Санкт–Петербургский государственный университет

ПОНОМАРЕВ Евгений Владимирович

Выпускная квалификационная работа

Проектирование сервиса управления процессом обработки данных на специализированной распределенной вычислительной системе SPD Online filter

Уровень образования: магистратура

Направление: 02.04.02 «Фундаментальная информатика и информационные технологии»

Основная образовательная программа «Распределенные вычислительные технологии» №21/5827/1

Научный руководитель:
доцент, Кафедра компьютерного моделирования
и многопроцессорных систем,
к.ф. – м.н. Корхов Владимир Владиславович

Руководитель от организации:
ведущий программист, МЛИТ, ОИЯИ,
к.т.н. Олейник Данила Анатольевич

Рецензент:
ведущий программист, МЛИТ, ОИЯИ,
к.т.н. Петросян Артём Шмавонович

Санкт-Петербург
2023

Saint–Petersburg State University

PONOMAREV Evgenii Vladimirovich

Graduation project

Designing a data processing process management service on a specialized distributed computing system SPD Online filter

Level of education: Master level

Main field of study 02.04.02 «Fundamental Informatics and Information Technology»

Academic programme title «Distributed Computational Technologies»

No21/5827/1

Scientific supervisor:

PhD in Physics and Mathematics

V.V .Korkhov

Supervisor from organization:

lead software developer, MLIT, JINR

PhD in Technical science

D.A.Oleynik

Reviewer:

lead software developer, MLIT, JINR

PhD in Technical science

A.Sh.Petrosyan

Saint–Petersburg

2023

Список сокращений и условных обозначений

SPD — Spin Physics Detector

NICA — Nuclotron based Ion Collider fAcility

DAQ — Data Acquisition System

CWL — Common Workflow Language

ATLAS — A Toroidal LHC Apparatus

JEDI — Job Execution and Definition Interface

DEFT — Database Engine for Tasks

ETL/ELT — Extraction, Transformation, Loading

API — Application Programming Interface

WfMS — Workflow Management System (система управления процессом обработки данных)

DMS — Data Management System (система управления данными)

WMS — Workload Management System (система управления нагрузкой)

Введение

SPD (Spin Physics Detector) [1] это строящийся эксперимент на коллайдере NICA [2], мегасайенс-установке, которая строится в ОИЯИ (г.Дубна, Россия) см. Рис. 1.

Основная цель эксперимента — проверка основ квантовой хромодинамики путем изучения поляризованной структуры нуклона и спиновых явлений при столкновении продольно и поперечно поляризованных протонов и дейтронов с энергией центра масс до 27 ГэВ и светимостью до 10^{32} $см^{-2}$ $с^{-1}$.

Детектор SPD задуман как универсальный 4π-спектрометр, основанный на современных технологиях. Общее количество каналов регистрации в установке SPD составляет около 500000. С учетом ожидаемой максимальной частоты возникновения интересующих взаимодействий пучков частиц около 3 МГц, суммарный поток данных с детектора можно оценить как 20 ГБ / с, что эквивалентно 200 ПБ/год (для эксперимента предполагается выделить 30% пучкового времени коллайдера). Сбор, обработка и хранение такого объема данных представляет собой серьезную проблему для вычислительной инфраструктуры эксперимента и требует разработки новых методов и подходов для реконструкции событий, моделирования и физического анализа данных с использованием высокопроизводительных и распределенных вычислений.

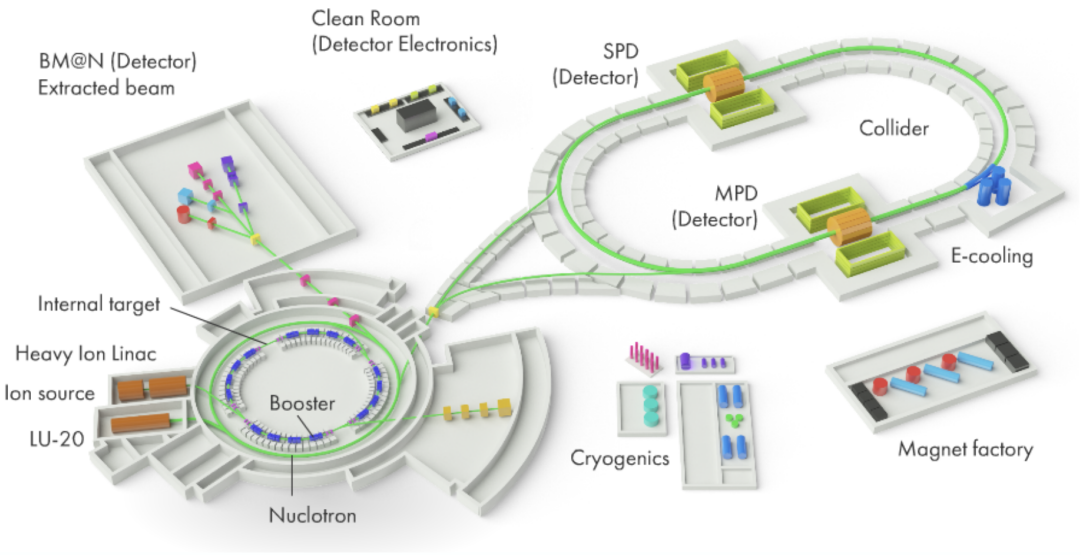

Рисунок 1. Схема комплекса NICA

Существенный объем получаемых с детектора данных не будет являться практически значимым для конкретного физического исследования, однако применение методики отбора данных по триггерному сигналу не применимо к детектору SPD ввиду его конструктивных особенностей. Для решения задачи первоначальной подготовки и фильтрации данных ведется разработка специализированной вычислительной системы SPD Online Filter. Для разработки данной системы необходимо учитывать, что процессы обработки данных являются многоступенчатыми: необходимо выявлять «события» в полученных с DAQ [4] данных, фильтровать события по заданным критериям, оптимизировать данные для обработки, передачи и хранения и т.д.

Организация SPD Online Filter

Так как SPD Online Filter работает с большим объемом данных, он должен уметь их правильно и быстро обрабатывать. Для этого большие наборы данных разбиваются на более мелкие части, которые можно обработать на ограниченном вычислительном ресурсе за разумное время с достаточным уровнем надежности. Обработка маленьких частей выполняется одновременно на отдельных вычислительных ресурсах. Такой подход помогает снизить риск ошибок, которые могут возникнуть при работе с большими наборами данных, нивелировать последействия определённого количества сбоев в обработке. Кроме того, данный подход обеспечивает большую масштабируемость, поскольку в процесс обработки можно добавить дополнительные вычислительные ресурсы для обработки еще больших наборов данных.

По результатам анализа первичных требований к системе были выделены основные компоненты:

1. Workflow management system
2. Data management system
3. Workload management system & pilot

Архитектура SPD online filter представлена на Рис. 2

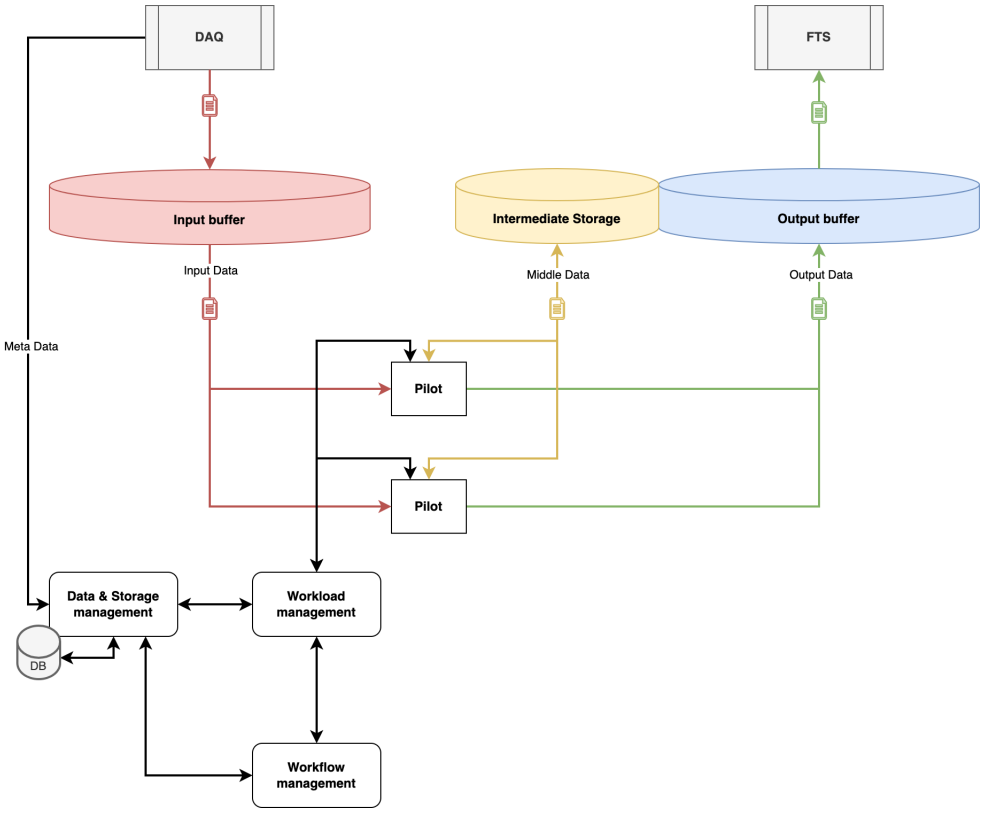

Рисунок 2. Архитектура SPD Online filter

Входные данные приходят в систему из системы сбора данных (DAQ). DAQ SPD обеспечивает считывание и передачу данных со всех подсистем детектора. К ним относятся:

- вершинный детектор: кремниевый трековый детектор на основе двойной полосы для точного определения первичной точки взаимодействия и измерения вторичных вершин по распадам короткоживущих частиц;
- строу-трекер: основной трековый детектор для измерения импульсов первичных и вторичных частиц по кривизне треков в магнитном поле, а также для измерения ионизационных потерь;
- детекторы идентификации частиц: времяпролетные резистивные пластинчатые камеры и аэрогелевые черенковские счетчики;
- электромагнитный калориметр для измерения энергии гамма-излучения и электронов (позитронов);
- полигонная система обнаружения и идентификации мюонов;

- счетчики пучков для локальной поляриметрии и контроля столкновений пучков;
- калориметр нулевого градуса для локальной поляриметрии с использованием прямых нейтронов и для измерения светимости.

Более подробно про DAQ см. [4].

После попадания в SPD Online Filter, данные регистрируются в системе управления данными. После регистрации, данные готовы к обработке с помощью системы управления рабочим процессом. WfMS берет данные, сопоставляет их с некоторым шаблоном цепочки, который задается оператором обработки данных , и для каждого этапа обработки данной цепочки создает задания. Задания отправляются в систему управления нагрузкой, с помощью которой они обрабатываются.

Постановка задачи

Цель данного проекта заключается в проектировании сервиса, предназначенного для эффективного управления процессом обработки данных. Сервис должен автоматически прогонять данные через предопределенные последовательности обработки, генерируя соответствующие задания и отслеживая их текущий статус. Система должна обеспечивать возможность одновременного выполнения нескольких цепочек обработки данных, что является ее ключевой функцией.

Глава 1. Обзор существующих решений и инструментов

Workflow Management System (WfMS) — это ключевая компонента множества разных систем, обеспечивающая высокопотоковую многоступенчатую обработку данных. В данной главе будет проанализирована WfMS схожей системы, чтобы выявить ее преимущества и недостатки и возможность интеграции с другими системами. Также будет произведен обзор инструментов, которые потенциально можно применить для управления процессом обработки.

В процессе обзора будет уделено внимание таким аспектам, как функциональность и одно из самых главных — интеграционные возможности. Кроме того, будут рассмотрены вопросы удобства использования. Все эти аспекты являются важными факторами при выборе подходящего WfMS.

На основе данной главы, можно будет получить представление о WfMS ATLAS и понять возможность ее использования с нашей системой. В случае невозможности интегрирования, необходимо выделить узкие места, чтобы избежать их в SPD Online Filter.

ATLAS production system

Программная система Production and Distributed Analysis (PanDA) [7] управляет рабочей нагрузкой эксперимента ATLAS (Рис. 3) [8] на распределенных ресурсах Worldwide LHC Computing Grid.

The ATLAS detector

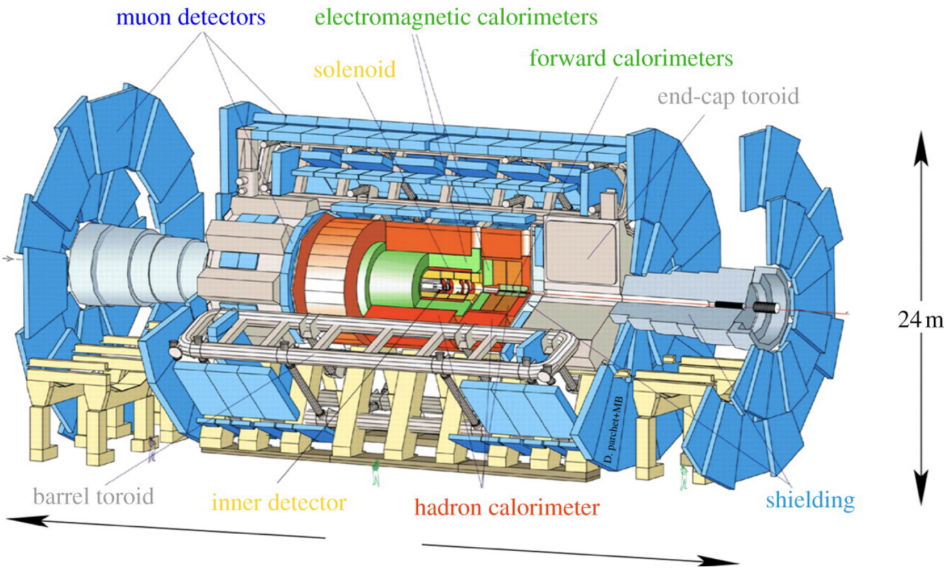

Рисунок 3. The ATLAS detector

Типичный рабочий процесс обработки для ATLAS состоит из множества шагов преобразования данных. Он может состоять из моделирования методом Монте-Карло, который в свою очередь состоит из многих этапов: создание сложных процессов, адронизация сигналов и событий с минимальным смещением, моделирование выделения энергии в детекторе ATLAS, оцифровка отклика электроники, моделирование триггеров, реконструкция данных, преобразование восстановленных данных в сокращенные формы для физический анализ [5].

В ATLAS есть десятки рабочих процессов, используемых разными группами пользователей. Все рабочие процессы имеют схожий поток действий, показанный на рис. 4. Первые три слоя на данном рисунке это и есть система управления процессами обработки. Сначала пользователь указывает данные для обработки располагая информацию во внешнем источнике, таком как структурированная электронная таблица Google, или во внутреннем источнике, таком как веб-форма, или в ранее созданном шаблоне.

После подачи описания данных определяются недостающие параметры по паттернам из шаблонов. После того, как заявка готова, она

подается в обработку. На этом этапе задание инициируется с использованием определения задания. Если задание определить не удалось, создается ошибка и уведомляется пользователь. Определенные задания выполняются JEDI, и их можно отслеживать и управлять ими через веб-интерфейс системы управления процессами обработки. Пользователь получает уведомление, если задание не может быть отправлена. После выполнения задания инициируется постобработка, основной целью которой является заполнение метаданных и удаление наборов данных, которые больше не нужны. [6]

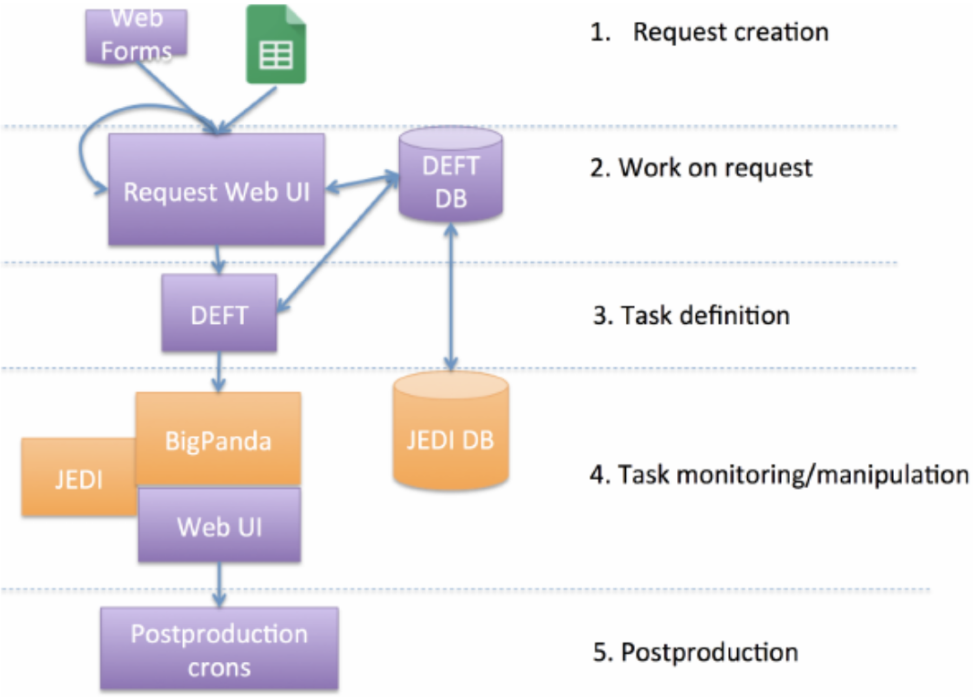

Рисунок 4. Рабочий процесс системы управления процессами обработки

В моменте создания задания, после установки параметров система проверяет их согласованность и совместимость. Например, система проверяет, достаточно ли событий во входном наборе данных или можно ли использовать определенные параметры для выбранной версии программного обеспечения. В то же время система проверяет, были ли уже созданы подобные задания, и, если это так, отклоняет задание, поскольку она может создавать повторяющиеся события. Во время определения задания, когда система управления процессами обработки обнаруживает что некоторые события из входного набора данных уже использовались, DEFT

устанавливает смещение, и новое задание использует или генерирует только неповторяющиеся события. В случае, если задание не может быть определено, DEFT создаст тикет с подробным описанием и уведомит менеджера о задании. Если все проверки успешно пройдены, DEFT отправляет задание в JEDI. [5]

На основе представленной информации, система управления рабочими процессами ProdSys2 значительно ориентирована на ATLAS. У SPD Online Filter отличается метод получения данных и формат данных. Самая главная проблема заключается в отсутствии интеграции с другими системами. У данной WfMS предусмотрена интеграция только с PanDA-JEDI, однако отсутствует информация об этом в документации.

Использование готовых инструментов для управления процессом обработки данных. Apache Airflow.

В данном разделе рассматривается один из популярных инструментов для управления обработкой данных - Apache Airflow [10].

Apache Airflow представляет собой открытое решение с открытым исходным кодом, которое служит в качестве оркестратора для разработки, планирования и мониторинга сложных рабочих процессов. Он также выполняет функции планировщика для процессов ETL/ELT и использует язык программирования Python [10].

Основная концепция Airflow основана на направленном ациклическом графе, который описывает процессы обработки данных и определяет связи и правила выполнения задач.

Airflow состоит из нескольких компонентов, но ключевыми из них являются планировщик (scheduler) и веб-сервер (webserver). Они являются неотъемлемой частью функционирования системы.

Планировщик отслеживает задачи и DAG, а затем запускает экземпляры задач после выполнения их зависимостей. В фоновом режиме планировщик запускает подпроцесс, который следит и синхронизирует все

DAG в указанном каталоге. По умолчанию он проверяет результаты синтаксического анализа DAG каждую минуту и определяет, нужно ли запустить какие-либо активные задачи [10].

WebServer отвечает за отображение веб-интерфейса и аутентификацию пользователей, а также решает, к какой группе должен относиться тот или иной пользователь в соответствии с конфигурационным файлом. Более подробно см. [10].

Apache Airflow обладает множеством преимуществ и широким функционалом, но есть ситуации, когда его использование может быть неоптимальным — задачи обработки в реальном времени. Apache Airflow предназначен для планирования и выполнения задач в определенное время, а не для обработки данных в реальном времени. Если необходимо обработать данные по мере их поступления или в ситуациях с высокой пропускной способностью, то в этом случае не стоит использовать данный инструмент [19]. В SPD Online filter как раз и нужно обрабатывать данные по мере их поступления.

Помимо всего прочего, нужно учитывать, что WfMS должна общаться с другими системами, отправлять им запросы, ждать ответов и исходя из этого принимать дальнейшие решения. С помощью Apache Airflow довольно тяжело организовать такую систему.

Исходя из этого, в SPD Online Filter не используется Apache Airflow.

Глава 2. Определение требований к WfMS.

Функциональные требования к системе wfms

- Организация последовательностей (цепочек) обработки данных
  - Последовательность — это набор преобразований данных разбитых на логические шаги. Результат работы предыдущего шага является входными данными для последующего
  - Шаг обработки представляет собой однотипное действие над набором данных.
  - Шаги могут быть двух типов:
    - одинаковая обработка для каждого элемента данных в наборе (map).
    - Объединение гомогенных данных (merge)
  - Шаблон описывается с помощью CWL [3].
- Формирование запроса на обработку данных по определенной последовательности.
  - Для каждого шага обработки задаются значения переменных:
    - входной набор данных
    - тип обработки (map, reduce)
    - обработчик (приложение)
    - шаблон входных параметров для обработчика
    - выходной набор данных
    - Граничные условия: минимальный и достаточный порог выполнения работы от объема обрабатываемых данных.
    - Количество повторений, в случае возникновения ошибки
- Выполнение запроса на обработку: последовательное формирование заданий на обработку, передача заданий в систему управления нагрузкой, контроль выполнения посредством опроса системы управления нагрузкой.

Основные шаги обработки данных:

В основном процессе обработки данных в SPD Online filter можно выделить следующие шаги

1. Декодирование данных
2. Выявление событий и реструктуризация данных
3. Фильтрация выявленных событий
4. Верификация данных
5. Объединение файлов (Уменьшение количества файлов, увеличение размера файлов)

Такой набор последовательных шагов обработки будет называться цепочкой обработки. Каждый из шагов в такой цепочке может быть определен шаблоном.

Глава 3. Организация данных

Входные данные

Входные данные приходят из DAQ во входной буфер.

- **Период набора** — ассоциирован с физической задачей. Состоит из набора ранов (run). Имеет дату начала, дату окончания.
- **Ран** — интервал, когда условия эксперимента неизменны (калибровки например). Измеряется часами, может быть связан с Condition Database (напрямую или опосредованно). Состоит из фреймов.
- **Фрейм** — нумерованный блок данных с детектора. Может содержать информацию о множестве событий. Продолжительность секунды.
- **Датасет —** логическое объединение файлов в наборы для обработки. Датасет является единицей обработки для одного задания.
- **Файл** — количество данных для обработки одной задачей.

Промежуточные данные

Промежуточные данные — данные, образующиеся в процессе обработки, но еще не находящиеся в состоянии готовности для использования в системах оффлайн обработки.

Есть привязка к одному или нескольким датасетам, рану и периоду.

Файлы группируются в датасеты (логические наборы) для последующего шага в процессе обработки.

Промежуточные данные живут только в рамках системы и не подразумевается их долговременное хранение => они должны регулярно удаляться.

Выходные данные

Выходные данные — данные для последующей обработки или использования вне функционала онлайн фильтра.

Ассоциированы с периодом набора и раном.

Логическая группировка в наборы, с учетом того что один и тот же файл может быть в разных группах (минимум в одной)

Глава 4. Взаимодействие с другими системами SPD Online Filter

Взаимодействие с Data Management System

Далее представлено подробное описание процесса взаимодействия с Data Management System (DMS). Основной задачей данного взаимодействия является получение новых данных, регистрация промежуточных данных и выгрузка выходных данных. Помимо основных функция этого взаимодействия, оно должно быть реализовано с учетом обеспечения высокой эффективности и максимальной скорости взаимодействия, исключении узких мест, а также минимизации возможных ошибок.

Целью проектирования данного взаимодействия является устранение возможных узких мест, которые могут возникнуть при коммуникации с DMS. Для этого применяются способы взаимодействия, которые позволяют максимально эффективно управлять процессом взаимодействия.

Взаимодействие с DMS:

1. Получение входного набора данных. После поступления данных в SPD Online Filter, DMS регистрирует их у себя в системе. После данного шага, WfMS может получить их информацию об этом датасете с помощью Rest api.
2. В процессе обработки данных по некоторому шаблону, будут появляться промежуточные данные, если они получены после любого промежуточного этапа обработки, и выходные данные, если они получен после последнего этапа обработки. Эти новые данные WfMS должна регистрировать в DMS. Но для этого нужно понимать, какое хранилище использовать: хранилище промежуточных данных или хранилище выходных данных. Эти хранилища могут меняться по ходу работы системы. Поэтому следующим взаимодействием с системой

DMS можно выделить получение Id хранилища и URI хранилища для регистрации этих данных. Здесь также используется Rest api.

3. Перед любым этапом обработки данных, WfMS должен зарегистрировать выходной набор данных. Это нужно для того, чтобы понимать, где и как искать полученные данные. Здесь также используется Rest api. Стоит отметить, что для промежуточных и выходных данных в процессе обработки образуются логи, которые также нужно нужно регистрировать аналогичным способом.

4. По мере прохождения цепочки обработки, появляются промежуточные данные. Если в цепочке будет n число этапов, то при сохранении всех промежуточных данных, после заключительного этапа обработки будет n новых датасетов, что сильно увеличит объем задействованной памяти. В связи с этим, было принято решение сохранять только один набор входных данных и один набор промежуточных данных. Т.е. после успешного прохождения некоторого этапа обработки данных и появления нового датасета, входные данные для этого этапа должны удаляться. Входные данные для первого этапа остаются до тех пор, пока не появятся выходные данные (данные пройдут всю цепочку обработки). Это нужно для восстановления данных в случае сбоя системы. В данном взаимодействии используется асинхронный вариант. Это позволяет не останавливать работу WfMS и не ждать удаления данных. Стоит отметить, что для промежуточных и выходных данных в процессе обработки образуются логи, которые также нужно удалять аналогичным способом.

5. При получении выходного датасета после прохождения последнего этапа обработки данных их нужно выгрузить в другую систему. WfMS передает информацию об этом датасете DMS и создает заявку на выгрузку этих данных. Здесь также происходит асинхронное взаимодействие, что позволяет не останавливать систему на этом месте.

Далее представлена таблица (см. Табл. 1) взаимодействия WfMS и DMS со всеми необходимыми параметрами:

Таблица 1. Взаимодействие с DMS

Запрос	**Тип запроса**	**Input 1**	**Input 2**	**Output**	**Примечания**
Получение входного набора данных	Rest api	timestamp/None.	—	[{ ● uid датасета ● name датасета ● timestamp регистрации ● метаинформация ● кол-во файлов}, ….]	Возвращает список параметров датасетов зарегистрированных в системе позднее timestamp. В случае None возвращает все датасеты
Получение информации о хранилище	Rest api	Type intermediate data/output/logs	—	● Id хранилища ● URI хранилища	
Создание выходног	Rest api	name датасета	metadata	—	Имя для выходного

о набора					набора моя система создает сама
Запрос на статус датасета	Rest api	uid датасета	—	status	По статусу необходимо понимать, можно ли работать с датасетом
Заявка на удаление набора	RabbitMQ	uid датасета	—	—	Запрос на удаление набора отправляется в систему очередей. Нет необходимости знать, что набор точно удалился, т.к. это зона ответственности DMS
Заявка на выгрузку данных	RabbitMQ	uid датасета	—	—	Запрос на выгрузку данных отправляется в систему очередей. Нет необходимости знать, что набор точно выгрузился, т.к.

					это зона ответственности DMS

Таким образом, в данном пункте подробно описано взаимодействие с DMS. Описаны зоны ответственности каждой из систем, определен список всех необходимых входных и выходных параметров. Для достижения высокой эффективности взаимодействия с DMS мы применяем различные стратегии и технологии. В некоторых случаях, где асинхронность является важным фактором, мы используем асинхронные запросы. Это позволяет нам выполнять несколько операций параллельно и максимально утилизировать ресурсы системы, ускоряя процесс взаимодействия.

Взаимодействие с Workload Management System

Далее представлено подробное описание процесса взаимодействия с Workload Management System(WMS). WMS должен распределять нагрузку по узлам. В SPD Online Filter WfMS формирует задания на основе некоторого шаблона для каждого этапа обработки и передает эти задания WMS, который в свою очередь отвечает за их выполнение. Аналогично взаимодействию с DMS, оно должно быть реализовано с учетом обеспечения высокой эффективности и максимальной скорости взаимодействия, исключении узких мест, а также минимизации возможных ошибок.

SPD Online Filter работает с большим потоком данных, которые нужно уметь быстро обрабатывать. Для этого применяются способы взаимодействия, которые позволяют максимально эффективно выполнять требуемый функционал.

Взаимодействие с WMS:

1. После получения новых данных WfMS сопоставляет их с некоторым шаблоном цепочки обработки. Далее система управления процессом обработки данных формирует задание по первому этапу цепочки обработки и отдает его WMS. WMS обрабатывает данные по этому заданию и на выходе получаются новые данные. Для них в свою очередь WfMS формирует новое задание для следующего этапа обработки и опять отдает их WMS. Данный процесс повторяется до тех пор, пока данные не пройдут все этапы обработки и не получатся выходные данные.
2. После того как задание ушло на обработку в WMS необходимо понимать, что с ним происходит и когда задание будет выполнено. Для этого WfMS регулярно опрашивает WMS информацию о количестве обработанных файлов и количестве ошибок.
3. В ходе работы системы могут случаться экстренные ситуации. Например, входной буфер будет переполняться данными быстрее, чем SPD Online Filter будет их обрабатывать. В таких ситуациях необходимо

производить действия по ускорению обработки. В качестве таких действий, которые может сделать WfMS, выступает изменение приоритетов заданий.

4. Также необходимо уметь отменять некоторые задания. Это может понадобиться в тех случаях, если количество ошибок будет превышать количество обработанных файлов.

Далее представлена таблица (см. Табл. 2) взаимодействия WfMS и WMS со всеми необходимыми параметрами:

Таблица 2. Взаимодействие WfMS и DMS

Запрос	**Тип запроса**	**Input 1**	**Input 2**	**Output**	**Примечания**
Регистрация задания	RabbitMQ	Task	—	—	Описание задания будет представлено ниже

Получение информации о статусе обработки задач	Rest api	id задания	—	{
				● total
				● running
				● canceled
				● killed
				● failed
				}
					Данный запрос будет использоваться не только для автоматической работы WfMS, но также может использоваться оператором обработки

					данных, для верхнеуровневого мониторинга
Изменение приоритета задания	Rest api	id задания	priority	Task info	—
Отмена задания		id задания	priority	Task info	—

Задание представляет собой единицу рабочей нагрузки, связанную с обработкой определенного датасета. Если обработка данных выполняется в несколько этапов, каждый этап соотносится с отдельным заданием. Задание формально определяется как набор входных данных и обработчик, который выполняет необходимые операции над данными и генерирует выходной набор данных и логов.

Далее представлено описание задания, которое описано в формате JSON (см. Таблицу 3)

Таблица 3. Описание задания

Ключ	***Описание***
uid	уникальный идентификатор задания
executable	исполняемый файл для запуска

args	флаги для исполняемого файла задания
rank	назначенный WfMS приоритет задания, в дальнейшем изменяющийся по следующей формуле: $rank_{i+1} = \alpha \cdot x_i + \beta \cdot y_i + \gamma \cdot rank_i$ $x_i - aging$ $y_i - retries$
device_type	Определение выбранного вычислительного ресурса, который будет использоваться для выполнения задания ● CPU ● CPU/GPU
mode	указание, какой тип задания будет выполняться ● Map ● Merge
dat_in_uid	уникальный идентификатор входного датасета
retry	Определение максимального числа попыток выполнения для задач,

	связанных с этим заданием, в случае неудачного выполнения.
dat_out_uid	уникальный идентификатор выходного датасета
dat_stor_url	унифицированный указатель ресурса выходного хранилища
dat_out_url	унифицированный указатель ресурса выходного датасета
log_out_uid	уникальный идентификатор выходного датасета логов
log_stor_url	унифицированный указатель ресурса выходного хранилища для хранения логов
log_out_url	унифицированный указатель ресурса выходного датасета логов

Таким образом, в данном пункте подробно описано взаимодействие с Workload Management System. Описаны зоны ответственности каждой из систем, определен список всех необходимых входных и выходных параметров. Для достижения высокой эффективности взаимодействия с WMS мы применяем различные стратегии и технологии. В некоторых случаях, где

асинхронность является важным фактором, мы используем асинхронные запросы (для отправки заданий). Это позволяет нам выполнять несколько операций параллельно и максимально утилизировать ресурсы системы, ускоряя процесс взаимодействия.

Глава 5. Сервисы Workflow Management System

Исходя из функциональных требований и взаимодействия других с другими системами можно выделить несколько сервисов:

1. Сервис для взаимодействия с оператором обработки данных
2. Сервис для опроса DMS
3. Сервис для генерации заданий
4. Сервис для опроса WMS и очистки промежуточных данных

Для данных сервисов использовался следующий стек технологий:

- Python 3.11
- Postgres 15
- Asyncio
- Sqlalchemy
- sqlalchemy.ext.asyncio
- FastAPI
- Uvicorn
- Flask
- Rest
- RabbitMQ
- Pika
- Docker
- Docker-compose

В данной системе используется микросервисная архитектура. В микросервисной архитектуре приложение структурируется на отдельные сервисы, которые могут быть независимо развернуты и масштабированы. Взаимодействие между сервисами осуществляется через API-интерфейсы. Этот подход позволяет каждому сервису функционировать независимо от других. В отличие от монолитной архитектуры, микросервисы позволяют

более быстро внедрять новые возможности и вносить изменения, не требуя переписывания больших фрагментов существующего кода.

Монолитное приложение — это единый общий модуль, в то время как микросервисная архитектура представляет собой набор небольших независимо развертываемых служб. [9]

Главными преимуществом микросервисной архитектуры являются:

- повышение показателей доступности и отказоустойчивости. Ошибки и отказ одного из сервисов не приведут к отказу всей системы целиком.
- Масштабируемость. Когда микросервис достигает предельной нагрузки, можно выполнить развертывание новых экземпляров данной службы и распределить нагрузку. С таким подходом мы можно поддерживать экземпляры гораздо большего количества.

Сервис для взаимодействия с оператором обработки данных

Как уже было описано в предыдущих главах, до начала работы всей системы оператор обработки данных задает шаблон цепочки обработки. Делает он это с помощью Common Workflow Language (CWL). CWL — это открытый стандарт для описания того, как запускать инструменты командной строки и подключать их для создания рабочих процессов.

Инструменты и цепочки обработки, описанные с использованием CWL, переносимы на различные платформы, поддерживающие стандарты CWL.

CWL предоставляет решение для описания переносимых, многократно используемых цепочек обработки, а также запуска этих процессов с помощью командной строки. [3]

В WfMS используется только формат описания цепочек обработки, т.к. инструменты для запуска процессов не удовлетворяют необходимым условиям SPD Online Filter.

Интеграция открытых стандартов в данную систему позволит обеспечить ее долговечность. При использовании CWL WfMS может взаимодействовать с другими системами, которые также следуют этим стандартам. Использование открытых стандартов помогает избавится от разработки и поддержания своих, а также избежать возможного возникновения ошибок при разработке своих стандартов.

В Common Workflow Language для описание цепочек используется формат YAML.

Пример вложенных объектов, описанный с помощью CWL:

```
cwlVersion: v1.0
class: CommandLineTool
baseCommand: echo
inputs: # this key has an object value
  example_flag: # so does this one
    type: boolean
    inputBinding: # and this one too
      position: 1
      prefix: -f
```

Приведенный выше YAML иллюстрирует, как можно относительно быстро создавать описания сложных вложенных объектов. Карта inputs содержит один ключ, example_flag, который сам содержит два ключа, type и inputBinding, а один из этих дочерних элементов, inputBinding, содержит еще две пары ключ-значение (позиция и префикс). Графическое представление объекта ввода, который он описывает Рис. 5:

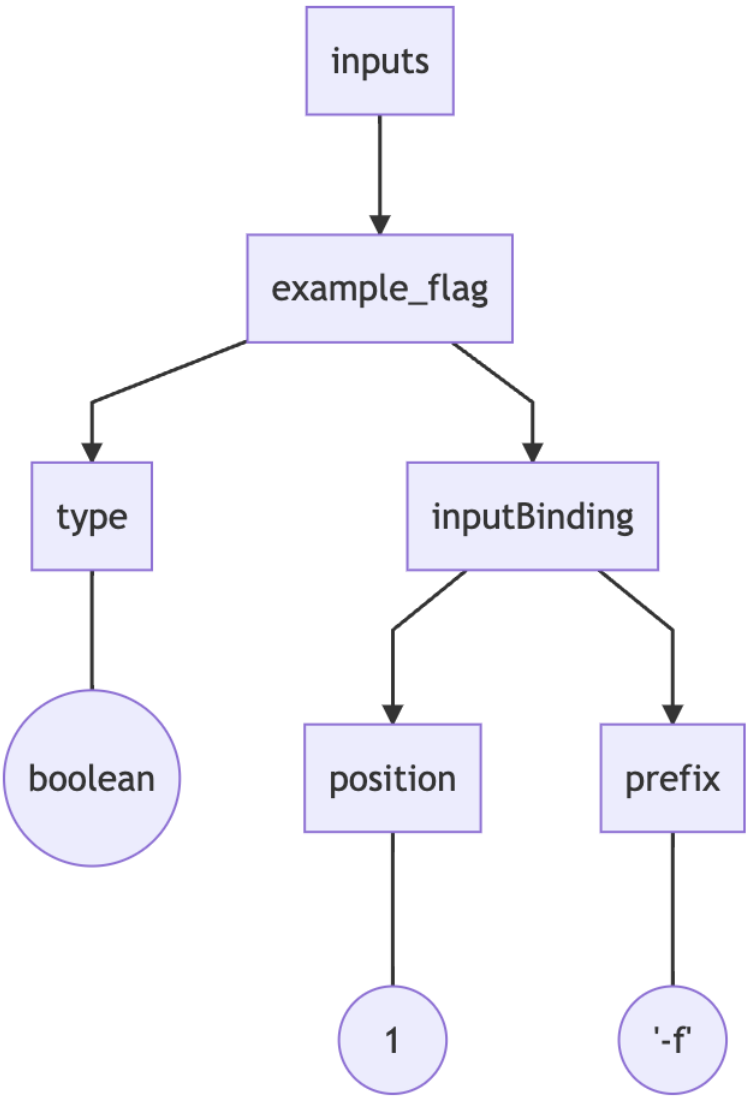

Рисунок 5. Графическое представление объекта ввода в CWL

Главное для WfMS это модуль workflow. Workflow — это модуль обработки CWL, который описывает (а также исполняет инструменты командной строки, но не в нашем случае) цепочки обработки в виде шагов. Он должен иметь входы, выходы и шаги, определенные в документе CWL. На рис. 6 представлена схема описания workflow:

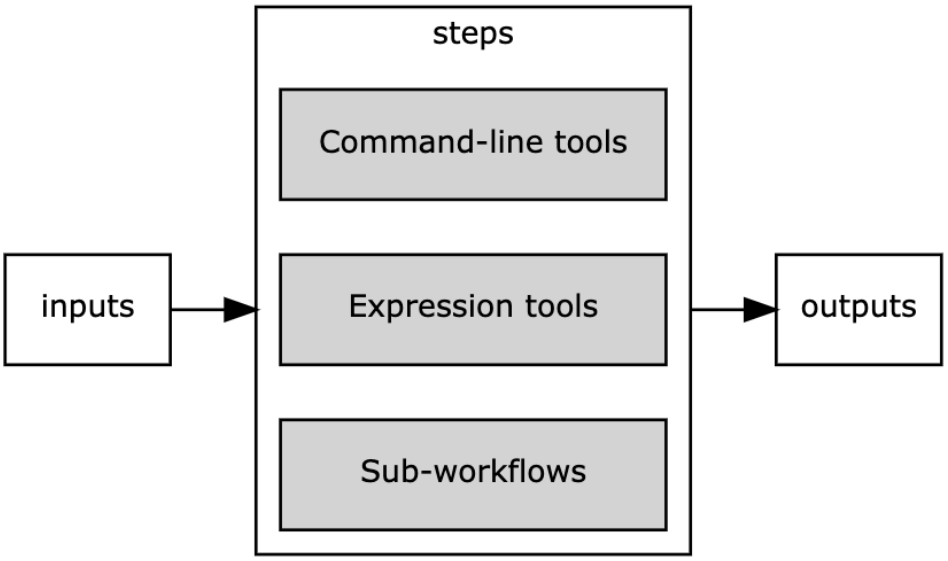

Рисунок 6. CWL workflow

Более подробно о разных модулях можно посмотреть в [3].

Также данный сервис позволяет смотреть выводить список заданий, активных и завершенных. По активным занятиям оператор обработки данных может посмотреть статистику по количеству обработанных файлов, количеству ошибок. Также оператор обработки данных может отменить выполнение некоторого задания. Данный сервис написан на Flask, а на на FastAPI, т.к. для него не требуется высокая скорость и асинхронность.

Api для взаимодействия с данным сервисом:

Таблица 4. Api для взаимодействия с первым сервисом

Запрос	**Тип запроса**	**Input 1**	**Input 2**	**Output**	**Примечания**
Добавление нового шаблона	Rest api POST	string	name template	response status	В параметре string находится содержимое CWL

Получение списка шаблонов	Rest api GET	—	—	{
				● name
				● string
				}
					—

Удаление шаблона	Rest api DELETE	name	—	response status	—
Получение списка заданий	Rest api GET	—	—	[{ ● task_id, ● status }, …]	status в ответе содержит информацию о том, активно ли задание
Отмена задания	Rest api POST	● task_id	—	response status	—

Шаблоны цепочек обработки данных хранятся в базе данных postgresql. После того, как оператор обработки данных описывает и отправляет шаблон цепочки, он конвертируется в json.

В postgresql есть специальный тип JSON. Он предназначены для хранения данных JSON согласно стандарту RFC 4627. Такие данные можно хранить и в типе text, но типы JSON лучше тем, что проверяют, соответствует ли вводимое значение формату JSON. Для работы с ним есть также несколько специальных функций и операторов. [11]

Существуют два типа данных JSON: json и jsonb. Они принимают на вход почти одинаковые наборы значений, но основное их отличие в эффективности. Тип json сохраняет точную копию введённого текста, которую функции обработки должны разбирать заново при каждом выполнении, тогда как данные jsonb сохраняются в разобранном двоичном формате, что несколько замедляет ввод из-за преобразования, но значительно ускоряет обработку, не требуя многократного разбора текста. Кроме того, jsonb поддерживает индексацию, что тоже может быть очень полезно.

Так как тип json сохраняет точную копию введённого текста, он сохраняет семантически незначащие пробелы между элементами, а также порядок ключей в JSON-объектах. И если JSON-объект внутри содержит повторяющиеся ключи, этот тип сохранит все пары ключ/значение. (Функции обработки будут считать действительной последнюю пару.) Тип jsonb, напротив, не сохраняет пробелы, порядок ключей и значения с дублирующимися ключами. Если во входных данных оказываются дублирующиеся ключи, сохраняется только последнее значение.

Для большинства приложений предпочтительнее хранить данные JSON в типе jsonb (если нет особых противопоказаний, например важны прежние предположения о порядке ключей объектов).

PostgreSQL позволяет использовать только одну кодировку символов в базе данных, поэтому данные JSON не будут полностью соответствовать спецификации, если кодировка базы данных не UTF-8. При этом нельзя будет вставить символы, непредставимые в кодировке сервера, и наоборот, допустимыми будут символы, представимые в кодировке сервера, но не в UTF-8. [11]

Исходя из этого, шаблон цепочек обработки хранится в postgresql с типом jsonb.

Далее представлены поля базы данных, для хранения цепочек:

Таблица 5. БД для хранения шаблонов цепочек

Поле	**Описание**	**Тип**
Id	Уникальный id	integer
name	Имя шаблона	string
template	json, в котором хранится сам шаблон, преобразованный из yaml	jsonb

Сервис для опроса DMS

Для того чтобы получать информацию о новых данных, WfMS периодически опрашивает DMS на предмет появления новых данных. Параметр, который отвечает за то, с какой частотой будет происходить запрос на новые данные указывается при запуске программы в качестве параметра. При всем этом WfMS должен хранить значение времени последнего полученного датасета. Это нужно для того, чтобы система управления данными понимала какие датасеты передавать и не отдавать информацию об одних и тех же датасетов несколько раз.

При самом первом опросе, timestamp не передается и DMS возвращает информацию о всех зарегистрированных датасетах. В полученном json, у каждого датасета есть параметр timestamp регистрации. WfMS ищет самый поздний timestamp и сохраняет его. И при каждом следующем запросе отправляет его в качестве параметра и обновляет при необходимости.

Чтобы не потерять этот параметр в случае перезагрузки системы или в случае сбоя, он сериализуется.

Сериализация — процесс перевода структуры данных в битовую последовательность. Обратной к операции сериализации является операция

десериализации — создание структуры данных из битовой последовательности. [12]

Это операция позволяет быстро сохранять в долгую память и считывать некоторые объекты. В данном случае это очень удобно, так как работа с базой данных будет медленнее и затратнее.

Python предлагает модуль pickle, который реализует двоичные протоколы для сериализации и десериализации объекта Python. Идея состоит в том, чтобы использовать pickle.dump(obj, file) функция, которая преобразует объект Python obj в поток байтов, а затем записывает его в файловый объект file. Чтобы сделать обратное, то есть преобразовать поток байтов из двоичного файла обратно в объект Python, нужно использ метод pickle.load() функция [13].

После получения информации о новых датасетах, данные сервис сопоставляет его с уже готовым шаблоном цепочки обработки и получает необходимые параметры тем самым получая конкретную цепочку обработки. После этого, данный сервис отправляет информацию о датасете и цепочке с конкретными параметрами в сервис для генерации заданий с помощью Rest API.

Сервис для генерации заданий

Данный сервис необходим для генерации заданий и передачи их в систему управления нагрузкой.

После получения необходимых данных из от сервиса, описанного в прошлом пункте, данный сервис записывает цепочку в следующем виде Табл. 6:

Таблица 6. БД для хранения цепочек

Поле	**Описание**	**Тип**
Id	Уникальный id	integer
id_template	Поле, связанное с	integer

	полем id БД для хранения шаблонов цепочек	
params	Параметры для шагов обработки	jsonb

Данные о датасетах хранятся в следующем виде Табл. 7:

Таблица 7. БД для хранения информации о датасетах

Поле	**Описание**	**Тип**
uid	Уникальный id датасета	integer
type	logs or normal	string
name	Уникальное имя датасета	string
timestamp	Время регистрации в DMS	datetime
meta	Мета информация	jsonb
count_files	Кол-ва файлов в датасете	integer
id_chain	Связан с полем id БД для хранения цепочек	integer
step	Текущий шаг обработки(у первого датасета None)	string
status	in_progress/ready/finishe	string

	d(у первого датасета finished)	

После записи в таблицу, данный сервис находит следующий шаг для обработки. Если данного шага нет он обновляет статус на finished и создает заявку на выгрузку данных в DMS. Если есть следующий шаг, то WfMS получает информацию о хранилище, регистрирует в DMS выходной датасет, также заносит его в бд (аналогично для логов) и меняет поле status на in_progress. Получая всю необходимую информацию он создает новое задание, отправляет его в WMS с помощью RabbitMQ. Также он отправляет эти данные в сервис для опроса WMS с помощью Rest API, который периодически опрашивает статус задания.

Api для других систем:

Таблица 8. API сервиса для генерации заданий

Запрос	**Тип запроса**	**Input 1**	**Output**	**Примечания**

Добавление нового датасета	Rest api Post	body:
		{
		info about dataset*
		}
			response status	Имеет те же ключи, что и в таблице 7. Однако ключи в поле meta раскрываются следующим образом:
				meta_key1: value,
				…

Уведомление о завершении задания	Rest api Post	{
		uid_prev: “uid_prev”,
		uid: “uid”
		}
			response status	—

Получить список заданий	Rest api Get	—	[{ uid: “uid” }, …]	—

Сервис для опроса WMS и удаления промежуточных данных

Данный сервис нужен для того, чтобы следить за выполнением задания. Он получает из очереди задание, записывает в свою бд.

Описание базы данных заданий представлена в таблице 9:

Таблица 9. БД для хранения заданий

Поле	Описание	Тип
uid	уникальный идентификатор задания	integer
executable	исполняемый файл для запуска	string

args	флаги для исполняемого файла задания	string
rank	Для всех новых заданий по дефолту приоритет равен 1.	integer
device_type	Определение выбранного вычислительного ресурса, который будет использоваться для выполнения задания ● CPU ● CPU/GPU	string
mode	указание, какой тип задания будет выполняться ● Map ● Merge	string
dat_in_uid	уникальный идентификатор входного датасета	integer
retry	Определение максимального числа попыток выполнения для задач, связанных с этим заданием, в случае	integer

	неудачного выполнения.	
dat_out_uid	уникальный идентификатор выходного датасета	integer
dat_stor_url	унифицированный указатель ресурса выходного хранилища	string
dat_out_url	унифицированный указатель ресурса выходного датасета	string
log_out_uid	уникальный идентификатор выходного датасета логов	integer
log_stor_url	унифицированный указатель ресурса выходного хранилища для хранения логов	sting
log_out_url	унифицированный указатель ресурса выходного датасета логов	string
status	in_progress/finished/canceled	string

После этого, данный сервис периодически опрашивает WMS завершилось ли задание. Как только задание завершается, данный сервис меняет в БД для хранения заданий статус на finished и отправляет информацию сервису для генерации заданий. После этого данный сервис передает некоторую информацию в DMS для очистки промежуточных данных, а именно передает поле uid (и для обычных датасетов и для логов). Эту информацию он передает с помощью RabbitMQ. [14]. Также данные сервис передает информацию (uid входного датасета и uid выходного) в сервис для генерации задании. По этим данным, сервис для генерации заданий меняет статусы датасетам на finished и ready соответственно и начинает генерировать задание для последнего.

Описанные в таблице 9 поля для каждого задания приходят по Rest API методом POST.

Выбор технологий

Выбор языка программирования

На данный момент, весь программный стек SPD основан на python и c++. Учитывая, что вычисления производятся не в самой системе, то для построения микросервисной архитектуры целесообразно выбрать python.

Выбор асинхронного фреймворка

Для всех сервисов, которые не взаимодействуют с оператором обработки данных используется асинхронный фреймворк для python FastAPI.

FastAPI — это современный асинхронный фреймворк для создания веб-приложений и REST API на основе Python. Вот основные преимущества из-за которых был выбран FastAPI:

- Высокая производительность: FastAPI использует особенности асинхронного программирования и механизмы компиляции для

достижения высокой производительности. FastAPI способен обрабатывать большое количество запросов с высокой скоростью.

- Поддержка асинхронности: FastAPI предоставляет полную поддержку асинхронного программирования. Он позволяет создавать асинхронные функции-обработчики запросов, использовать асинхронные операции базы данных и вызывать асинхронные зависимости. Это позволяет эффективно использовать системные ресурсы и обеспечивает хорошую отзывчивость приложения

- Интеграция с другими инструментами: FastAPI легко интегрируется с другими инструментами и библиотеками Python. Он поддерживает широкий спектр баз данных (SQL и NoSQL), аутентификацию и авторизацию, асинхронные клиенты HTTP, логирование и многое другое. FastAPI также может быть использован с множеством инструментов развертывания и контейнеризации, таких как Docker и Kubernetes

Выбор брокера сообщений

Брокеры сообщений представляют собой программные модули, которые обеспечивают взаимодействие и передачу информации между приложениями, сервисами и системами. Они выполняют преобразование сообщений из различных формальных протоколов обмена, что позволяет связанным службам обмениваться данными напрямую, независимо от языка программирования или платформы, на которой они функционируют.

Брокеры сообщений отправляют сообщения, определяют их маршрут, хранят и доставляют сообщения конечным получателям. Они действуют как посредники между приложениями, обеспечивая возможность отправителям отправлять сообщения, не имея точной информации о количестве и местонахождении потребителей.

Благодаря использованию брокеров сообщений, создается гибкая и масштабируемая архитектура, где различные компоненты системы могут взаимодействовать асинхронно, обмениваясь сообщениями через брокер. Брокеры сообщений широко применяются для реализации распределенных систем, микросервисных архитектур, обработки событий и других сценариев, где требуется надежная и гибкая передача сообщений между компонентами системы.

Рассмотрим два популярных инструмента — RabbitMQ и Apache Kafka.

RabbitMQ

RabbitMQ представляет собой распределенный брокер сообщений с открытым исходным кодом, предназначенный для эффективной доставки сообщений в сложных сценариях маршрутизации. Этот инструмент называется "распределенным", поскольку обычно функционирует как кластер узлов, где очереди распределяются (реплицируются) по узлам, чтобы обеспечить высокую доступность и устойчивость к сбоям.

В RabbitMQ применяется push модель: это помогает предотвратить перегрузку пользователей через ограничение предварительной выборки, настроенное консьюмером. Идеальный подход для обмена сообщениями с малой задержкой, который отлично работает с архитектурой RabbitMQ на основе очередей. Этот инструмент можно представить как почтовое отделение, которое получает, хранит и отправляет почту; только RabbitMQ принимает, хранит и передает сообщения с двоичными данными. [15]

В упрощенном виде принцип работы RabbitMQ рис.7: [16]

- Отправители (publishers) отправляют сообщения на обменники (exchange);
- обменники отправляют сообщения в очереди и в другие обменники;
- при получении сообщения RabbitMQ отправляет подтверждения отправителям;

- получатели (consumers) поддерживают постоянные TCP-соединения с RabbitMQ и объявляют, какую очередь они получают;
- RabbitMQ проталкивает (push) сообщения получателям;
- получатели отправляют подтверждения успеха или ошибки получения сообщения;
- после успешного получения сообщение удаляется из очереди.

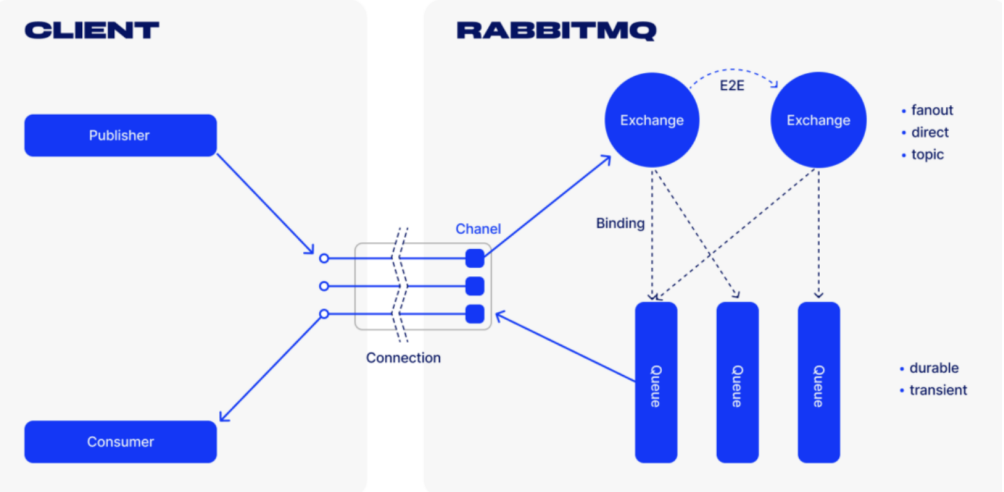

Рисунок 7. Работа RabbitMQ

Apache Kafka

Apache Kafka — это открытая распределенная платформа потоковой передачи событий, которая обеспечивает высокую пропускную способность. Написанная на Java и Scala, Kafka представляет собой систему шины сообщений Pub/Sub, ориентированную на потоки и воспроизведение данных с высокой интенсивностью. В отличие от традиционных очередей, Kafka не полагается на хранение сообщений в очереди, а добавляет их в журнал и сохраняет до достижения предела хранения или пока получатель не прочтет эти сообщения.

В Кафке применяется подход на основе pull модели, позволяющий пользователям запрашивать пакеты сообщений с определенных оффсетов.

Пользователи могут использовать пакетную обработку сообщений для повышения пропускной способности и эффективной доставки сообщений. [15]

Общие рекомендации в каких случаях что использовать описанные в [17]:

Используйте Kafka для:

- хранения, чтения и повторного чтения потоковых данных;
- обработки и анализа данных в режиме реального времени.

RabbitMQ используют для обработки высокопроизводительных фоновых заданий. Также его выбирают для сложной маршрутизации и интеграции нескольких приложений и сервисов с нетривиальной логикой.

Используйте RabbitMQ для:

- длительных задач;
- высокопроизводительных фоновых заданий;
- интеграции между приложениями и внутри них.

Исходя из вышеописанного, на данном этапе был выбран RabbitMQ, однако в дальнейшем, он может быть заменен на Kafka.

Глава 6. Итоговая архитектура Workflow Management System

Резюмируя предыдущие главы, была спроектирована следующая архитектура Рис. 8:

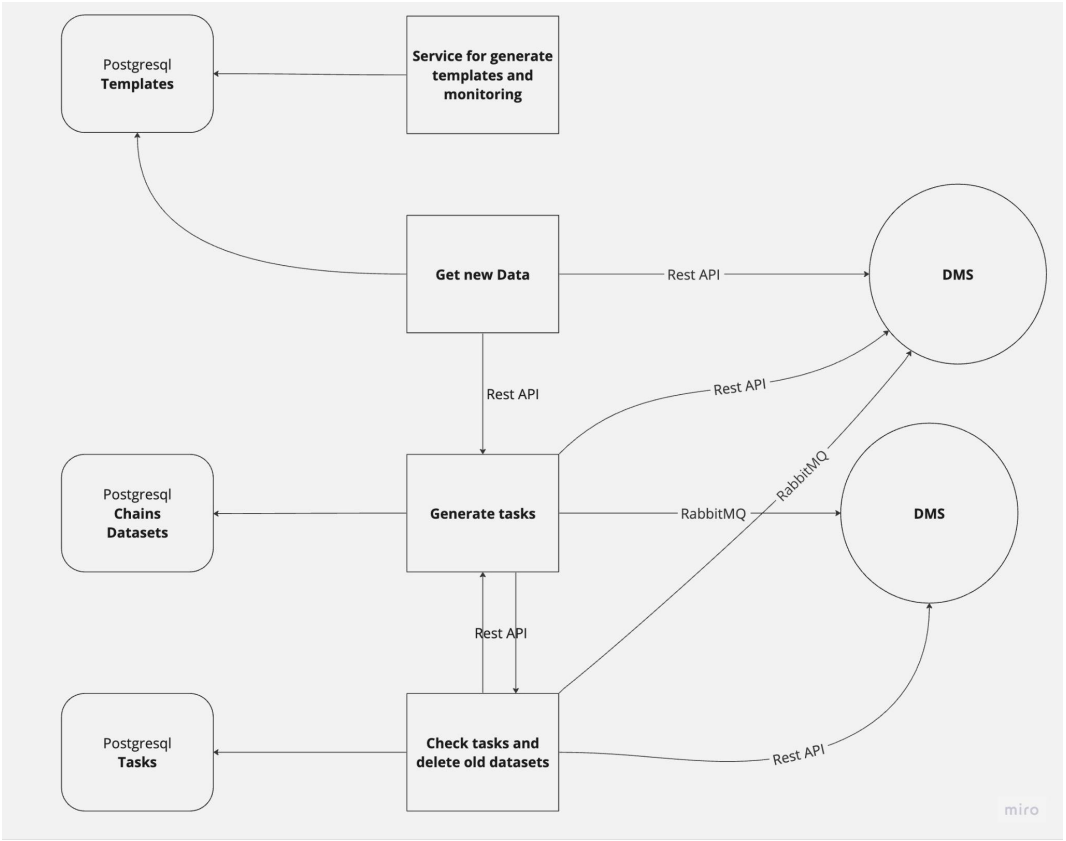

Рисунок 8. Архитектура WfMS

Вся инфраструктура развертывается с использованием docker-compose [18], что позволяет независимо работать с различными сервисами без ограничений по выбору инфраструктуры.

Каждый сервис имеет свою собственную базу данных, к которой другие сервисы не имеют прямого доступа. Это предотвращает возникновение узких мест, поскольку при одновременной работе нескольких сервисов с одной и той же таблицей возникают задержки.

Более того, данная архитектура обеспечивает возможность горизонтального масштабирования. Наибольшая нагрузка приходится на сервис генерации заданий, поэтому количество его экземпляров может быть увеличено для обработки большего объема работы.

Выводы

Проведен анализ существующих решений, определена общая архитектура SPD Online Filter, и в частности, Workflow Management System. Были разграничены зоны ответственности для каждой подсистемы. Был выделен требуемый функционал для WfMS. Согласовано взаимодействие с другими системами. Определены достаточный уровень функциональных требований, спроектирована архитектура WfMS.

Заключение

В результате получен сервис, позволяющий задавать шаблоны цепочек обработки данных, запускать в параллельном режиме цепочки обработки данных, а также отслеживать их состояние. Разрабатываемая система управления процессом обработки данных является важной компонентой всей системы SPD Online Filter и требует дальнейшей доработки.

В дальнейшем планируется доработка данной системы.

Список литературы

[1] Эксперимент SPD [Электронный ресурс] // URL: http://spd.jinr.ru/wp-content/uploads/2021/04/SPD_Korzenev_DIS2021.pdf

[2] NICA [Электронный ресурс] // URL: https://nica.jinr.ru/ru/

[3] CWL [Электронный ресурс] // URL: https://www.commonwl.org

[4] V.M. Abazov, V. Abramov, L.G. Afanasyev, «Conceptual design of the Spin Physics Detector» [Электронный ресурс] // URL: https://arxiv.org/abs/2102.00442

[5] F Barreiro, M. Borodin, K. De, D. Golubkov, A. Klimentov, T. Maeno, R. Mashinistov, S.Padolski, T. Wenaus, «ATLAS production system», 2016

[6] G. Aad, E. Abat, J. Abdallah, A. Abdelalim, A. Abdesselam, O. Abdinov, B. Abi, M. Abolins, H. Abramowicz. «The ATLAS experiment at the CERN large hadron collider Journal of Instrumentation», vol. 3, no. 08, p. S08003, 2008

[7] J. Phys. Conf. Ser. 664 062035, «The future of PanDA in ATLAS distributed computing», 2015

[8] Hadron Collider J, «The ATLAS Experiment at the CERN» Inst. 3, S08003, 2008

[9] Микросервисная архитектура [Электронный ресурс] // https://www.atlassian.com/ru/microservices/microservices-architecture

[10] Apache Airflow [Электронный ресурс] // https://airflow.apache.org/

[11] Тип JSON в Postgresql [Электронный ресурс] // https://postgrespro.ru/docs/postgresql/9.4/datatype-json

[12] Сериализация [Электронный ресурс] // https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B8%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F

[13] Сериализация и десериализация объекта Python [Электронный ресурс] // https://www.techiedelight.com/ru/serialize-deserialize-python-object/

[14] RabbitMQ [Электронный ресурс] // https://www.rabbitmq.com/

[15] Сравнение Apache Kafka и RabbitMQ [Электронный ресурс] // https://habr.com/ru/companies/southbridge/articles/666326/
[16] Сравнение Apache Kafka и RabbitMQ [Электронный ресурс] // https://tproger.ru/articles/pochemu-my-ispolzuem-kafka-vmesto-rabbitmq-sravnenie-i-preimushhestva/
[17] Сравнение Apache Kafka и RabbitMQ [Электронный ресурс] // https://slurm.io/tpost/phdmogo9y1-rabbitmq-i-apache-kafka-chto-vibrat-i-mo
[18] Docker-compose [Электронный ресурс] // https://docs.docker.com/compose/
[19] Основные сущности AirFlow [Электронный ресурс] // https://mcs.mail.ru/blog/airflow-what-it-is-how-it-works