Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ”

ОТЧЕТ
О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

ТЕОРЕТИЧЕСКОЕ ИССЛЕДОВАНИЕ ПО ТЕМАТИКЕ МАГИСТЕРСКОЙ ДИССЕРТАЦИИ

«ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ИНТЕГРИРОВАННОГО РЕШЕНИЯ ПО КОНСОЛИДАЦИИ РЕСУРСОВ СХД СЕРИИ EMC VNX»

Автор Трофимов Владислав Александрович _____
(Фамилия, Имя, Отчество) (Подпись)

Направление подготовки 09.04.02 Разработка корпоративных информационных систем

Квалификация Магистр
(бакалавр, инженер, магистр)

Руководитель Маятин А.В., доцент, к.п.н. _____
(Фамилия, И., О., учёное звание, степень) (Подпись)

К защите допустить

Зав. кафедрой ИС Парфёнов В.Г., проф., д.т.н. _____
(Фамилия, И., О., учёное звание, степень) (Подпись)

“___” _____ 2016 г.

Санкт-Петербург, 2017 г.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ

- *Система Хранения Данных* – комплексное программно-аппаратное решение по организации надёжного хранения информационных ресурсов и предоставления гарантированного доступа к ним [1].
- *Fully Automated Storage Tiering for Virtual Pools* – технология автоматического перемещения данных *LUN* в оптимальное расположение внутри *Storage Pool* [2].
- *LUN* – номер, используемый для идентификации логического устройства хранения данных, адресуемого протоколами iSCSI и FCP [3].
- *Multicore Cache* – кэш, основанный на дополнительной DRAM памяти *СХД* [2].
- *Multicore FAST Cache* – высокопроизводительных кэш большого объёма, основанный на твердотельных накопителях [2].
- *Storage Pool* – единое хранилище гомогенных или гетерогенных физических дисков, на основе которого могут быть созданы *LUN* [2].
- *Redundant Array of Independent Disks* – технология виртуализации данных, которая объединяет несколько дисков в логический элемент для избыточности и повышения производительности [4].
- *Virtual Pool* – абстракция над нижележащими *Storage Pools* с определёнными характеристиками производительности и защиты данных [5].

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

– *СХД* – Система Хранения Данных.
– *FAST VP* – Fully Automated Storage Tiering for Virtual Pools.
– *DRAM Cache* – Multicore Cache.
– *FAST Cache* – Multicore FAST Cache.
– *SP* – Storage Pool.
– *RAID* – Redundant Array of Independent Disks.
– *VP* – Virtual Pool.

ВВЕДЕНИЕ

Выделение ресурсов *СХД* на основе требований приложений к динамическим показателям производительности является одной из ключевых особенностей современных *СХД* компании EMC. Данная функциональность реализована только в старших поколениях *СХД* и не поддерживается более бюджетными конфигурациями *СХД*, такими, как EMC VNX. Для реализации данной концепции для *СХД* рассматриваемой серии необходимо создание прототипа решения по интеграции 3 различных продуктов компании EMC, которое позволит автоматизировать процесс консолидации ресурсов *СХД* в *VP*, который будет удовлетворять требованиям заданного приложения к времени отклика пространства блокового доступа, с последующим выделением на нем *LUN*. Управление ресурсами *СХД* является глобальной задачей, выходящей за рамки одного центра обработки данных на уровень взаимодействия между различными *СХД*, расположенных в различных частях света. Построение отказоустойчивого распределенного решения позволяет обеспечить надежность процесса и консистентность потоков данных при управлении ресурсами *СХД* даже при выходе каналов связи, соединяющих один или несколько центров обработки данных с другими.

1. Обзор предметной области

Современные *СХД* используются для хранения данных разнообразных приложений: баз данных (Oracle, MSSQL), почтовых серверов (MS Exchange), файловых хранилищ (Samba), виртуальных рабочих станций (VMware, Citrix Xen) и других. Каждому типу приложений требуются ресурсы *СХД* с различными параметрами хранения:

– Производительность (пропускная способность, время отклика и др.)

– Надёжность (политики репликации, резервного копирования)

– Доступ (различные протоколы доступа и организация защищённого доступа)

СХД содержат большое количество разнообразных дисков (SATA, NL-SAS, SAS, SSD). Концепция *SP* подразумевает объединение однотипных (гомогенный *SP*) или разнотипных (гетерогенный *SP*) дисков в единое хранилище для дальнейшего выделения на нем *LUN* с экспортом различным сетевым устройствам по одному из поддерживаемых *СХД* протоколов. При этом объединённые в один *SP* диски автоматически конфигурируются в наборы *RAID* массивов одинакового уровня, указываемого при создании соответствующего *SP*. Примеры различных гомогенных *SP* приведены на рис. 1.

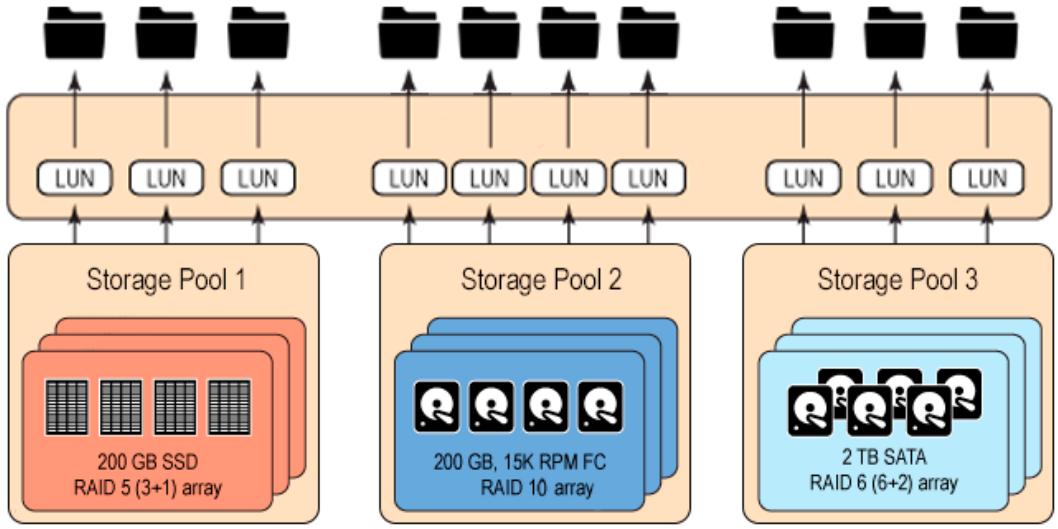

Рисунок 1 – Гомогенные *SP*

Основным ресурсом блокового доступа *СХД* являются *LUN*, которые, в зависимости от статических (тип дисков, количество дисков, объем дисков и др.) и динамических (утилизация, количество запросов в единицу времени и др.) характеристик *SP*, имеют различные показатели производительности.

На параметры производительности отдельно взятого *LUN* влияет не только нагрузка, оказываемая использующим его приложением, а также нагрузка, оказываемая другими приложениями на *LUN*, выделенные на том же *SP*. Учитывая, что динамические показатели производительности *SP* постоянно меняются, их рассматривают в совокупности за

определённый промежуток времени с представлением в виде конкретной метрики, например, в виде среднего значения за рассматриваемый период.

СХД серии EMC VNX поддерживают технологии *DRAM Cache* и *FAST Cache*, которые в реальном времени анализируют используемые ресурсы *СХД* и автоматически перемещают наиболее активно используемые данные в *DRAM Cache*, а также на более высокопроизводительные дисковые носители в пределах одного *SP*. Использование данных технологий позволяет нивелировать воздействие нескольких *LUN* внутри одного *SP* друг на друга. Однако, поддержка этих технологий подразумевает установку в *СХД* дополнительных дорогостоящих компонентов, поэтому простые конфигурации *СХД* серии EMC VNX не поддерживают данные технологии и используют гомогенные *SP*. Именно *СХД* подобного типа являются объектами рассмотрения в данной работе.

Для более удобного и быстрого управления ресурсами различных *СХД* компанией EMC была разработана концепция *Virtual Pool*, которая позволяет консолидировать разнообразные *SP* по определённому параметру, причём как в рамках одной *СХД*, так и в рамках большого количества *СХД*, в том числе расположенных в различных центрах обработки данных. *VP* представляет собой сервис хранения данных, на основе которого можно выделять *LUN*.

Существует 3 вида *VP*: блоковый, файловый и объектный. Каждый из них определяет вид предоставляемых данным *VP* ресурсов. В данной работе рассматриваются блоковые *VP*. Для каждого из них администратор может задать следующий набор параметров для консолидации:

- Протокол доступа (FCP, iSCSI).
- Тип *СХД* (VPLEX, VMAX, VNX, Isilon, NetApp, Hitachi и др.)
- Параметры защиты (репликация и резервное копирование).
- Параметры производительности (Тип дисков и др.)

Примеры различных конфигураций *VP* приведены на рис. 2.

Администратор также может определить, будет ли операция по подбору и включению в *VP* подходящих под заданные параметры *SP* производиться автоматически, либо вручную включить в *VP* только определённое подмножество из предложенных вариантов *SP*.

Важной особенностью является то, что концепция *Virtual Pool* поддерживает консолидацию только по статическим параметрам производительности и не поддерживает различные динамические параметры производительности, такие как время отклика.

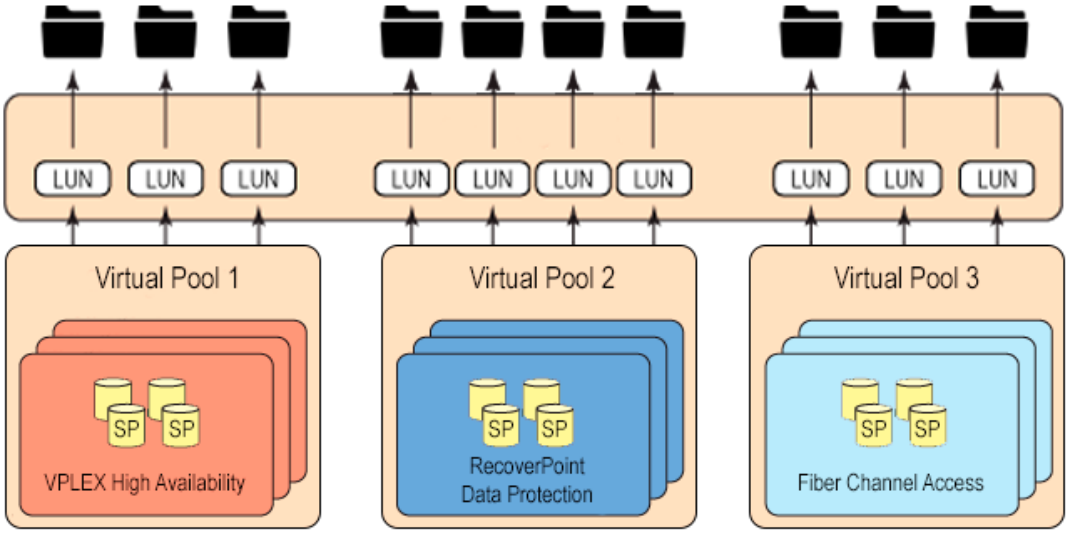

Рисунок 2 – Различные конфигурации *VP*

Консолидация и выделение ресурсов *СХД* на основе требований к динамическим показателям производительности имеет широкое распространение в *СХД* более высокого уровня. Примером является серия *СХД* EMC VMAX3. При выделении *LUN* на таких *СХД* у администратора запрашивается описание типового приложения, под которое требуется выделить *LUN*, а также требуемое время отклика, после чего *СХД* при помощи *DRAM Cache* особо большого объёма, а также *FAST Cache* пытается предоставить требуемое время отклика для всех *LUN*, выделенных на данной *СХД*, независимо от производимой на них нагрузки.

Целью данной работы является улучшение прототипа интегрированного решения по консолидации ресурсов *СХД* серии EMC VNX на основе требований приложений к времени отклика используемого пространства блокового доступа. При этом консолидация достигается путём включения подходящих *SP* в создаваемый *VP*. Данный прототип был создан в рамках ранее выполненной ВКР и обладал ограниченной функциональностью.

Стоит отметить, что подобная консолидация подразумевает рассмотрение 3 ключевых сценариев работы:

а) Первоначальный подбор подходящих *SP* при объединении в *VP*.

б) Выделение *LUN* на *VP* с проверкой соответствия хотя бы одного *SP* требованиям к производительности.

в) Активный мониторинг использования выделенных *LUN* на *VP* с донесением до администратора информации о слишком загруженных *SP* или неэффективном их использовании.

Разработанный на этапе выполнения ВКР прототип реализовывает только первый из рассмотренных пунктов. В данной НИР рассматривается улучшение имеющегося прототипа с целью реализации пункта «б».

1.1. Описание интегрируемых продуктов

1.1.1. ViPR Controller

EMC ViPR Controller представляет собой программный продукт для централизованного и унифицированного управления и работы с ресурсами разнообразных *СХД* как компании EMC, так и сторонних производителей [6]. Имеет версию с открытым исходным кодом – CoprHD.

Данный программный продукт привносит новый уровень абстракции над ресурсами *СХД – Virtual Pool*. Подключение к *СХД* осуществляется посредством специального SMI-S провайдера, уникального для каждой серии *СХД*.

Среди большого разнообразия возможностей, предоставляемых данным продуктом, стоит выделить следующие:

- Предоставление статической информации о *СХД*, её внутренней структуре и ресурсах.
- Выполнение операций над *VP*: создание, изменение и удаление.
- Выделение и удаление *LUN* на созданных *VP*.
- Экспорт выделенных *LUN* сетевым устройствам.

Данный программный продукт имеет веб интерфейс для администрирования и выполнения различных операций над подключёнными *СХД*. Помимо пользовательского интерфейса имеется REST API для выполнения тех же операций, что и через пользовательский интерфейс [7]. Также имеет реализации программных библиотек для различных языков программирования, в том числе Java, которые инкапсулируют работу с REST API [8].

1.1.2. ViPR Storage Resource Manager

EMC ViPR SRM представляет собой комплексное решение для мониторинга и анализа использования подключённых *СХД* и их ресурсов в реальном времени [9]. Предоставляет наглядную визуализацию взаимосвязей *СХД*, позволяет анализировать конфигурации и рост ёмкости.

Подключение к *СХД* осуществляется напрямую посредством бинарного протокола конкретной *СХД*, без каких-либо промежуточных компонентов, как в случае с продуктом

ViPR Controller. Данный способ подключения позволяет извлекать из *СХД* более детальную информацию, которую нельзя извлечь посредством SMI-S провайдера.

Основной функциональной возможностью в рамках решаемой задачи является предоставление статической и динамической информации о *СХД* и её ресурсах в совокупности за определённый промежуток времени с представлением в виде различных метрик.

Имеет пользовательский веб интерфейс для отображения информации предоставляемой информации в виде схем, графиков и таблиц. Помимо пользовательского интерфейса имеет SOAP API с описанием структуры запросов и ответной информации в виде WSDL файла.

1.1.3. VNX Sizer

EMC VNX Sizer представляет собой настольную утилиту, которая позволяет рассчитывать характеристики *СХД* серии EMC VNX и входящих в них *SP* под заданной в различных форматах нагрузкой приложений.

В отличие от ранее описанных продуктов, она не имеет сетевого интерфейса. Взаимодействие с ней возможно только посредством пользовательского интерфейса, а также в виде консольной утилиты посредством системных вызовов с указанием входного и выходного файлов. Данные файлы имеют структуру в виде формата JSON.

В рамках ВКР для данной утилиты была разработана специальная сетевая оболочка, позволяющая работать с данной утилитой посредством REST интерфейса.

1.2. Выполнение рассматриваемого процесса вручную

Задача по группировке в *VP* подходящих по времени отклика конкретным типам приложений *SP* была автоматизирована с помощью прототипа, созданного в рамках ВКР. Последующее выделение *LUN* на созданном *VP* с проверкой соответствия требуемому времени отклика решается вручную.

На рис. 3 и 4 соответственно изображены диаграммы последовательности, наглядно иллюстрирующие процесс ручного выполнения администратором действий по созданию *VP* и последующему выделению на нем *LUN* заданного размера. Соответствующие данным процессам варианты автоматизации, реализованные в ВКР и предлагаемые к автоматизации в НИР представлены на рис. 5 и 6 соответственно.

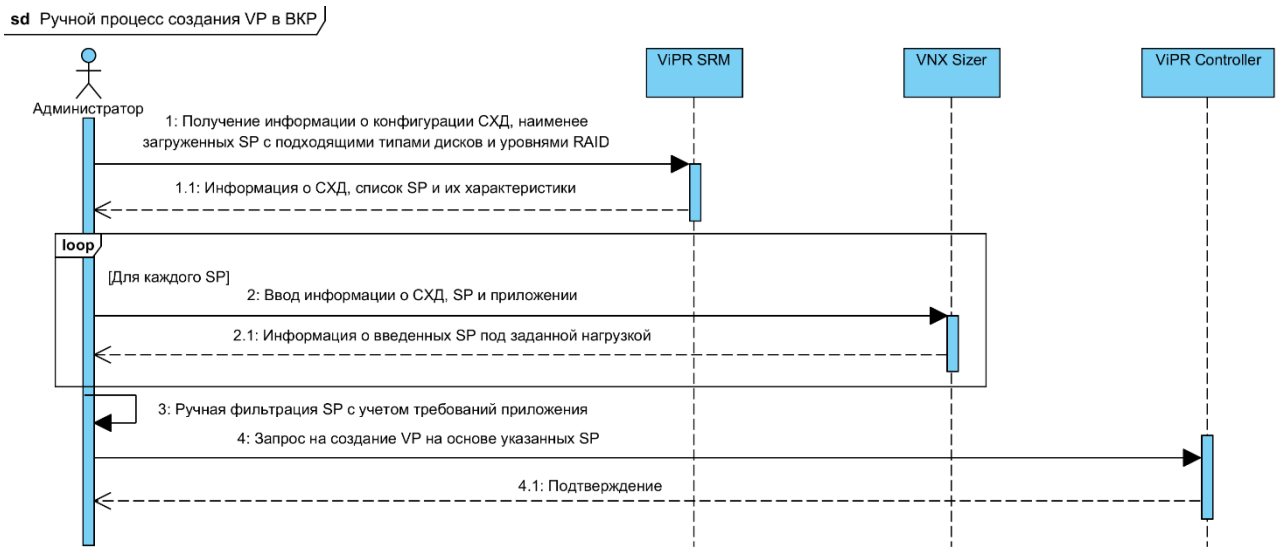

Рисунок 3 – Выполнение процесса по созданию VP вручную в рамках ВКР

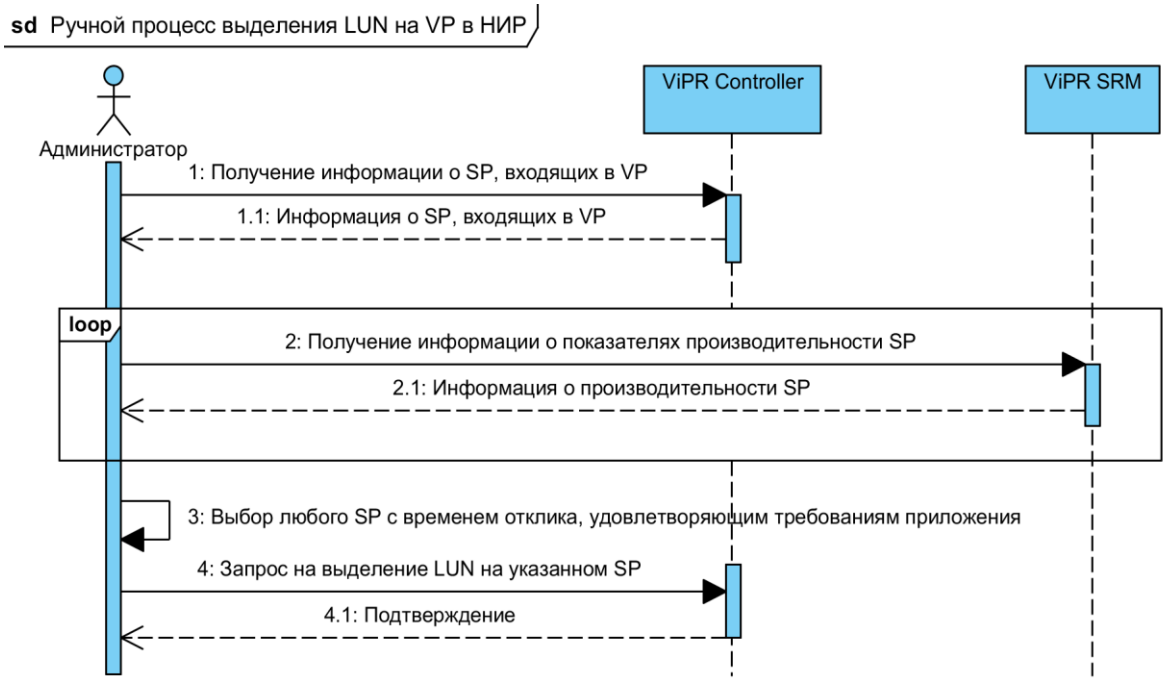

Рисунок 4 – Выполнение процесса по выделению LUN на VP вручную в рамках НИР

1.3. Предлагаемый вариант автоматизации

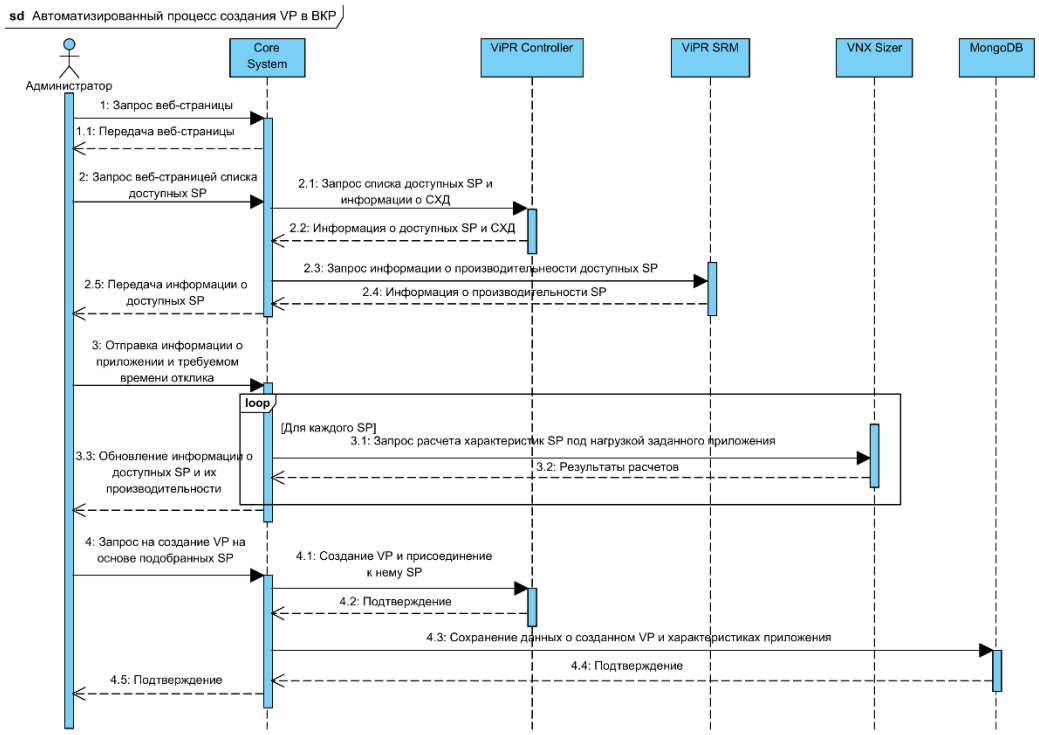

Рисунок 5 – Автоматизированное выполнение процесса по созданию VP

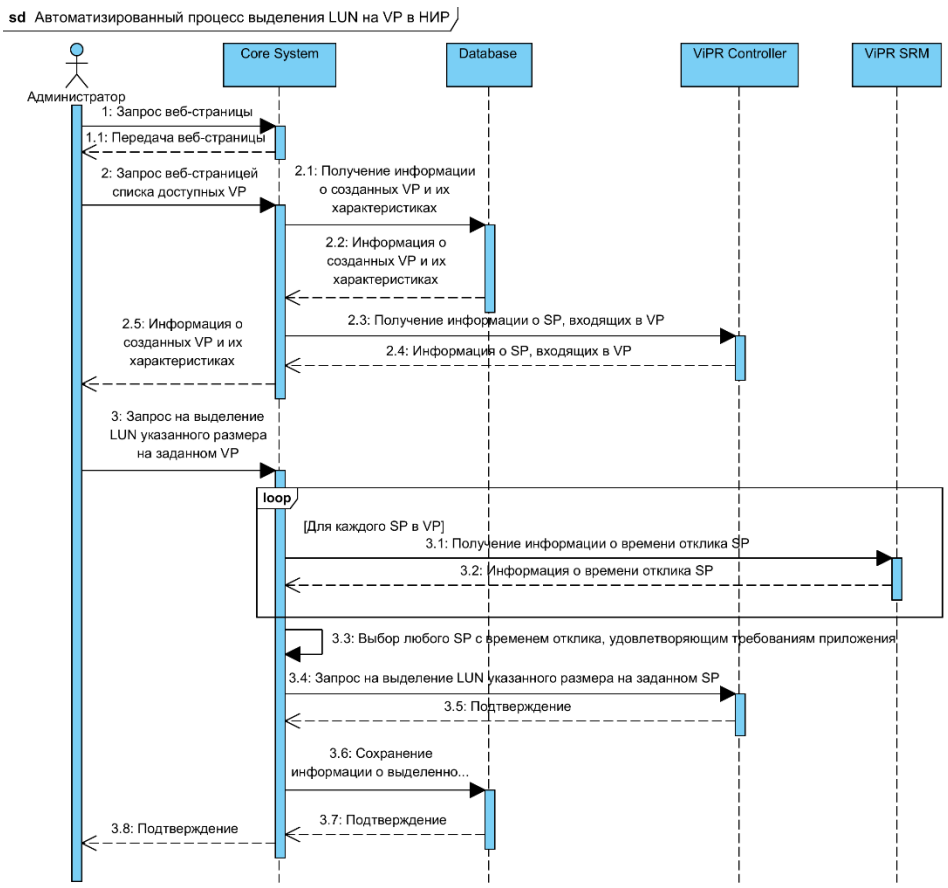

Рисунок 6 – Автоматизированное выполнение процесса по выделению LUN на созданном VP

2. Проектирование

2.1. Функциональная архитектура

В рамках ВКР в качестве функционального требования выделялся только один вариант использования, подразумевающий непосредственную консолидацию подходящих под динамические параметры производительности приложения *SP* в *VP* (рис. 7).

В контексте НИР функциональные требования расширяются одним новым вариантом использования (рис. 8) и рядом функциональных требований:

– Поддержка многопользовательского режима

– Поддержка распределенного режима работы

Под многопользовательским режимом подразумевается добавление идентификации и аутентификации в разрабатываемый прототип с возможностью одновременного выполнения различными пользователями операций, предусмотренных основными вариантами использования (рис. 7, 8). В рамках разрабатываемого прототипа авторизацию для всех пользователей можно считать безусловной.

Работа в распределенном режиме подразумевает возможность запуска как одного, так и нескольких копий приложения с помощью изменения конфигурации. При этом копии приложения должны корректно учитывать изменения, вносимые как различными пользователями внутри одной копии, так и между различными копиями, и исключать ситуации некорректного совместного использования логических ресурсов системы, имеющих изменяющееся во времени состояние (доступные *SP* и характеристики их использования). Система должна быть спроектирована с учетом высокой надежности взаимодействующих узлов и низкой надежности соединяющих их каналов связи. Интегрируемые компоненты ViPR Controller и ViPR SRM поддерживают распределенный режим работы. Компонент VNX Sizer является легковесным REST-сервисом без сохранения состояния, что позволяет осуществить его развертывание в каждом центре обработки данных независимо. Подразумевается, что в каждом центре обработки данных, в котором разворачивается копия системы, данные внешние системы присутствуют. Во время «изоляции» центра обработки данных при выходе из строя каналов связи копия системы должна поддерживать управление ресурсами *СХД* в локальном центре обработки данных. При восстановлении связи между центрами обработки данных копии приложения должны автоматически обменяться информацией об независимых изменениях, внесенных администраторами во время изоляции центра обработки данных.

До внесения изменений каким-то одним пользователем все остальные пользователи должны видеть одно и то же состояние доступных *SP*, т.е. следует не загружать информацию

о состоянии *SP* для каждого пользователя в отдельности, а организовать механизм обновления данной информации для всех пользователей сразу.

Основные варианты использования предусматривают одновременную работу до 10 пользователей (администраторов центров обработки данных). Требование к работе в режиме кластера обусловлено распределенной архитектурой интегрируемых компонентов ViPR Controller и ViPR SRM. Данные продукты позволяют объединять в единую информационную систему различные центры обработки данных в разных частях света. В критических ситуациях отдельный центр обработки данных может оказаться изолирован от остальных, но при этом управление ресурсами СХД внутри него все еще является актуальной задачей.

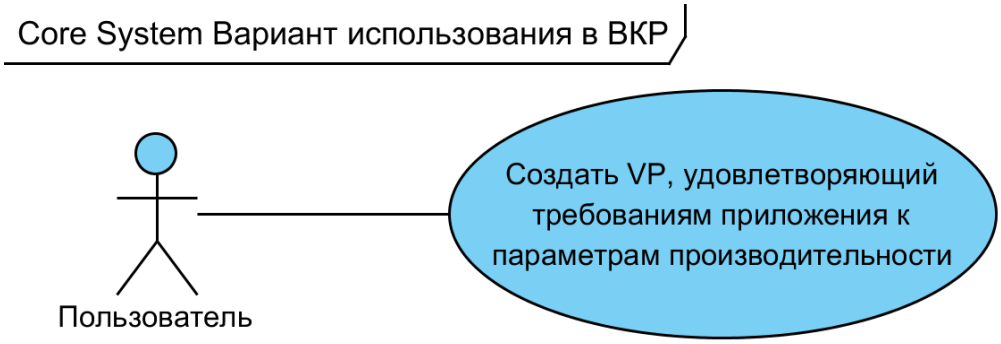

Рисунок 7 – Вариант использования главного компонента системы в ВКР

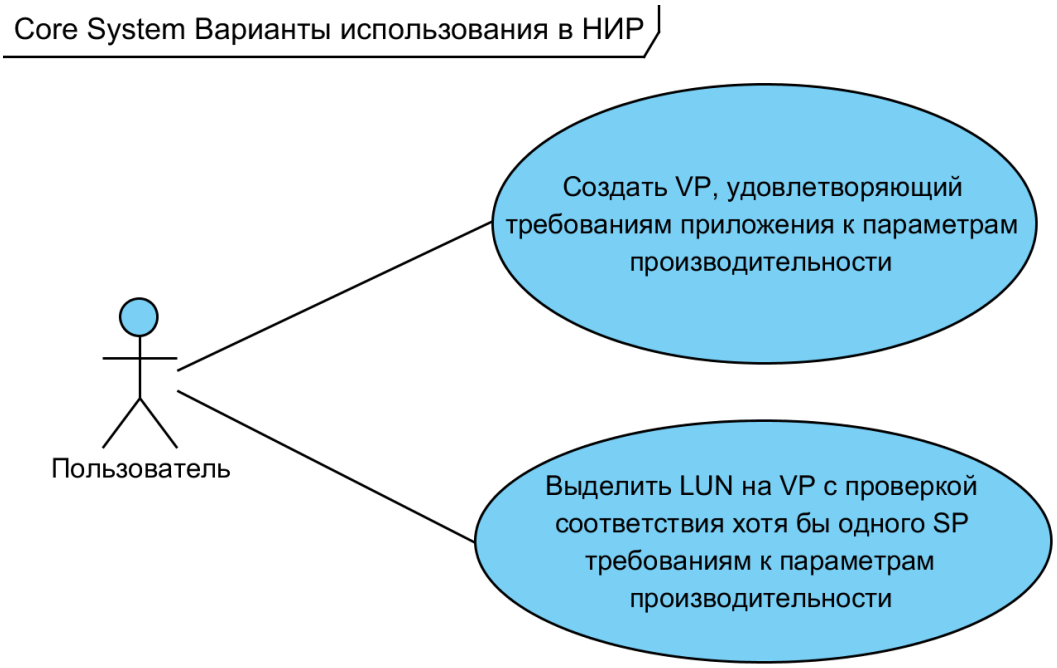

Рисунок 8 – Вариант использования главного компонента системы в НИР

Сценарии, соответствующие вариантам использования по созданию *VP* и последующего выделения на нем *LUN* отображены на рис. 9 и 10 соответственно.

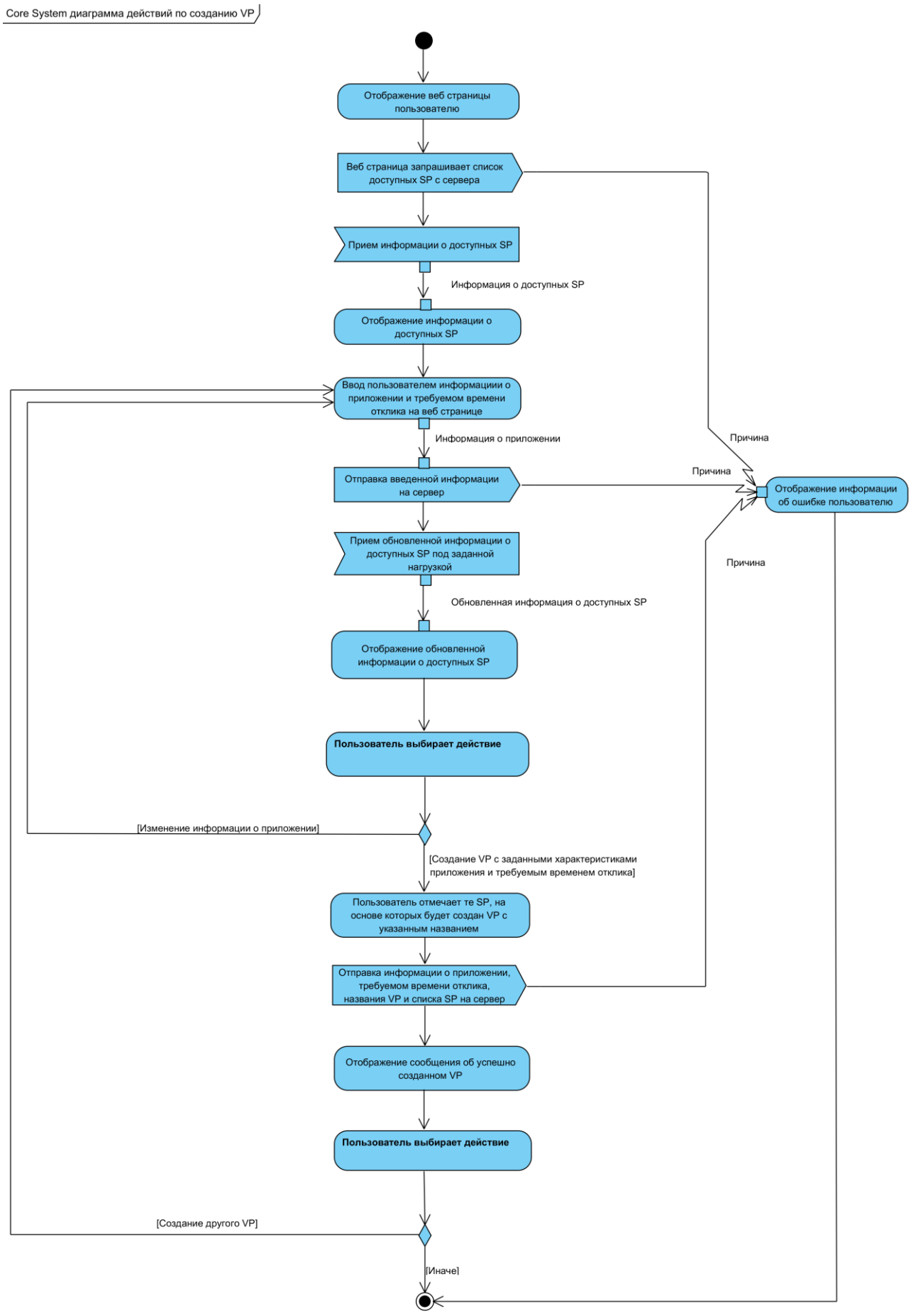

Рисунок 9 – Диаграмма деятельности по созданию *VP*

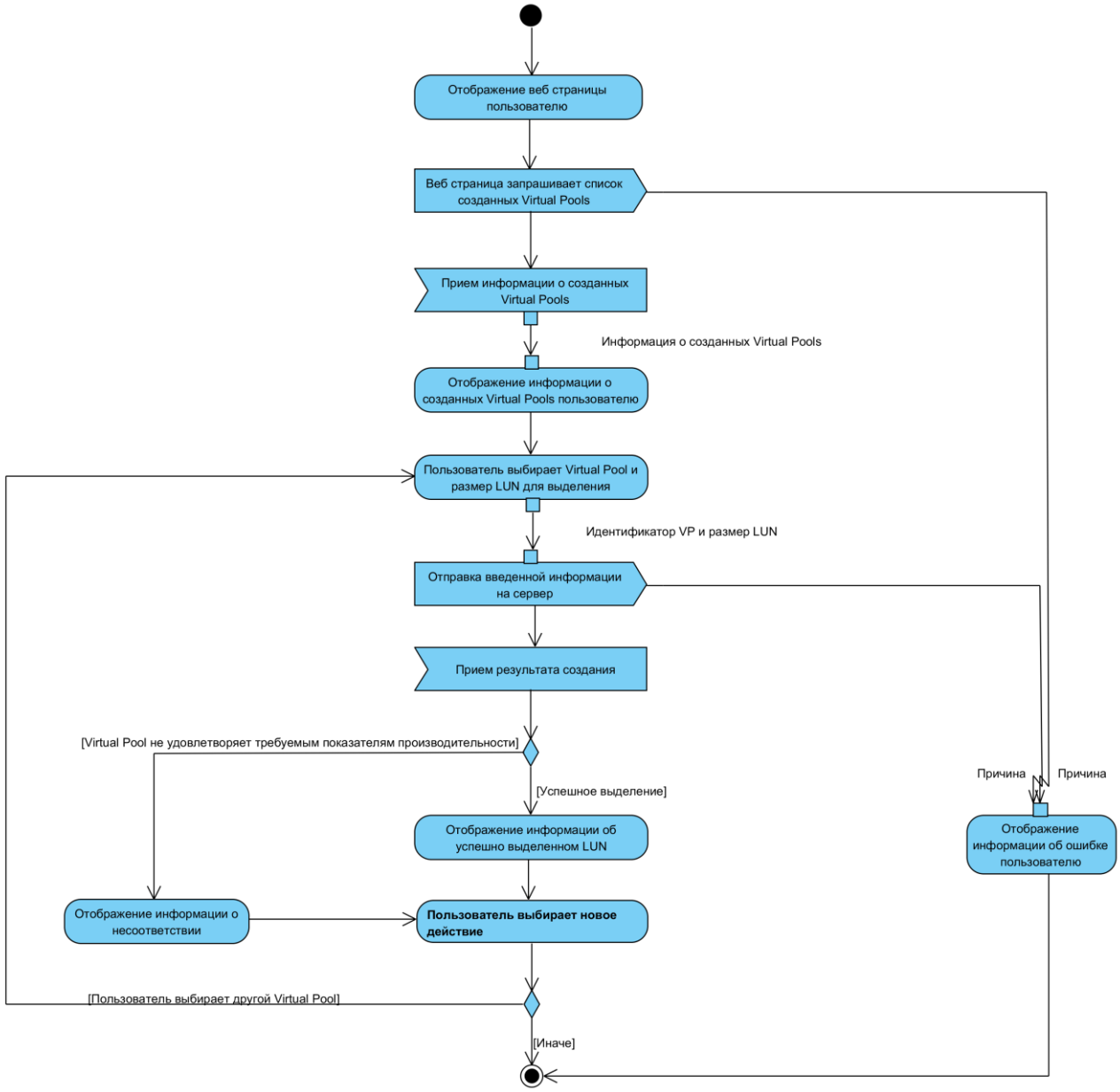

Рисунок 10 – Диаграмма деятельности по выделению *LUN* на ранее создавнном *VP*

2.2. Проблемы, исследуемые в рамках НИР

Преобразование однопользовательского прототипа в многопользовательскую распределенную систему подразумевает не только добавление новой функциональности к имеющемуся прототипу, но и дополнительный анализ и переработку имеющейся архитектуры как на системном, так и на программном уровне, а также уровне данных. Предметом исследования в данной НИР является выявление наилучшего способа организации многопользовательской работы с логическими ресурсами в распределенной системе с учетом используемого стека технологий и системной архитектуры.

В источнике [13] большое внимание уделяется разнице между доступностью и надежностью системы. Под доступностью подразумевается «способность системы к

выполнению операций по требованию», а под надежностью «способность системы безошибочно выполнять операции, для которых она предназначена» [13, с.5]. Разрабатываемая система должна соответствовать как требованию к высокой доступности, так и надежности.

Данный источник [13, с.12] выделяет базовые принципы, которые необходимо соблюдать для увеличения доступности системы:

- Мониторинг копий приложений: периодическая проверка состояния и производительности копий приложения с целью предположения скорого выхода из строя одной или нескольких из них.
- Отслеживание изменений в конфигурации: своевременное обнаружение и принятие соответствующих мер при выходе копии приложения из строя или восстановления работоспособности ранее вышедшей из строя копии.

Источник [14, с. 78] также рекомендует исключить единую точку отказа при организации коммуникаций между копиями системы, за счет настройки избыточности сетевой конфигурации [14, с. 214-225]. Различные конфигурации такой избыточности приведены на рис. 11 и 12 [14, с. 219, 223]. Вариант с подключением нескольких сетевых интерфейсов сервера к одной сети является предпочтительным для высоконагруженных систем с большим потоком обмена данными, а подключение к нескольким различным сетям для систем с требованиями к высокой отказоустойчивости [14, с. 227]. Так как разрабатываемая система относится к классу отказоустойчивых систем, то использование второго варианта будет более предпочтительным.

На рис. 13 изображена схема, иллюстрирующая простую архитектуру организации взаимодействия двух серверов с клиентами [14, с. 370]. Она подразумевает исключение единой точки отказа за счет дублирования каждого компонента системы – сервера, используемых ими хранилищ данных и сетевых интерфейсов.

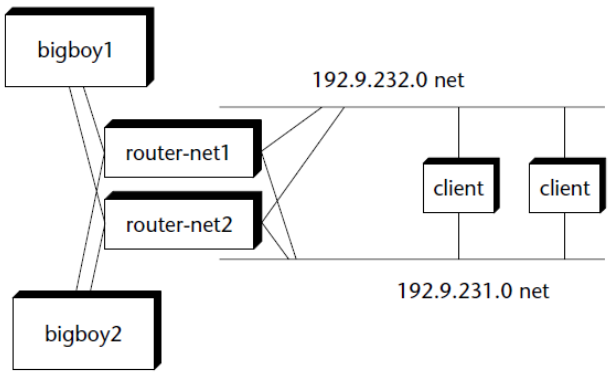

Рисунок 11 – Избыточность сети за счет подключения каждого сервера к нескольким сетям

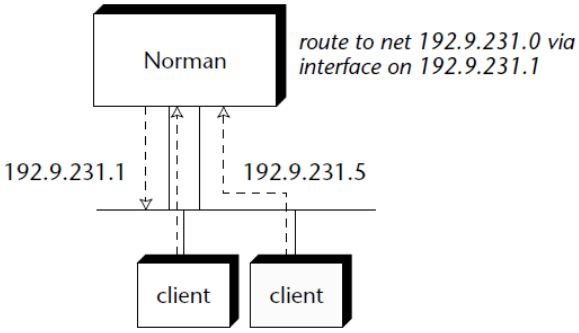

Рисунок 12 – Избыточность сети за счет подключения сервера к сети посредством нескольких интерфейсов сразу

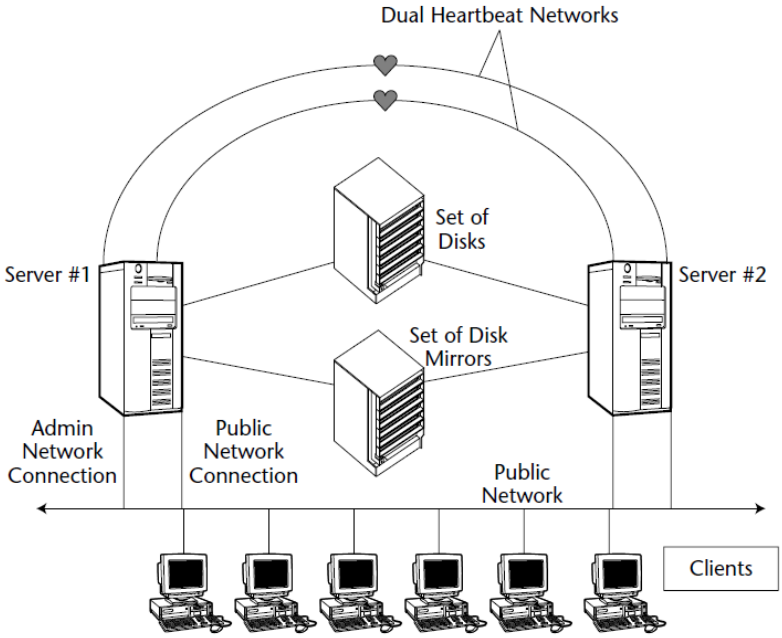

Рисунок 13 – Простая архитектура отказоустойчивого кластера из двух серверов

В классических трудах по проектированию распределенных информационных систем [12] описываются различные подходы, которые как по-отдельности, так и в совокупности могут помочь в решении поставленной задачи.

Используемая в имеющейся системе база данных MongoDB поддерживает работу в распределенном режиме за счет репликации (рис. 14, 15) [16, с.80-81] и шардирования (рис. 16) [17, с. 34]. Минусом данного подхода является использование архитектуры «мастер-реплика» (рис. 14) отсутствие поддержки архитектуры «мастер-мастер», что затрудняет использование данной СУБД без дополнительных средств обеспечения целостности данных при потере соединения между центрами обработки данных. Также это усложняет сетевое взаимодействие при большом числе активных копий системы, так как все коммуникации будут производиться через один узел.

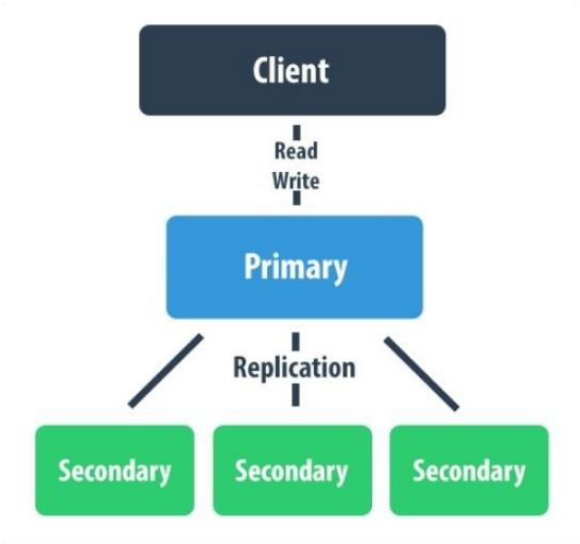

Рисунок 14 – Схема работы MongoDB при использовании репликации

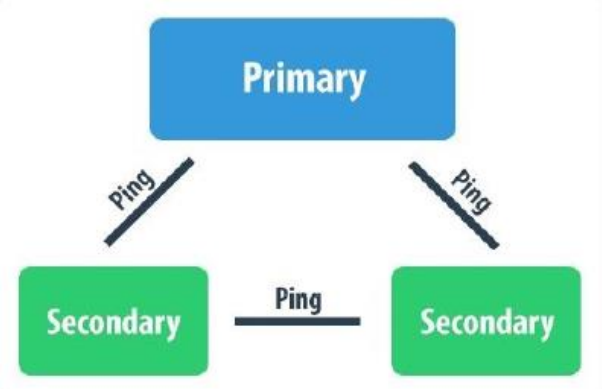

Рисунок 15 – Схема работы мониторинга состояния узлов MongoDB при использовании репликации

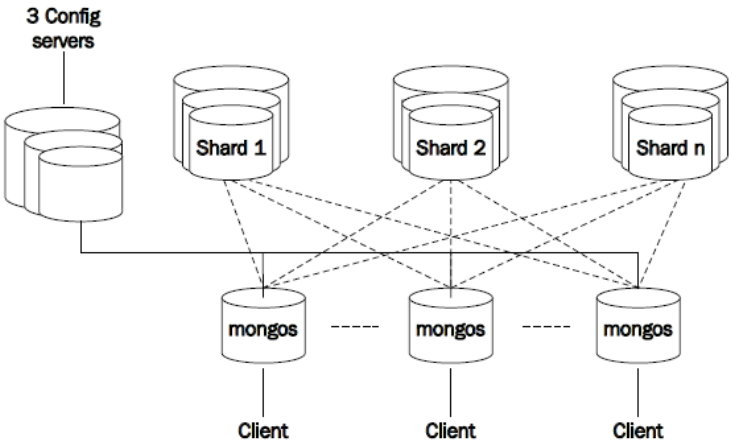

Рисунок 16 – Схема работы MongoDB при использовании шардирования

В трудах, посвященных лучшим практикам настройки и развертывания данной СУБД как в одиночном режиме, так и в центрах обработки данных (рис. 17) [18], подробно описываются недостатки использования данной СУБД при необходимости организации транзакционной логики приложения ввиду отсутствия встроенной поддержки транзакций и

сложностью синхронизации изменений при частом обрыве соединения и переизбранием нового мастер-узла [18, 323-329; 17, с. 17-37].

Однако, использование данной СУБД имеет большие преимущества в случае независимого развертывания копии данной СУБД в каждом центре обработки данных без организации репликации / шардирования между ними при наличии внешней синхронизации и поддержки консистентности данных. Данный способ также позволит избежать проектирования ER модели и перепроектирования логики работы с БД для использования SQL СУБД с поддержкой вышеописанной функциональности на уровне БД.

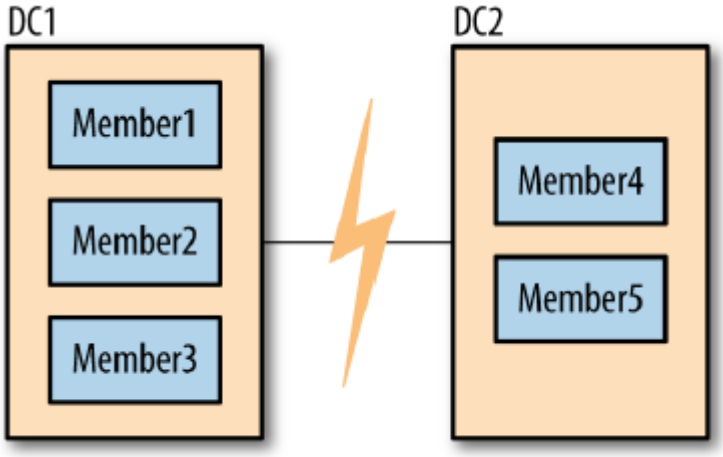

Рисунок 17 – Иллюстрация кластера узлов MongoDB в разных центрах обработки данных

СУБД Oracle предоставляет очень гибкий подход к архитектуре взаимодействия узлов СУБД. Она также поддерживает репликацию и шардирование, имеет поддержку репликации «мастер-мастер» [15, с. 9-1 – 9-3; 9-16 – 9-17], «мастер-реплика» [15, с. 9-4 – 9-15], а также настраиваемый механизм синхронизации данных после восстановления связи между узлами [15, 13-1 – 13-11]. Данная СУБД предоставляет богатые возможности по обеспечению консистентности данных, а также поддерживает транзакции. Однако, настройка каждого узла будет являться очень трудоемкой задачей, требующей высокой квалификации в области данной СУБД. Также, конфигурация данной СУБД будет сильно зависеть от размера кластера, что усложняет масштабирование.

Для решения проблемы синхронизации данных между копиями приложения можно воспользоваться модификацией подходов, описанных в источнике [12].

Современные системы кэширования / хранимые в оперативной памяти БД имеют богатый набор функциональности по работе в распределенном режиме, эффективной синхронизации данных при репликации и автоматической балансировке потока данных при использовании сетевых топологий, изображенных на рис. 11 и 12. Так как узлы проектируемой системы обладают высокой надежностью, данный подход имеет преимущество над подходами, использующими «классические», хранящие данные на

файловой системе, СУБД, за счет легкости масштабирования и скорости работы. Сложностью при применении данного подхода является синхронизация содержимого кэша с используемой СУБД, т.к. несмотря на высокую надежность самих узлов, вероятность их выхода как самой копии приложения, так и используемой СУБД все равно присутствует.

На рис. 18 [12, с. 13] изображена классическая схема применения глобального кэша для увеличения доступа к наиболее часто используемым данным.

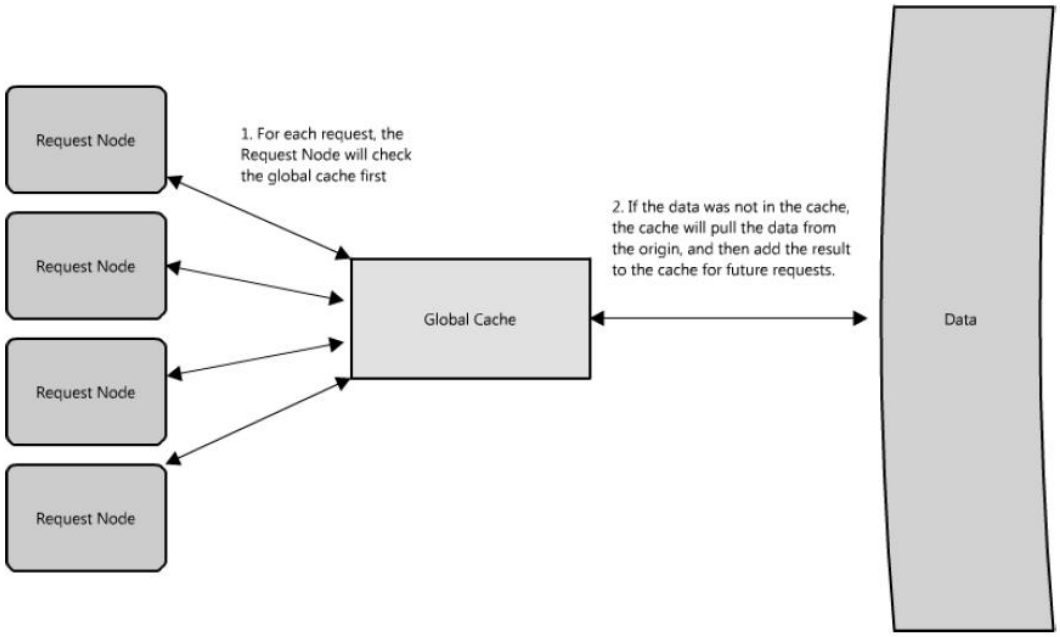

Рисунок 18 – Использование глобального кеша как хранимой в опреативной памяти базы данных

Если использовать хранимую в оперативной памяти базу данных подобно данному кэшу, но не только для чтения данных, но и для их сохранения, мы получаем унифицированный интерфейс работы с данными, сохраняющий все преимущества классических СУБД и нивелирующий их недостатки за счет поддержки большинством современных хранимых в оперативной памяти баз данных исключительных и распределенных блокировок, а также поддержкой некоторых из них автоматического сохранения данных в подключенную СУБД.

Использование выделенного глобального узла для такой БД является нарушением точки единого отказа, поэтому целесообразно использовать распределенные узлы БД, по узлу для каждой копии приложения в центре обработки данных, что соответствует подходу, изображенному на рис. 19 [12, с. 14]. Так как для работы каждого узла системы в каждый момент времени не требуется информация со всех узлов сразу, то данную БД можно использовать в роли кэша – на постоянной основе хранить только данные, непосредственно связанные с локальным узлом, а данные с других узлов, необходимые для расчета, динамически загружать с других узлов и кэшировать. При изоляции одного или нескольких

узлов от других необходимость в данных, хранящихся на них, до момента восстановления соединения, отсутствует. Данный фактор позволяет исключить проблему полного дублирования информации в кластере. Такой подход позволит соблюсти принципы отказоустойчивых систем, изложенные в источнике [14].

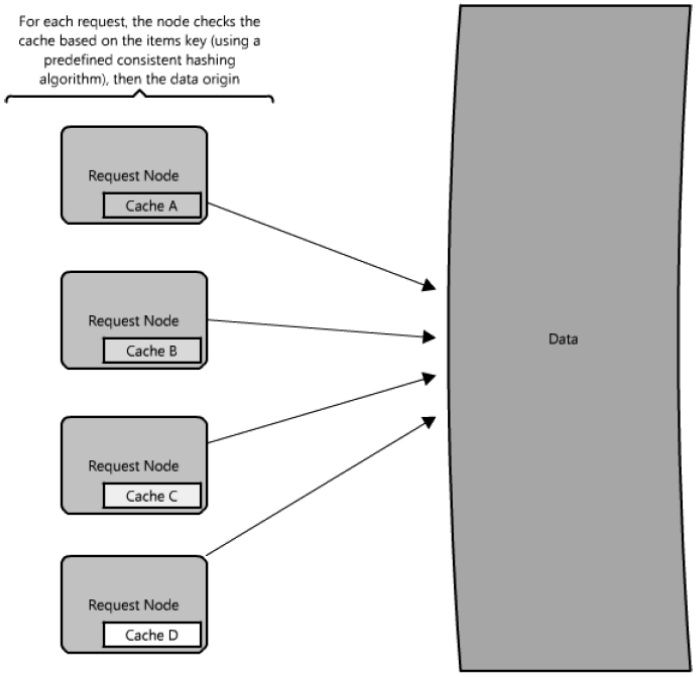

Рисунок 19 – Использование распределенного кеша как хранимой в оперативной памяти базы данных

2.3. Системная архитектура

Рис. 20 и 21 демонстрируют системную архитектуру прототипа в рамках выполнения ВКР и НИР соответственно. Салатовым цветом на них выделены те компоненты, которые были разработаны во время соответствующего этапа работы по разработке данной системы. Как можно заметить, утилита VXN Sizer и ее сетевая оболочка, разработанные в рамках ВКР, в рамках НИР уже становятся внешними системами по отношению к модифицируемым и разрабатываемым компонентам. На рис. 22 представлена архитектура отдельно взятого узла системы.

На данных диаграммах также использованы следующие обозначения:

- EMC VNX Storage System – используемые *СХД* рассматриваемой серии и конфигурации.
- Storage Processor Unit – комплекс аппаратно-программных средств, реализующий взаимодействие и управление *СХД* по определённому протоколу.
- Binary Protocol / Binary VNX Protocol – низкоуровневый протокол взаимодействия с *СХД*, специфичный для каждой модели *СХД*.

– SMI-S Provider for VNX – провайдер для *СХД* рассматриваемой серии. Инкапсулирует выполнение операций по низкоуровневому протоколу. Позволяет управлять одновременно несколькими *СХД* рассматриваемой серии.
– SMI-S API – высокоуровневый протокол взаимодействия с *СХД*, специфичный для каждой серии *СХД*.
– ViPR SRM SOAP API / SOAP API – SOAP-интерфейс, предоставляемый продуктом ViPR SRM для сетевого взаимодействия с ним.
– ViPR REST API / REST API – REST-интерфейс, предоставляемый продуктом ViPR Controller для сетевого взаимодействия с ним.
– VNX Sizer Engine – непосредственно утилита VNX Sizer.
– VNX Sizer WS API Wrapper – компонент-оболочка для утилиты VNX Sizer.
– VNX Sizer WS API / VNX Sizer REST API – REST-интерфейс, предоставляемый компонентом VNX Sizer WS API Wrapper для сетевого взаимодействия с ним.
– Core System / Core System Node – главный компонент реализуемой системы.
– Core System WS API / Core System REST API – веб-интерфейс, предоставляемый компонентом Core System для сетевого взаимодействия с ним.

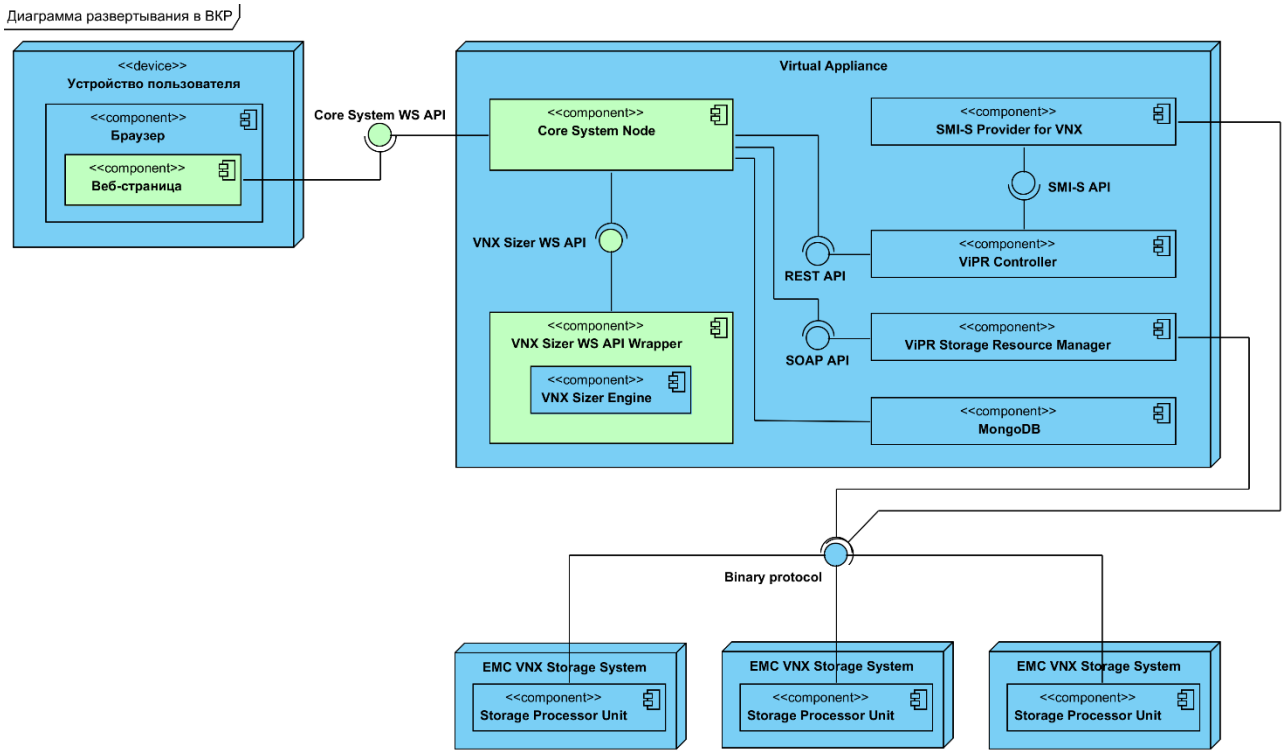

Рисунок 20 – Диаграмма развертывания системы в ВКР

В соответствии с требованиями к высокой доступности разрабатываемого приложения в условиях высокой отказоустойчивости узлов и низкой отказоустойчивости каналов связи, с помощью агрегации имеющихся подходов, рассмотренных в главе 2.2, было принято решение по переходу от монолитной архитектуры реализованного в рамках ВКР

решения к распределенной архитектуре с горизонтальным масштабированием и избыточному подходу к организации каналов связи между отдельными узлами приложения.

Наличие функциональных требований по организации автоматической синхронизации обновлений, произведенных различными узлами приложения после восстановления каналов связи, а также организации блокировок совместно используемых администраторами ресурсов между различными узлами приложения, было принято решение использовать технологию, которая позволяет внедрить данную функциональность без применения серьезных и сложных подходов по ее реализации вручную. Для этого был выбран высокопроизводительная база данных с типом хранения ключ-значение с открытым исходным кодом Hazelcast [19]. Её использование оправдывается максимальной степенью интеграции с уже используемым стеком технологий – она написана на языке высокого уровня Java и предоставляет такие высокоуровневые примитивы синхронизации, как распределенный семафор, распределенные блокировки, распределенные коллекции с конкурентным и блокирующим доступом. Также, ее базовая функциональность позволяет организовывать работу в больших и сложных сетевых инфраструктурах, осуществлять прослушивание и синхронизацию с другими узлами с использованием нескольких сетевых интерфейсов как одной, так и разных сетей. В дополнение, он поддерживает гибкий механизм организации как полных реплик, так и традиционное партиционирование и шардирование. Дополнительной отличительной особенностью, которая также хорошо покрывает имеющиеся функциональные требования, является тонкая настройка политик автоматизированной синхронизации данных между узлами после восстановления связи в результате сбоев сети.

Рисунок 21 демонстрирует структуру компонентов системы. Для организации отказоустойчивого сетевого доступа используется подход с типом подключения между узлами системы «полный граф». С учетом отсутствия требования по большой масштабируемости разрабатываемого решения и ограниченном формальном списке функциональных требований данный подход является приемлемым решением.

Для организации наиболее прозрачного горизонтального масштабирования и простоты развертывания системы в той же виртуальной инфраструктуре принято решение по изоляции каждой копии узла приложения в отдельную виртуальную машину. Подход по использованию больших виртуальных контейнеров, содержащих до нескольких десятков виртуальных машин, объединенных одной или несколькими сетями, является очень распространенным в компании EMC – он упрощает процесс развертывания, быстрого масштабирования и замены вышедших из строя узлов системы на новые. При этом все накладные расходы по организации высокой доступности самих узлом и их коммуникации

между различными центрами обработки данных, как и функционированию виртуальных машин в условиях изоляции центра обработки данных при аварии, берет на себя непосредственно комплекс программно-аппаратных средств, предоставляющий сервис данной виртуальной инфраструктуры.

Так как каждый узел приложения должен предоставлять полную функциональность даже в отсутствие других активных узлов, то вполне логичным является то, что внутри каждой виртуальной машины должны быть развернуты все базовые компоненты системы – ViPR Controller, ViPR SRM, VNX Sizer Engine и непосредственно главный компонент системы. Также каждый узел должен обладать независимой СУБД для долговременного сохранения всех внесенных изменений, а также копию СУБД Hazelcast для прозрачной интеграции и взаимодействия с остальными копиями приложения.

Для упрощения работы с компонентом ViPR Controller в распределенном режиме было принято решение не объединять все его копии в единый логический кластер средствами продукта ViPR Controller, так как это может привести к проблемам когерентности информации, видимой различными узлами приложения после сбоев в сети, с информацией, доступной в СУБД Hazelcast – в конечной счете, по истечению достаточно большого периода времени после сбоя оба информационных пространства, как ViPR Controller, так и Hazelcast, придут в состояние событийной консистентности, но организация механизмов отслеживания подобных сбоев и корректная обработка ошибок в данной ситуации не является целесообразным, ведь основным вариантом использования как продута ViPR Controller, так и продукта ViPR SRM в рамках локальной виртуальной машины является управление теми *СХД*, которые подключены именно к локальным копиям рассматриваемых продуктов. Получение информации и управление *СХД*, подключенными к другим узлам системы можно осуществлять посредством прямых обращений к копиям соответствующих продуктов, развернутых на удаленных узлах приложения. Такой подход позволяет не только снизить накладные расходы сети на излишнюю синхронизацию между копиями данных продуктов при работе в распределенном режиме, но и упростить работу с совместно используемым ресурсами, ведь в таком случае появляется дополнительный контракт, в соответствии с которым каждый совместно используемый ресурс принадлежит одному и только одному узлу, а работу по организации конкурентного доступа к нему возьмет на себя СУБД Hazelcast. По этой же причине нет смысла организовывать сложную распределенную архитектуру СУБД, отвечающий за постоянное хранение данных в системе, т.к. каждый узел, с учетом всех функциональных требований, должен хранить только информацию об изменениях, выполненных над совместно используемыми ресурсами, находящимися в его непосредственном управлении.

Предлагаемая архитектура относится к классу архитектур «узел-узел», в которой каждая копия является равноправной, без выделенного узла, осуществляющего координацию обычных узлов между собой.

Таким образом, каждый узел предоставляет наружу следующий ряд интерфейсов:

- Core System REST API – для выполнения высокоуровневых операций через веб-интерфейс, доступен для пользователя снаружи виртуального контейнера.
- ViPR REST API – для выполнения сервисных операций над совместно используемыми ресурсами системы, недоступен снаружи виртуального контейнера.
- ViPR SRM SOAP API – для получения сервисной информации о показателях производительности совместно используемых ресурсов в реальном времени, недоступен снаружи виртуального контейнера.
- Hazelcast Binary API – низкоуровневый интерфейс меж узлового взаимодействия Hazelcast, недоступен снаружи виртуального контейнера

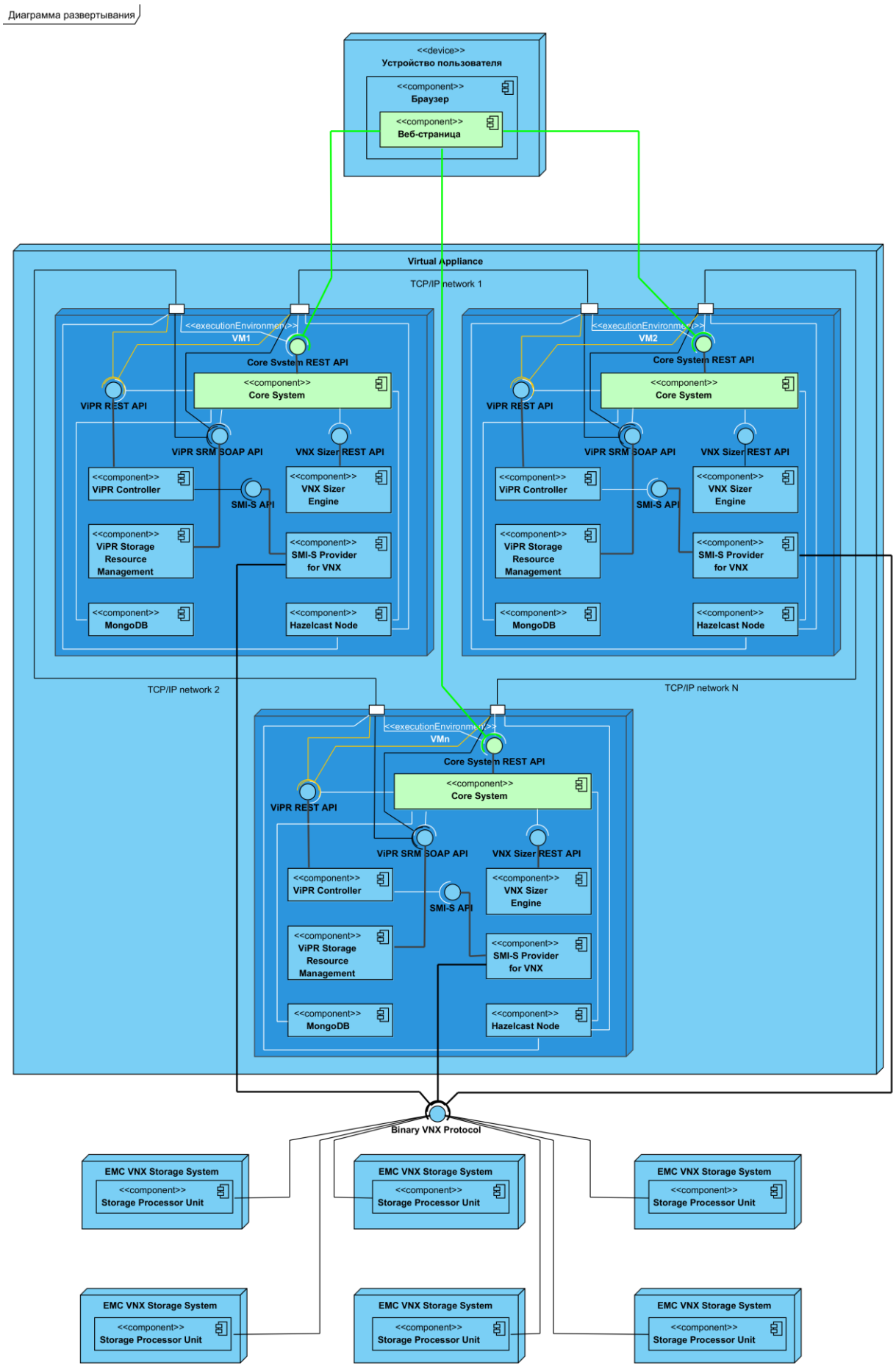

Рисунок 21 – Диаграмма развертывания системы в НИР

Таким образом, каждый узел системы сможет выступать в роли точки входа кластер для выполнения основных операций в рассмотренных в главе 2.1 рамках вариантов использования. Получение информации о совместно используемых ресурсах и их показателях производительности будет осуществляться посредством вызовов API соответствующих продуктов на удаленных узлах.

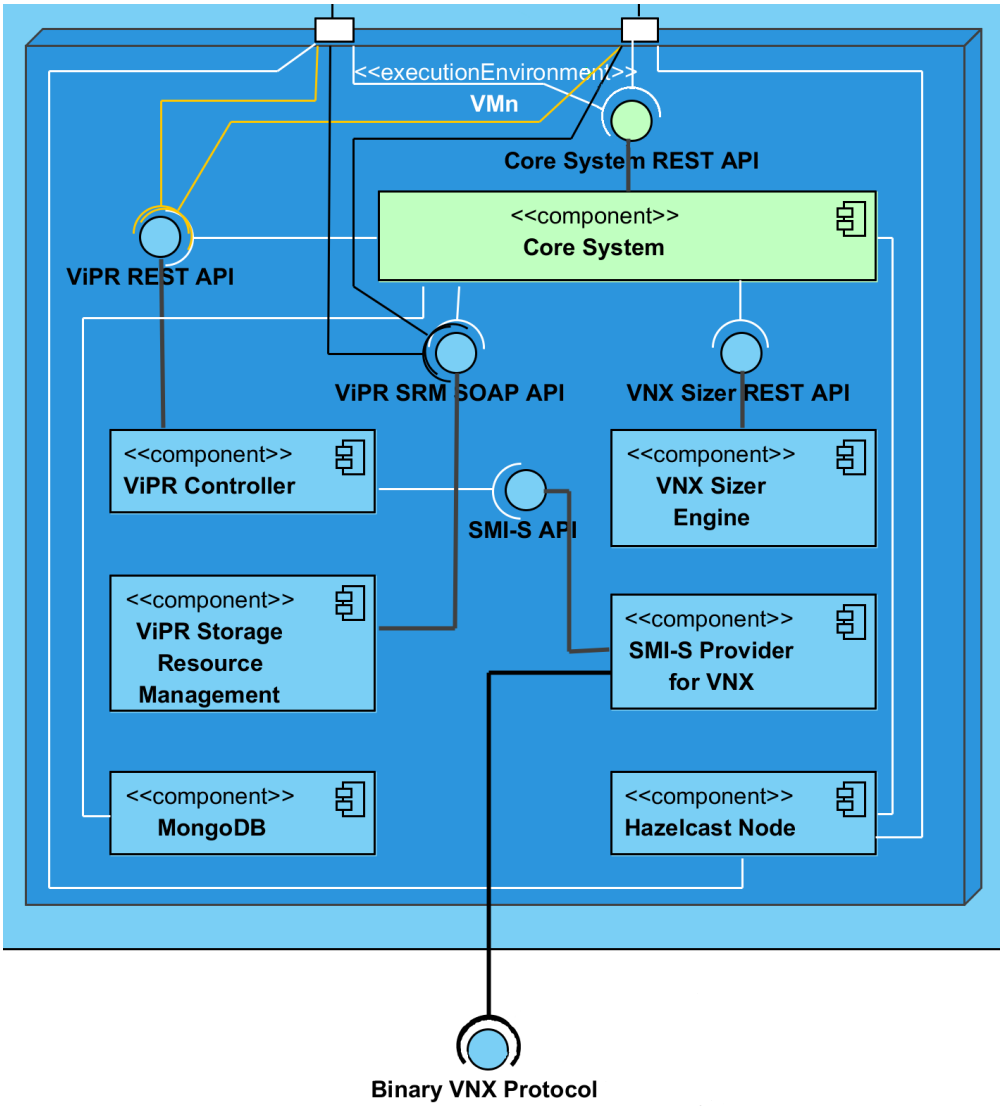

Рисунок 22 – Диаграмма развертывания отдельно взятого узла системы в НИР

2.4. Программная архитектура

Ввиду многообразности различных аспектов реализации разрабатываемой системы в соответствии со спроектированной архитектурой, даже основываясь на имеющейся детализированной программной архитектуре прототипа, реализованного в рамках ВКР (рис. 23), сформировать на этапе проектирования даже высокоуровневую программную архитектуру является достаточно трудоемкой задачей, требующей более детального перекрестного анализа и изучения документаций последних версий используемых в выбранном стеке технологий. На рис. 24 представлена высокоуровневая диаграмма пакетов, планируемая к разработке.

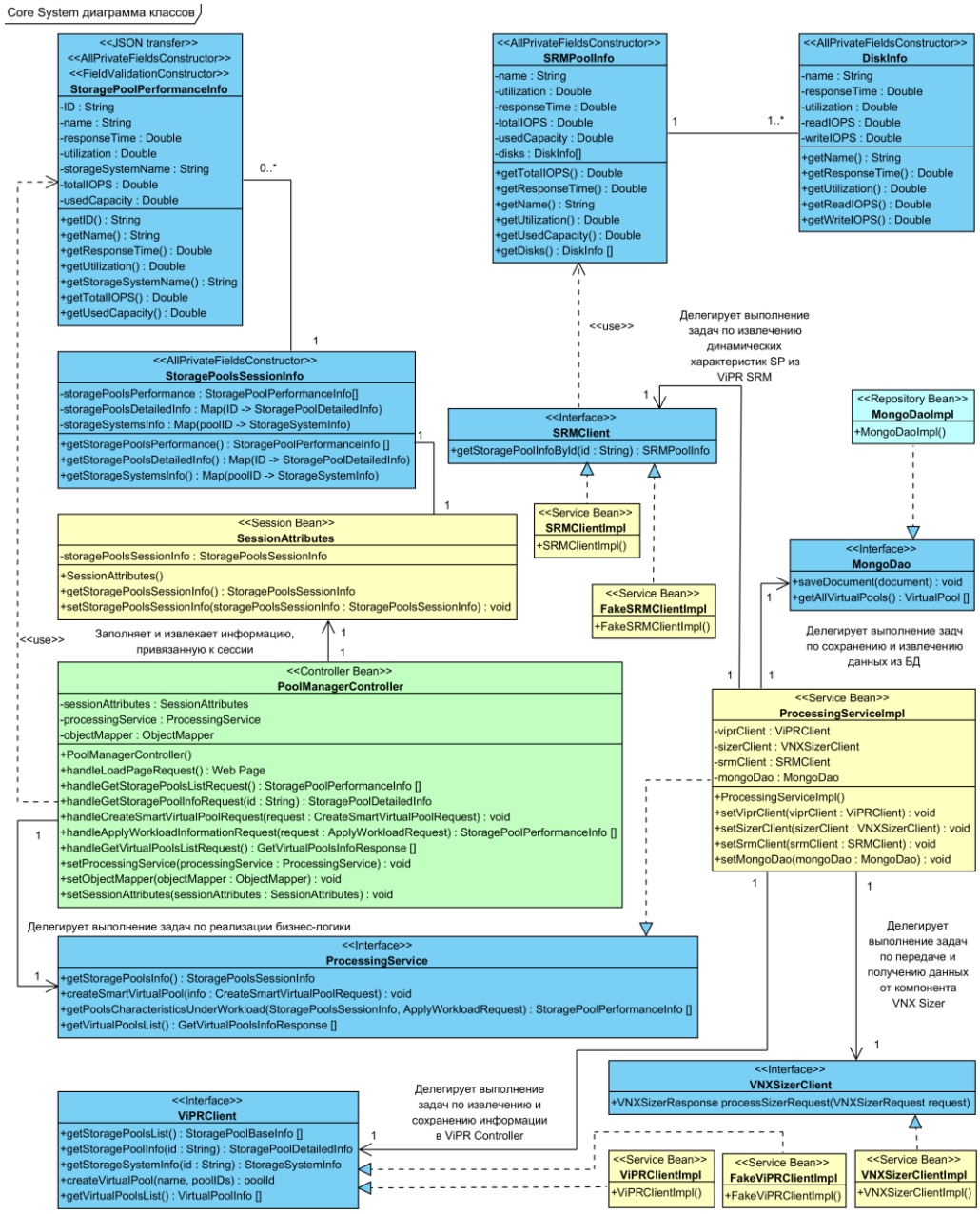

Рисунок 23 – Диаграмма классов главного компонента прототипа в рамках ВКР

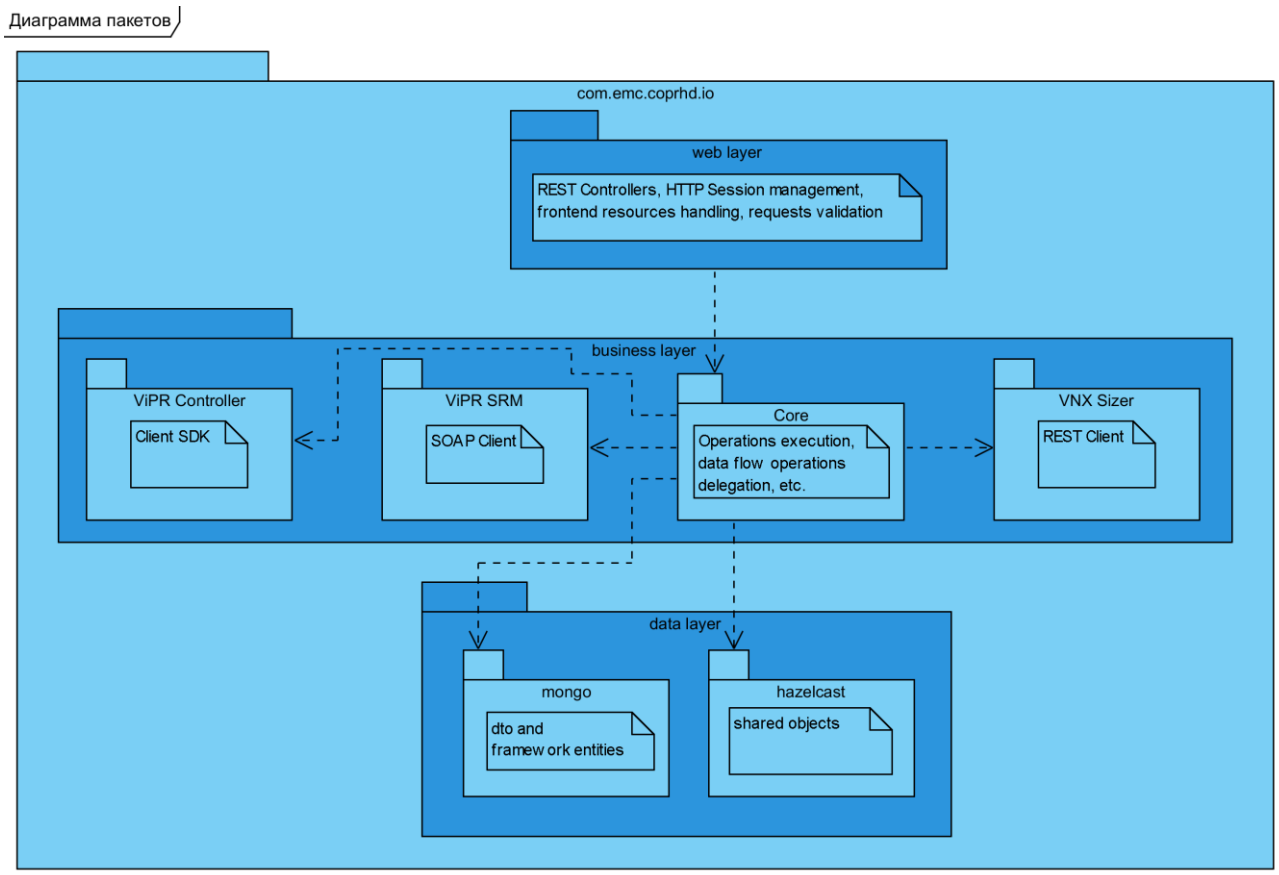

Рисунок 24 – Диаграмма пакетов решения в рамках НИР

2.5. Архитектура данных

2.5.1. Архитектура постоянно хранимых данных

Так как реализация нового варианта использования по выделению *LUN* на ранее созданном *VP* не подразумевает сохранения какой-либо дополнительной информации (вся информация может быть запрошена посредством ViPR Controller), то и структура хранимых данных, как и используемая СУБД для хранения, по сравнению с ВКР, не изменяются. Сохраняется отображение идентификатора созданного *VP* на его целевое время отклика дискового пространства и списка типовых приложений для развертывания с их характеристиками. Диаграмма, представленная на рис. 25, наглядно иллюстрирует вышеописанную структуру.

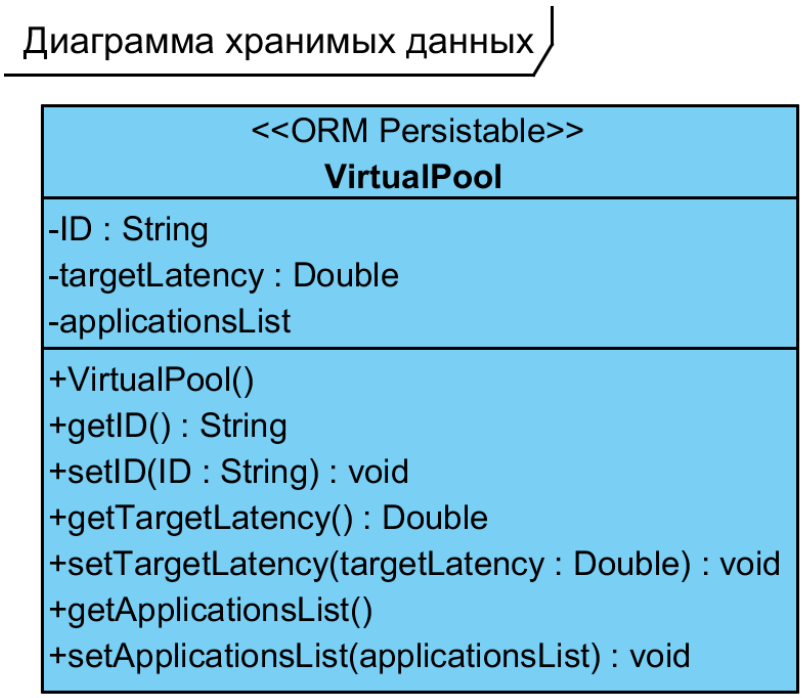

Рисунок 25 – Диаграмма постоянно хранимых данных

2.5.2. Архитектура API

В соответствии с реализуемым в рамках данной работы дополнительным вариантом использования по выделению *LUN* на уже созданном ранее *VP*, к разработанному в рамках ВКР API (рис. 26) добавляется дополнительный метод, который принимает идентификатор выбранного *VP* и желаемый размер выделяемого *LUN* (рис. 27).

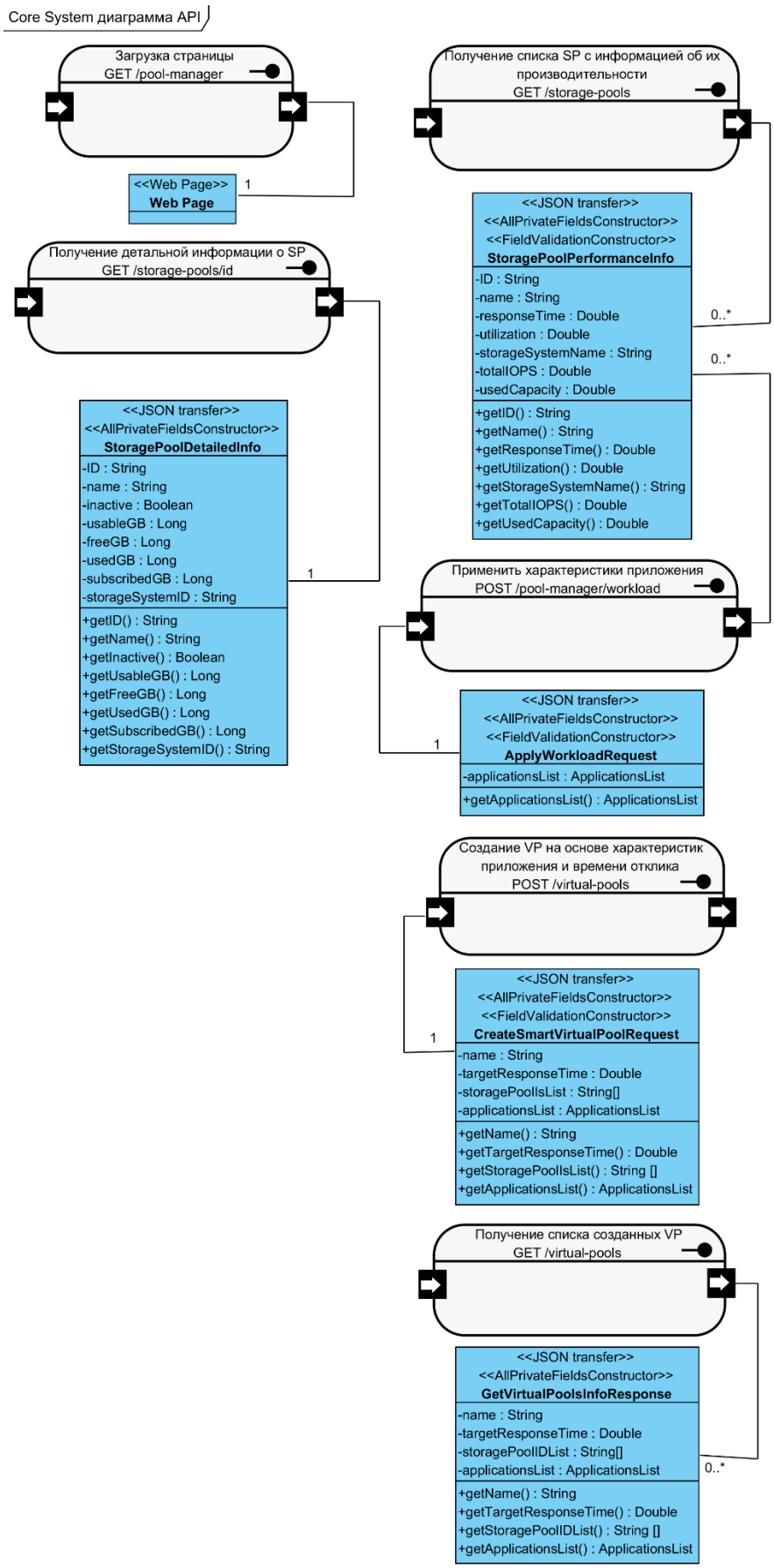

Рисунок 26 – Диаграмма API в ВРК

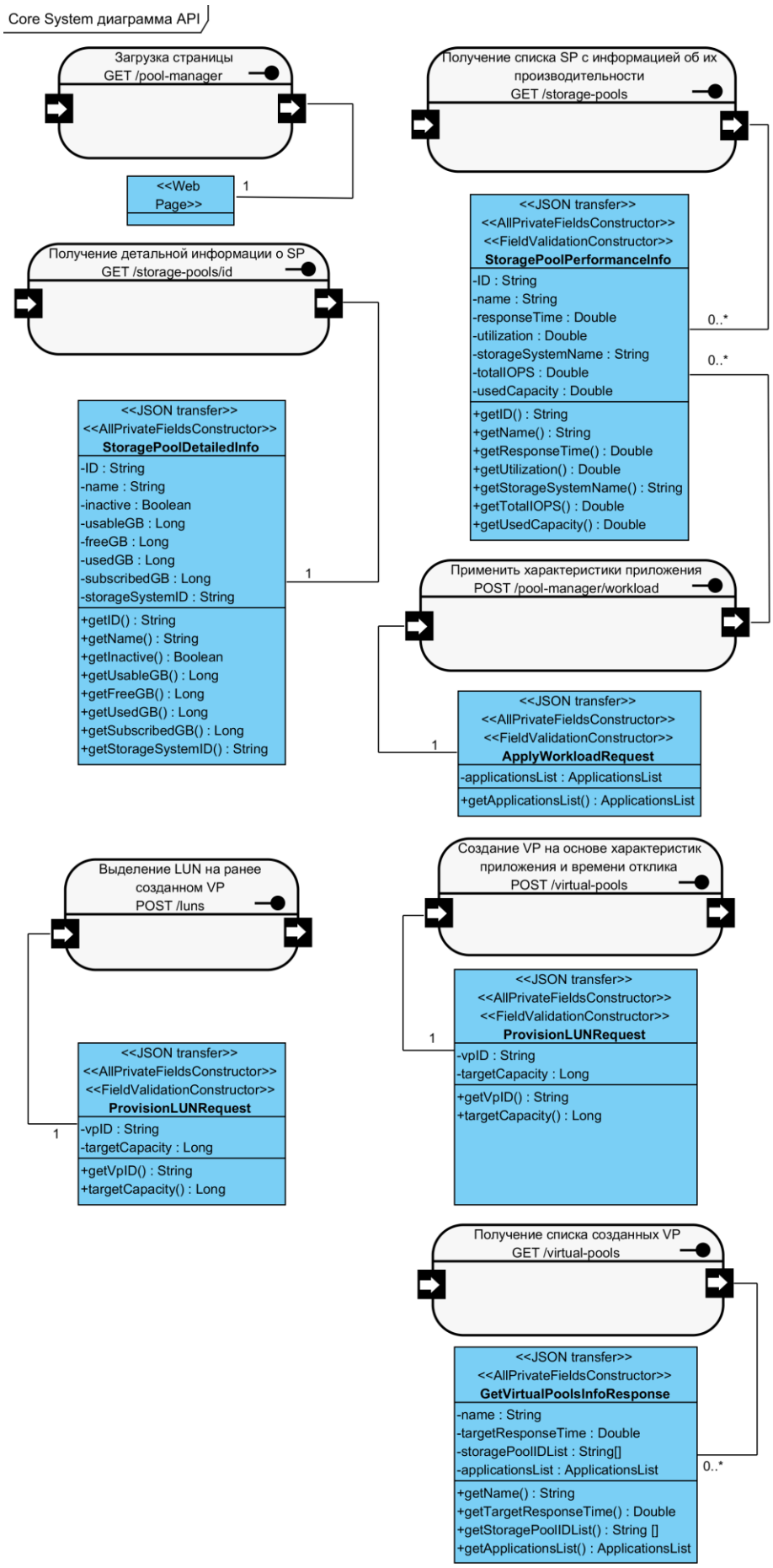

Рисунок 27 – Диаграмма API в НИР

ЗАКЛЮЧЕНИЕ

В рамках данного этапа НИР была определена и подтверждена актуальность и практическая значимость выбранной темы магистерской диссертации, сформирован терминологический базис и обзор прикладной предметной области с описанием используемых компонентов и систем, подлежащих интеграции. В соответствии с целями и задачами, решаемыми в данной диссертации, были подобраны различные практики и подходы к решению задач подобного класса, произведен их базовый анализ в соответствии с имеющейся архитектурой и используемым стеком решения. С учетом функциональных требований и требований к высокой доступности приложения была сформирована системная, высокоуровневая программная архитектуры, а также архитектура хранимых данных и проектируемых интерфейсов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Отдел информатизации образования, «Система Хранения Данных (СХД)», [Электронный ресурс]. URL: http://hotuser.ru/shd (дата обращения: 25.06.2017).

2 EMC Corporation, “Introduction to the EMC VNX2 Series”, [Электронный ресурс]. URL: https://www.emc.com/collateral/white-papers/h12145-intro-new-vnx-series-wp.pdf — 6-7с. (дата обращения: 25.06.2017).

3 Techopedia, “Logical Unit Number (LUN)”, [Электронный ресурс]. URL: https://www.techopedia.com/definition/321/logical-unit-number-lun (дата обращения: 25.06.2017).

4 Справочный центр RU-TLD, “RAID”, [Электронный ресурс]. URL: https://ru-tld.ru/h/help_system:servera:raid:raid (дата обращения 25.06.2017).

5 EMC Corporation, “What is a ViPR Virtual Pool?”, [Электронный ресурс]. URL: https://www.emc.com/techpubs/vipr/what_is_virtual_pool-1.htm (дата обращения: 25.06.2017).

6 EMC Corporation, “ViPR Controller”, [Электронный ресурс]. URL: https://www.emc.com/products/storage/software-defined-storage/vipr-controller.htm (дата обращения: 25.06.2017).

7 EMC Corporation, “EMC ViPR REST API”, [Электронный ресурс]. URL: https://build.coprhd.org/jenkins/job/CH-coprhd-controller-master/ws/CH-coprhd-controller-master/build/gradle/tools/apidocs/apidocs/index.html (дата обращения: 25.06.2017).

8 EMC Corporation, “ViPR Controller Java Client”, [Электронный ресурс]. URL: https://community.emc.com/docs/DOC-33848 (дата обращения: 25.06.2017).

9 EMC Corporation, “ViPR SRM”, [Электронный ресурс]. URL: https://www.emc.com/data-center-management/vipr-srm.htm#!resources (дата обращения: 25.06.2017).

10 Pivotal Software, Inc., “Spring Framework Reference Documentation”, [Электронный ресурс]. URL: http://docs.spring.io/spring/docs/current/spring-framework-reference/pdf/spring-framework-reference.pdf (дата обращения: 25.06.2017).

11 Pivotal Software, Inc., “Spring Boot Reference Guide”, [Электронный ресурс]. URL: http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/pdf/spring-boot-reference.pdf (дата обращения: 25.06.2017).

12 Brown A. The Architecture of Open Source Applications, Volume II / Brown A., Wilson G. – Raleigh, North Carolina, United States: Lulu, 2008.

13 Atchison L. Architecting for Scale: High Availability for Your Growing Applications / Atchison L. – Sebastopol, CA, United States: O'Reilly Media, Inc, 2016.
14 Marcus E. Blueprints for High Availability / Marcus E., Stern H. – Indianapolis, Indiana: Wiley Publishing, 2003.
15 Oracle, Inc., "Oracle Database: High Availability Best Practices 11g" [Электронный ресурс]. URL: https://docs.oracle.com/cd/E11882_01/server.112/e10803.pdf (дата обращения: 25.06.2017).
16 Mehrabani, A. MongoDB High Availability / Mehrabani, A. – Birmingham, UK: Packt Publishing, 2014.
17 Dasadia, C. MongoDB Cookbook / Dasadia, C., Nayak, A. – Birmingham, UK: Packt Publishing, 2016.
18 Chodorow, K. MongoDB: The Definitive Guide / Chodorow, K. – Sebastopol, CA, United States: O'Reilly Media, Inc, 2013.
19 Hazelcast, Inc. "Hazelcast Documentation, version 3.8.2", [Электронный ресурс]. URL: http://docs.hazelcast.org/docs/3.8.2/manual/pdf/hazelcast-documentation-3.8.2.pdf (дата обращения 25.06.2017)