAS detector). The truth term was defined in Section 4.1, while `AntiKt` is a jet-finding algorithm used in particle collisions (and simulations of such collisions) that behaves like an idealized cone algorithm whose most notable property is the resilience of the jet boundaries it provides with respect to soft radiation present in the jet cone that may disturb its shape. The issue of jet reconstruction algorithms is extensive, as they consist one of the main tools used to analyze data from collisions, and this project report is not going to explain them analytically - all that is needed to know for our work is that the used algorithm provides reliable jet identification for the PYTHIA simulations studied. For more information on this particular algorithm one can refer to [49]. Once the DAOD is loaded with ROOT we can draw any of the histograms, for example the p_T of the jets, through the command line by calling the `draw` method of the branch that includes the wanted histogram and specifying the path to it as follows:

```
CollectionTree->Draw("AntiKt4TruthJetsAux.AntiKt4TruthJetsAux.pt")
```

4.2.1.1 p_T comparison

The first quantity we want to study is the jet transverse momenta p_T. It was considered practical to actually develop the ROOT-based comparison scripts that are needed using the two DAOD files produced from the two jobs that were just processed on Iridium and compare those first. If performed correctly this comparison would confirm that the 2 jobs were generated using different seeds and the produced code can then be used for the cross-platform comparisons that are needed with the proper input files on each occasion. An example of the p_T comparison code is provided as a whole in Appendix A.2.

It can be noticed that two scripts are available, one for comparison of unweighted p_T and one for weighted p_T. The second one will be analyzed in Appendix B. The scripts provided as examples at Appendix A are the ones used later for Grid-generated and virtual machine-generated DAOD files, but their structure is identical.

For the comparison of unweighted p_T spectra, we first define a `TFile` stream that accesses the DAOD of Job 0 and also define an 1-dimensional float histogram that will store the wanted p_T distribution:

```
TFile *firidium0=TFile::Open("MC15.304874.Pythia8EvtGen_A14NNPDF23LO_jetjet
_Powerlaw.py_1_800000000.root");
TH1F *hiridium0=new TH1F("hiridium0","TruthJets_pt_unweighted_sample_iridium_job_0;
p_{T} [MeV];
N_{entries}",100,0,4000000);
```

Then a `TTree` object needs to be defined that will store the `CollectionTree` `TTree` object in the DAOD, which contains the information that is needed. From that new `TTree` the `Draw` method is called to draw the transverse momentum distribution, as previously described, and, additionally, store it in the created histogram. The Canvas on which this was drawn is then saved as a `.eps` image (a graphics file format chosen due to the graph quality and resolution it provides) shown in Figure 4.1:

```
TTree *MyTreeiridium0 = 0;
firidium0->GetObject("CollectionTree",MyTreeiridium0);
```

```
MyTreeiridium->Draw("AntiKt4TruthJetsAux.AntiKt4TruthJetsAux.pt>>hiridium0");
C1->SaveAs("unweighted_iridium_pt_job_0.eps");
```

After that, a new `TFile` stream is defined that creates a `.root` file in which the newly created histogram is stored and the file streams are then terminated:

```
TFile *foutiridium0 = new TFile("comparison_pt_iridium.root","RECREATE");
TH1F *houtgrid1 = (TH1F*)fgrid1->Get("hiridium0");
firidium0->GetList()->Write();
firidium0->Close();
foutiridium0->Close();
```

This process is repeated for the same quantity for Job 1 and the histogram seen in Figure 4.2 is plotted and stored in the `.root` file previously created.

Now that both histograms have been stored to a common `.root` file, a `TFile` stream accesses this file and 2 new histograms, in which the previously created p_T distributions are stored, are created. A third histogram that will store the comparison plot is created. Then the `Divide()` function of the `TH1` histogram class is used to store in the last histogram the bin-by-bin division of the 2 previously created p_T distributions that will provide a visual representation of the needed comparison:

```
TFile *fcompiridium = TFile::Open("comparison_pt_iridium.root","UPDATE");
TH1F *hcompiridium0 = (TH1F*)fcompiridium->Get("hgrid1");
TH1F *hcompiridium1 = (TH1F*)fcompGVM1->Get("hVM1");
TH1F *hcompGVM1 = new TH1F("hcompGVM1","Grid_VM_comparison_unweighted_pt_job_1;
p_{T} [MeV];N_{Grid}/N_{VM}",100,0,4000000);
hcompGVM1->Divide(hcompgridGVM1,hcompVMGVM1);
```

An important step in the process of this comparison is the *error assessment* in the comparison graph. To this extent, we opt for a typical error propagation analysis as no actual errors are provided to the p_T distribution (this is just a Monte Carlo simulation after all and not real data). In this analysis, the bin error in the transverse momentum distributions of the two studied jobs is taken as the square root of the bin content. Then, since what we essentially have is a bin-by-bin division of the two histograms stored in a 3rd one ($h_{comp} = \frac{h_0}{h_1}$) and through common error propagation theory, it is deduced that:

$$\sigma_{hcomp} = \sqrt{(\frac{\partial h_{comp}}{\partial h_0})^2 \cdot {\sigma_{h_0}}^2 + (\frac{\partial h_{comp}}{\partial h_1})^2 \cdot {\sigma_{h_1}}^2} = \sqrt{\frac{h_1^2\sigma_{h_0}^2 + h_0^2\sigma_{h_1}^2}{h_1^4}} = \frac{\sqrt{h_1^2 h_0 + h_0^2 h_1}}{h_1^2}$$
$$\Rightarrow \sigma_{hcomp} = \frac{\sqrt{h_0 h_1 (h_0 + h_1)}}{h_1^2} \tag{4.1}$$

where h_0 and h_1 represent the equivalent bin contents in each histogram for Job 0 and Job 1 respectively.

To code the above error analysis, first we store to an integer the total number of bins the histograms have (it was defined to be the same) using the `GetSize()` function of the `TH1` class and then two arrays are defined to store the bin contents for the two histograms compared. A third array is defined to store the produced errors in, with a nominal size equal to the number of bins:

```
Int_t *nbinsiridium = hcompiridium0->GetSize();
double *BinContentiridium0 = new double[nbinsiridium];
double *BinContentiridium1 = new double[nbinsiridium];
double *DivisionErrorsiridium = new double[nbinsiridium];
```

Through a `for` loop the first two arrays are filled with the bin content of the respective histograms for jobs 0 and 1. Then the `DivisionErrorsiridium` array is filled with the values produced by equation 4.1 but, before that, a conditional `if` expression is used to make sure that the bin content for Job 1 is non-zero, as that bin content finds itself in the denominator of the error propagation formula and possible zero values (and there are such empty bins) would cause infinities to appear and the code to break down. Thus such a check is necessary. After an optional print out of the error values to ensure that they appear legitimate, the calculated errors are incorporated in the histogram through the use of the `SetBinError()` function. The whole loop that achieves everything aforementioned is coded as follows:

```
for(int i = 0;i<nbinsiridium;i++)
        {
          BinContentiridium0[i] = hcompiridium0->GetBinContent(i);
          BinContentiridium1[i] = hcompiridium1->GetBinContent(i);
          DivisionErrorsiridium[i] = 0;
          if(BinContentiridium1[i])
          {
            DivisionErrorsiridium[i] = sqrt(BinContentiridium0[i]*
            BinContentiridium1[i]*
            (BinContentiridium0[i]+BinContentiridium1[i]))/
            (BinContentiridium1[i]*BinContentiridium1[i]);
            cout << i << " " << DivisionErrorsiridium[i] << endl;
      }
      hcompiridium->SetBinError(i,DivisionErrorsiridium[i]);
    }
```

After that the comparison graph is drawn with the error bars displayed and the canvas on which it was created is saved a `.eps` image. This comparison graph can be seen in Figure 4.3. As it can be seen, it largely confirms that the two p_T are not the same and were thus generated using a different seed. The spectra themselves are statistically identical - meaning that the distributions are the same from a statistical point of view (the same generator is used). If the seed was the exact same, the same events would be generated and the distributions of the same quantity would be identical *by construction*, not because of statistical convergence as the amount of events increases (for an infinite amount of events generated in each sample, a different seed distribution comparison would converge to 1).

For the weighted p_T analysis see Appendix B.1.

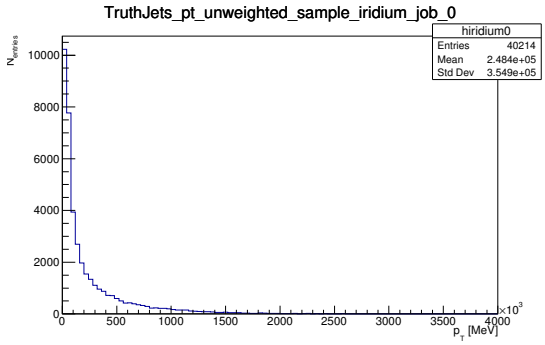

Figure 4.1: Unweighted p_T distribution for Job 0.

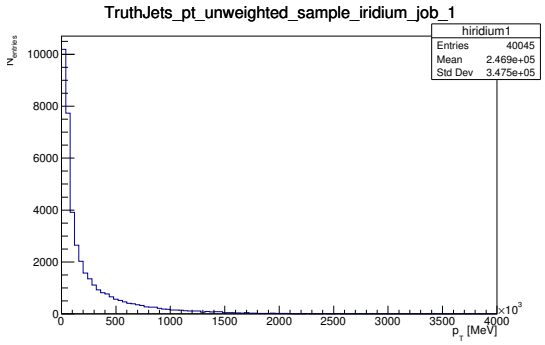

Figure 4.2: Unweighted p_T distribution for Job 1.

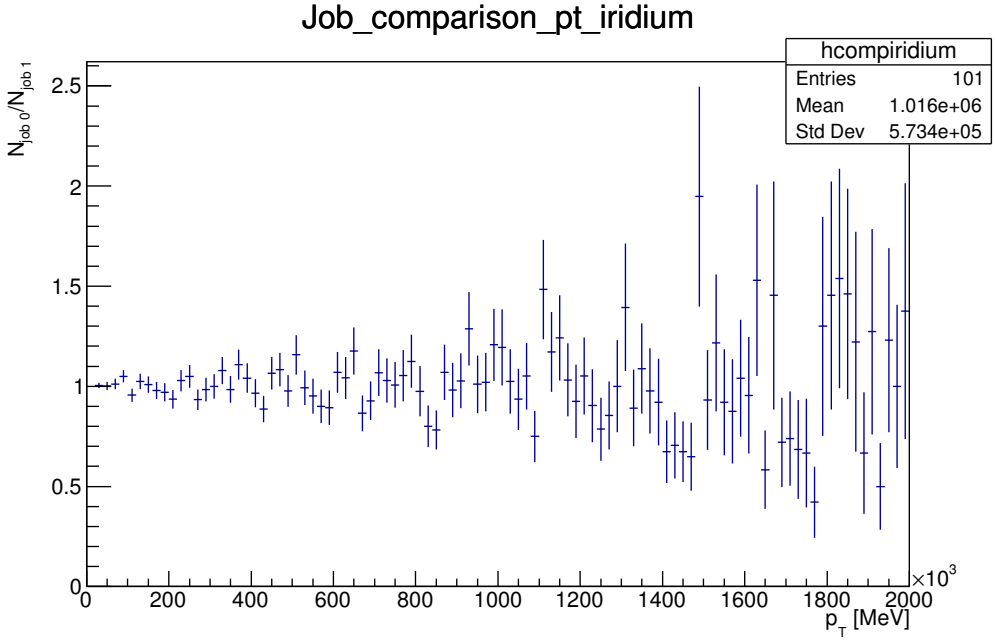

Figure 4.3: Comparison of the unweighted p_T spectra for the two jobs when processed on Iridium. It shows that the distributions are statistically identical but they were indeed generated using different seeds.

4.2.1.2 m_{jj} comparison

The next quantity to be studied is the *invariant mass distribution of jet pairs (dijets)* m_{jj}, which represents the mass of a two-parton system for partons emerging from the collision shower that later hadronize. In QCD processes, the production of high-p_T dijet events is very common - such events can reach some of the highest mass scales that can be accessed with proton-proton collisions at LHC. The invariant dijet mass we seek to calculate refers to the total invariant mass when the two most energetic jets (meaning the jets with the highest p_T) in an event (real or simulated), the *leading jet* and the *sub-leading jet*, are combined [50].

In order to calculate the invariant mass there are two ROOT classes that are essential: the `TLorentzVector` class and the `TSelector` class.

The **TLorentzVector class** is the basic class used to describe four-vectors and, in

particle physics research, such a vector can include either space and time information (x,y,z,t) or momentum and energy (or mass equivalently) information (p_x, p_y, p_z, E) or (p_x, p_y, p_z, M). In this case, it is needed to describe momentum and energy, as the analysis will be based on the principle that the square of the total four momentum of a particle or a jet or a pair of jets is equal to the square of its invariant mass ($P^2 = M^2$) and thus for a `TLorentzVector` object that contains momentum and energy information, the magnitude of that object has the meaning of invariant mass. The `TLorentzVector` class already offers the class functions `M()` and `M2()` to calculate the magnitude and squared magnitude of a given four-vector.

The values of the components of a `TLorentzVector` object can be expressed in other coordinate systems, such as the detector-specific coordinate system used in ATLAS, so that the 4-vector can be expressed in terms of transverse momentum, azimuthial angle, pseudorapidity (which is another formulation of the polar angle) and jet mass/energy (p_T, η, ϕ, m). The `TLorentzVector` class provides for that purpose a `SetPtEtaPhiM`(p_T, η, ϕ, m) function to set those coordinates and the magnitude of this 4-vector corresponds again to the invariant mass. This is the function that will be used in this invariant mass calculation, as the data that are available for the jets studied is provided in those detector-specific coordinates.

The **TSelector class** is a ROOT class specialized in the analysis of event and event-like data. A number of class member functions are derived and implemented from the `TSelector` class with some specific analysis algorithms, which are called by ROOT sequentially when the analysis is running, processing the wanted data. An analysis of a `TTree` can especially benefit from the use of a `TSelector`. ROOT already provides the `TTree: MakeSelector` function to generate a skeleton class from a given `TTree` or a specific branch of this tree. In this analysis this is exactly what is done:

```
void massiridium0()
{//creating a selector to calculate dijet mass for Job 0 processed on Iridium
    TFile *giridium0 =
    TFile::Open("MC15.304874.Pythia8EvtGen_A14NNPDF23LO_jetjet_Powerlaw.py
    _0_800000000.root");
    TTree *Riridium0 = nullptr;
    giridium0->GetObject("CollectionTree",Riridium0);
    Riridium0->MakeSelector("MySelectorIridium0","AntiKt4TruthJetsAux.");
}
```

Executing a script containing the above code block (provided as a whole for another comparison in Appendix A.3.2) creates two files : A class header `MySelectorIridium0.h` and the class body code `MySelectorIridium0.C`. The header file includes a `TTreeReader` object definition to parse through the events one by one and a number of `TTreeReaderArray` objects, which are data accessors that can iterate through collections and are each assigned one of the leafs in the branch used to generate the selector class, containing the wanted information. It also includes a number of constructors and destructors and various function definitions. The body of the class includes upon generation the definitions of 5 member functions whose functionality is well described in the generated file itself. It is briefly mentioned here that the major ones are the `Begin()` function that is called once when the selector starts to run, the `Process()` function that iterates over all events and includes the body of the work that needs to be done and the `Terminate()` function that is called once after `Process()` has iterated over all available events.

The basic idea for the calculation that is needed is to iterate over every event, store the 4 wanted coordinates (p_T, η, ϕ, m) in a 4-vector (where every coordinate will be the sum of the corresponding quantities for the leading and the sub-leading jet) and then calculate the magnitude of this 4-vector (corresponding to the invariant dijet mass for this event) before filling them in in a histogram. For the current analysis those files are enriched with some additional object definitions. In the header file the following items are added:

```
Int_t fNumberOfEventsiridium0;
Int_t dijetmassiridium0;
TLorentzVector evntiridium0;
```

(the Iridium tag is added to specify the environment they are processed on - for the same process on a different environment e.g. the Grid, the variables and functions are tagged accordingly. Same with the 0 indicator that indicates which Job is processed - in this case Job 0). The number of events will only be used for observation purposes during the generation to ensure the process run smoothly through all the events. The rest of the header file is left untouched apart from an initialization of the number of events variable to zero.

In the body of the class, a `TH1F` class histogram `invmassiridium0` is defined, before the `Begin()` function, that will store the dijet invariant mass spectrum for Job 0 and then in the `Process()` function body the following block of code is inserted:

```
fReaderiridium0.SetLocalEntry(entry);
  ++fNumberOfEventsiridium0;
  if(AntiKt4TruthJetsAux_pt.GetSize())
  {
  if(AntiKt4TruthJetsAux_pt[0] != 0)
  {
     if(AntiKt4TruthJetsAux_pt[1] != 0)
     {
           evntiridium0.SetPtEtaPhiM(
           AntiKt4TruthJetsAux_pt[0]+AntiKt4TruthJetsAux_pt[1],
           AntiKt4TruthJetsAux_eta[0]+AntiKt4TruthJetsAux_eta[1],
           AntiKt4TruthJetsAux_phi[0]+AntiKt4TruthJetsAux_phi[1],
           AntiKt4TruthJetsAux_m[0]+AntiKt4TruthJetsAux_m[1]);
           dijetmassiridium0 = evntiridium0.M();

           invmassiridium0->Fill(dijetmassiridium0);
     }
  }
  }
  return kTRUE;
}
```

First, the `TTreeReader` gets the data from the entry number `entry` in the studied tree. The `Process()` loops through all the tree entries. Then the number of entries number is incremented by 1. Its final value when the loop is completed must be the total number of events (for the jobs processed this is 10000). Then a number of conditions

must be set because such is the nature of Monte Carlo samples that not every event will include jets or it may include only one jet. So a three-step check condition is set checking if there are jets at all (with the help of the `GetSize()` function - if it returns zero there are no jets in the event and the if condition fails), if there is a leading jet and then if there is a sub-leading jet (the first element registered as array element zero (e.g. `AntiKt4TruthJetsAux_pt[0]`) in the relevant p_T array always corresponds to the leading jet while the second element registered as 1 in the same array corresponds to the sub-leading jet - the two most energetic jets in the event). If all conditions are satisfied the Lorentz vector's coordinates are set to the sums of the corresponding quantities for the leading and sub-leading jets in the event currently processed. The invariable dijet mass is then calculated with the `M()` function and the value is stored in the `dijetmassiridium0` variable which is then used to fill the histogram previously created. The code loops over until all events in the tree have been processed.

Finally, in the `Terminate()` function, we ask for the total number of events to be printed to confirm all the events have been processed and then the created histogram, with the dijet invariant mass spectrum created, is drawn and saved in a `.eps` image shown in Figure 4.4. In order to apply the selector to the chosen tree the selector is not called directly but it has to be passed to the tree which then runs it as follows:

```
CollectionTree->Process("MySelectorIridium0.C");
```

A complete version of the `TSelector` class files code (again for a Grid-VM job comparison done later but fully applicable to this process with a change to the input DAOD files and the variable names) is provided in Appendix A.3.1.

The exact same process has to be repeated for Job 1 and the generated invariant dijet mass distribution is shown in Figure 4.5.

Finally, a comparison script has to be generated that follows the structure of the file shown in Appendix A.3.3. It's structure is largely similar to that of the p_T comparison code, with the only difference being the processing of the selectors using the defined trees that store the `CollectionTree` from the DAOD files, which generate the histograms that are then saved in a newly created `.root` file. The newly created file is then opened in the last step of the process, the contained histograms are divided and the previously described error analysis, with some standard error propagation, is applied bin-by-bin. The end result is the graph shown in Figure 4.6 which is saved as a `.eps` image and showcases the bin-by-bin ratio of the dijet invariant mass spectra for the data included in the DAOD files of Job 0 **and** Job 1. Indeed, it is again confirmed that these distributions are not identical and thus they are derived from different generation seeds.

The code developed in this section both for p_T comparison and m_{jj} comparison proves to be efficient and accurate and will be used for the cross platform reference sample comparison it is needed for in Chapter 6.

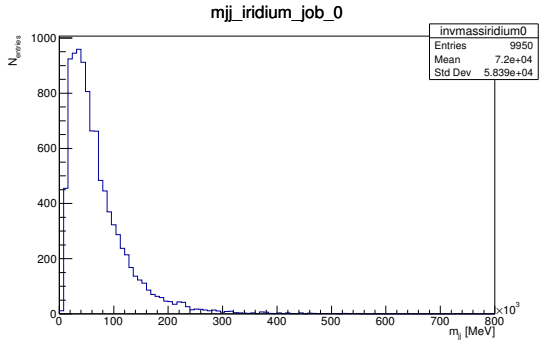

Figure 4.4: Invariant m_{jj} distribution for Job 0.

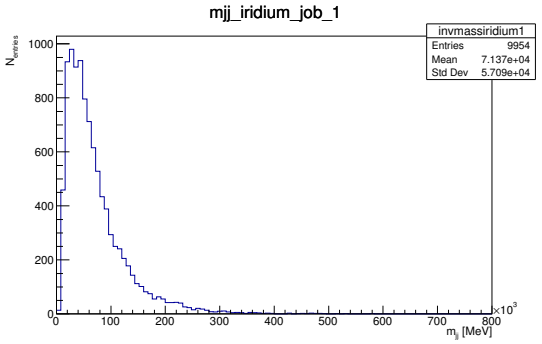

Figure 4.5: Invariant m_{jj} distribution for Job 1.

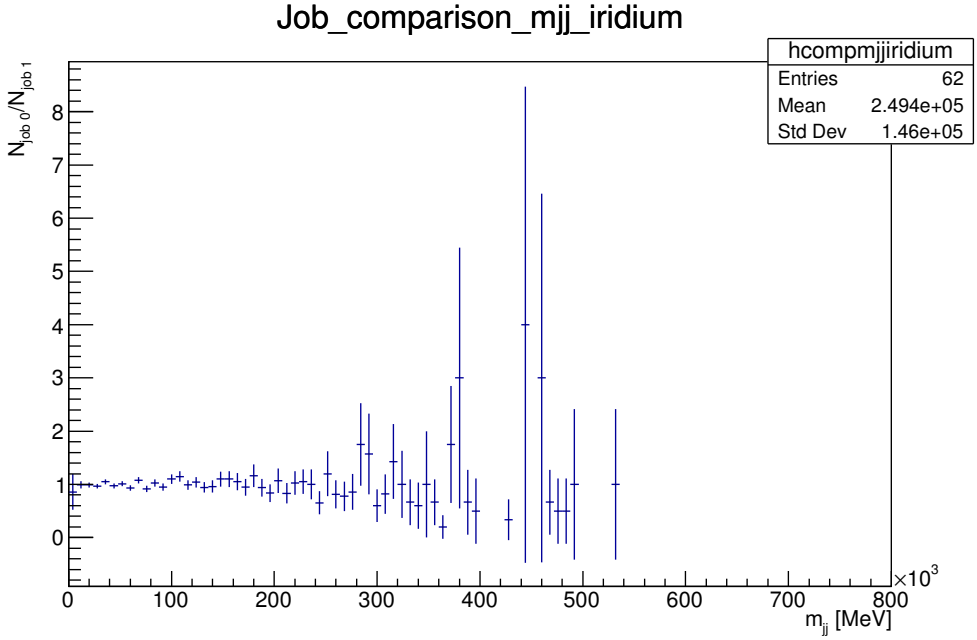

Figure 4.6: Comparison of the dijet mass m_{jj} for the two jobs when processed on Iridium. It shows that the statistical distribution is the same but the spectra compared were generated using different seeds.

4.2.2 Grid samples

In order to be able to run jobs on the Grid, some method is needed to send the relevant software and files, that include the data we want to run, over to the Grid. The main tool used to submit jobs to the Grid is called *PanDA*. For the PanDA client to be enabled on a machine, that machine must already have installed on it the ATLAS software environment. The PanDA-client package includes a number of tools, such as `pathena` (to submit Athena jobs), `prun` (to submit more general jobs), `pbook` (a bookkeeping tool for all PanDA analysis jobs) and `psequencer` (used to implement an analysis chain). A monitoring site for Grid jobs submitted through PanDA exists, called bigPanDA [51].

The tool used to download and list job outputs, part of the Grid, is called *Rucio*, another tool provided by the ATLAS software package. It is a distributed data management system that provides a convenient way to manage files on the Grid. The basic unit that

it handles is called a *DID (Data Identifier)*, which can be any storage format on the Grid, like a file, a dataset that includes a number of files, or a container that includes a number of datasets. Those DIDs are stored to certain Grid sites, but they are all registered in a certain central location. The datasets are organized using a concept called *scopes*, which are equivalent to the namespaces in C++, for example. Every user, that has a valid Grid certificate and can generate a user proxy in order to setup the Rucio client in first place, has their own scope. For example, to list all the DIDs a user has within their scope, they have to use the command `rucio list-dids "user.${USER}:*"`. This will list the DIDs included in the scope of the user who setup Rucio. In this project all the output files are stored in the author's scope `user.dsidirop` and downloaded using Rucio for further examining (`rucio download "user.dsidirop:dataset/container/file name"`).

Before we proceed, the reasons that make it a necessity to have MC generation validated on the Grid should be explained. Three basic reasons motivate that choice:

1. As it must have been understood from the description of its structure in Chapter 3.3, ATLAS@Home must operate like a regular Grid site that sees jobs submitted to it (it even uses the same job management system - PanDA), processed on a remote processing unit and the result is then returned to the BOINC server. Thus, a successful submission and generation on the Grid would function as a good indicator of the success this process may have when done on the ATLAS@Home application and server.

2. Additionally any problems that may occur either with the script submission and processing or a server malfunction can be easier to debug on a normal Grid site, which is much more easily monitored rather than ATLAS@Home.

3. Finally, such a submission, if successful, can be used for further result validation by offering another source of reference samples, produced in a different way other than on a local cluster - it offers more feedback with respect to the *reproducibility* of the results in a reliable way.

The processing of the sample scripts through the Grid proved to be a much more complicated matter than initially thought. The complications that arose were related to the demand that both processes should occur in *one submission/one task* and, if possible, without having to store the EVNT file, whose size, for the amount of events processed in one job, was close to 500 MB, on the Grid storage or locally. These restrictions were related to the way the submission of a similar job should function on ATLAS@Home. The platform should accept a *single defined task* and the user should be able to choose this task and download one work unit of it, that would have as final output just the DAOD file, starting from the running of a PYTHIA generation, using the given Job Options, to the DAOD generation, which is the wanted outcome for the researcher submitting the task. Also, the production and storage of big EVNT files on their local units could discourage volunteers from participating. The effort that is done aims to minimize the amount of data the jobs on ATLAS@Home have to transfer for input and output. If both processes can be done in a single job, it means that only the, much smaller, DAOD files need to be transferred, avoiding the uploading of EVNT files as large as 500 MB per job.

Several attempts were made to solve this issue, with some of them failing to give a proper generation of the needed output, and some others succeeding in producing the wanted files but failing to comply to one or both the aforementioned conditions that would make them suitable to run on the ATLAS@Home application. A brief description

of 3 of them is provided, for some further insight, before the final solution that solved the issue is presented.

4.2.2.1 "Piped" generation using the same pathena command

The first attempt was based on the basic idea of running multiple transformations in a single `pathena` job submission. `Pathena` is a client-tool used to submit user-defined Athena jobs to distributed analysis systems - as both the generation of the EVNT file and the derivation of the DAOD are Athena jobs, the `pathena` tool is used. The generation of the EVNT file is performed by running the PYTHIA generator, with that file then directly "piped" as input to a transformation job to produce the wanted DAOD file. An example of the coding behind such an attempt is shown fully in Appendix A.7. The PanDA submission is as follows:

```
pathena --trf "Generate_tf.py --ecmEnergy=13000 --runNumber=304874
--firstEvent=0 --maxEvents=10000 --randomSeed=996496117
--jobConfig=MC15.304874.Pythia8EvtGen_A14NNPDF23LO_jetjet_Powerlaw.py
--outputEVNTFile=event.root | grep cross-section | tee output.log;
Reco_tf.py --inputEVNTFile=event.root
--outputDAODFile=%OUT.output.root --reductionConf TRUTH3 --loglevel FATAL"
--outDS=user.dsidirop.Pythia8EvtGen.EVNTxAOD10.test.v10
```

This submission, no matter how is it formulated, leads to a failed task that does not produce the wanted DAOD files in the output (where the output dataset name is clearly defined - this is part of a PanDA submission setup). In order to understand why that happened we had to look back to the job scripts previously processed on Iridium. It is noticed that before the event generation and the `reco` (track reconstruction - transformation to DAOD) processes are run, special *projects* of the Athena framework have to be setup. Such projects cannot coexist in a shell. If one project is setup currently and we attempt to setup a second one, it will override and overwrite the first. Also, different projects use different releases of the Athena framework in order to be fully functional. In general, while the Athena repository contains all the code that can be built in an ATLAS software release, each such release consists only of a consistent subset of that code. Each such code subset is what is called *a project*. When a specific Athena project is built, the build result encodes the project name and, consequently and independent of release number, the `AtlasProduction` and `AtlasDerivation` projects (which are Athena-based ATLAS analysis-specific projects). These projects are setup through the job scripts previously studied for event generation and DAOD derivation respectively and are each built from different code. Given that before such a PanDA submission, as the one provided above, only a single project can be defined, without the option to change it in the middle of the submission, one of the processes cannot be performed because it will not have access to the needed software tools. Unfortunately, this "piped" submission command is not yet featured in any validated release of the ATLAS software - and PanDA specifically.

4.2.2.2 psequencer submission

The `psequencer` is a tool, part of the PanDA-client package. It is an advanced tool that constructs a sequence of different tasks to be submitted. A sequence is composed of multiple steps, with each step executing one or more commands. The sequence is written in a plain `.txt` file consisting of a series of steps and then an execution sequence (that

defines how steps are processed), where each step and the sequence are tagged with a triple number sign # before their name. The steps can have any name (except `Sequence`, which is reserved for sequences) but that is the name that must be used in the sequence to access it and execute it. The sequence is coded in regular Python language syntax. Its main function, which is to execute the steps, is done using the `execute()` function, and the result is retrieved using the `result()` function at the end of the sequence. The sequence file that was created for this occasion can be seen in Appendix A.6. The first step sets up an Athena release with the `AtlasProduction` project build, and then executes the event generation as previously seen in the job scripts used. Similarly, the second step sets up another Athena release with the `AtlasDerivation` project build, and then executes the DAOD derivation. The sequence executes Step 1, retrieves the result and then checks its status. If it succeeds, it should proceed to print a confirmation message of successful generation and then set a variable to which the output EVNT file of Step 1 is used as input to the execution of Step 2.

Certain variables accessed in the sequence section (`result.Failed`, `result.Out` for example) are special attributes that should be available to a step that runs a `pathena` job, as both steps in this process are running such jobs.

The sequence is executed using the command `psequencer sequencesubmission.txt`. Unfortunately, the execution of the sequencer failed to generate the wanted output, leading to an error saying it does not recognize the special `pathena` attributes mentioned above that the `psequencer` documentation clearly states are available directly for steps with `pathena` tasks. An inquiry with the Distributed Analysis Software Team at ATLAS resulted in acknowledging that `psequencer` is an obsolete feature of PanDA and certain features of it (such as the `pathena` special attributes) are not compatible with later Athena releases used for the analysis here. Thus a very promising effort had to be dropped and move on.

4.2.2.3 2-task submission

The third effort was based on the idea of creating a single script that would first submit an event generation PanDA task, then introduce a buffer with a loop that checks if the EVNT file has been created on Grid and, when it is finally generated, use it as input to submit the DAOD derivation Grid task. It was thought that such a coding scheme could overcome the issue of a needed single submission for ATLAS@Home. In order to achieve that the initial master Python script, that was described in Section 4.1, is modified to add a section that creates a `bash` script that translates into code this process. This modified script is provided as a whole in Appendix A.5. A brief explanation of the modifications made will follow.

First of all, a new submission file `pandafile=open('submitAllSubmissions_panda.sh')` is created and it's turned into a bash executable (`pandafile.write("#!/bin/bash\n")`). A `for` loop, similar to the one used for the creation of the job scripts in the initial Python script, is initiated that generates the number of job scripts the user has asked for when executing the script, and names them using a format similar to the one the initial Python script used as well. It then opens the file to write in:

```
for jobNumber in xrange(0, numberOfJobs) :
  pandascriptName =
 "MC15."+str(channelNumber)+".Pythia8EvtGen_A14NNPDF23LO_jetjet_"+
 name+"_"+str(jobNumber)+"_"+str(offset)+"_gridsubmission.sh"
```

```
pandascript = open(pandascriptName,"w")
```

The PanDA submission script is made `bash` executable and then two strings are defined that ensure the generation of a different name for the output datasets for the event generation and the DAOD derivation (identical output dataset names "confuse" PanDA and can lead to failed or overwritten tasks and outputs on the Grid) - a random number generated every time the `bash` script is executed, attached to a predefined string, ensures this difference in output dataset names:

```
pandascript.write("""#!/bin/bash


str1="user.dsidirop.Pythia8EvtGen.EVNT.v"
str2="user.dsidirop.Pythia8EvtGen.xAOD.v"
SubNum=$RANDOM
outDSEVNT=$str1$SubNum
outDSxAOD=$str2$SubNum
```

After the Job Options are copied in the current directory, where the scripts are created, the ATLAS software is setup, followed by the setup of the appropriate Athena release to run the `AtlasProduction` project. This is followed by a `pathena` submission of the event generation job.

After the submission is done a `while` loop is introduced. First, the Rucio tool is setup to be able to use its file listing features. Two empty string objects are defined that will be used to store the names of possibly generated files in the dataset of the previously generated submission, and then the following `while` loop is introduced:

```
while [ -z "$EVNTfile" -a $SECONDS -le 7200 ];
do
rucio list-files "user.dsidirop:"$outDSEVNT"_EXT0/" >> tempFile$SubNum.txt
RUCIOline="$(grep .root tempFile$SubNum.txt)"
EVNTfile="$(echo $RUCIOline| cut -d'|' -f 2)"
echo  "Stil in the loop"
sleep 5m

rm tempFile$SubNum.txt
done
```

This bash-coded loop checks with Rucio if the EVNT file has been generated on the Grid and uses the `EVNTfile` variable to store the name of that file. If the `EVNTfile` variable is not empty and thus contains a string, which means that an EVNT file has been generated, the loop breaks and the script moves on to use this EVNT file for the DAOD derivation task. If Rucio did not list an EVNT file, because it has not been generated yet, the string is empty and the loop continues until the file is generated. A 5 minute sleep interval is introduced between iterations so that there doesn't have to be a constant check as it is a processing power-consuming and memory-consuming process.

An additional condition of exiting the loop after 2 hours have passed is introduced, so that, in case the EVNT generation task does not proceed smoothly and fails, the `while` loop will not be endless. A 2 hour period was considered a rational time interval, given previous experience with event generation tasks on the Grid and how much processing time they usually need. After a successful break from the loop and the creation of the DAOD file, the job script is made executable and a running command is added to the previously generated submission file:

```
#making it executable
commands.getoutput("chmod 755 "+pandascriptName)

#running it
pandafile.write("bash " + pandascriptName + "\n")
```

Running this script indeed generates a successful result. It has the disadvantage of binding a shell that has to run the `while` loop until the EVNT file is generated and then submit the DAOD derivation job, but, in the end, on the Grid, there are indeed two successful tasks, one that generated the EVNT file and the other that generated the wanted DAOD that can be now downloaded. Unfortunately, this solution of a single script submission was not satisfactory. The dissatisfaction had to do with the fact that a major condition, the need to not store the quite heavy EVNT file, was not essentially fulfilled. If the same job was submitted to the ATLAS@Home BOINC server that distributes the jobs to the volunteers' personal computers, at the end of the job the outputs of both tasks, event generation and DAOD derivation, would still have to be uploaded to the Grid, and having to move multiple 500 MB EVNT files around (depending on the number of jobs submitted) would not be acceptable to most volunteers. Thus this effort was also dropped.

4.2.2.4 prun job script submission

The final (and successful) effort was based on the use of another tool available from the PanDA client, `prun`. The `prun` tool is used to submit more general jobs to PanDA, jobs that can run ROOT scripts, including C++-based ROOT scripts, Python scripts and `bash` executables. It is meant to support non-Athena types of analysis (for example, ROOT-based analysis, where Athena runtime is not always available), although it can work on both Athena and non-Athena runtime environments. A `prun` submitted job can include multiple sub-jobs that run in parallel on different work units, a feature we will take advantage of in this project. In this project, its use is based on the idea that the `bash` executable job scripts the initial python script generates, seem to satisfy the necessary conditions, and so all that needs to be done is to execute those bash executable scripts on the Grid. The master Python script used to generate the `bash` executables, that will then be submitted with `prun` to the Grid, is given in Appendix A.4 in full. The first difference that can be seen is a change in the base directory `baseDir` and the output directory `outputDir`:

```
baseDir = "./MCProduction"
outputDir = "./MCProduction/results"
```

That is because `prun` can "see" and gather only files within the working directory (`workDir`), which, by default, for scripts submitted to the Grid through `prun` is noted

with ./, before sending them to worker nodes. Apart from the numberofJobs variable, a second variable is setup, so its value can be set by the user upon execution, called numberofsubJobs. This variable will be used to set the total number of sub-jobs (and thus the total number of events) of the script the user wants. The Job Options generation is left unaltered. An execution file that will store the prun submission command is created:

```
execFile = open('submitAllSubmissionsprun.sh','w')
```

It is then turned into a bash executable:

```
execFile.write("#!/bin/bash\n")
```

and sets up the ATLAS environment by defining the appropriate variables in order to install the PanDA client:

```
execFile.write("export ATLAS_LOCAL_ROOT_BASE=/cvmfs/atlas.cern.ch/repo/
ATLASLocalRootBase\n")
execFile.write("source $ATLAS_LOCAL_ROOT_BASE/user/atlasLocalSetup.sh\n")
execFile.write("lsetup panda\n")
```

Then the job scripts are generated through the same for loop used in the initial master script, with the total number of generated scripts being equal to the number of jobs the user defines. In the script, the creation of the temporary library is removed and the Job Options are transferred to the working directory in order for prun to gather them.

The ATLAS software environment is then setup, before the appropriate release to run the AtlasProduction project is also setup, followed by the event generation command, with one difference: The random seed is no longer generated by the Python script that generates the bash executable job script like before, where it was defined as --randomSeed=str(offset+jobNumber) but the random number generation is *transferred to the bash executable script* using the shuf function of bash as follows:

```
--randomSeed="$(shuf -i 100000000-999999999 -n 1)"
```

This is done because, if the previous configuration was kept, all sub-jobs generated with prun would have the exact same seed and would generate the exact same events in repeat, something very counterproductive for the purposes the simulations are needed. The sub-jobs need to generate events based on a different seed per sub-job and thus the random number generation needs to be transferred to the bash executable itself, which is the file the prun tool executes. The random seed could also be generated from the prun submission command, as it provides such a feature, but this option is not pursued here, since a working solution was found in the script itself. After the event generation is complete, the appropriate Athena release for AtlasDerivation to be setup is installed and the DAOD derivation task is executed, with a command identical to that of the initial script. The execution file is then updated to include the prun command that needs to be run:

```
execFile.write("prun --outDS user.dsidirop.evgen.testv"+str(version)+"
--noBuild
--nJobs="+str(numberofsubJobs)+"
--outputs DAOD_TRUTH3."+outputJOFileName+"_"+str(jobNumber)+"_"
+str(offset)+".root
--exec \"bash "+jobscriptName+"\" \n" )
```

An executable example of the above format in a produced script would be like this:

```
prun --outDS user.dsidirop.evgen.testv68464 --noBuild --nJobs=2
--outputs DAOD_TRUTH3.MC15.304874.Pythia8EvtGen_A14NNPDF23LO_jetjet_
Powerlaw.py_0_342162485.root
--exec "bash MC15.304874.Pythia8EvtGen_A14NNPDF23LO_jetjet_Powerlaw_
0_342162485.sh"
```

This command defines the name of the output dataset, in which the output files, whose name is defined in `--outputs`, are stored upon generation. We opt to skip the `buildGen` phase, through the `--noBuild` command, as the submitted task executes a script consisting mostly of standard Athena processes (the event generation and the DAOD derivation) so nothing really has to be compiled through a build step - the build step is required when more general, personal coding schemes have to be submitted with `prun`. By omitting the building step, the process is accelerated as well, as the building step has to be processed as a different task than the actual running job. 2 sub-jobs are defined, which means that when the `bash` executable is executed with the `--exec "bash filename"` command, it will essentially be executed 2 times and, given the coding we opted for to decide the seed of the event generation, every time the `bash` executable script is executed, a different random seed will be generated for each job.

A `prun` command such as the above was executed, and the Grid submission initiated from it proved successful, solving all the present problems as the whole process ran inside a single job, with no need to store the EVNT files on the Grid. The first part of the job created the EVNT file, then, when that was generated, the second part simply read it from the same directory. That way, only the final output (12 MB DAOD) needs to be uploaded to the Grid, as it was specified in the `--output` option. This is ideal for an ATLAS@Home task submission.

Using the 2 DAOD files generated from this successful submission, a comparison of the unweighted jet p_T and the invariant m_{jj} was done (using the coding scheme explained in Section 4.2.1), to ensure that indeed the 2 sub-jobs generated DAOD files using different random seeds. The comparison graphs for unweighted transverse momentum and dijet mass spectra are shown in Figures 4.7 and 4.8. It can be seen that, while there is some early convergence in the p_T comparison and some convergence at higher values for the m_{jj} comparison, in general the distributions are different. If the seed was the same, there should be absolute agreement, as it will be seen in the next chapter's analysis.

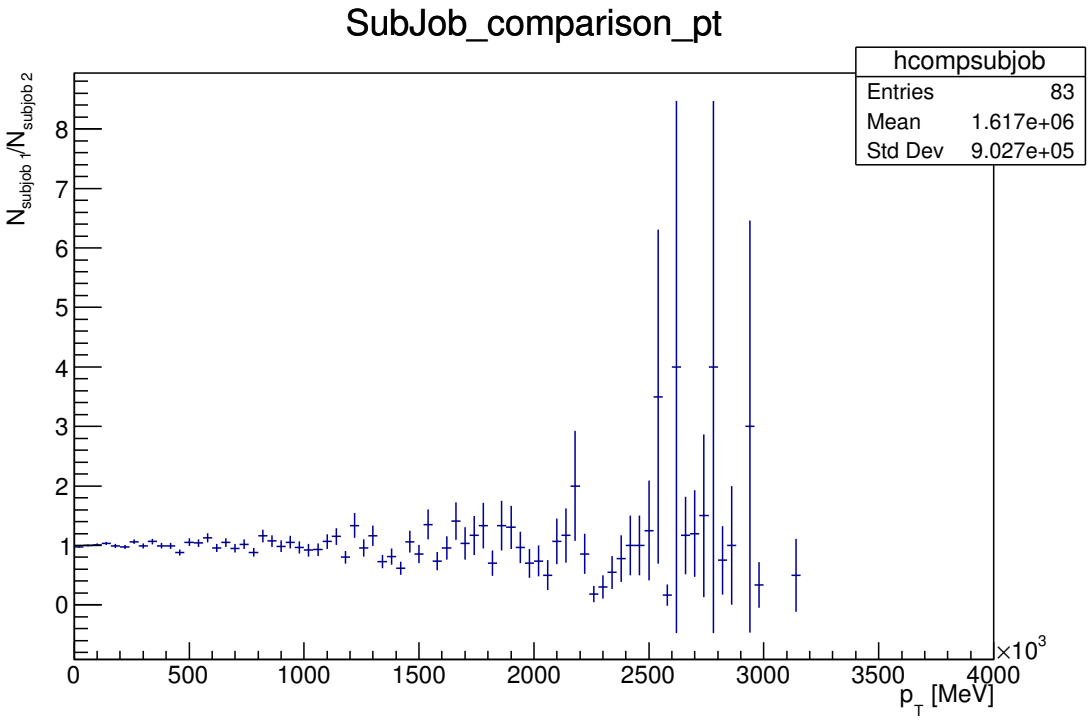

Figure 4.7: Comparison of the unweighted transverse momentum spectra from DAOD files generated from sub-jobs created on the Grid with a prun command submission. It confirms that the statistical distribution is the same but the spectra were generated using different seeds.

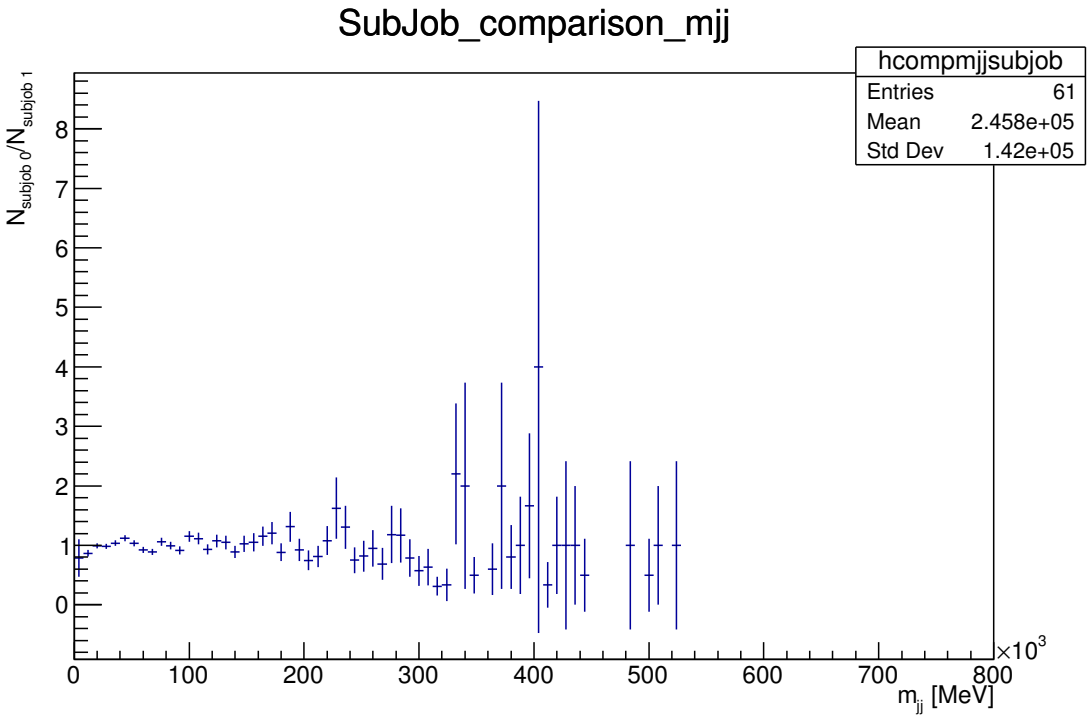

Figure 4.8: Comparison of the invariant dijet mass spectra from DAOD files generated from sub-jobs created on the Grid with a prun command submission. It confirms that the statistical distribution is the same but the spectra were generated using different seeds.

Chapter 5

ATLAS@Home virtual machine setup

In this chapter, the specifics of the virtual machine environment created on the user's machine, when a project on BOINC is chosen, will be explained in more detail. After that, a reference sample comparison is made and a task submission to an ATLAS@Home test server and application is attempted in Chapter 6.

The virtual machine environment has been produced using Oracle VirtualBox for the purposes of this project, in order to attempt to run the test scripts on it. Should the generation occur successfully, and produce the wanted DAOD files, then we have acquired a strong indication that similar processes should be able to run successfully on a volunteer's machine that picks a work unit through BOINC, since the VM environment BOINC sets up in the background will be the same.

It must be made clear that to achieve this goal the virtual machine environment has been accessed *directly*, not through the application but by downloading the relevant virtual machine image - a *CernVM* [52] release - and set it up in the same way it is setup for ATLAS@Home, taking advantage of a process called *contextualization*, with the scripts transferred to that specially setup VM environment from the Iridium cluster.

But what is a CernVM ? It is nothing more that a virtual machine, intended on running on any hardware and host operating system (Linux, Windows etc), in order to provide a consistent and effortless installation of the needed experimental software. The currently used version, CernVM 3.6, contains a Scientific Linux 6 operating system, and consists of an image of approximately 20 MB, that contains the bare essentials: the Linux kernel and a *CVMFS client*, which downloads and caches on demand the rest of the operating system and any needed software for a particular analysis. Such an approach creates a *highly configurable virtual software appliance*, to be used in common by all major LHC experiments, allowing the customization of this virtual environment with the wanted software releases in an efficient way.

5.1 Contextualization

According to the official CernVM page [53] a context is *"a small (up to 16kB), usually human-readable snippet that is used to apply a role to a virtual machine."* It is what allows a single virtual machine image to be highly versatile, by allowing it to back a variety of virtual machine instances, as long as those instances can adapt to the various infrastructures and use cases that depend on the context. Through the contextualization process, instance-specific settings, in the form of a context made available from an infrastructure, can be injected and applied dynamically into the virtual machine during the deployment

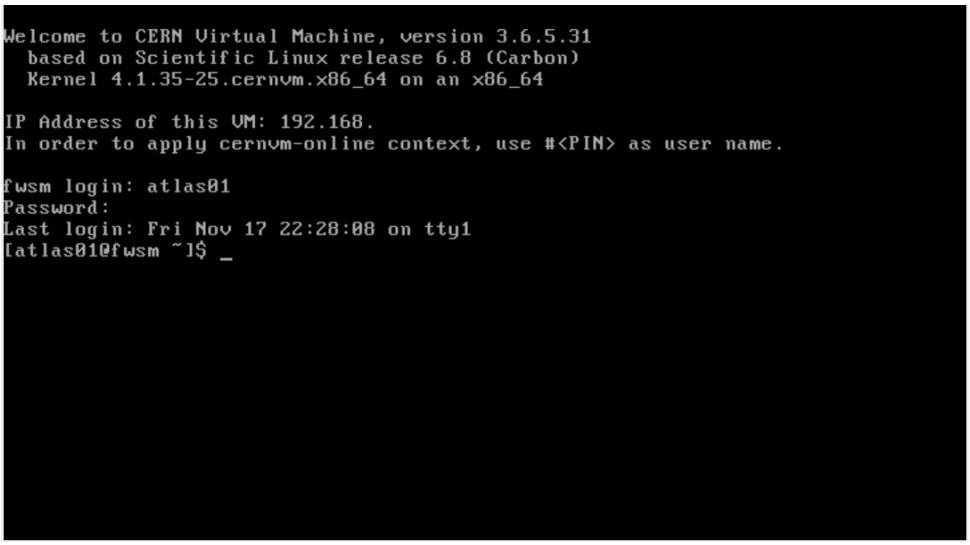

Figure 5.1: A screenshot of the CernVM environment used, paired with the ATLAS@Home context (the VM IP address has been partially erased as a precaution).

phase (the boot process of the VM) and the virtual machine interprets that context. Such settings included in a context may correspond to certain services, user creation, network configurations (to make sure each created instance is assigned a unique-IP address) or parameter configuration.

During the contextualization process, the data needs to be separated into *meta-data*, which is available through the infrastructure that provides the VM instance and are non-modifiable by the user (e.g. instance IP issued by cloud infrastructure and the VM's assigned network parameters), and *user-data*, which is provided by the user of the virtual machine upon the creation of that machine. Both types of data can be accessed through an HTTP server using a private, unique IP address.

This contextualization mechanism is what is used in the case of CernVM images as a method to decrease the size of those distributed images. It applies the settings of the chosen context on a generic CernVM image on first boot. The site that provides all the relevant CernVM resources also provides an easy way for the users to actually *create* the context they want to apply to their virtual machine image themselves [54]. The user can specify the various customization options like the name of the context,who it will be visible to, add users to the created system, set up services that will load on boot, add environmental variables and other CernVM specific configuration options. More importantly, they can configure the CVMFS, the tool that delivers the software needed, to specify the software groups whose software the user wants to use in this VM instance (e.g. `atlas`, `grid`, `cms` etc).

Apart from creating their own context, users can choose from a collection of pre-made contexts, usually from research groups using CernVM environments, through a marketplace [55]. One of them is the ATLAS@Home project that offers a *public context* in the marketplace, ready to use for ATLAS@Home virtual machine images. This is the context that will be used in this project as well.

First of all, the latest versions of the VirtualBox virtualization software is downloaded (at the time of the download that would be version 5.1.22) and then the CernVM ISO image file (version 3.6.5 at the time it was downloaded) for VirtualBox is downloaded from the CernVM on line web page and manually imported into VirtualBox. During the process

of configuration of the ISO image file in VirtualBox, a virtual machine template, to which the image will be attached, must be created using the relevant tools the virtualization software provides. By opting to create a new virtual machine in VirtualBox, various configuration options for this machine can be decided, such as the RAM memory allocation for the VM, along with the hard drive partition and the amount of actual hard drive space that will be dedicated to the CernVM image. In this particular instance, 2048 MB of memory are allocated to the virtual machine, while a VDI (VirtualDisk Image) virtual hard drive is created with a size of 20 GB that will be dynamically allocated (meaning that the virtual hard drive file will grow as it is used and filled up, and will not be created on a fixed size equal to the 20 GB chosen). The created virtual machine will appear in the VirtualBox interface. Finally, the ISO image file should be stored in the newly created virtual machine through the relevant "Storage" tab at the virtual machine's settings.

The basic CernVM template is now theoretically ready to be launched, but, as it was mentioned before, this image file contains the bare essentials of an operating system - it needs to be *paired* to a context, the *ATLAS@Home public context* available at the marketplace. Pairing a context to a CernVM requires minimal effort as the process has been simplified to the point where the introduction of a simple PIN code is enough to contextualize a CernVM. When the wanted ATLAS@Home context is chosen from the marketplace, a pair option appears. If chosen, it prompts the generation of a pin code that the user is prompted to enter to the virtual machine - the log-in screen of the virtual machine gives the relevant instructions that can be seen at Figure 5.1, which provides the log-in environment of the CernVM used in this project: "In order to apply cernvm-online context, user #<PIN> as username". All the user has to do is type the pin code they were given on the Cern VM online webpage and the ATLAS@Home context is paired to the CernVM image they have setup. What we were able to achieve is to *create a virtual machine environment identical to the one a BOINC user downloads to their computer once they choose to process an ATLAS@Home simulation.* The main difference is we have full control over this environment. Scripts can be loaded in this environment and processed. If they are processed successfully and produce DAOD files that are identical to those produced on other environments (analyzed in Chapter 4), we have a good indication that such Monte Carlo scripts, generating large numbers of events, can be processed on ATLAS@Home. This is the next step that will be taken in the following chapter.

Chapter 6

Results

6.1 Comparison with Reference Samples

In Chapter 4.2.2, one of the two basic conditions, in order to have a successful generation of large Monte Carlo samples on ATLAS@Home, was met - the ability to perform the generation on the Grid in a singular task, without storing the heavy EVNT files. Now the second condition will be tested (using the coding scheme developed and explained in Chapter 4.2.1) and that is the *reproducibility of results on the ATLAS@Home properly contextualized virtual machine*. That will be tested through the comparison of characteristic quantities and their distributions in DAOD files produced on the virtual machine, through the execution of the scripts analyzed in Chapter 4, to the same quantities and for the same generation seeds from DAOD files generated on the Iridium cluster and the Grid with the methods thoroughly explained in Sections 4.2.1 and 4.2.2.

First, we transfer the master Python script, described in Section 4.1, on the virtual machine environment from Iridium. The only change that will be made concerns the method of submission for the `bash` executables. This virtual machine lacks the features Iridium had that allowed the parallel submission of multiple tasks on different nodes of a cluster through the batch interface, using the `sbatch` command. Thus any `sbatch` command is substituted by a simple `bash` execution command. This is featured already in the master Python script in Appendix A.1, in the section where the execution commands are written in the `submitAllSubmissions_8.sh` file:

```
submissionfile.write("sbatch "+jobscriptName+" \n") #Iridium
#submissionfile.write("bash "+jobscriptName+" \n") #generic
```

The only other thing that must be taken care of is to ensure that the same generation seeds, as those used for the 2 reference samples in the Iridium analysis section, are used here, so the offset value has to be set to the value used in Section 4.2.1. Every other aspect of the script is left unchanged in order to ensure ideal reproducibility. Then the ATLAS software environment has to be setup through the set of variable defining and source commands:

```
export ATLAS_LOCAL_ROOT_BASE=/cvmfs/atlas.cern.ch/repo/ATLASLocalRootBase
source ${ATLAS_LOCAL_ROOT_BASE}/user/atlasLocalSetup.sh
```

The setup is successful. The Python script is run through a generic `python file.py` submission and 2 job scripts are produced with the same event generation seeds as the

samples on Iridium. The bash executables are executed as previously described and, after a period of time (longer than the one that had to pass on Iridium as the scripts could not be processed in parallel), the DAOD files are generated. Since Iridium offers a graphic interface, that this virtual machine does not have, it is easier to transfer those DAOD files and write the code needed to perform the needed comparisons on the Iridium cluster, both for the Iridium-VM and the Grid-VM comparisons.

6.1.1 Iridium - Virtual Machine comparison

The DAOD files for Job 0 and Job 1 from Iridium and the VM are brought in a common directory. Using the appropriate versions of the scripts in A.2 and A.3, with the appropriate DAOD files as input, the comparison of unweighted jet transverse momentum spectra and invariant dijet mass spectra between the DAOD files produced from Job 0 on Iridium and the ATLAS@Home virtual machine (and then between those for Job 1 as well) is performed. The results of comparisons of the distributions of those characteristic quantities for the DAOD files produced from Job 0 can be seen in Figures 6.1 and 6.2 (the actual spectra for both Jobs generated on both environments are provided in appendices C.1 and C.2). Overall an ideal reproducibility is observed. Apart from some minor divergences in both spectra, due to standard bin migration occurrences that are statistically insignificant, the rations are largely equal to unity. This means the distributions are identical as expected. For completion, the set of Iridium-VM comparison graphs for Job 1 is provided in Figures 6.5 and 6.6, while the weighted jet p_T comparison is provided in appendix B.2. In order to further ensure the reproducibility, a comparison between the VM results and those produced on the Grid is also made in the next section.

6.1.2 Grid - Virtual Machine Comparison

First there is a need to generate DAOD files on the Grid with the same generation seed as those produced on the VM and Iridium. The `prun` submission method will be used as this was the successful one that satisfied all needed Grid generation criteria analyzed in Chapter 4.2.2. Thus the script provided in Appendix A.4 must be run but this time we have to take a step back and remove the random seed generation from the `bash` script - it will be replaced in each case by the standard values used to process the reference samples in the other platforms (in this case these numbers are 800000000 and 800000001). Performing two `prun` submissions of two `bash` executables with the above generation seeds leads to two successful Grid tasks that produce the wanted DAOD files. The files are downloaded using Rucio and, following the same comparison process as with the Iridium-VM comparisons, the comparison graphs for unweighted jet p_T spectra and invariant m_{jj} spectra are produced and shown in Figures 6.3 and 6.4 for Job 0 (the actual spectra for both Jobs generated on both environments are provided in appendices C.2 and C.3). Those comparison graphs do not even suffer from the minor bin migration issues that were observed in the Iridium-VM comparisons of the previous section. An assumption that can be made to explain this difference is that this occurs due to the fact that Iridium has AMD processors while the Grid and the PC simulation (VM) both use Intel processors. The reproducibility of results is ideal for both compared quantities. For completion, the set of Grid-VM comparison graphs for Job 1 is provided in Figures 6.7 and 6.8, while a comparison of the weighted p_T spectra is provided in appendix B.3.

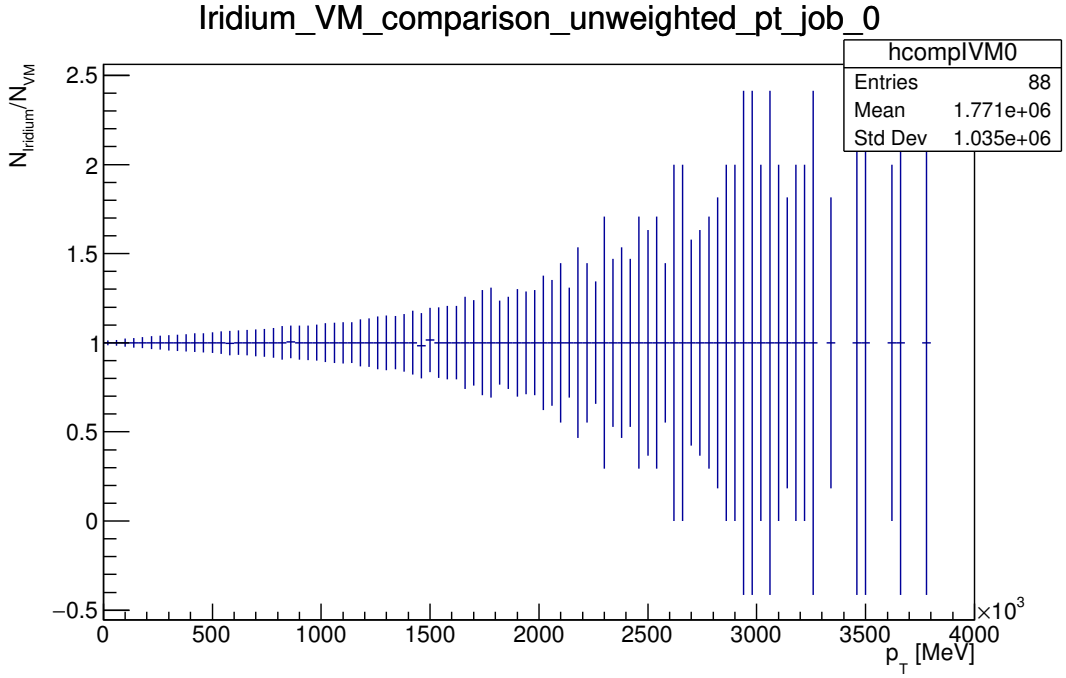

Figure 6.1: Comparison of the jet transverse momentum p_T distribution for the same Job 0 (same seed) when processed on Iridium and the VM environment for unweighted data. The comparison shows that the generation is independent of the environment in which it was developed, being identical for the same seed, with the exception of minor bin migration occurrences.

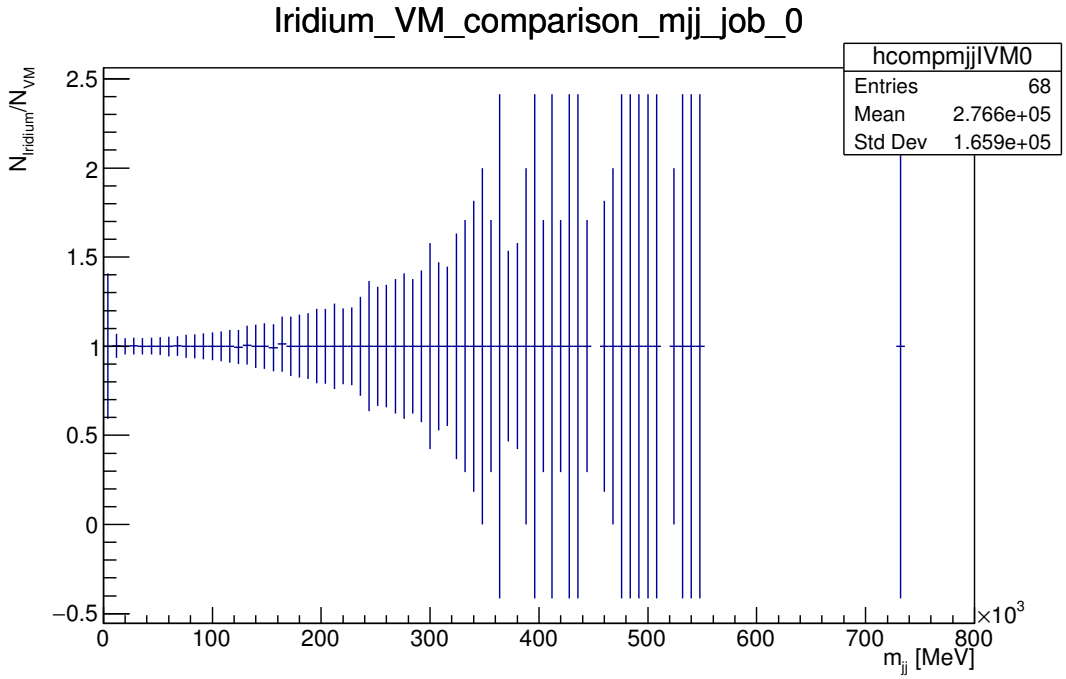

Figure 6.2: Comparison of the invariant dijet mass m_{jj} for the same Job 0 (same seed) when processed on Iridium and the VM environment. The comparison shows that the generation is independent of the environment in which it was developed, being identical for the same seed, with the exception of minor bin migration occurrences.

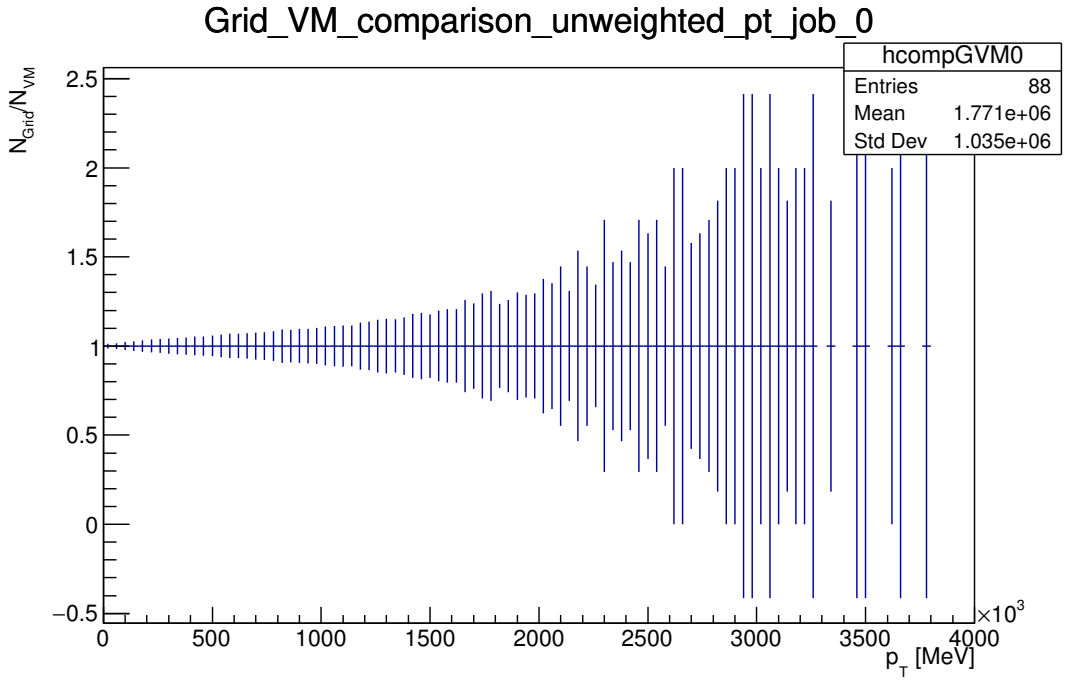

Figure 6.3: Comparison of the jet transverse momentum p_T distribution for the same Job 0 (same seed) when processed on the Grid and the VM environment for unweighted data. The comparison shows that the generation is independent of the environment in which it was developed, being identical for the same seed.

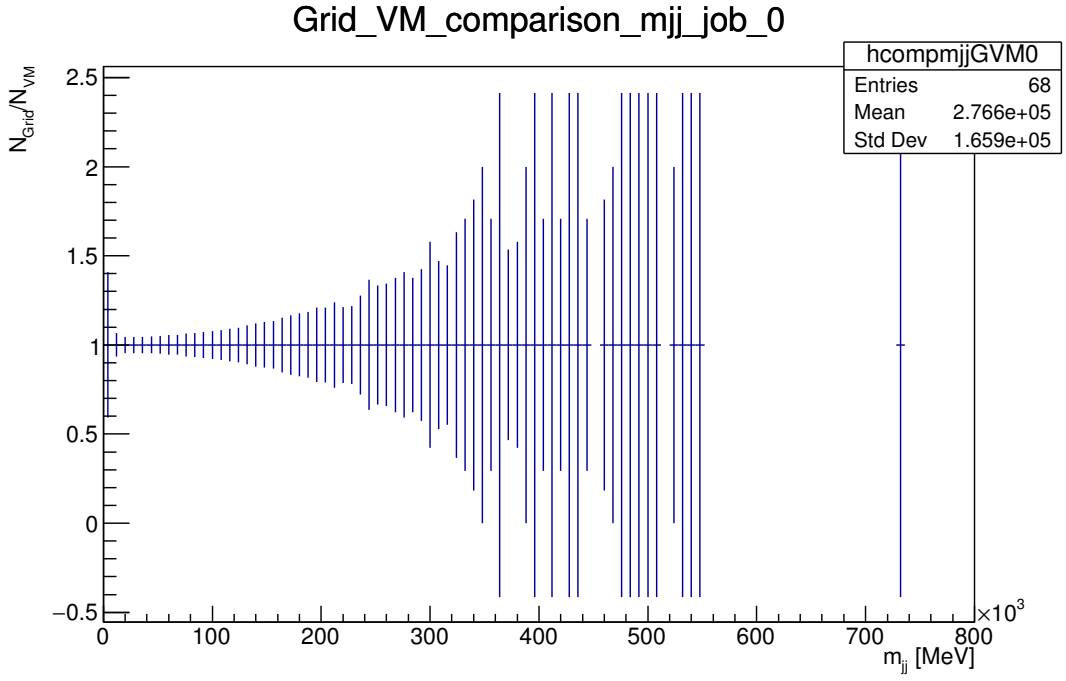

Figure 6.4: Comparison of the dijet mass m_{jj} for the same Job 0 (same seed) when processed on the Grid and the VM environment. The comparison shows that the generation is independent of the environment in which it was developed, being identical for the same seed.

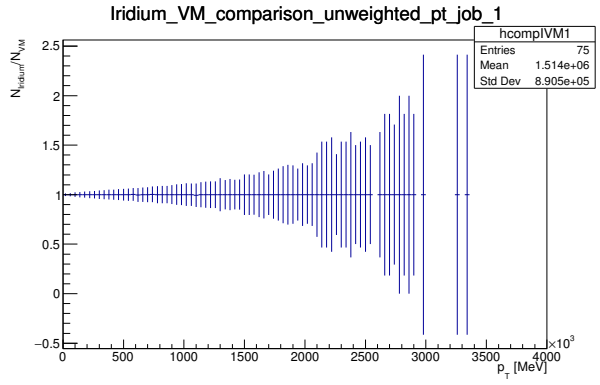

Figure 6.5: Comparison of the jet transverse momentum p_T distribution for the same Job 1 (same seed) when processed on Iridium and the VM environment for unweighted data. The comparison shows that the generation is independent of the environment in which it was developed, being identical for the same seed, with the exception of minor bin migration occurrences.

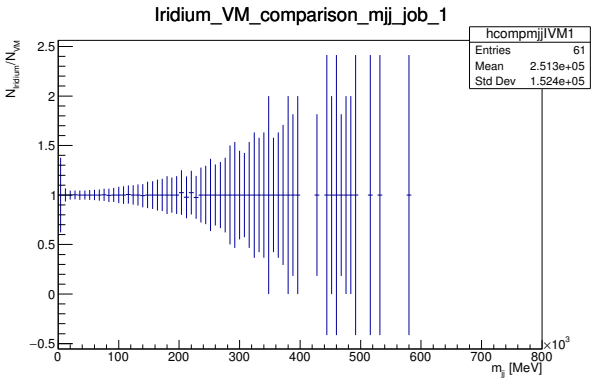

Figure 6.6: Comparison of the dijet mass m_{jj} for the same Job 1 (same seed) when processed on Iridium and the VM environment. The comparison shows that the generation is independent of the environment in which it was developed, being identical for the same seed, with the exception of minor bin migration occurrences.

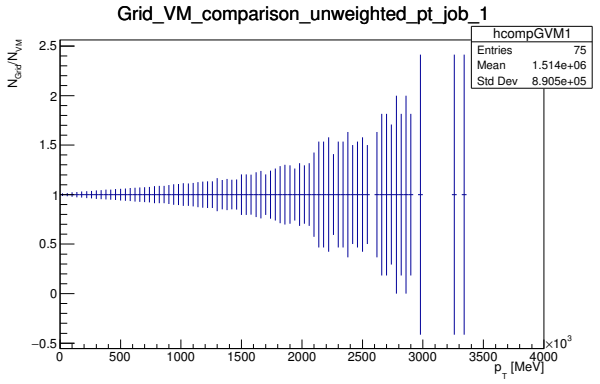

Figure 6.7: Comparison of the jet transverse momentum p_T distribution for the same Job 1 (same seed) when processed on the Grid and the VM environment for unweighted data. The comparison shows that the generation is independent of the environment in which it was developed, being identical for the same seed.

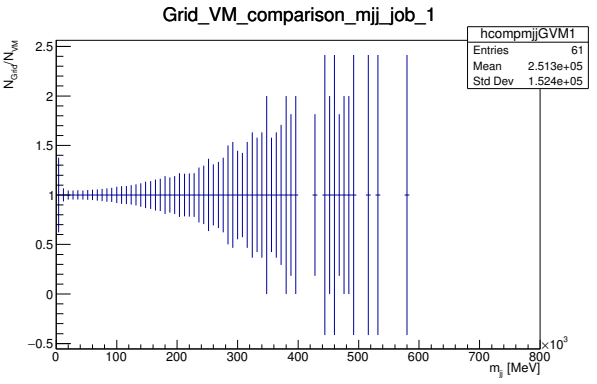

Figure 6.8: Comparison of the invariant dijet mass m_{jj} for the same Job 1 (same seed) when processed on the Grid and the VM environment. The comparison shows that the generation is independent of the environment in which it was developed, being identical for the same seed.

6.2 Submission to the ATLAS@Home server

Both conditions that needed to be fulfilled to have a task run successfully on ATLAS@Home (at least on a theoretical level) have been fulfilled. A singular Grid submission with no EVNT file generation has been made and results have shown they can be reliably reproduced on the virtual machine the users' personal computers will be downloading when they click on a task in the BOINC manager, as the comparison with samples produced on reliable sites (a local cluster and the Grid) has shown. At this point it must be reminded that ATLAS@Home is just another Grid site, as seen by PanDA, and thus a submission will be done using PanDA to send the tasks to a specific "site" (`ANALY_BOINC`), connected to a test BOINC server. This server has the jobs submitted to it directed to an application (`ATLAS Event Generation v0.01 (vbox64)`) especially setup to test those Monte Carlo generation tasks. This application is part of a test project [56] that is separate from the "real" ATLAS@Home project where new kinds of tasks and submissions are tested [57]. The test BOINC server has been connected to a number of desktop PC units where the task are processed through the ATLAS Event Generation application. The `prun` tool provides the ability to choose a specific site for submission, as well as the memory allocation, and so the `prun` submission command, in the `prun` Python master script A.4, is modified as follows in order to submit to this test server:

```
execFile.write("prun --outDS user.dsidirop.evgen.testv"+str(version)+"
--noBuild --nJobs="+str(numberofsubJobs)+"
--outputs DAOD_TRUTH3."+outputJOFileName+"_"+str(jobNumber)+"_"
+str(offset)+".root
--exec \"bash "+jobscriptName+"\" --site ANALY_BOINC --memory 1024  \n" )
```

The maximum memory size a job can use on the Grid - and in this case in the virtual machine it will be distributed to through the Grid, is set through the `--memory` option. The value is set empirically - a task of the kind we want to submit requires about 450 MB of memory (a quantity that can be checked on the task page on the bigPanDA monitoring website) and, in order to leave some extra memory space for the operating system it operates, the memory of the VM is set to 1 GB (counted in MB so 1024 MB = 1 GB).

We begin with the submission of one sub-job and it processes successfully on the desktop unit connected to the test BOINC server. Then the number of sub-jobs is increased to 5 and eventually to 100, generating in total 1 million simulated events (link for the 100 job task on PanDA : https://bigpanda.cern.ch/task/12698576/ [cited 20171203]). All of them process successfully and generate the wanted number of DAOD files with an average size of about 13 MB each, which is acceptable for volunteer personal computing units that may want to download and process such tasks even with a larger number of subtasks. In the near future a description of the task will be provided, so that it can be brought to the main ATLAS@Home project, past the test phase, where the description will be displayed, and volunteers from around the world can choose to process large Monte Carlo generation tasks by running this application.

Chapter 7

Summary, conclusions and outlook

During this project the possibility of taking advantage of the volunteer computing concept in order to make up for the computational power deficiencies the CERN Grid starts to show, to process large Monte Carlo samples generations, has been investigated. In particular, the project was focused on helping members of the ATLAS collaboration who perform beyond the Standard Model research, specifically looking for Dark Matter candidate particles such as WIMPs, to acquire large amounts of Monte Carlo simulated events that will provide the necessary robust background on which data can be fitted, in order to look for evidence of the existence of such hypothesized particles that interact only through the weak force with luminous, normal matter. The ATLAS@Home platform was considered an ideal candidate to provide the needed computational power to deliver the large amounts of Monte Carlo simulated events needed in the search for exotic particles, with its considerable computational power coming from personal computers that volunteers from around the globe have contributed. Some reference scripts, that generate Monte Carlo simulations, have been provided to test the possibility of launching such generations on ATLAS@Home. The viability of such a task was tested on the basis of two conditions:

1. ***Reproducibility*** of the results of Monte Carlo generation across a number of environments, like the Grid and a local cluster used for such task processing, but most importantly on the virtual machine environment utilized by ATLAS@Home. In such a VM environment, each task is processed when the user chooses to process it on their personal computer through the BOINC software.

2. ***Creation of a singular Grid submission*** that performs both the event generation job and the derivation of the file format currently used for data analysis, that includes only physics objects, the DAOD. At the same time it must be ensured that the memory consuming EVNT files produced by the generation job do not need to be transferred back to the application server.

Through trial and error, especially for the singular submission creation demand, both these conditions were met, which led to the successful submission of Monte Carlo generation tasks to a test project, running parallel to ATLAS@Home that is used to probe into the viability of such new kinds of tasks for the main platform. **This is the first time ever that MC generation has been performed at ATLAS@Home**.

Since the submitted jobs succeeded in the test stage, the future is promising. Large, demanding Monte Carlo generation tasks will soon be processed on ATLAS@Home. The next steps in this process include the assignment of a description to these tasks that provides basic information on what they actually do, so volunteers can inform themselves and

decide if they want to contribute to such tasks. Promoting the platform as a valid alternative for large Monte Carlo simulation production to more ATLAS affiliated researchers, who may have difficulties ensuring sufficient Grid space to process their tasks, is the next goal, once such applications are already up and running on the platform. This significant increase in computing resources will hopefully entice growing parts of the ATLAS community to participate. More Monte Carlo simulations will lead to a higher accuracy of the ATLAS results and that has the possibility to expand our knowledge of physics possibly beyond the Standard Model as it stands today and lead to possible discoveries of new phenomena.

With such an innovative project, open to computing power contributions from everyone around the globe, the possibilities seem endless.

Appendices

Appendix A

Code Appendix

A.1 Master Python Script

```
#!/bin/python
import commands
import random
import os
import sys

cwd = os.getcwd()
baseDir = cwd+"/MCProduction"
outputDir = cwd+"/MCProduction/results"
random.seed(None)
centerOfMass = "13000"
channelNumber = "304874"
#offset = 800000000
offset = random.randint(100000000,999999999)


#numberOfJobs =2
numberOfJobs = int(sys.argv[1])
numberOfEvents = "10000"


Names = [

"Powerlaw",

]

TunesImport = [

"Pythia8_A14_NNPDF23LO_EvtGen_Common.py",

]

ExtraLines = [

""" """,
]

if not os.path.exists(baseDir+"/jobOpt"):
          os.makedirs(baseDir+"/jobOpt")


#print zip(VectorOf_M_ExQ, ChannelNumbers)

submissionfile = open('submitAllSubmissions_8.sh','w')
submissionfile.write("#!/bin/bash\n")
submissionfile.write("#SBATCH --job-name='MCProdCaterina8'\n")

for name, (tuneImport, extraLines) in zip(Names, zip(TunesImport, ExtraLines)) :
```

```
    #let's write the JO
    outputJOFileName = "MC15."+str(channelNumber)+".Pythia8EvtGen_A14NNPDF23LO_jetjet_"
    +name+".py"
    #outputJOFile = open("jobOpt/"+outputJOFileName,"w")
    outputJOFile = open(baseDir+"/jobOpt/"+outputJOFileName,"w")

    #Mass Scale parameter (Lambda, in GeV)

    outputJOFile.write("""

# author: Caterina Doglioni

evgenConfig.description= "Dijet truth jets, with power law"
evgenConfig.keywords= ["QCD", "jets", "SM"]
evgenConfig.generators += ["Pythia8"]
evgenConfig.contact = ["Harinder Singh Bawa <bawa@cern.ch>"]

include("MC15JobOptions/"""+tuneImport+""""")

""")


    outputJOFile.write("""

genSeq.Pythia8.Commands += ["HardQCD:all = on",
        "PhaseSpace:pTHatMin = 20.0",
        "PhaseSpace:bias2Selection = on",
        "PhaseSpace:bias2SelectionPow = 5.0"

#    """+extraLines+"""
]

evgenConfig.minevents = 1000
    """)


    #parallelisation starts now!

    for jobNumber in xrange(0, numberOfJobs) :
      #now that we have the JO, let's write a script to run it
      jobscriptName = "MC15."+str(channelNumber)+".Pythia8EvtGen_A14NNPDF23LO_jetjet_"+name+"_"+str(jobNumber)+
      "_"+str(offset)+".sh"
      jobscript = open(jobscriptName,"w")

      jobscript.write("""#!/bin/bash
# Simple submission script for MC15 validation

# for tracking when and where you run
hostname
date

#to avoid NFS problems
#sleep $(perl -e '$ran=int(rand(120)); print STDOUT "$ran\\n";')

# to make and use a temporary directory
tmpDir=`mktemp -d`
cd ${tmpDir}
echo 'we are in '${PWD}

# Setup ROOT and various libraries
#cp /atlas/users/doglioni/authentication.xml .
#export CORAL_AUTH_PATH=.:${CORAL_AUTH_PATH}
#export PYTHON_EGG_CACHE=/atlas/users/doglioni/.python-eggs

#copy everything we need into this temp directory
if [ ! -d  """+baseDir+"""/jobOpt ]; then
mkdir -p """+baseDir+"""/jobOpt
fi

cp """+baseDir+"""/jobOpt/"""+outputJOFileName+""" .

#module load enableATLAS
#setupATLAS
```

```
#asetup 19.2.4.16,AtlasProduction
source $ATLAS_LOCAL_ROOT_BASE/user/atlasLocalSetup.sh
source $AtlasSetup/scripts/asetup.sh 19.2.4.16,AtlasProduction
cd ${tmpDir}

#run the job transform
Generate_tf.py --ecmEnergy="""+centerOfMass+""" --runNumber="""+str(channelNumber)+"""  --firstEvent=0
--maxEvents="""+numberOfEvents+"""--randomSeed="""+str(jobNumber+offset)+""" --jobConfig="""+outputJOFileName+"""
--outputEVNTFile=event.root | grep cross-section | tee output.log
#evgenJobOpts=MC15JobOpts-00-09-16_v1.tar.gz

#copy the output
#if [ ! -d  """+baseDir+"""/OutputNtuples/SplitQCD_EVNT ]; then
#mkdir -p """+baseDir+"""/OutputNtuples/SplitQCD_EVNT
#fi

#cp event.root """+baseDir+"""/OutputNtuples/SplitQCD_EVNT/"""+outputJOFileName+"""_"""+str(jobNumber)+"""_"""+
str(offset)+"""EVNT.root

source $ATLAS_LOCAL_ROOT_BASE/user/atlasLocalSetup.sh
#asetup 20.1.8.3,AtlasDerivation
source $AtlasSetup/scripts/asetup.sh 20.1.8.3,AtlasDerivation
cd ${tmpDir}


#run the ntuplemaker
Reco_tf.py --inputEVNTFile=event.root --outputDAODFile=output.root --reductionConf TRUTH3  --loglevel FATAL

#copy the output
if [ ! -d  """+outputDir+"""/OutputNtuples/SplitQCD ]; then
mkdir -p """+outputDir+"""/OutputNtuples/SplitQCD
fi

cp DAOD_TRUTH3.output.root """+outputDir+"""/OutputNtuples/SplitQCD/"""+outputJOFileName+"""_"""+
str(jobNumber)+"""_"""+str(offset)+""".root
cp output.log """+outputDir+"""/OutputNtuples/SplitQCD/"""+outputJOFileName+
"""_"""+str(jobNumber)+"""_"""+str(offset)+""".log
ls
rm *
"""
  )

      #making it executable

      commands.getoutput("chmod 755 "+jobscriptName)

      #running it
      submissionfile.write("sbatch "+jobscriptName+" \n") #Iridium
      #submissionfile.write("bash "+jobscriptName+" \n") #generic
```

A.2 p_T plot comparison code

A.2.1 Unweighted p_T

```
#include <iostream>
#include <TFile.h>
#include <TH1.h>
#include <TCanvas.h>
#include <TTree.h>
#include <TStyle.h>

using namespace std;

void masterGVM1();

void masterGVM1()
{
    TCanvas *C1 = new TCanvas("C1","C1",900,600);
        //Drawing and storing histogram from Grid job
        TFile *fgrid1 = TFile::Open("MC15.304874.Pythia8EvtGen_A14NNPDF23LO_jetjet_Powerlaw.py_1_800000000grid.root");
```

```
        TH1F *hgrid1 = new TH1F("hgrid1","TruthJets_pt_unweighted_sample_grid_job_1;
    p_{T}[MeV];N_{entries}",100,0,4000000);
        TTree *MyTreegrid1 = 0;
        fgrid1->GetObject("CollectionTree",MyTreegrid1);
        MyTreegrid1->Draw("AntiKt4TruthJetsAux.AntiKt4TruthJetsAux.pt>>hgrid1");
        C1->SaveAs("unweighted_grid_pt_job_1.eps");
        TFile *foutgrid1 = new TFile("comparison_GVM_pt_job_1.root","RECREATE");
        TH1F *houtgrid1 = (TH1F*)fgrid1->Get("hgrid1");
        fgrid1->GetList()->Write();
        fgrid1->Close();
        foutgrid1->Close();


        //Drawing and storing histogram from virtual machine job
        TFile *fVM1 =TFile::Open("MC15.304874.Pythia8EvtGen_A14NNPDF23LO_jetjet_Powerlaw.py_1_800000000vm.root");
        TH1F *hVM1 = new TH1F("hVM1","TruthJets_pt_unweighted_sample_VM_job_1;
    p_{T} [MeV];N_{entries}",100,0,4000000);
        TTree *MyTreeVM1 = 0;
        fVM1->GetObject("CollectionTree",MyTreeVM1);
        MyTreeVM1->Draw("AntiKt4TruthJetsAux.AntiKt4TruthJetsAux.pt>>hVM1");
        C1->SaveAs("unweighted_VM_pt_job_1.eps");
        TFile *foutVM1 = TFile::Open("comparison_GVM_pt_job_1.root","UPDATE");
        TH1F *houtVM1 = (TH1F*)fVM1->Get("hVM1");
        fVM1->GetList()->Write();
        fVM1->Close();
        foutVM1->Close();

    TCanvas *CcompGVM1 = new TCanvas("CcompGVM1","CcompGVM1",900,600);

        //comparing the 2 histograms stored in the new file
        TFile *fcompGVM1 = TFile::Open("comparison_GVM_pt_job_1.root","UPDATE");
        TH1F *hcompgridGVM1 = (TH1F*)fcompGVM1->Get("hgrid1");
        TH1F *hcompVMGVM1 = (TH1F*)fcompGVM1->Get("hVM1");
        TH1F *hcompGVM1 = new TH1F("hcompGVM1","Grid_VM_comparison_unweighted_pt_job_1;
    p_{T} [MeV];N_{Grid}\N_{VM}",100,0,4000000);
        hcompGVM1->Divide(hcompgridGVM1,hcompVMGVM1);

        // Error progation for histogram division
        Int_t nbinsGVM1 = hcompGVM1->GetSize();
        double *BinContentgridGVM1 = new double[nbinsGVM1];
        double *BinContentVMGVM1 = new double[nbinsGVM1];
        double *DivisionErrorsGVM1 = new double[nbinsGVM1];
        for(int i = 0;i<nbinsGVM1;i++)
        {
          BinContentgridGVM1[i] = hcompgridGVM1->GetBinContent(i);
          BinContentVMGVM1[i] = hcompVMGVM1->GetBinContent(i);
          DivisionErrorsGVM1[i] = 0;
          if(BinContentVMGVM1[i])
          {
            DivisionErrorsGVM1[i] = sqrt(BinContentgridGVM1[i]* BinContentVMGVM1[i]*

        (BinContentgridGVM1[i]+BinContentVMGVM1[i]))/(BinContentVMGVM1[i]*BinContentVMGVM1[i]);
            cout << i << " " << DivisionErrorsGVM1[i] << endl;
      }
      hcompGVM1->SetBinError(i,DivisionErrorsGVM1[i]);
    }
        hcompGVM1->Draw("E");
        fcompGVM1->Write();
        CcompGVM1->SaveAs("comparison_GVM_pt_job_1.eps");
        fcompGVM1->Close();
}
```

A.2.2 Weighted p_T

```
#include <iostream>
#include <TFile.h>
#include <TH1.h>
#include <TCanvas.h>
#include <TTree.h>
#include <TStyle.h>

using namespace std;
```

```
void weightsGVM1();

void weightsGVM1()
{
  // Storing the weighted pT histogram created in the grid for Job 1 to a root file
  TFile *fgrid1 = TFile::Open("MC15.304874.Pythia8EvtGen_A14NNPDF23LO_jetjet_Powerlaw.py_1_800000000grid.root");
  TTree *grid1 = 0;
  fgrid1->GetObject("CollectionTree",grid1);
  TH1F *hweightsgrid1 = new TH1F("hweightsgrid1","TruthJets_pt_weighted_sample_grid_job_1;
  p_{T} [MeV];N_{entries}",100,0,4000000);
  TCanvas *c1 = new TCanvas("c1","c1",900,600);
  gPad->SetLogy();
  grid1->Draw("AntiKt4TruthJetsAux.AntiKt4TruthJetsAux.pt>>hweightsgrid1","McEventInfo.m_event_type.m_mc_event_weights");
  c1->SaveAs("weighted_grid_pt_job_1.eps");
  TFile *foutgrid1 = new TFile("comparison_WGVM_pt_job_1.root","RECREATE");
  TH1F *houtgrid1 = (TH1F*)fgrid1->Get("hweightsgrid1");
  fgrid1->GetList()->Write();
  fgrid1->Close();
  foutgrid1->Close();

  // Storing the weighted pT histogram created on the VM for Job 1 to a root file
  TFile *fVM1 = TFile::Open("MC15.304874.Pythia8EvtGen_A14NNPDF23LO_jetjet_Powerlaw.py_1_800000000vm.root");
  TTree *VM1 = 0;
  fVM1->GetObject("CollectionTree",VM1);
  TH1F *hweightsVM1 = new TH1F("hweightsVM1","TruthJets_pt_weighted_sample_VM_job_1;
  p_{T} [MeV];N_{entries}",100,0,4000000);
  gPad->SetLogy();
  VM1->Draw("AntiKt4TruthJetsAux.AntiKt4TruthJetsAux.pt>>hweightsVM1","McEventInfo.m_event_type.m_mc_event_weights");
  c1->SaveAs("weighted_VM_pt_job_1.eps");
  TFile *foutVM1 = TFile::Open("comparison_WGVM_pt_job_1.root","UPDATE");
  TH1F *houtVM1 = (TH1F*)fVM1->Get("hweightsVM1");
  fVM1->GetList()->Write();
  fVM1->Close();
  foutVM1->Close();

  TCanvas *CcompWGVM1 = new TCanvas("CcompWGVM1","CcompWGVM1",900,600);

  //comparing the 2 histograms stored in the new file
  TFile *fcompWGVM1 = TFile::Open("comparison_WGVM_pt_job_1.root","UPDATE");
  TH1F *hcompgridWGVM1 = (TH1F*)fcompWGVM1->Get("hweightsgrid1");
  TH1F *hcompVMWGVM1 = (TH1F*)fcompWGVM1->Get("hweightsVM1");
  TH1F *hcompWGVM1 = new TH1F("hcompWGVM1","Grid_VM_comparison_pt_weighted_job_1;
  p_{T} [MeV];N_{Grid}/N_{VM}",100,0,4000000);
  hcompWGVM1->Divide(hcompgridWGVM1,hcompVMWGVM1);
  hcompWGVM1->Draw("E");
  fcompWGVM1->Write();
  CcompWGVM1->SaveAs("comparison_WGVM_pt_job_1.eps");
  fcompWGVM1->Close();

}
```

A.3 m_{jj} plot comparison code

A.3.1 TSelector code

A.3.1.1 MySelectorGrid1.h (header file)

```
//////////////////////////////////////////////////////////
// This class has been automatically generated on
// Wed Oct 25 03:33:02 2017 by ROOT version 6.10/04
// from TTree CollectionTree/CollectionTree
// found on file: MC15.304874.Pythia8EvtGen_A14NNPDF23LO_jetjet_Powerlaw.py_1_800000000grid.root
//////////////////////////////////////////////////////////

#ifndef MySelectorGrid1_h
#define MySelectorGrid1_h

#include <TROOT.h>
```

```
#include <TChain.h>
#include <TFile.h>
#include <TSelector.h>
#include <TTreeReader.h>
#include <TTreeReaderValue.h>
#include <TTreeReaderArray.h>
#include <TLorentzVector.h>

// Headers needed by this particular selector
#include <vector>

#include <set>



class MySelectorGrid1 : public TSelector {
public :
   TTreeReader     fReadergrid1;  //!the tree reader
   TTree          *fChain = 0;   //!pointer to the analyzed TTree or TChain

   // Readers to access the data (delete the ones you do not need).
   TTreeReaderArray<float> AntiKt4TruthJetsAux_pt = {fReadergrid1, "AntiKt4TruthJetsAux.pt"};
   TTreeReaderArray<float> AntiKt4TruthJetsAux_eta = {fReadergrid1, "AntiKt4TruthJetsAux.eta"};
   TTreeReaderArray<float> AntiKt4TruthJetsAux_phi = {fReadergrid1, "AntiKt4TruthJetsAux.phi"};
   TTreeReaderArray<float> AntiKt4TruthJetsAux_m = {fReadergrid1, "AntiKt4TruthJetsAux.m"};
   //TTreeReaderArray<vector<ElementLink<DataVector<xAOD::IParticle> > >> AntiKt4TruthJetsAux_constituentLinks =
   {fReadergrid1, "AntiKt4TruthJetsAux.constituentLinks"};
   //TTreeReaderArray<vector<float>> AntiKt4TruthJetsAux_constituentWeights =
   {fReadergrid1, "AntiKt4TruthJetsAux.constituentWeights"};
   Int_t fNumberOfEventsgrid1;
   Int_t dijetmassgrid1;
   TLorentzVector evntgrid1;


   MySelectorGrid1(TTree * /*tree*/ =0):
   //evntgrid1.SetPtEtaPhiE(0.,0.,0.,0.),
   fNumberOfEventsgrid1(0){}
   virtual ~MySelectorGrid1() { }
   virtual Int_t   Version() const { return 2; }
   virtual void    Begin(TTree *tree);
   virtual void    SlaveBegin(TTree *tree);
   virtual void    Init(TTree *tree);
   virtual Bool_t  Notify();
   virtual Bool_t  Process(Long64_t entry);
   virtual Int_t   GetEntry(Long64_t entry, Int_t getall = 0)
   { return fChain ? fChain->GetTree()->GetEntry(entry, getall) : 0; }
   virtual void    SetOption(const char *option) { fOption = option; }
   virtual void    SetObject(TObject *obj) { fObject = obj; }
   virtual void    SetInputList(TList *input) { fInput = input; }
   virtual TList  *GetOutputList() const { return fOutput; }
   virtual void    SlaveTerminate();
   virtual void    Terminate();

   ClassDef(MySelectorGrid1,0);

};

#endif

#ifdef MySelectorGrid1_cxx
void MySelectorGrid1::Init(TTree *tree)
{
   // The Init() function is called when the selector needs to initialize
   // a new tree or chain. Typically here the reader is initialized.
   // It is normally not necessary to make changes to the generated
   // code, but the routine can be extended by the user if needed.
   // Init() will be called many times when running on PROOF
   // (once per file to be processed).

   fReadergrid1.SetTree(tree);
}

Bool_t MySelectorGrid1::Notify()
```

```
{
   // The Notify() function is called when a new file is opened. This
   // can be either for a new TTree in a TChain or when when a new TTree
   // is started when using PROOF. It is normally not necessary to make changes
   // to the generated code, but the routine can be extended by the
   // user if needed. The return value is currently not used.

   return kTRUE;
}


#endif // #ifdef MySelectorGrid1_cxx
```

A.3.1.2 MySelectorGrid1.C

```
#define MySelectorGrid1_cxx
// The class definition in MySelectorGrid1.h has been generated automatically
// by the ROOT utility TTree::MakeSelector(). This class is derived
// from the ROOT class TSelector. For more information on the TSelector
// framework see $ROOTSYS/README/README.SELECTOR or the ROOT User Manual.


// The following methods are defined in this file:
//    Begin():        called every time a loop on the tree starts,
//                    a convenient place to create your histograms.
//    SlaveBegin():   called after Begin(), when on PROOF called only on the
//                    slave servers.
//    Process():      called for each event, in this function you decide what
//                    to read and fill your histograms.
//    SlaveTerminate: called at the end of the loop on the tree, when on PROOF
//                    called only on the slave servers.
//    Terminate():    called at the end of the loop on the tree,
//                    a convenient place to draw/fit your histograms.
//
// To use this file, try the following session on your Tree T:
//
// root> T->Process("MySelectorGrid1.C")
// root> T->Process("MySelectorGrid1.C","some options")
// root> T->Process("MySelectorGrid1.C+")
//


#include "MySelectorGrid1.h"
#include <TH2.h>
#include <TH1.h>
#include <TStyle.h>

TH1F *invmassgrid1 = new TH1F("invmassgrid1","mjj_grid_job_1;m_{jj} [MeV];N_{entries}",100,0,800000);

void MySelectorGrid1::Begin(TTree * /*tree*/)
{
   // The Begin() function is called at the start of the query.
   // When running with PROOF Begin() is only called on the client.
   // The tree argument is deprecated (on PROOF 0 is passed).

   TString option = GetOption();
}

void MySelectorGrid1::SlaveBegin(TTree * /*tree*/)
{
   // The SlaveBegin() function is called after the Begin() function.
   // When running with PROOF SlaveBegin() is called on each slave server.
   // The tree argument is deprecated (on PROOF 0 is passed).

   TString option = GetOption();

}

Bool_t MySelectorGrid1::Process(Long64_t entry)
{
   // The Process() function is called for each entry in the tree (or possibly
   // keyed object in the case of PROOF) to be processed. The entry argument
```

```
   // specifies which entry in the currently loaded tree is to be processed.
   // When processing keyed objects with PROOF, the object is already loaded
   // and is available via the fObject pointer.
   //
   // This function should contain the \"body\" of the analysis. It can contain
   // simple or elaborate selection criteria, run algorithms on the data
   // of the event and typically fill histograms.
   //
   // The processing can be stopped by calling Abort().
   //
   // Use fStatus to set the return value of TTree::Process().
   //
   // The return value is currently not used.

   fReadergrid1.SetLocalEntry(entry);
   ++fNumberOfEventsgrid1;
   if(AntiKt4TruthJetsAux_pt.GetSize())
   {
           if(AntiKt4TruthJetsAux_pt[0] != 0)
           {
              if(AntiKt4TruthJetsAux_pt[1] != 0)
              {
            evntgrid1.SetPtEtaPhiM(AntiKt4TruthJetsAux_pt[0]+AntiKt4TruthJetsAux_pt[1],
            AntiKt4TruthJetsAux_eta[0]+AntiKt4TruthJetsAux_eta[1],
            AntiKt4TruthJetsAux_phi[0]+AntiKt4TruthJetsAux_phi[1],
            AntiKt4TruthJetsAux_m[0]+AntiKt4TruthJetsAux_m[1]);
            dijetmassgrid1 = evntgrid1.M();

            invmassgrid1->Fill(dijetmassgrid1);
              }
           }
   }
   return kTRUE;
}

void MySelectorGrid1::SlaveTerminate()
{
   // The SlaveTerminate() function is called after all entries or objects
   // have been processed. When running with PROOF SlaveTerminate() is called
   // on each slave server.

}

void MySelectorGrid1::Terminate()
{
   // The Terminate() function is the last function to be called during
   // a query. It always runs on the client, it can be used to present
   // the results graphically or save the results to file.
   printf("\nTotal Number of Events: %d\n",fNumberOfEventsgrid1);
   //TFile *foutgrid1 = TFile::Open("dijet_mass_comparison_job_1_grid.root","RECREATE");
   TCanvas *Cgrid1 = new TCanvas("C1grid1","C1grid1",900,600);
   invmassgrid1->Draw();



   Cgrid1->SaveAs("inv_mass_job_1_grid.eps");
   //foutgrid1->Close();

}
```

A.3.2 TSelector for dijet mass calculation creation

```
#include <iostream>
#include <TFile.h>
#include <TH1.h>
#include <TCanvas.h>
#include <TTree.h>
#include <TStyle.h>
#include <TTreeReader.h>
#include <TLorentzVector.h>
using namespace std;
```

```
void massgrid1();

void massgrid1()
{   //creating a selector to calculate dijet mass for Job 1 processed on the Grid
    TFile *ggrid1 = TFile::Open("MC15.304874.Pythia8EvtGen_A14NNPDF23LO_jetjet_Powerlaw.py_1_800000000grid.root");
        TTree *Rgrid1 = nullptr;
        //TLorentzVector *evntgrid1 = new TLorentzVector;
        ggrid1->GetObject("CollectionTree",Rgrid1);
        Rgrid1->MakeSelector("MySelectorGrid1","AntiKt4TruthJetsAux.");
}
```

A.3.3 Dijet mass m_{jj} comparison script

```
#include <iostream>
#include <TFile.h>
#include <TH1.h>
#include <TCanvas.h>
#include <TTree.h>
#include <TStyle.h>
#include <TTreeReader.h>
#include <TLorentzVector.h>
using namespace std;

void dijet_compGVM_1();

void dijet_compGVM_1()
{
  //getting the dijet mass histogram for the grid processed DAOD files
  TFile *ggrid1 = TFile::Open("MC15.304874.Pythia8EvtGen_A14NNPDF23LO_jetjet_Powerlaw.py_1_800000000grid.root");
  TTree *Rgrid1 = 0;
  ggrid1->GetObject("CollectionTree",Rgrid1);
  Rgrid1->Process("MySelectorGrid1.C");
  TFile *goutgrid1 = new TFile("dijet_mass_comp_GVM_job_1.root","RECREATE");
  TH1F *houtgrid1 = (TH1F*)ggrid1->Get("invmassgrid1");
  ggrid1->GetList()->Write();
  ggrid1->Close();
  goutgrid1->Close();

  //getting the dijet mass histogram for the VM processed DAOD files
  TFile *gVM1 = TFile::Open("MC15.304874.Pythia8EvtGen_A14NNPDF23LO_jetjet_Powerlaw.py_1_800000000vm.root");
  TTree *RVM1 = 0;
  gVM1->GetObject("CollectionTree",RVM1);
  RVM1->Process("MySelectorVM1.C");
  TFile *goutVM1 = new TFile("dijet_mass_comp_GVM_job_1.root","UPDATE");
  TH1F *houtVM1 = (TH1F*)gVM1->Get("invmassVM1");
  gVM1->GetList()->Write();
  gVM1->Close();
  goutVM1->Close();

  TCanvas *CcompmjjGVM1 = new TCanvas("CcompmjjGVM1","CcompmjjGVM1",900,600);

  //comparing the 2 histograms stored in the new file
  TFile *gcompmjjGVM1 = TFile::Open("dijet_mass_comp_GVM_job_1.root","UPDATE");
  TH1F *hcompmjjgridGVM1 = (TH1F*)gcompmjjGVM1->Get("invmassgrid1");
  TH1F *hcompmjjVMGVM1 = (TH1F*)gcompmjjGVM1->Get("invmassVM1");
  TH1F *hcompmjjGVM1 = new TH1F("hcompmjjGVM1","Grid_VM_comparison_mjj_job_1;m_{jj} [MeV];N_{Grid}/N_{VM}",100,0,800000);
  hcompmjjGVM1->Divide(hcompmjjgridGVM1,hcompmjjVMGVM1);

  // Error propagation for histogram division
  Int_t nbinsmjjGVM1 = hcompmjjGVM1->GetSize();
  double *BinContentmjjgridGVM1 = new double[nbinsmjjGVM1];
  double *BinContentmjjVMGVM1 = new double[nbinsmjjGVM1];
  double *DivisionErrorsmjjGVM1 = new double[nbinsmjjGVM1];
  for(int i = 0;i<nbinsmjjGVM1;i++)
  {
        BinContentmjjgridGVM1[i] = hcompmjjgridGVM1->GetBinContent(i);
    BinContentmjjVMGVM1[i] = hcompmjjVMGVM1->GetBinContent(i);
        DivisionErrorsmjjGVM1[i] = 0;
        if(BinContentmjjVMGVM1[i])
        {
          DivisionErrorsmjjGVM1[i] = sqrt(BinContentmjjgridGVM1[i]* BinContentmjjVMGVM1[i]*
      (BinContentmjjgridGVM1[i]+BinContentmjjVMGVM1[i]))/(BinContentmjjVMGVM1[i]*BinContentmjjVMGVM1[i]);
```

```
          cout << i << " " << DivisionErrorsmjjGVM1[i] << endl;
    }
    hcompmjjGVM1->SetBinError(i,DivisionErrorsmjjGVM1[i]);
  }
  hcompmjjGVM1->Draw("E");
  gcompmjjGVM1->Write();
  CcompmjjGVM1->SaveAs("comparison_mjj_GVM_job_1.eps");
  gcompmjjGVM1->Close();
}
```

A.4 Grid submission using prun python Master Code

```
#!/bin/python
import commands
import random
import os
import sys

cwd = os.getcwd()
baseDir = "./MCProduction"
outputDir = "./MCProduction/results"
random.seed(None)
centerOfMass = "13000"
channelNumber = "304874"
#offset = 800000000
offset = random.randint(100000000,999999999)
version = random.randint(0,99999)

#numberOfJobs =2
numberOfJobs = int(sys.argv[1])
numberOfEvents = "10000"
numberofsubJobs = int(sys.argv[2])


Names = [

"Powerlaw",

]

TunesImport = [

"Pythia8_A14_NNPDF23LO_EvtGen_Common.py",

]

ExtraLines = [

""" """,
]

if not os.path.exists(baseDir+"/jobOpt"):
          os.makedirs(baseDir+"/jobOpt")


#print zip(VectorOf_M_ExQ, ChannelNumbers)

submissionfile = open('submitAllSubmissions_8.sh','w')
submissionfile.write("#!/bin/bash\n")
submissionfile.write("#SBATCH --job-name='MCProdCaterina8'\n")
execFile = open('submitAllSubmissionsprun.sh','w')
execFile.write("#!/bin/bash\n")
execFile.write("export ATLAS_LOCAL_ROOT_BASE=/cvmfs/atlas.cern.ch/repo/ATLASLocalRootBase\n")
execFile.write("source $ATLAS_LOCAL_ROOT_BASE/user/atlasLocalSetup.sh\n")
execFile.write("lsetup panda\n")

for name, (tuneImport, extraLines) in zip(Names, zip(TunesImport, ExtraLines)) :

    #let's write the JO
    outputJOFileName = "MC15."+str(channelNumber)+".Pythia8EvtGen_A14NNPDF23LO_jetjet_"+name+".py"
```

```
    #outputJOFile = open("jobOpt/"+outputJOFileName,"w")
    outputJOFile = open(baseDir+"/jobOpt/"+outputJOFileName,"w")


    #Mass Scale parameter (Lambda, in GeV)

    outputJOFile.write("""

# author: Caterina Doglioni

evgenConfig.description= "Dijet truth jets, with power law"
evgenConfig.keywords= ["QCD", "jets", "SM"]
evgenConfig.generators += ["Pythia8"]
evgenConfig.contact = ["Harinder Singh Bawa <bawa@cern.ch>"]

include("MC15JobOptions/"""+tuneImport+""""")

""")


    outputJOFile.write("""

genSeq.Pythia8.Commands += ["HardQCD:all = on",
        "PhaseSpace:pTHatMin = 20.0",
        "PhaseSpace:bias2Selection = on",
        "PhaseSpace:bias2SelectionPow = 5.0"

#    """+extraLines+"""
]

evgenConfig.minevents = 1000
    """)

    #parallelisation starts now!

    for jobNumber in xrange(0, numberOfJobs) :
      #now that we have the JO, let's write a script to run it
      jobscriptName = "MC15."+str(channelNumber)+".Pythia8EvtGen_A14NNPDF23LO_jetjet_"+name+"_"
      +str(jobNumber)+"_"+str(offset)+".sh"
      jobscript = open(jobscriptName,"w")

      jobscript.write("""#!/bin/bash
# Simple submission script for MC15 validation

# for tracking when and where you run
hostname
date

# Setup ROOT and various libraries
#cp /atlas/users/doglioni/authentication.xml .
#export CORAL_AUTH_PATH=.:${CORAL_AUTH_PATH}
#export PYTHON_EGG_CACHE=/atlas/users/doglioni/.python-eggs

#copy everything we need into the working directory
if [ ! -d  """+baseDir+"""/jobOpt ]; then
mkdir -p """+baseDir+"""/jobOpt
fi

cp """+baseDir+"""/jobOpt/"""+outputJOFileName+""" .

source $ATLAS_LOCAL_ROOT_BASE/user/atlasLocalSetup.sh
source $AtlasSetup/scripts/asetup.sh 19.2.4.16,AtlasProduction

#run the job transform
Generate_tf.py --ecmEnergy="""+centerOfMass+""" --runNumber="""+str(channelNumber)+"""  --firstEvent=0
--maxEvents="""+numberOfEvents+""" --randomSeed="$(shuf -i 100000000-999999999 -n 1)"
--jobConfig="""+outputJOFileName+""" --outputEVNTFile=event.root | grep cross-section | tee output.log
#evgenJobOpts=MC15JobOpts-00-09-16_v1.tar.gz

source $ATLAS_LOCAL_ROOT_BASE/user/atlasLocalSetup.sh
source $AtlasSetup/scripts/asetup.sh 20.1.8.3,AtlasDerivation
```

```
#run the ntuplemaker
Reco_tf.py --inputEVNTFile=event.root --outputDAODFile="""+outputJOFileName+"""_"""+str(jobNumber)+"""_"""+
str(offset)+""".root --reductionConf TRUTH3  --loglevel FATAL

ls

"""
  )

      #making it executable

      commands.getoutput("chmod 755 "+jobscriptName)

      #running it
      submissionfile.write("bash "+jobscriptName+" \n")
      execFile.write("prun --outDS user.dsidirop.evgen.testv"+str(version+jobNumber)+" --noBuild
      --nJobs="+str(numberofsubJobs)+"
      --outputs DAOD_TRUTH3."+outputJOFileName+"_"+str(jobNumber)+"_"+str(offset)+".root
      --exec \"bash "+jobscriptName+"\" \n" )
```

A.5 2-task submission python Master Code (dropped effort)

```
#!/bin/python
import commands
import random
import os
import sys
import shutil

cwd = os.getcwd()
baseDir = cwd+"/MCProduction"
outputDir = cwd+"/MCProduction/results"
random.seed(None)
centerOfMass = "13000"
channelNumber = "304874"
#offset = 800000000
offset = random.randint(100000000,999999999)


numberOfJobs =2
#numberOfJobs = int(sys.argv[1])
numberOfEvents = "10000"


Names = [

"Powerlaw",

]

TunesImport = [

"Pythia8_A14_NNPDF23LO_EvtGen_Common.py",

]

ExtraLines = [

""" """,
]

if not os.path.exists(baseDir+"/jobOpt"):
          os.makedirs(baseDir+"/jobOpt")


#print zip(VectorOf_M_ExQ, ChannelNumbers)
```

```
submissionfile = open('submitAllSubmissions_8.sh','w')
submissionfile.write("#!/bin/bash\n")
submissionfile.write("#SBATCH --job-name='MCProdCaterina8'\n")

pandafile = open('submitAllSubmissions_panda.sh','w')
pandafile.write("#!/bin/bash\n")

for name, (tuneImport, extraLines) in zip(Names, zip(TunesImport, ExtraLines)) :

    #let's write the JO
    outputJOFileName = "MC15."+str(channelNumber)+".Pythia8EvtGen_A14NNPDF23LO_jetjet_"+name+".py"
    #outputJOFile = open("jobOpt/"+outputJOFileName,"w")
    outputJOFile = open(baseDir+"/jobOpt/"+outputJOFileName,"w")

    #Mass Scale parameter (Lambda, in GeV)

    outputJOFile.write("""

# author: Caterina Doglioni

evgenConfig.description= "Dijet truth jets, with power law"
evgenConfig.keywords= ["QCD", "jets", "SM"]
evgenConfig.generators += ["Pythia8"]
evgenConfig.contact = ["Harinder Singh Bawa <bawa@cern.ch>"]

include("MC15JobOptions/"""+tuneImport+""""")

""")


    outputJOFile.write("""

genSeq.Pythia8.Commands += ["HardQCD:all = on",
        "PhaseSpace:pTHatMin = 20.0",
        "PhaseSpace:bias2Selection = on",
        "PhaseSpace:bias2SelectionPow = 5.0"

    """+extraLines+"""
]

evgenConfig.minevents = 1000
    """)


    #parallelisation starts now!

    for jobNumber in xrange(0, numberOfJobs) :
      #now that we have the JO, let's write a script to run it
      jobscriptName = "MC15."+str(channelNumber)+".Pythia8EvtGen_A14NNPDF23LO_jetjet_"+name+"_"+
      str(jobNumber)+"_"+str(offset)+".sh"
      jobscript = open(jobscriptName,"w")

      jobscript.write("""#!/bin/bash
# Simple submission script for MC15 validation

# for tracking when and where you run
hostname
date

#to avoid NFS problems
#sleep $(perl -e '$ran=int(rand(120)); print STDOUT "$ran\\n";')

# to make and use a temporary directory
tmpDir=`mktemp -d`
cd ${tmpDir}
echo 'we are in '${PWD}

# Setup ROOT and various libraries
#cp /atlas/users/doglioni/authentication.xml .
#export CORAL_AUTH_PATH=.:${CORAL_AUTH_PATH}
#export PYTHON_EGG_CACHE=/atlas/users/doglioni/.python-eggs

#copy everything we need into this temp directory
```

```
if [ ! -d  """+baseDir+"""/jobOpt ]; then
mkdir -p """+baseDir+"""/jobOpt
fi

cp """+baseDir+"""/jobOpt/"""+outputJOFileName+""" .

module load enableATLAS
setupATLAS
source $ATLAS_LOCAL_ROOT_BASE/user/atlasLocalSetup.sh
asetup 19.2.4.16,AtlasProduction
source $AtlasSetup/scripts/asetup.sh 19.2.4.16,AtlasProduction
cd ${tmpDir}

#run the job transform
Generate_tf.py --ecmEnergy="""+centerOfMass+""" --runNumber="""+str(channelNumber)+"""  --firstEvent=0
--maxEvents="""+numberOfEvents+""" --randomSeed="""+str(jobNumber+offset)+"""
--jobConfig="""+outputJOFileName+""" --outputEVNTFile=event.root
| grep cross-section | tee output.log
#evgenJobOpts=MC15JobOpts-00-09-16_v1.tar.gz

#copy the output
#if [ ! -d  """+baseDir+"""/OutputNtuples/SplitQCD_EVNT ]; then
#mkdir -p """+baseDir+"""/OutputNtuples/SplitQCD_EVNT
#fi

#cp event.root """+baseDir+"""/OutputNtuples/SplitQCD_EVNT/"""+outputJOFileName+"""_"""+str(jobNumber)+
"""_"""+str(offset)+"""EVNT.root

source $ATLAS_LOCAL_ROOT_BASE/user/atlasLocalSetup.sh
asetup 20.1.8.3,AtlasDerivation
source $AtlasSetup/scripts/asetup.sh 20.1.8.3,AtlasDerivation
cd ${tmpDir}


#run the ntuplemaker
Reco_tf.py --inputEVNTFile=event.root --outputDAODFile=output.root --reductionConf TRUTH3  --loglevel FATAL

#copy the output
if [ ! -d  """+outputDir+"""/OutputNtuples/SplitQCD ]; then
mkdir -p """+outputDir+"""/OutputNtuples/SplitQCD
fi

cp DAOD_TRUTH3.output.root """+outputDir+"""/OutputNtuples/SplitQCD/"""+outputJOFileName+"""_"""+
str(jobNumber)+"""_"""+str(offset)+""".root
cp output.log """+outputDir+"""/OutputNtuples/SplitQCD/"""+outputJOFileName+"""_"""+
str(jobNumber)+"""_"""+str(offset)+""".log
ls
rm *
"""
  )

      #making it executable

      commands.getoutput("chmod 755 "+jobscriptName)

      #running it
      submissionfile.write("bash "+jobscriptName+" \n")

#creating script for PanDA submission

    for jobNumber in xrange(0, numberOfJobs) :
      pandascriptName =
     "MC15."+str(channelNumber)+".Pythia8EvtGen_A14NNPDF23LO_jetjet_"+name+"_"+str(jobNumber)+"_"+
     str(offset)+"_gridsubmission.sh"
     pandascript = open(pandascriptName,"w")

      pandascript.write("""#!/bin/bash

#source $ATLAS_LOCAL_ROOT_BASE/user/atlasLocalSetup.sh
#asetup 20.1.8.3,AtlasDerivation,here
#source $AtlasSetup/scripts/asetup.sh 20.1.8.3,AtlasDerivation,here
#asetup 21.2.6.0,AthDerivation
#lsetup "asetup 19.2.4.16,AtlasDerivation,here" panda
```

```
#lsetup "asetup 21.2.6.0,AthDerivation,here" panda

str1="user.dsidirop.Pythia8EvtGen.EVNT.v"
str2="user.dsidirop.Pythia8EvtGen.xAOD.v"
SubNum=$RANDOM
outDSEVNT=$str1$SubNum
outDSxAOD=$str2$SubNum

cp """+baseDir+"""/jobOpt/"""+outputJOFileName+""" .

export ATLAS_LOCAL_ROOT_BASE=/cvmfs/atlas.cern.ch/repo/ATLASLocalRootBase
source ${ATLAS_LOCAL_ROOT_BASE}/user/atlasLocalSetup.sh
lsetup "asetup 19.2.4.16,AtlasProduction,here" panda
pathena --trf "Generate_tf.py --ecmEnergy=13000 --runNumber=304874  --firstEvent=0 --maxEvents=10000 --randomSeed="""+
str(jobNumber+offset)+""" --jobConfig="""+outputJOFileName+""" --outputEVNTFile=%OUT.event.root
| grep cross-section | tee output.log" --outDS=$outDSEVNT

lsetup rucio

Rucioline=""
EVNTfile=""

sleep 1m

while [ -z "$EVNTfile" -a $SECONDS -le 7200 ];
do
rucio list-files "user.dsidirop:"$outDSEVNT"_EXT0/" >> tempFile$SubNum.txt
RUCIOline="$(grep .root tempFile$SubNum.txt)"
EVNTfile="$(echo $RUCIOline| cut -d'|' -f 2)"
echo  "Stil in the loop"
sleep 5m

rm tempFile$SubNum.txt
done

export ATLAS_LOCAL_ROOT_BASE=/cvmfs/atlas.cern.ch/repo/ATLASLocalRootBase
source ${ATLAS_LOCAL_ROOT_BASE}/user/atlasLocalSetup.sh
lsetup "asetup 21.2.6.0,AthDerivation,here" panda
pathena --trf "Reco_tf.py --inputEVNTFile=%IN --outputDAODFile=%OUT.output.root --reductionConf TRUTH3  --loglevel FATAL"
--inDS=$outDSEVNT"_EXT0/" --outDS=$outDSxAOD

rm """+outputJOFileName+"""
"""
)

      #making it executable
      commands.getoutput("chmod 755 "+pandascriptName)

      #running it
      pandafile.write("bash " + pandascriptName + "\n")
```

A.6 psequencer script (dropped effort)

```
### Step1
source $ATLAS_LOCAL_ROOT_BASE/user/atlasLocalSetup.sh
lsetup "asetup 19.2.4.16,AtlasProduction,here" panda
pathena --trf "Generate_tf.py --ecmEnergy=13000 --runNumber=304874  --firstEvent=0 --maxEvents=10000
--randomSeed=996496117 --jobConfig=MC15.304874.Pythia8EvtGen_A14NNPDF23LO_jetjet_Powerlaw.py
--outputEVNTFile=%OUT.event.root | grep cross-section | tee output.log"
--outDS=user.dsidirop.Pythia8EvtGen.EVNTseq6.vseq6

### Step2
source $ATLAS_LOCAL_ROOT_BASE/user/atlasLocalSetup.sh
lsetup "asetup 21.2.6.0,AthDerivation,here" panda
pathena --trf "Reco_tf.py --inputEVNTFile=%IN --outputDAODFile=%OUT.output.root --reductionConf TRUTH3  --loglevel FATAL"
--inDS=$OUTDATASET --outDS=user.dsidirop.Pythia8EvtGen.xAODseq6.vseq6

### Sequence
Step1.execute()
result = Step1.result()
```

```
if result.status == 0 and result.Failed == 0:
     print "Succesful generation !"
     var = {}
     var['OUTDATASET'] = result.Out
     Step2.execute(env=var)
```

A.7 Generation and Transformation in one pathena submission (piped submission) (dropped effort)

```
#!/bin/bash

#module load enableATLAS
#setupATLAS
source $ATLAS_LOCAL_ROOT_BASE/user/atlasLocalSetup.sh
source $AtlasSetup/scripts/asetup.sh 20.1.8.3,AtlasDerivation,here
#asetup 20.1.8.3,AtlasDerivation,here
#asetup 21.2.6.0,AthDerivation
#lsetup "asetup 19.2.4.16,AtlasDerivation,here" panda
#lsetup "asetup 21.2.6.0,AthDerivation,here" panda

lsetup panda

pathena --trf "Generate_tf.py --ecmEnergy=13000 --runNumber=304874  --firstEvent=0 --maxEvents=10000
--randomSeed=996496117 --jobConfig=MC15.304874.Pythia8EvtGen_A14NNPDF23LO_jetjet_Powerlaw.py
--outputEVNTFile=event.root | grep cross-section | tee output.log;
Reco_tf.py --inputEVNTFile=event.root --outputDAODFile=%OUT.output.root --reductionConf TRUTH3 --loglevel FATAL"
--outDS=user.dsidirop.Pythia8EvtGen.EVNTxAOD10.test.v10
```

Appendix B

Weighted p_T analysis

The *weighted* p_T *spectra* refer to transverse momentum distributions of the generated jets in the events but only after an event by event weight (available also in the `TTree`) is applied to all events generated (at least those that included jets). Those event by event weights describe essentially the contribution of each event in the total cross section of the process simulated (something that will not be studied here). In this appendix the weighted p_T analysis performed for the different comparisons mentioned in the main thesis text will be presented.

B.1 Iridium job comparison

An overall similar procedure with the one followed for the unweighted jet p_T is followed for the weighted jet p_T spectra. The coding of this comparison process is provided in full in Appendix A.2.2 (again for a different comparison done later but equally applicable here). The only differences that can be noticed pertain to the coding of the drawing of the histograms:

```
Tiridium0->Draw("AntiKt4TruthJetsAux.AntiKt4TruthJetsAux.pt>>hweightsiridium0,
"McEventInfo.m_event_type.m_mc_event_weights");
```

where `McEventInfo.m_event_type.m_mc_event_weights` is another leaf in a branch of the `CollectionTree TTree` that contains the needed event weights for each event. The only other change is the fact that the y-axis was set to a logarithmic scale to ensure proper visualization of the resulting histograms for Job 0 and Job 1, provided in Figures B.1 and B.2 respectively, given that each bin content is now normalized, using the total number of events as normalization factor.

The division of the histograms progresses the same as with the unweighted p_T case. The final comparison graph is saved as a `.eps` image and it is shown in Figure B.3. The comparison of weighted p_T spectra also confirms the fact that the two jobs were generated using different seeds. Sections B.2 and B.3 consist applications of the same coding scheme to compare Job 0 and Job 1 weighted p_T distributions between Iridium and the virtual machine and the Grid and the virtual machine.

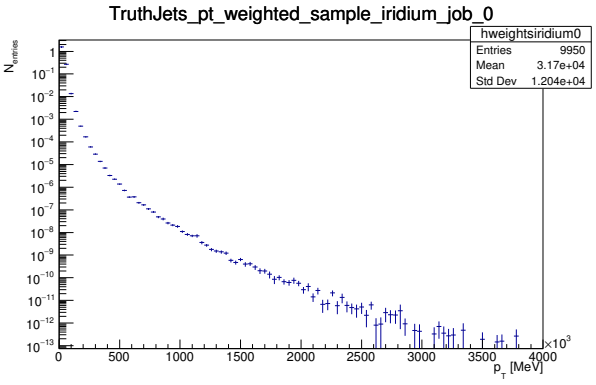

Figure B.1: Weighted p_T distribution for Job 0.

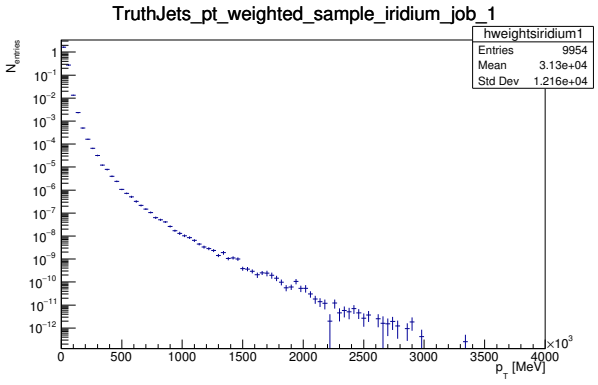

Figure B.2: Weighted p_T distribution for Job 0.

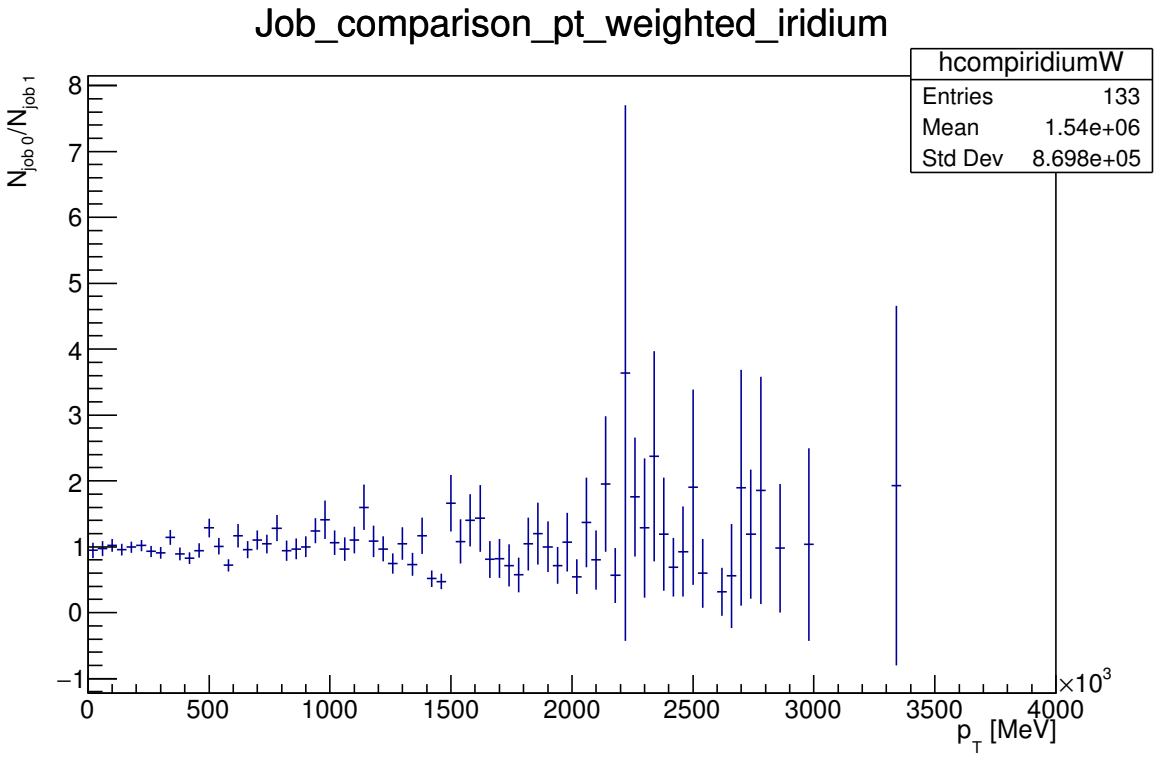

Figure B.3: Comparison of the weighted p_T spectra for the two jobs when processed on Iridium. The comparison shows that the statistical distribution is the same but the spectra compared were generated using different seeds.

B.2 Iridium - VM Comparison

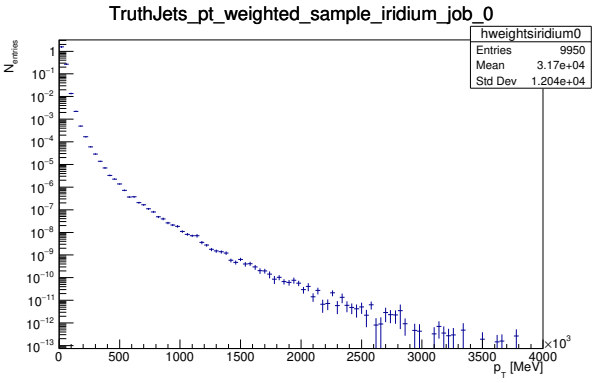

Figure B.4: Weighted p_T distribution for Job 0 processed on Iridium.

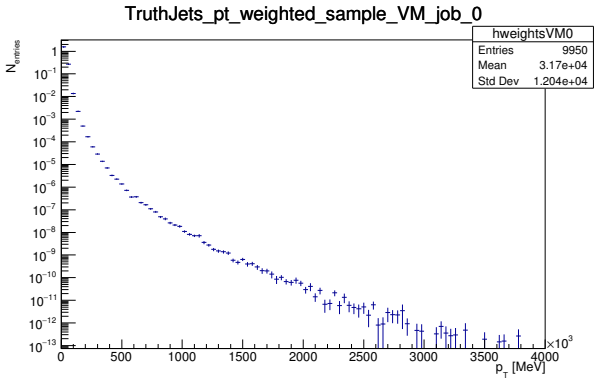

Figure B.5: Weighted p_T distribution for Job 0 processed in the Virtual Machine.

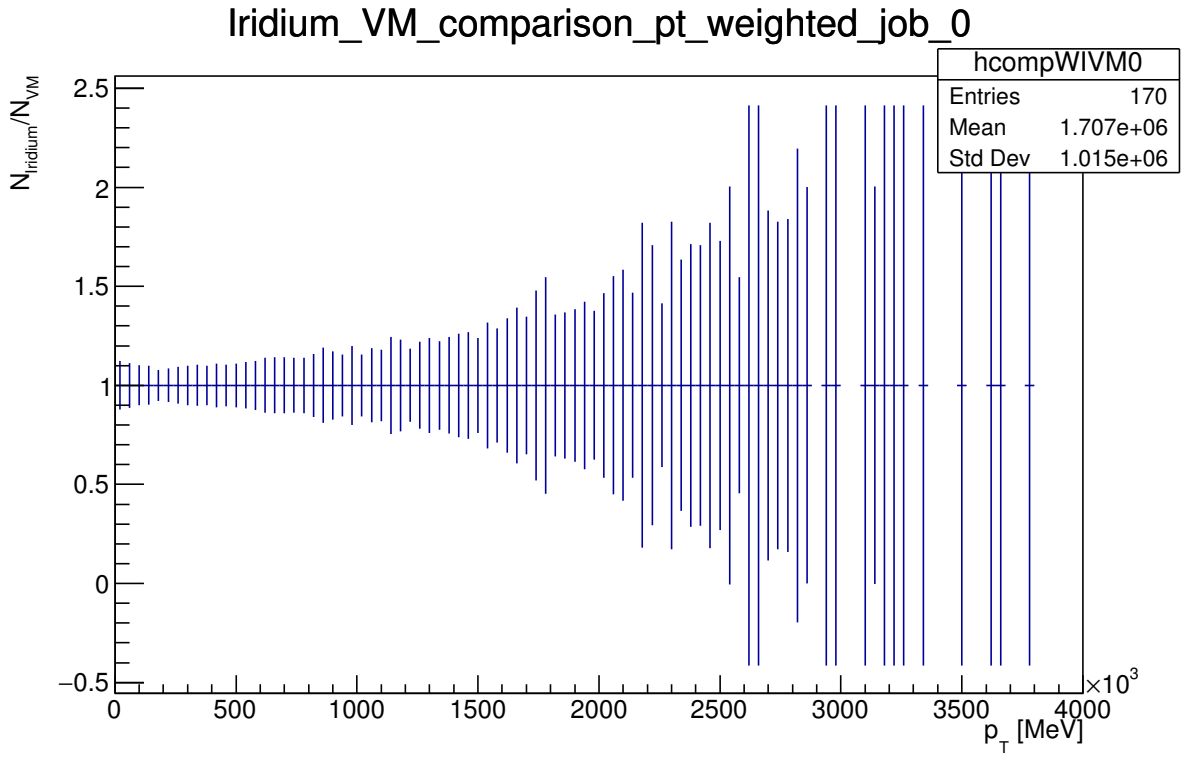

Figure B.6: Comparison of the jet transverse momentum p_T distribution for the same Job 0 (same seed) when processed on Iridium and the VM environment for data to which an event by event weight has been applied. The comparison shows that the generation is independent of the environment in which it was developed, being identical for the same seed.

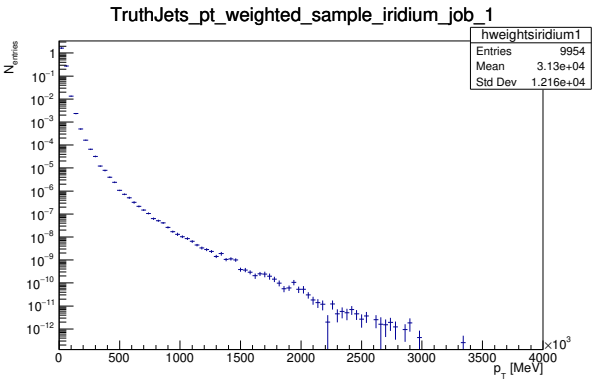

Figure B.7: Weighted p_T distribution for Job 1 processed on Iridium.

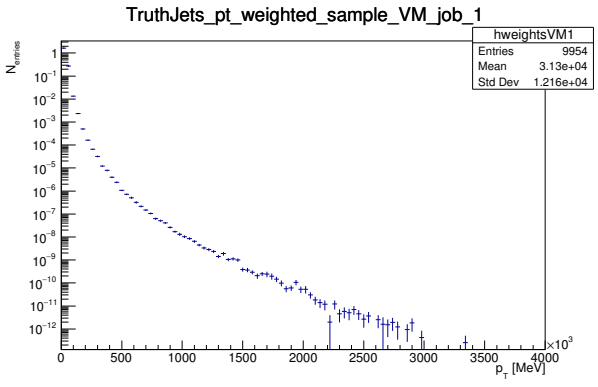

Figure B.8: Weighted p_T distribution for Job 1 processed in the Virtual Machine.

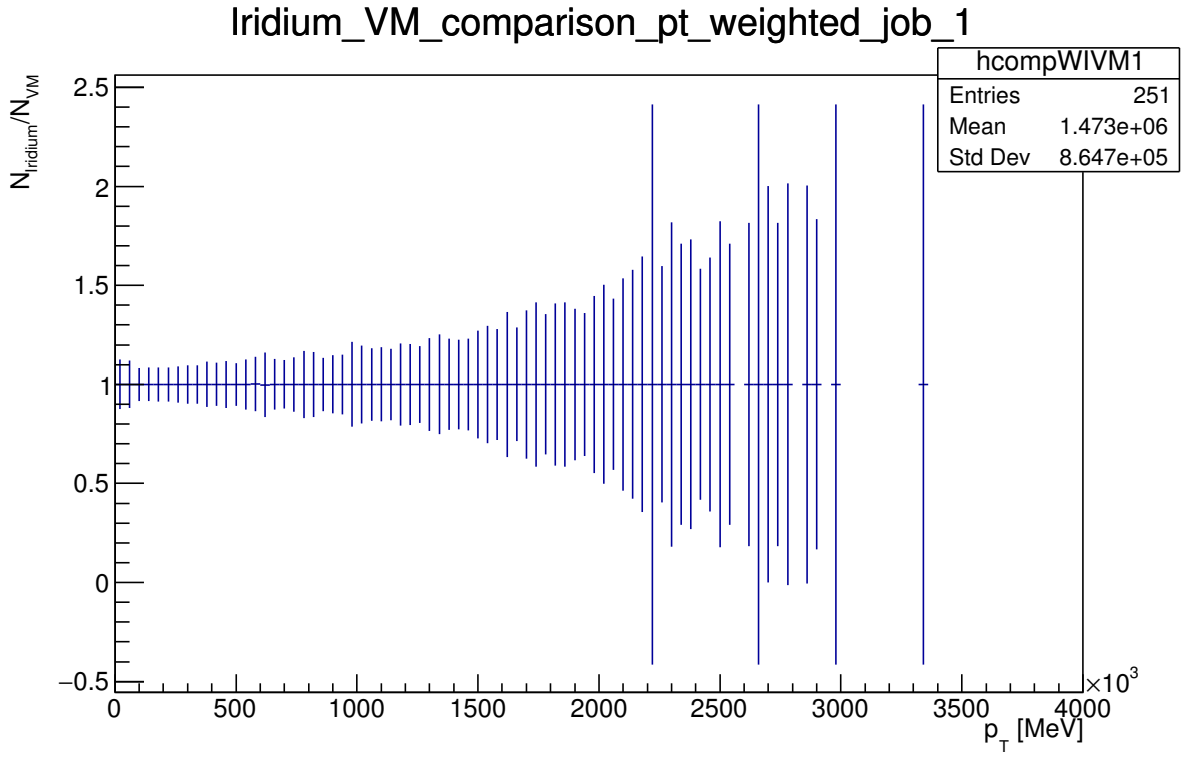

Figure B.9: Comparison of the jet transverse momentum p_T distribution for the same Job 1 (same seed) when processed on Iridium and the VM environment for data to which an event by event weight has been applied. The comparison shows that the generation is independent of the environment in which it was developed, being identical for the same seed, with the exception of minor bin migration occurrences.

B.3 Grid - VM comparison

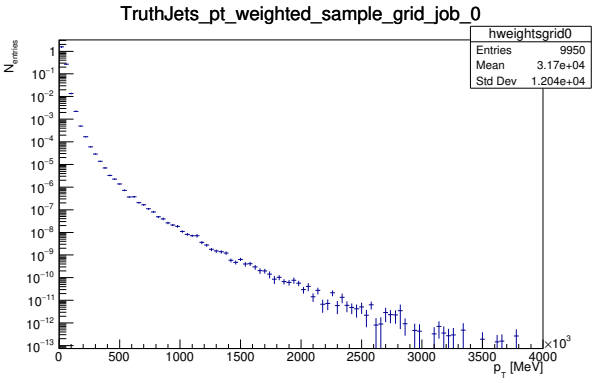

Figure B.10: Weighted p_T distribution for Job 0 processed on the Grid.

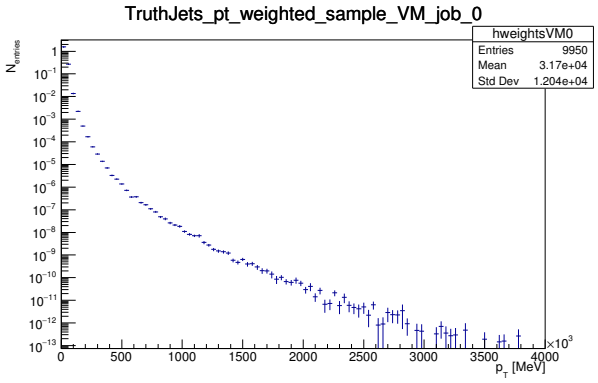

Figure B.11: Weighted p_T distribution for Job 0 processed in the Virtual Machine.

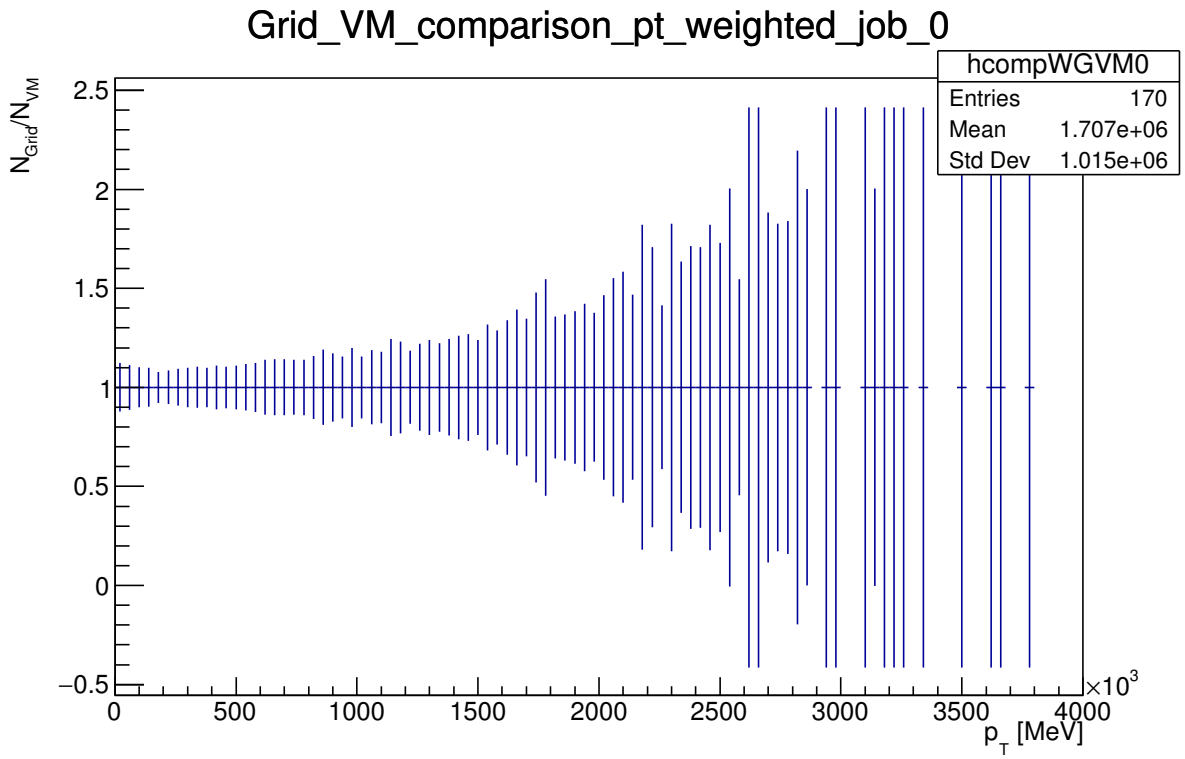

Figure B.12: Comparison of the jet transverse momentum p_T distribution for the same Job 0 (same seed) when processed on the Grid and the VM environment for data to which an event by event weight has been applied. The comparison shows that the generation is independent of the environment in which it was developed, being identical for the same seed.

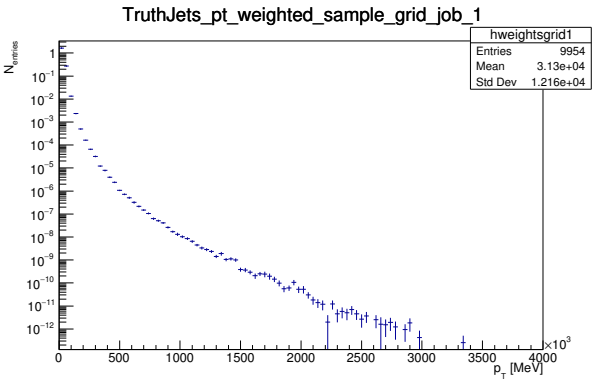

Figure B.13: Weighted p_T distribution for Job 1 processed on the Grid.

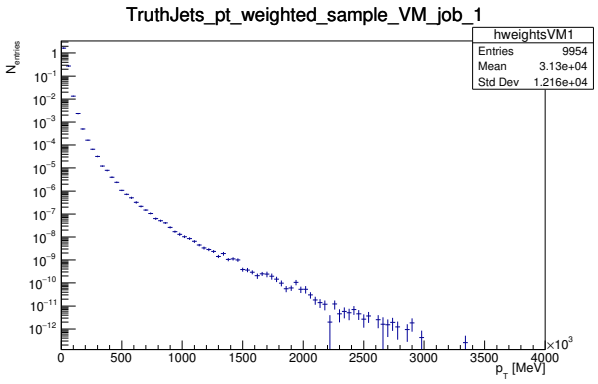

Figure B.14: Weighted p_T distribution for Job 1 processed in the Virtual Machine.

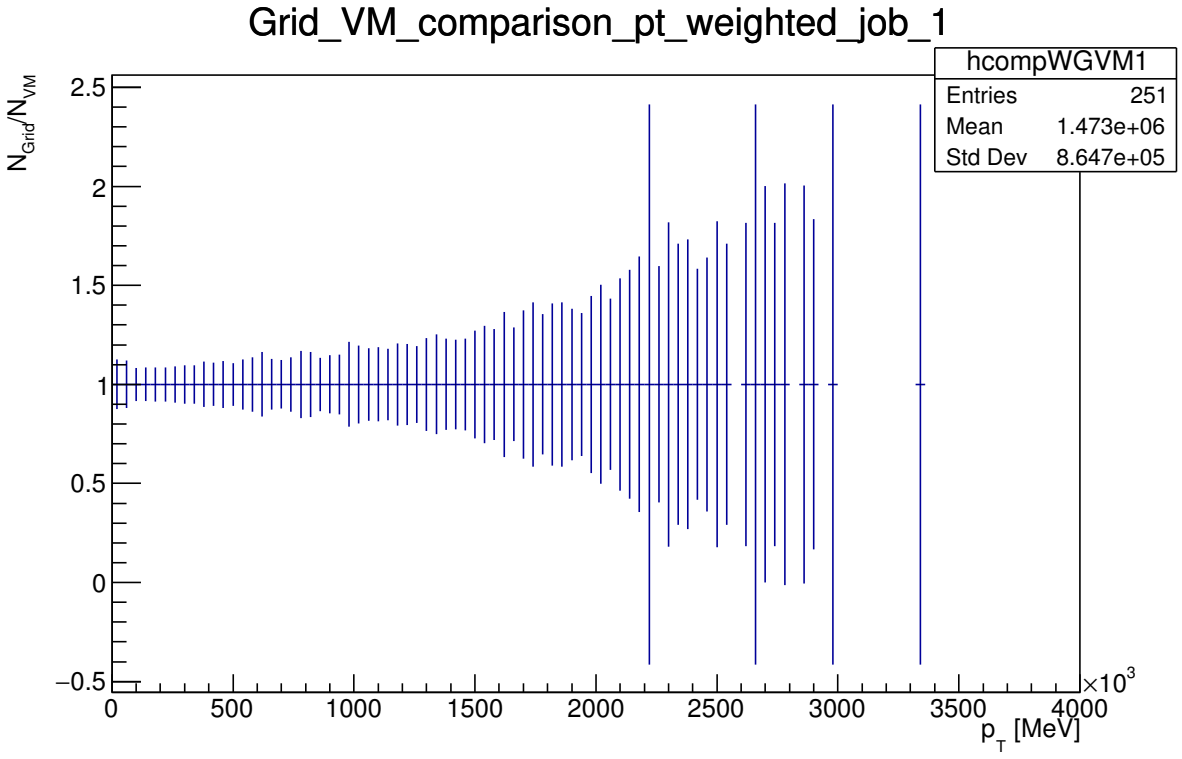

Figure B.15: Comparison of the jet transverse momentum p_T distribution for the same Job 1 (same seed) when processed on the Grid and the VM environment for data to which an event by event weight has been applied. The comparison shows that the generation is independent of the environment in which it was developed, being identical for the same seed.

Appendix C

General Plot Appendix

C.1 Unweighted p_T and invariant m_{jj} spectra for Job 0 and Job 1 from Iridium.

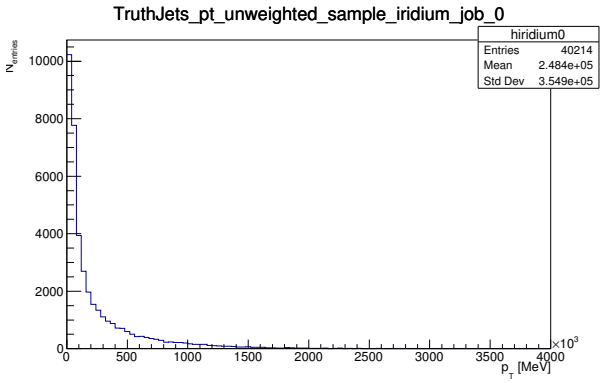

Figure C.1: Unweighted p_T distribution for Job 0 processed on Iridium.

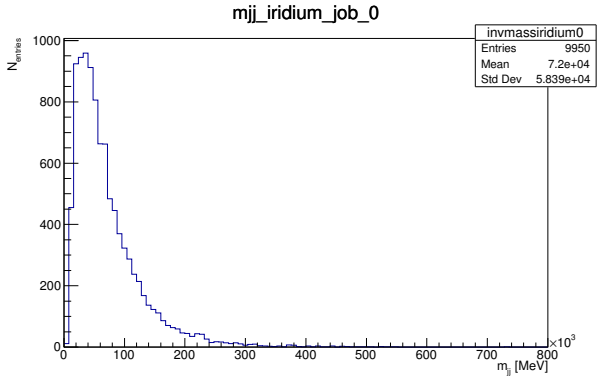

Figure C.2: Dijet mass m_{jj} distribution for Job 0 processed on Iridium.

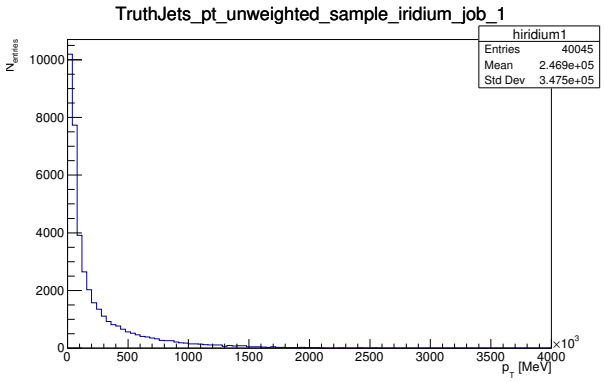

Figure C.3: Unweighted p_T distribution for Job 1 processed on Iridium.

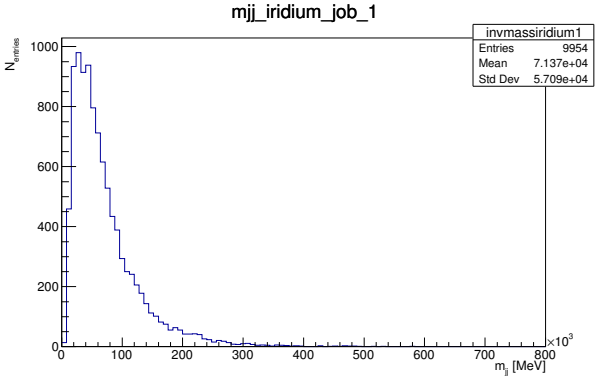

Figure C.4: Dijet mass m_{jj} distribution for Job 1 processed on Iridium.

C.2 Unweighted p_T and invariant m_{jj} spectra for Job 0 and Job 1 from the Virtual Machine

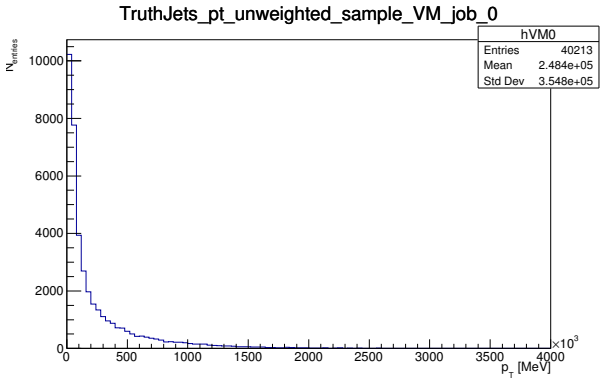

Figure C.5: Unweighted p_T distribution for Job 0 processed in the Virtual Machine.

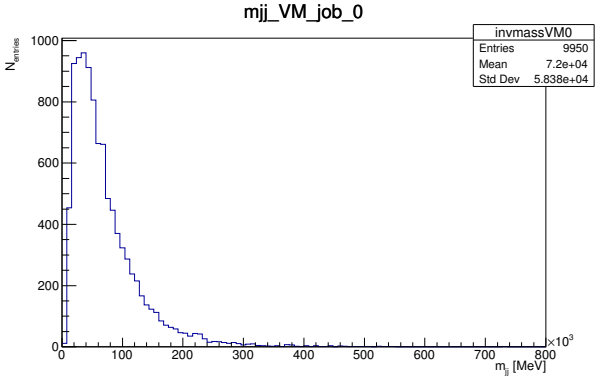

Figure C.6: Dijet mass m_{jj} distribution for Job 0 processed in the Virtual Machine.

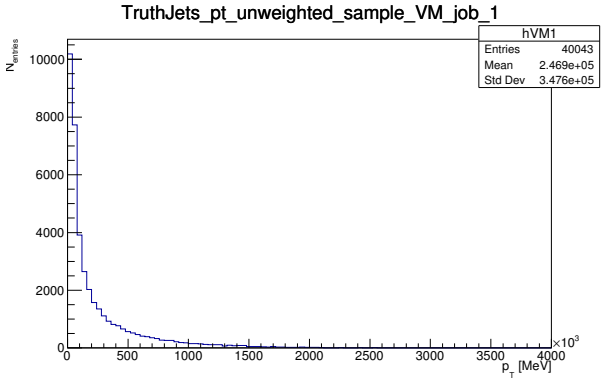

Figure C.7: Unweighted p_T distribution for Job 1 processed in the Virtual Machine.

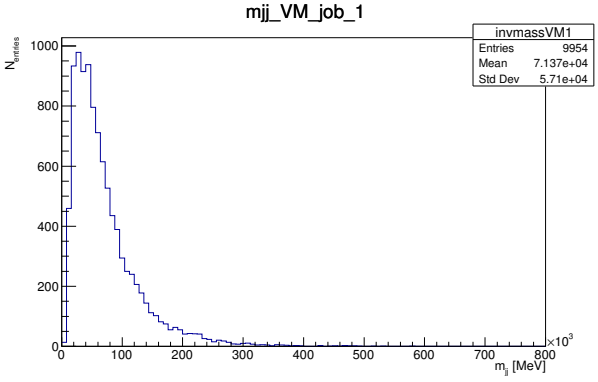

Figure C.8: Dijet mass m_{jj} distribution for Job 1 processed in the Virtual Machine.

C.3 Unweighted p_T and invariant m_{jj} spectra for Job 0 and Job 1 from the Grid

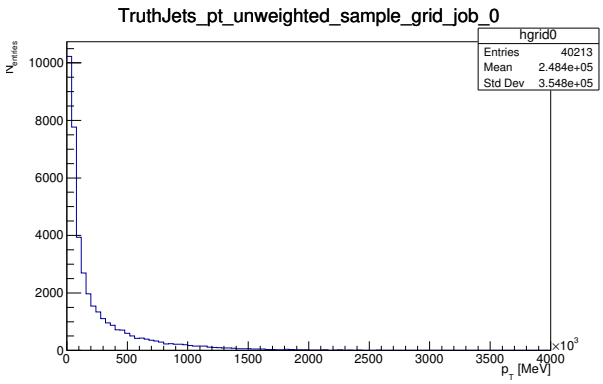

Figure C.9: Unweighted p_T distribution for Job 0 processed on the Grid.

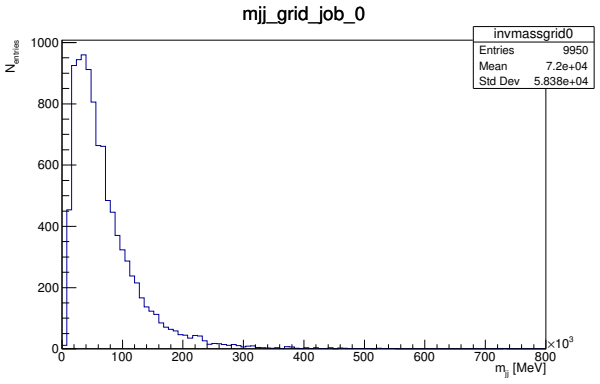

Figure C.10: Dijet mass m_{jj} distribution for Job 0 processed on the Grid.

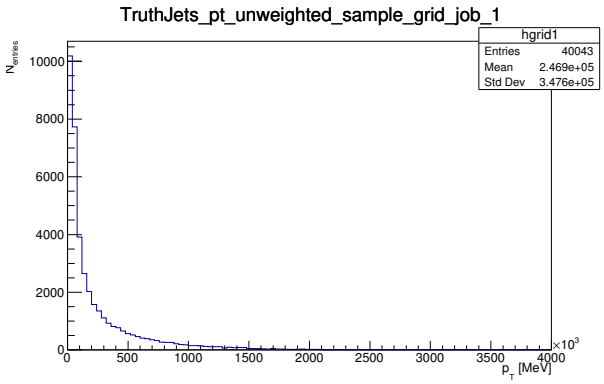

Figure C.11: Unweighted p_T distribution for Job 1 processed on the Grid.

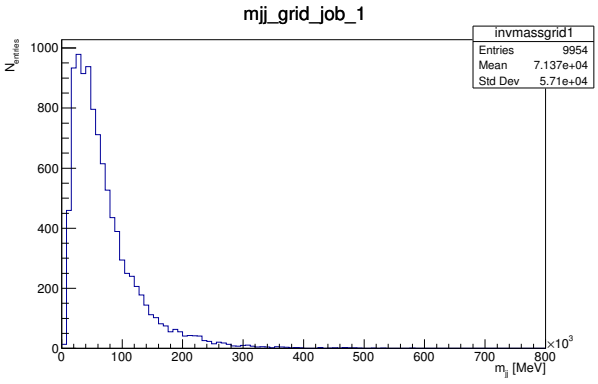

Figure C.12: Dijet mass m_{jj} distribution for Job 1 processed on the Grid.

Bibliography

[1] GLASHOW, Sheldon L. Partial-symmetries of weak interactions. Nuclear Physics, 1961, 22.4: 579-588.

[2] WEINBERG, Steven. A model of leptons. Physical review letters, 1967, 19.21: 1264.

[3] SALAM, A. inProceedings of the VIII Nobel Symposium, edited by N. Svartholm. 1968.

[4] VELTMAN, Martinus, et al. Regularization and renormalization of gauge fields. Nuclear Physics B, 1972, 44.1: 189-213.

[5] DJOUADI, Abdelhak. The anatomy of electroweak symmetry breaking: Tome I: The Higgs boson in the Standard Model. Physics reports, 2008, 457.1: 1-216.

[6] ABBOTT, Benjamin P., et al. Observation of gravitational waves from a binary black hole merger. Physical review letters, 2016, 116.6: 061102.

[7] COTTINGHAM, W. Noel; GREENWOOD, Derek A. An introduction to the standard model of particle physics. Cambridge university press, 2007.

[8] BRYNGEMARK, Lene. Search for new phenomena in dijet angular distributions at $\sqrt{s}$= 8 and 13 TeV. 2016. PhD Thesis. Lund U.

[9] ENGLERT, François; BROUT, Robert. Broken symmetry and the mass of gauge vector mesons. Physical Review Letters, 1964, 13.9: 321.

[10] HIGGS, Peter W. Broken symmetries and the masses of gauge bosons. Physical Review Letters, 1964, 13.16: 508.

[11] GURALNIK, Gerald S.; HAGEN, Carl R.; KIBBLE, Thomas WB. Global conservation laws and massless particles. Physical Review Letters, 1964, 13.20: 585.

[12] HIGGS, Peter W. Spontaneous symmetry breakdown without massless bosons. Physical Review, 1966, 145.4: 1156.

[13] KIBBLE, T. W. B. Symmetry breaking in non-Abelian gauge theories. Physical Review, 1967, 155.5: 1554.

[14] ALEPH COLLABORATION, et al. Search for the Standard Model Higgs boson at LEP. Physics Letters B, 2003, 565: 61-75.

[15] GAGNON, Pauline.The Standard Model : A beautiful but flawed theory.Quantum Diaries [cited 20171211] Available from : https://www.quantumdiaries.org/2014/03/14/the-standard-model-a-beautiful-but-flawed-theory/

[16] AAD, Georges, et al. Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC. Physics Letters B, 2012, 716.1: 1-29.

[17] CHATRCHYAN, Serguei, et al. Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC. Physics Letters B, 2012, 716.1: 30-61.

[18] AD, Georges, et al. Combined Measurement of the Higgs boson Mass in p p Collisions at s= 7 and 8 TeV with the ATLAS and CMS Experiments. Physical review letters, 2015, 114.19: 191803.

[19] AAD, Georges, et al. Search for Dark Matter candidates and large extra dimensions in events with a jet and missing transverse momentum with the ATLAS detector. Journal of High Energy Physics, 2013, 2013.4: 75.

[20] AAD, Georges, et al. The ATLAS experiment at the CERN large hadron collider. Journal of Instrumentation, 2008, 3.8: S08003-S08003.

[21] ATLAS, Detector & Technology [cited 20171116] Available from : https://atlas.cern/discover/detector

[22] ATLAS undergoes some delicate gymnastics. Cern Courier,27/09/13 [cited 20171116] Available from : http://cerncourier.com/cws/article/cern/54676

[23] BAGNAIA, P., et al. Measurement of production and properties of jets at the CERN $\bar{p}p$ collider. Zeitschrift für Physik C Particles and Fields, 1983, 20.2: 117-134.

[24] ELLIS, Stephen D.; SOPER, Davison E. Successive combination jet algorithm for hadron collisions. Physical Review D, 1993, 48.7: 3160.

[25] HATAKEYAMA, Kenichi. (for the CDF Collaboration). Jet Physics at CDF. International Symposium of Multiparticle Dynamics, August 5-9, 2007. [cited 20171205] Available from : https://www-cdf.fnal.gov/physics/talks_transp/2007/ismd_jetphys_hatakeyama.pdf

[26] SCHIEFERDECKER, Philip. Jet Algorithms. Vivian's meeting, Karlsuhe Institute of Technology (KIT), 17 April, 2009. [cited 20171205] Available from : https://twiki.cern.ch/twiki/bin/viewfile/Sandbox/Lecture?rev=1;filename=Philipp_Schieferdeckers_Lecture.pdf

[27] SJÖSTRAND, Torbjörn; MRENNA, Stephen; SKANDS, Peter. A brief introduction to PYTHIA 8.1. Computer Physics Communications, 2008, 178.11: 852-867.

[28] MARCHESINI, Giuseppe, et al. HERWIG 5.1-a Monte Carlo event generator for simulating hadron emission reactions with interfering gluons. Computer Physics Communications, 1992, 67.3: 465-508.

[29] GLEISBERG, Tanju, et al. Event generation with SHERPA 1.1. Journal of High Energy Physics, 2009, 2009.02: 007.

[30] AGOSTINELLI, Sea, et al. GEANT4—a simulation toolkit. Nuclear instruments and methods in physics research section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 2003, 506.3: 250-303.

[31] LENZI, Bruno, et al. The Physics Analysis Tools project for the ATLAS experiment. In: AIP Conference Proceedings. AIP, 2012. p. 987-990.

[32] BOINC, Open-source software for volunteer computing [cited 20171122] Available from : https://boinc.berkeley.edu/

[33] PAGANELLI,Florido. Operating systems Linux Installation, MNXB01 programming course, Lund University, 2016 [cited 20171120] Available from : http://www.hep.lu.se/courses/MNXB01/2016/MNXB01-tutorial-1a.pdf

[34] LAWTON, Kevin P. Bochs: A portable pc emulator for unix/x. Linux Journal, 1996, 1996.29es: 7.

[35] VirtualBox, About VirtualBox [cited 20171120] Available from : https://www.virtualbox.org/wiki/VirtualBox

[36] BOURDARIOS, Claire, et al. Volunteer Computing Experience with ATLAS@ Home. ATL-COM-SOFT-2016-140, 2016.

[37] KORPELA, Eric, et al. SETI@ HOME—massively distributed computing for SETI. Computing in science & engineering, 2001, 3.1: 78-83.

[38] BOINC, Choosing BOINC projects [cited 20180114] Available from : https://boinc.berkeley.edu/projects.php

[39] GIOVANNOZZI, M., et al. LHC@ HOME: A volunteer computing system for massive numerical simulations of beam dynamics and high energy physics events. In: Conf. Proc. 2012. p. MOPPD061.

[40] CAMERON, David, et al. on behalf of the ATLAS Collaboration. Volunteer Computing Experience with ATLAS@Home,University of Oslo. 13/10/2016 [cited 20171122] Available from : https://indico.cern.ch/event/505613/contributions/2230707/attachments/1346591/2030567/Oral-153.pdf

[41] BLOMER, Jakob, et al. Distributing LHC application software and conditions databases using the CernVM file system. In: Journal of Physics: Conference Series. IOP Publishing, 2011. p. 042003.

[42] MAENO, Tadashi. PanDA: distributed production and distributed analysis system for ATLAS. In: Journal of Physics: Conference Series. IOP Publishing, 2008. p. 062036.

[43] NORDUGRID, ARC Compute Element [cited 20171122] Available from : http://www.nordugrid.org/arc/ce/

[44] KITTELMANN, Thomas, et al. The virtual point 1 event display for the ATLAS experiment. In: Journal of Physics: Conference Series. IOP Publishing, 2010. p. 032012.

[45] DOBBS, Matt; HANSEN, Jørgen Beck. HepMC: a C++ Event Record for Monte-Carlo Generators. Comput. Phys. Commun, 2001, 134: 41.

[46] ANTCHEVA, Ilka, et al. ROOT—A C++ framework for petabyte data storage, statistical analysis and visualization. Computer Physics Communications, 2011, 182.6: 1384-1385.

[47] MARSHALL, Zachary. TruthDAOD,PhysicsModellingGroup,AtlasPhysics,AtlasProtected Web. TWiki. 29-11-2017 [cited 20171206] Available from : https://twiki.cern.ch/twiki/bin/viewauth/AtlasProtected/TruthDAOD (CERN account required)

[48] PAGANELLI, Florido. pfwiki, Iridium Cluster [cited 20171129] Available from : http://www.hep.lu.se/staff/paganelli/doku.php/iridium_cluster

[49] CACCIARI, Matteo; SALAM, Gavin P.; SOYEZ, Gregory. The anti-kt jet clustering algorithm. Journal of High Energy Physics, 2008, 2008.04: 063.

[50] ATLAS COLLABORATION, et al. Search for new phenomena in the dijet mass distribution using pp collision data at $\sqrt{s} = 8$ TeV with the ATLAS detector. arXiv preprint arXiv:1407.1376, 2014.

[51] ATLAS PanDA, ATLAS PanDA monitor home [cited 20171203] Available from : https://bigpanda.cern.ch/ ("not secure" warning)

[52] BUNCIC, Predrag, et al. CernVM–a virtual software appliance for LHC applications. In: Journal of Physics: Conference Series. IOP Publishing, 2010. p. 042003.

[53] CernVM Software Appliance, Cloud Contextualization [cited 20171124] Available from : http://cernvm.cern.ch/portal/contextualisation#bootloader

[54] CernVM Online, Create a Context in CernVM Online [cited 20171126] Available from : https://cernvm.web.cern.ch/portal/online/documentation/create-new-context

[55] CernVM online, CernVM Marketplace [cited 20171126] Available from : https://cernvm-online.cern.ch/market/list

[56] ATLAS-test [cited 20171204] Available from : http://boincai04.cern.ch/Atlas-test/index.php

[57] ATLAS-test, Applications [cited 20171204] Available from : http://boincai04.cern.ch/Atlas-test/apps.php