МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «СЕВЕРО-ОСЕТИНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ им. К.Л. ХЕТАГУРОВА»

Факультет: Физико-технический
Кафедра: Физики конденсированного состояния

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

«Разработка каталога физических событий эксперимента SPD на ускорителя частиц NICA: Оптимизация загрузки и чтения данных»

Исполнитель:
Бакалавр, 4 курс, очная форма обучения,
Направление подготовки: 03.03.02. Физика
Будтуева Замира Алановна
Научный руководитель:
к.ф.м.н., доцент, доцент кафедры физики конденсированного состояния
Тваури Инга Васильевна
Научный консультант:
к.ф.м.н., с.н.с. лаборатории ядерных проблем ОИЯИ
Прокошин Фёдор Валерьевич

«Допущен к защите»

Зав. кафедрой _____(профессор, доктор физико-математических наук, заведующий кафедрой физики конденсированного состояния Тамерлан Таймуразович Магкоев)

«_____»_____2024г.

Владикавказ 2024

СОДЕРЖАНИЕ

РЕФЕРАТ.....3
ВВЕДЕНИЕ.....5
Глава 1. Физика элементарных частиц.....7
1.1. Ускорители частиц: эксперимент на коллайдере и эксперимент с фиксированной мишенью.....7
1.2. Экспериментальные установки в физике элементарных частиц.....11
Вывод по главе 1.....15
Глава 2. Анализ данных.....16
2.1. Основные этапы анализа данных.....16
2.1.1. Отбор событий.....17
2.1.2. Реконструкция физических объектов и процессов.....17
2.1.3. Физический анализ.....19
2.2. Системы обработки данных.....21
2.2.1. DAQ.....21
2.2.2. SPD On-line Filter.....22
2.2.3. Event Index.....24
Вывод по главе 2.....27
Глава 3. Оптимизация загрузки и чтения данных о событиях.....28
3.1. Основные этапы работы.....28
3.1.1. Генерация событий.....28
3.1.2. Реализация чтения и записи данных о событиях.....29
3.2. Оптимизация процессов генерации, чтения и записи событий.....29
3.2.1. Тестирование модулей.....30
3.2.2. Конфигурации PostgreSQL.....36
3.2.3. Профилирование.....37
3.2.4. Дальнейшие задачи.....41
Вывод по главе 3.....43
ЗАКЛЮЧЕНИЕ.....44
СПИСОК ЛИТЕРАТУРЫ.....46

РЕФЕРАТ

Дипломная работа на тему “Разработка каталога физических событий эксперимента SPD на ускорителя частиц NICA: Оптимизация загрузки и чтения данных” состоит из введения, 3 разделов, выводов, заключения и списка используемых источников, который включает в себя 15 ссылок. Работа изложена на 44 страницах печатного текста, 7 таблиц и 12 рисунков.

Целью данного исследования является разработка и совершенствование методов обработки данных в области физики частиц и высоких энергий для повышения точности и эффективности в идентификации и измерении физических явлений, а также для проверки и уточнения теоретических моделей.

Основные задачи исследования заключаются в следующем:

1. Изучение экспериментальных установок для регистрации частиц в современной физике;
2. Рассмотрение систем отбора событий и реконструкции объектов, методов обработки и анализа данных, используемых в физике частиц и высоких энергий;
3. Разработка новых систем для хранения экспериментальных данных;
4. Оптимизация систем хранения и обработки данных для обеспечения эффективного доступа и анализа больших объёмов данных.

Объектом исследования являются экспериментальные данные, полученные в результате работы крупных экспериментальных коллайдеров, таких как NICA, а также данные, зарегистрированные различными детекторами частиц.

Предметом исследования являются системы, разрабатываемые для отбора, анализа и хранения данных о событиях, полученных в результате эксперимента.

Ключевые слова: Event Index, SPD, NICA, DAQ, On-line filter, ускорители частиц, физика частиц, анализ данных, оптимизация.

Результаты проведённых исследований были представлены в виде доклада на 10-ой Международной конференции “Распределенные вычисления и GRID технологии в науке и образовании” 2023, Весенней школе по информационным технологиям ОИЯИ 2024 (диплом I степени), а также опубликованы в статье “SPD EventIndex” в журнале “Физика элементарных частиц и атомного ядра” [1, 2, 3]. Журнал «Физика элементарных частиц и атомного ядра» входит в перечень рецензируемых научных изданий ВАК России, рекомендованных для публикаций статей, содержащих материалы кандидатских и докторских диссертаций.

ВВЕДЕНИЕ

Актуальность исследования:

В современной физике частиц и высоких энергий анализ данных играет ключевую роль в понимании фундаментальных вопросов о природе материи и взаимодействиях между частицами. Экспериментальные исследования в этой области часто связаны с использованием крупных коллайдеров, таких как Большой адронный коллайдер (БАК) в CERN или ускорительный комплекс NICA (Nuclotron based Ion Collider fAcility) в г. Дубне, а также с использованием сложных детекторов, способных регистрировать различные типы частиц и их взаимодействия.

Процесс анализа данных экспериментов играет важную роль в физике частиц и высоких энергий. Методы обработки и анализа данных включают в себя применение статистических методов, машинного обучения, а также разработку специализированных программных инструментов для идентификации и измерения различных явлений. Результаты этих анализов позволяют проверять теоретические модели, искать новые частицы или явления, а также проверять фундаментальные принципы физики на экспериментальном уровне.

Процесс анализа данных проходит в несколько этапов: отбор событий, реконструкция физических объектов и собственно физический анализ. Каждый из этих этапов включают в себе несколько шагов для наиболее точного анализа данных.

Экспериментальные данные могут быть необходимы для нескольких физических анализов, и доступ к ним может понадобиться множество раз. При этом из огромного количества событий отбираются только те, что представляют наибольший интерес для исследований.

Для анализа данных о событиях, разрабатываются и используются системы отбора данных, системы индексации событий и распределённые системы хранения и обработки данных.

Целью данного исследования является разработка и совершенствование методов обработки данных в области физики частиц и высоких энергий для повышения точности и эффективности в идентификации и измерении физических явлений, а также для проверки и уточнения теоретических моделей.

Основные задачи исследования заключаются в следующем:

1. Изучение экспериментальных установок для регистрации частиц в современной физике;
2. Рассмотрение систем отбора событий и реконструкции объектов, методов обработки и анализа данных, используемых в физике частиц и высоких энергий;
3. Разработка новых систем для хранения экспериментальных данных;
4. Оптимизация систем хранения и обработки данных для обеспечения эффективного доступа и анализа больших объёмов данных.

Объектом исследования являются экспериментальные данные, полученные в результате работы крупных экспериментальных коллайдеров, таких как NICA, а также данные, зарегистрированные различными детекторами частиц.

Предметом исследования являются системы, разрабатываемые для отбора, анализа и хранения данных о событиях, полученных в результате эксперимента.

Результаты проведённых исследований были представлены в виде доклада на 10-ой Международной конференции “Распределенные вычисления и GRID технологии в науке и образовании” 2023, Весенней школе по информационным технологиям ОИЯИ 2024 (диплом I степени), а также опубликованы в статье “SPD EventIndex” в журнале “Физика элементарных частиц и атомного ядра” [1, 2, 3]. Журнал «Физика элементарных частиц и атомного ядра» входит в перечень рецензируемых научных изданий ВАК России, рекомендованных для публикаций статей, содержащих материалы кандидатских и докторских диссертаций.

Глава 1. Физика элементарных частиц

Физика элементарных частиц — это часть физики, которая занимается изучением структуры и свойства элементарных частиц и их взаимодействия. Также она носит название субъядерной физики.

В экспериментальной физике частиц и высоких энергий можно выделить несколько направлений в зависимости от источника исследуемых объектов.

- Ускорители, обеспечивающие большие потоки частиц в широком диапазоне энергий
- Космические лучи различного происхождения, характеризуемые относительно малыми потоками, но большим предельным значением энергий, недостижимым на ускорителях.
- Реакторы, в том числе импульсные.

Остановимся на ускорителях, которые могут различаться по принципу и технологиям ускорения и удержания пучка частиц, но среди могут быть выделены две большие группы: ускорители с фиксированной мишенью и коллайдеры, в которых сталкиваются встречные пучки.

1.1. Ускорители частиц: эксперимент на коллайдере и эксперимент с фиксированной мишенью

Ускорители частиц – это ряд устройств физики элементарных частиц, который необходим для изучения структуры материи на самых малых масштабах (продукты, получаемые в результате столкновения частиц) и при самых высоких энергиях [4].

Эти устройства работают на взаимодействии заряженных частиц с магнитным и электрическим полями. Электрическое поле необходимо для ускорения частиц, в то время как магнитное поле, с помощью силы Лоренца, отклоняет эти частицы, не меняя их энергии.

Ускорители частиц классифицируются по своей конструкции на линейные, в которых ускорительные промежутки частицы проходят лишь один

раз, и циклические, где частиц проходят ускоряющие промежутки множество раз, двигаясь по замкнутой траектории.

К классу линейных ускорителей относятся:

1. Высоковольтные ускорители (ускорители прямого действия) – класс устройств, в которых частицы ускоряются постоянным электрическим полем и прямолинейно двигаются по вакуумной камере, вдоль которой располагаются ускоряющие электроды. Преимуществом этого типа устройств выступает возможность получения малого разброса по энергиям частиц.

2. Линейные резонансные ускорители (линаки) – тип устройств, ускорение в которых происходит благодаря электрическому полю высокочастотных резонаторов. Преимуществом выступает отсутствие потерь энергии на излучение.

3. Линейные индукционные ускорители – устройства, ускорение в которых идёт за счёт вихревого электрического поля. Это происходит с помощью ферромагнитного кольца, которое установлено вдоль оси пучка.

К классу циклических ускорителей относятся:

1. Бетатрон – тип циклического ускорителя, в котором частицы ускоряются за счёт вихревого электрического поля, индуцируемого изменением магнитного потока. Они применяются для ускорения заряженных частиц до энергий 10-100 МэВ.

2. Циклотрон – циклический ускоритель, в котором ускорение частиц происходит за счёт переменного электрического и постоянного магнитного полей. Основным преимуществом циклотрона является ускорение частиц до значительных энергий за короткое время.

3. Микротрон – тип циклического ускорителя, который использует магнитное поле для многократного направления частиц через одну и ту же ускорительную структуру, увеличивая их энергию. Преимущество – достижение высоких энергий при компактных размерах установки.

4. Фазотрон (синхроциклотрон) – это тип циклического ускорителя, в котором частицы ускоряются за счёт высокочастотного электрического поля,

синхронизированного с их движением. Преимущество заключается в способности увеличивать магнитное поле по мере роста энергии для поддержания постоянного радиуса орбиты частиц.

5. Синхротрон – тип циклического ускорителя, в котором частицы ускоряются до высоких энергий с помощью изменяющегося во времени магнитного поля, синхронизированного с их скоростью.

6. Синхрофазотрон – тип циклического ускорителя, в котором магнитное поле увеличивается по мере ускорения частиц, чтобы удерживать их на постоянной орбите, а электрическое поле синхронизируется с движением частиц, чтобы ускорять их.

Ускорители частиц широко применяются для экспериментов на коллайдерах, которые предназначены для изучения продуктов соударений встречных пучков частиц. Используя такие установки, исследователи могут придавать элементарным частицам высокую кинетическую энергию и направлять их навстречу друг другу для их столкновения.

Помимо столкновений пучков частиц на ускорителях проводятся эксперименты с фиксированной мишенью (fixed-target experiment) [8]. Суть таких экспериментов заключается в том, что пучок заряженных частиц (электронов или протонов) ускоряется до релятивистской скорости и сталкивается с неподвижной мишенью, которой может быть твёрдый блок, жидкая или газообразная среда.

В чём же принципиальное отличие эксперимента с фиксированной мишенью от эксперимента на коллайдере?

Энергия, которая затрачивается в эксперименте с фиксированной мишенью, в 4 раза меньше, чем энергия, затрачиваемая на коллайдере при разгоне двух пучков частиц. Более того, кинетическая энергия частиц не полностью используется на создание новых частиц, часть этой энергии уходит на придание скорости вторичным частицам, которые летят вдоль направления первичных частиц. Та энергия, которая идёт на создание новых частиц в

эксперименте с фиксированной мишенью примерно пропорциональная квадратному корню из налетевших частиц:

$$E = \sqrt{2(\gamma + 1)}mc^2 \quad (1)$$

В случае, с экспериментом на коллайдере кинетическая энергия каждой частицы полностью перейдёт в энергию возникших частиц:

$$E = 2\gamma mc^2 \quad (2)$$

$$\gamma = \frac{\varepsilon}{mc^2} = \frac{1}{\sqrt{1-\beta^2}} \quad (3)$$

При взаимодействии пучка с фиксированной мишенью продукты реакции направляются вперёд по движению налетевшей частицы в узкий конус (Рис. 1) и раствор этого конуса тем меньше, чем выше энергия частиц.

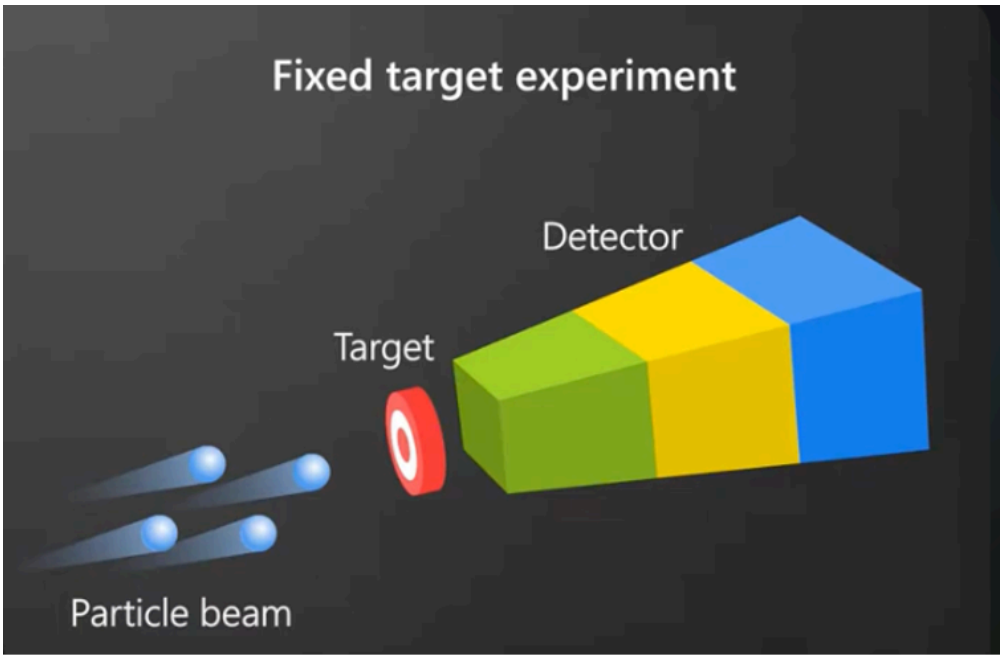


Рисунок 1. Схема эксперимента с фиксированной мишенью

Поэтому, например, поиск бозона Хиггса в эксперименте с неподвижной мишенью – это совершенно безнадёжная задача, так как все продукты реакции будут лететь строго по направлению летевшего пучка и освободиться от фона первичных частиц будет невозможно. В случае эксперимента на относительно низких энергиях можно поставить эксперимент на фиксированной мишени и создать детектор, который достаточно эффективно воспримет частицы, которые

родились в результате взаимодействия. Но при этом, при изменении энергии угол разлёта этих вторичных частиц будет меняться и с этим будет меняться эффективность детектора. Коллайдерный эксперимент свободен от этого недостатка. В независимости от энергии сталкиваемых частиц продукты реакции разлетаются во все стороны и могут быть достаточно эффективно зарегистрированы. Недоступна для регистрации оказывается только очень маленькая область, которая направлена вдоль налетающих частиц встречных пучков. Поэтому можно сделать детектор, эффективность которого не зависит от энергии, и который позволят перехватить практически полный поток вторичных частиц.

Однако эксперимент с фиксированной мишенью имеет значительное преимущество для экспериментов, которые требуют более высокой светимости. Светимость – темп производства исследуемых частиц, нормированный на сечение реакций образования этих частиц:

$$L = \frac{N_{enent}}{\sigma} = \frac{1}{\sigma}\frac{dN}{dt} \quad (4)$$

N_{enent} – число зарегистрированных событий;

σ – сечение реакции – это вероятность рождения частицы в данном столкновении.

1.2. Экспериментальные установки в физике элементарных частиц

Коллайдеры используются в большинстве современных экспериментах при высоких энергиях, тогда эксперименты с фиксированной мишенью широко распространены в нейтринных экспериментах, и в других случаях, когда требуется высокая светимость.

Создаваемый в Объединённом институте ядерных исследований (ОИЯИ), расположенном в городе Дубна, ускорительный комплекс NICA (Nuclotron-based Ion Collider fAcility) (Рис. 2) представляет собой коллайдер, предназначенный для ускорения пучков тяжелых ионов (вплоть до ядер золота) либо поляризованных пучков протонов и дейтронов [9].

Рисунок 2. Ускорительный комплекс NICA, г. Дубна

В области пересечения пучков будут установлены две детекторных установки. Для исследований в области релятивистской ядерной физики в столкновении пучков тяжелых ионов предназначена установка MPD — Multi Purpose Detector (многоцелевой детектор). Для исследования спиновой физики в области высоких и средних энергий на поляризованных пучках создается детекторный комплекс SPD – Spin Physics Detector (Рис. 3) [7] .

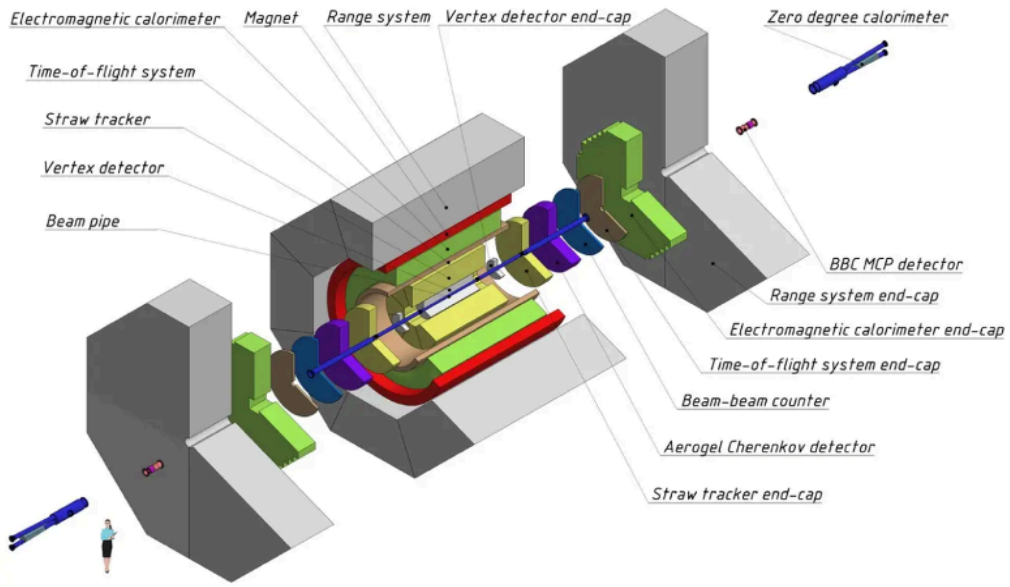


Рисунок 3. Устройство детектора SPD

Конструкция современных детекторных установок зависит от физических задач конкретного эксперимента, вместе с тем есть общие принципы устройства

характерные для универсальных детекторов на встречных пучках. Цель такого устройства заключается в том, чтобы максимально полно уловить продукты взаимодействия и затем как можно точно восстановить их параметры. Более того, 4π геометрия (охват телесного угла вокруг точки столкновения) и герметичность установки позволяют восстанавливать параметры таких «невидимых» частиц, как, например, нейтрино и промежуточных продуктов столкновения с коротким временем жизни.

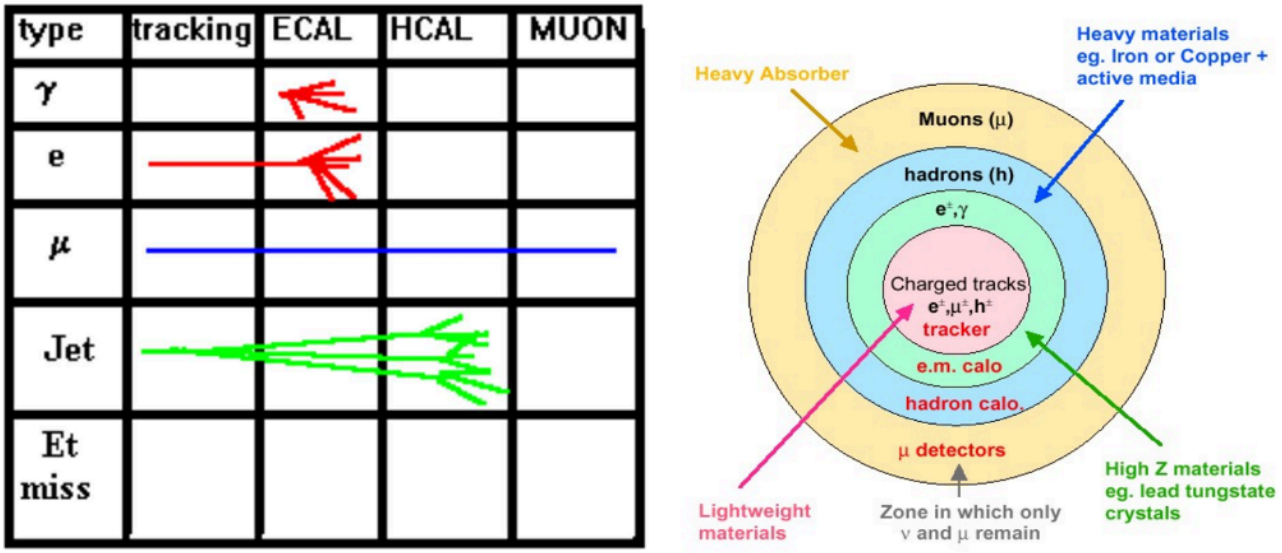


Рисунок 4. Структура универсального детектора

Установки, как правило, имеют осевую симметрию (Рис. 4) [5]. В центральной части находится трековая система (трекер), заключенная в магнитное поле, которое закручивает электроны, g-кванты и струи. Её задача — максимально точно определить траектории заряженных частиц, по изгибу которых можно вычислить их импульсы.

За трекером находятся электромагнитный и адронный калориметры, задача которых – измерить энергию частиц. Размеры и материалы калориметров подобраны таким образом, чтобы электроны и гамма-кванты теряли максимальную энергию в толще электромагнитного калориметра, а продукты лавинообразного распада адронов (адронные струи) — в толще адронного калориметра. За пределы адронного калориметра в основном проникают только

мюоны, попадающие в мюонный детектор, и «невидимые» нейтрино, параметры которых в некоторых случаях могут быть восстановлены из энергетического баланса события.

Каждая детекторная система состоит из множества элементов, число которых может достигать сотен тысяч (например, в трековых системах, где требуется большое разрешение).

Сигнал с детекторного элемента преобразуется в цифровой формат, содержащий информацию об одном или нескольких параметрах сигнала: амплитуде, длительности, заряде.

Таким образом «сырая» картина события представляет собой набор цифровых значений, относящихся к различным элементам детектора. Сигналы с детекторных элементов поступают в систему сбора данных (DAQ) по электронным «каналам», где могут проходить первичный отсев, обработку и мультиплексирование.

Объемы информации, поступающие по каналам сбора данных как правило огромны, что делает невозможным сохранение всех данных, необходим предварительный отбор. При классическом подходе на основании информации от триггерных детекторов вырабатывается триггерный сигнал, помечающий событие как полезное. Такие события поступают в обработку, все остальные события отбрасываются.

Вывод по главе 1

Физика частиц и высоких энергий исследует наиболее фундаментальные свойства материи, а именно — свойства основных компоненты, из которых состоит окружающий мир и взаимодействия между ними. Исследования в этой области позволяют пролить свет на самые ранние этапы формирования вселенной, а также предлагает подход к разрешению фундаментальных проблем современной космологии — темной материи и темной энергии. Следует отметить, что фундаментальные исследования как правило оказывают громадный эффект на развитие прикладного знания в различных областях. Кроме того, эксперименты в современной физике стимулируют развитие в инженерных и информационных технологиях, необходимых для их проведения. Хорошим примером может служить всемирная сеть World Wide Web, разработанная в CERN для хранения и обмена информацией в рамках международного научного сотрудничества.

В экспериментальной физике частиц и высоких энергий использование коллайдеров позволяет получить большие энергии в центре масс по сравнению с фиксированной мишенью, и получить более удобную геометрию событий, в особенности при высоких энергиях, тогда как на фиксированных мишенях достигается большее столкновений (светимость). Коллайдеры используются в большинстве современных экспериментах при высоких энергиях, тогда эксперименты с фиксированной мишенью широко распространены в нейтринных экспериментах, и в других случаях, когда требуется высокая светимость.

Глава 2. Анализ данных

2.1. Основные этапы анализа данных

Анализ данных в физике частиц и высоких энергий играет ключевую роль в разъяснении результатов экспериментов и подтверждении теоретических моделей. Современные эксперименты генерируют огромные объёмы данных, которые требуют сложной обработки и анализа и могут иметь различный формат.

Данные, получаемые непосредственно с установки, называются «сырыми» (raw data), они содержат сигналы с детекторов. Данные (реконструированные), прошедшие обработку и анализ, содержат в себе информацию о взаимодействиях элементарных частиц, их кинематике, энергетических характеристиках и других немаловажных физических параметрах. В моделированных данных также содержится информация о “настоящих” частицах (“truth particles”).

Спектр задач для которых производятся анализы данных в физике частиц достаточно широк. Это измерение и уточнение параметров Стандартной модели и характеристик ее элементов, обнаружение редких процессов и композитных объектов, изучение различных состояний ядерного вещества (таких как кварк-глюонная плазма), а также поиск отклонений от Стандартной модели, которые могут указывать на новую физику. Для достижения этих цели существуют несколько этапов анализа данных (Рис. 5) [6].

Рисунок 5. Схема этапов анализа данных в физике частиц и высоких энергий

2.1.1. Отбор событий

Первым этапом, играющим важную роль в процессе анализа данных экспериментов физики частиц и высоких энергий, является отбор событий (event selection), в которых могут проявляться физические процессы, интересующие исследователя.

Отбор производится в несколько этапов. Непосредственно при наборе данных происходит фильтрация сигналов с детекторов по амплитудным и временным параметрам, затем данные поступают на этап онлайн отбора.

Это могут быть либо триггерные системы, отбирающие детекторные объекты и события, в целом, до записи в системы хранения, либо, при без триггерной организации системы набора данных, онлайн фильтры, анализирующие данные за определенные временные промежутки. В обоих случаях данные проверяются на соответствие каким-то наборам критериев и результаты проверки сохраняются для дальнейшего использования.

Как правило, экспериментальные данные используются для нескольких физических анализов, для которых представляют интерес разные физические процессы с разной вероятностью проявления в условиях эксперимента. Отбор «сигнальных» событий, которые могут содержать такие процессы — сложная задача, составляющая существенную часть работы в рамках физического анализа. В конкретном анализе из миллионов событий могут быть отобраны как относительно большие наборы — десятки и сотни тысяч, так и достаточно маленькие - десятки и сотни.

2.1.2. Реконструкция физических объектов и процессов

Вторым этом анализа данных является реконструкция физических объектов и процессов. Это наиболее сложный этап, включающий в себя преобразование «сырых» данных, собранных детекторами, в полезную информацию о частицах и их взаимодействиях.

Реконструкция включает в себя несколько шагов, первым из которых является калибровка – установление соответствия между откликом детектора и физическими величинами, которые он измеряет. Калибровка проводится постоянно в течение всего эксперимента. В процессе этого происходят: юстировка, выравнивание коэффициентов усиления, измерение скорости дрейфа в дрейфовых камерах, синхронизация и калибровка t_0 (опорный момент времени, соответствующий моменту столкновения частиц или началу эксперимента, от которого отсчитывается время всех остальных событий), а также удаление фоновые шумов и артефактов, которые могут исказить результаты анализа.

После калибровки начинается процесс реконструкции треков частиц, Этот процесс включает в себя восстановление траекторий частиц на основе данных, полученных с трековых детекторов. Алгоритмы реконструкции треков, таких как метод Хафа, позволяют определить параметры движения частиц, включая их импульсы и направления. Далее происходит реконструкция вершин, в ходе которой определяется место взаимодействия частиц. Для сложных событий, включающих в себя множество взаимодействий, используются алгоритмы, позволяющие точно определять каждую вершину и связанные с ней треки.

Следующий шаг реконструкции физических объектов и процессов является идентификация типа частиц с использованием информации с детекторов. На этом этапе в последнее время постепенно начали внедрять технологии машинного обучения.

И заключающими шагами реконструкции являются: оценка энергетических и кинематических параметров с помощью калориметров, коррекция и валидация данных. В процессе коррекции вносятся поправки для учёта различных систематических ошибок, таких как неидеальная калибровка детекторов, потери энергии или частиц, вторичные взаимодействия и т.д. Валидация заключается в сравнении реконструированных данных с моделями и

симуляциями, которое позволяет убедиться в точности реконструкции и корректности полученных данных.

2.1.3. Физический анализ

Третьим этапом анализа данных в физике частиц является физический анализ, который позволяет интерпретировать результаты и делать выводы о физических процессах, происходящих на субатомном уровне. Два важнейших инструмента, которые используются на этом этапе – это статистические методы и метод Монте-Карло, которые помогают в обработке данных, оценке неопределённостей и моделировании сложных физических процессов [10, 11].

Статистические методы позволяют оценивать значимость наблюдаемых явлений и делать выводы о свойствах частиц и их взаимодействиях. К основным задачам, решаемым с помощью статистических методов, относятся:

1. Оценка значений физических параметров на основе экспериментальных данных с применением методов правдоподобия (функция, описывающая вероятность нахождения данных при заданных значениях параметров) и наименьших квадратов (минимизация суммы квадратов отклонения наблюдаемых значений от предсказанных моделью)

2. Проверка гипотез о наличии или отсутствии сигналов новых частиц и процессов, с применением критерия Стьюдента (t-критерий используется для сравнения средних значений двух выборок и определения статистической значимости различий между ними) и критерия (позволяет оценить степень наблюдаемых частот ожидаемым), для оценки согласия данных с теоретической моделью.

3. Вычисление p-значений (мера для принятия или отклонения нулевой гипотезы) для определения вероятности наблюдения экспериментального результата при условии справедливости нулевой гипотезы (или нуль-гипотеза – основная проверяемая гипотеза; фон).

4. Использование метода ложноположительных отклонений (false discovery rate) для учёта множественных проверок гипотезы.

5. Разделение и анализ источников ошибок, возникающих из-за ограниченной статистики выборки и систематических неопределённостей в измерениях и моделях.

Методы Монте-Карло широко применяются для моделирования и анализа сложных физических систем, где аналитические решения труднодостижимы. Эти методы основываются на использовании случайных чисел и статистических выборок для имитации физических процессов. Основные задачи методов Монте-Карло в физике частиц заключаются в следующем:

1. Создание детализированных ситуаций работы детекторов для понимания их отклика на прохождение частиц.

2. Включение всех физических процессов, таких как рассеяние, поглощение и генерация вторичных частиц, для точного воспроизведения реальных условий эксперимента.

3. Имитация столкновений частиц с использованием генераторов событий, таких как:

- HERDWIG (предоставляет полное генераторное моделирование процессов столкновения высоких энергий),
- PYTHIA (программный пакет инструментов для моделирования и симуляции процессов столкновения элементарных частиц при высоких энергиях на ускорителях частиц методом Монте-Карло),
- SHERPA (генератор полного цикла, созданный для генерации высокоэнергетических реакций частиц в столкновения),
- GEANT (программный пакет инструментов для моделирования детекторов и симуляции прохождения частиц через них) [12].

4. Исследование различных гипотетических сценариев и проверка теоретических моделей путём сравнения симуляционных данных с экспериментальными. А также проведение многократных симуляций с вариациями параметров для оценки диапазона возможных отклонений

5. Изучение влияния различных факторов, таких как разрешение детекторов, на измеряемые величины.

2.2. Системы обработки данных

2.2.1. DAQ

Каждая детекторная система состоит из множества датчиков, число которых может достигать сотен тысяч (например в трековых системах, где требуется большое разрешение). Сигнал с каждого датчика преобразуется в цифровой формат, содержащий информацию об одном или нескольких параметрах сигнала (амплитуде, длительности, заряде) в зависимости от типа датчика. Сигналы с детекторных элементов поступают в систему сбора данных (DAQ) по электронным «каналам», где могут проходить первичный отсев, обработку и мультиплексирование.

Объемы информации поступающие по каналам сбора данных как правило огромны, что делает невозможным сохранение всех данных, необходим предварительный отбор. При классическом подходе на основании информации от триггерных детекторов вырабатывается триггерный сигнал, помечающий событие как полезное. Такие события поступают в обработку, все остальные события отбрасываются. Триггерная система (Рис. 6) используется например на установке Atlas на большом адронном коллайдере. Критерии отбора событий определяются для нужд различных физических анализов, их набор постоянно оптимизируется чтобы обеспечить баланс между числом полезных событий и пропускной способностью системы хранения данных.

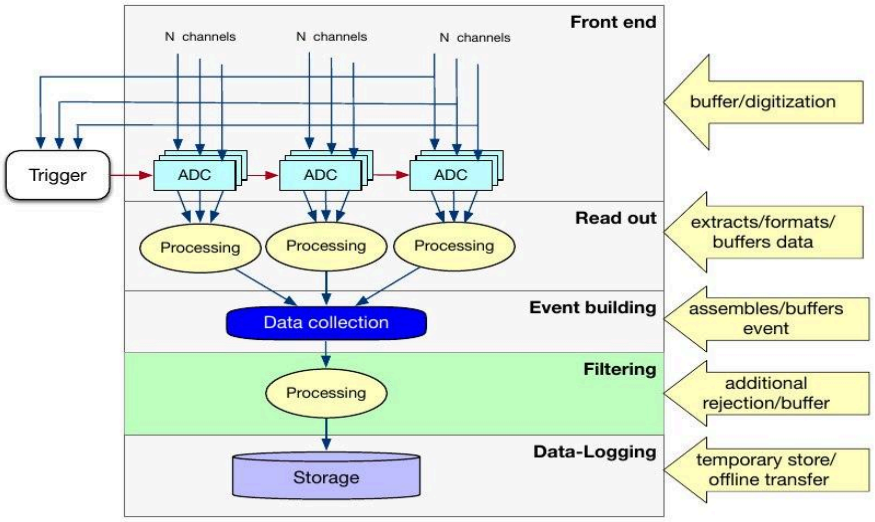


Рисунок 6: Схема триггерной DAQ

Альтернативный подход предполагает использование бестриггерной системы (Рис. 7), позволяющей использовать сложные алгоритмы отбора событий для снижения систематических погрешностей.

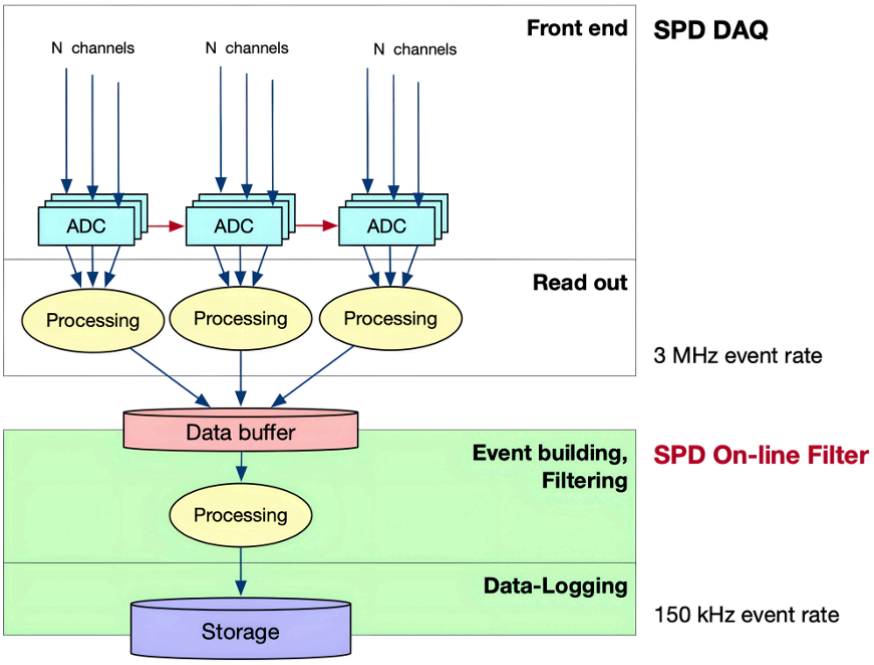


Рисунок 7: Схема бестриггерной DAQ

2.2.2. On-line filter

Суммарный поток данных, ожидаемый с детектора SPD оценивается в 20 Гб/с (это 200 петабайт в год), исходя из того, что максимальная ожидаемая частота столкновений частиц в коллайдере примерно 3 МГц. Для отбора сигнальных событий необходимы восстановление импульса и вершин, что делает невозможным реализацию аппаратной триггерной системы со сравнительно простыми критериями отбора для детектора SPD.

В триггерной системе сигналы с датчиков сразу ассоциируются с каким-либо событием, а в бестриггерной системе из полученного набора сигналов разных датчиков надо выявить события и из них отобрать нужные. Для реализации бестриггерной системы разрабатывается специальная

вычислительная установка, которая имеет название SPD On-line Filter (Рис.7) [15].

Система On-line Filter нужна для решения следующих задач:

- Раскодировка полученных от DAQ данных
- Выделение события
- Фильтровка событий. Отсеиваются «скучные» события
- Объединение событий в файлы, и далее в наборы данных
- Подготовка данных для системы контроля качества данных

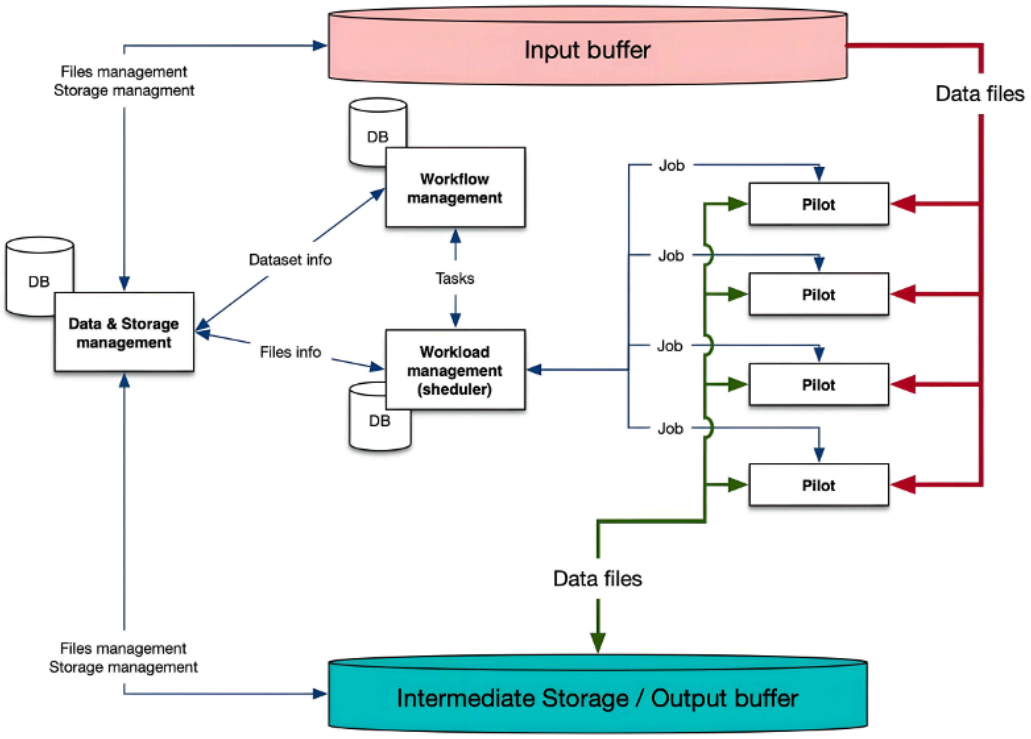


Рисунок 7: Архитектура системы SPD On-line Filter

SPD On-line filter использует технологии машинного обучения для реконструкции большей части событий, поскольку использование классических алгоритмов не позволяет обеспечить требуемую производительность.

После он-лайн фильтра ожидается двадцатикратное сокращение потока данных, примерно до 20-23 тысяч событий в секунду, что приблизительно равно 30 миллиардам событий в год.

2.2.3. Event Index

Данные о событиях, которые представляют интерес для конкретного физического анализа могут храниться среди множества файлов, разбросанных по серверам распределенных систем хранения. Поиск набора событий путем сканирования петабайт данных требует больших затрат вычислительных мощностей. В ходе анализа доступ к этим событиям может понадобится множество раз, и каждый раз придется повторять сканирование.

Для оптимизации доступа события могут быть проиндексированы, при этом ссылки на файлы, в которых хранятся эти события, а также небольшой набор существенных параметрах сохраняется в относительно небольшой базе данных, что дает возможность быстрого поиска и отбора идентификаторов событий и получения доступа к их данным.

Система, предназначенная для индексации данных, хранения информации о событиях и обеспечения доступа к этим данным посредством программных интерфейсов и интерфейсов пользователя получила название Event Index (Рис. 8) [1, 1].

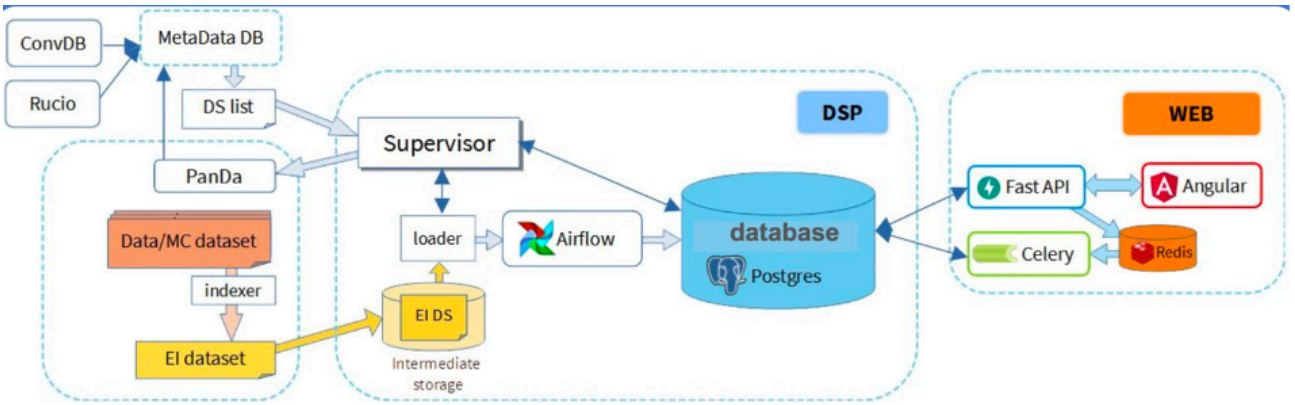


Рисунок 8: Архитектура системы Event Index

Разрабатываемая для эксперимента SPD система имеет ряд областей применения:

1. Подсчет и предварительный отбор событий для анализа по результатам работы он-лайн фильтра и набора важных параметров события (каких именно — предстоит определить группе физического анализа)

2. Доступ по идентификаторам к наборам событий, полученным по результатам процедуры отбора в рамках физических анализов

- Создание «виртуальных датасетов» (датасет – набор данных) из отобранных событий

3. Проверка целостности наборов события в датасетах:

- Обнаружение не читаемых/поврежденных либо потерянных событий
- Обнаружение дублирующихся событий внутри файлов и датасетов

Система Event Index разрабатывается как комплексная информационная система, обеспечивающая:

1. Получение информации о событиях эксперимента или смоделированных данных путём индексирования файлов, которые содержат данные о событиях

2. Передачу и запись в базы данных этой информации

3. Доступ к информации для программ анализа обработки и анализа данных через приложения и API, а также для пользователей с помощью интерактивных и асинхронных интерфейсов.

Запись событий в систему Event Index должно содержать в себе некоторые поля:

1. run_numder – номер run-а (запуска)

2. event_number – номер события в run-е

3. olf_result – информация о решениях он-лайн фильтра, в виде битовой маски

4. fuid_raw – уникальный идентификатор файла с «сырыми» данными формата RAW, в котором содержится данное событие. UUID файла даёт возможность получить доступ к нему через распределённую систему хранения

5. dsid_raw – идентификатор датасета, к которому относится данный файл

По мере обработки данных будут создаваться новые экземпляры восстановленных событий в формате оптимизированным для физического анализа – AOD. Идентификаторы файлов и датасетов содержащих эти

экземпляры будут добавлятся в запись события. В запись можно добавлять важные параметры события, которые можно использовать для классификации и отбора.

Для хранения и обработки данных была выбрана система управления базами данных (СУБД) PostgreSQL, особенностями которой выступают многопоточность и возможность обрабатывать большие объёмы данных.

Для создания интерфейса, обменивающегося данными, был использован RESTful API, а для создания сервера – Flask (микрофреймворк для языка программирования Python). Визуальная часть (frontend) разработан с помощью фреймворка Angular.

RabbitMQ представляет собой брокер сообщений, позволяющий отправлять, получать и направлять сообщения между компонентами приложений в асинхронном режиме. Celery – система, позволяющая выполнять операции в фоновом режиме. Совместное использование этих двух инструментов способствует улучшению системы для асинхронной обработки задач. FastAPI сосредотачивается на обработке HTTP-запросов, в то время как обработка задач передаётся RabbitMQ и Celery, давая гибкую и модульную архитектуру приложения.

Интеграция всех этих инструментов дала возможность создать гибкий и мощный интерфейс для обмена данными, давая более эффективную обработку информации в системе Event Index.

Выводы по главе 2

Анализ данных в физике частиц и высоких энергий представляет собой сложный и многоэтапный процесс, включающий в себя несколько ключевых этапов.

В начале производится отбор событий, при котором события фильтруются так, чтобы остались только те, которые содержат в себе интересующую исследователей информацию. Далее следует реконструкция физических объектов и процессов, при котором происходит преобразование «сырых» данных в более удобный формат, который позволяет идентифицировать и изучать события. В конце производится физический анализ, включающий в себя статистическую обработку данных и сравнение полученных результатов с теоретическими моделями.

Для реализации каждого этапа анализа используются специализированные системы обработки данных, такие как система сбора данных (DAQ), система фильтрации событий (SPD On-line Filter) и система индексации событий (Event Index), обеспечивающие эффективную обработку и анализ больших объёмов данных, генерируемых современными ускорителями частиц.

Тщательное проведение всех этапа анализа данных играет важную роль в экспериментальной физике частиц и высоких энергий, позволяя учёным изучать фундаментальные вопросы о природе материи и взаимодействиях между ее элементарными составляющими.

Глава 3: Event Index: Оптимизация загрузки и чтения данных о событиях

3.1. Основные этапы работы

3.1.1. Генерация данных

Для тестирования прототипа системы генерируются наборы данных Event Index. Формат этих наборов - JSON, и он не зависит от формата, в котором будут храниться данные с детектора. Для каждого события генерируются идентификаторы (run_number и event_number), случайный olf_result, а также fuid_raw и dsid_raw. В дальнейшем, на основе этих наборов создаются псевдоданные для AOD файлов и затем записываются в таблицы в базе данных.

База данных разрабатываемой системы индексации включает в себя основные две таблицы: events (Таблица 1) и datasets (Таблица 2). Таблица events хранит в себе записи о каждом событии, а таблица datasets хранит информацию о наборах данных (датасетах). ID датасета служит внешним ключом для таблицы событий.

run_numder	event_numder	olf_result	dsid_raw	fuid_raw
280308061	12	21810	1	dfcdd203-a473-4c0c-909d-a35252512595
280308061	30	17705	1	dfcdd203-a473-4c0c-909d-a35252512595
280308061	43	19125	1	dfcdd203-a473-4c0c-909d-a35252512595

Таблица 1: events

dsid_raw	dsn_raw
1	data28_pp_9p4GeV.280308061.RAW.t0001
2	data28_pp_9p4GeV.280308123.RAW.t0001
3	data28_pp_9p4GeV.280308042.RAW.t0001

Таблица 2: datasets

Для создания и управления базами данных Event Index используется PostgreSQL – мощная и расширяемая объектно-реляционная система управления базами данных. Данная СУБД имеет ряд преимуществ:

- Поддержка сложных SQL-запросов, триггеров, хранимых процедур и других расширенных функций обеспечивает гибкость и высокий уровень контроля над данными.
- Поддержка множества расширений и дополнений
- Эффективное управление ресурсами и оптимизация запросов, что обеспечивает высокую производительность при работе с большими объёмами данных [13].

3.1.2. Реализация чтения и записи данных о событиях

После генерации событий запускается следующий этап. Для процессов чтения и записи данных используется Python-скрипт, который соединяется с PostgreSQL с помощью специального модуля для их взаимодействия. После подключения к базе данных, скрипт начинает чтение JSON-файлов, которые хранят в себе сгенерированные наборы данных. Затем, с помощью SQL-запросов, прочитанные данные, записываются в таблицы events и datasets.

3.2. Оптимизация скорости процессов чтения и записи данных

Использование в первоначальной версии интерфейса процедура PUT (полная или частичная замена текущих данных на данные, отправленные в запросе) показала недостаточную скорость при тестировании даже на относительно небольших массивах данных. При этом перед нами стоит задача по созданию системы, способной справиться с потоком в десятки тысяч событий в секунду. Исходя из этого, были исследованы и протестированы различные методы оптимизации скорости процессов чтения и записи данных в таблицы.

3.2.1. Тестирование модулей

Одним из методов оптимизации стало тестирование различных модулей для взаимодействия Python-скрипта с PostgreSQL для чтения и записи данных о событиях. Целью тестирования являлось определение модуля с наилучшим результатом выполнения задачи для последующей оптимизации. Были протестированы следующие семь модулей: asyncpg, aiopg, pg8000, psycopg2, PyGreSQL, py-postgresql, SQLAlchemy (используемый с psycopg2, который взят за модуль нижнего уровня).

Модули asyncpg, aiopg, SQLAlchemy являются асинхронными модулями, которые позволяют писать код, обрабатывающий множество параллельных задач и операций ввода-вывода.

1. Asyncpg обладает высокой производительностью, благодаря основе, написанной на Rust, и оптимизации для скорости, что делает его одним из самых быстрых модулей для PostgreSQL. Полная асинхронность позволяет эффективно использовать ресурсы и обрабатывать множество запросов параллельно. Кроме того, поддержка современных функций PostgreSQL, таких как работа с JSON и асинхронные транзакции, обеспечивает гибкость и мощность при работе с данными.

2. Aiopg хорошо интегрирован с asyncio, что делает его естественным выбором для асинхронных приложений на Python. Этот модуль представляет собой обёртку над популярным синхронным драйвером psycopg2. Простота использования обеспечивает простой и понятный интерфейс для работы с PostgreSQL, делая переход от синхронного к асинхронному программированию более лёгким. Кроме того, поддержка SQLAlchemy позволяет использовать aiopg с ORM и взаимодействовать с базой данных в асинхронном режиме.

3. SQLAlchemy предоставляет высокоуровневую абстракцию для работы с базами данных через ORM, упрощая манипуляции с данными. Поддержка различных систем управления базами данных, таких как PostgreSQL, MySQL и SQLite, делает модуль универсальным инструментом для различных проектов.

Богатый функционал SQLAlchemy включает широкие возможности для построения сложных запросов, транзакций и связей между таблицами.

Модули psycopg2, pg8000, PyGreSQL, py-postgresql являются синхронными модулями, которые выполняют операции последовательно и блокируют выполнение программы до завершения каждой операции.

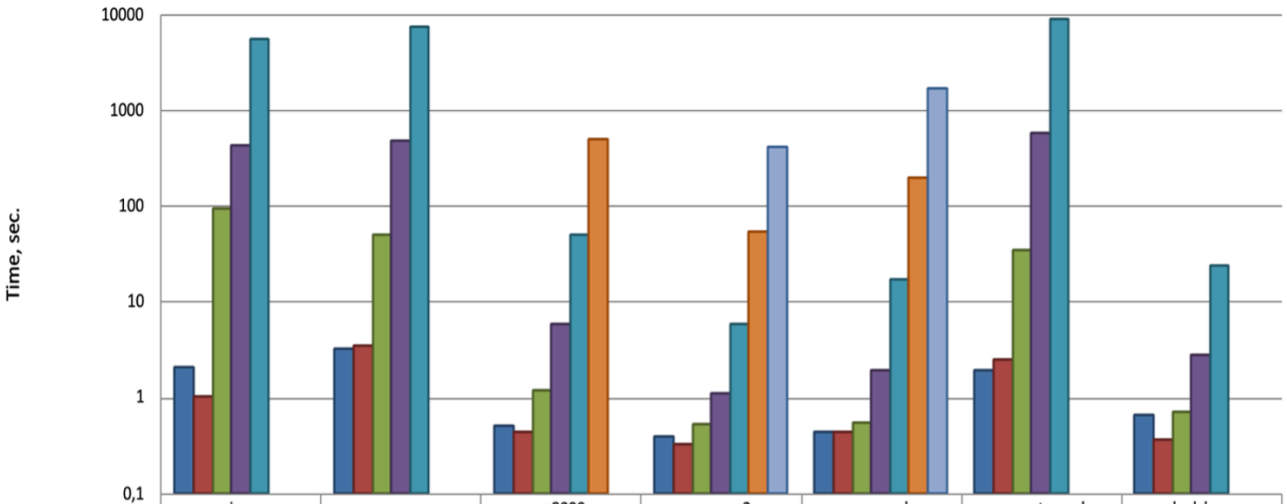
1. Psycopg2 является самым популярным и широко используемым синхронным модулем для PostgreSQL в Python, известный своей стабильностью и производительностью. Он поддерживает полный набор возможностей PostgreSQL, включая выполнение сложных запросов и управление транзакциями. Дополнительно, psycopg2 обладает хорошей документацией и большим сообществом пользователей, что упрощает его использование и интеграцию в проекты.

2. Pg8000, полностью реализованный на Python модуль, который поддерживает все основные функции PostgreSQL, представляя надёжный и гибкий инструмент для работы с базой данных. Благодаря отсутствию внешних зависимостей, он легко устанавливается и используется, что делает его удобным для разработки в различных средах, включая виртуальные окружения и облачные сервисы.

3. PyGreSQL предоставляет как низкоуровневый доступ к базе данных через модуль pg, так и высокоуровневый через интерфейс DB-API, обеспечивая гибкость для различных сценариев использования. Он известен своей стабильностью и богатым набором функций, поддерживающих полноценное взаимодействие с PostgreSQL.

4. Py-postgresql – модуль для PostgreSQL, разработанный с акцентом на производительность и расширяемость. Модуль предоставляет высокоуровневые API для выполнения запросов и управления соединениями. Все операции выполняются последовательно (синхронно), что делает py-postgresql простым в использовании для случаев, когда асинхронность не является критически важной.

Каждый модуль был протестирован с несколькими версиями скрипта. В первоначальной версии кода сгенерированные события записывались в базу данных одно за другим посредством запросов INSERT (запрос на вставку). Результаты тестирования такого python-скрипта приведены (Рис. 9).

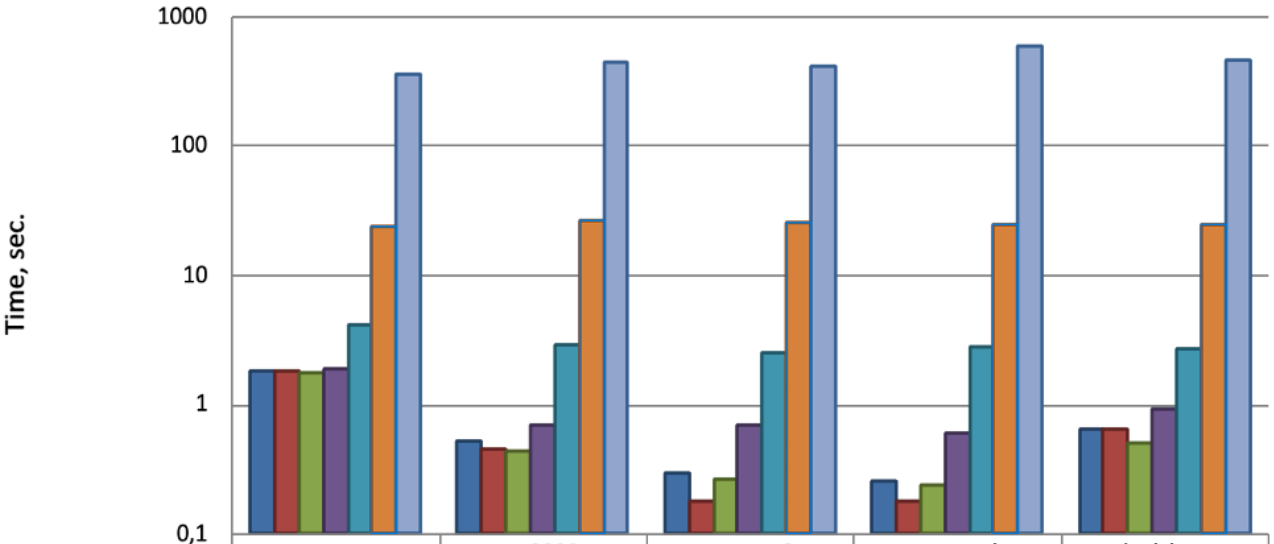


	aiopg	asyncpg	pg8000	psycopg2	pygresql	py-postgresql	sql_alchemy
10 events	2,05	3,21	0,50	0,40	0,44	1,93	0,66
100 events	1,02	3,45	0,45	0,33	0,44	2,47	0,36
1 000 events	93,47	49,84	1,17	0,52	0,55	34,84	0,70
10 000 events	432,35	470,17	5,86	1,11	1,89	581,96	2,77
100 000 events	5484,15	7296,31	50,53	5,87	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 9: Результаты тестирования модулей при использовании запроса INSERT (по горизонтали: тестируемые модули; по вертикали: логарифмическая шкала времени, в секундах)

Чтение и запись небольшого количества событий выполнялось достаточно быстро для всех модулей: например, 10 событий, в среднем, записывались за 1,3 секунды. С увеличением объёма данных скорость значительно замедлилась. Так, на 10 тысяч событий, в среднем, выполнялись за 3 минуты 33 секунд. Начиная со 100 тысяч событий и выше, процесс записи у всех модулей занимал в среднем больше двух с половиной часов. Такая скорость записи данных не подходила для поставленной задачи, учитывая, что ожидается получение примерно 30 миллиардов событий в год.

Тестирование показало, что необходима дальнейшая оптимизация первоначальной версии кода. Исходя из этого, вторая версия была доработана за счёт замена запроса INSERT на COPY (запрос на копирование). Данный отлично подходит для применения метода «массовой загрузки» («bulk loading») данных, при котором события записываются в таблицы большими блоками, что значительно ускоряет выполнение кода.

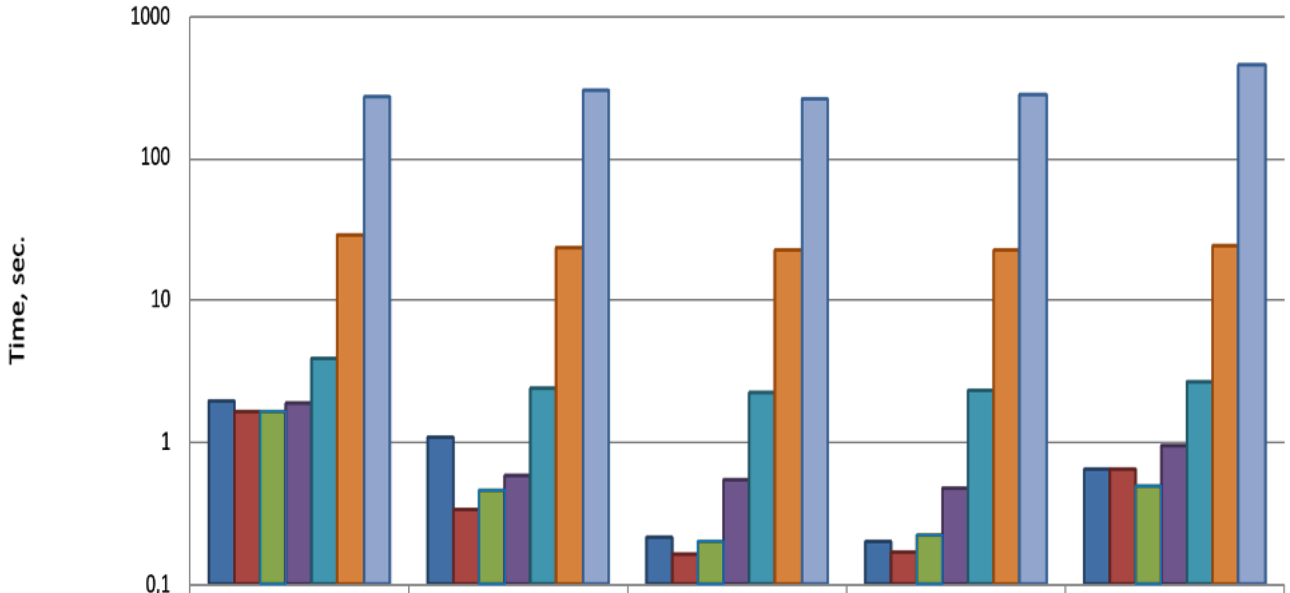


	asyncpg	pg8000	psycopg2	pygresql	sql_alchemy
10 events	1,79	0,52	0,29	0,25	0,64
100 events	1,79	0,44	0,18	0,18	0,65
1 000 events	1,74	0,44	0,27	0,24	0,49
10 000 events	1,89	0,70	0,69	0,61	0,93
100 000 events	4,08	2,85	2,52	2,73	2,69
1 000 000 events	23,44	25,87	24,92	24,13	24,33
10 000 000 events	353,36	441,02	412,77	574,04	449,64

Рисунок 10: Результаты тестирования модулей при использовании метода «массовой загрузки» и запроса COPY (по горизонтали: тестируемые модули; по вертикали: логарифмическая шкала времени, в секундах)

По данным тестирования можно сделать вывод, что использование запроса на копирование ускорило выполнение процессов чтения и записи данных в таблицу примерно в 2 раза. Однако, модули py-postgresql и aiopg не поддерживают использование «массовой загрузки» через запрос COPY. Результаты тестирования остальные пяти модулей приведены (Рис. 10). Так, для работы с небольшими массивами данных (от 10 до 100 тысяч событий)

наиболее эффективно использование модуля PyGreSQL, который, к примеру, записывает 100 тысяч событий приблизительно за 3 секунды. На больших количествах данных лучше всего использовать модуль asyncpg, записывающий 10 миллионов событий за 5 минуты 53 секунды.

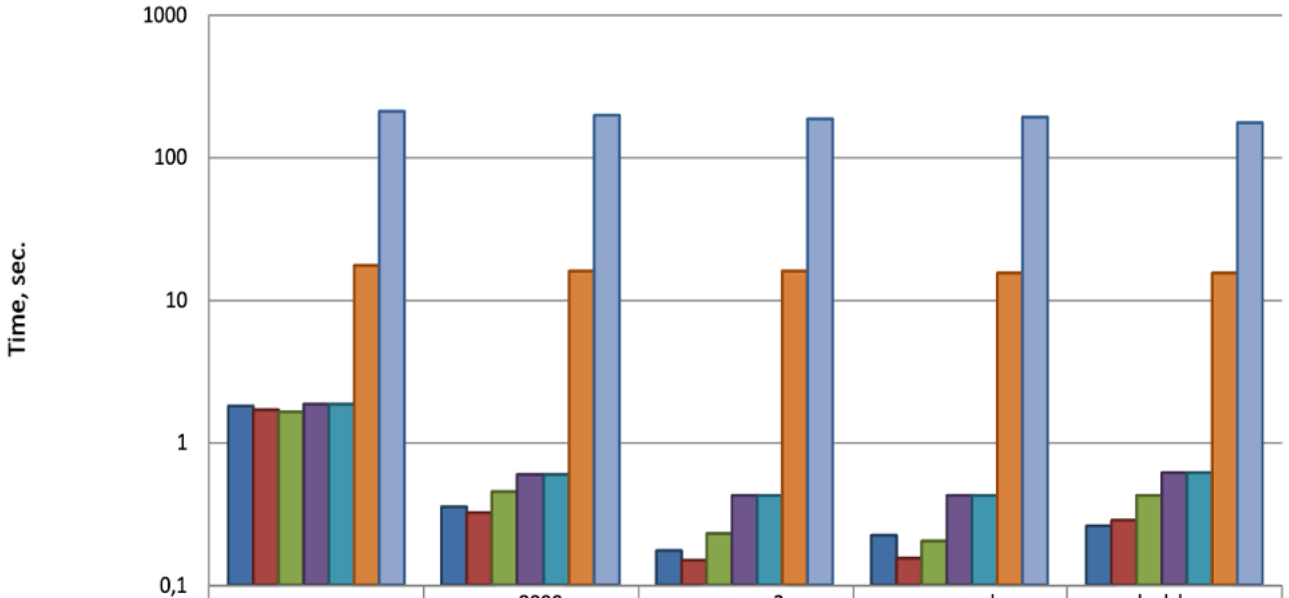


	asyncpg	pg8000	psycopg2	pygresql	sql_alchemy
10 events	1,96	1,10	0,21	0,20	0,64
100 events	1,64	0,34	0,16	0,17	0,65
1 000 events	1,66	0,45	0,20	0,22	0,49
10 000 events	1,86	0,58	0,55	0,48	0,93
100 000 events	3,84	2,42	2,26	2,29	2,69
1 000 000 events	28,74	23,22	22,82	22,83	24,01
10 000 000 events	266,47	298,59	262,14	276,97	449,64

Рисунок 11: Результаты тестирования модулей при использовании временных файлов для записи данных в таблицы (по горизонтали: тестируемые модули; по вертикали: логарифмическая шкала времени, в секундах)

Третьим изменением для оптимизации скорости записи данных являлось использование временных файлов (файлы, создаваемые для хранения данных, которые нужны для выполнения текущих процессов). Для начала эти файлы были включены только в процесс записи событий в таблицы. Данные генерировались в JSON, после они перезаписывались во временные файлы, откуда передавались в таблицы. Результаты тестирования показали, что с такой версией кода наиболее эффективным для большого количества данных является

модуль psycopg2, который записывает 10 миллионов событий за 4 минуты 33 секунды (Рис. 11). На маленьких массивах модуль PyGreSQL по-прежнему работает лучше, записывая, к примеру, 100 тысяч событий за 2,3 секунды.



	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 12: Результаты тестирования модулей при использовании временных файлов для генерации данных и их записи в таблицы (по горизонтали: тестируемые модули; по вертикали: логарифмическая шкала времени, в секундах)

Заключительным изменением стала полная замена файлов формата JSON на временные файлы. В этой версии кода данный файлы использовались не только для записи данных в таблицу, но и для генерации наборов этих данных в начале. Данное изменение оптимизировало скорость выполнение программы, что вытекает из результатов тестирования модулей (Рис. 12). На небольшом количестве событий лучше работает модули psycopg2 и PyGreSQL, которые записывают 100 тысяч событий приблизительно за 0,43 секунды. Однако, на

больших наборах данных самым эффективным является модуль SQLAlchemy, записывающий 10 миллионов событий за 2 минуты 54 секунды. На этом тестирование модулей на данный момент завершилось.

3.2.2. Конфигурации PostgreSQL

Следующим методом оптимизации стало изменение различных параметров конфигурации PostgreSQL [16]. Эти параметры позволяют настраивать поведение сервера базы данных в соответствии с поставленной задачей. Их можно задавать в файле ‘postgresql.conf’ через SQL-команды. Была проведена работа с такими параметрами, как:

1. ‘max_wal_size’: определяет верхний предел объёма дискового пространства, которое может быть занято файлами WAL (журналы предзаписи или Write-Ahead Logging – метод журналирования, при котором все изменения сначала записываются в журнал на диске перед внесением их непосредственно в базу данных) между контрольными точками (checkpoints – важные события в жизненном цикле базы данных, при которых все изменённые данные, находящиеся в памяти, сбрасываются на диск);
2. ‘min_wal_size’: задаёт минимальный объём дискового пространства, выделяемого для хранения журналов предзаписи.
3. ‘checkpoint_timeout’: устанавливает максимальное значение времени, которое может пройти между двумя контрольными точками.
4. ‘effective_io_concurrency’: определяет количество операций ввода-вывода (I/O) для сбора данных с диска, которые могут быть одновременно запущены в рамках одного запроса.
5. ‘synchronous_commit’: контролирует как сервер обрабатывает подтверждение транзакций в отношении синхронной репликации. По умолчанию, когда этот параметр установлен в ‘on’, PostgreSQL ждёт, пока запись о транзакции будет записана в журнал предзаписи и подтверждена на

всех синхронных репликах (если такие существуют), прежде чем сообщить пользователю об успешном завершении транзакции.

6. ‘max_wal_senders’: задаёт максимальное количество отправок журналов предзаписи на сервер. Эти процессы необходимы для поддержки потоковой репликации, при которой WAL передаются от основного сервера к репликам.

7. ‘wal_level’: устанавливает объём и тип информации, записываемой в журнал предзаписи. Он напрямую влияет на возможности репликации и резервного копирования в PostgreSQL.

8. ‘maintenance_work_mem’: определяет максимальный объём памяти, выделяемой для выполнения задач обслуживания базы данных. К этим задачам относятся выполнения операций создания индексов, восстановление базы данных из резервной копии и добавление внешних ключей.

Изменяя значения этих параметров, были получены различные результаты, как положительные (например, изменение параметра ‘maintenance_work_mem’ может ускорить чтение и запись 10 миллионов событий на 1 минуту 37 секунд), так и отрицательные (например, изменение параметра ‘effective_io_concurrency’ может снизить скорость чтения и записи данных для 10 миллионов событий на 54 секунды). Однако, нужно учитывать, что эти изменения могут приводить не только к увеличению производительности, но и к снижению надёжности (потере данных). Дальнейшие настройки конфигураций PostgreSQL целесообразнее проводить при работе с высокопроизводительным оборудованием.

3.2.3. Профилирование

Для более эффективной оптимизации скорости выполнения процессов чтения и записи данных в таблицы была проведена процедура профилирования [17]. Она необходима для анализа производительности программы или системы с целью выявления узких мест и оптимизации кода.

Профилирование включает в себя сбор данных о том, как программа использует системные ресурсы. Это может включать время выполнения отдельных функций, использование памяти и другие метрики производительности.

Анализ собранных данных помогает выявить наиболее узкие места в программе – те части кода, которые наиболее сильно влияют на производительность. Узкие места могут быть связаны с неэффективными алгоритмами, неправильным использованием ресурсов или другими факторами. На основании данных профилирования можно вносить изменения в код для оптимизации производительности.

В дополнение к измерению времени выполнения кода, профилирование может также охватывать анализ использования памяти. Это помогает выявить утечки памяти и оптимизировать её использование. Однако при профилировании необходимо учитывать внешние факторы, такие как нагрузка на сервер, объём данных, доступность сети и т.д. Эти факторы могут сильно влиять на результаты профилирования.

Python имеет встроенный модуль ‘cProfile’, который можно использовать как напрямую из кода, так и из командной строки для профилирования скриптов. Этот модуль позволяет собирать следующие данные:

1. ‘ncalls’ – количество вызовов функции в ходе выполнения профилируемого кода;
2. ‘tottime’ – общее время выполнения функции без учёта времени, затраченного на вызов других функций, которые вложенные в данную;
3. ‘percall’ – среднее время выполнения одного вызова функции (первое вхождение, вычисляется как ‘tottime’/‘ncalls’);
4. ‘cumtime’ – суммарное время выполнения функции, включая время, затраченное на выполнение всех вложенных в неё функций;
5. ‘percall’ – среднее время выполнения одного вызова функции, с учётом вложенных в неё функций (второе вхождение, вычисляется как ‘cumtime’/‘ncalls’);

6. ‘filename:lineno(function)’ – имя файла и номер строки, где определена функция, а также имя функции.

Для каждого из пяти протестированных модулей взаимодействия Python с PostgreSQL была применена процедура профилирования с использованием ‘cProfile’. Исходя из результатов тестирования asyncpg (Таблица 3), SQLAlchemy (Таблица 4), pg8000 (Таблица 5), PyGreSQL (Таблица 6), psycopg2 (Таблица 7), можно сделать вывод, что в основном ресурсы затрачиваются на выполнения функций, запускающих процессы чтения и записи событий в таблицы базы данных.

ASYNCPG					
ncalls	**tottime**	**percall**	**cumtime**	**percall**	**filename:lineno(function)**
10004275	4.208	0.000	4.208	0.000	{method 'split' of 'str' object}
10001558	1.413	0.000	1.413	0.000	{method 'strip' of 'str' object}
196	4.764	0.024	4.983	0.025	connection.py:982(copy_records_to_table)
9875	0.223	0.000	0.244	0.000	base_events.py:770(_call_soon)
4261920	0.810	0.000	0.810	0.000	hmac.py:144(_current)
1041	0.718	0.001	0.726	0.001	selectors.py:366(unregister)
1	0.173	0.173	0.173	0.173	store_events.py:145(<listcomp>)
11160	0.252	0.000	0.252	0.000	{built-in method __new__ of type object at 0x7fb1a51377c0}
123097	0.135	0.000	0.135	0.000	{built-in method _codecs.utf_8_decode}
2132004	0.627	0.000	0.627	0.000	{built-in method _hashlib.get_fips_mode}
4330737	0.682	0.000	0.682	0.000	{built-in method builtins.isinstance}
10024133/10023850	0.960	0.000	0.960	0.000	{built-in method builtins.len}
520	0.164	0.000	0.164	0.000	{built-in method 'connect' of '_socket.socket' objects}
1125	0.104	0.000	0.104	0.000	{built-in method 'extend' of 'list' objects}
114	0.187	0.002	0.187	0.002	{method 'read' of _io.BufferedReader' objects}
2600	0.006	0.000	0.433	0.000	connect_utils.py:796(_connect_addr)
196	4.764	0.024	4.983	0.025	connection.py:982(copy_records_to_table)
4261920	2.409	0.000	9.053	0.000	hmac.py:156(digest)
2132000	1.284	0.000	9.543	0.000	hmac.py:172(new)
2132000	2.309	0.000	8.259	0.000	hmac.py:39(__init__)
2132000	1.417	0.000	4.658	0.000	hmac.py:68(_init_hmac)
769	**19.181**	**0.025**	**34.939**	**0.045**	**store_events.py:34(main)**
2132000	3.240	0.000	3.240	0.000	{built-in method _hashlib.hmac_new}
10035314	1.135	0.000	1.135	0.000	{method 'append' of list' objects}
1041	0.718	0.001	0.726	0.001	selectors.py:366(unregister)
1	0.173	0.173	0.173	0.173	store_events.py:145(<listcomp>)
11160	0.252	0.000	0.252	0.000	{built-in method __new__ of type object at 0x7fb1a51377c0}
123097	0.135	0.000	0.135	0.000	{built-in method _codecs.utf_8_decode}
4330737	0.682	0.000	0.682	0.000	{built-in method builtins.isinstance}
520	0.164	0.000	0.164	0.000	{method 'connect' of _socket.socket' objects}
1972	**11.603**	**0.006**	**30.343**	**0.015**	**{method data_received' of 'asyncpg.protocol.protocol.BaseProtocol' objects}**
4261920	5.835	0.000	5.835	0.000	{method 'digest' of _hashlib.HMAC' objects}
3937	**148.143**	**0.038**	**148.143**	**0.038**	**{method 'poll' of 'select.epoll' objects}**
1000	2.053	0.002	2.273	0.002	{method 'readlines' of '_io._IOBase' objects}
13457	1.152	0.000	68.307	0.005	{method 'run' of 'Context' objects}

Таблица 3: Результаты профилирования для модуля asyncpg

SQL Alchemy					
ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
1	0.164	0.164	0.164	0.164	store_events.py:142(<listcomp>)
1	18.948	18.948	182.055	182.055	store_events.py:31(main)
123724	0.130	0.000	0.130	0.000	{built-in method _codecs.utf_8_decode}
50	9.047	0.181	152.693	3.054	eipsyco.py:994(store_raw_event)
10011253/10010694	0.971	0.000	0.971	0.000	{built-in method builtins.len}
53	0.865	0.016	0.870	0.016	{built-in method psycopg2._psycopg._connect}
10013250	1.226	0.000	1.226	0.000	{method 'append' of 'list' objects}
52	7.219	0.139	7.219	0.139	{method'commit' of 'psycopg2.extensions.connection' objects}
51	119.633	2.346	119.633	2.346	{method 'copy_expert' of 'psycopg2.extensions.cursor' objects}
10000022	1.726	0.000	1.726	0.000	{method 'encode' of 'str' objects}
10005264	11.960	0.000	11.960	0.000	{method 'join' of 'str' objects}
1000	1.695	0.002	1.906	0.002	{method 'readlines' of '_io._IOBase' objects}
6	1.204	0.201	183.259	30.543	{method 'run' of 'Context' objects}
10000901	4.284	0.000	4.284	0.000	{method 'split' of 'str' objects}
10004393	1.578	0.000	1.578	0.000	{method 'strip' of 'str' objects}
10000000	1.366	0.000	1.366	0.000	{method 'write' of '_io.BytesIO' objects}

Таблица 4: Результаты профилирования для модуля SQLAlchemy

PG8000					
ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
52	0.204	0.004	6.192	0.119	core.py:252(__init__)
469062	0.253	0.000	0.590	0.000	core.py:165(_write)
156151	0.093	0.000	34.443	0.000	core.py:142(_flush)
156151	0.243	0.000	34.079	0.000	socket.py:713(write)
1	1.235	1.235	237.268	237.268	store_events.py:1(<module>)
1	0.176	0.176	0.176	0.176	store_events.py:143(<listcomp>)
1	23.752	23.752	235.362	235.362	store_events.py:32(main)
52	0.180	0.003	5.621	0.108	utils.py:13(hi)
212992	1.463	0.000	3.226	0.000	utils.py:21(xor)
7028736	1.764	0.000	1.764	0.000	utils.py:22(<genexpr>)
50	7.115	0.142	169.914	3.398	eipsyco.py:934(store_raw_event)
60000000	15.884	0.000	15.884	0.000	eipsyco.py:952(<genexpr>)
213200	0.166	0.000	0.610	0.000	hmac.py:156(digest)
213200	0.161	0.000	1.262	0.000	hmac.py:172(new)
213200	0.309	0.000	1.101	0.000	hmac.py:39(__init__)
213200	0.187	0.000	0.616	0.000	hmac.py:68(_init_hmac)
123099	0.214	0.000	0.214	0.000	{built-in method _codecs.utf_8_decode}
213200	0.428	0.000	0.428	0.000	{built-in method _hashlib.hmac_new}
51	0.571	0.011	39.410	0.773	core.py:457(handle_COPY_IN_RESPONSE)
10172552/10172110	1.170	0.000	1.170	0.000	{built-in method builtins.len}
156151	0.094	0.000	0.155	0.000	{method '_checkWritable' of '_io._IOBase'
10018444	1.381	0.000	1.381	0.000	{method 'append' of 'list' objects}
213200	0.385	0.000	0.385	0.000	{method 'digest' of _hashlib.HMAC' objects}
156443	0.097	0.000	0.097	0.000	{method 'encode' of 'str' objects}
1147	0.122	0.000	0.122	0.000	{method 'extend' of 'list' objects}
156151	0.270	0.000	34.349	0.000	{method flush' of _io.BufferedRWPair' objects}
10003356	11.140	0.000	27.029	0.000	{method 'join' of 'str' object}
155597	3.128	0.000	3.128	0.000	{method 'read' of '_io.StringIO' object}
1015	31.217	0.031	a	0.031	{method 'readlines' of '_io._IOBase' object}
783	88.817	0.113	88.817	0.113	{method 'recv_into' of '_socket.socket' objects}
156151	33.627	0.000	33.627	0.000	{method 'send' of '_socket.socket' objects}
10000390	5.163	0.000	5.163	0.000	{method 'split' of 'str' objects}
10001038	1.710	0.000	1.710	0.000	{method 'strip' of 'str' objects}
469062	0.337	0.000	0.337	0.000	{method 'write' of '_io.BufferedRWPair' objects}
10000024	1.890	0.000	1.890	0.000	{method 'write' of _io.StringIO' objects}
213200	0.346	0.000	2.217	0.000	utils.py:5(hmac)
155582	0.374	0.000	3.783	0.000	core.py:478(ri)
123099	0.150	0.000	0.364	0.000	codecs.py:319(decode)

Таблица 5: Результаты профилирования для модуля pg8000

PyGreSQL					
ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
10000000	1.965	0.000	1.965	0.000	{method 'write' of '_io.StringIO' objects}
10000009	5.226	0.000	5.226	0.000	{method 'split' of 'str' objects}
10001000	1.781	0.000	1.781	0.000	{method 'strip' of 'str' objects}
1000	31.024	0.031	31.377	0.031	{method 'readlines' of '_io._IOBase' objects}
10000985	11.612	0.000	28.323	0.000	{method 'join' of 'str' objects}
1030	0.115	0.000	0.115	0.000	{method 'extend' of 'list' objects}
51	121.523	2.383	121.524	2.383	{method 'copy_from' of 'psycopg2.extensions.cursor' objects}
51	5.162	0.101	5.162	0.101	{method 'commit' of 'psycopg2.extensions.connection' objects}
10001472	1.451	0.000	1.451	0.000	{method 'append' of 'list' objects}
10001702/10001624	1.175	0.000	1.175	0.000	{built-in method builtins.len}
1	1.375	1.375	232.130	232.130	store_events.py:1(<module>)
1	0.254	0.254	0.254	0.254	store_events.py:140(<listcomp>)
1	23.892	23.892	230.611	230.611	store_events.py:29(main)
50	7.423	0.148	165.193	3.304	eipsyco.py:914(store_raw_event)
60000000	16.711	0.000	16.711	0.000	eipsyco.py:933(<genexpr>)
123097	0.150	0.000	0.353	0.000	codecs.py:319(decode)

Таблица 6: Результаты профилирования для модуля asyncpg

PSYCOPG2					
ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
10000000	1.944	0.000	1.944	0.000	{method 'write' of '_io.StringIO' objects}
50	7.406	0.148	170.818	3.416	eipsyco.py:796(store_raw_event)
60000000	16.389	0.000	16.389	0.000	eipsyco.py:818(<genexpr>)
1	1.253	1.253	201.794	201.794	store_events.py:1(<module>)
1	0.168	0.168	0.168	0.168	store_events.py:143(<listcomp>)
1	19.559	19.559	200.397	200.397	store_events.py:32(main)
123152	0.140	0.000	0.140	0.000	{built-in method _codecs.utf_8_decode}
10003798/10003573	0.962	0.000	0.962	0.000	{built-in method builtins.len}
52	0.799	0.015	0.802	0.015	{built-in method psycopg2._psycopg._connect}
10003647	1.143	0.000	1.143	0.000	{method 'append' of 'list' objects}
51	6.051	0.119	6.051	0.119	{method 'commit' of 'psycopg2.extensions.connection' objects}
51	126.753	2.485	126.753	2.485	{method copy_expert' psycopg2.extensions.cursor' objects}
10002032	11.485	0.000	27.874	0.000	{method 'join' of 'str' objects}
1000	1.755	0.002	1.983	0.002	{method 'readlines' of '_io._IOBase' objects}
10000016	4.116	0.000	4.116	0.000	{method 'split' of 'str' objects}
10001034	1.450	0.000	1.450	0.000	{method 'strip' of 'str' objects}

Таблица 7: Результаты профилирования для модуля psycopg2

3.2.4. Дальнейшие задачи

Дальнейшая разработка подразумевает решение следующих задач:

1. Интеграция модуля загрузки данных с системой управления потоками данных.

2. Адаптация API чтения физических данных для их индексации.

3. Создание соответствующего программного модуля для использования в системе распределённых вычислений panDA.

PanDA (Production and Distributed Analysis) – это программное обеспечение, разработанное для управления и распределённого анализа данных в экспериментах по физике частиц, таких как ATLAS (A Toroidal LHC ApparatuS) и CMS (Compact Muon Solenoid) на Большом адронном коллайдере в ЦЕРНе. Будет применятся также в эксперименте SPD на NICA.

4. Оптимизация схема базы данных для уменьшения размера и повышения производительности;

5. Оптимизация загрузки данных в рамках полного цикла на высокопроизводительном оборудовании.

Вывод по главе 3

Были протестированы различные способы для оптимизации скорости выполнения программы такие, как:

- использование метода массовой загрузки;
- тестирование синхронных и асинхронных модулей взаимодействия python с PostgreSQL;
- замена SQL-запросов: с INSERT на COPY;
- использование временных файлов для генерации данных и записи их в таблицы;
- применение профилирования с помощью модуля ‘cProfile’.

На первом этапе оптимизации на чтение и запись данных у тестируемых модулей в среднем занимала около 2 часов для 100 тысяч событий. На данный момент для выполнения тех же процессов уходит около 3 минут для 10 миллионов событий.

По результатам нескольких этапов тестирования можно выделить модуль SQLAlchemy, который записал 10 миллионов событий за 2,5 минуты. Благодаря профилированию, можно удостовериться, что ресурсы затрачиваются в основном на процессы чтения данных и записи их в таблицы PostgreSQL. Это указывает на возможность улучшения производительности при переходе с тестовой системы на виртуальных машинах с минимальными ресурсами на крупные сервера БД с профессиональной конфигурацией и поддержкой и высокопроизводительным оборудованием.

ЗАКЛЮЧЕНИЕ

В рамках данного исследования была проведена всесторонняя работа по изучению различных аспектов физики элементарных частиц, анализа данных и оптимизации процессов обработки информации.

Все поставленные задачи были выполнены. Во-первых, была рассмотрены основы экспериментальной физики частиц, включающая в себя работу ускорителей, таких как коллайдеры и эксперименты с фиксированной мишенью. Были детально описаны общие принципы работы экспериментальных установок, которые играют ключевую роль в регистрации и изучении взаимодействий частиц.

Во-вторых, изучены методы анализа данных в физике частиц и высоких энергий. Основные этапы включают отбор событий, реконструкцию физических объектов и физический анализ, каждый из них важен для точного интерпретирования экспериментальных данных. Также были рассмотрены разрабатываемые системы сбора данных с детекторов установки такие, как DAQ и SPD On-line Filter, обеспечивающие получение информации о событиях с детектора и их отбор, а также каталог событий Event Index, предоставляющий, эффективный доступ к большим объёмам физических данных.

В-третьих, представлена оптимизация процессов загрузки и чтения информации о событиях эксперимента. Были подробно рассмотрены этапы генерации событий, реализации чтения и записи данных, а также методы оптимизации этих процессов. Особое внимание было уделено тестированию модулей взаимодействия Python-скрипта с системой управления базами данных, настройке конфигураций PostgreSQL и процедуре профилирования, что позволит повысить производительность системы Event Index.

Данное исследование демонстрирует важность и сложность анализа данных в физике частиц и высоких энергий. Улучшение методов и систем обработки данных способствует более точному пониманию фундаментальных принципов физики, проверке теоретических моделей и открытию новых

явлений. Разрабатываемые и оптимизируемые методы анализа данных играют ключевую роль в успешном проведении научных исследований и дальнейшем развитии физики элементарных частиц.

СПИСОК ЛИТЕРАТУРЫ

1. Прокошин Ф.В., SPD EventIndex / Ф.В. Прокошин, И.В. Тваури, З.А. Будтуева, Р.З. Гурциев, А.В. Газзаев // Физика элементарных частиц и атомного ядра. – 2024 – Т. 55, № 3 – С. 717-721
2. 10-я Международной конференции “Распределенные вычисления и GRID технологии в науке и образовании”. – Режим доступа: https://indico.jinr.ru/event/3505/
3. Весенняя школа по информационным технологиям ОИЯИ 2024 – Режим доступа: https://indico.jinr.ru/event/4419/overview
4. Ускорители заряженных частиц. – Режим доступа: https://ru.wikipedia.org/wiki/Ускорители_заряженных_частиц
5. Курс лекций «Metascience project NICA: collider». – Режим доступа: https://edu.jinr.ru/courses/course/view.php?id=44
6. Ускорительный комплекс NICA. – Режим доступа: https://nica.jinr.ru/
7. NICA Spin Physics Detector (SPD). – Режим доступа: https://nica.jinr.ru/projects/spd.php
8. Шматов, С.В. / At the frontiers of particle physics. – Режим доступа: https://indico-hlit.jinr.ru/event/394/contributions/2307/attachments/660/1125/16-10_04_Shmatov.pdf
9. Жемчугов, А.С. / Прикладное программное обеспечение для физических экспериментов. – Режим доступа: https://indico-hlit.jinr.ru/event/394/contributions/2335/attachments/667/1132/19-10_01_zhemchugov_itschool2023.pdf
10. Статистические методы анализа данных. – Режим доступа: https://www.vsavm.by/knigi/kniga3/780.html
11. Статья (википедия) «Методы Монте-Карло». – Режим доступа: https://ru.wikipedia.org/wiki/Метод_Монте-Карло
12. Солдатов Е.Ю. / MC генераторы событий. – Режим доступа: https://indico.particle.mephi.ru/event/111/attachments/1285/1898/MC_.pdf

13. Олейник Д.А. / SPD Online Filter. Первичная обработка экспериментальных данных эксперимента SPD. – Режим доступа: https://indico-hlit.jinr.ru/event/394/contributions/2339/attachments/681/1147/19-10_05_JINR_IT_school_SPD_Online_Filter_2023_.pdf

14. Прокошин, Ф.В. / SPD Information system – Режим доступ: https://indico.jinr.ru/event/4419/contributions/25174/attachments/18267/31227/SPD%20Information%20systems.big.pdf

15. СУБД PostgreSQL. – Режим доступа: https://www.postgresql.org/docs/

16. Параметры конфигурации PostgreSQL. – Режим доступа: https://postgrespro.ru/docs/postgrespro/9.5/config-setting

17. Процедура профилирования (для оптимизации). – Режим доступа: https://ru.wikipedia.org/wiki/Профилирование_(информатика)