A COMPARISON OF MUDST AND PICODST DATA ANALYSIS METHODS

An Undergraduate Research Scholars Thesis

by

SETH HALL

Submitted to the LAUNCH: Undergraduate Research office at
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by
Faculty Research Advisor: Dr. Saskia Mioduszewski

May 2022

Major: Physics, B.S.

RESEARCH COMPLIANCE CERTIFICATION

Research activities involving the use of human subjects, vertebrate animals, and/or biohazards must be reviewed and approved by the appropriate Texas A&M University regulatory research committee (i.e., IRB, IACUC, IBC) before the activity can commence. This requirement applies to activities conducted at Texas A&M and to activities conducted at non-Texas A&M facilities or institutions. In both cases, students are responsible for working with the relevant Texas A&M research compliance program to ensure and document that all Texas A&M compliance obligations are met before the study begins.

I, Seth Hall, certify that all research compliance requirements related to this Undergraduate Research Scholars thesis have been addressed with my Research Faculty Advisor prior to the collection of any data used in this final thesis submission.

This project did not require approval from the Texas A&M University Research Compliance & Biosafety office.

TABLE OF CONTENTS

ABSTRACT

A Comparison of MuDST and PicoDST Data Analysis Methods

Seth Hall
Department of Physics and Astronomy
Texas A&M University

Research Faculty Advisor: Saskia Mioduszewski, PhD
Department of Physics and Astronomy
Texas A&M University

For STAR-experiment associated research at Brookhaven National Laboratory, the primary objective is the study and characterization of the Quark-Gluon Plasma, or QGP. The data is stored in subsets of the origninal, full 'Data Summary Tape' or DST format. The PicoDST format is the new, more compact way in which all data from particle collider experiments is stored. Because this data format is relatively new, much of the existing data analysis scripts written by the TAMU group to run on root4star (the modified ROOT software package used for the STAR experiment) work only with MuDST-type files, which is the older format. While extensive documentation exists for both the PicoDST and MuDST formats, not every aspect is exactly analogous, and therefore methods to convert older MuDST-scripts to perform the same analytical functions with PicoDST files must be created. In this paper, the differences between the PicoDST and MuDST formats will be considered, as well as the differences between algorithms written to accomplish the same task with either format. In addition, a significantly higher time-efficiency is demonstrated for a particular PicoDST algorithm when measured against its MuDST counterpart.

ACKNOWLEDGMENTS

Contributors

I would like to thank my faculty advisor, Dr. Mioduszewski, for her guidance and support throughout the course of this research.

Thanks also go to my friends and colleagues and the department faculty and staff for making my time at Texas A&M University a great experience.

Funding Sources

No financial sponsors or external funding of any kind were used to produce the research presented in this paper, except for those materials which are explicitly stated to have come from different publications, and are reproduced here according to the licensing guidelines of their original publishers.

NOMENCLATURE

BNL	Brookhaven National Laboratory
RHIC	Relativistic Heavy-Ion Collider
STAR	Solenoidal Tracker At RHIC, from which the STAR experimental group gets its name
QGP	Quark-Gluon Plasma, the main object of study of the STAR experiment
MuDST	The Micro-DST data format, abbreviated as Mu-DST
picoDST	The Pico-DST data format, which in 2018 supplanted MuDST as the standard format for STAR experimental data and analysis software
SSDC	Scientific Data and Computing Center
run	An experiment with a BNL particle accelerator, for which data is available.
event	During a run, detectors surrounding the beam measure an incident particle that was thrown off by the interaction of beam particles. Information gathered about this particle is stored as an *event*.
hit	Energy registered above a certain threshold by a detector, determined to be a particle-candidate.
tower	A BEMC tower, one of the main calorimeter detectors important to PicoDST analysis.
track	Particles registerd by several detectors can be 'traced' back to a single point of origin within the beam, this is called a *track*.
pointer	A C++ or C object, which 'points' to the memory address of some specific piece of data.
argument	Any object which must be given, or 'passed', to a function or class when it is called, so that the function can have direct access to it.

1. INTRODUCTION

1.1 The STAR Experiment

The STAR experiment, located at Brookhaven National Laboratory (BNL) in Brookhaven, New York, is primarily concerned with the study of Quark-Gluon Plasma, QGP, a very-high energy state of matter in which hadrons are no longer bound; their constituent quarks and associated gluons may separate and interact in ways not possible at normal temperatures [1]. To measure properties of this unique state of matter, the STAR experiment makes use of the Relativistic Heavy Ion Collider, or RHIC [2]. In one particular type of experiment, gold nuclei are collided at extremely high energies to annihilate their subatomic components, which may briefly create a region of dense QGP [3]. New particles may be created by quarks and gluons interacting and re-combining as they exit this energetic state. As these new particles fly outwards from the epicenter of the nuclear collision, they are intercepted by one of the many detectors which encircle the point of collision [2] of the many detectors used at the RHIC, the data collected by the various components of the STAR detector is used in the study of QGP.

1.2 Data Collected by the STAR Detector

The STAR detectors collect a variety of information about the particles which pass through them, and they send this data to be stored in large databases. Because the nature of QGP is not well known, and because of its inherit complexity, any particular STAR experiment requires a very large number of measurements to yield a meaningful result [3]. These factors mean that an enormous amount of data must be saved from each experiment. As a result, the amount of data accumulated over years of experimentation became quite large and required more resources to manage.

In response to these technical challenges, the PicoDST data format was created. Unlike the older MuDST format, PicoDST files only store experimental data which is strictly necessary for physical analysis, discarding much of the detector's data that would have otherwise been saved with MuDST [4]. A goal of this reduction, for instance, is to access all STAR data without the

need for additional external storage, which would require a data size reduction from MuDST by **a factor of approximately 10** [5].

Because of this, the same data analysis techniques used for MuDST files do not necessarily work for PicoDST files. Therefore, algorithms designed to analyze MuDST data are rendered obsolete, or must be re-written. The conversion of the data analysis scripts which were designed for MuDST data files is not necessarily trivial; the fundamental structure of PicoDST and MuDST – although designed to be similar – differ in important ways.

1.3 The Goals and Applications of PicoDST and MuDST Analysis Software

The ultimate goal of processing experimental from the STAR detectors by the TAMU STAR group is the study and characterization of the QGP. The purpose of the software discussed in this paper is to process the raw data that is available in the MuDST and PicoDST formats into usable experimental data. The data that these algorithms output must further be processed by different software to produce meaningful experimental results. So although the end product of these algorithms is not the final data-analysis step, it is a vital step to gaining meaningful scientific results from the raw data.

An example of a practical application of the software discussed in this paper is shown in Figure 1.1.

The ability to discriminate π^0 or γ-rich hits is indispensable to generating useful physical analysis from the raw data of PicoDST or MuDST. This is a consequence of the different physical properties of those two particle types, despite the fact that they both make energetic deposits within the various detectors that are indistinguishable to any individual calorimeter.

Because of this, MuDST or PicoDST analysis software must be able to analyze the data from a broad range of different types of detectors to help researchers determine whether a particular hit was likely a π^0 or γ. One way this is done is by calculating the **TSP**, which is addressed in greater detail in section 2.5.

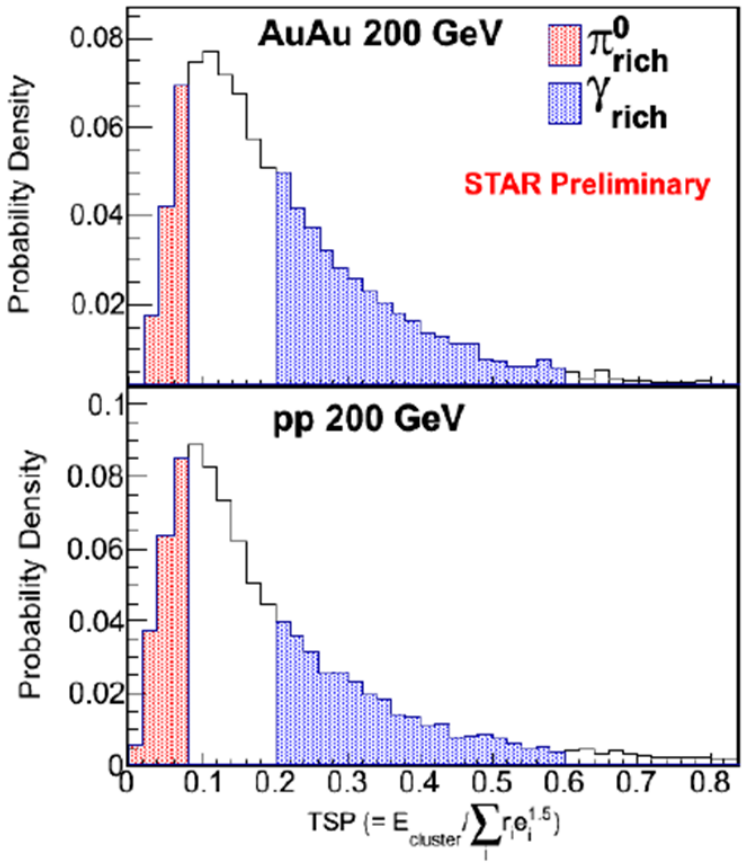

Figure 1.1: A diagram demonstrating the π^0 or γ-richness of different hits, which is determined by the TSP. In particular, the data analyzed in this paper is from **AuAu 200 GeV** experiments. Reproduced from *Jet-like Correlations with Direct-Photon and Neutral-Pion Triggers at* $\sqrt{s_{NN}} = 200$ *GeV* [6].

In this paper, the fundamental differences between these two data formats will be considered. Specifically, how these differences affect existing data analysis methods and software, and the structural differences that arise between MuDST and PicoDST analysis software as a consequence of these differences will be considered. In addition, the time-efficiency of both formats will be measured and compared using existing software for the MuDST and modified software for the PicoDST.

2. METHODS

2.1 Accessing and Editing Code Remotely

For the research conducted in this paper, the PicoDST code is edited and tested on the Scientific Data and Computing Center, or SSDC, which is on servers run by BNL. To access the SSDC, one must use Secure Shell protocols, or SSH.

A program called PuTTY is used to establish this connection, which requires a unique key to be generated and used every time the user connects to the SSDC. Once a user successfully connects to the SSDC, they may use the command *rterm* within the PuTTY terminal to connect to a node, which is where a user can actually edit and run software remotely on BNL servers.

The node is accessed by an X-terminal; the program used for the purposes of this research is Xming. The *rterm* command automatically selects the node that is the least occupied, so that users do not need to manually select the nodes with the least traffic.

The operating system which the nodes operate on in a Linux-terminal based system. EMACS is an open-sourced text editing software that can be opened within the terminal to edit and save files within any directory.

2.2 Types of Files

Within the Xming X-Terminal, the procedure for editing different types of software can vary greatly. To clarify these differences, the different types of software that are necessary for PicoDST data analysis are detailed below.

The first type of files that will be detailed here are **maker files**. Maker files are C++ source-code files that lie within the StRoot directory. These files are written by scientists wishing to perform a very specific analysis of PicoDST (or MuDST) data, and they make use of other classes such as **StPicoDSTMaker** and **StPicoDSTEvent** to accomplish this. The exact role of these classes within maker files will be explained in greater in section 2.3.

The second type of file that will be discussed are the classes that are pre-made for use by

STAR researchers for many different purposes. These files are also C++ files, and they can also be in the **StRoot** directory, albeit under their own separate sub-directory. Most of the time, they function as an intermediate step in the data-analysis process by providing access to raw data. Just a few of the very important roles they serve in include pre-organizing and grouping experimental data for users to analyse, and storing important geometrical information about the physical layout of different detectors.

The next important type of file is the **C-macro**, so named for the language they are written in, C. Their primary function is preparing and facilitating the actual application of the *maker*-algorithms to the data files which they will analyze. As an example: an experimenter writes a *maker*-file whose purpose is to count the number of detectors in each run that registered a *hit* (a *run* is a particle accelerator session in which the resulting data was stored for later analysis.) In *Pico* and *MuDST*, the runs are automatically separated into *events*, each event corresponding to a major particle-hit that was registered by a detector. This experimenter would then also need to construct a *macro*. This macro would effectively tell their *maker*-code which raw data file to use, which pre-made classes to use with it (*MuDST* or *PicoDST*,) and where the *maker* should store its output data. It is this Macro which most directly interfaces with root4star software, which means that the lines of code within the macro are actually ROOT commands. The macros are not inside the StRoot directory, but rather in the directory which also contains StRoot.

Perhaps the most important file (even though all are equally crucial to the successful analysis of experimental data) is the output file. Because the software and data types discussed in this report are all fundamentally ROOT[1] or ROOT-derived, the file-extension for these output files is *.root*. an example may be *outputTest.root*. These files are designed to be opened in ROOT or root4star, and the contents of the analysis can be processed there. These may be anything from histograms produced with data from each event, to raw data kept by the 'maker' code.

Just as important as this are raw data files. A *maker*, in general, has no upper limit on the number of raw data files which can be processed into one output file, but one file is often sufficient

[1]ROOT is a data analysis framework created at CERN which is used by researchers for its ability to process large quantities of experimental data [7].

for testing purposes. The raw data file is interpreted and converted into more useful data by the pre-made classes detailed in the next section.

2.3 Pre-Made STAR Software Classes

As mentioned above, star has many published software classes that are necessary to interpret *PicoDST* and *MuDST* raw data files. Many of the most notable classes are mentioned below, although it should be noted that many additional classes indispensable to PicoDST data analysis are not included.

StPicoDSTMaker() – This class is designed specifically to convert MuDST raw experimental data files into PicoDST data files. This has already been done beforehand for many MuDST files, and as such many available MuDST data files also already have published PicoDST files of the same data. Furthermore, data will only be available in the PicoDST format in the future, which might seem to render this class obsolete. However, this class provides a tremendous insight into the differences between MuDST and PicoDST, as its sole funciton is to convert one to the other. As such, it will be considered extensively in this paper.

StPicoDST() – Together with the **StPicoEvent()** class described below, This is perhaps the most important class to PicoDST data analysis. From this class the *maker*-code accesses important information, including the pointer to **StPicoEvent()** itself. Other very important pointers accessed via this class include various different types of *hit* and *track* classes, which handle information about registered tower hits and reconstructed particle tracks, respectively. The macro responsible for facilitating the *maker*-code passes the pointer to this class directly as an argument, which is often the only pointer which is passed between the *maker* and macro, making the **StPicoDST()** one of the only direct links between the two files during the processing of a run.

StPicoEvent() – This class is very important to directly accessing a variety of important information about each event. As mentioned before, each event generally corresponds to one large registered hit on a tower, and thus **StPicoDST()** is largely responsible for handling the **primary vector**, a position in which a particular event is thought to have originated in the beam path. For any particular run, thousands of gold nuclei collide within the detector. Therefore, each event is

carefully analyzed to determine where the collision took place for particle this event is concerned with. This class also counts the number of total registered hits for a wide variety of detectors.

StEmcGeom() – Shortened from 'EMC Geometry', this class stores information about the specific geometrical layout of the towers of various types. In this paper, the three types of towers that will principally be considered are the **BEMC** referred to in this paper as **towers**, and the two types of **SMD strips**, **eta** and **phi**. The BEMC towers are square, and they are the detectors initially searched to determine if any particular detected particle constitutes an event. However, they occupy a relatively broad region of so-called *eta-phi space*. Therefore, to provide greater angular resolution for the detection of particle energy deposition, the SMD tower has strips in both eta and phi. **Eta** is the pseudo-rapidity, which is related to the polar angle to the beam-direction axis and ranges from -1 to 1. **Phi** is the angle of azimuth around the beam direction, which ranges from -π to π. The **StEmcGeom()** class provides information about the physical arrangement of these specific towers, so one may determine the relative spatial positions of the detected particles, and also find neighboring strips/towers. This is necessary as most of the other data-accessing classes only sort towers by ID, which is not necessarily spatially chronological.

StPicoEmcTrigger() – This class, the pointer for which is obtained form **StPicoDST()**, stores information about BEMC towers which registered a particle hit above a certain energy threshold. Although this class does not store the pointers to the associated SMD eta and phi hits itself, it does identify which eta and phi hits kept within the **StPicoDST()** class are associated with its' particular tower.

StSmdEHit() & **StSmdPHit()** – These classes, which serve a similar function to the class above, store hits registered on the SMD-strips and deemed important enough to data analysis by **PicoDSTMaker()** algorithms to keep within the event. **PicoDSTMaker()** makes these decisions by keeping data from strips only within a certain radius of a BEMC trigger, thereby saving storage space. Because SMD-Eta and -Phi strips are made smaller to provide better angular resolution in eta-phi space, there are over 18,000 of them within the entire detector barrel. Therefore, and also because only a relatively limited number of researchers use them for data analysis, there was a

strong motivation among the designers of the PicoDST format to reduce the number of SMD-strips kept to be only those in the vicinity of a BEMC trigger.

2.4 ROOT Analysis and Classes

Root is a powerful software useful for analysing enormous quantities of data. The form that this data analysis comes in is often graphical; The statistical nature of particle- and high-energy nuclear physics often mans this comes in the form of histograms, either 1 or 2 dimensional.

As mentioned above, the output *.root* files still contain large amounts of relatively 'raw' data, and oftentimes they must still be processed or manipulated to yield useful experimental results. One very useful and versatile example is the 2D histogram, which in reality, despite its name, has three degrees of freedom in which to express statistical results, especially in 'heat map' configurations. This is demonstrated in Figure 2.1. Figures of this type are common in ROOT-type analysis because they can take full advantage of the sheer volume of the data collected from many different experimental iterations.

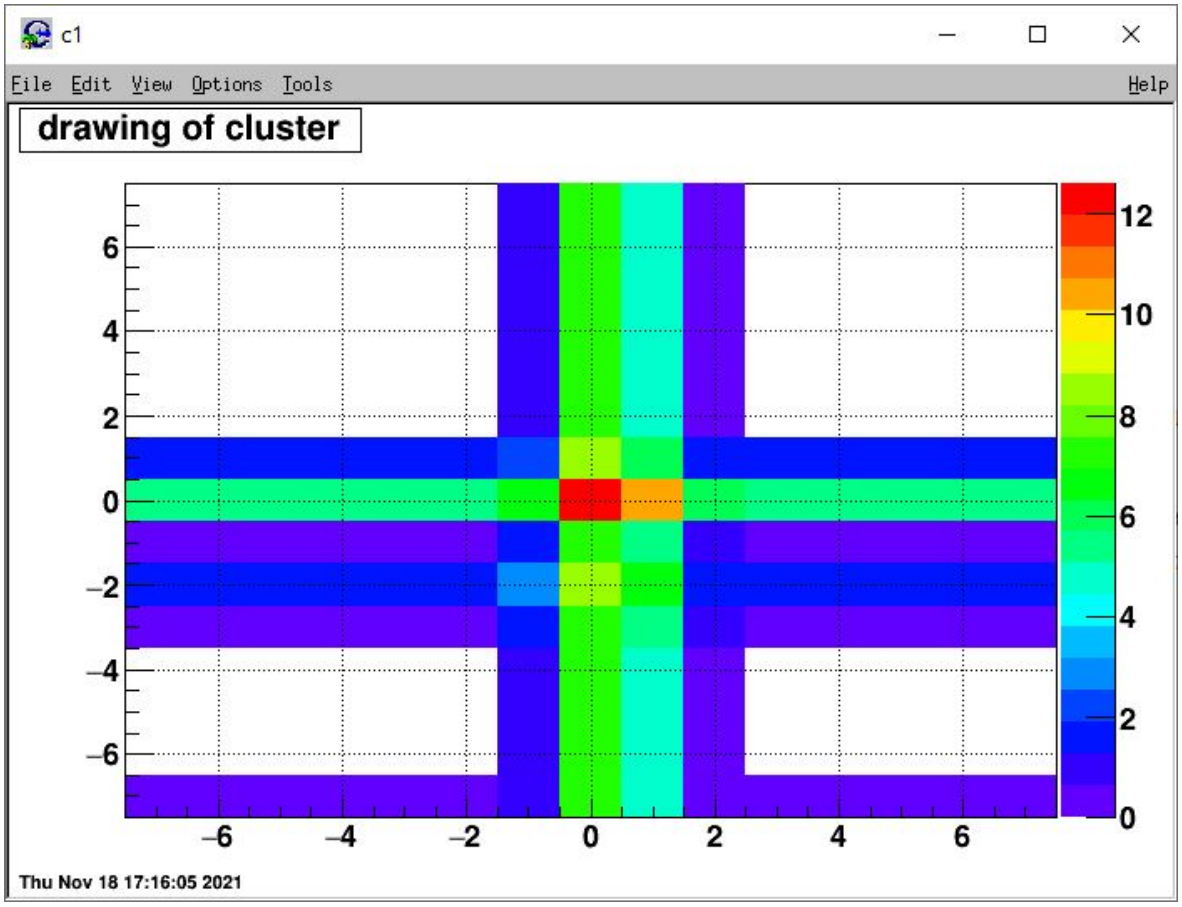

Figure 2.1: The SMD Eta and Phi strips located in the vicinity of tower 429 of event 338. This is an example of a 2D 'heat map' style histogram that can be produced directly from *.root* output files. The axes represent each eta- or phi-strip's position, 0 representing the central strips. Horizontal strips are Phi-strips, while the vertical are Eta-strips. The color axis is in units of GeV.

2.5 Specific Structure of the Source Code – MuDST

For this particular report, the research concerned one piece of analysis software in particular. The principal function of this code, called the **JetTreeThridMaker**, is to cluster certain types of 'hits' found in raw data files (originally MuDST, now PicoDST) that pass certain criteria. This criteria includes, but is not limited to, the total amount of energy registered on the BEMC and SMD detectors that is 'associated' with the hit, as well as the type of particle that the hit is associated with. this 'association' mentioned above is a fundamental function of the JetTreeMaker

code. Many of the algorithms contained therein are designed to automatically group this data into a collection called a **cluster**. The simultaneous energy measurements made by a series of neighboring detectors is *clustered* when it meets this aforementioned criteria, this means it is considered to have been caused by a single particle. For this particular code, the particles of interest are **photons** and **neutral Pions**, often notated γ and π^0, respectively. The code processes some of the SMD sensor data for the explicit purpose of determining which of these two particles has been detected, a process which will be described in greater detail in a later section.

As mentioned above, the starting place of the source code is the *Macro*, which is usually the first piece of code invoked by the user to begin the full execution of the code. Technically, a script is usually used to actually run the Macro, which is not strictly necessary, but it saves the user from having to manually open root and run the macro themselves. Depending on what the user wants to do with their code, they can either submit their code to the BNL servers to be executed on any available CPU in batch mode, or run the macro on a local root terminal in their own directory. Submitting the code to be run on the BNL servers is more efficient for processing many raw data files at once. If one desires to test changes made to their source code, however, it is much more time-efficient to run the code on a local root-terminal, with only one raw data file. This is because the BNL servers are more efficient at processing the raw data files themselves, and they will continue to run even if the user is no longer logged in, but users must wait in a queue for their submitted files to be processed.

Because the focus of this paper is the function and development of the code itself, the data presented here will be exclusively generated from local root terminals with one data file. The data generated thereby is more than sufficient for the analysis of the code's function, as the raw data file used contains more than 500 events, each of which represents a separate iteration of the analysis algorithms in question.

First, the general structure of the Macro used for the **MuDST JetTreeThirdMaker** is enumerated below:

1. The macro begins by enumerating the different star and root4star libraries that will be nec-

essary for analysis, like StChain, StMuDSTMaker, and StEventUtilities.

2. The next step in the analysis is the initialization of the **StChain**, which serves the purpose of running all *makers* on each event, including StJetTreeThirdMaker. Each event contributes data to the **TTree** of the output file. Each 'maker', including the ThirdMaker, are necessary to generate a usable, complete TTree.

3. Once all necessary makers have been added to the StChain, the iteration begins. For every event that can be found in the data file, all makers are run, and the ThirdMaker fills out the TTree of the output file with the data generated form the raw MuDST File.

4. Finally, the **Finish()** method of the StChain is called, closing all makers and enabling the completed output file to be read for further analysis.

Next, the structure of the **MuDST ThridMaker source code** is described below:

1. Although it wont be covered in explicit detail here, there is a header file, **StJetTreeThirdMaker.h** that is very important to the actual source code file, **StJetTreeThirdMaker.cxx**. The header file contains declarations of the classes whose function is defined by **StJetTreeThirdMaker.cxx**, as well as the declaration of some important variables that will be referenced throughout the code. The actual data processing and analysis, however, occurs exclusively within the source code, or *.cxx* file.

2. Inside the actual **StJetTreeThirdMaker.cxx** file itself, the file starts with **#include** statements for both standard C++ libraries in addition to the declaration of several STAR-specific classes that will be used later on.

3. The next part of the code defines several useful classes, such as the constructor, StJetTreeThirdMaker(), as well as Init() among a few others. These particular classes contain little more than the definition of a few useful variable and objects that will be used elsewhere, but they are vital to the inclusion of StJetTreeThirdMaker as part of the StChain.

4. The **Make()** class is the primary focus of this paper, out of all the classes within StJetTreeThirdMaker.cxx. It performs the bulk of the analysis in terms of clustering and saving data to the TTree. Figure 2.2 provides a graphical summary of the MuDST-version Make() function.

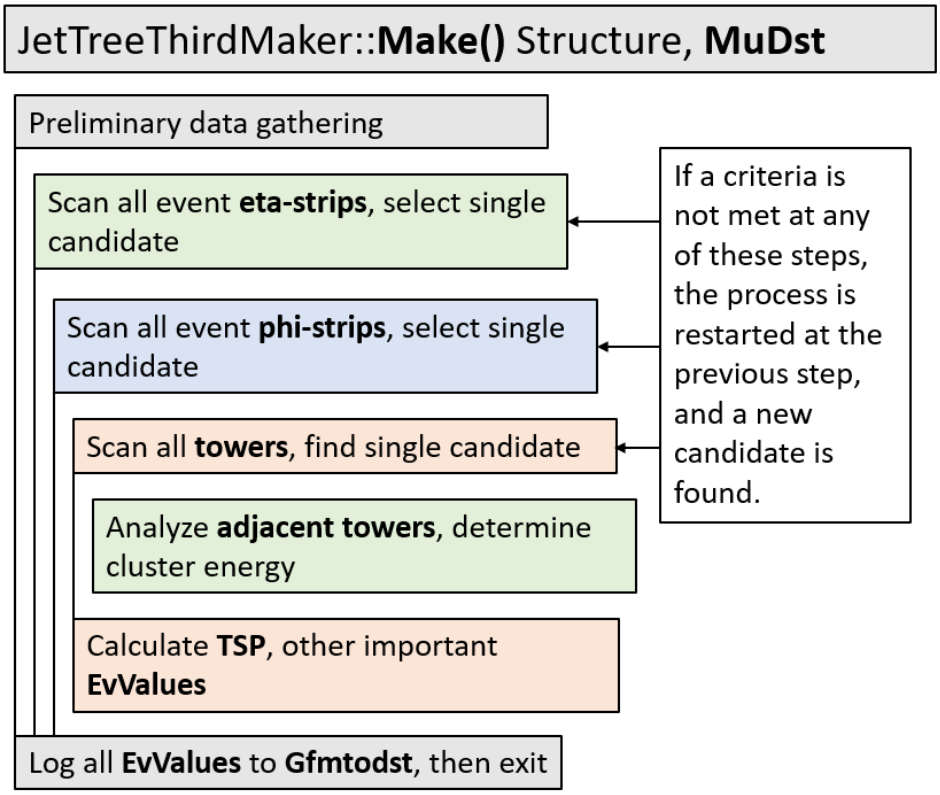

Figure 2.2: A diagram of the MuDST-JetTreeThirdMaker source code's structure, specifically the **Make()** main algorithm. Each horizontal level represents one or more layers of nested loops within the code.

5. Make() is broken into many stages, the first several of which are concerned exclusively with successfully obtaining a StMuDST pointer. This class is where most of relevant raw data will be obtained from, alongside StMuEvent.

6. After Make() determines that the data within the current event is accessible, It extracts some miscellaneous data about the event itself, including the run number, event number, checks

on towers and other detectors (this is primarily to ensure that the data contained in this event is reliable.) It then stores some of this data in the **EvValues array**, which will later be transferred directly to the TTree of the output file.

7. The algorithm within Make() then starts its main analysis phase. This begins with the Eta-SMD strips These strips are the starting point of finding a valid cluster that will be saved later on, the code scans over each strip, scanning the energy of each one and saving them chronologically in an array. Specifically, the ThirdMaker code scans over each **module**, of which there are 120 in total. This is done because the pointer to strip data (such as energy and position) are indexed under their specific module number within the relevant class, in MuDST. Each module keeps a list of 'hits' kept on the SMD detectors, and each 'hit' is itself a pointer to the relevant data, such as the measured energy of the hit, and specific strip which measured it.

8. Once a hit with a sufficiently high energy is found, the code checks all of the neighboring strips to verify that the 'central strip' is indeed the one with the highest recorded energy in its vicinity. Once this is done, and a few other energy thresholds are passed (to ignore low-energy hits, below about 0.5 GeV.) the Phi strips are analyzed next.

9. A similar process to the one described above is repeated for the Phi strips. All the hits for each module are loaded into an array, organized spatially, and scanned to find those with the highest energy. Of these, the strip with the highest energy is checked to ensure it has the highest in its own vicinity, and for the Phi-strips, it must be verified that its position within eta-phi space is within about a tower's width to the *central* eta strip found in the previous step.

10. Once the phi strips have been selected and recorded, along with those that are adjacent, the code checks the tower hits recorded for the same module for which the eta and phi strips are found, and finds a BEMC-hit whose associated tower is within a certain radius in eta- and phi-space. The code then loops over all towers in each module again, to run some checks on

the *central* tower's id to ensure that the hit found there is valid, as some towers may falsely register hits, called **hot towers**.

11. The code then enters what will be refereed to in this paper as the **adjacent tower algorithm**. The purpose of this algorithm is to analyze the adjacent towers, and determine if they should be included in the total **cluster energy**. This is done if the central Eta and Phi strips are near the edge of the central tower, meaning that some of the incident particle's energy may have 'spilled over' into an adjacent tower. this is the final step in the *clustering* section of the algorithm, in which the data from various neighboring detectors (both BEMC and SMD) is gathered.

12. From here, the TSP is finally calculated. The TSP is a measure of the spatial distribution of energy deposited by the particle which caused the hit, described by Equation Eq. 2.1.

$$TSP \equiv E_c / \Sigma E_i r_i^{1.5} \qquad \text{(Eq. 2.1)}$$

In which $\mathbf{E}_c$ is the tower cluster energy, and ($\mathbf{E}_i$ $\mathbf{r}_i$) is each eta and phi SMD energy, multiplied by its distance to the center of the cluster raised to the 1.5^{th} power [8]. Depending on the value of this quantity, later analysis can determine if the particle in question was either a photon or a neutral Pion.

13. Finally, the **EvValues array** (which corresponds directly to **GmftoDST**) is filled. This is the step in which the data is finally deposited into the TTree, which is the ultimate final product of the ThirdMaker code.

14. There are several more smaller class definitions lie after Make() in StJetTreeMaker.cxx, but they generally serve smaller roles assisting the analysis described above.

Note that the above code structure applies only to the MuDST code. The parts of the PicoDST format that were fundamentally changed necessitated that some of the above analysis

steps be altered, or re-ordered completely. Those changes are described in general detail below, in terms of what is changed from the above steps:

1. The Macro remains essentially unchanged for PicoDST, save for a few differences with the format of the StChain when used with PicoDST, and they way events are iterated, but there is little practical difference in terms of performance between the two macros.

2. As with the macro, the first sections of the updated PicoDST version of the source code ThridMaker file are nearly identical to the MuDST version, save for a few declarations of classes that are necessary to use with PicoDST (and had not been with MuDST.)

3. The Make() function is where the structures of the two different ThirdMaker versions diverge. This is shown graphically in Figure 2.3 The first part of the Make() method is generally the same, in which the code checks to ensure that vaild pointers to the relevant classes have been obtained. After this step, however, the Hits are processsed very differently.

4. Because ThirdMaker makes extensive use of SMD strips, its clustering algorithm is affected greatly by the fact that PicoDST contains far less SMD strip data than MuDST. Also, the MuDST code stores SMD-hit information module-wise, meaning that each module stores hit pointers independently.

   PicoDST, however, stores pointers to SMD-hit data event-wise, meaning for the fewer SMD-hits its does keep, they are not sorted by module. Instead, they all stored together and 'indexed', meaning they are assigned an integer number corresponding to their arbitrary placement within the array of all SMD-hit pointers. MuDST also stores tower-hits module-wise. PicoDST, howerver, stores hit information as 'triggers', which means BEMC hits that pass a certain minimum energy threshold. A variety of different types of data can be accessed from StPicoEmcTrigger, mainly in the form of pointers to *other* classes with more specific information. For example, it enables the ThirdMaker to access pointers to BEMC-hit data, and SMD hit data, by recording the 'index' (or placement in the overall list of all hits) of the specific BEMC and SMD hits associated with a particular Trigger of interest.

For example: One Emc Trigger returns the tower Id with which it is associated, and also the **indicies** of nearby SMD hits; With this information, the ThirdMaker can tell which SMD-hits are good candidates to include in its hit-clustering.

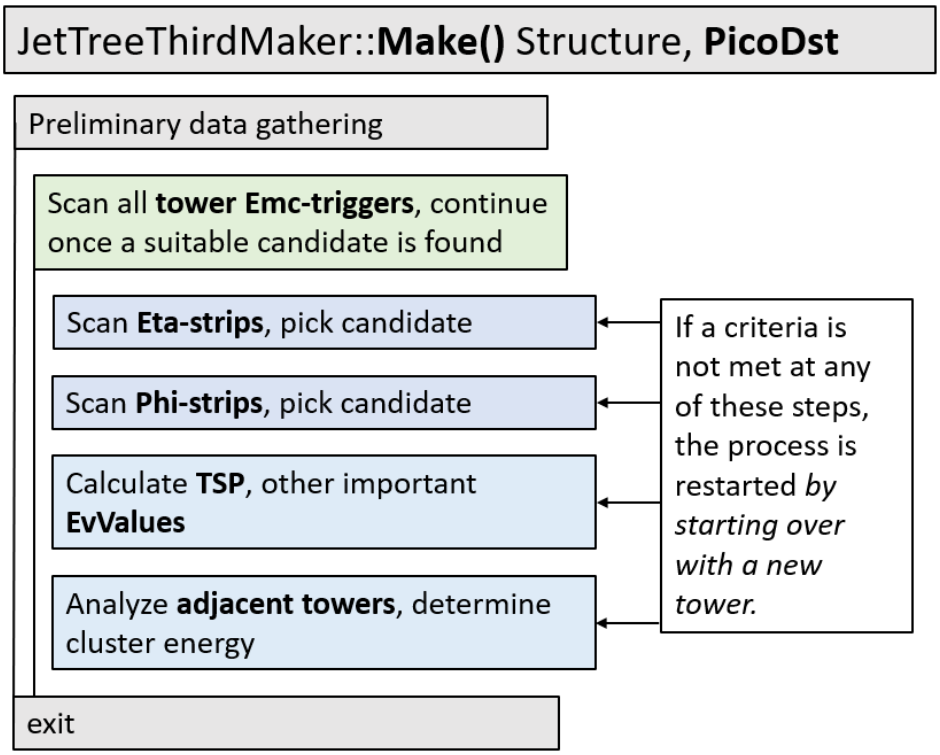

Figure 2.3: A diagram of the PicoDST-JetTreeThirdMaker source code's structure, specifically the *Make()* main algorithm. Each horizontal level represents one or more nested loops within the code.

The consequences of this last point are largely responsible for the structural differences between the clustering algorithms of the MuDST and PicoDST versions of the ThirdMaker. The essential point it this: MuDST stores all SMD-eta, SMD-phi, and BEMC hits in a hits class, each segregated by module. The MuDST ThirdMaker code must, therefore, manually associate eta-SMD strips, phi-SMD strips, and a BEMC tower into one cluster before further analysis can continue.

PicoDST, However, by use of the StPicoEmcTrigger class, **already associates** SMD-eta and -phi strips with a particular BEMC hit, and also stores information about energy (and also which energy thresholds it meets.) All that is left to do, therefore, is retrieve the data on these strips and decide which ones ought to be kept depending on the specific needs of any arbitrary analysis being made. This not only saves time and space in the execution of ThirdMaker source code, but eliminates SMD data that is not necessary for analysis, on the basis of being too far from any meaningful BEMC-hit to be useful.

2.6 Obtaining Data for the Comparsion of MuDST and PicoDST

There are several metrics by which MuDST and PicoDST-based analysis programs can be compared. The most obvious would be to compare them in terms of time-efficiency, in which the time each program takes to perform the same analysis is compared, either by a local root terminal or by the submitting the job to the BNL servers, and then using ROOT with the resultant data to generate a histogram to compare time used as a function of event quantity, or any other arbitrary metric.

In this paper, a single raw data file, identical save for the fact that it exists in both MuDST and PicoDST formats, will be used. The Event contains 473 events, which should be more than sufficient for drawing conclusions about the time-efficieny of the MuDST and PicoDST version of the ThirdMaker algorithm. The efficiency of the thirdMaker algorithm alone will be considered, as the time consumed by the respective marcos is not as large of a practical concern for bulk data analysis, as it consumes very little time between each event, and requires only seconds to initiate the ThirdMaker analysis once a new raw data file is loaded.

The **TStopwatch()** class is used to measure elapsed time, specifically, the **TStopwatch():: RealTime()** method. This measures the real time elapsed between the initial invocation of an instance of TStopWatch(), and when the RealTime() method is called. This is facilitated by each code version's respective macro, and the elapsed time between each iteration of the ThirdMaker algorithm is measured and stored.

It is important to note that the PicoDST version of the ThirdMaker algorithm is not entirely

finished. However, each vital, substantive function of the MuDST ThirdMaker has a functional counterpart in the PicoDST code. The differences that remain between the two are only in small discrepancies in output data, but it can be said with confidence at present that any final changes that will be made to the PicoDST ThirdMaker algorithm have very little chance to significantly effect the time required by the algorithm to complete its analysis. The MuDST algorithm, however, is considered complete.

Once the time taken by each ThirdMaker to complete each event is measured, the times are matched for each event and plotted. This is displayed in the following section, with Figure 3.1.

3. RESULTS

The results of the efficiency analysis described in section 2.6 are shown in Figure 3.1.

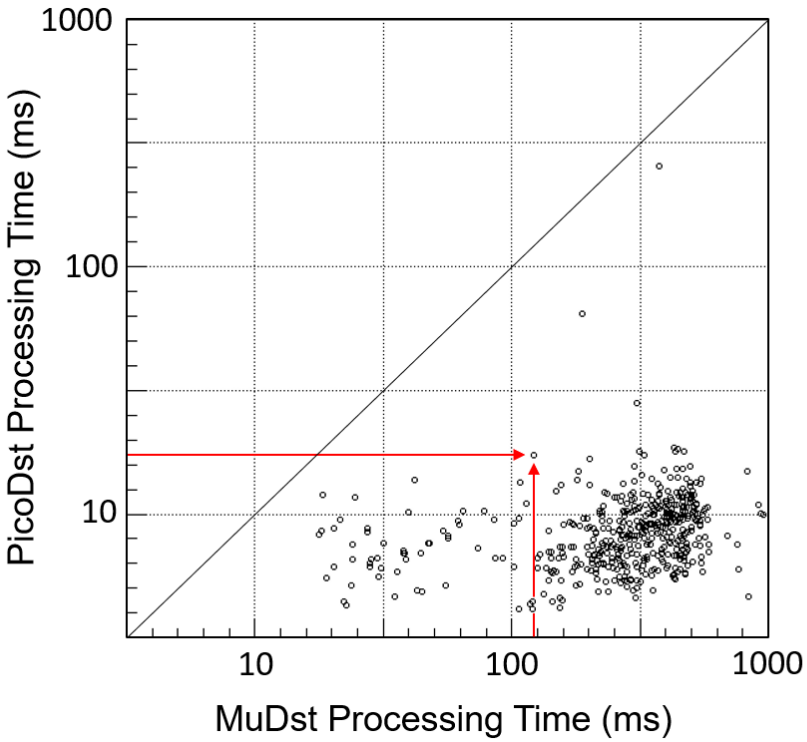

Figure 3.1: A comparison between the time taken to process Events by the MuDST and PicoDST JetTreeThridMaker algorithm. Each point represents a distinct event, with its x-coordinate representing the MuDST processing time, and y-coordinate representing PicoDST processing time, all on a logarithmic scale. 474 events are represented in this sample.The significance of the black and red annotations are explained below.

The **black diagonal** line on Figure 3.1 represents the Equal-time line, meaning that any event lying beneath it took less time to process with PicoDST than with MuDST. As it can be seen on the graph, each measured event took less time to process with the PicoDST version.

The **red arrows** serve as an example for interpreting the elapsed times of each event. The horizontal arrow, which can be used to determine the PicoDST-elapsed time of the arbitrary event,

indicates a measured time of just above 10 ms. The vertical arrow indicated a MuDST-elapsed time of just over 100ms. This is a good estimate of the actual values for this event, which happen to be **17.4 ms** for PicoDST, and **121.3 ms** for MuDST.

Table 3.1: Statistical Results of the Efficiency Test

Version	**Mean**	**1-σ range**	**1-σ factor**
PicoDST.	8.39 ms	12.0 ms - 5.89 ms	1.43
MuDST.	255 ms	116 ms - 558 ms	2.19

Because the data is presented here in a logarithmic format, the **1-σ factor** is meant to illustrate the relative spread of the two data sets. What it represents is the factor by which the mean can be divided or multiplied to obtain either end of the 1-σ distance to the mean.

As can be seen in Table 3.1, the difference in the time both versions take to execute ThirdMaker are quite large, nearly as large as 2 orders of magnitude for some particular events.

This can likely be attributed to the fact that the PicoDST raw data format already does much of the association steps that MuDST must do on its own. A great portion of the MuDST algorithm is concerned with gathering data from disparate sources and clustering it into more useful groups, while this is already done beforehand with PicoDST.

It should also be noted, presumably for the same reasons listed above, that the standard deviation of measured times for the MuDST data is much larger than for PicoDST.

4. CONCLUSION

For reason elaborated in particular in Section 2.5 there are substantial structural differences between the two code versions which are likely responsible for the measured time differences observed in Figure 3.1. These differences are substantial, the mean values alone approximately 1.5 orders of magnitude apart. Not only has the structure of PicoDST presumably saved disk space, which can be evaluated in further research, but it has very clearly afforded time savings, at the very least in the performance of the ThirdMaker algorithm.

The PicoDST software discussed in this paper was created in the course of this research project, and it will be used to perform the same data-analysis tasks that the MuDST previously had, as new data produced for the STAR experiment will generally only be available in the PicoDST format going forward. As discussed above, the results of this paper support that this newer software is able to perform the same analysis in less time, and with smaller raw data files.

Topics of further research would include a full-scale application of both ThirdMakers to a very large quantity of bulk data, including many distinct raw data files, that could be measured and submitted to the BNL servers for bulk processing. Also not explicitly measured in this report is the **elapsed CPU-time**, which TStopwatch() is also capable of measuring, but it is reasonable to assume that such an comparison conducted in a similar way would yield similar results, as CPU-time elapsed could not possibly be longer than real-time elapsed.

Another topic of interest, which was a central motivation to the creation of the PicoDST format, is the file size of each data format. It is reasonable to assume that PicoDST uses less, by virtue of its much smaller SMD strip data storage, among other factors, but the exact degree to which this is true is not directly measured in this paper.

The generally greater simplicity of PicoDST, and the smaller amount of time it needs to perform the same analysis, could mean that researchers can develop and test their own makers on a shorter timescale. Even disregarding the results presented above, it was observed anecdotally

over the course of conducting this research that the MuDST ThirdMaker is more cumbersome to test, by virtue of the fact that it takes much longer perform the same analysis. Therefore, the results quantified by this paper are verified, if only anecdotally, to have an effect on the ability of researchers to develop and test MuDST-based makers. Although it should be noted that this may not necessarily be true of MuDST-based makers in general, rather just MuDST-ThirdMaker.

REFERENCES

[1] S. A. Bass, M. Gyulassy, H. Stoecker, and W. Greiner, "Signatures of quark gluon plasma formation in high-energy heavy ion collisions: A Critical review," *J. Phys. G*, vol. 25, pp. R1–R57, 1999.

[2] K. Ackermann *et al.*, "Star detector overview," *Nucl. Instrum. Meth.*, vol. 499, no. 2, pp. 624–632, 2003. The Relativistic Heavy Ion Collider Project: RHIC and its Detectors.

[3] J. Adams *et al.*, "Experimental and theoretical challenges in the search for the quark gluon plasma: The STAR Collaboration's critical assessment of the evidence from RHIC collisions," *Nucl. Phys. A*, vol. 757, pp. 102–183, 2005.

[4] Nigmathkulov, "Picodst: status and step forward.." Web, 2018.

[5] Y. Fisyak, "Picodst issues." Web, 2018.

[6] L. Adamczyk *et al.*, "Jet-like Correlations with Direct-Photon and Neutral-Pion Triggers at $\sqrt{s_{NN}} = 200$ GeV," *Phys. Lett. B*, vol. 760, pp. 689–696, 2016.

[7] "About root." Web. *root.cern.ch*.

[8] S. Mioduszewski, "Towards a γ+jet measurement in star." Web, 2016.