ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
"МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ
М.В.ЛОМОНОСОВА"

ФИЛИАЛ МГУ в г. Дубне

Профиль: Фундаментальная и прикладная ядерная физика

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

"Организация хранения и обработки физических данных эксперимента SPD"

Выполнила студентка:
Будтуева Замира Алановна

_____
(подпись студента)

Научный руководитель:
академик РАН,
доктор физ.-мат. наук
Трубников Григорий Владимирович

_____
(подпись научного руководителя)

Научный консультант:
кандидат физ.-мат. наук
Прокошин Федор Валерьевич

_____
(подпись научного консультанта)

Допущена к защите:

_____
(дата)

Руководитель образовательной
программы:

_____
(подпись руководителя ОП)

г. Дубна
2026 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Физика высоких энергий нацелена на исследование фундаментальных свойств материи, элементарных частиц и механизмов их взаимодействий. Эксперименты в этой области проводятся на ускорительных комплексах, оборудованных сложными системами детектирования для регистрации продуктов столкновения частиц. Полученные данные используются для проверки теоретических моделей и поиска новых физических явлений.

Одним из нерешённых вопросов в области физики высоких энергий является вопрос о распределении спинов нуклонов между их составляющими – «спиновый кризис». Результаты эксперимента EMC (ЦЕРН, 1980-е гг.) [1] показали, что вклад спинов кварков составляет лишь часть полного спина нуклона, а значительная часть зависит от спина глюонов и орбитального движения партонов. Изучение спиновой структуры является важной задачей в современной физике, так как спин определяет фундаментальные свойства природы, но наши знания о его внутренней структуре ограничены, особенно в части вклада глюонов.

Решение задач спиновой физики требует проведения экспериментов с использованием высокоточных детекторных систем. Одним из таких экспериментов является эксперимент SPD (Spin Physics Detector) [2], реализуемый на ускорительном комплексе NICA (Nuclotron-based Ion Collider fAcility) [3]. Основной целью SPD является изучение спиновой структуры протона и дейтрона, а также других связанных со спином явлений в столкновениях поляризованных пучков при энергии $\sqrt{s} = 27$ ГэВ и светимостью порядка 10^{32} см$^{-2}$сек$^{-1}$.

При проведении подобных экспериментов детекторы регистрируют большое количество физических событий, каждое из которых представляет собой набор данных о взаимодействии частиц в системе. В эксперименте SPD предполагается использование бестриггерной системы сбора данных, которая будет обеспечивать регистрацию максимально возможного количества событий. Такой подход приводит к формированию больших

массивов информации, которые требуют разработки систем эффективной организации хранения и обработки данных физических событий. Подобные системы обеспечивают быстрый доступ к нужной информации, выполнение физического анализа и эффективное использование вычислительных ресурсов.

Целью работы является создание программного обеспечения для индексирования файлов данных эксперимента SPD в рамках разработки Event Index – каталога физических событий, как моделированных, так и полученных с экспериментальной установки.

В рамках поставленной цели были выявлены следующие **задачи**:

1. Ознакомиться с целями и задачами эксперимента SPD;
2. Рассмотреть системы хранения и обработки данных SPD;
3. Изучить программное обеспечение для обработки и анализа данных эксперимента;
4. Оценить требования к хранению и поиску событий для использования в SPD Event Index;
5. Разработать прототип программных модулей для формирования Event Header, содержащего идентификатор событий и служебные метаданные;
6. Отработать методики записи и чтения файлов с добавленной идентификационной информацией событий.

Объектом исследования является система Event Index – каталог физических событий эксперимента SPD, предназначенный для индексирования, поиска и локализации моделированных и регистрируемых установкой событий в распределённой системе хранения данных.

Предметом исследования являются методы и программные средства формирования, записи и чтения идентификационной информации событий (Event Header), а также подходы к загрузке проиндексированных данных в центральное хранилище Event Index, обеспечивающих производительность при ожидаемом объёме данных эксперимента SPD.

Глава 1. Построение эксперимента в физике высоких энергий

1.1. Физические основы взаимодействий частиц при высоких энергиях

В основе теоретических моделей физики элементарных частиц лежит Стандартная модель. Она включает 6 кварков, 6 лептонов, 12 калибровочных бозонов (фотон, бозоны W^+, W^- и Z^0, а также 8 глюонов) и бозон Хиггса. Стандартная модель описывает три типа фундаментальных взаимодействий: сильное, слабое и электромагнитное (гравитационное взаимодействие в её рамки не входит). Дальнейшие знания можно получить, пытаясь понять, что происходит при более высоких энергиях (соответствующих меньшим расстояниям), где могут быть обнаружены новые частицы или выявляться несоответствия в Стандартной модели.

Каждое фундаментальное взаимодействие характеризуется своей константой связи α_i, которая определяет его интенсивность. Однако при исследовании процессов во всё более высоких энергетических областях было установлено, что эти константы не являются строго постоянными величинами, а зависят от масштаба энергии взаимодействия [4]. Гравитационное взаимодействие в данном случае (Табл. 1) не рассматривается, поскольку его константа чрезвычайно мала: при энергии порядка E = 1 ТэВ она составляет примерно 10^{-34}.

Энергия, ГэВ	**Константа сильного взаимодействия α_s**	**Константа электромагнитного взаимодействия α_e**	**Константа слабого взаимодействия α_w**
0,01	10	1/137	-
0,1	1	1/135	1/27
1	0,40	1/133	1/28
100	0,12	1/128	1/30

Таблица 1. Зависимость констант взаимодействий от энергий [4]

Уменьшение константы сильного взаимодействия α_s при увеличении энергии связано с увеличением антиэкранировки цветового заряда. Именно

этот эффект приводит к асимптотической свободе на малых расстояниях, то есть при уменьшении расстояния между цветными кварками, взаимодействие между ними становится слабее. В отличии от электромагнитного случая, глюоны сами несут цветовой заряд и взаимодействуют не только с кварками, но и друг с другом. В результате распределение цветового поля вокруг кварка существенно отличается от распределения электрического заряда вокруг электрона. Цветной кварк оказывается окружён в основном цветовым зарядом того же типа. Поэтому пробный цветовой заряд, приближаясь к кварку, проникает внутрь этого облака, и эффективная величина измеряемого цветового заряда уменьшается.

При этом для сильного взаимодействия характерна противоположная особенность: при увеличении расстояния между кварками, начиная примерно с масштабов больше 10^{-13} см, сила их связи возрастает. Это явление называется конфайнментом и объясняет невозможность наблюдения свободных кварков.

Для электромагнитного взаимодействия ситуация иная. В квантовой электродинамике (КЭД) заряд электрона экранируется за счёт поляризации вакуума. Электрон может испускать виртуальные фотоны, которые, в свою очередь, порождают виртуальные электрон-позитронные пары. Позитрон из такой пары отталкивается от исходного электрона, тогда как электрон притягивается к нему. В результате на больших расстояниях наблюдаемый заряд оказывается частично экранированным. При переходе к меньшим расстояниям, то есть при увеличении энергии процесса, экранирование ослабевает, и эффективная константа электромагнитного взаимодействия возрастает. Наиболее заметный рост с увеличением энергии наблюдается у константы слабого взаимодействия.

В экспериментах на адронных коллайдерах особую роль играет сильное взаимодействие, поскольку сталкивающиеся протоны являются составными объектами, состоящими из кварков и глюонов. Поэтому характер протон-протонного взаимодействия существенно зависит от переданного

импульса, то есть от пространственного масштаба, на котором исследуется внутренняя структура протона.

При малых переданных импульсах (в мягких процессах) протоны взаимодействуют друг с другом как сложные составные объекты. В этом случае описание процесса с помощью теории возмущений квантовой хромодинамики (КХД) неприменимо, поскольку константа сильного взаимодействия велика. Напротив, при больших переданных импульсах взаимодействие происходит на малых расстояниях, где, вследствие асимптотической свободы, кварки и глюоны взаимодействуют сравнительно слабо. Это позволяет использовать пертурбативный подход.

В жёстких процессах частица, которой обмениваются взаимодействующие объекты, имеет большой импульс и, следовательно, малую длину волны. Благодаря этому становится возможным «разрешить» партонную структуру адронов. Поскольку в таких условиях величина α_s уменьшается, взаимодействие можно описывать в рамках теории возмущений. Основной вклад при этом дают процессы низших порядков, например, взаимодействия типа $2 \to 2$, в результате которых в конечном состоянии возникают два высокоэнергетичных партона. Достаточно большие энергии сталкивающихся частиц также необходимы для рождения тяжёлых частиц, например, бозона Хиггса [5].

Важно учитывать, что наряду с жёстким масштабом, задаваемым переданными импульсом, в процессе присутствует и мягкий адронный масштаб, связанный с величиной $\Lambda_{\text{кхд}}$. Непертурбативная составляющая процесса входит в описание через партонные функции распределения в протоне. Эти функции считаются универсальными, то есть не зависящими от конкретного рассматриваемого процесса.

Для разделения пертурбативной и непертурбативной частей используется теорема факторизации [6, 7]. Согласно этому подходу, наблюдаемые величины, в частности сечения процессов, можно представить как свёртку партонных функций распределения в сталкивающихся протонах

и сечения соответствующего жёсткого партонного подпроцесса, вычисляемого методами пертурбативной КХД.

На практике часто применяется схема коллинеарной факторизации. В этом приближении поперечными импульсами партонов внутри протонов пренебрегают по сравнению с их продольными импульсами. Например, инклюзивное рождение бозона Хиггса H в протон-протонных столкновениях в рамках коллинеарной факторизации [8] может быть записано в виде:

$$\sigma(pp \to H + X) =_{a,b} \sum \int dx_1 dx_2 f_a(x_1, \mu_F^2) f_b(x_2, \mu_F^2) \hat{\sigma}_{ab}(x_1, x_2, \mu_R^2, \mu_F^2) \quad (1)$$

Здесь суммирование идёт по типам взаимодействующих партонов a и b, f_a и f_b – функции распределения партонов в протоне, $\hat{\sigma}_{ab}$ – сечение жёсткого партонного подпроцесса. μ_F – масштаб факторизации, отделяющий мягкую непертурбативную часть подпроцесса от жёсткой пертурбативной компоненты, а μ_R – масштаб перенормировки, связанный с бегущей константой сильного взаимодействия α_s. Выбор оптимальных значений μ_F и μ_R важен, поскольку от него зависят теоретические неопределённости расчётов. В данной формуле учитываются взаимодействия партонов различных типов, включая глюоны и кварки, а функции распределения описывают вероятность найти в протоне партон заданного типа с определённой долей продольного импульса.

Сечения партонных подпроцессов определяется вероятностью взаимодействия партонов a и b. В свою очередь, вероятность обнаружить внутри протона кварк или глюон, несущий зарядную долю продольного импульса, задаётся партонными функциями распределения. Важно отметить, что сами функции распределения партонов не могут быть вычислены в рамках пертурбативной КХД. Однако их изменение при переходе от одного масштаба факторизации к другому описываются уравнением эволюции партонных распределений, в современной форме известным как уравнение Докшицера – Грибова – Липатова – Альтарелли – Паризи, или уравнение ДГЛА [9].

1.2. Физические цели экспериментов и требования к детекторным системам

Физика высоких энергий – это направление экспериментальной физики, которая изучает фундаментальные составляющие материи и закономерности их взаимодействий путём экспериментального наблюдения за реакциями частиц. Так как пространственный масштаб, который можно измерить, определяется длиной волны де Бройля ($\lambda = h/p$), наличие пучков частиц высоких энергий важно для изучения структуры вещества на малых расстояниях. В то же время импульс, возникающий при столкновении частиц, может быть использован для рождения новой частицы, отличной от тех, что были до столкновения. Это означает, что чем больше импульс или энергия столкновения, тем тяжелее будут образовавшиеся частицы. Потому используются частицы с высокой энергией. Развитие физики высоких энергий напрямую связано с постоянным повышением энергии ускорителей частиц.

Цели экспериментов в области физики высоких энергий, проводимых на ускорителях частиц, можно условно разделить на два класса:

1. Эксперименты, направленные на поиск новых частиц через преобразование энергии столкновения в массу частицы. Такие эксперименты дают возможность получить новую частицу в результате резонанса или излучения в конечном состоянии. Простейшим примером является резонансное состояние X при e^+e^--аннигиляции (Рис. 1а). В большинстве случаев, а то и во всех столкновениях, состояние X соответствует известной частице (частицам), например, $Z^{(*)}/\gamma$ может содержать вклад от новой частицы. Если такая новая частица вносит свой вклад, то инвариантная масса продукта распада, восстановленная с использованием энергии и импульса всех частиц, образовавшихся в результате распада X, может показать пик резонанса, который соответствует массе неизвестной частицы. Самый простой пример: если предположить, что X распадается на две частицы, мы можем наблюдать пик в спектре инвариантной массы этих двух частиц.

2. Эксперименты, которые нацелены на исследование природы взаимодействия и внутренней структуры частиц. Данные эксперименты могут быть связаны с косвенным определением вклада новой частицы во взаимодействие или обнаружением подструктуры элементарных частиц с помощью точных измерений (например, углов рассеяния). Примером являются процессы, в которых входящая частица сталкивается с другой, обмениваясь состоянием (Рис. 1б). В зависимости от характера обмена состоянием рассеяние даёт определённые предположения относительно углового распределения и распределения по импульсам частиц в конечном состоянии. Например, в случае кулоновского рассеяния пучка электронов на тяжёлой точечной мишени (например, ядре) дифференциальное сечение описывается формулой Резерфорда $d\sigma/d\Omega \propto 1/\sin^4(\theta/2)$, где θ – угол рассеяния по отношению к направлению падающего пучка в системе отсчёта, связанной с покоем мишени. Вклад новой частицы, обмен которой даёт дополнительный канал к фотонному, внёс бы небольшие отклонения от этой зависимости.

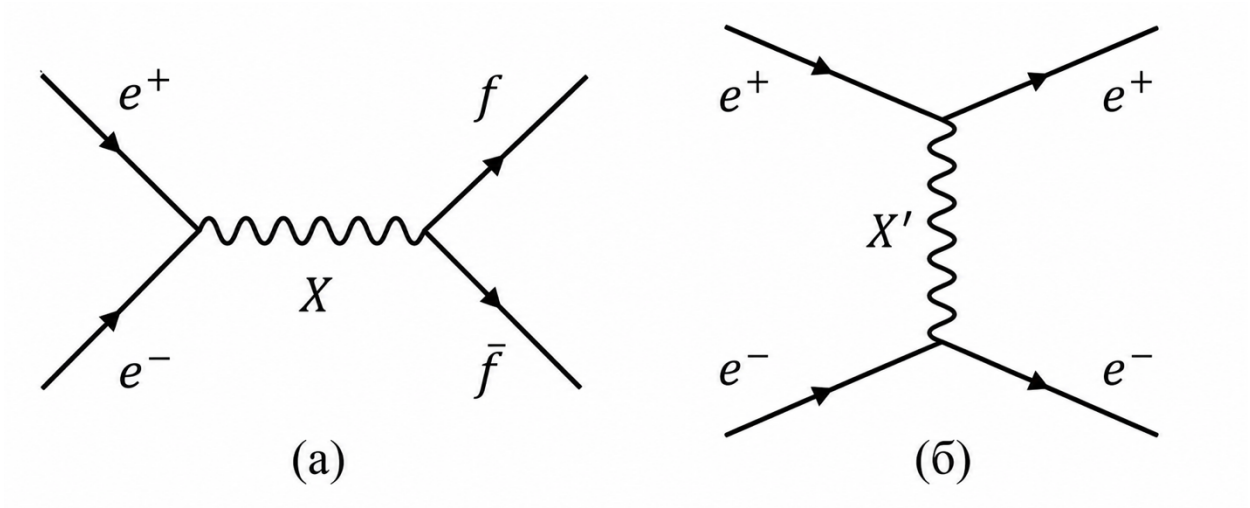


Рисунок 1. Фейнмановские диаграммы e^+e^- взаимодействий: а) аннигилируют друг с другом, образуя виртуальную частицу X (распадающуюся на пару частиц); б)обмениваются X' между собой [10].

Процессы $2 \to 2$ (Рис. 1) могут быть полностью реконструированы, если измерены импульсы и энергии двух входящих частиц. В реальных экспериментах число частиц, которые нужно измерить, бывает больше двух,

особенно для типичных высокоэнергетичных рассеяний. В эту группу входят продукты распада тяжёлых элементарных частиц (топ-кварки, W- и Z-бозоны), которые далее распадаются на множество частиц.

Ситуация усложняется в случае, когда среди вылетающих частиц есть нейтрино или другие неизвестные нейтральные частицы, которые слабо взаимодействуют с материалами детектора. Такие частицы всегда покидают детектор и зарегистрировать их напрямую на эксперименте почти невозможно. Единственный способ зафиксировать их – это измерить «недостающий импульс» с помощью закона сохранения четырёхимпульса. Чтобы получить информацию о конечном состоянии всех остальных частиц, детектор должен окружать точку взаимодействия с телесным углом покрытия, близким к 4π, что делает современные высокоэнергетичные коллайдерные эксперименты практически герметичными.

Плотность частиц в детекторе также является проблемой при высоких энергиях. По мере роста энергии столкновения средняя множественность рождённых частиц растёт логарифмически – как $\ln(s)$ или $(\ln(s))^2$, где s – квадрат энергии в системе центра масс. Это приводит к увеличению рождения частиц в малой угловой области, когда рождается и фрагментируется энергетичный кварк или глюон. Такие пучки частиц называются струнами или джетами. Их наличие требует, чтобы детектор имел малую сегментацию для идентификации пары частиц, рождённых близко друг к другу, как отдельных частиц.

Помимо этого, современные высокоэнергетические эксперименты должны иметь дело со многими различными типами стабильных частиц. Детектор должен очень точно измерять электроны, мюоны и фотоны. Высокоточное измерение энергии заряженных адронов или барионов особенно сложно. Идентификация видов адронов, таких как пион, каон и другие частицы, может быть желательной, если необходимо изучать цепочки распадов определённых мезонов.

Таким образом, современные детекторы на коллайдерах высоких энергий должны быть универсальными и работать со всеми видами стабильных лептонов и адронов. Они должны точно измерять известные процессы Стандартной модели и одновременно быть чувствительными к необычным сигналам, которые могут указывать на существование новых частиц. Важными признаками таких процессов являются b-кварки и τ-лептоны с большим импульсом. Например, b-кварк часто используется при изучении топ-кварка, так как топ-кварк почти всегда распадается на W-бозон и b-кварк. Такие частицы пролетают очень малое расстояние перед распадом, поэтому для их идентификации необходимы высокоточные трекинговые детекторы.

С ростом энергии столкновений сечение жёстких точечных процессов, таких как e^+e^- или партон-партонное рассеяние, по соображениям размерности убывает как $\sigma \propto 1/s$, то есть $\sigma \propto 1/E^2$ (где s- квадрат энергии в системе центра масс, E – энергия налетающей частицы при симметричных столкновениях). Значит, вероятность зарегистрировать интересующее взаимодействие подавлена множителем $1/E^2$. Этот эффект компенсируют ростом светимости (числа столкновений в единицу времени), которую увеличивают за счет более частых пересечений пучков, большей интенсивности и лучшей фокусировки.

Вместе с полезными сигналами рост светимости также увеличивает фоновые процессы. Важной проблемой становится наложение нескольких столкновений (pile up) [10]. Для его подавления нужны детекторы с высокими пространственными и временными разрешениями, а также эффективная система сбора данных.

Для точных измерений также необходимы хорошо настроенные симуляции эксперимента. Методы оценки фона по экспериментальным данным (data-driven методы) помогают уменьшить неопределённости при оценке фона, а современные статистические методы и машинное обучение повышают чувствительность анализа.

Глава 2. Обзор экспериментальной установки

Несмотря на успехи Стандартной модели в описании фундаментальных взаимодействий, внутренняя структура барионной материи остаётся предметом активных исследований. Протон и нейтрон являются основными составляющими атомного ядра, но до сих пор являются не полностью изученными объектами. Их свойства объясняются комбинацией валентных кварков. Эта кварковая модель предсказывает такие свойства, как изоспин, электрический заряд и магнитный момент. Однако такое описание не учитывает динамику глюонов и орбитального движения партонов, которые важны для формирования наблюдаемых характеристик нуклонов.

Современным инструментом для описания сильного взаимодействия является квантовая хромодинамика, которая охватывает широкий спектр процессов при высоких энергиях. Её основным недостатком является непертурбативные эффекты в области низких энергий, усложняющие прямое вычисление свойств адронов из первых принципов. В частности, проблема распределения углового момента внутри нуклона остаётся одной из ключевых нерешённых задач современной физики.

Для экспериментального изучения спиновой структуры нуклонов реализуется проект Spin Physics Detector (SPD), который будет установлен в одной из двух точек взаимодействия пучков коллайдера NICA (ОИЯИ, г. Дубна).

2.1. Ускорительный комплекс NICA

На территории России основной центр изучения физики высоких энергий находится в Объединённом институте ядерных исследований (ОИЯИ, г. Дубна), на базе которого достраивается новый ускорительный комплекс – Nuclotron-based Ion Collider fAcility (NICA, Рис. 2). Комплекс направлен на изучение свойств сильного взаимодействия.

Комплекс NICA гарантирует широкий спектр пучков: от протонных и дейтронных до пучков тяжёлых ионов (золото). Протоны будут ускоряться

до энергий 12,6 ГэВ, а тяжёлые ядра – до энергий вплоть до 4,5 ГэВ/нуклон. Сердце комплекса – модернизированный нуклотрон, предназначенный для работы в трёх основных режимах [11], два из которых ускоряют частицы для инжекции в коллайдер.

В первой из двух точек взаимодействий коллайдера NICA установлен многоцелевой детектор – MultiPurpose Detector (MPD), который направлен на исследования плотной барионной материи. Во второй точке будет размещён детектор SPD, который предназначен для изучения спиновой структуры протона и дейтрона.

Рисунок 2. Схема ускорительного комплекса NICA

2.2. Эксперимент SPD

Одной из главных проблем современной физики сильных взаимодействий является происхождение спина, равного ½, у протонов и дейтронов. В наивной кварковой модели нуклон состоит из трёх валентных кварков. Экспериментальные данные показали, что вклад их спинов в суммарный спин нуклона составляет только часть от полного значения (менее 30%) [1]. Это указывает на существенный вклад глюонов, «морских кварков» и орбитального углового момента партонов в формирование спина нуклона.

Международный исследовательский эксперимент SPD (Spin Physics Detector) направлен на изучение спиновой структуры протона и дейтрона в поляризованных pp- ($\sqrt{s} < 27$ ГэВ) и dd- ($\sqrt{s} < 13{,}5$ ГэВ) столкновениях. Эксперимент ориентирован на получение количественной информации о распределении углового момента нуклона в рамках квантовой хромодинамики.

2.2.1. Изучаемые процессы

Для описания внутренней структуры нуклона используется функция партонных распределений (PDF), которые определяют вероятность обнаружения партона с заданной долей продольного импульса.

Предусмотрено три канала для определения глюонной составляющей поляризованных глюонов в протонах и дейтронах (Рис. 3):

1. Инклюзивное рождение чармониев;
2. Инклюзивное рождение частиц с открытым очарованием;
3. Инклюзивное рождение быстрых фотонов.

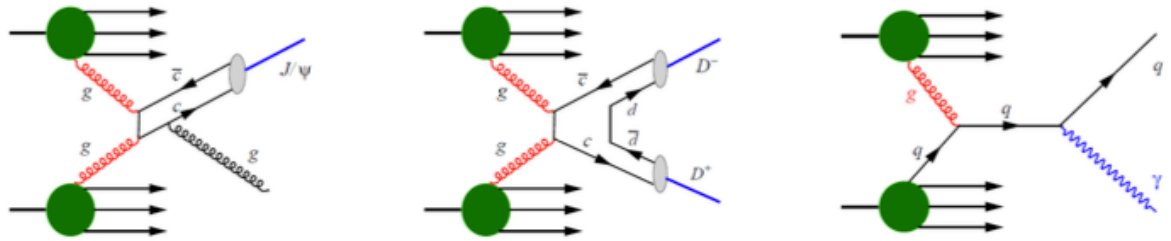


Рисунок 3. Диаграммы, демонстрирующие три канала для исследования распределения поляризованных глюонов в протоне и дейтроне в эксперименте SPD: рождения (a) чармония, (b) частиц с открытым очарованием, (c) быстрых фотонов [12]

Выбор этих процессов объясняется их высокой чувствительностью к глюон-глюонным взаимодействиям и возможностью изучения распределения глюонов в нуклоне. Рождение чармония предоставляет чистый экспериментальный сигнал и большое сечение, процессы рождения частиц с

открытым очарованием помогает напрямую исследовать глюонную плотность при больших значениях переменной Бьёркена, а рождение быстрых фотонов даёт информацию о партонной динамике из-за отсутствия последующего сильного взаимодействия фотонов.

2.2.2. Установка SPD

Установка SPD спроектирована как универсальный детектор с 4π-геометрией (охватывает телесный угол вокруг точки столкновения), который обладает расширенными возможностями регистрации и идентификации частиц на основе современных технологий (Рис. 4).

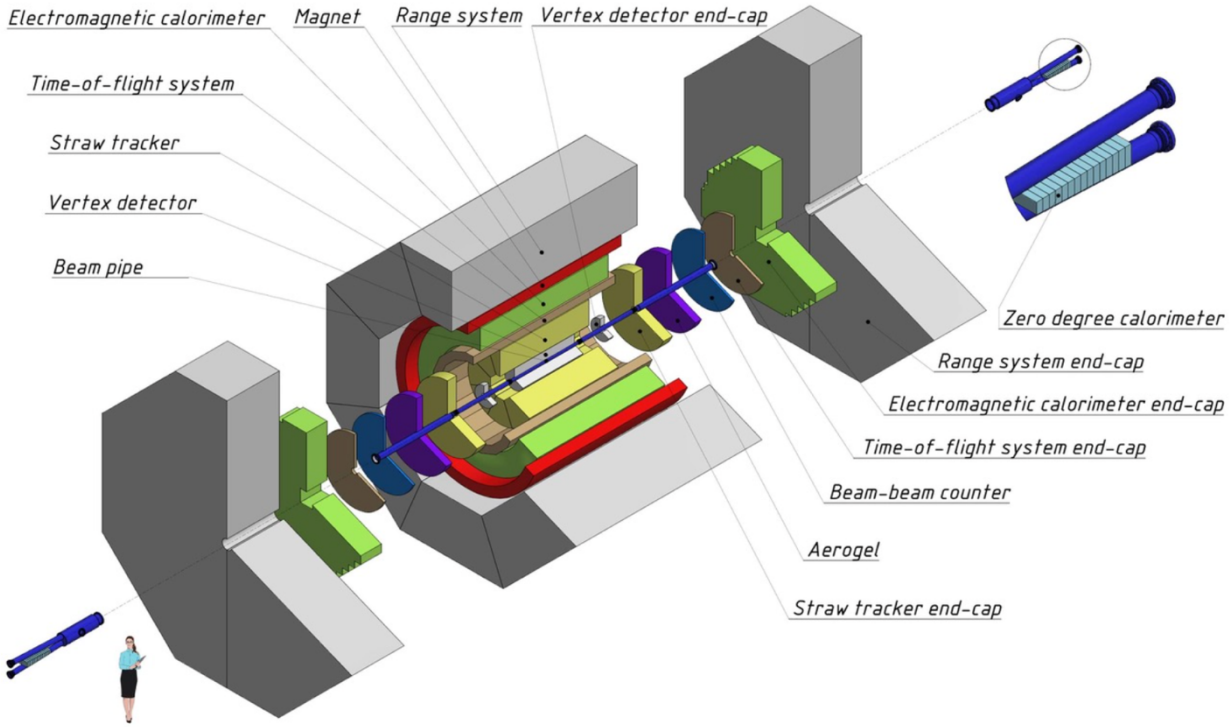


Рисунок 4. Схема экспериментальной установки SPD

Детектор состоит из нескольких подсистем, которые обеспечивают регистрацию и идентификацию частиц. В центральной части проходит вакуумная трубка, вокруг которой размещён вершинный детектор.

Кремниевый вершинный детектор (VD) обеспечивает разрешение положения вершины на уровне ниже 100 мкм, необходимое для реконструкции вторичных вершин распадом D-мезонов.

Следующим слоем является трековая система (TS) на основе дрейфовых трубок, заключённая внутри магнитного поля соленоида, который занимает внешнюю часть центрального объёма. TS максимально точно определяет траектории частиц, по изгибу которых можно вычислить их импульсы. В торцевых частях установки также расположены элементы трековой системы, обеспечивающие регистрацию частиц, которые вылетают под малыми углами к оси пучка.

Вокруг трековой системы установлена времяпролетная система (TOF) с временным разрешением около 60 пс, необходимая для идентификации частиц по времени их прохождения через детектор (например, разделение π/K и K/p). В торцевых частях также установлены элементы этой системы, которые идентифицируют частицы, движущиеся вдоль оси пучка. Далее расположен электромагнитный калориметр (ECal), задача которого – измерять энергию частиц.

За пределы калориметра в основном проникают только мюоны. Они попадают в мюонную систему (RS), которая идентифицирует мюоны и частично заменяет адронный калориметр.

Вблизи оси пучка в торцевых частях установлены дополнительные детекторы:

1. калориметры нулевого угла (ZDC – регистрируют нейтральные частицы, распространяющиеся под малыми углами к направлению пучка, а также контролируют светимость и поляризацию пучка);

2. счётчики столкновений (BBC – регистрируют факт столкновения и измерения характеристик пучка).

2.2.3. Сбор данных

Регистрация «события» в эксперименте осуществляется с помощью детекторов, состоящих из множества чувствительных элементов. Число таких элементов может достигать сотен тысяч, как например, в трековых системах, в которых необходимо большое разрешение. В эксперименте SPD

число таких каналов считывания составит около 600000 [6]. Каждый элемент детектора передаёт сигнал, который преобразуется в цифровой формат и содержит в себе информацию о параметрах сигнала, таких как заряд, амплитуда, длительность. Следовательно, на начальном этапе «событие» представляет собой набор цифровых данных, поступающих от детектора в систему сбора данных (DAQ), где могут проходить первичную фильтрацию, обработку и мультиплексирование.

Так как объём данных, поступающих с детектора, огромен, задача эффективного сбора, передачи и обработки данных становится одной из первостепенных. В классических экспериментах это решается использованием аппаратного триггера, то есть с помощью триггерных детекторов формируется сигнал, по которому далее идёт отбор событий для дальнейшей обработки.

Эксперимент SPD отличается тем, что в нём отказались от идеи аппаратного триггера. В нём данные с детектора считываются непрерывно. Это связано с тем, что в данном эксперименте решение о значимости «события» невозможно принять на аппаратном уровне, так как оно зависит от измерения импульса частиц и положения вершины взаимодействия.

Такая система сбора данных называется бестриггерной. Данные группируются в слайсы с привязанной временной меткой, из которых на этапе последующей обработки происходит восстановление «событий» и их отбор.

Глава 3. Системы хранения и обработки данных эксперимента SPD

3.1. DAQ

Параметры коллайдера NICA и выбранный режим работы детектора SPD напрямую определяют устройство системы сбора данных (Data AcQuisition system, DAQ). При максимальной энергии в системе центра масс $\sqrt{s} = 27$ ГэВ и светимости 10^{32} см$^{-2}$с$^{-1}$. Ожидаемая частота взаимодействий составляет $3 \cdot 10^6$ см$^{-1}$ при периоде следования банчей 76 нс [13]. Близость фоновых и сигнальных процессов, а также невозможность отбора событий на аппаратном уровне без предварительной реконструкции импульса частиц и положения вершины детектора делают применение классической триггерной схемы нереализуемым. Это принципиальное отличие SPD от предшествующих экспериментов (Рис. 5).

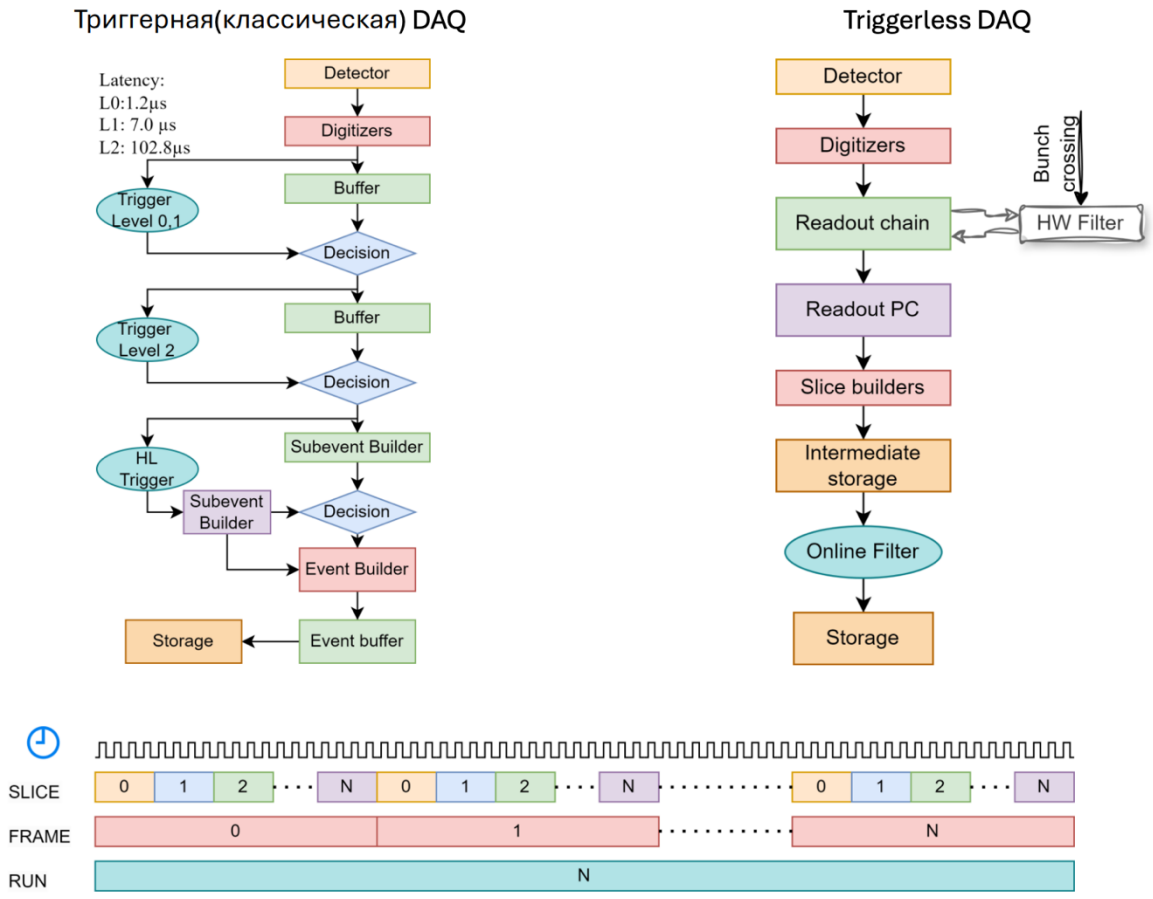


Рисунок 5. Сравнение триггерной (слева) и бестриггерной (справа) систем сбора данных; временная структура потока данных в DAQ SPD [14]

В классической схеме (Рис. 5, слева) сигналы с детектора последовательно проходят несколько уровней отбора: на каждом уровне триггер (Trigger Level 0, Trigger Level 1, Trigger Level 2, High-Level Trigger) на основе предварительно сформированных критериев принимает решение о том, сохранять событие или отбрасывать его. Только прошедшие отбор данные передаются через конструкторы подсобытий (Subevent Builder) и событий (Event Builder) в постоянное хранилище. Характерные значения времени задержки между регистрацией сигнала и принятием триггерного решения на каждом уровне (например, в эксперименте ALICE [15] Run1/Run2 L0: 1,2 мкс; L1: 7,0 мкс, L2: 102,8 мкс) задают жёсткие требования к скорости принятия решения и к буферизации данных в ожидании его.

В бестриггерной схеме (Рис. 5, справа), реализуемой в SPD, цепочка обработки выстроена иначе: данные с оцифровщиков (Digitizers) поступают в цепочку чтения (Readout chain) непрерывно, без предварительного аппаратного отбора, и далее последовательно передаются на считывающие компьютеры (Readout PC), в систему построения слайсов (Slice builders) и на промежуточное хранилище (Intermediate storage). Отбор событий выполняется уже на программном уровне, и только после него данные попадают в постоянное хранилище. При этом единственным аппаратным элементом, который выполняет фильтрацию на ранней стадии, является опциональный HW Filter, связанный с пересечением банчей (Bunch crossing) и работающий совместно с цепочкой чтения.

Масштаб задачи, стоящей перед такой DAQ, удобно оценить количественно. На первой стадии эксперимента число каналов считывания не превышает 180 тысяч, а в полной конфигурации возрастает примерно до 600 тысяч [6]. С увеличением числа каналов и частоты взаимодействий растёт и поток данных: оценка, основанная на моделировании и результатах пучковых тестов детекторов экспериментов MPD и ALICE, при множественности 8 заряженных и 6 нейтральных частиц на событие даёт суммарный поток порядка 20 Гб/с в полномасштабной конфигурации, тогда как на первой

стадии он будет меньше на 1-2 порядка [6]. В пересчёте на годовой объём это соответствует около 10 ПБ необработанных данных и приблизительно 30 миллиардам событий – это величина, сопоставимая с крупнейшими экспериментами на LHC.

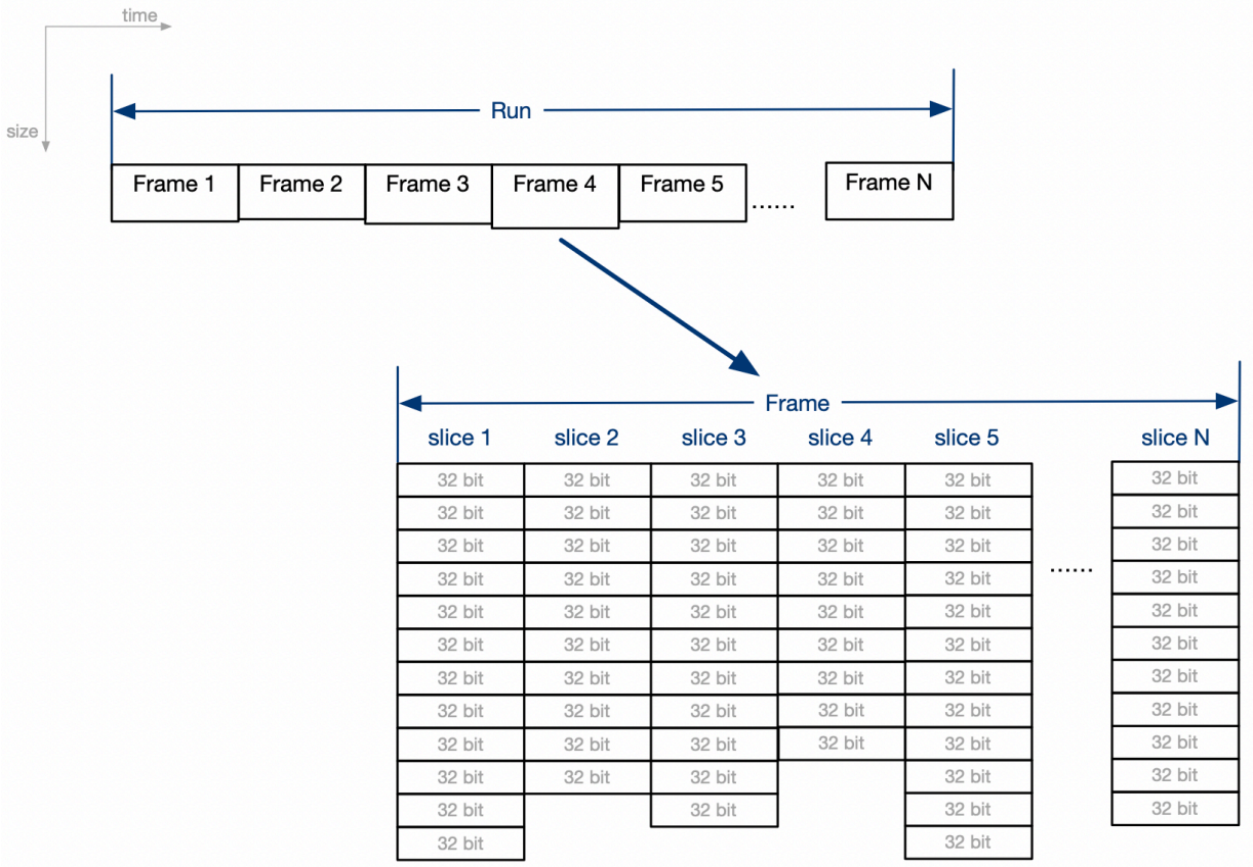


Рисунок 6. Организация данных DAQ SPD [16]

Чтобы упорядочить такой поток и сохранить возможность временной корреляции срабатываний детектора, DAQ организует данные в иерархическую временную структуру (Рис. 6). Все сигналы, превышающие пороги во фронтенд-электронике, считываются вместе с их временными метками и группируются в слайсы (slice) длительностью от 10 до 100 мкс, которые, в свою очередь, объединяются во фреймы (frame) длительностью от 0,1 до 10 с; фреймы, относящиеся к одному непрерывному сеансу набора данных, образуют run (сеанс). Каждый слайс однозначно идентифицируется номером фрейма, определяющим его положение в пределах сеанса, и порядковым номером слайса внутри фрейма (на схеме – слайсы с номерами 0, 1, 2, …, N, повторяющиеся в каждом фрейме). Временные метки каждого

срабатывания (hit) измеряются относительно начала фрейма, поэтому именно фрейм задаёт максимальный интервал, в пределах которого возможна корреляция событий по времени.

Корректность временной структуры обеспечивается подсистемой синхронизации Time Synchronization System (TSS), основанной на технологии White Rabbit. Она распределяет глобальный тактовый сигнал частотой 125 МГц и синхронные команды по всем элементам DAQ, обеспечивая точность синхронизации лучше 1 нс при джиттере менее 50 пс. Логически DAQ включает три основные подсистемы: TSS, цепочки чтения (Readout chains) и систему построения слайсов (Slice Builder System).

Передача данных от детектора осуществляется через двухуровневую систему концентраторов L1 и L2. Концентраторы L1 принимают данные от фронтенд-электроники FEE, выполняют буферизацию и передачу данных по оптическим каналам со скоростью до 10 Гбит/с.

Поскольку данные одного slice распределены между несколькими считывающими компьютерами, их объединение выполняет система построения слайсов. Она собирает данные во временные блоки (chunk), формирует полный слайс и записывает его в промежуточное хранилище. Работа системы обеспечивается процессами чтения (reader), процессом-супервайзером (supervisor) и процессами-сборщиками (builder), которые управляют распределением задач, сборкой данных и записью результатов.

3.2. SPD Online Filter

После системы сбора данных первым этапом обработки является SPD Online Filter. Это высокопроизводительная вычислительная система, которая предназначена для обработки данных. Главная цель Online Filter – это быстрая первичная реконструкция событий и уменьшение объёма данных минимум в 20 раз перед дальнейшим хранением и обработкой. В отличие от DAQ, которая отвечает за непрерывное считывание и построение слайсов,

Online Filter работает уже с файлами данных и управляет их прохождением через последовательность вычислительных этапов.

Входные данные из DAQ поступают в Input buffer в виде файлов согласованного размера. Далее они регистрируются в системе управления данными и передаются в цепочку программной обработки. После выполнения вычислительных задач результаты записываются в Intermediate Storage / Output buffer, откуда подготовленные данные могут быть переданы на этап Offline Processing.

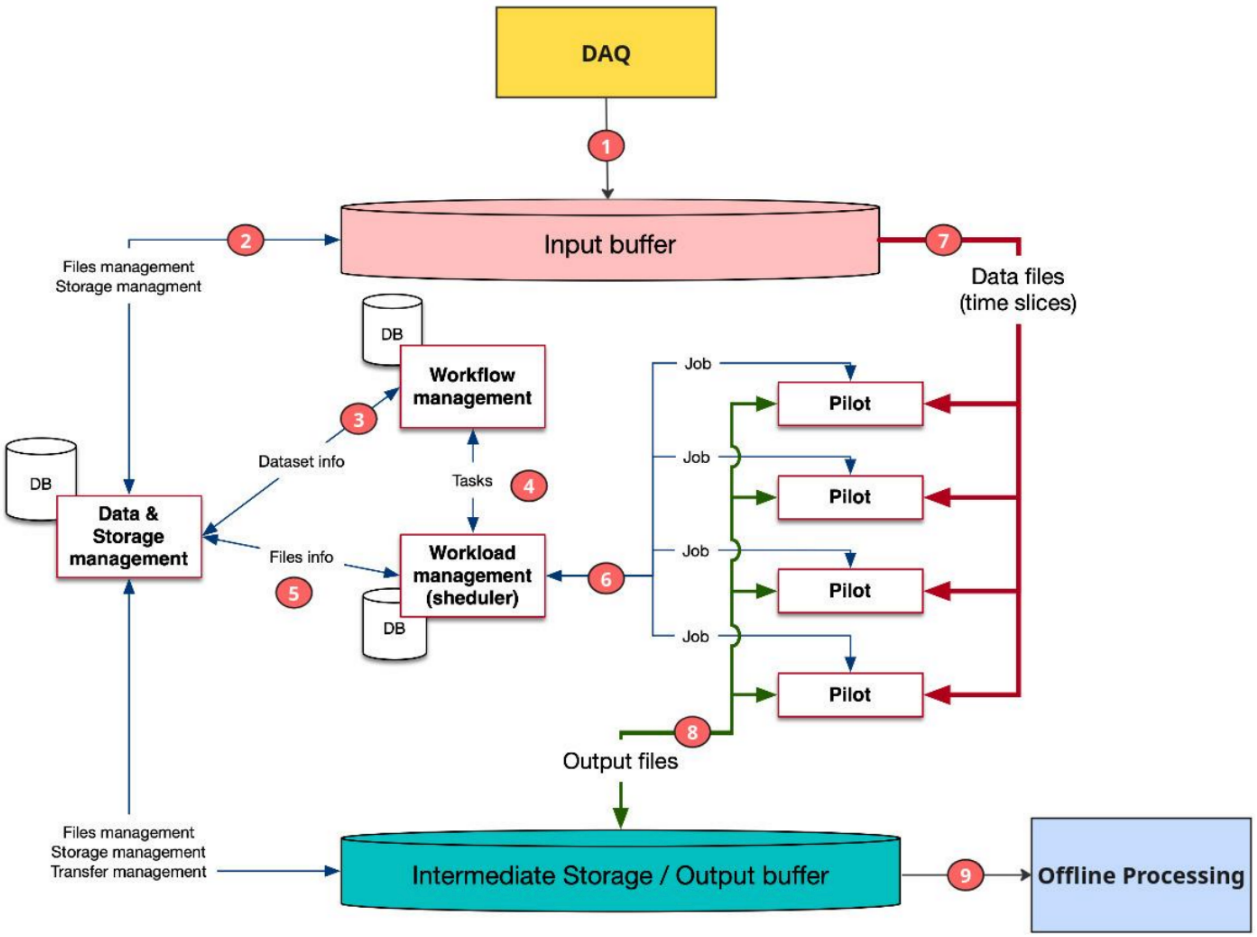


Рисунок 7. Архитектура SPD Online Filter [17]

Промежуточное программное обеспечение Online Filter включает следующие основные компоненты:

1. Система управления данными (Data & Storage Management) отвечает за регистрацию файлов, объединение их в датасеты, хранение служебной информации и контроль согласованности каталога с физическим размещением данных. Входные датасеты формируются после поступления

файлов из DAQ. Промежуточные датасеты возникают на отдельных этапах обработки и используются внутри Online Filter. Выходные датасеты содержат данные, подготовленные для дальнейшей офлайн-обработки.

2. Система управления процессами обработки (Workflow Management) задаёт порядок выполнения этапов. Она связывает входные датасеты с соответствующими шаблонами обработки и формирует цепочку задач. Такая цепочка может включать распаковку данных, реконструкцию, построение и отбор событий, контроль качества и упаковку результата. Выход одного этапа используется как вход для следующего, поэтому весь процесс имеет многоступенчатую структуру.

3. Система управления нагрузкой (Workload Management) получает задачи от Workflow Management и преобразует их в набор отдельных заданий. Обычно одно задание соответствует обработке одного файла или ограниченной группы файлов. Такой подход позволяет распараллелить обработку и эффективно использовать вычислительный кластер. Дальнейшее исполнение заданий выполняют Pilot-агенты, которые запускаются на вычислительных узлах, получают задание, обращаются к нужным входным файлам, запускают прикладную программу и сохраняют результат.

В результате работы Online Filter формирует несколько потоков информации.

3.3. PanDA

Система PanDA (система производства данных и распределённого анализа) была разработана в рамках проекта ATLAS [18] для удовлетворения потребностей в системе управления рабочей нагрузкой для производства и распределённой обработки аналитических данных в масштабах обработки данных на Большом адронном коллайдере (БАК). PanDA позволяет управлять пользовательскими и производственными задачами через единый интерфейс, обрабатывая от 2500 до 300 тысяч заданий на более чем 170 площадках ежедневно.

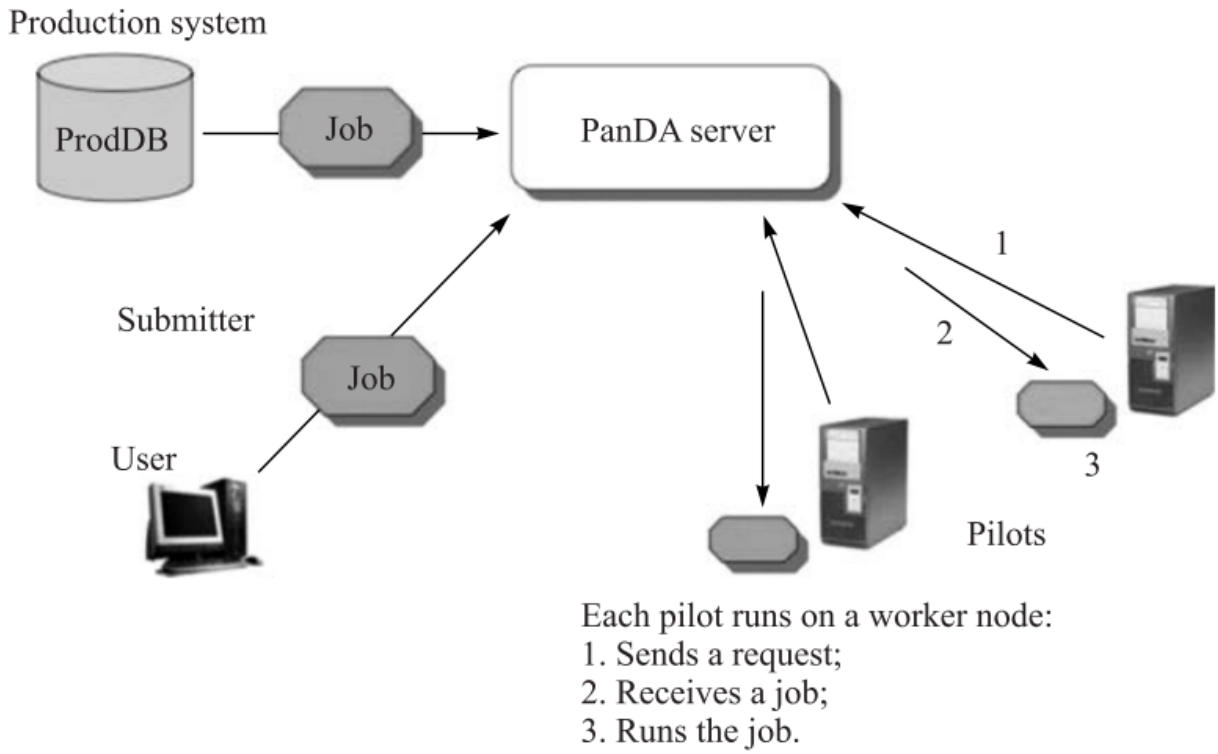


Рисунок 8. Данные PanDA [18]

3.4. Rucio

Rucio – это программный фреймворк с открытым исходным кодом, который обеспечивает научным коллаборациям функциональность организации, управления и доступа к данным в масштабах порядка эксабайт. Данные могут распределяться по центрам обработки данных, разделённым географически. Изначально Rucio был разработан для удовлетворения требованиям эксперимента по физике высоких энергий ATLAS и в настоящее время продолжает поддерживать сообщество LHC-экспериментов и другие разнообразные научные сообщества [19].

Rucio был создан как комплексное решение для организации данных, управления ими и доступа к научным экспериментам, которое включает кооперацию существующих инструментов и делает их простыми для взаимодействия друг с другом. Одним из главных принципов Rucio является автономность и автоматизация, а также проектирование с учётом эволюции данных.

Для эксперимента SPD (NICA) система рассматривается как центральный компонент распределённого управления данными – фактически аналог того, как Rucio используется в ATLAS (LHC), но адаптированный под инфраструктуру ОИЯИ и задачи SPD.

В связке с ним работают:

1. PanDA – система управления задачами. Rucio поставляет для PanDA информацию о данных, доступных репликах, свободном месте в хранилище и наборах данных для обработки;
2. FTS [20] – сервис передачи файлов между storage-сайтами. Rucio инициирует и контролирует передачи через FTS3;
3. CRIC [21] – каталог вычислительных и storage-ресурсов. Из CRIC в Rucio импортируется конфигурация storage endpoints и RSE;
4. IAM [22] – система аутентификации и авторизации. Rucio использует IAM как центральный источник информации о пользователях.

Основная мотивация внедрения Rucio в SPD связана с большими объёмами данных, сравнимыми с масштабами ATLAS. Для управления всеми файлами необходима специальная система.

В настоящее время Rucio используется для:

1. каталогизации данных, представленных в виде файлов и наборов файлов (датасетов);
2. единого взаимодействия с разнородными инфраструктурами хранения;
3. восстановления данных;
4. адаптивной репликации [23].

3.5. SAMPO

SAMPO – это платформа для прикладного программного обеспечения эксперимента SPD, разрабатываемая на базе фреймворка Gaudi [24]. Она предназначена для решения тех задач, которые возникают в эксперименте до и после стадий, реализованных в DAQ и Online Filter: Монте-Карло

генерации первичных вершин, моделирования прохождения частиц через вещество детектора, оцифровки, реконструкции и последующего анализа смоделированных и реальных данных. До этого аналогичные задачи в SPD решались пакетом SPDRoot [25] на базе FairRoot, однако из-за ряда недостатков руководством эксперимента было принято решение о переходе на новую платформу.

Выбор Gaudi в качестве основы обусловлен его надёжностью, подтверждённой использованием в крупных коллаборациях физики высоких энергий (ATLAS, LHCb), а также поддержкой многопоточной обработки через расширение GaudiHive. Архитектура Gaudi построена на наборе типовых компонентов – алгоритмов, сервисов и инструментов (tools), взаимодействующих между собой через интерфейсы. Алгоритмы выполняют преобразования над данными в рамках одного события, сервисы решают общие задачи приложения (доступ к хранилищу, вывод сообщений, генерация случайных чисел), а инструменты служат настраиваемыми единицами поведения, которыми могут пользоваться и алгоритмы, и сервисы. Конфигурация всего приложения задаётся механизмом JobOptions на языке Python: именно через свойства компонентов пользователь определяет, какие алгоритмы и в каком порядке будут выполняться, какие сервисы будут активны и какие инструменты они будут использовать.

В рамках SAMPO в инфраструктуру Gaudi интегрированы основные библиотеки стека ПО физики высоких энергий: для генерации первичных вершин, HepMC3[26] как универсальный формат представления событий, GeoModel для описания геометрии установки, а также Geant4 [27] для моделирования взаимодействия частиц с веществом. Отдельно реализован сервис расчёта магнитного поля по карте поля, хранящегося в виде базы данных SQLite-база данных, Все эти библиотеки изолированы внутри соответствующих сервисов и классов-адаптеров, что позволяет при необходимости заменять или добавлять их без изменения вышележащего кода.

Такая организация даёт несколько практических следствий. Любой этап обработки данных SAMPO собирается из одних и тех же строительных блоков и конфигурируется единообразно через JobOptions, что и обеспечивает «многоэтапность», вынесенную в название платформы. Использование GaudiHive открывает возможность распараллелить обработку между событиями и между независимыми алгоритмами внутри одного события, что важно при ожидаемых для SPD объёмах данных.

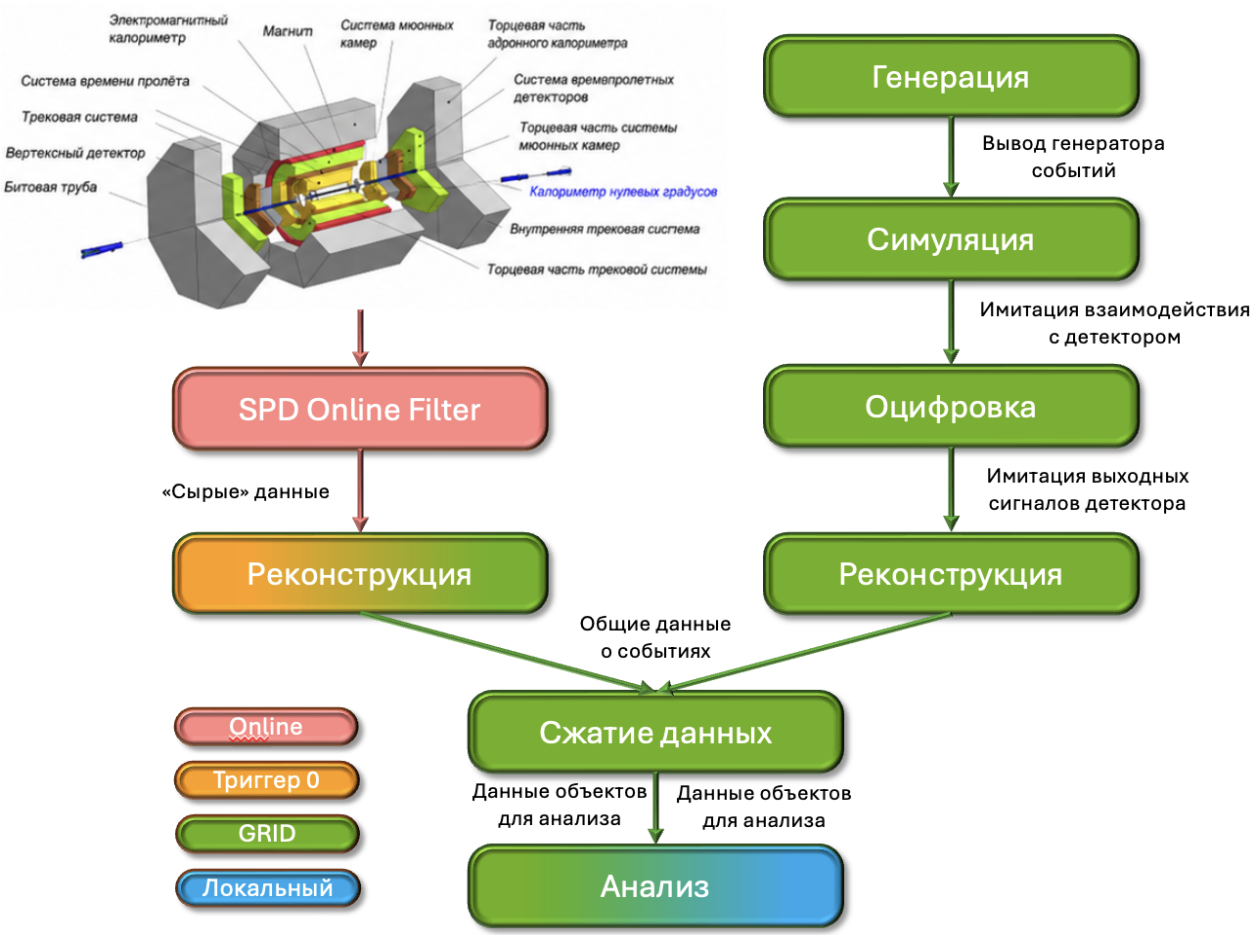


Рисунок 9. Этапы обработки данных

В контексте общей цепочки обработки задачи, реализованные на платформе SAMPO, выполняются сразу за DAQ и Online Filter: они получают на вход либо смоделированные первичные вершины, либо данные, прошедшие первичный отбор в онлайн-фильтре, и проводят их через моделирование отклика детектора, оцифровку и реконструкцию. Платформа находится в стадии активного развития: в неё встраиваются новые компоненты, обеспечивающие работу с данными установки SPD, а также

интеграцию с внешними информационными системами. Каталогизация полученных файлов и датасетов выполняется уже описанной системой Rucio, управление задачами обработки – PanDA, а индексирование отдельных событий и их атрибутов – системой Event Index.

3.6. Event Index

Event Index – каталог физических событий эксперимента SPD. Он обеспечивает доступ к данным конкретного события или группы событий в нужном формате и версии, а также поиск событий по их параметрам.

На стадии подготовки эксперимента система работает с моделированными событиями, на которых отрабатываются структуры данных, процедуры индексирования и пользовательские интерфейсы. В дальнейшем те же подходы будут применены к реальным данным с установки.

Объёмы данных и требования к производительности системы задаются параметрами эксперимента. Ожидается поток до нескольких десятков тысяч событий в секунду на первом этапе и до 150 тысяч – на втором, что даёт около 1 трлн событий в год при размере записи 160 байт. Полный объём каталога оценивается в сотни ТБ. Event Index должен обеспечить производительность, необходимую для своевременной индексации поступающих данных, а также выполнять запросы пользователей за разумное время.

К основным задачам Event Index относятся:

1. доступ к событиям по идентификатору (Event Picking); отбор событий по метаданным:
2. создание виртуальных датасетов и проверка целостности данных – поиск дубликатов;
3. сверка количества событий между информационными системами и сравнение датасетов на разных стадиях обработки.

Глава 4. Обзор существующих решений

Прямой аналог разрабатываемой системы – это ATLAS Event Index, каталог событий эксперимента ATLAS на ускорителе LHC в ЦЕРН. Его опыт – это основной ориентир при проектировании Event Index для эксперимента SPD, поэтому ниже рассмотрены требования, архитектура и эволюций этой системы по описанию в работе [28].

4.1. Контекст и предшествующая система

Эксперимент ATLAS работает с 2009 года. За первые два периода набора данных (Run 1 в 2009-2013 гг. Run 2 в 2015-2018 гг.) было собрано почти 25 миллиардов физических событий; помимо реальных данных, генерируется ещё примерно в три раза больше моделированных данных. Файлы данных хранятся в распределённой системе и каталогизируются по датасетам через систему Rucio, которая управляет более чем 100 миллионами файлов на 120 сайтах, расположенных по всему миру при суммарном объёме хранения свыше 200 ПБ. Однако Rucio оперирует файлами и датасетами и не хранит информацию об отдельных событиях, тогда как для ряда задач (построение event display, проверка корректности реконструкции, отбор отдельных событий из больших датасетов) нужен доступ именно к уровню событий.

До создания Event Index эту задачу решал каталог первого поколения TAG DB на Oracle. Он импортировал событийную информацию из специальных TAG-файлов, но столкнулся с двумя проблемами:

1. TAG-данные содержали как неизменяемую информацию (идентификация событий, триггерные решения), так и изменяемую при перереконструкции (физические величины), причём при перереконструкции TAG-файлы заново не создавались, и изменяемая часть быстро устаревала;

2. Структура базы повторяла структуру TAG-файла, поэтому изменяемое содержимое нельзя было отделить от неизменяемого, а

индексирование всех колонок ради быстрого поиска требовало непропорционально большого объёма хранилища.

При проектировании Event Index эти проблемы были устранены: система получила собственные задачи сбора данных, не зависящие от центральной цепочки производства ATLAS, и собирает событийные метаданные из файлов любой стадии обработки (не только AOD). В новой системе сознательно отказались от хранения изменяемого содержимого, сосредоточившись на неизменяемых параметрах события, а в качестве технологической базы выбрали инструменты BigData.

4.2. Варианты использования

Основной сценарий Event Index – event picking, то есть извлечение полной информации об одном или нескольких событиях по их идентификаторам, например, для построения event display или проверки корректности реконструкции. Помимо этого, система решает задачи контроля качества данных: проверки полноты обработки путём сверки числа событий во входных и выходных датасетах, выявления дубликатов и контроля целостности файлов на диске. Третий блок задач связан с триггерной информацией (подсчётом срабатываний триггерных цепочек, построение матриц перекрытий между цепочками и между производными датасетами), что используется для оптимизации триггерных меню и сокращения числа деривационных потоков.

4.3. Содержимое записи и требования к производительности

Каждая запись Event Index содержит три блока:

1. идентификатор события (run number, event number, trigger stream, формат, версия обработки, тип данных, временная метка6 условия LHC, luminosity block number, для моделированных данных = вес события и идентификатор процесса);

2. триггерная информация (маски L1, L2 и HLT, ключ триггера SMK для декодирования, ключ prescale);

3. информация о расположении (GUID файла, содержащего событие, и указатели на upstream-файлы в цепочке обработки - provenance).

Каталог должен обеспечить темп индексации не ниже скорости поступления данных: для Run 2 – около 1 кГц для реальных данных и примерно столько же для моделированных, для Run 3 – не менее 10 кГц с запасом на возможные задержки. Простые задачи (event lookup) должны отрабатывать менее чем за секунду, тяжёлые пакетные запросы – порядка часа.

4.4. Архитектура

Поток данных ATLAS Event Index линеен, поэтому система разбита на четыре компонента, соответствующих стадиям обработки:

1. Data Production (извлечение метаданных из файлов на местах их создания);
2. Data Collection (передача данных в центральное хранилище ЦЕРН с проверкой полноты и декодированием триггеров);
3. Data Storage (хранилище и интерфейсы запросов);
4. Monitoring (мониторинг всех компонентов)

Партиционированная архитектура (Рис. 10) и чётко определённые интерфейсы между компонентами позволяют независимо развивать и заменять каждый из них.

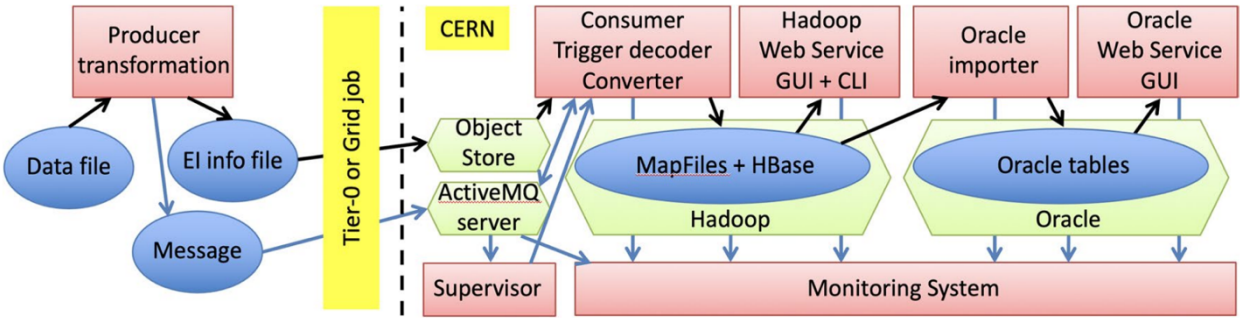


Рисунок 10. Глобальна архитектура системы ATLAS Even Index, реализованная по завершении второго этапа работы на БАК [28]

Извлечение метаданных выполняет Producer – программа на Python, реализованная как трансформация в стандартном фреймворке Athena, что позволяет запускать её на Tier-0 и на ресурсах WLCG(Worldwide LHC Computing Grid). Программа читает только заголовки событий, поэтому достаточно стабильна между релизами. Producer работает в два этапа: чтение событий и сохранение нужной информации во временный файл, затем передача файла в центральное хранилище в ЦЕРН. Дополнительно в выходной файл записывается служебная информация – идентификаторы задач и задания PanDA, имя входного датасета, число файлов и событий, временные метки, GUID и число событий каждого читаемого файла. На этом же этапе проверяется уникальность событий внутри файла, и при обнаружении дубликатов ATLAS получает уведомление.

Запуск индексирующих задач организован через систему PanDA: датасеты, помеченные в системе метаданных AMI как полные и пригодные для анализа, отправляются на обработку через сценарий-трансформацию. По типичной статистике обработка одного события занимает от 10 до 50 мс в зависимости от того, нужна ли триггерная информация. Одно задание индексирует несколько файлов за 30-60 минут. Суммарное потребление CPU задачами Producer составляет менее 10^{-4} от общего потребления вычислительных ресурсов ATLAS.

4.5. Сбор данных

В первой реализации Data Collection данные передавались через брокеры сообщений ActiveMQ по протоколу STOMP. С ростом нагрузки от такой схемы отказались: данные перенесены в Object Store, а через ActiveMQ передаются только короткие управляющие сообщения с URI объекта. Это сократило поток через брокеры с мегабайтов до десятков байт на задание. Формат сериализации тоже эволюционировал: от SQLite3 с парами ключ-значение перешли к Stream of Protocol Buffer Message с gzip-сжатием, что даёт степень сжатия более 86%.

Центральную роль в Data Collection играет Supervisor. Он принимает управляющие сообщения от Producer, сверяет число обработанных событий с информацией в Rucio, отслеживает выполнение задач через системы мониторинга console (для Tier-0) и PanDA (для Grid), формирует списки готовых к импорту данных и оповещает Consumers.

Consumers – многопоточные Java-программы, которые читают объекты из Object Store, преобразуют данные в нужный формат и записывают их в Hadoop. Один Consumers обрабатывает в среднем 15 кГц событий, в максимуме до 28 кГц.

4.6. Хранилище данных

Основное хранилище реализовано в Hadoop в формате MapFile – это пара SequenceFile (данные и индекс) с парами ключ-значение, отсортированными по ключу. Первичный ключ для Event Index – пара Run Number – Event Number. Все MapFile регистрируются в Catalog, реализованном как таблица HBase.

Триггерные маски в исходных файлах хранятся в виде битовых полей, где каждый бит соответствует триггерной цепочке. Для декодирования используются таблицы соответствий: COMA для реальных данных и TriggerDBMC для моделированных. Если SMK отсутствует в записи события, его получают из run number (для реальных данных) или из reconstruction tag (для MC). Таблицы реплицируются из исходных баз в HBase, а декодирование происходит уже там; декодированные имена триггерных цепочек сохраняются обратно в запись события. После импорта датасета автоматически вычисляются производные таблицы: матрицы перекрытий датасетов, матрицы перекрытий триггерных цепочек и статистика срабатываний.

К концу 2015 года (первый год Run 2) при использованных тогда версиях Hadoop/HBase и аппаратной конфигурации CERN стало понятно, что простые запросы event lookup для 10 событий выполняются более минуты, а

подсчет событий по большим датасетам – десятки минут. Чтобы поддержать критические сценарии для реальных данных, часть информации (без триггерных решений, составляющих около 80% объёма) была продублирована в Oracle. Структура реляционной схемы строится вокруг двух основных таблиц – Datasets (один ряд на датасет, имя датасета разложено на шесть колонок: project, RunID, stream, формат, AMI tag, шаг производства) и Events (события датасета, до трёх GUID на событие, плюс LBN и BCID). Дополнительные таблицы Event Duplicates, Dataset Overlaps и LBN Counts хранят дубликаты, перекрытия и агрегированную статистику. Применение «basic» компрессии Oracle, сжатия ключевого индекса и хранения GUID в формате RAW (16 байт вместо 36) сокращает объём примерно в 10 раз – до 20 байт на событие. С учётом индексов хранение 25 миллиардов событий укладывается менее чем в 1 ТБ.

Доступ к каталогу организован через клиентские команды (catalog, ei, el, ti, inspect) и веб-интерфейс на Tomcat, реализующий REST API. Графический интерфейс EIO Browser позволяет интерактивно фильтровать датасеты и формировать отчёты: Event Lookup, Dataset Report, Dataset Overlaps Report (с двумерной матрицей перекрытий), Duplicate Event Report, Missing Event Report, Count by BCID, Count by LB and GUID.

4.7. Мониторинг

Состояние всех компонентов отслеживается отдельной подсистемой. Первая её версия на Kibana столкнулась с проблемами производительности и была заменена связкой InfluxDB и Grafana. Сбор этой информации выполняют Python-скрипты, запускаемые по расписанию. Они запрашивают журналы HDFS, REST-эндпоинты и веб-страницы и отправляют данные через HTTP-эндпоинт в формате JSON. Подсистема различает четыре статуса компонентов: доступен (available), частично доступен (degraded), недоступен (unavailable) и нет данных (N/A).

4.8. Эволюция системы

Ожидаемый рост частоты событий в Run 3 (100 миллиардов реальных и 300 миллиардов моделированных событий по проекту HL-LHC) потребовал пересмотра базовой технологии хранения. Был выбран переход на HBase с надстройкой Apache Phoenix, обеспечивающий SQL-доступ к таблицам HBase.

HBase поддерживает разбиение колонок на семейства, что позволяет хранить информацию о расположении, происхождении и триггерах в разных семействах и читать только нужное. Схема таблиц для Phoenix спроектирована близко к Oracle-схеме.

Параллельно с заменой основного хранилища обновляются и другие компоненты: Producer переходит на современные библиотеки (stomp.py, Protocol Buffers), Data Collection – на Spark, Supervisor расширяется на весь цикл от выбора датасетов до загрузки в HBase. Обнаружение дубликатов и подсчёт статистики переносятся в процесс импорта.

Дополнительно с 2021 года развивается отдельный сервис – Event Picking Service. Это веб-сервис, принимающий список событий для извлечения и автоматически выполняющий весь цикл: запрос к Event Index для получения GUID файлов, отправку заданий PanDA, повторные попытки при сбоях, сохранение результатов в центральном хранилище и информирование пользователя о статусе. Сервис ориентирован на массовые запросы – выборки по 10 тысяч событий и более, которые нужны физическим анализам, запускающим на предварительно отобранных событиях собственные процедуры.

4.9. Применимость опыта ATLAS к SPD Event Index

Опыт ATLAS Event Index значим для проектирования SPD Event Index по ряду причин:

1. Он подтверждает жизнеспособность подхода, при котором каталог событий хранит не сами данные, а служебную информацию, достаточную для локализации событий в распределённой системе хранения.

2. Основные варианты использования совпадают с задачами SPD Event Index.

3. Ожидаемые в SPD на втором этапе объёмы (порядка триллиона событий в год) лежат в том же диапазоне, что и плановые показатели ATLAS Run 3 и Run 4, что позволяет ориентироваться на отработанные архитектурные решения.

4. Разделение системы на компоненты Data Production, Data Collection, Data Storage и Monitoring с управлением через Supervisor и привязкой к PanDA напрямую заложено в архитектуру SPD Event Index.

В SPD применяется близкая к ATLAS организация распределённой обработкой и хранения данных: задачи обработки управляются системой PanDA, а файлы и датасеты каталогизируются и реплицируются средствами Rucio, что позволяет переносить отработанные в ATLAS схемы взаимодействия Event Index с этими системами.

При этом архитектура SPD Event Index отличается от ATLAS Event Index в некоторый моментах. В качестве основного хранилища выбрана реляционная СУБД PostgreSQL, а не связка Hadoop с Oracle или HBase с Phoenix. Это связано с меньшим начальным масштабом эксперимента и отсутствием выделенной инфраструктуры Hadoop, требующей больших усилий для поддержания. Декодирование триггерной информации в SPD не требуется, поскольку эксперимент использует бестриггерной систему сбора данных. Соответственно, структура записи события будет более определённой. Передача данных индексирования организована не через объектное хранилище и брокер сообщений, тем не менее эти технологии возможно будут использованы для обмена информацией между Event Index и информационными системами управления распределённой обработкой и хранением данных.

Глава 5. Практическая часть: SPD Event Index

5.1. Требования к Event Index

Требования к Event Index определяются двумя группами факторов: параметрами потока данных эксперимента SPD и характером задач, которые решаются с использованием каталога.

На первом этапе эксперимента поток данных от установки оценивается в несколько тысяч событий в секунду, на втором этапе достигает 150 тысяч событий в секунду. Скорость индексирования должна устойчиво превышать скорость поступления данных, иначе будет накапливаться очередь необработанных датасетов. Расчёт исходя из 1 триллиона событий в год при размере записи 150 байт даёт полный объём каталога порядка сотен ТБ, поэтому требование производительности относится и к разовому пиковому потоку, и к долговременной работе с большим объёмом хранимых данных [29]. Запросы к Event Index выполняются в ходе анализа, и их время не должно заметно тормозить работу пользователей: по опыту аналогичной системы эксперимента ATLAS, для запросов, возвращающих большую выборку событий, ориентиром служит лимит в 60 минут, тогда как работа с отдельными событиями должна выполняться значительно быстрее, и здесь верхняя граница задаётся временем обработки одного события в SAMPO.

Базовое требование к содержимому каталога – уникальный идентификация каждого смоделированного и зарегистрированного установкой события. Идентификатор, сохраняющийся неизменным на всех последующих стадиях обработки, состоит из двух частей: 32-битного числа, к которому привязываются метаданные общие для датасета, и 64-битного номера событий в датасете. Для MC-данных он присваивается на стадии генерации: 32-битный ProductionID и номер события в Production. Данные, сгенерированные вне централизованной системы, должны проходить дополнительную обработку с присвоением идентификаторов. Для данных с детектора идентификатор формируется на SPD Online Filter для событий в

формате RAW: 32 бита – номер Run, 64 бита – номер события в Run. Использование составного идентификатора позволяет оптимизировать объём данных в EI (Event Index), разделив метаданные уровней датасета и событий, а также повысить производительность системы.

Помимо идентификатора, для каждого события в каталоге хранятся метаданные, ссылка на исходный RAW-файл и ссылки на файлы последующих стадий обработки. При повторной обработке возможно создание нескольких версий файлов, и все они должны привязываться к существующей записи. Помимо данных событий, в системе ведутся списки проиндексированных датасетов с информацией о статусе индексации. Сгенерированные события на данный момент записываются в формате HepMC2, в SAMPO предполагается использование HepMC3, на следующих стадиях обработки появятся другие форматы. Полный набор форматов и структура событий пока не определены, поэтому Event Index должен опираться на унифицированную модель с общими для всех форматов элементами – идентификаторами и метаданными.

Отдельная группу требований связана с организацией процесса индексирования. Файлы и датасеты создаются в количествах, исключающих ручное управление индексацией, поэтому система должна поддерживать оба механизма обнаружения новых данных: оповещение от системы управления вычислениями по завершении создания датасета и периодический опрос каталога событий; опыт ATLAS показывает, что полная переиндексация в ходе эксперимента может потребовать неоднократно. Помимо этого, часть событий может оказаться нечитаемой, повторятся и отсутствовать; для моделированных данных доля «плохих» событий на начальнov этапе может достигать десятков процентов. Event Index должен поддерживать поиск дубликатов по идентификатору и по метаданным, сверку количества событий между датасетами на разных стадиях обработки и сопоставление с информацией в других системах эксперимента.

Наконец, доступ к каталогу должен быть организован через программный интерфейс для использования в задачах обработки, а также через клиентские приложения с возможностью поиска по параметрам событий.

5.2. Архитектура промежуточного программного обеспечения

Архитектура SPD Event Index построена вокруг центрального хранилища на базе СУБД PostgreSQL [34] и связывает его с компонентами распределённой обработки эксперимента: Rucio, PanDA и FTS. Важный вопрос при проектировании такой системы – это способ передачи данных от индексирующей задачи на удалённом узле в центральный каталог. От выбора этого способа зависят нагрузка на сеть, общая производительность индексирования и устойчивость системы при работе с удалёнными датасетами.

Рассматривалось три варианта обмена данными: синхронный обмен, пакетный обмен для каждого файла и пакетный обмен для всего датасета.

В синхронном обмене данных к Event Index выполняется на каждое событие: индексирующая программа считывает событие, собирает метаданные и отправляет их к Event Index вместе с идентификатором содержащего его файла, после чего ожидает подтверждения записи и переходит к следующему событию. Такая схема проста в реализации, но накладывает ограничения на производительность, особенно при индексировании данных на удалённых серверах. Частично проблему ожидания снимает использование брокера сообщений, позволяющего не ждать ответ EI перед обработкой следующего события.

Пакетный пофайловый вариант группирует обмен по файлам. Индексирующая программа последовательно считывает события файла, накапливает идентификаторы и метаданные в буфере (transient store), а по завершению обработки файла передаёт собранный пакет в Event Index вместе с идентификатором файла. Это снижает число сетевых обращений по

сравнению с синхронной схемой, но не разделяет полностью этапы индексирования и передачи данных.

В пакетном варианте по датасету индексирующая задача формирует собственный EI-датасет. В него для каждого исходного файла записываются его идентификаторы событий и метаданные. По завершению индексирования задача передаёт в Event Index ссылку на готовый EI-датасет, который скачивается посредством Rucio/FTS, проверяется и импортируется в центральное хранилище. Такой подход позволяет разнести во времени индексирование, передачу данных и их запись в каталог, оптимально распределяя нагрузку между компонентами системы (Рис. 11). Логически её можно разделить на три части: индексирование на удалённых узлах, центральный сервер Event Index с компонентами управления данными и их хранения, а также backend-сервер с клиентским сервисом.

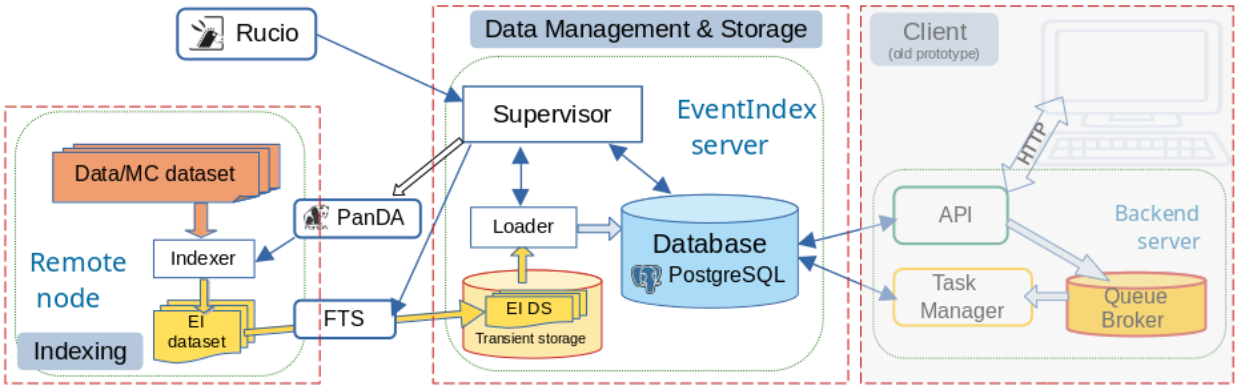


Рисунок 11. Схема Event Index [30]

Индексирование выполняется на узлах, где физически расположены датасеты с данными эксперимента. Запуск индексирующих задач организован через PanDA, а сама индексирующая программа реализуется во фреймворке SAMPO, что позволяет использовать стандартные процедуры работы с данными. Программа считывает события из исходного датасета (Data/MC datasets), извлекает идентификаторы событий, идентификаторы файлов и метаданные, а затем формирует на их основе EI-датасет. Сформированный датасет загружается в центральную базу данных Event Index одним из двух способов: либо через Rucio с предварительной

передачей файлов на сервере EI и последующей загрузкой в БД, либо напрямую с узла хранения в ОИЯИ, минуя промежуточный сервер.

Центральную роль в координации процесса играет компонент Supervisor. Он получает от Rucio информацию о новых датасетах, требующих индексирования, ставит соответствующие задачи в PanDA, принимает результаты (EI-датасеты) через FTS или Rucio во временное хранилище (transient storage) и запускает компонент Loader.

Loader забирает данные из временного хранилища, проверяет полученную информацию и загружает её в базу данных PostgreSQL. В самой базе хранятся записи о событиях (идентификатор, метаданные, ссылки на исходный RAW-файл, ссылки на файлы последующих стадий обработки), а также списки проиндексированных датасетов с информацией о статусе индексации. Записи событий пополняются по мере появления новых производных файлов: индексация AOD и других файлов последующих стадий обработки идёт по той же цепочке и добавляет ссылки в уже существующие записи.

Доступ пользователей к каталогу обеспечивает отдельный backend-сервер. Запросы от клиентского приложения поступают на HTTP в API, который обращается к БД PostgreSQL и возвращает результаты. Долгие запросы (например, выборка большого числа событий) обрабатываются асинхронно: API передаёт через брокер сообщений (Queue Broker) компоненту диспетчеру задач Task Manager, который и выполняет работу с базой. Управление задачами должно быть организовано, таким образом, чтобы обеспечивать эффективное обслуживание многочисленных «коротких» запросов на одно или несколько событий и объёмных задач. Клиентское приложение предоставляет графический интерфейс для поиска по параметрам событий, создания запросов и получения ответов.

5.3. Размеры записи и объёмов данных

Объём данных, который предстоит хранить в Event Index, определяется размером одной записи события и числом событий, поступающих за год работы эксперимента. В составе каждого идентификатора событий 4 байта приходятся на Run Number и 8 байт на Event Number, ссылка на файл хранится в виде 16-байтного UUID. Для каждого производного AOD-файла дополнительно записывается 16 байт UUID и 3 байта на номер версии.

В простейшем варианте, когда запись содержит только идентификатор события и ссылку на исходный RAW-файл, её размер составляет 28 байт. При добавлении ссылки на один AOD-файл и сопутствующих полей размер вырастет до 33 байт. Пересчёт на ожидаемый поток данных даёт следующие оценки годового объёма каталога: на первом этапе эксперимента – около 4,2 ТБ для конфигурации EventID+RAW и около 5 ТБ для конфигурации EventID+Raw+1 AOD, на втором этапе – 42 ТБ и 50 ТБ соответственно (Табл. 2).

Состав записи	Байт	За год Stage I	За год Stage II
EvenID+ DID RAW	28	4,2 ТБайт	42 ТБайт
EvenID+ DID RAW + 1 параметр +1 AOD	50	7,5 ТБайт	50 ТБайт
EvenID+ DID RAW + 3 параметр +3AOD	94	14 ТБайт	141 Тбайт
EvenID+ DID RAW + 3 параметр +3 AOD (3 параметра)	130	20 ТБайт	295 ТБайт

Таблица 2. Оценка объёмов хранения Event ID по составу записи и этапу эксперимента

Помимо записей событий, в Event Index хранятся списки проиндексированных датасетов с информацией о статусе индексации, поэтому полный объём каталога превышает приведённые значения. Эти

оценки задают нижнюю границу требований к хранилищу и подтверждают необходимость использования СУБД с поддержкой больших таблиц.

5.4. Тестирование загрузки в базу данных

Для оценки производительности центрального хранилища системы EI было проведено тестирование загрузки в базу данных больших объёмов моделированных данных с минимальным набором метаданных. После оптимизации процедуры загрузки удалось добиться производительности, достаточной для приёма максимальных ожидаемых потоков данных эксперимента (150 событий/с). Тестирование при этом проводилось на виртуальной машине, размещённой на совместно используемом оборудовании, что позволяет ожидать существенно лучших показателей на выделенном высокопроизводительном оборудовании с оптимизированным программным обеспечением.

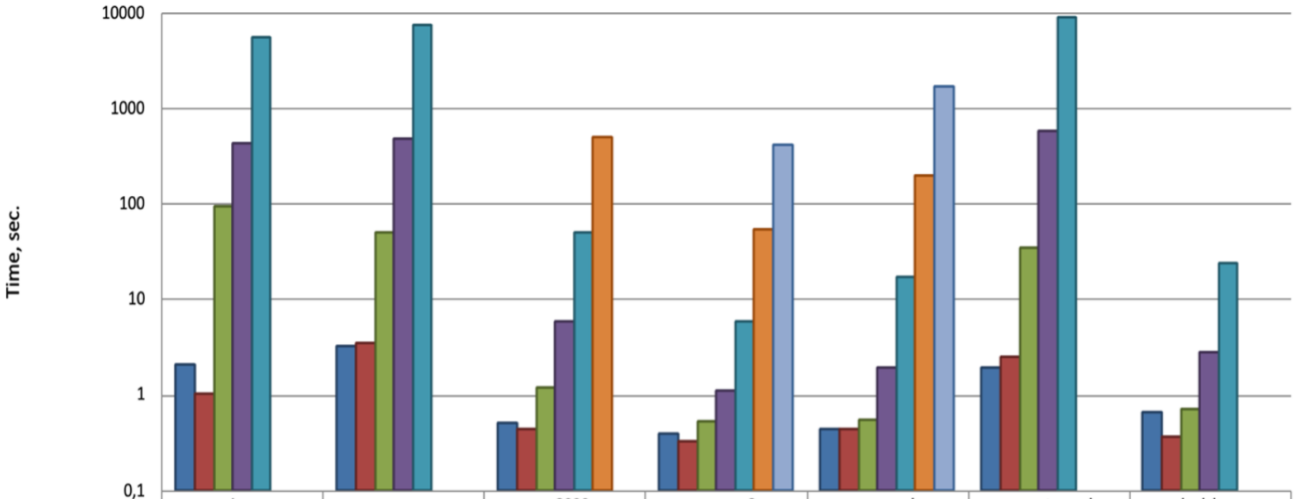
Тестирование проводилось в несколько этапов [34]. На каждом этапе изменялась либо версия python-скрипта, либо способ взаимодействия с PostgreSQL, после чего замерялось время чтения и записи в таблицы БД для разных объёмов данных. Для тестирования использовались семь модулей: asyncpg, aiopg, pg8000, psycopg2, PyGreSQL, py-postgresql и SQLAlchemy.

5.4.1. Оптимизация скрипта на JSON-файлах

На первом этапе сгенерированные события записывались в базу по одному с помощью запросов INSERT (Рис. 12). На не больших объёмах все модули справлялись за секунды, однако с ростом числа событий время увеличивалось непропорционально. На 100 тысяч событий asyncpg, aiopg и py-postgresql превышали два часа, что для задачи неприемлемо – ожидаемый поток с детектора 1 трлн событий в год.

На втором этапе запрос INSERT был заменён на COPY с массовой загрузкой данных. Модули aiopg и py-postgresql такой режим не поддерживают, поэтому в дальнейших тестах они не участвовали. Переход на

COPY ускорил процесс примерно вдвое. На малых объёмах (до 100 тысяч событий), лучший результат показал PyGreSQL – 100 тысяч событий за 2,7с. На больших объёмах вперёд вышел модуль asyncpg – 10 млн событий за 6 минут.

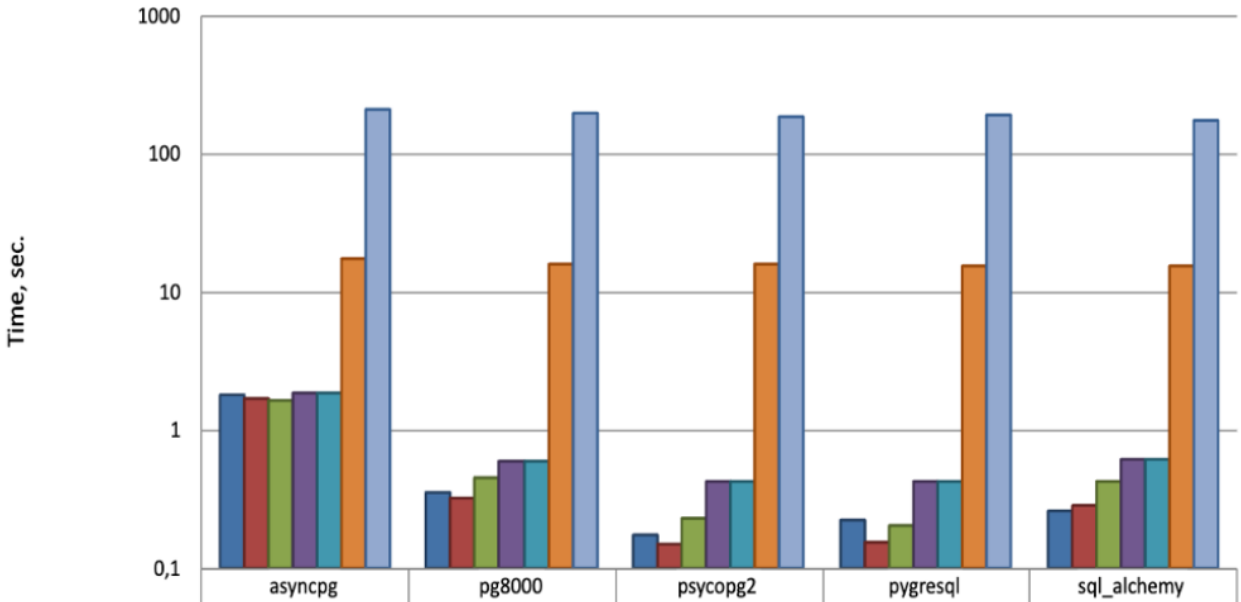


	aiopg	asyncpg	pg8000	psycopg2	pygresql	py-postgresql	sql_alchemy
10 events	2,05	3,21	0,50	0,40	0,44	1,93	0,66
100 events	1,02	3,45	0,45	0,33	0,44	2,47	0,36
1 000 events	93,47	49,84	1,17	0,52	0,55	34,84	0,70
10 000 events	432,35	470,17	5,86	1,11	1,89	581,96	2,77
100 000 events	5484,15	7296,31	50,53	5,87	17,18	9037,89	24,24
1 000 000 events			500,55	54,21	194,08		
10 000 000 events				415,95	1698,68		

Рисунок 12. Результаты тестирования модулей при использовании запроса INSERT (по горизонтали: тестируемые модели; по вертикали: логарифмическая шкала времени, в секундах)

На третьем этапе в процесс записи были введены временные файлы: данные сначала генерировались в JSON, затем перезаписывались во временный файл и уже оттуда передавались в таблицу. На этой версии скрипта лидером стал модуль psycopg2 – 10 млн событий за 4,5 минуты.

На заключительном этапе JSON-файлы были полностью убраны, и временные файлы использовались и при генерации, и при записи в таблицы (Рис. 13). Вперёд вышел SQLAlchemy – 10 млн событий за 3 минуты,



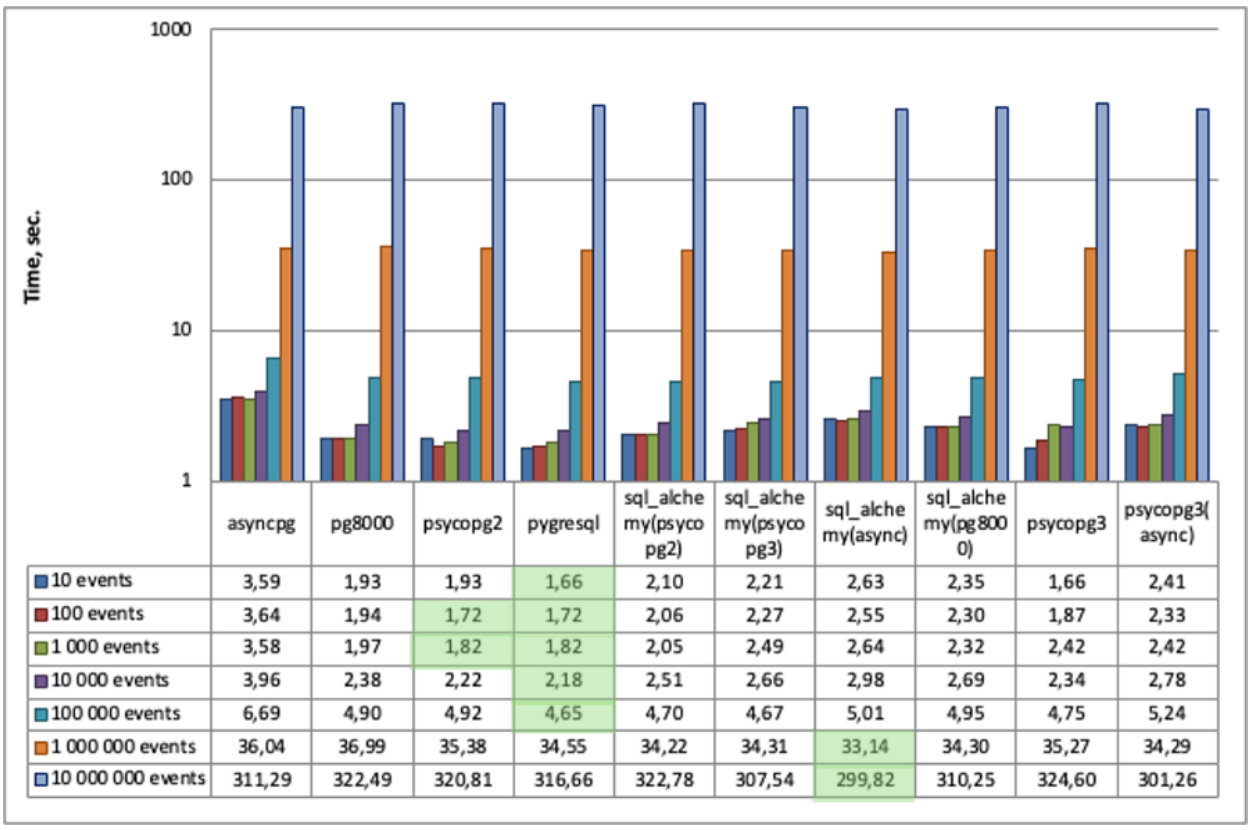
	asyncpg	pg8000	psycopg2	pygresql	sql_alchemy
10 events	1,79	0,35	0,18	0,22	0,26
100 events	1,69	0,32	0,15	0,15	0,28
1 000 events	1,65	0,44	0,23	0,21	0,42
10 000 events	1,86	0,59	0,43	0,42	0,62
100 000 events	1,86	0,59	0,43	0,42	0,62
1 000 000 events	17,13	15,83	15,59	15,38	15,21
10 000 000 events	206,40	195,27	186,57	186,86	174,13

Рисунок 13. Результаты тестирования модулей при использовании временных файлов для генерации и записи данных в таблицы (по горизонтали: тестируемые модели; по вертикали: логарифмическая шкала времени, в секундах)

5.4.2. Переход на ROOT-файлы и тестирование новых модулей

В эксперименте SPD для хранения регистрируемых событий планируется использовать ROOT-файлы. Соответственной тестовый код был доработан так, чтобы данные генерировались сразу в этом формате. Все ранее отобранные модули были перенесены и протестированы с чтением событий из ROOT-файлов и записью в БД (Рис. 14). Лучше всех показал себя модуль psycopg2 – 10 млн событий за 6,5 минут.

Дополнительно был протестирован psycopg3 – новая версия драйвера. Она использует возможности современных версий Python и PostgreSQL и предоставляет синхронный и асинхронный интерфейсы.



	asyncpg	pg8000	psycopg2	pygresql	sql_alchemy(psycopg2)	sql_alchemy(psycopg3)	sql_alchemy(async)	sql_alchemy(pg8000)	psycopg3	psycopg3(async)
10 events	3,59	1,93	1,93	1,66	2,10	2,21	2,63	2,35	1,66	2,41
100 events	3,64	1,94	1,72	1,72	2,06	2,27	2,55	2,30	1,87	2,33
1 000 events	3,58	1,97	1,82	1,82	2,05	2,49	2,64	2,32	2,42	2,42
10 000 events	3,96	2,38	2,22	2,18	2,51	2,66	2,98	2,69	2,34	2,78
100 000 events	6,69	4,90	4,92	4,65	4,70	4,67	5,01	4,95	4,75	5,24
1 000 000 events	36,04	36,99	35,38	34,55	34,22	34,31	33,14	34,30	35,27	34,29
10 000 000 events	311,29	322,49	320,81	316,66	322,78	307,54	299,82	310,25	324,60	301,26

Рисунок 14. Результаты тестирования модулей с чтением данных из ROOT-файлов (по горизонтали: тестируемые модели; по вертикали: логарифмическая шкала времени, в секундах)

5.4.3. Профилирование

Для понимания того, на что именно уходит время, к python-скрипту был применён встроенный в Python модуль cProfile. Профилирование позволяет получить по каждой функции количество вызовов, собственное время выполнения, суммарное время с учётом вложенных вызовов.

Профили строились для всех пяти модулей, прошедших предыдущие этапы оптимизации. Во всех случаях основная доля ресурсов приходилась на функции, которые выполняют чтение и запись данных в БД, а вспомогательные операции дают пренебрежимо малый вклад. Это означает, что дальнейшая оптимизация будет проводится уже на высокопроизводительном оборудовании в рамках полного цикла.

5.5 Проверка данных и контроль качества

Опыт эксперимента ATLAS показывает, что данные событий могут оказаться «плохими», то есть нечитаемыми, дублироваться или отсутствовать. В первую очередь это относится к моделированным данным, где доля «плохих» событий на начальном этапе может достигать десятков процентов, но подобные проблемы возникают и с реальными данными в процессе обработки. Индексирование на каждом этапе обработки и проверки их целостности и согласованности выявляет проблемы качества данных и помогает принимать меры по их устранению.

При первичном индексировании события считываются целиком, после чего записываются заново с добавлением идентификатора и метаданных. Такая процедура одновременно решает две задачи: формирует запись в каталоге и выявляет событие, которое не удалось прочитать. Проверка уникальности набора метаданных позволяет обнаружить повторяющиеся события, а сравнение числа событий с разных этапов обработки и сопоставление с информацией в системе – выявить утерянные.

Этот набор проверок входит в основные задачи Event Index и охватывает поиск внутри файла или датасета, поиск между датасетами, сверку числа событий и поиск дубликатов.

Чтобы перечисленные проверки были возможны, к каждому событию необходимо иметь устойчивый доступ по уникальному идентификатору и единообразному набору метаданных, не зависящего от формата исходного файла. Решение этой задачи определяет требование к структуре записи событий – к Event Header.

5.6. Event Header

Для того чтобы Event Index мог работать с событиями единообразно вне зависимости от формата исходного файла, в запись каждого события предлагается добавить отдельный блок, содержащий идентификатор событий

и набор метаданных. Состав метаданных определяется задачами обработки и анализа.

Для моделированных событий в заголовке может храниться seed. Для данных с детектора – классификатор событий, формируемый в SPD Online Filter. Помимо этого, в заголовок могут добавляться параметры события, которые могут быть использованы для предварительного отбора. Метаданные могут дополняться в процессе обработки, но для конкретного типа данных (RAW, AOD) набор метаданных одинаков, что обеспечивает единообразие записей в каталоге.

Процедуру записи и чтения заголовка предполагается реализовать в виде алгоритма SAMPO с использованием средств фреймворка. Такой подход позволяет переносить содержимое заголовка из исходного датасета в выходной с автоматическим изменением формата при переходе между стадиями обработки.

Отдельная задача связана с хранением метаданных, общих для всех событий в файле. Стандартные средства Gaudi обеспечивают только пособытийную запись, поэтому требуется отдельное изучение способов. Позволяющих записывать общие метаданные на стадии инициализации или завершения. Ожидается, что наличие такой структуры не повлияет на работу стандартных средств пособытийного чтения и записи.

5.6.1. Прототип Event Header

В рамках моей работы реализован прототип процедура записи Event Header6 который может быть встроен в цепочку обработки SAMPO и использоваться как основа для формирования записей в Event Index. Прототип реализован в виде Gaudi-алгоритма HepMCHeaderWriterAlg на C++ с конфигурацией на Python (Приложение А, листинг А;1-А.3). Алгоритм работает с событиями в формате HepMC3 и дописывает каждому из них атрибуты уровня события, которые сохраняются в выходном файле HepMC ASCII v3 строками вида A 0 <name> <value>.

На входе алгоритм получает указатель на HepMC-событие и значение seed, читаемые из транзитного хранилища данных SAMPO по адресам TDSAddresses::MC::HepMC и TDSAddresses::MC::Seed (Приложение А, листинг А.1). На стадии инициализации проверяется заданный путь выходного файла и создаётся объект HepMC::WriterAscii; при пустом пути или неудачной инициализации записывающего объекта алгоритм возвращает ошибку. На стадии завершения файл закрывается (Приложение А, листинг А.2).

В заголовок записываются пять атрибутов (Приложение А, листинг А.2, метода addHeaderAttributes):

- sampo_run_number – глобальный номер запуска, задаваемый через свойства алгоритма runNumber;
- sampo_event_number – глобальный номер события. Если в HepMC-событии номер не задан или не положителен, используется номер события из контекста Gaudi (Gaudi::Hive::currentContext().evt()) с добавлением настраиваемого смещения;
- sampo_seed – seed генератора, считанный из транзитного хранилища;
- sampo_fuid – составной идентификатор события, собираемый из номера запуска и номера события через разделитель, задаваемым свойством fuidSeparator (по умолчанию);
- sampo_metadata – строка с базовыми параметрами события6 содержащая количество частиц и количество вершин в формате n_particles=<N>; n_vertices=<M>. Эти значения тестово выбраны как всегда доступные в GenEvent. Конфигурация алгоритма выполняется через модуль HepMCHeaderConfig.py (Приложение А, листинг А.3).

5.6.2. Структура выходного файла

Логически событие в выходном HepMC-файле состоит из двух частей (Рис. 15, слева): блока Event Header с атрибутами и физического

содержимого события (частицы, вершины, кинематические параметры). В выходном файле формата HepMC ASCII v3 заголовок появляется в виде строк атрибутов (Рис. 15, справа) после служебных строк события E, U, W, перед строками частиц P (см. Приложение А6 Листинг А. 2, метод addHeader Attributes).

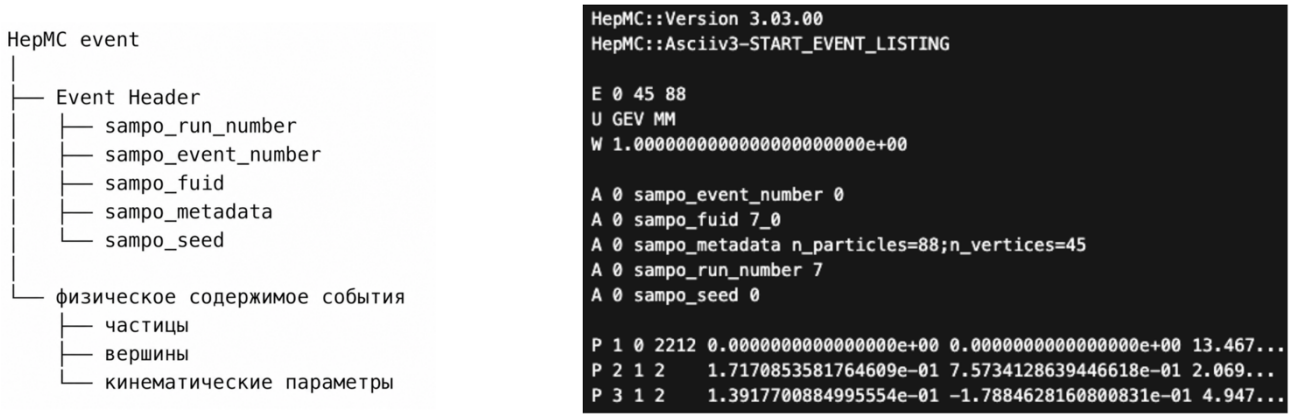


Рисунок 15. Слева: предварительная структура события в HepMC-файле; справа: выходной файл с Event Header

Полученный файл подтверждает, что прототип Event Header встраивается в поток обработки SAMPO и служебная информация сохраняется вместе с событием. В дальнейшем эта информация может использоваться для заполнения записей Event Index: при индексировании из заголовка события считывается служебная информация, к которой затем добавляются ссылки на фай, датасет и производные данные, полученные на последующих стадиях обработки. Набор полей заголовка является предварительным и будет уточняться в ходе разработки.

ВЫВОДЫ

Event Index представляет собой каталог физических событий эксперимента SPD, обеспечивающий доступ к данным конкретного события или группы событий по идентификатору и метаданным, а также поиск событий по их параметрам. Система занимает положение в общей цепочке обработки данных SPD после DAQ, Online Filter и SAMPO и взаимодействует с системами Rucio, PanDA и FTS, что определило выбор архитектуры и способа передачи данных индексирования.

Опыт системы ATLAS Event Index показал жизнеспособность подхода с записью служебной информации вместо самих данных и стал основным ориентиром при работе над SPD Event Index. Вместе с тем для разрабатываемой системы выбраны другие технологии: реляционная СУБД PostgreSQL, передача данных через EI-датасеты средствами Rucio и FTS и отказ от декодирования триггерной информации в связи с бестриггерной схемой сбора данных. Из трёх каналов обмена данными между индексатором и центральной БД для Event Index SPD был выбран пакетный по датасету. Он разносит во времени индексирование, передачу данных и их запись в каталог и согласуется с общей моделью распределённого управления данными.

Оценка размера записи показала, что при конфигурации EventID и ссылки на RAW-файл запись занимает 28 байт; при добавлении туда ссылки на один AOD-файл – 33 байта. Эти оценки задают нижнюю границу требований к хранилищу и подтверждают необходимость использования промышленной СУБД с поддержкой больших таблиц.

Для единообразной работы с событиями вне значимости от формата исходного файла предложенная структура Event Header, включающая идентификатор событий и набор служебной информации.

ЗАКЛЮЧЕНИЕ

В ходе работы был проведён обзор эксперимента SPD на ускорительном комплексе NICA, его физических задач и систем хранения и обработки данных: DAQ, Online Filter, PanDA, Rucio, SAMPO. Рассмотрена система Event Index, её место в общей цепочке обработки и требования, которые накладывают на неё параметры эксперимента и характер решаемых задач.

Был проанализирован опыт системы ATLAS Event Index как прямого аналога разрабатываемой системы. Рассмотрены её архитектура, варианты использования, эволюция технологий хранения и сбора данных. На основе этого анализа определены те решения, которые применимы к SPD Event Index, и те аспекты, в которых архитектуры EI отличаются.

Была рассмотрена архитектура промежуточного программного обеспечения Event Index. Описаны компоненты этой системы: Supervisor, Loader, backend-сервер с API и клиентским приложением. Проведена оценка размера событий и готового объёма каталога для двух этапов эксперимента.

Также был разработан прототип Event Header – блока, содержащего идентификатор и набор служебной информации. Прототип реализован в виде Gaudi-алгоритма на C++ с конфигурацией на Python и работает с событиями в формате HepMC3, добавляя к каждому событию по пять атрибутов, которые сохраняются в выходном файле и могут использоваться при индексировании. В дальнейшем работа будет связана с:

1. развитием самого прототипа Event Index и всей цепочки индексирования;
2. оптимизацией схемы БД для уменьшения размера и повышения производительности;
3. оптимизацией загрузки данных в рамках полного цикла и на высокопроизводительном уровне;
4. модернизацией и оптимизацией клиента и клиент-серверного взаимодействия.

Студент _____ Будтуева З. А.

Научный руководитель _____ Трубников Г. В.

Руководитель образовательной программы _____ Трубников Г. В.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Ashman J. [и др.]. A Measurement of the Spin Asymmetry and Determination of the Structure Function g_1 in Deep Inelastic Muon-Proton Scattering // Phys. Lett. B. – 1988. – Т. 206. – С. 364.

2. Abazov V. [и др.]. (SPD Collaboration). Technical Design Report of the Spin Physics Detector at NICA // Natural Sci. Rev. – 2024. – Т. 1. – С. 1.

3. V. D. Kekelidze, R. Lednicky, V. A. Matveev, I. N. Meshkov, A. S. Sorin, and G. V. Trubnikov. Three stages of the NICA accelerator complex // Eur. Phys. J. A. – 2016. – Т. 52. – Article 211.

4. Ишханов Б. С.,Кэбин Э. И.Антиматерия – М.:Университетская книга МГУ, 2012. – С. 352. – ISBN 978-5-91304-273-6. – Учебное пособие.

5. Окунь Л. Б.Лептоны и кварки – 2-е изд – М.:Наука,1990 – С.346 – ISBN 5-02-014027-9. – 2-е изд., перераб. и доп.

6. Ellis R., Stirling W., Webber B. QCD and Collider Physics. // Camb. Monogr. Part. Phys. Nucl. Phys. Cosmol. – 1996. – Т. 8. – С. 1-435.

7. Collins J., Soper D., Sterman G. Factorization of Hard Processes in QCD // Adv.Ser.Direct.High Energy Phys. – 1989. – Т. 5. – С. 1-91.

8. Липатов А. В. Жесткие процессы КХД за рамками коллинеарного приближения: Докторская диссертация / Липатов Артем Владимирович. – Москва: Московский государственный университет имени М.В. Ломоносова, Научно-исследовательский институт ядерной физики имени Д.В.Скобельцына, 2022. – Специальность 1.3.15 – «Физика атомных ядер и элементарных частиц, физика высоких энергий» (01.04.23 – «Физика высоких энергий»).

9. Грибов В. Н., Липатов Л. Н. Глубоконеупругое ep-рассеяние в теории возмущений // Ядерная физика. – 1972. – Т. 15. – С. 781–807.

10. Experimental Techniques in Modern High-Energy Physics: A Beginner‘s Guide / K. Hanagaki [и др.]. – 01.2022. – ISBN 978-4-431-56929-9. – DOI:10.1007/978-4-431-56931-2.

11. Syresin E. [и др.] NUCLOTRON Development for NICA Acceleration Complex [Электронный ресурс]. // Part of Proceedings, 10th International Particle Accelerator Conference – 2019. – URL: https://inspirehep.net/literature/1622215 (дата обр. 20.04.2026).

12. Arbuzov A. [и др.] On the physics potential to study the gluon content of proton and deuteron at NICA SPD // Prog. Part. Nucl. Phys. – 2021. – Vol. 119. – P. 103858. – DOI:10.48550/arXiv.2011.15005

13. Afanasyev L. [и др.] The design of the data acquisition system for SPD experiment // arXiv. – 2025. – DOI:10.48550/arXiv.2509.12790

14. Бойков А. Система сбора данных эксперимента SPD на коллайдере NICA [Электронный ресурс]. – 2025. – URL: https://indico.inr.ac.ru/event/5/contributions/NICA.pdf. (дата обр. 20.04.2026).

15. SoldI D, Chiozzi S., Level Zero Trigger Processor for the NA62 experiment // Journal of Instrumentation – 2018. – Vol. 22. – DOI:10.1088/1748-0221/13/05/P05004 – URL: https://iopscience.iop.org/article/10.1088/1748-0221/13/05/P05004 (дата обр. 20.04.2026).

16. Greben, N., Oleynik, D. & Romanychev, L. Workload Management System for SPD Online Filter [Электронный ресурс]. // Physics of Particles and Nuclei Letters – 2025. – Vol. 22. – P. 1044–1047. – DOI:10.1134/S1547477125701067 – URL: https://link.springer.com/article/10.1134/S1547477125701067 (дата обр. 20.04.2026).

17. Oleynik D. SPD On-Line Filter: Workflow and Data Management Systems // Phys. Part. Nuclei – 2024 – Vol. 55 – P. 603-605 – DOI:10.1134/S1063779624030869 [Электронный ресурс]. – URL: https://link.springer.com/article/10.1134/S1063779624030869 (дата обр. 21.04.2026).

18. Petrosyan V. PanDA for COMPASS at JINR [Электронный ресурс]. // Physics of Particles and Nuclei Letters – 2016 – Vol. 13 – P. 708-710. – DOI:10.1134/S1547477116050393 – URL:

https://link.springer.com/article/10.1134/S1547477116050393 (дата обр. 20.04.2026).

19. Barisits M., Beermann, T., Berghaus, F. [и др.] Rucio: Scientific data management [Электронный ресурс]. // Computing and Software for Big Science. – 2019. – Vol. 3 – Article 11. – DOI:10.1007/s41781-019-0026-3 – URL: https://link.springer.com/article/10.1007/s41781-019-0026-3 (дата обр. 20.04.2026).

20. Murray S. [и др.] FTS Service Evolution and LHC Run-3 Operations [Электронный ресурс]. // In Proceedings of the 26th International Conference on Computing in High Energy and Nuclear Physics – 2024. – Vol. 291. – P. 01031. – DOI:10.1051/epjconf/202429501031 – URL: https://www.epj-conferences.org/articles/epjconf/abs/2024/05/epjconf_chep2024_01031/epjconf_chep2024_01031.html (дата обр. 21.05.2026).

21. Anisenkov A. [и др.] CRIC: Computing Resource Information Catalogue as a unified topology system for a large scale, heterogeneous and dynamic computing infrastructure [Электронный ресурс]. // In Proceedings of the 24th International Conference on Computing in High Energy and Nuclear Physics. – 2020. – Vol. 245. – P. 03032. – DOI:10.1051/epjconf/202024503032 – URL: https://doi.org/10.1051/epjconf/202024503032 (дата обр. 21.04.2026).

22. Автоматизация регистрации учётных записей в системе управления данными Rucio / А. С. Конак, А. Ш. Петросян, Д. А. Олейник, А. А. Сычугов // Известия Тульского государственного университета. Технические науки. – 2025. – Вып. 10 – С. 122-133. – DOI:10.24412/2071-6168-2025-10-122-123.

23. Konak A., Petrosyan A. The First Experience of Using Rucio to Manage SPD Data [Электронный ресурс]. // Phys.Part.Nucl.Lett. – 2025. – Vol. 22. – № 5. – DOI:10.1134/S1547477125701055 – URL: https://inspirehep.net/literature/3063999 (дата обр. 20.04.2026).

24. Симбирянтин Л.Л. Программная платформа для многоэтапной обработки данных эксперимент SPD: дис. на соискание степени магистра [Электронный ресурс]. – 2025. – URL: https://spd.jinr.ru/wp-

content/uploads/2025/07/Diplom_Simbiryatin_2025.06.pdf (дата обр. 21.04.2026).

25. SPDRoot software. [Электронный ресурс]. – URL: https://git.jinr.ru/nica/spdroot (дата обр. 21.04.2026).

26. Buckley A. [и др.] The HepMC3 event record library for Monte Carlo event generators [Электронный ресурс]. // Comput. Phys. Commun. – 2021. – Vol. 260. – Article 107310. – DOI:10.1016/j.cpc.2020.107310 – URL: https://www.sciencedirect.com/science/article/abs/pii/S0010465520301181?via%3Dihub (дата обр. 21.04.2026).

27. Agostinelli S. [и др.] Geant4–A simulation toolkit // Nucl. Instrum. Methods Phys. Res., – Sect. A. 506. – P. 250–303 – DOI: 10.1016/S0168-9002(03)01368-8

28. Barberis D., Alexandrov, I., Alexandrov, E. [и др.] The ATLAS Event Index: A BigData Catalogue for All ATLAS Experiment Events [Электронный ресурс]. // Computing and Software for Big Science. – 2023. – Vol. 7. – Article 2. – DOI:10.1007/s41781-023-00096-8 – URL: https://link.springer.com/article/10.1007/s41781-023-00096-8 (дата обр. 21.04.2026).

29. Петросян А. Система управления процессами обработки эксперимента SPD [Электронный ресурс]. // Письма в ЭЧАЯ– 2023. – Т. 56. – Выпуск № 6. – URL: https://pepan.jinr.ru/index.php/Pepan/article/view/2186 (дата обр. 21.05.2026).

30. Prokoshin, F., Tvauri, I., Budtueva, Z. [и др.] SPD Event Index [Электронный ресурс]. // Physics of Particles and Nuclei. – 2024. – Т. 55. – С. 617-620. – DOI:10.1134/S1063779624030699 – URL: https://link.springer.com/article/10.1134/S1063779624030699 (дата обр. 21.05.2026).

ПРИЛОЖЕНИЕ А

```
#include <memory>
#include <string>
#include <GaudiKernel/DataObjectHandle.h>
#include "GaudiKernel/Algorithm.h"
#include "HepMC3/GenEvent.h"
#include "HepMC3/Writer.h"
#include "TDSAddresses.h"

class HepMCHeaderWriterAlg : public Algorithm {
public:
    using Algorithm::Algorithm;

    StatusCode initialize() override;
    StatusCode execute() override;
    StatusCode finalize() override;

private:
    long resolveEventNumber(const HepMC3::GenEvent& event) const;
    std::string buildFUID(long runNumber, long eventNumber) const;
    std::string buildMetadata(const HepMC3::GenEvent& event) const;
    void addHeaderAttributes(HepMC3::GenEvent& event, long seed) const;
    std::shared_ptr<HepMC3::Writer> m_writer;

   DataObjectReadHandle<std::shared_ptr<HepMC3::GenEvent>> m_hepmcReadHandle{
        this,
        "hepmc",
        TDSAddresses::MC::HepMC
    };
    DataObjectReadHandle<long> m_seedReadHandle{this, "seed",
TDSAddresses::MC::Seed};

    Gaudi::Property<std::string> m_filePath{
        this,
        "filePath",
        "HepMC_with_header.dat",
        "Output HepMC file path with event header attributes"
    };
    Gaudi::Property<long> m_runNumber{this, "runNumber", 1, "Global run
number"};
    Gaudi::Property<bool> m_useContextEventNumberIfMissing{
        this,
        "useContextEventNumberIfMissing",
        true,
        "Use Gaudi event number when HepMC event_number is not positive"
    };
    Gaudi::Property<long> m_eventNumberOffset{
        this,
        "eventNumberOffset",
        1,
        "Offset added to Gaudi zero-based event number"
    };
```

```
50.     Gaudi::Property<std::string> m_fuidSeparator{this, "fuidSeparator", "_",
   "Separator in fuid"};
51. };
```

Листинг А.1: Заголовочный файл HepMCHeaderWriterAlg.h

```
#include <sstream>
#include "GaudiKernel/ThreadLocalContext.h"
#include "HepMC3/Attribute.h"
#include "HepMC3/WriterAscii.h"
#include "HepMCHeaderWriterAlg.h"

namespace {

bool ends_with(const std::string& str, const std::string& suffix) {
    return str.size() >= suffix.size() && str.compare(str.size() - suffix.size(),
suffix.size(), suffix) == 0;
}

template <typename T>
std::shared_ptr<HepMC3::StringAttribute> makeStringAttribute(const T& value) {
    std::ostringstream stream;
    stream << value;
    return std::make_shared<HepMC3::StringAttribute>(stream.str());
}

}
StatusCode HepMCHeaderWriterAlg::initialize() {
    if (m_filePath.value().empty()) {
        error() << "Empty output HepMC file path" << endmsg;
        return StatusCode::FAILURE;
    }

    if (!ends_with(m_filePath.value(), ".dat")) {
        warning() << "HepMCHeaderWriterAlg writes HepMC ASCII v3; expected output file
extension is .dat" << endmsg;
    }

    m_writer = std::make_shared<HepMC3::WriterAscii>(m_filePath.value());
    if (m_writer->failed()) {
        error() << "HepMC WriterAscii creation error for " << m_filePath.value() <<
endmsg;
        return StatusCode::FAILURE;
    }

    info() << "HepMCHeaderWriterAlg output file: " << m_filePath.value() << endmsg;
    info() << "HepMCHeaderWriterAlg run number: " << m_runNumber.value() << endmsg;

    return StatusCode::SUCCESS;
}
```

```
StatusCode HepMCHeaderWriterAlg::execute() {
    auto hepmc = *m_hepmcReadHandle.get();
    auto seed = *m_seedReadHandle.get();

    if (!hepmc) {
        error() << "Input HepMC event is null" << endmsg;
        return StatusCode::FAILURE;
    }

    addHeaderAttributes(*hepmc, seed);

    m_writer->write_event(*hepmc);
    if (m_writer->failed()) {
        error() << "Can't write HepMC event with header attributes" << endmsg;
        return StatusCode::FAILURE;
    }

    return StatusCode::SUCCESS;
}

StatusCode HepMCHeaderWriterAlg::finalize() {
    if (m_writer) {
        m_writer->close();
    }

    return StatusCode::SUCCESS;
}

long HepMCHeaderWriterAlg::resolveEventNumber(const HepMC3::GenEvent& event) const {
    const long hepmcEventNumber = event.event_number();
    if (hepmcEventNumber > 0 || !m_useContextEventNumberIfMissing.value()) {
        return hepmcEventNumber;
    }

    return static_cast<long>(Gaudi::Hive::currentContext().evt()) +
m_eventNumberOffset.value();
}

std::string HepMCHeaderWriterAlg::buildFUID(long runNumber, long eventNumber) const {
    return std::to_string(runNumber) + m_fuidSeparator.value() +
std::to_string(eventNumber);
}

std::string HepMCHeaderWriterAlg::buildMetadata(const HepMC3::GenEvent& event) const {
    std::ostringstream metadata;
    metadata << "n_particles=" << event.particles().size() << ";n_vertices=" <<
event.vertices().size();
    return metadata.str();
}

void HepMCHeaderWriterAlg::addHeaderAttributes(HepMC3::GenEvent& event, long seed)
const {
    const long runNumber = m_runNumber.value();
```

```
    const long eventNumber = resolveEventNumber(event);
    const std::string fuid = buildFUID(runNumber, eventNumber);
    const std::string metadata = buildMetadata(event);

    event.set_event_number(static_cast<int>(eventNumber));
    event.add_attribute("sampo_run_number", makeStringAttribute(runNumber));
    event.add_attribute("sampo_event_number", makeStringAttribute(eventNumber));
    event.add_attribute("sampo_seed", makeStringAttribute(seed));
    event.add_attribute("sampo_fuid", makeStringAttribute(fuid));
    event.add_attribute("sampo_metadata", makeStringAttribute(metadata));
}

DECLARE_COMPONENT(HepMCHeaderWriterAlg)
```

Листинг А.2: Файл реализации HepMCHeaderWriterAlg.cpp

```
import dataclasses

from GaudiConfig2 import Configurables as C
from SampoConfiguration.JobConfigurator import JobConfigurator


@dataclasses.dataclass
class HepMCHeaderSettings:
    FilePath: str = "HepMC_with_header.dat"
    RunNumber: int = 1


def registerHepMCHeaderSettings(configBuilder):
    configBuilder.registerSettings(HepMCHeaderSettings, "HepMCHeader")


def HepMCHeaderWriterAlgCfg(config, filePath=None, runNumber=None):
    jc = JobConfigurator()

    writer = C.HepMCHeaderWriterAlg()
    writer.Cardinality = 1

    writer.filePath = filePath if filePath is not None else
config.HepMCHeader.FilePath

    writer.runNumber = runNumber if runNumber is not None else
config.HepMCHeader.RunNumber

    jc.addAlgorithm(writer)
    return jc
```

Листинг А.3: Конфигурация алгоритма HepMCHeaderConfig.py