Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский ядерный университет «МИФИ»

УДК 004.415.2

ОТЧЁТ
О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

СИСТЕМА УПРАВЛЕНИЯ ДАННЫМИ ДЛЯ СПЕЦИАЛИЗИРОВАННОГО ВЫЧИСЛИТЕЛЬНОГО КОМПЛЕКСА SPD ONLINE FILTER

Студент _____ П. А. Коршунова

Научный руководитель _____ Е. Ю. Солдатов

Научный консультант _____ Д. А. Олейник

Москва 2024

Содержание

Введение

Одной из неразрешенных проблем современной физики высоких энергий является «спиновый кризис», который заключается в том, что мы не знаем как спины нуклонов распределены между их составляющими (кварками и глюонами).

Данный кризис был вызван экспериментом EMC [1], в котором пытались определить распределение спина внутри протона. Ожидалось, что весь спин протона несут валентные кварки, однако оказалось, что это не так.

Изучение спиновой структуры нуклона имеет большое значение, так как он отвечает за фундаментальные свойства природы. Но наши знания о его внутренней структуре все еще ограничены, особенно в отношении вклада глюонов. Именно поэтому строится новая установка SPD (являющаяся часть ускорительного комплекса NICA) для всестороннего изучения глюонного состава нуклона [2].

Здесь возникает проблема, связанная с тем, что в результате столкновения пучков на коллайдере (событие) образуется большой поток данных с детектора (около 200 ПБ/год), который надо как-то обрабатывать и хранить. При этом из-за сложности и многообразия изучаемых процессов невозможно задать критерии отбора данных на аппаратном уровне. В связи с этим возникает необходимость разработки специализированной вычислительной системы, которая будет частично реконструировать события, отфильтровывать «скучные» и подготавливать данные для долговременного хранения.

Именно такой системой является вычислительный комплекс SPD Online Filter, который был рассмотрен в данной работе.

1 Организация SPD Online Filter

SPD Online Filter — это высокопроизводительная вычислительная система для высокопропускной обработки данных. Основной целью данной системы является быстрая реконстукция событий и сокращение объема данных для долговременного хранения.

В качестве входных данных система получает файлы «сырых» данных в формате, определяемом электроникой и системой DAQ (Data Acquisition System). Результатом обработки данных является набор реконструированных событий, содержащий также исходную информацию.

1.1 Требование к SPD Online Filter

Данная вычислительная система должна выполнять следующие задачи:

- раскодирование данных, полученных от DAQ;
- выделение событий;
- фильтрация «скучных» событий в соответствии с заданными физическими критериями;
- упорядочивание выходных данных, объединение событий в файлы, а файлы в наборы данных для дальнейшей обработки;
- подготовка данных для системы контроля качества данных.

1.2 Архитектура SPD Online Filter

Для выполнения данных задач предполагается следующая архитектура системы (рис. 1):

- высокоскоростное хранилище входных данных, полученных с помощью системы сбора данных (DAQ);
- буфер для промежуточных данных и данных, подготовленных к передаче в долгосрочное хранилище и будущей обработке;

- комплекс промежуточного программного обеспечения для автоматизации рабочего процесса, в который будут входить следующие компоненты:
  - *Система управления данными* (регистрация новых данных, каталогизация/структуризация, контроль целостности);
  - *Система управления процессами обработки* (формирование и контроль исполнения этапов обработки данных);
  - *Система управления нагрузкой* (реализация этапов обработки, посредством формирования и выполнения необходимого количества задач для обработки набора данных);
    - *Pilot* (агентское приложение, работающее на вычислительном узле и исполняющее задачи, поставляемые от системы управления нагрузкой).

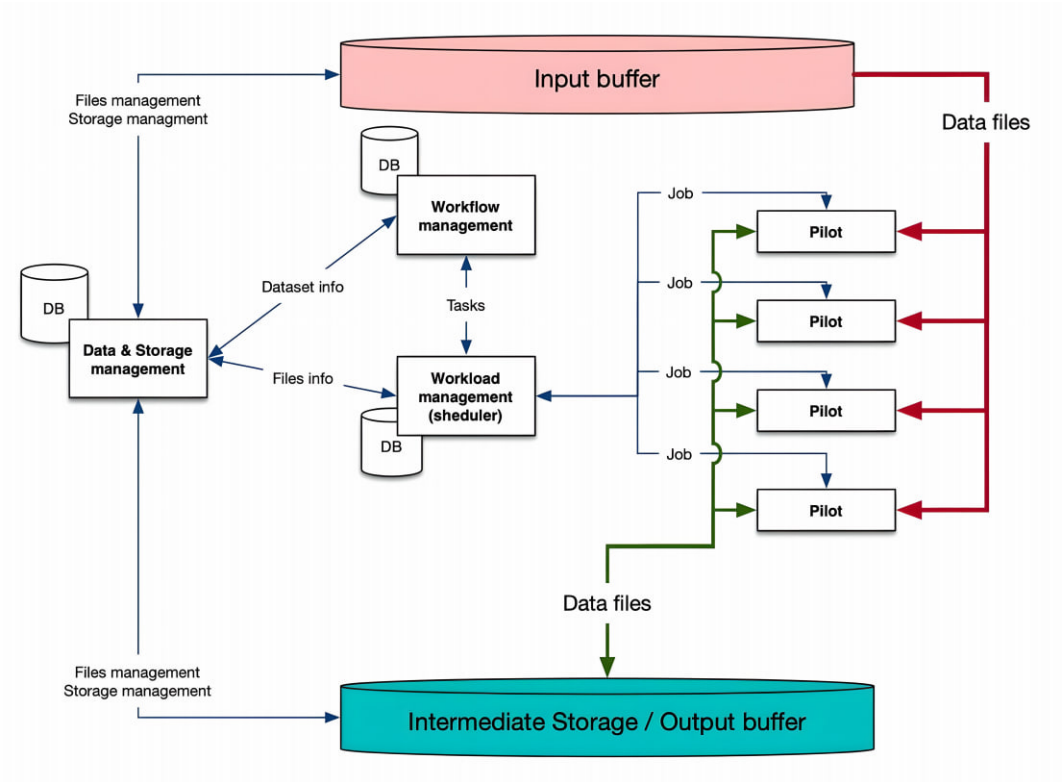

Рисунок 1 – Архитектура SPD Online Filter.

1.3 Поток данных в системе SPD Online Filter

К потоку данных относится то, как они циркулируют в системе. Рассмотрим основной поток данных (рис. 2).

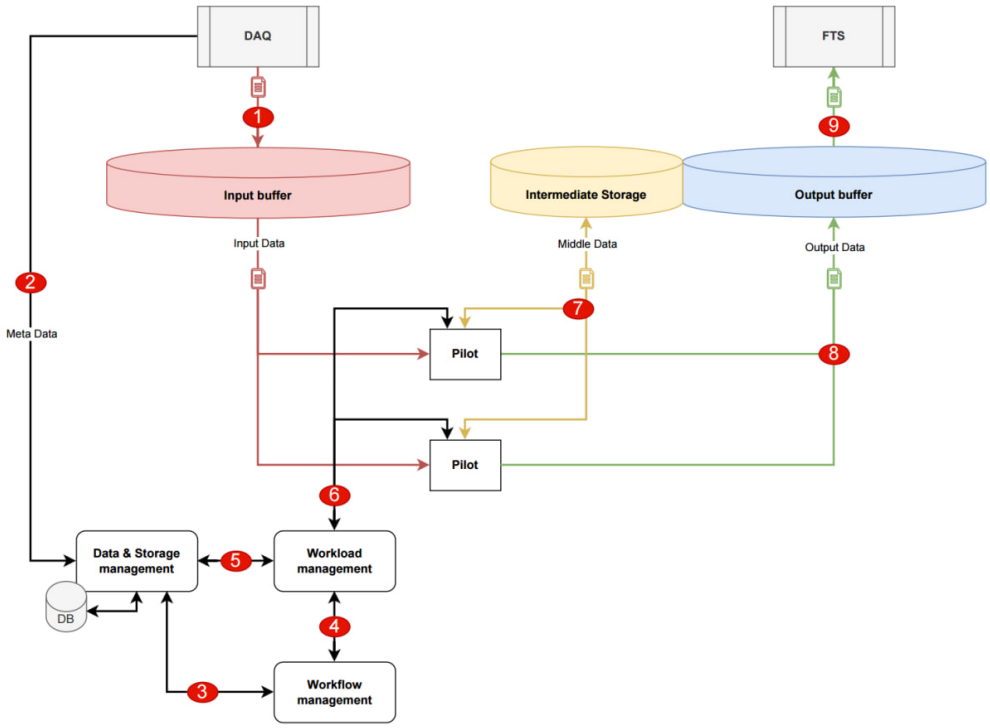

Рисунок 2 – Поток данных в SPD Online Filter.

1-2) DAQ загружает набор однородных файлов на входной буфер, и информация о записанном наборе поступает в систему управления данными для первичной регистрации.

3-4) Система управления процессом обработки запрашивает данные о каком либо наборе данных без детальной информации о каждом файле. После чего эта информация поступает в систему управления нагрузкой в составе описания задания (task).

5) Система управления нагрузкой запрашивает информацию о составе набора (о конкретных файлах входящих в набор). Формирует задачи (jobs).

6-7-8) Пилоты вместе с описанием задач получают ссылки на входные файлы и имена выходных файлов. Пилоты записывают выходные файлы в хранилище и информируют систему управления нагрузкой.

5) Система управления нагрузкой регистрирует выходные файлы в выходном наборе и по окончанию обработки входного набора закрывает вы-

ходной набор.

3) Система управления процессом контролирует, какие промежуточные наборы нужно удалить, а какие передать для долговременного хранения.

9) FTS (File Transfer System) по итогу всего рабочего процесса забирает выходные данные с выходного буфера.

2 Система управления данными

Так как в процессе обработки данных образуется большой объем вторичных данных, то нам необходима специальная система для управления этими данными. Такой системой является система управления данными.

2.1 Требования к системе

Система должна обеспечивать контроль над хранением, организацией, а также целостностью данных.

Для того чтобы система могла узнать о существовании файла в хранилище, необходим интерфейс для регистрации файлов, который включает в себя получение информации о местоположении файла (имя, физический путь, метаданные) и внесение его в каталог со ссылкой на требуемый набор данных.

Из функциональности выше следует то, что информация о файлах и наборах должна храниться в каталоге. Это означает наличие интерфейса к каталогу данных, через который можно размещать информацию о файлах, запрашивать информацию из каталога, удалять информацию в каталоге.

Так как каждый шаг обработки данных оперирует не самим файлом, а набором из них, то система управления данными должна предоставлять возможность управления наборами (dataset-ми). Сюда входят следующие функции: создать dataset, добавить файл в dataset, закрыть dataset, удалить dataset, дать информацию о содержимом dataset (файлах в датасете).

Для корректного функционирования всей системы требуются фоновые сервисы, которые бы осуществляли удаление файлов в хранилищах, контроль целостности файлов и общий контроль использования хранилища.

2.2 Архитектура

Концептуальная архитектура системы управления данными с учетом требуемой функциональности представлена на рисунке 3.

Система состоит из трех независимых друг от друга сервисов [3]:

- **dsm-register**. Сервис, принимающий в асинхронном режиме (через

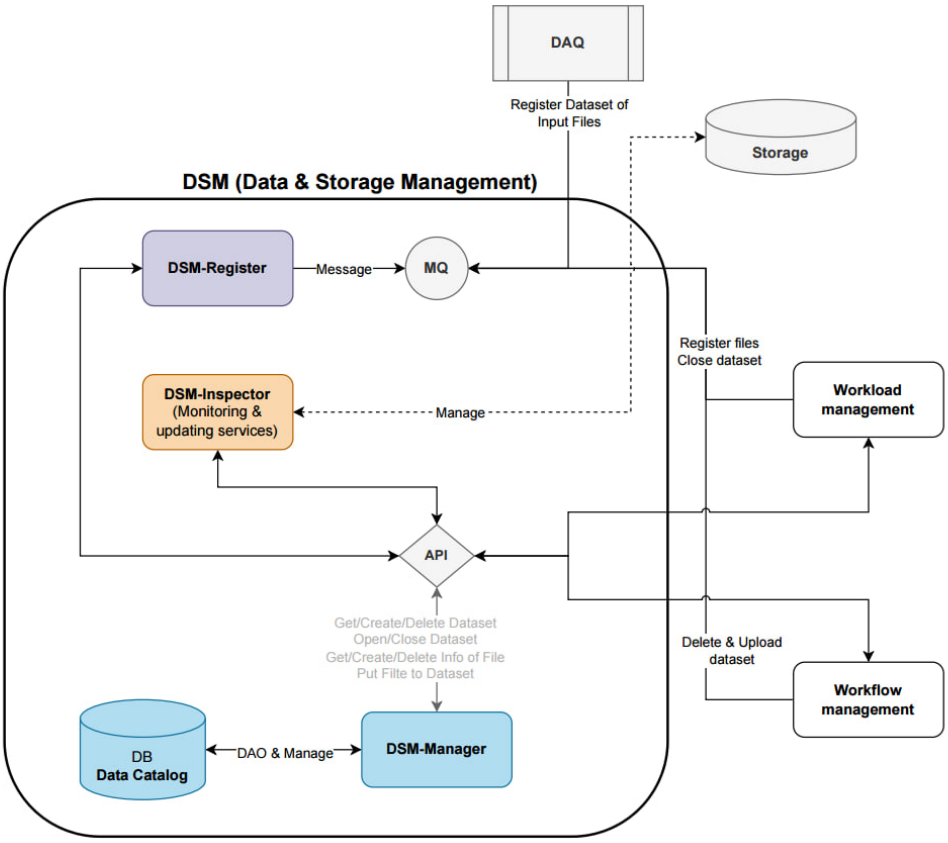

Рисунок 3 – Концептуальная архитектура системы управления данными.

очередь сообщений) заявки на добавление/удаление данных в системе. При обработке заявок сервис вносит изменения в каталог данных через API сервиса dsm-manager;

- **dsm-manager**. Сервис, предоставляющий REST API к каталогу данных (размещение данных в каталоге, обращение к каталогу, изменение данных в каталоге). Он нужен как для внутреннего функционирования системы, так и для внешнего взаимодействия;

- **dsm-inspector**. Набор фоновых сервисов для мониторинга и контроля состояния данных в хранилище (проверка целостности файлов, контроль использования хранилища, удаление файлов на хранилищах).

В тех случаях, где требуется мгновенный ответ от сервиса, используется HTTP протокол обмена данными. Для остальных случаев используется AMQP протокол для асинхронной передачи данных в виде сообщений.

3 Модель данных

Исходя из таких критериев, как качество поддержки, функциональность, широкий круг пользователей, доступность, в качестве СУБД был выбран PostgreSQL 12 [4].

Концептуальная модель данных / ER-диаграмма представлена на рисунке 4. Сюда вошел следующий набор таблиц:

- *DAT_FILE* — каталог файлов, обрабатываемых системой;
- *DAT_DATASET* — каталог наборов файлов, созданных системой;
- *DAT_FILE_HISTORY* и *DAT_DATASET_HISTORY* — архивные таблицы по файлам и наборам;
- *DAT_STORAGE* — информация о хранилищах;
- *DIC_FILE_STATUS* и *DIC_DATASET_STATUS* — справочники, хранящие возможные статусы по файлам и наборам соответственно.

Каталог файлов является ключёвой таблицей в базе данных. В её состав атрибутов вошла та информация, которой достаточно для покрытия следующих целей:

1. Идентификация файла на хранилище (имя файла, путь на хранилище, идентификатор хранилища);
2. Контроль целостности файла (размер файла и его контрольная сумма);
3. Фиксация жизненного цикла файла (статус файла).

Помимо этого предусмотрена возможность принадлежности файла к одному или нескольким наборам. Это осуществляется с помощью ассоциативной таблицы, реализующей отношение «многие ко многим» [5].

Каталог наборов служит для хранения логических группировок файлов, не релевантных к физическому расположению. Каждый набор имеет уникальное наименование, определяемое логикой группировки. Также в

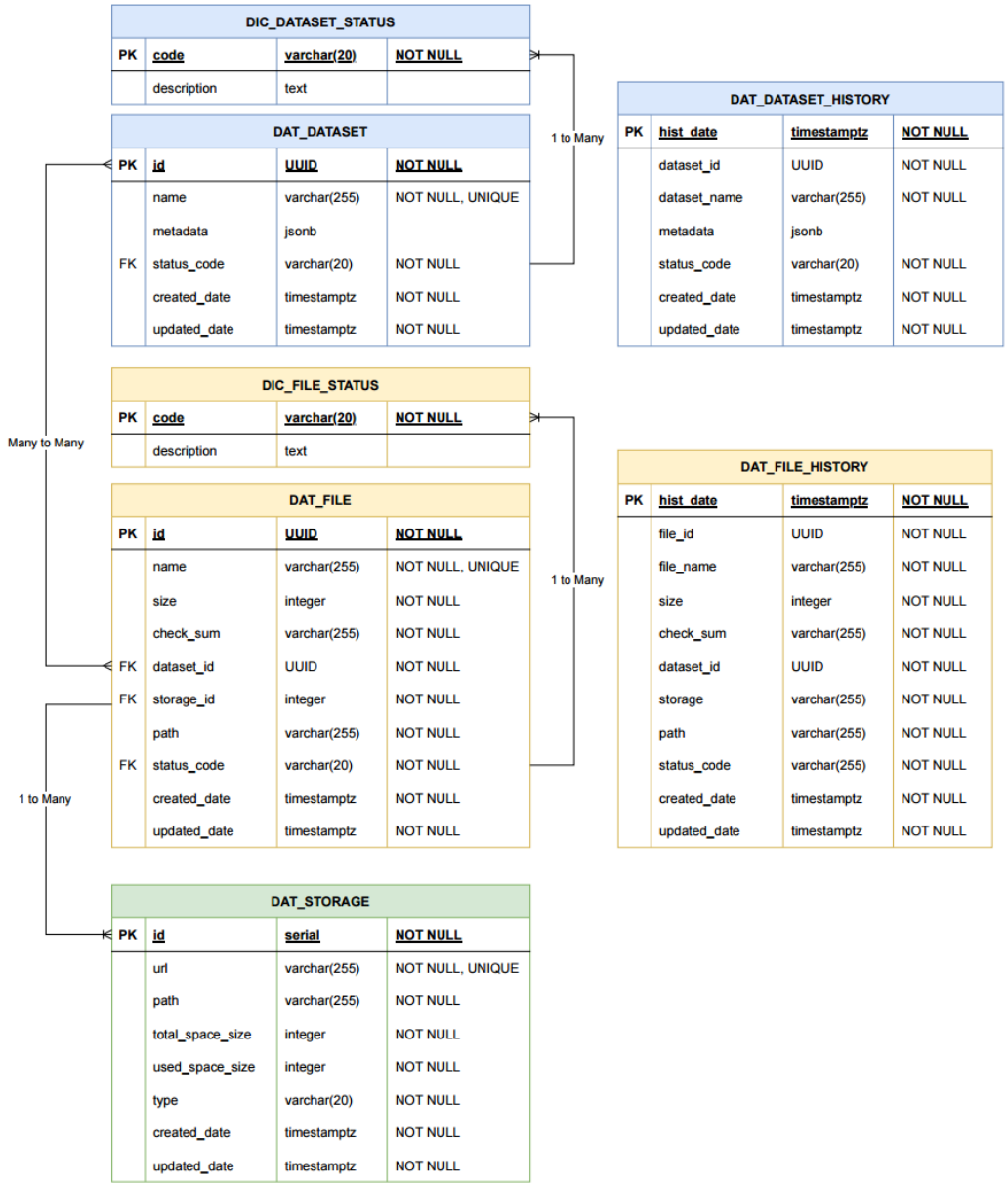

Рисунок 4 – Концептуальная модель данных.

данном каталоге может быть размещена дополнительная неструктурированная мета-информация о наборе в JSON поле (например, в случае входного набора файлов, поступающих с DAQ, это может быть информация о временном срезе данных: номер фрейма, номер рана и т.п.). Набор, как и файл, имеет свой жизненный цикл, поэтому для его фиксации также имеется поле со статусом.

Хранилища помимо адреса могут характеризоваться также дополнительной важной информацией:

1. Тип хранилища (например, входное, промежуточное, выходное);

2. Полный путь до хранилища (внешний и внутренний адрес);

3. Состояние хранилища (размер всего дискового пространства и уже занятого)

Т.к. изменения в каталоге файлов и наборов может происходить достаточно часто, были введены дополнительные **history таблицы** для фиксации предыдущих состояний данных. Это полезно для последующего анализа корректности работы системы. Заполнение такого рода таблиц предполагается осуществлять автоматически при помощи триггера.

Отдельные **справочные таблицы** предназначены для строгого контроля возможных статусов файлов и наборов (см. таблицы 1, 2).

Код статуса	**Описание**
CREATED	Файл добавлен в систему
DAMAGED	Файл поврежден
TO_DELETE	Файл с пометкой «на удалении»
UPLOADING	Файл выгружается
DELETED	Файл удален из системы

Таблица 1: Справочник статусов файла (DIC_FILE_STATUS)

Код статуса	**Описание**
OPEN	Набор открыт
CLOSED	Набор закрыт
FROZEN	Набор временно «заморожен»
TO_UPLOAD	Набор с пометкой «на выгрузку»
UPLOADING	Набор выгружается
TO_DELETE	Набор с пометкой «на удалении»
DELETED	Набор удален из системы

Таблица 2: Справочник статусов набора (DIC_DATASET_STATUS)

4 Сервис dsm-register

4.1 Требования к сервису

Сервис должен слушать очередь сообщений и обрабатывать заявки на добавление/удаление данных в системе. В таблице 3 представлены шлюзы приёма сообщений. В качестве AMQP-брокера, который осуществляет маршрутизацию и подписку на нужные очереди, выбран RabbitMQ [6].

Exchange	**Routing Key**	**Назначение**
dsm.register (direct)	file.input	Приём информации о поступивших файлах на входной буфер
	file.process	Приём информации о новых файлах, полученных в процессе обработки
	dataset.close	Приём заявки на закрытие набора файлов
	dataset.upload	Приём заявки на выгрузку файлов в наборе во внешнее хранилище
	dataset.delete	Приём заявки на удаление файлов в наборе на внутреннем хранилище

Таблица 3: Перечень шлюзов приёма сообщений сервиса dsm-register

Шлюз **dsm.register.file.input** принимает сообщения вида: путь до хранилища, путь к файлу на хранилище, имя файла, его размер и контрольная сумма. Далее, выполняются следующие шаги:

1. Заводится набор по номеру frame-а со статусом OPEN;
2. Регистрируются файлы с привязкой к этому набору. Устанавливается первичный статус CREATE.

Шлюз **dsm.register.file.process** принимает сообщения вида: идентификатор набора, информация о файлах в наборе (путь к файлу на хранилище, включая имя файла, идентификатор хранилища, размер файла и его контрольная сумма). Далее происходит регистрация новых файлов (в статусе CREATE) с привязкой к указанному набору.

Шлюз **dsm.register.dataset.close** принимает сообщения вида: идентификатор набора и контрольный список файлов (имена файлов). Далее, выполняются следующие шаги:

1. Запрашивается список зарегистрированных файлов в указанном наборе;
2. Если данный список совпадает с контрольным списком, то у набора устанавливается статус CLOSE. Иначе сообщение возвращается обратно в очередь для отложенного исполнения.

Шлюзы **dsm.register.dataset.upload** и **dsm.register.dataset.delete** принимают только идентификатор набора. Далее, выполняются следующие шаги:

1. Запрашивается текущий статус набора;
2. Если набор находится в статусе OPEN, то сообщение возвращается обратно в очередь для отложенного исполнения;
3. Иначе устанавливается статус TO_UPLOAD и TO_DELETE соответственно.

Также при регистрации файлов, полученных в процессе обработки, необходимо отправлять в очередь ответ о статусе регистрации. Шлюз отправки сообщений представлен в таблице 4.

Exchange	**Routing Key**	**Назначение**
dsm.register (direct)	file.process.reply	Отправка информации о статусе регистрации файлов, полученных в процессе обработки

Таблица 4: Перечень шлюзов отправки сообщений сервиса dsm-register

Шлюз **dsm.register.file.process.reply** отправляет сообщения вида: статус регистрации файла (SUCCESS или ERROR), детали регистрации (если SUCCESS – полное имя зарегистрированного файла, если ERROR – полное имя файла и тип возникшей ошибки).

4.2 Взаимодействие с RabbitMQ

Главна причина использования использования брокера сообщений — асинхронный обмен. Он предполагает отправку сообщения от одного сервиса к другому, при этом деятельность сервиса-отправителя не приостанавливается в ожидании ответа от получателя.

Как уже упоминалось выше, в качестве брокера сообщений был выбран RabbitMQ — популярный брокер, который служит посредником для обмена информацией между различными системами. Он осуществляет передачу сообщений посредством очередей. Основные компоненты RabbitMQ представлены на рисунке 5.

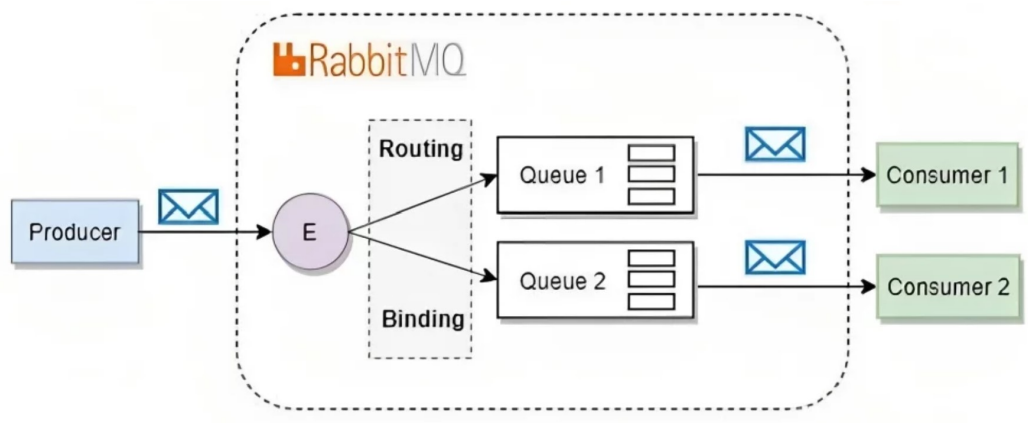

Рисунок 5 – Основные компоненты RabbitMQ.

Работа RabbitMQ происходит следующим образом [7]:

1. Отправитель (producer) отправляет сообщение в обменник (exchange);
2. Обменник направляет сообщение в нужную очередь (queue);
3. Получатель (consumer) подписывается на очередь и начинает получать сообщения из нее;
4. Получатель обрабатывает сообщение и подтверждает его получение, тем самым исключая из очереди;
5. Если сообщение не было подтверждено в течение определенного времени, оно повторно отправляется в очередь.

4.3 Прототип сервиса

Описание структуры проекта представлено на рисунке 6.

```
├── README.md                          <- Файл верхнего уровня для разработчиков.
│
├── app
│   ├── executor
│   │   ├── main.py                    <- Точка входа (построение приложения).
│   │   ├── config                     <- Настройки всех компонент приложения.
│   │   ├── domain                     <- Модели данных приложения.
│   │   ├── dto
│   │   │   ├── rabbit                 <- DTO модели сообщений RabbitMQ.
│   │   │   └── manager                <- DTO модели DSM-Manager.
│   │   ├── rabbit                     <- Модуль работы с RabbitMQ.
│   │   │   ├── consumers              <- Модуль приёма сообщений из очередей RabbitMQ.
│   │   │   ├── reply                  <- Модуль отправки ответов в очереди RabbitMQ.
│   │   │   ├── rabbit_connector.py    <- Объект доступа к серверу RabbitMQ.
│   │   │   └── rabbit_server.py       <- Старт компоненты работы с RabbitMQ.
│   │   ├── services
│   │   │   ├── processor              <- Бизнес логика обработки сообщений.
│   │   │   └── manager                <- Adapter к сервису DSM-Manager.
│   │   └── utils                      <- Общие методы.
│   └── settings                       <- Настройки приложения.
│
├── Dockerfile                         <- Файл с настройками Docker образа приложения.
│
├── docker-compose.yml                 <- Docker Compose файл с настройкой инфраструктуры.
│
└── pyproject.toml                     <- Poetry файл с зависимостями приложения.
```

Рисунок 6 – Описание структуры проекта dsm-manager.

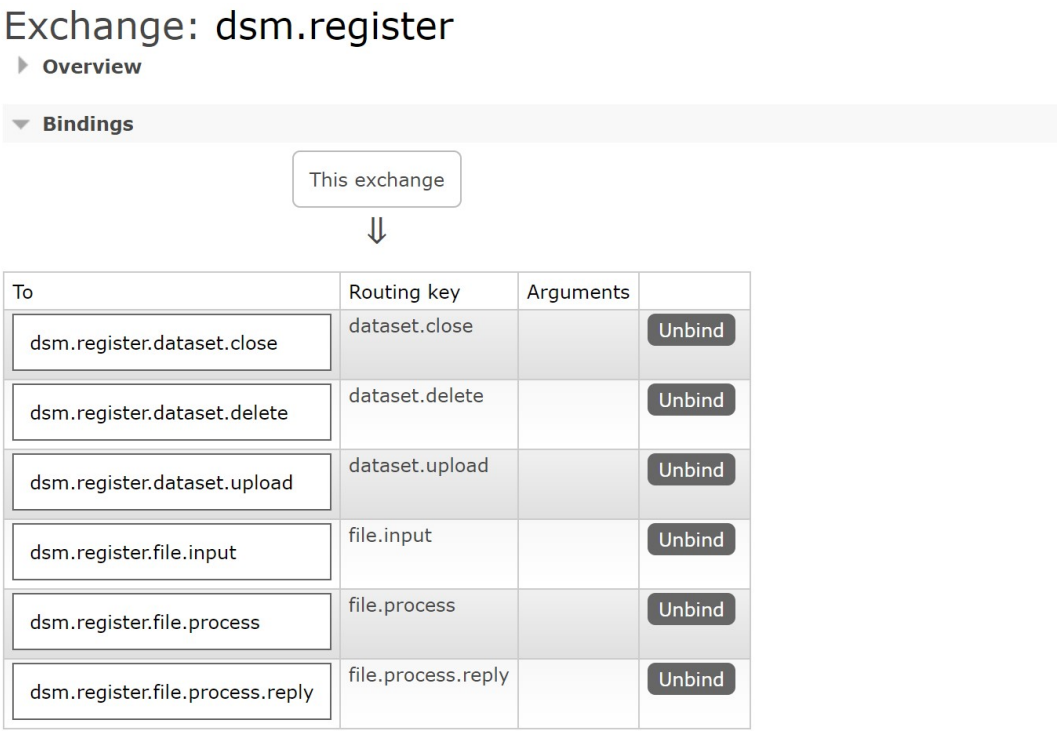

Рисунок 7 – Сконфигурированные очереди RabbitMQ.

Сервис dsm-register работает с брокером сообщений RabbitMQ, вслед-

ствие чего необходимо предусмотреть настройку очередей и их обработчиков. Сконфигурированные очереди для сервиса представлены на рис. 7.

Также на сервисе была предусмотрена возможность отклонения некорректных сообщений. Т.е. в случае, если при обработке сообщения возникает исключение, срабатывает обработчик исключения. При этом, чтобы такого рода сообщения не были полностью утеряны, для каждой очереди реализована дополнительная «очередь мёртвых сообщений» (DLQ/Dead letter queue) (рисунок 8).

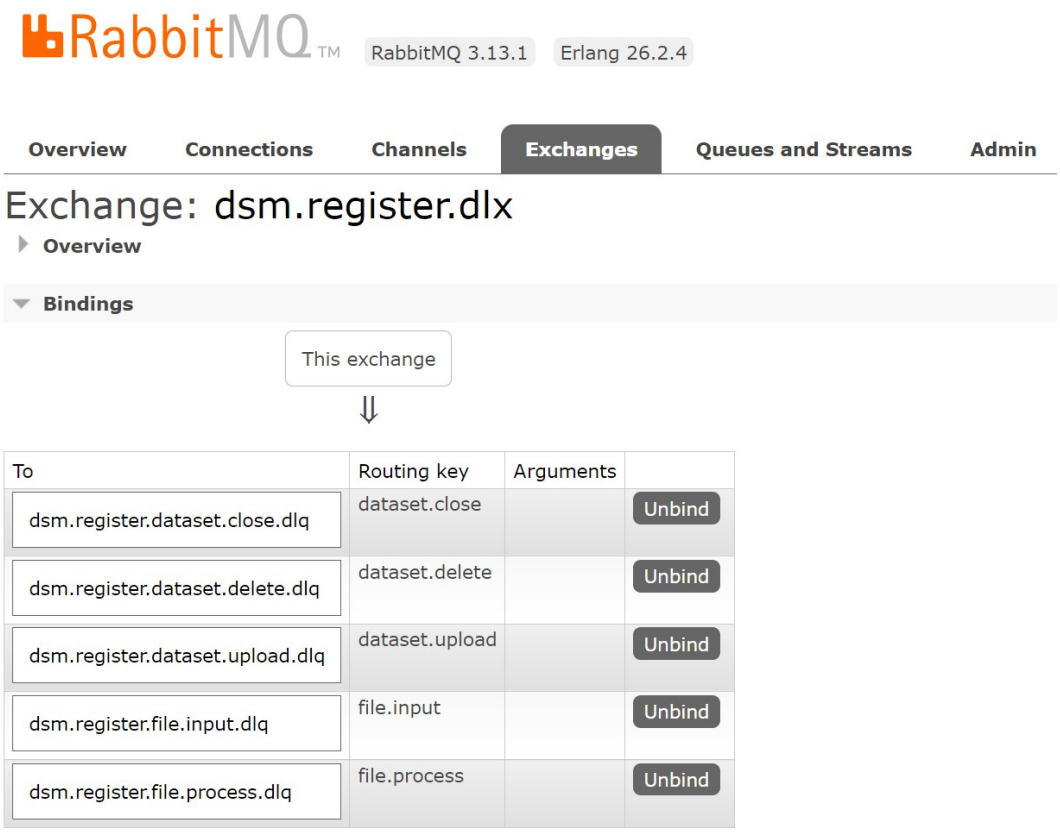

Рисунок 8 – Сконфигурированные «очереди мёртвых сообщений» RabbitMQ.

4.4 Проверка работоспособности

В ходе работы был создан новый comsumer для приема сообщений из очереди *dsm.register.file.process* и реализована логика обработки сообщений из нее (регистрация промежуточных файлов и отправка сообщений о статусе регистрации). Для этого потребовалось дополнительно создать очередь *dsm.register.file.process.reply* и настроить механизм отправки ответов в нее.

Для начала через панель администрирования RabbitMQ попробуем отправить в очередь dsm.register.file.process сообщение с информацией о промежуточном файле (рисунок 9).

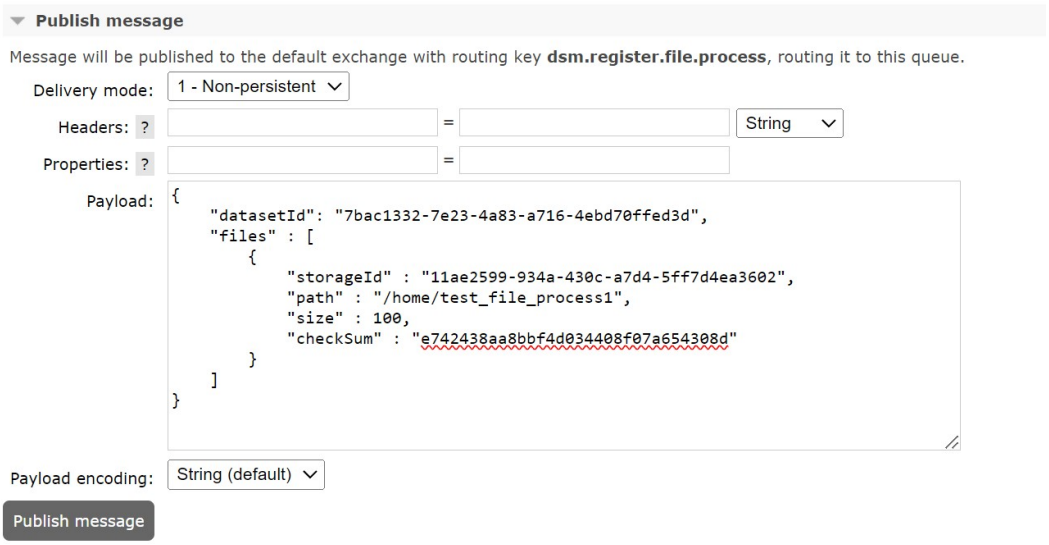

Рисунок 9 – Отправка сообщения в очередь промежуточных файлов dsm-register.

Проверим, что файл появился в системе с привязкой к нужному набору (рисунок 10) и что в очередь dsm.register.file.process.reply было отправлено сообщение об успешной регистрации файла (рисунок 11).

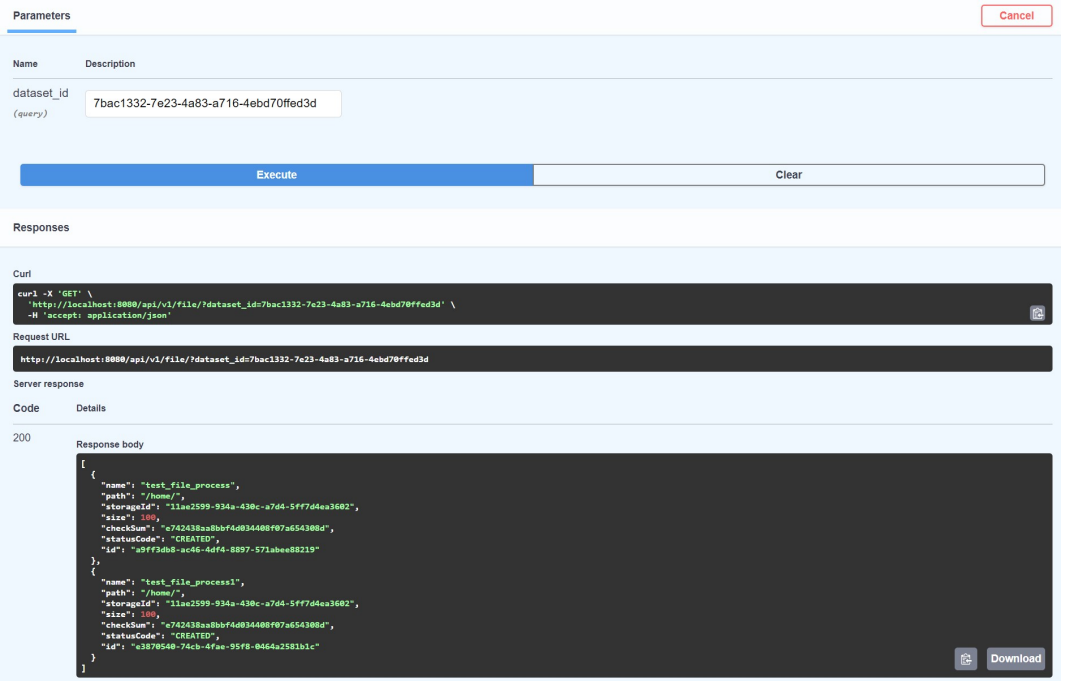

Рисунок 10 – Проверка регистрации промежуточного файла.

Рисунок 11 – Проверка отправки ответа в очередь.

Теперь временно приостановим работу сервиса dsm-manager и снова отправим в очередь dsm.register.file.process сообщение с информацией о файле (рисунок 12).

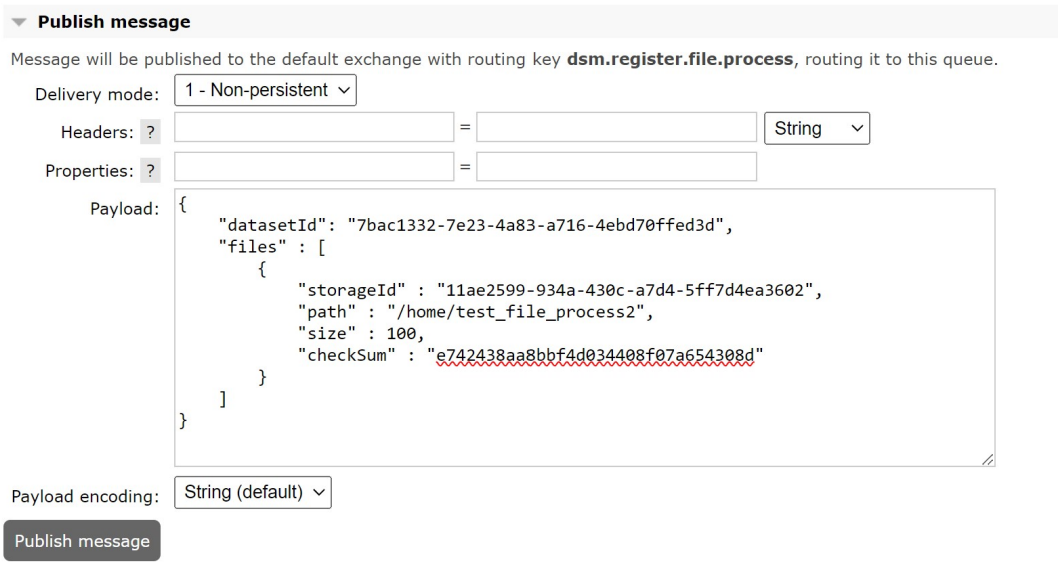

Рисунок 12 – Отправка сообщения в очередь промежуточных файлов dsm-register.

Проверим, что в очередь dsm.register.file.process.reply поступило сообщение об ошибке регистрации файла (рисунок 13).

Рисунок 13 – Проверка отправки ответа в очередь.

Заключение

При проведении экспериментов на коллайдере чрезвычайно важно решить проблему обработки и хранения огромного объема данных, получаемого в результате столкновений.

В связи с этим в контексте данной работы рассмотрен вычислительный комплекс SPD Online Filter, который предназначен для быстрой реконструкции событий и сокращения объема выходных данных без потери их физической ценности. Это является важным шагом в обеспечении эффективной обработки и анализа экспериментальных данных. И, как следствие, является ключом к пониманию спиновой структуры нуклона.

Также была рассмотрена одна из составляющих системы SPD Online Filter — система управления данными.

В работе подробно разбирается функционал сервиса dsm-register, который принимает заявки на добавление/удаление данных в системе.

Для данного сервиса была реализована логика обработки сообщений из очереди dsm.register.file.process, которая предназначена для приема информации о новых файлах, полученных в процессе обработки. Сюда входит получение информации о файлах, их регистрация в системе и отправка ответов о статусе регистрации в новую очередь dsm.register.file.process.reply.

В дальнейшем планируется закончить реализацию микросервисов системы управления данными, а также доработать взаимодействие данной системы с системами управления процессами и нагрузкой.

Список используемых 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 / под ред. V. W. Hughes, C. Cavata. — 1988. — Т. 206. — С. 364.
2. *Abazov V. M.* [и др.]. Conceptual design of the Spin Physics Detector. — 2021. — Янв. — arXiv: 2102.00442 [hep-ex].
3. *Tereshenko D.* Designing a Data Management Service in a Specialized Distributed Computing System SPD Online Filter. — 2023.
4. PostgreSQL 12.15 Documentation. — URL: https://www.postgresql.org/docs/12/index.html.
5. Связи между таблицами базы данных. — URL: https://habr.com/ru/articles/488054/.
6. RabbitMQ Documentation. — URL: https://www.rabbitmq.com/docs.
7. RabbitMQ: что это такое и как работает. — URL: https://blog.skillfactory.ru/rabbitmq-chto-eto-takoe-i-kak-rabotaet/.