Введение

1. Аналитическая часть

1.1. Проектирование общей структуры комплекса

В ходе разработки была составлена схема комплекса. Схема приведена на рис. 1. Функционал каждой компоненты описан в таблице 1.

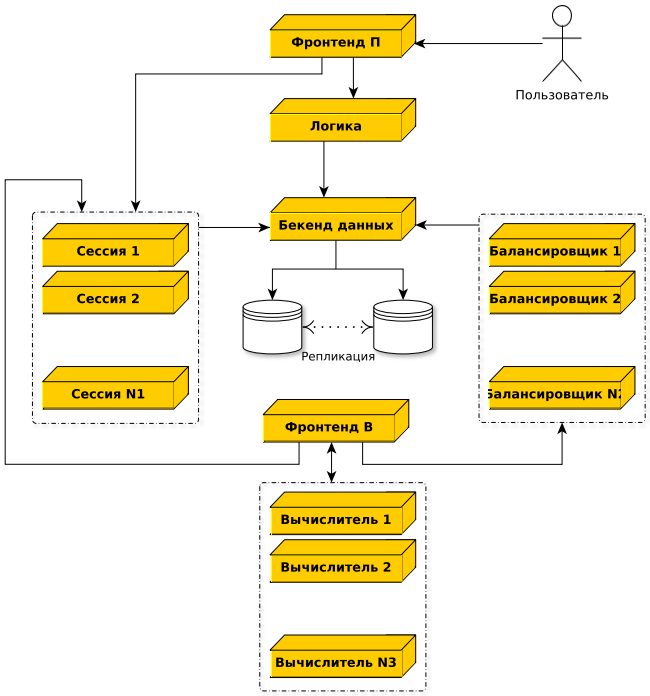

Рис. 1: Схема комплекса

Таблица 1: Описание элементов схемы на рис. 1

Название сервиса	Функционал сервиса
Логика	Реализация пользовательского интерфейса в виде API либо веб-страницы
Бекенд данных	Инкапсуляция доступа к БД
Сессия 1..N	Аутентификация и авторизация пользователей
Балансировщик 1..N	Отслеживание состояния вычислителей, выдача им новых задач, сбор результатов выполнения задач
Фронтенд П,В	Проверка прав пользователей на доступ к предоставляемому API

1.2. Проектирование базы данных

Определение требований к структуре БД

БД должна осуществлять функцию коммуникации между отдельными узлами сети. Это накладывает следующие требования на её структуру:

- Чтобы по возможности уменьшить степень дублирования похожих описаний задач, в отдельную сущность должны быть вынесены “черты” задач (traits). Чертой, к примеру, является требование задачи к вычислителю иметь окружение “Cuda v4.0” или “.net 3.5”.
- Чтобы позволить отслеживать состояние всех подзадач задачи, запущенной с дублированием вычислений, в отдельную сущность должны быть вынесены “подзадачи”, наследующие все атрибуты родительской задачи и хранящие данные, относящиеся непосредственно к ходу вычислений (на каком вычислителе производятся вычисления, результат вычислений и т д).
- Чтобы комплекс имел возможность выбора подходящего вычислительного узла для задачи, вычислительным узлам (Agents) должны соответствовать такие же наборы “черт”, как и диспетчеризуемой в данный момент задаче.

ER-диаграмма

С учётом вышеописанных ограничений на структуру, отношения сущностей в базе данных можно представить в виде ER-диаграммы на рис. 2.

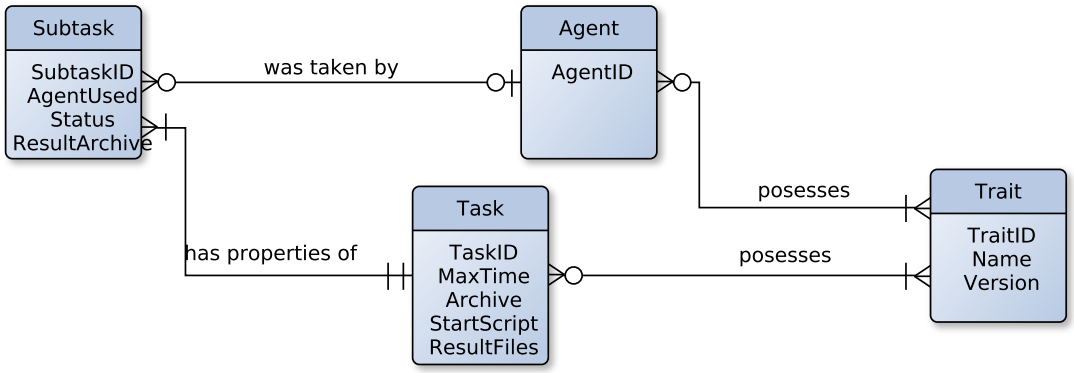

Рис. 2: ER-диаграмма сущностей в базе данных

Таблица 2: Типы полей таблиц

Таблица	Поле	Тип	Описание
Subtask	AgentUsed	UID?	Использованный узел
	Status	Scheduled, In process Terminated, N/A Completed	Статус задачи
	ResultArchive	string?	Имя архива с результатами
Task	Archive	string	Имя архива с файлами задачи
	MaxTime	int	Максимальное время расчёта задачи, в секундах
	StartScript	string	Имя скрипта, запускающего задачу
	ResultFiles	string	Через запятую, имена результирующих файлов
Agent	–	–	–
Trait	Name	string	Имя черты
	Version	string	Версия черты

Таблица полей и типов данных

Атрибуты отдельных сущностей в базе данных хранятся в полях типов, описанных в таблице 2. Каждой сущности соответствует отдельная таблица. Запись вида "T?" в столбце "Тип" означает, что значение необязательно. Обязательные поля-идентификаторы не указаны в таблице.

1.3. Проектирование API отдельных сервисов

В данном разделе описываются API отдельных сервисов.

В случае успешного выполнения запроса сервис отвечает сообщением с HTTP–кодом "200 OK" и указанным в соответствующей таблице значением в теле сообщения. В случае ошибки сервис отвечает сообщением с HTTP–кодом "422 Unprocessable Entry" и JSON–значением {"status"="error", "message"="..."} в теле сообщения. Поле message в теле сообщения содержит пояснения к ошибке.

Сервис мониторинга

Сервис мониторинга необходим для облегчения разворачивания и поддержки системы. СМ централизованно хранит список активных сервисов и их состояний. Если определённый сервис не может оповестить сервис мониторинга о своей активности в течение определённого времени, то он считается отключившимся и записи, соответствующие ему, удаляются из списков СМ. Сам отключившийся сервис должен пытаться перерегистрироваться в сети, с тем чтобы его работа возобновилась по устранению проблем с сетью.

API СМ приведено в таблице 3.

Таблица 3: API сервиса мониторинга

Ресурс	Метод	Параметры	Возвращаемое JSON – значение в случае успеха:
/services/type	POST	port, state	{“status”:“success”, “address”:“ipaddr:port”}
	GET		{address:{state:“...”, lastbeat:“...”}, ...}
/services/type/address	PUT	state	“status”=“success”
/services/type/address	GET		{“state”:“...”, “lastbeat”:“...”}

Сервис БД

Основная задача сервиса БД – инкапсуляция СУБД и отображение структуры БД на REST-API в виде ресурсов – объектов (Agents/id, /traits/id, tasks/id и т д).

Выдача результатов производится в формате JSON. Сервис предоставляет одинаковое API для доступа ко всем таблицам. API сервиса, относящееся к таблице БД “type”, приведено в таблице 4.

Таблица 4: API сервиса БД

Ресурс	Метод	JSON – параметры	Возвращаемое JSON – значение в случае успеха:
/type	GET		массив со списком идентификаторов объектов types
	POST	объект type	{“status”=“ok”}
/type/id	GET		объект type
	PUT	объект type	{“status”=“ok”}
	DELETE		{“status”=“ok”}

Балансировщик

Основная задача балансировщика – выдача новых задач вычислительным узлам, отслеживание состояния узлов и запись присланных вычислительными узлами результатов в базу данных. API балансировщика приведено в таблице 5. При обращении на ресурс "/result" в теле сообщения должен быть архив с результатами.

Таблица 5: API сервиса–балансировщика

Ресурс	Метод	Заголовки	Возвращаемое JSON – значение в случае успеха:
task	GET	AgentID	{"task "=объект task, "subtask"=объект subtask, "archive"=архив задачи}
result	POST	AgentID, SubtaskID, Status	{"status"="ok"}
hearbeat	POST	AgentID	{"status"="ok"}

Фронтенд вычислительных узлов

Дублирует эндпойнты балансировщика; добавлены функции:

- Регистрация нового узла в сети
- Подключение зарегистрированного узла к сети

Запросы на эндпойнты балансировщика проходят проверку безопасности и уходят к балансировщику(ам), запросы на регистрацию/подключение идут сразу к серверу сессии.

Сервер сессии

Отвечает за аутентификацию пользователей на обоих фронтендах. В случае несоответствия ключей безопасности ожидаемым запрос не пропускается "внутрь" комплекса. Внутри комплекса - "доверенная" область, проверок безопасности нет.

Сервер логики

Должен обеспечивать функционал:

- Регистрация нового пользователя
- Вход пользователя в свой аккаунт

- Постановка новой задачи на выполнение
- Просмотр статусов поставленных задач
- Для выполненых задач - получение результатов в виде архива

Пользовательский фронтенд

Дублирует эндпойнты сервера логики, проверяя ключи перед перенаправлением запроса на сервер логики.