ООО “ИКСИМИ”

Описание технической архитектуры программного обеспечения «Открытый квотировано-кластерный оркестратор ИИ (ОКОИИ) на базе архитектуры PBDR "OQOAI:PBDR 5.2"»

г. Зеленодольск 2026

Наименование программного обеспечения: «Открытый квотировано-кластерный оркестратор ИИ (ОКОИИ) на базе архитектуры PBDR "OQOAI:PBDR 5.2"»

Данный документ описывает базовые принципы и подходы к реализации различных компонентов ПО «Открытый квотировано-кластерный оркестратор ИИ (ОКОИИ) на базе архитектуры PBDR "OQOAI:PBDR 5.2"»

1. Общая характеристика архитектуры

Программное обеспечение «Открытый квотировано-кластерный оркестратор ИИ (ОКОИИ) на базе архитектуры PBDR "OQOAI:PBDR 5.2"» представляет собой модульную клиент-серверную информационную систему, реализованную на основе архитектуры передачи запросов. Архитектура построена с использованием принципов модульного проектирования и ориентирована на простоту сопровождения, прозрачность обработки данных и возможность быстрого развертывания как в локальной, так и в защищенной корпоративной инфраструктуре.

Основной функциональной особенностью системы является реализация архитектурного подхода распределеной маршрутизации PBDR подробно описаного в публикации http://doi.org/10.17513/doi.26 , сочетающего методы, где каждый клиент независимо вычисляет вектор стоимости и применяет вектор весов политики через скалярное произведение для выбора оптимального вычислительного узла

При проектировании системы были приняты следующие архитектурные решения:

- Децентрализация: Отсутствие центрального планировщика, балансировщика нагрузки или очереди. Это устраняет единые точки отказа и узкие места производительности.
- Управляемые клиентом решения: Логика маршрутизации находится в клиенте, который вычисляет стоимость для каждого узла на основе его текущего состояния.
- Управление на основе политик: Логика маршрутизации не фиксирована; она определяется настраиваемым вектором политики. Это позволяет различным отделам (например, поддержка и инженерия) или приложениям использовать разные стратегии маршрутизации.
- Модульность: Система спроектирована как расширяемая и совместимая с любым ИИ-LLM сервером, совместимым с OpenAI (например, vLLM, Ollama, llama.cpp).
- Не сохраняющие состояние узлы: Вычислительные узлы не сохраняют состояние с точки зрения маршрутизации, что упрощает их управление, замену и масштабирование.

2. Структурные элементы системы и их интерфейсы

2.1. Уровень представления (Presentation Layer)

Уровень представления реализует взаимодействие пользователя и администратора с системой через специализированные интерфейсы.

Основные компоненты:

1. Клиентский API-шлюз (Client API Gateway) Обеспечивает взаимодействие внешних приложений и пользователей с системой для выполнения задач инференса (генерации) LLM.

Основные функции:

- прием запросов на генерацию через OpenAI-совместимый REST API (эндпоинты `/v1/chat/completions`, `/v1/models`);
- потоковое отображение генерируемых ответов (Server-Sent Events / WebSocket);
- аутентификация и авторизация входящих запросов (API-ключи);
- поддержка пользовательских сессий.

Интерфейсы взаимодействия:

- HTTP/REST API (OpenAI-совместимый);
- WebSocket (для потоковой передачи ответов языковой модели).

2. Панель администратора (Admin Web UI) Веб-интерфейс для централизованного управления кластером.

Основные функции:

- Обнаружение и мониторинг доступа узлов
- Администрирование клиентских и серверных узлов
- Мониторинг и сбор метрик аппаратных ресурсов узлов
- Управление расширенной конфигурацией серверов и клиентов

Интерфейсы взаимодействия:

- HTTP/REST API (внутренний Admin API);
- WebSocket (для обновления метрик в реальном времени).

Стек реализации:

- HTML, CSS, JavaScript

2.2. Уровень приложения (Application Layer)

Уровень приложения реализует основную бизнес-логику системы и состоит из трех ключевых компонентов: **PBDR Client**, **PBDR Server** и **PBDR Admin**, каждый из которых содержит свои функциональные модули.

2.2.1. Компонент PBDR Client (маршрутизация)

Модуль, устанавливаемый на рабочих станциях пользователей, принимает запросы и выполняет децентрализованную маршрутизацию.

В его состав входят следующие функциональные части:

1. Приема запросов (Request Handler)

Назначение:

- прием запросов по OpenAI-совместимому API;
- извлечение параметров запроса (модель, промпт, температура, `max_tokens`);
- парсинг и валидация входящих данных.

2. Обнаружения узлов (Node Discovery)

Назначение:

- опрос серверных узлов через Monitor API;
- сбор актуальных метрик состояния;
- использование версионного протокола для минимизации трафика.

Используемые технологии:

- `aiohttp` для асинхронных HTTP-запросов;
- Версионный протокол.

3. Модуль расчета стоимости и выбора узла (Policy-Driven Optimizer)

Назначение:

- построение вектора стоимости (Cost Vector) из 10 компонентов;
- вычисление скалярной стоимости как скалярного произведения вектора политики и вектора стоимости;
- применение политик (весовых коэффициентов);
- реализация исследования-эксплуатации (Exploration-Exploitation) с UCB-бонусом.

4. Модуль проксирования запросов (Request Proxy)

Назначение:

- отправка запроса на выбранный LLM-сервер;
- обработка потокового (stream) и обычного режимов;
- возврат ответа пользователю.

5. Модуль администрирования и конфигурации (Client Config)

Назначение:

- чтение и применение локальной конфигурации;
- прием обновлений конфигурации от Admin-компонента;
- управление текущей политикой.

2.2.2. Компонент PBDR Server

Модуль, устанавливаемый на вычислительных узлах (серверах\ПК с GPU).

В его состав входят следующие функциональные части:

1. Мониторинга оборудования (Hardware Monitor) *Назначение:*

- сбор метрик GPU (модель, загрузка, VRAM, температура, потребляемая мощность);
- сбор метрик CPU и RAM;
- поддержка NVIDIA и AMD

2. Взаимодействия с LLM (LLM Interface)

Назначение:

- обнаружение доступных моделей (через `/api/tags` для Ollama или `/v1/models` для llama.cpp);
- отслеживание загруженной модели;
- проксирование запросов к LLM-серверу;
- поддержка OpenAI-совместимого API.

3. Публикации состояния (Monitor API)

Назначение:

- предоставление состояния узла по запросу (`GET /status`);
- поддержка версионного протокола (заголовки `If-Version` и `X-State-Version`);

4. Модуль управления конфигурацией (Server Config)

Назначение:

- управление настройками узла (прием новых задач, максимальная очередь);
- прием и применение обновлений конфигурации от Admin-компонента.

2.2.3. Компонент PBDR Admin (Модуль администрирования)

Модуль для централизованного управления кластером.

В его состав входят следующие функциональные модули:

1. Модуль управления устройствами (Device Manager) *Назначение:*

- получение и отображение списка всех устройств (клиентов и серверов);
- проверка статуса и доступности устройств;
- добавление/удаление устройств

2.3. Уровень данных (Data Layer)

Уровень данных обеспечивает хранение и управление информацией о состоянии системы.

Основные компоненты:

1. Конфигурационные данные

- Локальные файлы конфигурации (формат JSON);
- Хранятся на каждом узле и обновляются через Admin API.

2. Состояние кластера (временное)

- Хранится в оперативной памяти каждого клиентского узла (кеш состояния);
- Периодически обновляется через Monitor API.

3. Политики маршрутизации

- Хранятся в конфигурационных файлах клиентов;
- Управляются через Admin UI.

Общие интерфейсы взаимодействия

Интерфейс	**Используется в**	**Назначение**
OpenAI API	PBDR Client (вход), PBDR Server (выход к LLM)	Прием запросов на генерацию, взаимодействие с LLM

Интерфейс	Используется в	Назначение
Monitor API	PBDR Client → PBDR Server	Получение состояния узлов (метрик)
Admin API	PBDR Admin → PBDR Client/Server	Управление конфигурацией, получение статуса
WebSocket	PBDR Client → Пользователь, PBDR Admin → Администратор	Потоковая передача данных в реальном времени
NVML/ROCm API	PBDR Server → Оборудование	Сбор метрик GPU

3. Взаимодействие компонентов системы

Взаимодействие между уровнями и компонентами PBDR осуществляется следующим образом:

1. Пользователь / Приложение формирует запрос на генерацию (инференс) через OpenAI-совместимый API.

- Пользователь или внешнее приложение (например, Open WebUI) отправляет HTTP-запрос на **PBDR Client** (Уровень представления).
- Запрос содержит параметры: модель, список сообщений (промпт), температуру, `max_tokens` и флаг `stream`.

2. PBDR Client выполняет децентрализованную маршрутизацию запроса.

- **Обнаружение узлов:** Клиент опрашивает все доступные **PBDR Server** узлы через **Monitor API** (`GET /status`), собирая их актуальное состояние (метрики).
- **Расчет стоимости:** Для каждого узла клиент строит **Cost Vector** (вектор стоимости) из 10+ параметров: время ожидания, загрузка GPU/CPU, использование VRAM, температура, штраф за "холодный старт" модели, глубина очереди и др.
- **Применение политики:** Клиент применяет текущую **политику маршрутизации** (весовой вектор), вычисляя скалярную стоимость для каждого узла как скалярное произведение вектора политики и вектора стоимости
- **Выбор узла:** Выбирается узел с минимальной стоимостью

3. PBDR Client проксирует запрос на выбранный PBDR Server.

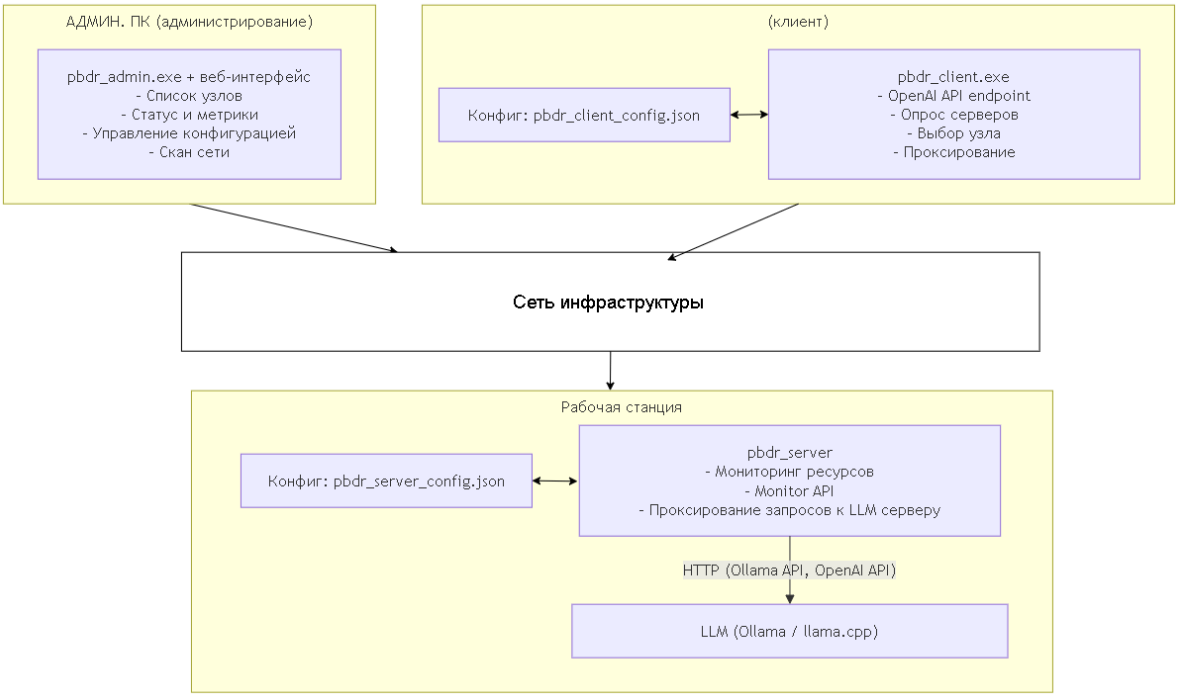
- Клиент отправляет оригинальный запрос к **LLM-серверу** (Ollama, llama.cpp и т.п.) через выбранный **PBDR Server**.
- PBDR Server принимает запрос, при необходимости преобразует его в формат, ожидаемый локальным LLM-сервером, и передает на выполнение.

4. PBDR Server получает ответ от LLM и возвращает его клиенту.

- LLM-сервер генерирует ответ (в обычном или стриминговом режиме).
- PBDR Server возвращает ответ **PBDR Client**.
- PBDR Client передает ответ обратно пользователю/приложению в формате OpenAI API.

5. Администратор управляет кластером через Admin Web UI.

- Администратор через веб-интерфейс **PBDR Admin** отправляет команды на **PBDR Client** и **PBDR Server** через **Admin API**.
- Администратор может просматривать состояние всех узлов, изменять конфигурации и политики



Пайплайн обработки запроса на генерацию (Inference Pipeline)

Пайплайн обработки запроса на генерацию включает следующие независимые стадии, которые могут заменяться без изменения архитектуры остальных компонентов

1. **Request (Прием запроса)** — получение запроса через OpenAI API, валидация параметров.
2. **Discovery (Обнаружение)** — опрос PBDR Server узлов, сбор метрик состояния.
3. **CostVector** — построение Cost Vector (10+ параметров) для каждого узла.
4. **Policy Application (Применение политик)** — вычисление скалярной стоимости с учетом весов политики.
5. **Routing (Маршрутизация)** — выбор оптимального узла и отправка запроса.
6. **Generation (Генерация)** — выполнение запроса на LLM-сервере и получение ответа.
7. **Response (Возврат ответа)** — возврат сгенерированного ответа пользователю в формате OpenAI API.

4. Архитектурный стиль

Основу архитектуры PBDR составляют следующие архитектурные подходы:

- **Слои архитектуры (трёхуровневая архитектура):**
  - **Уровень представления** — Клиентский API-шлюз и Admin Web UI.
  - **Уровень приложения** — PBDR Client, PBDR Server, PBDR Admin (каждый содержит свои функциональные модули).
  - **Уровень данных** — локальные файлы конфигурации, политики
- **Модули архитектуры (модульная организация компонентов):**
  - Каждый компонент (Client, Server, Admin) состоит из внутренних функциональных модулей
- **Пайплайны (конвейер обработки запросов):**
  - Обработка запроса на генерацию построена как последовательный пайплайн (Request → Discovery → CostVector→ Policy Application → Routing → Generation → Response).

При проектировании соблюдаются следующие принципы:

- **Децентрализация:** Отсутствие единой точки отказа (SPOF). Решения о маршрутизации принимаются на стороне клиента, что обеспечивает отказоустойчивость и линейную масштабируемость.
- **Прозрачность всех этапов обработки данных:** Все этапы обработки запроса логируются и могут быть отслежены для отладки и аудита.
- **Минимизация внешних зависимостей и архитектурных абстракций:** Система использует минимальное количество внешних библиотек, что упрощает развертывание и сопровождение.
- **Политико-ориентированное управление:** Гибкая настройка маршрутизации через весовые векторы (политики) позволяет адаптировать систему под различные сценарии использования.
- **Возможность быстрого развёртывания в закрытом контуре организации:** Автономная работа, отсутствие зависимости от внешних сервисов, поддержка Windows и Linux, упаковка в единый исполняемый файл (.exe).

5. Основные технические решения

В системе приняты следующие технические решения.

Компонент	Выбранное решение	Обоснование
Асинхронный HTTP-клиент/сервер	`aiohttp`	Высокая производительность, поддержка асинхронных операций, встроенная поддержка WebSocket.
Сбор системных метрик	`psutil`	Кроссплатформенность (Windows/Linux), легковесность, широкий набор метрик (CPU, RAM, диски).
Конфигурация	JSON-файлы	Человекочитаемый формат, легкость редактирования, поддержка вложенных структур.
Веб-интерфейс администрирования	HTML, CSS, JavaScript	Кроссплатформенность, не требует установки дополнительного ПО на стороне клиента.

Отказ от использования сложных оркестраторов (Kubernetes, Slurm) и тяжелых RAG-фреймворков обеспечивает:

- **Полный контроль** над процессами маршрутизации и обработки данных.
- **Уменьшение количества внешних зависимостей** и упрощение развертывания.
- **Повышение прозрачности** работы системы для администратора.

Микросервисная структура позволяет заменять отдельные компоненты без изменения архитектуры остальных подсистем.

Сторонние компоненты и лицензии

Список использованных сторонних компонентов с указанием их правообладателей и лицензий их распространения

Компонент / Библиотека	**Правообладатель**	**Лицензия**	**Ссылка на лицензию**
aiohttp	aiohttp team	Apache License 2.0	https://www.apache.org/licenses/LICENSE-2.0
psutil	Jay Loden, Dave Daeschler, Giampaolo Rodola'	BSD 3-Clause License	https://github.com/giampaolo/psutil/blob/master/LICENSE
json	Python Software Foundation	Python Software Foundation License	https://docs.python.org/3/license.html
asyncio	Python Software Foundation	Python Software Foundation License	https://docs.python.org/3/license.html
subprocess	Python Software Foundation	Python Software Foundation License	https://docs.python.org/3/license.html
platform	Python Software Foundation	Python Software Foundation License	https://docs.python.org/3/license.html
time	Python Software Foundation	Python Software Foundation License	https://docs.python.org/3/license.html
logging	Python Software Foundation	Python Software Foundation License	https://docs.python.org/3/license.html

Компонент / Библиотека	Правообладатель	Лицензия	Ссылка на лицензию
threading	Python Software Foundation	Python Software Foundation License	https://docs.python.org/3/license.html
datetime	Python Software Foundation	Python Software Foundation License	https://docs.python.org/3/license.html
hashlib	Python Software Foundation	Python Software Foundation License	https://docs.python.org/3/license.html
socket	Python Software Foundation	Python Software Foundation License	https://docs.python.org/3/license.html
ipaddress	Python Software Foundation	Python Software Foundation License	https://docs.python.org/3/license.html
math	Python Software Foundation	Python Software Foundation License	https://docs.python.org/3/license.html
random	Python Software Foundation	Python Software Foundation License	https://docs.python.org/3/license.html
dataclasses	Python Software Foundation	Python Software Foundation License	https://docs.python.org/3/license.html
typing	Python Software Foundation	Python Software Foundation License	https://docs.python.org/3/license.html

Системные зависимости с указанием их правообладателей и лицензий их распространения

Компонент	**Правообладатель**	**Лицензия**	**Ссылка на лицензию**
Python 3.XX	Python Software Foundation	PSF License Agreement	https://docs.python.org/3.XX/license.html
Ollama	Ollama	MIT License	https://github.com/ollama/ollama/blob/main/LICENSE

6. Требования к аппаратному обеспечению (рекомендованные значения)

- ЦП: x86_64 архитектура, от 2 ядер
- ОЗУ: от 4 ГБ (зависит от размера используемой языковой модели).
- GPU: Nvidia или AMD
- Дисковая подсистема: SSD от 25 ГБ (зависит от размера загруженной языковой модели).
- ОС: Рекомендуется **Astra Linux, ОСнова.**
  Поддержка:
  - Дистрибутив Linux с поддержкой python 3.8+ (Debian, Ubuntu),
  - Windows 10/11

Архитектура программного обеспечения «Открытый квотировано-кластерный оркестратор ИИ (ОКОИИ) на базе архитектуры PBDR "OQOAI:PBDR 5.2"» обеспечивает высокую производительность и эффективность кластера при сохранении простоты сопровождения и прозрачности обработки данных. Реализованный подход соответствует современным принципам построения децентрализованных распределенных систем маршрутизации и позволяет эффективно использовать программное обеспечение как в небольших организациях, так и в крупных корпоративных инфраструктурах с повышенными требованиями к безопасности и локальной обработки данных