Правительство Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
**«Национальный исследовательский университет
«Высшая школа экономики»**

Факультет информационных технологий и вычислительной техники

Вычислительные комплексы системы и сети (230101)

Кафедра информационно-коммуникационных технологий

ДИПЛОМНЫЙ ПРОЕКТ

Разработка системы управления и репликации базы данных
сервера доменных имен BIND

Выполнил

Студент группы № С-104

Кириллов Тимофей Андреевич

Научный руководитель

Старший преподаватель
Курилов Игорь Дмитриевич

Консультант

Генеральный директор ЗАО «Флант»
Столяров Дмитрий Олегович

Москва, 2013

Аннотация

Данный дипломный проект посвящен проектированию и разработке системы управления и репликации базы данных сервера доменных имен BIND. Полученная система предоставляет базовый механизм для создания кластеров служб доменных имен без единой точки отказа.

Содержание

Введение

Актуальность. В современном мире информационных технологий все большую значимость приобретают Web-технологии в сети Интернет. Служба доменных имен (DNS) является важнейшим механизмом создания иерархичных пространств имен для разделения глобальной сети на домены. Благодаря этому механизму мы имеем связующее звено для адресации интернет-узлов с помощью удобных человеку доменных имен. Большинство современных сетевых служб в той или иной мере опираются на систему доменных имен. От доступности данной системы зависит функционирование важнейших сетевых сервисов, таких как WWW, электронная почта, IP-телефония, web-сервисы и т.д. Поэтому вопрос создания отказоустойчивой инфраструктуры для службы доменных имен был и будет актуальным пока существует сеть Интернет.

Управление DNS записями путем прямой работы с конфигурацией DNS-серверов – подверженный человеческим ошибкам, трудоемкий процесс, требующий специальных знаний об особенностях конкретного используемого программного обеспечения. Для эффективной работы с конфигурацией необходима система управления, автоматизирующая рутинные части процесса конфигурации и предоставляющая стандартизованный интерфейс web-сервиса. Такое решение впишется в современную сервис-ориентированную архитектуру и позволит в дальнейшем пользоваться им как отдельным сервисом или встроить его в какое-либо комплексное решение для управления сетевой инфраструктурой.

Целью данной дипломной работы является повышение эффективности и отказоустойчивости работы со службой доменных имен путем создания масштабируемой системы управления и репликации базы данных сервера доменных имен ISC BIND.

Задачи, которые были решены в этой работе:

- формирование требований к создаваемой системе;
- анализ существующих технологических решений и подходов к организации отказоустойчивых кластеров серверов доменных имен на соответствие сформированным требованиям;
- проектирование и разработка web-сервиса для удаленного чтения/записи в базу данных (далее БД) с учетом специфики хранимых данных;
- проектирование и разработка системы репликации удаленных БД, позволяющей объединять несколько (2 и более) инсталляций системы в единый кластер;
- разработка комплекса функциональных тестов, покрывающих основные функции для всех компонентов системы;
- внедрение системы в эксплуатацию.

Практическая значимость данной работы подтверждена успешным внедрением системы в работу компании.

Апробация работы. Реализованная система прошла апробацию в ЗАО «Флант» («Акт о внедрении системы управления и репликации базы данных сервера доменных имен BIND»).

На защиту представляется реализованная базовая часть системы, включающая:

1. Web-сервис интерфейса доступа к БД сервера доменных имен.
2. Систему репликации БД сервера доменных имен.

Также к защите предоставляются исходные коды разработанной системы под свободной лицензией.

1. Обзорно-аналитическая часть

1.1. Выявление требований к системе на основе анализа используюшихся в компании решений

Создаваемая система должна включать в себя:

1. Сервер доменных имен (далее DNS-сервер).
2. Система управления БД сервера DNS-сервера.
3. Система синхронизации БД.

1.1.1. Организация данных DNS-сервера

Использующаяся в компании схема разделения данных DNS-сервера включает в себя такие понятия, как: арена, сегмент, зона, запись.

Все пространство DNS-данных делится на арены. Арена имеет имя и ключ доступа. Арена является способом разграничения доступа пользователей системы.

Внутри арены все данные можно разбить на сегменты. Каждый сегмент является набором зон. Зона – набор записей.

Арены имеют уникальные имена. Сегменты имеют уникальные имена только в рамках одной арены. Зоны уникальны в рамках всей системы.

1.1.2. Постановка задачи

К системе предъявляются следующие требования:

1. Система управления БД:
    1. Создание и удаление пользователей.
    2. Авторизованный удаленный доступ.
    3. Иерархическое разделение данных DNS-сервера в соответствии с принятой в компании схемой.

4. Возможность внесения изменений в БД на лету, без необходимости прерывать работу DNS-сервера.

2. Кластер DNS-серверов:
    1. Мульти-мастерная структура кластера. Кластер состоит из одноранговых узлов, без единой точки отказа.
    2. Возможность добавления в уже работающий кластер нового узла.
    3. Возможность чтения и записи данных в БД любого из DNS-серверов, состоящих в кластере, с дальнейшим распространением новых данных между узлами.
3. Высокая производительность используемого DNS-сервера.
4. Архитектура системы должна быть расчитана на высокие нагрузки.
5. Использование открытых технологий и стандартов, программного обеспечения с открытым исходным кодом.
6. Наличие функциональных тестов, покрывающих все основные функции каждого из компонентов системы.

Предполагаются сценарии использования системы, в которых осуществляется одновременное внесение данных только в один из узлов кластера.

Необходимо провести анализ существующих решений и подходов к организации отказоустойчивых кластеров DNS-серверов и их соответствие приведенным требованиям.

Так как не ставится задача создания универсального решения, работающего с любыми существующими DNS-серверами, необходимо провести анализ и выбор сервера. Данный выбор определит используемую базу данных и повлияет на остальные части системы.

1.2. Анализ существующих решений и подходов к организации отказоустойчивых кластеров серверов доменных имен

1.2.1. Панель управления хостингом

Панель управления хостингом – это класс программного обеспечения, включающий в себя помимо прочего решение таких задач, связанных с DNS, как:

- управление DNS-записями через web-интерфейс;
- создание кластеров DNS-серверов.

Приведем перечень распространенных панелей управления хостингом, решающих данные задачи: cPanel, ISPConfig, Virtualmin, Webmin, Plesk.

1.2.1.1. Плюсы

- наличие web-интерфейса для управления данными DNS;
- возможность создания иерархии DNS-серверов и ее настройки посредством web-интерфейса;
- возможность использования различных DNS-серверов, без привязки к какому-либо конкретному программному продукту.

1.2.1.2. Минусы

- единое комплексное решение для хостинга, навязывающее специфичную для данной комплексной задачи инфраструктуру, не позволяющее пользоваться только функциями управления DNS, как отдельным сервисом.
- отсутствие механизмов создания мульти-мастерных DNS-кластеров без наличия единой точки отказа.

1.2.1.3. Резюме

Общим минусом всех панелей управления хостингом является их перегруженность функциями и сложность, а порой и невозможность

интеграции с существующей инфраструктурой компании. Данные системы предлагают единое комплексное решение для хостинга, навязывают инфраструктуру, а не предлагают хорошее решение какой-либо одной задачи из указанных в требованиях данной работы.

1.2.2. Microsoft Active Directory

Active Directory – это LDAP-совместимая служба каталогов компании Microsoft для операционных систем семейства Windows NT.

1.2.2.1. Плюсы

- развитая система доступа к сервису управления через протокол Kerberos;
- поддержка мульти-мастерной репликации между распределенными узлами Active Directory;

1.2.2.2. Минусы

- полная зависимость от платформы Microsoft Windows;
- невозможность использования сторонних DNS-серверов;
- сложность интеграции с существующей Unix инфраструктурой;
- закрытый исходный код программного продукта;
- высокая совокупная стоимость владения, характерная для использования любых продуктов компании Microsoft;

1.2.2.3. Резюме

Несмотря на то, что данное решение в значительной мере удовлетворяет заданным требованиям, завязанность на платформу Microsoft Windows и закрытый исходный код не позволят в дальнейшем самостоятельно решать при необходимости новые задачи, а также усложняют интеграцию с уже существующей в компании Unix инфраструктурой.

1.2.3. VitalQIP

VitalQIP – это программное решение управления доменами и серверами DNS и DCNP, сетями и подсетями IP.

1.2.3.1. Плюсы

- поддержка внешних хранилищ для данных DNS;
- поддержка различных DNS-серверов.

1.2.3.2. Минусы

- проприетарное решение;
- возможность использования в качестве внешних хранилищ только проприетарных СУБД Sybase и Oracle.

1.2.3.3. Резюме

VitalQIP является закрытым и довольно ограниченным в выборе БД решением. Использование таких масштабных решений, как Sybase или Oracle для нужд хранения данных DNS является неоправданным.

1.2.4. DNS zone transfer

DNS zone transfer является способом репликации данных DNS, позволяющим организовать иерархию серверов со связями главный-подчиненный (master-slave или primary-secondary). Стандартом на службу DNS RFC 1035 определен специальный вид запроса к DNS серверу AXFR. По данному запросу master-сервер отдает все данные, которыми располагает по запрошенному домену. Так называемый «трансфер зоны». Через этот запрос slave-сервер получает всю информацию с master-сервера по нужному домену. Обычно AXFR запросы разрешены только для небольшой группы IP, чтобы не показывать посторонним лицам лишнюю информацию (например существование секретных поддоменов).

1.2.4.1. Плюсы

- стандартизованный запрос AXFR, любой DNS-сервер, реализующий трансфер зоны, способен учавствовать в репликации;
- возможность участия в репликации разных программных продуктов, реализующих функции DNS-сервера в соответствии со стандартом.

1.2.4.2. Минусы

- иерархия master-slave;
- в случае отказа master-сервера, slave-сервер способен отвечать на DNS запросы в режиме только для чтения, т. е. возникает проблема наличия единой точки отказа.

1.2.4.3. Резюме

Данная конфигурация хороша для базовой конфигурации первичных и вторичных серверов, однако недостаточна для решения проблем данной работы.

1.2.5. Синхронизация конфигурации

Данные основных популярных DNS-серверов, таких как ISC BIND, NSD, MaraDNS, в наиболее производительной версии настройки описываются в конфигурационных файлах. После запуска DNS-сервера, конфигурационный файл прочитывается в память, и сервер готов отвечать на запросы. Чтобы перенести конфигурацию на другой DNS-сервер, достаточно настроить синхронизацию файлов конфигурации и реализовать механизм оповещения для перечитывания обновленной конфигурации в случае ее изменения.

Общепринят подход с использованием программы rsync (Remote Synchronization) – это программа для UNIX-подобных систем, которая выполняет синхронизацию удаленных файлов и каталогов с минимизированием трафика, используя кодирование данных при необходимости.

1.2.5.1. Плюсы

- простота и прозрачность схемы;
- использование родного формата конфигурационных файлов позволяет использовать DNS-сервер на максимальной производительности;

1.2.5.2. Минусы

- сложность построения полноценной мульти-мастерной архитектуры;
- возрастание сложности настройки с добавлением новых узлов в кластер;
- использование файлов конфигурации привносит проблему долгой перезагрузки DNS-сервера после внесения изменений;
- необходимость самостоятельной реализации принципов ACID при работе с файлами конфигурации.

1.2.5.3. Резюме

Данная схема организации отказоустойчивого кластера DNS-серверов не позволит добиться внесения изменений в работу сервера на лету, из-за постоянной необходимости перезагружать файлы конфигурации. Также дополнительные сложности влечет необходимость создания механизма совместного доступа к конфигурации для работы совместно с системой управления DNS-данными и сложность реализации принципов ACID при работе с данными.

1.2.6. ISC BIND

BIND (Berkeley Internet Name Daemon) – открытая и наиболее распространенная реализация DNS-сервера, поддерживаемая организацией Internet Sysmtems Consortium. 10 из 13 корневых серверов DNS работают на BIND. Данный сервер является старейшим и наиболее проработанным программным продуктом, ставшим стандартом де-факто на DNS-сервера.

Однако стандартная версия BIND не лишена следующих недостатков:

- Сервер считывает DNS-данные из конфигурационного файла, в которых легко допустить ошибку, что делает внесение изменений ответственным процессом.
- BIND считывает все DNS-данные в оперативную память. Авторитетный сервер, отвечающий за большое количество зон, требует большого количества оперативной памяти.
- Процесс чтения массивных конфигурационных файлов может занимать большое количество времени, что делает невозможным обновление DNS-данных «на лету».

Для решения данных проблем было создано расширение для BIND под названием DLZ (Dynamically Loadable Zones). DLZ позволяет хранить DNS-данные в полноценной базе данных. Внесение изменений в БД не требует перезагрузки DNS-сервера, таким образом решена проблема долгого обновления DNS-данных.

На выбор предоставляются следующие базы данных: PostgreSQL, MySQL, Berkeley DB, ODBC-совместимый драйвер (Firebird, DB2, Oracle, Sybase, SAPDB), LDAP.

1.2.6.1. Оценка производительности при использовании различных БД

В следующей таблице приведено сравнение производительности при использовании различных БД и при использовании конфигурационных файлов:

Потоки		Ядро Linux 2.6.1									
		BIND	Postgres	MySQL	LDAP	FileSystem	Berkeley DB				
							BTREE	HASH	HPT-T	HPT-C	HPT-P
1	Запросов в секунду	16470	552	618	90	271	1057	956	4285	5890	8488
	Время запуска	647	1	1	483	2	2	1	0	0	0
2	Запросов в секунду	15570	673	–	91	244	1247	1117	4081	4749	5178
	Время запуска	1441	2	–	192	1	1	1	1	1	1

Таблица 1: сравнение прозводительности БД при использовании BIND DLZ

Видно, что при использовании конфигурационных файлов достигается наибольшая производительность. Следующей по производительности идет база данных Berkeley DB в различных режимах работы. Реляционные СУБД показали на порядки меньшую пропускную способность.

Использование Berkeley DB – высокопроизводительной базы данных для хранения данных в виде ключ-значение – является наиболее оправданным и дает проигрыш в производительности всего в 2 раза. Данный проигрыш можно компенсировать распределением нагрузки между узлами кластера.

1.2.6.2. Плюсы

- использование BDB вместо конфигурационных файлов для хранения DNS-данных позволит добавлять новые данные «на лету» и сделает перезагрузку DNS-сервера не зависящей от количества данных;
- использование BDB не существенно снижает производительность сервера;
- поддержка в BDB совместной работы нескольких процессов упростит

создание приложения репликации и системы управления БД.

1.2.6.3. Минусы

- жесткая привязка к DNS-серверу ISC BIND;
- необходимость создавать собственный механизм мульти-мастерной репликации БД;
- необходимость создавать собственную систему управления БД;

1.2.6.4. Резюме

Построение системы на основе использования DNS-сервера BIND DLZ в связке с Berkeley DB – позволит создать надежное и масштабируемое решение.

Привязка к одному DNS-серверу в данном случае не является существенным недостатком, данный сервер проверен временем и наиболее полно поддерживает стандарты DNS. Также в требованиях работы явно указано об отсутствии необходимости создавать универсальное решение.

1.3. Выводы

В таблице 2 (стр. 19) предоставлены результаты анализа существующих решений и подходов к организации кластеров DNS-серверов.

	Панель управления хостингом	Microsoft Active Directory	VitalQIP	DNS zone transfer	Синхронизация конфигурации	BIND DLZ
Система управления DNS-данными	+	+	+	*	*	*
Авторизованный удаленный доступ к системе управления	+	+	+	*	*	*
Внесение изменений «на лету»	+-	+	+-	*	-	+
Универсальность в выборе DNS-сервера	+	-	-	+	-	-
Возможность использования системы управления как web-сервиса	-	+	-	*	*	*
Открытый исходный код	+-	-	-	+	+	+
Мультимастерная репликация DNS-данных	-	+	-	-	+-	*
Простота интеграции с существующей инфраструктурой	-	-	-	+	+	+

Таблица 2: результаты анализа существующих решений

Условные обозначения таблицы:

- «+» – требование выполнимо;
- «+-» – требование частично выполнимо;
- «-» – требование невыполнимо;
- «*» – требование не выполняется, однако его возможно выполнить путем

дополнительной разработки.

Ни одно из рассмотренных готовых решений и подходов к организации кластеров DNS-серверов не удовлетворяет всем требованиям, предъявляемым в данной работе.

Однако, в ходе исследования было найдено решение с открытым исходным кодом, предоставляющее важнейший базовый механизм интеграции DNS-сервера с СУБД – BIND DLZ. Было принято решение создания собственной системы управления и репликации базы данных Berkeley DB на основе использования DNS-сервера BIND с расширением DLZ, которое позволит в полной мере удовлетворить всем требованиям данной работы.

2. Технологическая часть

2.1. Обзор сервис-ориентированной архитектуры программных систем

Требования, предъявляемые к системе управления БД DNS-сервера включают в себя:

1. Простоту интеграции в инфраструктуру.
2. Интерфейс, ориентированный на работу с системой по сети, скрывающий детали реализации системы (операционную систему, платформу, язык программирования).
3. Автономность системы, независимость от других сетевых сервисов.

Таким образом к системе управления предъявляются требования как к сервису в рамках сервис-ориентированной архитектуры.

Сервис-ориентированная архитектура – это принцип разработки программного обеспечения, основанный на использовании отдельных сетевых сервисов, выполняющих в совокупности функции какого-либо сложного приложения.

Требования предъявляемые к сервис-ориентированным платформам:

1. Сервисы придерживаются соглашения о взаимодействии, определенного в документе в стандартизованном формате описания сервисов и доступа к ним.
2. Слабая связность сервисов между собой.
3. Сервис предоставляет абстракцию, строго определенную интерфейсом взаимодействия и скрывает детали реализации.
4. Повторное использование сервисов. Сервисы создаются как механизмы в меру независимые и абстрагированные от деталей бизнес-логики, для простоты их дальнейшего использования в различных контекстах.
5. Автономность сервисов:

1. Автономность архитектуры – предполагает возможность эволюционирования сервиса с минимальным влиянием на его пользователей.
2. Автономность времени выполнения – независимость сервиса от среды, в которой он работает. Данный тип автономности достигается путем предоставления сервису выделенных ресурсов, локальных копий данных, кэшей и т. п.

6. Минимизация или полное отсутствие «состояний» при работе с сервисом.
7. Сервис имеет средства для его автоматического обнаружения в сети.
8. Компонуемость сервисов – предполагает возможность создания нового приложения путем повторного использования уже существующих сервисов с минимальными усилиями.

Сервис-ориентированная архитектура позволяет абстрагироваться от таких деталей реализации, как операционная система, платформа и язык программирования.

Перечень основных открытых технологий и протоколов, использующихся для сетевого взаимодействия с сервисами, включает в себя: SOAP, CORBA, REST.

2.1.1. SOAP

SOAP (Simple Object Access Protocol) – протокол обмена структурированными сообщениями в распределенной вычислительной среде. Данный протокол является одним из стандартов для организации взаимодействия web-служб.

SOAP может быть использован для удаленного вызова процедур или передачи произвольных сообщений. Для описания сообщений используется формат XML.

Протокол не привязан к какому-либо конкретному протоколу транспортного уровня. В качестве протокола транспортного уровня могут выступать: HTTP,

HTTPS, SMTP, FTP, JMS и другие.

SOAP является так называемым «stateful» протоколом. Это подразумевает, что сервер хранит информацию о клиентах и использует ее в рамках сессии между клиентом и сервером.

Достоинства:

- Независимость от транспортного уровня.
- Возможность как передачи данных, так и удаленного вызова процедур.
- Отсутствие проблем с работой через сетевые экраны и прокси-серверы за счет использования стандартных транспортных протоколов.
- Строгая типизация данных, гарантирующая их целостность при передаче между клиентом и сервером.

Недостатки:

- Использование языка разметки XML:
  - большой объем передаваемых данных;
  - плохое восприятие человеком (по сравнению с существующими альтернативами XML);
  - неоправданная программная сложность синтаксического анализа и разбора сообщений;
- Сложность и тяжеловесность спецификации протокола.
- Трудоемкость отлаживания протокола в неоднородной программной среде.

2.1.2. CORBA

CORBA (Common Object Request Broker Architecture) – обобщенная архитектура брокера объектных запросов – это технологический стандарт написания

распределенных приложений.

CORBA позволяет отдельным приложениям, написанным на различных языках программирования и запущенных на разных компьютерах работать друг с другом как единое приложение. Данная архитектура позволяет обобщить и нормализовать семантику вызова методов между объектами приложения независимо от того, находятся они в одном адресном пространстве (внутри одного приложения) или разделены сетью. Спецификация CORBA устанавливает принципы создания брокеров объектных запросов (далее ORB), которые и допускают такую интеграцию.

Запрос посылается от клиента к серверу. Клиент – это приложение, или нечто другое, выполняющее операцию над объектом, а реализация объекта – это код и данные, которые на самом деле выполняют эту операцию. ORB способен выполнить все действия, необходимые для нахождения реализация указанного объекта, подготовке этой реализации к обработке запроса и передаче данных, относящихся к запросу.

Ядро ORB обеспечивает основные механизмы для манипуляций объектами и выполнения запросов. Система объектов обеспечивает клиента набором сервисов. Клиент способен запросить некоторый сервис. Объект – это нечто, что обеспечивает один или более сервисов, которые клиент может запросить.

За исключением редкого случая прямых вызовов методов между классами одного и того же языка программирования необходим механизм кодирования вызова метода в некоторую последовательность байт у клиента и декодирования этой последовательности у сервера. Для этой цели спецификация CORBA определяет общий протокол обмена между брокерами объектных запросов (General Inter-Orb Protocol - GIOP). Кроме того, определен протокол передачи сообщений протокола GIOP поверх транспортного протокола TCP/IP, являющегося основным видом взаимодействия в сети Интернет, ввиду чего этот протокол получил название протокола обмена между брокерами объектных

запросов в сети Интернет (Internet Inter-Orb Protocol — IIOP).

Достоинства:

- Бинарный протокол передачи данных позволяет добиться большой скорости работы за счет меньшего количества передаваемых данных.
- CORBA предоставляет полноценную платформу для создания распределенных приложений, включающую:
  - протоколы передачи данных;
  - требования к качеству обслуживания (полоса пропускания, задержки, надежность и т. д.);
- Возможность гибкой и прозрачной компоновки взаимодействия между объектами, не зависящей от их местоположения (в рамках одного приложения или по сети).

Недостатки:

- Использование собственных протоколов взаимодействия между брокерами ведет к проблемам с работой через сетевые экраны и прокси-серверы.
- Сложность и тяжеловесность спецификации протокола.
- Использование бинарного протокола передачи данных ведет к трудоемкости отладки сетевого взаимодействия в неоднородной программной среде.
- Сложность интеграции в уже существующую инфраструктуру сервисов.

2.1.3. REST

REST (REpresentational State Transfer) – метод построения приложений в рамках сервисно-ориентированной архитектуры.

REST архитектура условно состоит из клиентов и серверов. Клиенты

инициируют запросы к серверам, сервера обрабатывают запросы и возвращают соответствующие результаты. В запросах и ответах передаются представления ресурсов. Ресурсом может быть любая имеющая смысл концепция. Представлением ресурса обычно является документ, в котором описано текущее состояние ресурса. Представление ресурса может содержать ссылки, которые могут быть использованы клиентом для перевода ресурса в другое состояние. Ярким примером распределенной системы построенной по данным принципам является всемирная паутина (World Wide Web).

Для работы с ресурсами предоставляются операции соответствующие базовым операциям работы с хранилищем данных (CRUD):

- Создание – HTTP POST;
- Чтение – HTTP GET;
- Обновление – HTTP PUT;
- Удаление – HTTP DELETE.

Обобщая вышесказанное можно сформулировать следующие принципы, которым должны следовать правильно спроектированные REST-сервисы:

1. Агенты пользователей взаимодействуют с ресурсами, которыми может быть всё, что можно поименовать и представить. К каждому ресурсу можно обратиться посредством уникального идентификатора URI (Uniform Resource Identifier – универсальный код ресурса).
2. Взаимодействие с ресурсами (обнаруживаемыми посредством их уникальных кодов URI) осуществляется с помощью единого интерфейса стандартных команд HTTP (GET, POST, PUT и DELETE). Для взаимодействия важно также объявить тип мультимедийного ресурса, который указывается с помощью заголовка типа содержимого HTTP (XHTML, XML, JPG, PNG, YAML, JSON – некоторые хорошо известные мультимедийные типы).

3. Ресурсы описывают себя сами. Вся информация, необходимая для обработки запроса ресурса, содержится в самом запросе (это позволяет обходиться без характеристики состояния служб).
4. В ресурсах содержатся ссылки на другие ресурсы (гиперсреда).

Достоинства:

- Возможность эффективного кэширования и масштабирования под требования высокой нагрузки.
- Команды PUT и DELETE являются идемпотентными, т. е. повторное выполнение одной и той же команды PUT или DELETE переводят ресурс в тоже самое состояние. Поэтому сервис остается устойчив при ошибках и сбоях в сети, приводящих к многократному выполнению одной и той же команды.
- Легкость интеграции с web-системами и функциональная совместимость. Для работы с REST-сервисом необходима лишь доступность библиотеки, реализующей протокол HTTP.

Недостатки:

- Отсутствие стандартного способа обнаружения сервисов в сети.
- Отсутствие стандартного способа передачи правил работы с сервисами.
- Отсутствие стандартных средств интеграции сервиса со средствами разработки.
- Отсутствие стандартной проверки типов данных параметров запросов.

2.1.4. Выводы

С учетом вышеприведенных требований наиболее подходящим, гибким и отвечающим современным требованиям подходом является REST-сервис. Гибкость и приспосабливаем протокола HTTP доказана долгим существованием

такого сервиса, как веб.

2.2. Обзор методов и подходов к организации репликации баз данных

Репликация баз данных – это механизм автоматической синхронизации содержимого нескольких (2 и более) копий одной и той же базы данных. Любая из баз данных, входящая в кластер далее называется репликой или узлом. Данный механизм преследует следующие цели:

1. Повышение надежности сервисов, которые опираются на использование БД за счет резервирования одних и тех же данных.
2. Распределение нагрузки на БД.
3. Возможность использования выделенной реплики для таких целей, как создание резервной копии БД и т. п.
4. Возможность использования отличной структуры хранения одних и тех же данных на соседних репликах (например, разный тип таблиц позволяет приспособить реплики для разных типов запросов к данным).

По принципу организации кластера БД репликация делится на ведущий-ведомый (master-slave) и мульти-мастер. По способу распространения данных репликация БД бывает синхронная и асинхронная.

Наиболее предсказуемая и надежная работа системы управления БД обеспечивается при соблюдении требований ACID. В соответствии с данными требованиями транзакционная система должна обладать:

1. Атомарностью (Atomicity) – гарантией, что никакая транзакция не будет зафиксирована в системе частично. Возможны 2 варианта выполнения транзакции: либо будут выполнены все ее подоперации, либо не будет выполнено ни одной.
2. Согласованностью (Consistency) – гарантия согласованности данных в БД до транзакции и после ее завершения.

3. Изолированность (Isolation) – гарантия того, что во время выполнения транзакции, параллельные транзакции не окажут влияния на ее результат.
4. Надежность (Durability) – гарантия того, что после завершения транзакции все изменения вступили в силу и не могут быть отменены вследствие какого-либо сбоя.

Далее будут подробно рассмотрены основные применяющиеся подходы к репликации.

2.2.1. Асинхронная репликация

При асинхронной репликации узел, вносящий изменения в БД, сразу производит их фиксацию. Все изменения записываются в специальный журнал. Удаленный узел кластера может запросить, получить и применить у себя изменения в БД из данного журнала. При данном подходе в журнал данного узла пишутся только изменения, внесенные непосредственно в этот узел, а не полученные из журнала другого узла кластера.

Реализация асинхронной репликации зависит полностью от правильной логики приложения, т. к. в данном режиме невозможно соблюсти требование согласованности данных.

Преимущества:

- Производительность операций записи не зависит от количества узлов в кластере.
- Небольшое количество сетевого трафика во время транзакций.

Недостатки:

- Неполное соответствие принципам ACID.
- Задержка перед появлением внесенных данных на других узлах.

2.2.2. Синхронная репликация

Синхронная репликация позволяет вносить изменения в БД сразу на всех узлах кластера. В основе данной репликации лежит двухфазный сетевой протокол.

На первой фазе данный узел рассылает всем остальным запрос на внесение изменений. Каждый узел получивший запрос проводит так называемую сертификацию вносимых данных. Данный процесс предполагает проверку возможности внесения изменений, отсутствие конфликтных ситуаций. После проведения сертификации узел отсылает соответствующий ответ.

Если получено подтверждение от всех узлов, начинается вторая фаза: отсылается команда на фиксацию запроса. Если на первой фазе хотя бы один из узлов не подтвердил сертификацию, то на второй фазе рассылается отмена запроса.

Таким образом, соблюдаются принципы ACID на всех узлах кластера.

Преимущества:

- Внесенные изменения фиксируются одновременно на всех узлах кластера.

Недостатки:

- Большое количество сетевого трафика по обмену подтверждениями.
- Существенное снижение производительности операций записи в БД, поскольку операция может считаться законченной только в случае, когда каждый из узлов подтвердит выполнение транзакции.

2.2.3. Режим ведущий-ведомый

В данном режиме в исходная база данных доступна для чтения и записи и является первичным авторитетным источником данных. Остальные реплики хранят те же данные, но доступны в режиме только для чтения.

Данный подход позволяет разделить нагрузку записи и чтения на ведущие и

ведомые узлы соответственно. Роль ведущих и ведомых узлов может меняться динамически без остановки системы управления БД. Однако в один момент времени возможен лишь один ведомый узел.

Распределение данных с ведущего сервера на ведомый может проходить в синхронном или асинхронном режимах.

Преимущества:

- Отсутствие конфликтных ситуаций при внесении новых данных, т. к. данные вносятся лишь в один из узлов кластера.
- Возможность эффективного масштабирования при увеличении нагрузки чтения данных.

Недостатки:

- Невозможность масштабирования при увеличении нагрузки записи данных.

2.2.4. Режим мульти-мастер

В режиме мульти-мастера любой из узлов кластера доступен для чтения или записи данных. Каждый ведущий узел скачала выполняет запрос у себя, затем синхронизирует его на других ведущих узлах.

Основная проблема, возникающая при мульти-мастерной репликации – синхронизация данных во время или после внесения данных в БД. В зависимости от способа распространения данных – синхронного или асинхронного – данная проблема решается по-разному.

2.2.5. Асинхронный мульти-мастер

Изменения вносятся в БД без немедленного обновления на других узлах. Во время обновления данных могут возникнуть конфликтные ситуации, которые можно разделить на 3 типа:

1. конфликт обновления;

2. конфликт уникальности;
3. конфликт удаления.

Первый тип ошибок происходит в том случае, когда локальное обновление какого-то объекта опережает по времени отложенное удаленное обновление этого же объекта, т. е. локальное обновление объекта происходит раньше, чем приходит запрос на его обновление от другого узла. В результате нарушается правильная последовательность событий для этого объекта.

Второй тип ошибок возникает в том случае, когда происходит создание дубля для уникального объекта.

Третий тип ошибок аналогичен первому: выполняется операция над глобальным объектом, который уже удален на другом узле, но данные между узлами еще не синхронизированы.

Алгоритм разрешения конфликтных ситуаций определяется конкретной реализацией системы управления БД. Однако, все алгоритмы придерживаются общего принципа. Сначала система пытается разрешить конфликтную ситуацию автоматически, следуя правилам, заданным администратором. В случае невозможности разрешения конфликта – система об этом сигнализирует и требуется ручное вмешательство администратора.

Автоматическое разрешение конфликтов также может проводиться по следующим принципам:

- Каждый сервер получает уникальный приоритет. Серверы с более высоким приоритетом "выигрывают" у серверов с более низким приоритетом.
- Временные метки: корректной считается наиболее старая или наиболее молодая транзакция, участвующая к конфликте.
- Разделение данных: данные делятся между серверами так, что

конкретный сервер может обновлять лишь свою часть данных. Например, с помощью разделения диапазона ID записей таблиц.

2.2.6. Синхронный мульти-мастер

В случае синхронного мульти-мастера возможны конфликты при одновременном выполнении транзакций над одними и теми же данными. Данные конфликты полностью разрешаются в автоматическом режиме на этапе сертификации вносимых данных в рамках двухфазного сетевого протокола.

2.2.7. Выводы

В соответствии с требованиями к серверу синхронизации БД необходимо реализовать мульти-мастерную схему репликации. Т. к. отдельным требованием стоит возможность простого добавления произвольного количества новых узлов, наиболее подходящим типом распространения изменений между БД является асинхронная репликация.

Для разрешения конфликтных ситуаций необходимо предусмотреть механизм проверки возможности внесения изменений. Этот механизм должен допускать одновременное изменение данных разных зон. Система должна оповещать о наличии неразрешимых конфликтов.

2.3. Обзор методики функционального тестирования программных систем

Функциональное тестирование – это тестирование ПО в целях проверки реализуемости функциональных требований, то есть способности ПО в определённых условиях решать задачи, нужные пользователям. Функциональные требования определяют, что именно делает ПО, какие задачи оно решает.

В рамках данной методики тестирования создаются наборы тестов, которые проверяют поведение системы и соответствие результата ее работы некоторому желаемому результату. Функциональное тестирование рассматривает заранее

указанное поведение и основывается на анализе спецификаций функциональности компонента или системы в целом.

Основное преимущество функционального тестирования состоит в том, что оно имитирует фактическое использование системы и позволяет на ранних этапах выявить проблемы, которые могут возникнуть при ее реальном использовании.

Данный вид тестирования позволит быстро развивать работающую систему и гарантировать ее работоспособность по мере развития.

В соответствии с выбранными технологиями и принципами работы систем управления и синхронизации БД можно выявить следующие требования к автоматизированным тестам:

1. Тесты системы управления БД должны:
    1. Осуществлять проверку корректности всех операций внесения изменений в БД.
    2. Генерировать тестовые данные на лету.
    3. Осуществлять проверку безопасности доступа к БД.
2. Тесты сервера синхронизации БД должны:
    1. Генерировать набор тестовых данных на главном тестируемом сервере.
    2. Осуществлять проверку распространения данных на остальные сервера кластера и их корректность.
    3. Иметь возможность запуска нескольких экземпляров тестов с разными главными серверами для проверки механизма разрешения конфликтов.
    4. Поддерживать режим стресс-тестирования. В данном режиме должна периодически производится некорректная остановка и перезапуск произвольного сервера кластера в произвольном порядке.

2.4. Обзор модели асинхронного программирования

Асинхронное программирование – это метод создания серверных приложений, в задачи которых входит обработка большого количества одновременных запросов.

Данный метод способен повысить эффективность программ, выполняющих большое количество операций ввода-вывода, таких как:

- сетевое взаимодействие;
- взаимодействие с постоянным хранилищем данных.

В таких программах при традиционном блокирующем подходе процессор большую часть времени простаивает, ожидая завершения операций ввода-вывода: чтение запроса от клиента, обращение к диску за данными, сетевые обращения к другим подсистемам (БД, кэширующие сервера, сетевые сервисы и т. п.), запись ответа клиенту. Во время этих операций ввода-вывода процессор можно загрузить обработкой запросов других клиентов.

Асинхронное программирование в своей базовой форме представляет собой использование однопоточной модели вычислений с использованием примитивов асинхронного ввода-вывода, предоставляемых операционной системой (select, poll, epoll, kqueue, i/o completion port). Однако программирование существенно усложняется, т. к. данные из сетевых сокетов поступают некоторыми «отрывками», причем за один цикл обработки данные поступают от разных соединений, находящихся в разных состояниях, часть соединений могут быть входящими от клиентов, часть – исходящими к внешним ресурсам (БД, другой сервер и т. п.).

В случае, если программе требуется, помимо прочего, выполнять вычислительные процедуры или блокирующийся ввод-вывод (когда не существует асинхронного аналога), обычно используют возможность многопоточного выполнения программ, предоставляемую современными

операционными системами. Поток – это наименьшая единица обработки, исполнение которой может быть назначено ядром операционной системы. Несколько потоков выполнения могут существовать в рамках одного и того же процесса и совместно использовать ресурсы, такие как память. Главный поток программы может поручить вычислительную или блокирующуюся операцию одному из предварительно созданных потоков (свободному) и продолжить обработку запросов. Как только операция завершена, подчиненный поток предоставляет результат главному и снова становится свободным.

Оба описанных подхода являются стандартными архитектурными паттернами проектирования программного обеспечения: Reactor и Proactor.

Т. к. к системе предъявляется требование стабильной работы под высокими нагрузками, рассмотренный подход асинхронной архитектуры является наиболее подходящим. В системе предполагается работа с встраиваемой базой данных Berkeley DB. Библиотеки, реализующие данную базу данных являются блокирующими, поэтому архитектура создаваемой системы должна соответствовать паттерну Proactor.

2.5. Выбор инструментария для реализации системы

2.5.1. Язык программирования

На сегодняшний день лидирующими языками в области создания сетевых сервисов являются: Python, Ruby, C, C++, Java, C#, Perl.

Одним из наиболее подходящих под требования открытости стандартов, спецификаций, платформы и библиотек является язык Python (в своей работе данным языком пользуются: Google, Yahoo, Red Hat, NASA, Nokia, IBM и другие). В качестве альтернативных вариантов можно рассмотреть языки C и C++. Каждый из этих языков является компилируемым и дает неоспоримое преимущество в скорости работы программ. Однако разработка с использованием этих языков требует больших временных затрат и повышает

сложность создания прототипа системы. Язык Python является одним из популярных интерпретируемых языков. Данный язык располагает большим количеством открытых библиотек и фреймворков, активно поддерживаемых сообществом, и идеально подходит для прототипирования больших программных систем. Также при использовании данного языка имеется возможность переписать критические по времени выполнения компоненты системы на язык C и интегрировать эти компоненты с уже существующим кодом на Python.

2.5.2. Proactor фреймворк

Для языка Python существует большое число готовых фреймворков и библиотек для поддержки концепции асинхронного программирования и реализации паттерна Proactor. Среди них: Concurrence, Eventlet, Tornado, Twisted, asyncore, gevent, circuits, Syncless.

	Concurrence	Eventlet	Tornado	Twisted	asyncore	gevent	circuits	Syncless
Возможность работы без компилирования модулей на C	-	+	+	+	+	-	+	-
Максимальная производительность достигается без компилирования модулей на C	-	+	+	+	+	-	+	-
Является стандартным модулем Python	-	-	-	-	+	-	-	-
Имеет встроенный асинхронный DNS клиент	-	+	-	+	-	+	-	+
Поддерживает таймауты для отдельных socket.send() и socked.recv() операций	+	+	-	+	-	+	-	+
Имеет WSGI сервер	+	+	+	+	-	+	+	+
Возможность запуска через Stackless Python	+	+	+	+	+	-	+	+
Реализация класса сокета на C	-	-	-	-	-	-	-	+

Поддержка SSL	-	+	-	+	-	+	+	+
Автоматический перехват ошибок	+	+	+	+	+	+	-	-
Возможность использования библиотеки libevent	+	+	-	+	-	+	-	+
Возможность использования высокопроизводительных примитивов асинхронного программирования (epoll, kqueue и др.)	+	+	+	+	-	+	+	+
Применяется в эксплуатации	+	+	+	+	+	+	+	-
Наличие сообщества (списки рассылки, irc)	+	+	+	+	-	+	+	-
Наличие недавней активности разработки	-	+	+	+	-	+	+	+
Поддержка пула потоков	-	+	-	+	-	-	+	+
Unit тестирование	+	+	-	+	+	+	+	+
Имеет реализацию протоколов прикладного уровня	+	-	+	+	-	-	+	-
Совместимость с другими фреймворками и библиотеками	-	+	-	+	-	-	+	+

Таблица 3: сравнение фреймворков и библиотек асинхронного программирования

Из приведенной таблицы видно, что наиболее развитым и широко поддерживаемым фреймворком является Twisted.

2.5.3. Работа с БД

Для работы с Berkeley DB из языка Python существует стандартная библиотека bsddb3.

2.5.4. Система контроля версий

В настоящее время для разработки любого программного обеспечения принято использовать систему контроля версий (СКВ). СКВ позволяет хранить полную историю развития проекта, несколько версий одних и тех же файлов, вести

совместную разработку.

Одной из наиболее распространенных СКВ является Git (среди пользователей данной СКВ: Linux Kernel, OpenVZ, udev, DragonFly BSD, Bacula, X.Org, GCC, GNU coreutils, KDE, GNOME и т. д.). Git идеально подходит для разработки данной системы.

3. Разработка

3.1. Общая архитектура

Обобщенная схема взаимодействия компонентов системы с ее окружением представлена на рисунке 1, стр.40.

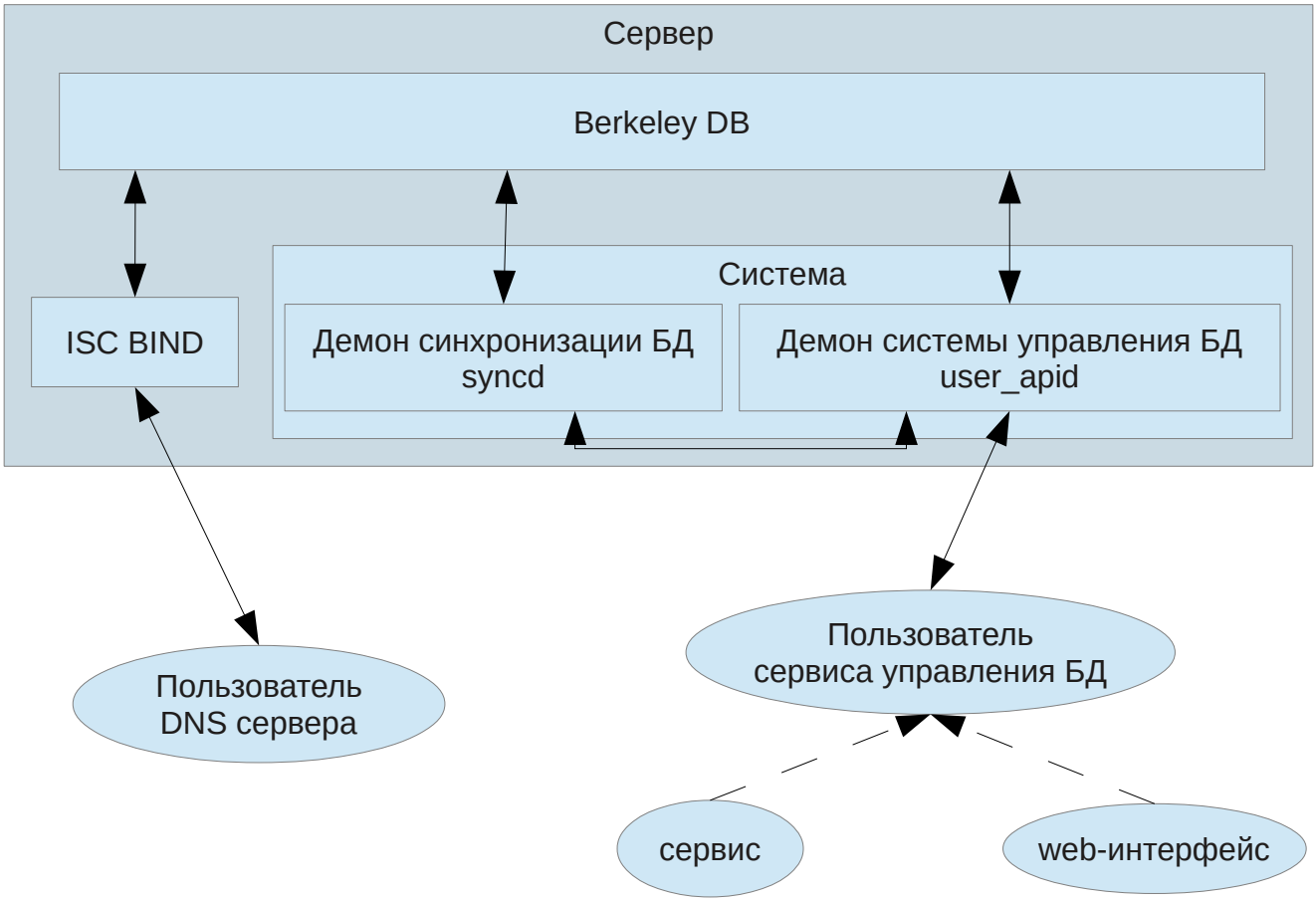

Рисунок 1: общая архитектура системы

Система реализует 2 слабосвязанные функции: синхронизация БД и интерфейс доступа к БД. Поэтому было решено разделить систему на 2 отдельных сервера:

1. Демон синхронизации БД – syncd. Взаимодействует с удаленными демонами синхронизации в соответствии с конфигурацией кластера.
2. Демон системы управления БД – user_apid. Предоставляет интерфейс web-сервиса.

Встраиваемая БД Berkeley DB представляет собой библиотеку, компонуемую с приложением для осуществления доступа к файлам БД на локальной файловой

системе. Поэтому все компоненты системы должны быть расположены на одном физическом сервере. Взаимодействие между процессами серверов происходит средствами межпроцессного взаимодействия, предоставляемыми операционной системой.

В качестве пользователя сервиса системы управления БД может выступать как web-интерфейс, так и другой сервис.

3.2. Сервер синхронизации БД

Схема взаимодействия серверов синхронизации БД представлена на рисунке 2, стр. 41.

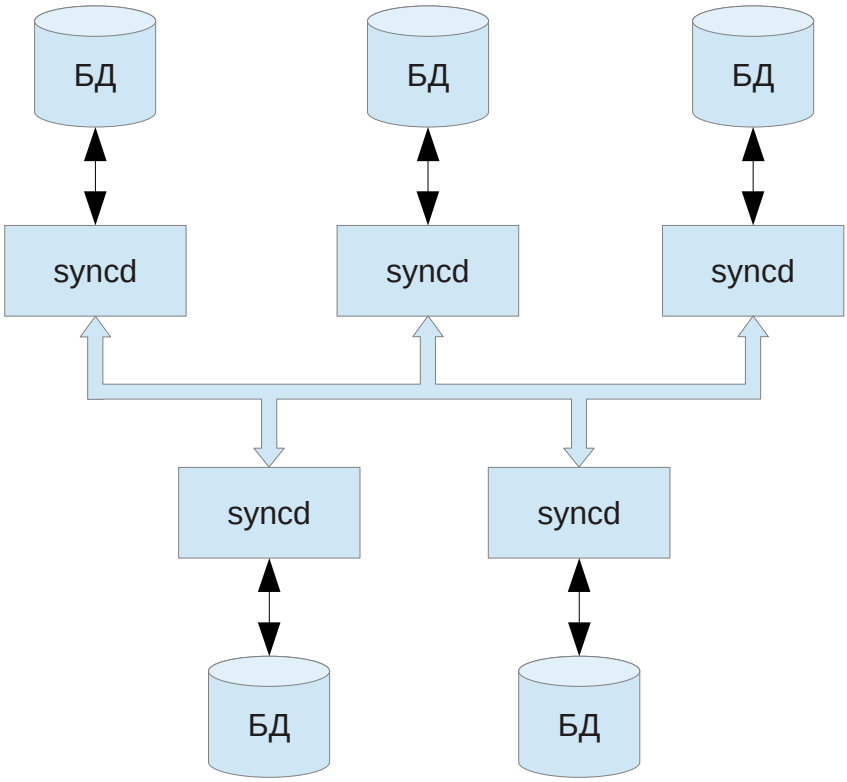

Рисунок 2: взаимодействие серверов синхронизации

3.2.1. Механизм репликации

Сервер реализует алгоритм асинхронной мульти-мастерной репликации. Поддерживается произвольное число участников кластера.

3.2.1.1. Действие

Для представления любого изменения в БД введено такое понятие, как **действие**. Действие представляет собой набор данных и алгоритм для его применения. Например, для создания DNS-записи необходимо действие, включающее в себя все данные записи и алгоритм для внесения этих данных в таблицы, для удаления записи – включающее в себя идентификационные данные записи и алгоритм удаления данных из таблиц и т. п. Выполнение алгоритма действия называется далее **применением действия**.

Реплицировать можно только те изменения в БД, которые представимы в виде действия. Т. е. репликация происходит не на уровне строк таблиц, действия представляют собой более высокоуровневый слой абстракции работы с БД и могут включать в себя одновременное добавление или удаление сразу нескольких строк из нескольких таблиц.

Berkeley DB в полной мере поддерживает требования ACID (см. п. 4.2.). С точки зрения транзакционности, действия должны поддерживать:

1. Применение одного действия в индивидуальной транзакции.
2. Применение нескольких действий в одной транзакции.
3. Применение действия с игнорированием транзакций.

Помимо применения к БД действия будут использоваться:

1. Для ведения журнала изменения БД.
2. Для реплицирования изменений на удаленные сервера.

Поэтому любое действие должно поддерживать **сериализацию**. Сериализация – это процесс перевода какой-либо структуры данных в последовательность байтов. Эта последовательность представляет собой либо текстовую информацию, либо неразборчивую для человека бинарную информацию, по которой можно полностью восстановить исходную структуру данных. Для эффективного использования памяти и быстроты передачи

действий по сети должна использоваться бинарная сериализация данных.

В качестве формата сериализации подходящим является формат BSON (бинарный JSON), использующийся для хранения и передачи данных в СУБД MongoDB.

3.2.1.2. Журнал действий

Для сохранения всех примененных на данном сервере действий необходим **журнал действий**. Журнал действий представляет собой непрерывную таблицу примененных действий (таблица action, рисунок 7, п. 3.4). Каждая запись журнала состоит из идентификатора записи, называемого **позицией**, и сериализованного действия.

Пространство идентификаторов записей журнала у каждого из серверов индивидуальное. В журнал попадают только те действия, которые применяются на данном узле кластера, и не попадают действия, реплицированные из удаленных узлов.

Каждый из серверов кластера хранит соответствие **актуальной позиции** каждого из удаленных серверов в **таблице соответствия** позиций (таблица peer, рисунок 7, п. 3.4). Соответствие позиции X серверу Y означает буквально следующее: «на данном сервере гарантированно применены действия, соответствующие позиции X в журнале удаленного сервера Y».

3.2.1.3. Разрешение конфликтов

Для разрешения возможных конфликтов, возникающих при одновременном добавлении, удалении или изменении данных на разных узлах кластера необходим механизм разрешения конфликтов. Целесообразно внести этот механизм в абстракцию действия.

Введем понятие **состояния** БД – это уникальный объект, который описывает (но не заменяет) некоторый «снимок» БД, т. е. наличие определенных данных.

Для поддержки проверки состояний, действия должны отвечать следующим

свойствам:

- При непосредственном внесении данных в узел кластера создается новое действие.
- Новое действие применимо к любому состоянию БД.
- После применения нового действия, ему автоматически привязывается текущее состояние БД, к которому оно было применено.
- Состояние сериализуется вместе с остальными данными действия.
- Дальнейшее применение действия вызывает проверку соответствия текущего состояния БД и записанного в действие допустимого состояния. Несоответствие состояний должно вызывать ошибку и не допускать применение данного действия.
- Действие должно допускать работу в режиме без проверки состояний.

Механизм получения и обновления состояний должен быть реализован в виде модуля к действию. Такая архитектура позволит создать несколько версий алгоритма и менять их, не затрагивая остальных частей программы.

3.2.1.4. Алгоритм создания состояний

Реализованный алгоритм создания состояний включает в себя 4 вида состояний БД:

1. Глобальное – охватывает данные всех арен, сегментов и зон.
2. Арены – охватывает данные всех сегментов и зон конкретной арены.
3. Сегмента – охватывает данные всех зон конкретного сегмента.
4. Зоны – охватывает данные конкретной зоны.

Любое из состояний представляет собой строковый результат применения хэш-функции MD5 над входными данными. Все состояния хранятся в **таблице состояний** (таблица dbstate, рисунок 7, п. 3.4) и перерасчитываются по мере

применения действий, изменяющих соответствующие данные БД.

Состояния связаны между собой иерархично. Для состояния зоны входными данными является текст всех записей зоны. Для состояния сегмента – набор состояний всех входящих в него зон. Для состояния арены – набор состояний всех входящих в нее сегментов. Для глобального состояния – набор состояний всех существующих арен.

3.2.1.4. Алгоритм работы репликации

Репликация данных происходит в соответствии со следующим порядком:

1. Каждый из узлов кластера периодически запрашивает обновление данных у каждого из соседних узлов (т. н. pull-запрос). В данном запросе отправляется актуальная позиция соответствующего сервера из таблицы соответствия.
2. Сервер, принявший pull-запрос, получает из локального журнала определенное количество действий, начиная с актуальной позиции, указанной в запросе.
3. Действия отправляются инициатору pull-запроса.
4. Исходный сервер применяет каждое из принятых действий. При этом в локальный журнал действия не пишутся.
5. Исходный сервер отправляет повторный pull-запрос.

Схема начального взаимодействия серверов кластера представлена на рисунке 3, стр. 46.

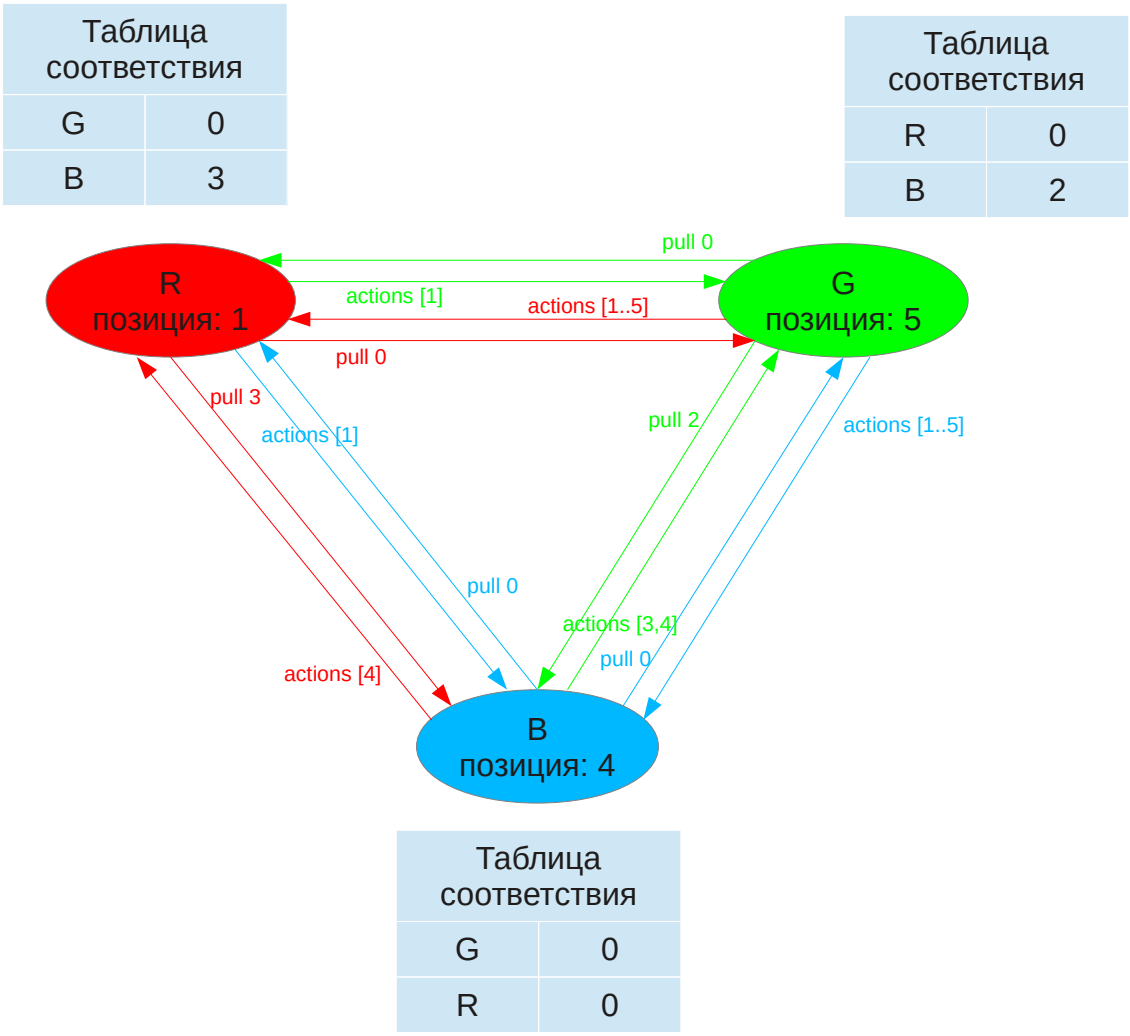

Рисунок 3: начальное взаимодействие серверов синхронизации

Таким образом сервер самостоятельно инициирует передачу новых действий от каждого из соседних узлов. Приведенный алгоритм повторяется до тех пор, пока актуальная позиция не сравняется с фактической позицией последней записи в журнале удаленного сервера.

Для того, чтобы новые данные распространялись с минимальными задержками необходим способ обновления, инициируемый источником обновления. Для этого вводится понятие активного сервера. **Активным источником** называется сервер, самостоятельно распространяющий примененные к нему действия. **Активным приемником** называется сервер, принимающий действия от активного источника.

После того, как актуальная позиция сервера сравнялась с фактической позицией удаленного сервера, удаленный сервер переходит в состояние активного источника и переводит исходный сервер в состояние активного приемника (с помощью т. н. wait-запроса). Данная операция происходит без разрыва сетевого соединения. В данном состоянии активный приемник способен мгновенно получать все актуальные изменения сразу после их внесения в активный источник.

Временная диаграмма описанного процесса представлена на рисунке 4, стр. 48. На данной диаграмме предполагается, что сервер отсылает за раз не более 50 действий.

В случае потери соединения оба сервера теряют статус активных. Для того, чтобы соединение не было разорвано из-за отсутствия сетевой активности, подключенный клиент периодически отсылает серверу пустое сообщение, требующее ответа (ping). В случае длительного отсутствия ответа, клиент самостоятельно разрывает соединение. Дальнейшее взаимодействие происходит по уже описанному алгоритму.

Каждый из серверов может быть одновременно активным источником и активным приемником. Таким образом, в нормальном режиме работы и при отсутствии каких-либо сбоев, кластер серверов синхронизации стремится к состоянию, в котором каждый сервер является активным источником для всех остальных и активным приемником от всех остальных серверов.

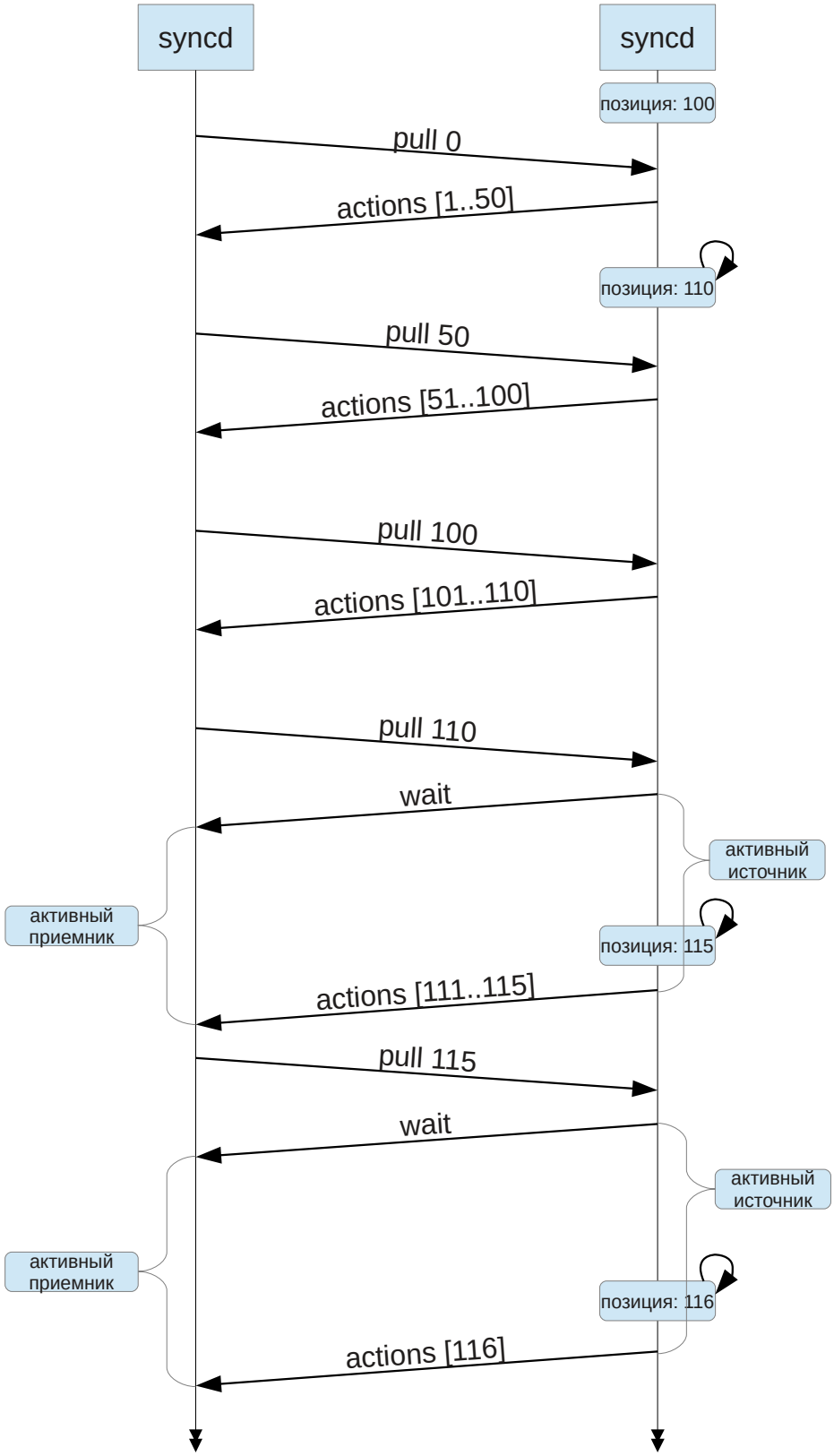

Рисунок 4: переход серверов в состояние активных

3.2.2. Сетевой протокол

Взаимодействие между серверами синхронизации происходит по разработанному текстовому протоколу. В основе разработанного протокола лежит текстовый формат сериализации данных YAML. Поддерживается:

- аутентификация серверов кластера (методами PAP и CHAP);
- работа поверх протокола транспортного TCP;
- работа поверх протокола обеспечения шифрования транспортного уровня TLS.

Каждый из узлов кластера выступает в ролях **клиента синхронизации** и **сервера синхронизации**. Клиент синхронизации подключается к серверу синхронизации для выполнения запроса на обновление данных.

3.2.2.1. Аутентификация

Поддерживается аутентификация методами PAP (Password Authentication Protocol) и CHAP (Challenge Handshake Authentication Protocol).

Для каждой пары серверов выбирается **пароль** и записывается в файле конфигурации.

В случае использования метода PAP, после подключения к серверу, клиент синхронизации посылает свое имя и пароль. Сервер синхронизации проверяет пароль путем сравнения.

При использовании метода CHAP, после подключения к серверу, клиент синхронизации посылает свое имя. Сервер синхронизации генерирует произвольную строку (называемую далее **солью**) и отсылает ее клиенту. Принятая клиентом строка комбинируется с паролем и к полученной строке применяется хэш-функция, результат отсылается серверу. Данная операция называется далее **расчетом строки согласования**. Сервер выполняет аналогичные действия и сравнивает самостоятельно рассчитанный результат с

принятым по сети. В случае совпадения клиент синхронизации считается аутентифицированным.

3.2.2.2. Клиент синхронизации

Клиент синхронизации может находится в следующих состояниях:

- CONNECTED – произошло подключение к серверу.
- AUTH_REQUEST_SENT – послан запрос на аутентификацию.
- AUTH_CHALLENGE_RECEIVED – получен запрос на проведение аутентификации методом CHAP.
- AUTH_CHALLENGE_SENT – послан результат расчета строки согласования.
- OPERATIONAL – базовое состояние, в котором клиент синхронизации периодически запрашивает у сервера обновления.
- PULL_REQUEST_SENT – послан запрос обновления данных (pull-запрос).
- WAIT – клиент синхронизации находится в состоянии активного приемника (см п. 5.2.1.4).

Конечный автомат клиентской стороны протокола синхронизации представлен на рисунке 5, стр. 51.

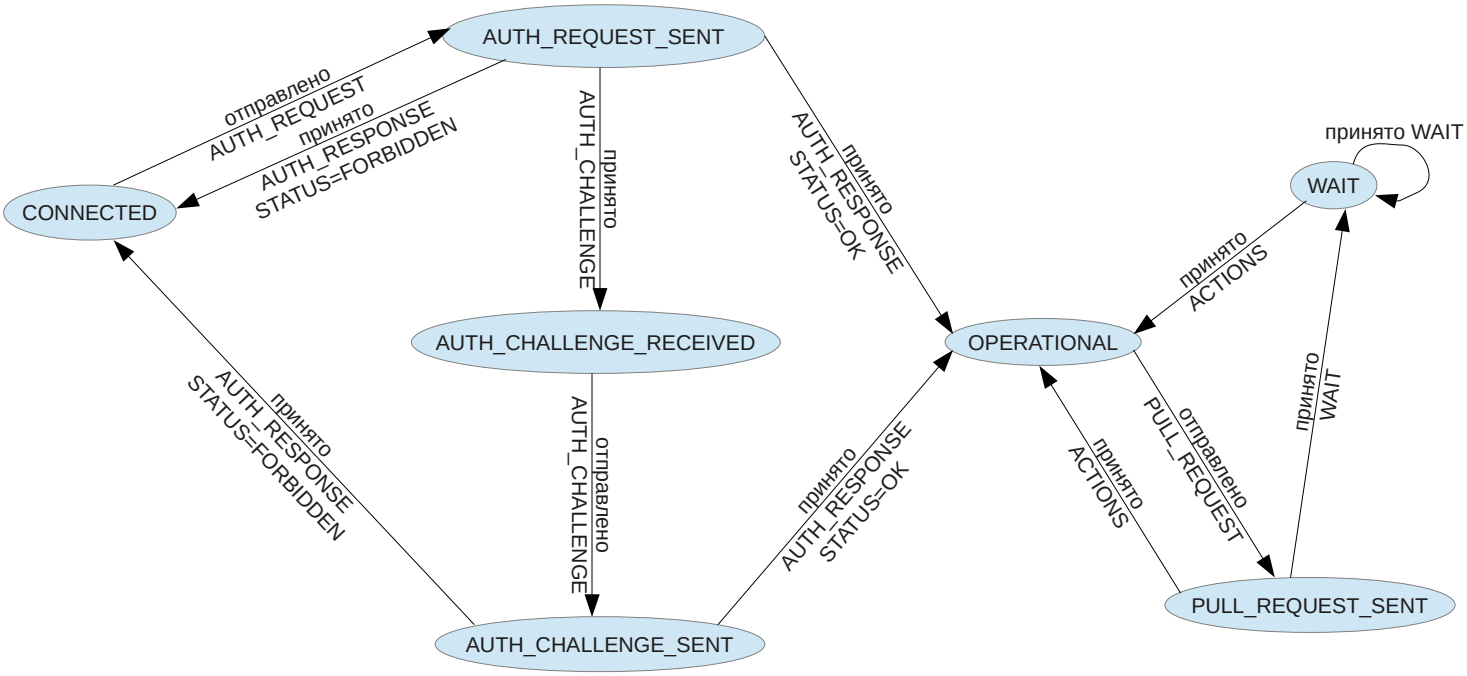

Рисунок 5: конечный автомат протокола клиента синхронизации

3.2.2.3. Сервер синхронизации

Сервер синхронизации может находится в следующих состояниях:

- CONNECTED – клиент синхронизации подключен.
- AUTH_REQUEST_RECEIVED – получен запрос на аутентификацию.
- AUTH_CHALLENGE_SENT – послан запрос на проведение аутентификации методом CHAP.
- AUTH_CHALLENGE_RECEIVED – получен результат расчета строки согласования.
- OPERATIONAL – базовое состояние, в котором сервер синхронизации принимает запросы на обновление данных.
- PULL_REQUEST_RECEIVED – получен запрос на обновление данных.
- WAIT_SENT – сервер синхронизации находится в состоянии активного источника (см п. 5.2.1.4).

Конечный автомат серверной стороны протокола синхронизации представлен на

рисунке 6, стр. 52.

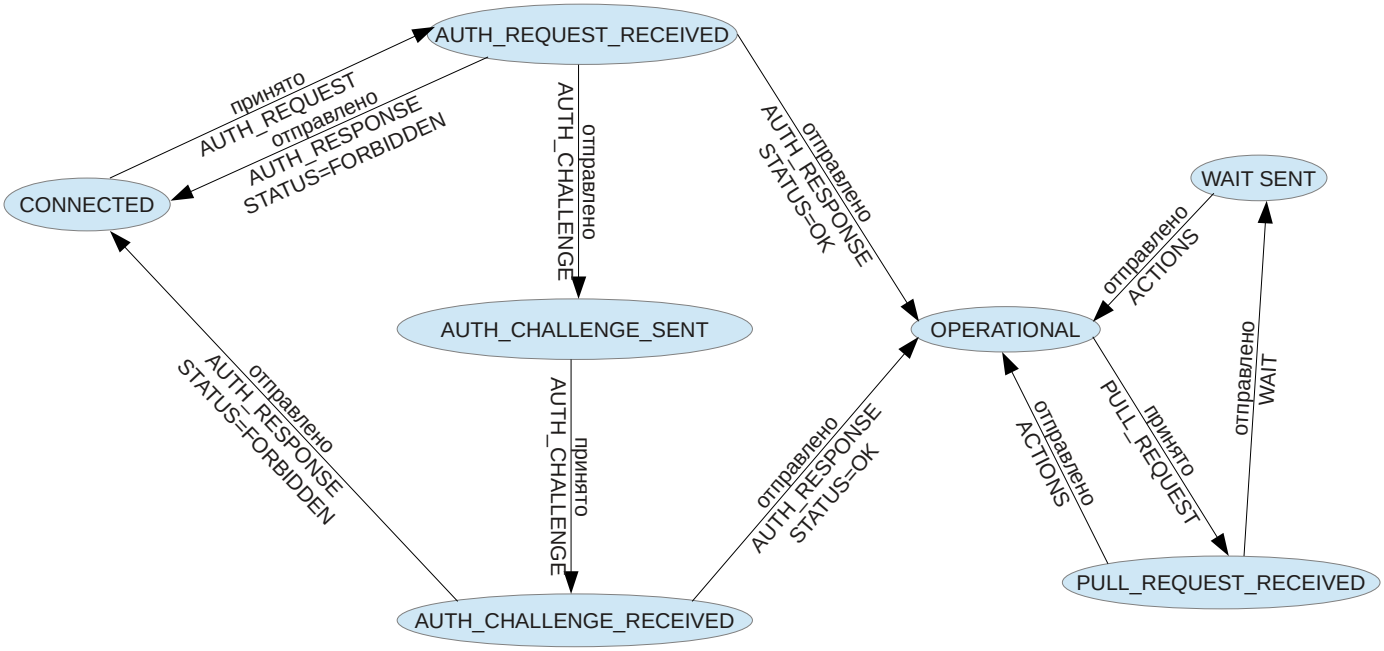

Рисунок 6: конечный автомат протокола сервера синхронизации

3.2.2.4. Команды протокола

Протокол синхронизации использует следующие команды и соответствующие им сообщения для осуществления взаимодействия между клиентом и сервером:

- AUTH_REQUEST – запрос аутентификации.

  При использовании метода аутентификации PAP:

```
cmd: auth_request
auth_schema: pap
name: <имя сервера>
pswd: <пароль>
```

  При использовании метода аутентификации CHAP:

```
cmd: auth_request
auth_schema: chap
name: <имя сервера>
```

- AUTH_CHALLENGE

  Сервер синхронизации запрашивает аутентификацию методом CHAP в сообщении:

```
cmd: auth_challenge
challenge: <соль>
```

Клиент синхронизации отправляет рассчитанную строку согласования в сообщении:

```
cmd: auth_challenge
challenge: <строка согласования>
```

- AUTH_RESPONSE – ответ сервера синхронизации на попытку аутентификации.

  Доступ разрешен:

```
cmd: auth_response
status: 0
```

  Доступ запрещен:

```
cmd: auth_response
status: 2
```

- PULL_REQUEST – запрос клиента синхронизации на реплицирование действий, начиная с определенной позиции.

```
cmd: pull_request
position: <актуальная позиция>
```

- ACTIONS – реплицирование действий от сервера синхронизации.

```
cmd: actions
status: 0
data:
    - action: <сериализованное действие>
      position: <позиция, к которой приводит действие>
    - action: <...>
      position: <...>
    …
```

  В случае, если клиент синхронизации указал неверную позицию в запросе, формируется сообщение:

```
cmd: actions
status: 1
```

- WAIT – перевод клиента синхронизации в состояние активного.

```
cmd: wait
```

- PING – проверка доступности удаленной стороны

```
cmd: ping
```

3.2.3. Действия

Перечень действий, поддерживаемых системой:

- AddArena – добавление арены.

  Параметры:

  - arena – имя арены. Уникально в глобальном пространстве имен арен.
  - key – ключ для доступа к арене.

  Допускается применение к любому состоянию БД.

- DelArena – удаление арены.

  Параметры:

  - arena – имя арены.

  Осуществляется проверка глобального состояния.

- ModAuth – изменение ключа арены.

  Параметры:

  - target – арена, для которой изменяется ключ.
  - key – новый ключ доступа к арене.

  Допускается применение к любому состоянию БД.

- AddSegment – добавление сегмента.

  Параметры:

  - arena – имя арены, в которую добавляется сегмент.
  - segment – имя сегмента. Уникально в пространстве имен сегментов данной арены.

Осуществляется проверка состояния арены, в которую добавляется сегмент.

- DelSegment – удаление сегмента.

  Параметры:

  - arena – имя арены, из которой удаляется сегмент.
  - segment – имя сегмента.

  Осуществляется проверка состояния арены, из которой удаляется сегмента.

- AddZone – добавление зоны.

  Параметры:

  - arena – имя арены, в которой искать сегмент.
  - segment – имя сегмента, к которому привязывается зона.
  - zone – имя зоны. Уникально в глобальном пространстве имен зон.

  Осуществляется проверка состояния сегмента, к которому привязывается зона.

- DelZone – удаление зоны.

  Параметры:

  - zone – имя зоны.

  Осуществляется проверка состояния сегмента, к которому привязана зона.

Далее рассмотрены действия для работы с DNS-записями, и приведены специфичные параметры для каждого. Общие параметры:

- Все действия:
  - zone – имя зоны, с которой связана запись.

- Добавляющие действия:
  - ttl – время жизни записи.

Перечень действий:

- AddRecord_A – добавление DNS-записи типа A.

  Параметры:

  - host – символьное имя внутри данной зоны. Совместно с зоной составляют имя в соответствие которому ставится данный IP-адрес.
  - ip – IP-адрес, который ставится в соответствие данному имени.
- DelRecord_A – удаление DNS-записи типа A.

  Параметры:

  - host – символьное имя внутри данной зоны, соответствующее искомой записи.
  - ip – IP-адрес, поставленный в соответствие данному имени.
- AddRecord_PTR – добавление DNS-записи типа PTR.

  Параметры:

  - zone – имя зоны для требуемого диапазона IP-адресов.
  - host – последняя часть IP-адреса, для которого ставится в соответствие символьное имя.
  - domain – символьное имя, которое ставится в соответствие данному IP-адресу.
- DelRecord_PTR – удаление DNS-записи типа PTR.

  Параметры:

  - zone – имя зоны для требуемого диапазона IP-адресов.

- host – последняя часть IP-адреса, соответствующего искомой записи.
- domain – символьное имя, поставленное в соответствие данному IP-адресу.

- AddRecord_MX – добавление DNS-записи типа MX.

  Параметры:

  - domain – адрес почтового сервера. Допускается имя внутри зоны или абсолютное доменное имя (FQDN – fully qualified domain name).
  - priority – приоритет данного почтового сервера (меньшее значение имеет больший приоритет).
- DelRecord_MX – удаление DNS-записи типа MX.

  Параметры:

  - domain – адрес почтового сервера.
- AddRecord_NS – добавление DNS-записи типа NS.

  Параметры:

  - domain – адрес сервера имен, авторитетного для данной зоны.
- DelRecord_NS – удаление DNS-записи типа NS.

  Параметры:

  - domain – адрес сервера имен.
- AddRecord_CNAME – добавление DNS-записи CNAME.

  Параметры:

  - host – символьное имя внутри данной зоны.
  - domain – символьное имя, для которого данное имя является псевдонимом. Допускается имя внутри зоны или абсолютное доменное

имя.

- DelRecord_CNAME – удаление DNS-записи CNAME.

  Параметры:

  - host – символьное имя внутри данной зоны.
  - domain – символьное имя, для которого данное имя является псевдонимом.

- AddRecord_DNAME – добавление DNS-записи типа DNAME.

  Параметры:

  - zone_dst – имя зоны, для которой данная зона является псевдонимом.

- DelRecord_DNAME

  Параметры:

  - zone_dst – имя зоны, для которой данная зона является псевдонимом.

- AddRecord_SRV – добавление DNS-записи типа SRV.

  Параметры:

  - priority – приоритет записи (меньшее значение соответствует большему приоритету).
  - weight – относительный вес записи среди записей с одинаковым приоритетом.
  - service – символьное имя сервиса.
  - port – сетевой порт, на котором находится сервис.
  - domain – символьное имя, соответствующее сервису. Допускается имя внутри зоны или абсолютное доменное имя.

- DelRecord_SRV – удаление DNS-записи типа SRV.

Параметры:

- service – символьное имя сервиса.
- port – сетевой порт, на котором находится сервис.
- domain – символьное имя, соответствующее сервису.

- AddRecord_TXT – добавление DNS-записи типа TXT.

  Параметры:

  - text – текст, связываемый с данной зоной.

- DelRecord_TXT – удаление DNS-записи типа TXT.

  Параметры:

  - text – текст искомой записи.

- AddRecord_SOA – добавление DNS-записи типа SOA.

  Параметры:

  - primary_ns – первичный сервер имен зоны.
  - resp_person – адрес электронной почты администратора зоны.
  - serial – номер ревизии зоны.
  - refresh – время (в секундах) жизни зоны на вторичном DNS-сервере, после которого зона запрашивается вновь.
  - retry – время (в секундах) вторичного DNS-сервера перед повторным обновлением зоны (refresh), если прошлая попытка обновления была провалена.
  - expire – время (в секундах) после которого вторичный DNS-сервер принудительно перестает отвечать на запросы по данной зоне.
  - minimum – минимальное время жизни записей зоны.

- DelRecord_SOA – удаление DNS-записи типа SOA.

  Параметры: отсутствуют. Удаляется SOA-запись данной зоны (стандартом на службу DNS допускается единственная SOA запись для каждой зоны).

3.2.4. Конфигурация кластера

Конфигурация сервера синхронизации описывается в формате YAML.

Поддерживаются следующие опции шифрования и аутентификации:

```
transport-encrypt: (true|false)
accept-auth: (chap|pap|any)
```

- transport-encrypt – шифрование транспортного уровня: включить/выключить.
- accept-auth – выбор метода аутентификации: pap, chap или any (любой из pap, chap).

Настройка секции сервера включает в себя опции:

```
server:
    name: NODE1
    interface: 0.0.0.0
    port: 1100
    private-key: /etc/dns_cluster/syncd/NODE1.key
    cert: /etc/dns_cluster/syncd/NODE1.pem
```

- name – символьное имя сервера, использующееся для его дальнейшей идентификации.
- interface – сетевой интерфейс, на котором доступен сервер.
- port – сетевой порт (протокола TCP), на котором доступен сервер.
- private-key – путь к приватному файлу, содержащему приватный ключ (необходим только при включенном шифровании транспортного уровня).
- cert – путь к файлу, содержащему сертификат (необходим только при включенном шифровании транспортного уровня).

Настройка базы данных:

```
database:
    dbenv_homedir: ./tests/databases/alpha
    dbfile: dlz.db
```

- dbenv_homedir – директория окружения базы данных Berkeley DB.
- dbfile – название файла, содержащего базу данных.

Настройка кластера включает в себя настройку секций для каждого из допустимых удаленных серверов.

```
peers:
    <имя сервера>:
        key: "topsecret"
        host: 192.168.1.1
        port: 1100
     ...
```

- <имя сервера> – символьное имя удаленного сервера.
- key – ключ аутентификации, используемый между данным сервером и указанным удаленным.
- host – IP-адрес удаленного сервера.
- port – сетевой порт удаленного сервера.

3.3. Сервер управления БД

Сервер управления БД реализует web-сервис, предоставляющий интерфейс работы с БД.

3.3.1. Сессии

Механизм сессий позволяет вносить изменения в БД, и иметь возможность дальнейшей отмены произведенных действий. Введем понятие **сессии** – это процесс работы с базой данных обладающий следующими особенностями:

- Обратимость – все действия, примененные в сессии можно отменить,

восстановив БД до состояния до начала сессии.

- Независимость от состояния сетевого соединения – после разрыва сетевого соединения и нового подключения клиента, сессию можно продолжить без каких-либо последствий.
- Работа с сессиями подразумевает выполнение следующих действий:
  - начало сессии;
  - фиксация сессии – подразумевает фиксацию всех примененных действий и окончание сессии;
  - откат сессии – подразумевает отмену всех примененных действий и окончание сессии;
  - продление сессии.
- При отсутствии активности в течение определенного времени происходит автоматический откат сессии.

3.3.1.1 Журнал действий

Для поддержки свойства обратимости сессии был разработан механизм журналирования действий внутри сессии. Данный журнал фактически представляет собой отдельную таблицу с записями, хранящими сериализованные действия (таблица action_journal, рисунок 7, п. 3.4).

Вводится понятие **противодействия** – это действие, которое нейтрализует изменения сделанные некоторым действием. После каждого применения действия внутри сессии в журнал записывается примененное действие и его противодействие.

При фиксации сессии все записи журнала, касающиеся данной сессии очищаются.

При откате сессии происходит применение противодействия для каждой записи

журнала, касающейся данной сессии, в обратном порядке, с последующей очисткой журнала от этих записей.

3.3.2. Блокировки

При совместной работе с сервисом управления БД могут возникать конфликты и непредсказуемое поведение механизма сессий, когда пользователи изменяют одни и те же данные. Для решения проблем данного рода был разработан механизм блокировок.

Ресурсом называется некоторая часть БД, к которой можно применить блокировку. Блокировка ресурса осуществляется внутри сессии. С блокированным ресурсом может работать лишь пользователь данной сессии. При этом остальная часть БД остается доступна.

Различаются следующие ресурсы:

- Глобальный ресурс – блокирует доступ ко всей базе данных целиком.
- Ресурс арены – блокирует доступ к определенной арене.
- Ресурс сегмента – блокирует доступ к определенному сегменту.
- Ресурс зоны – блокирует доступ к определенной зоне.

Таким образом понятие ресурса иерархично. Ресурс представлен в виде строки со специальным символом-разделителем. Примеры ресурсов:

- Глобальный: _global
- Ресурс арены с именем myarena: _global::myarena
- Ресурс сегмента с именем mysegment, принадлежащего арене myarena: _global::myarena::mysegment
- Ресурс зоны с именем myzone.com, принадлежащей сегменту mysegment: _global::myarena::mysegment::myzone.com

3.3.2.1. Блокировки в сессиях

В рамках одной сессии может быть поставлено произвольное число блокировок. После завершения сессии блокировки сбрасываются.

Взятие блокировок осуществляется по следующим правилам:

1. Допускается повторное взятие блокировки одного ресурса в одной сессии.
2. Допускается взятие блокировки вышестоящего или нижестоящего по иерархии ресурса по отношению к уже заблокированному в той же сессии.
3. Недопускается взятие блокировки одного ресурса в разных сессиях.
4. Недопускается взятие блокировки вышестоящего или нижестоящего по иерархии ресурса по отношению к уже заблокированному в другой сессии.

3.3.3. Операции

Для представления любого действия с данными БД введено понятие **операции**. Все операции поддерживаемые сервером управления БД составляют его основной интерфейс. Бывают следующие типы операций:

1. Работы с сессиями.
2. Администрирования.
3. Получения DNS-данных.
4. Добавления DNS-данных.
5. Удаления DNS-данных.

3.3.3.1. Операции работы с сессиями

Поддерживаются следующие операции работы с сессиями:

- SessionBeginOp – начало сессии.

Параметры:

- auth_arena – имя арены, к которой осуществляется пользовательский доступ (называемый далее **пользовательским логином**) или предопределенное имя для доступа ко всем аренам (называемого далее **логином администратора**).
- auth_key – ключ доступа.

Возвращаемое значение:

- sessid – идентификатор сессии, по которому осуществляется дальнейшая работа в сессии.

- SessionKeepaliveOp – продление сессии.

  Параметры:

  - sessid – идентификатор целевой сессии.

- SessionCommitOp – фиксация сессии.

  Параметры:

  - sessid – идентификатор целевой сессии.

- SessionRollbackOp – откат сессии.

  Параметры:

  - sessid – идентификатор целевой сессии.

3.3.3.2. Операции администрирования

Поддерживаются следующие операции администрирования:

- ModAuthOp – изменение ключа доступа к арене.

  Параметры:

  - target – имя целевой арены. В случае логина администратора является обязательным. При пользовательском логине в качестве арены

используется арена текущей сессии.

- Остальные параметры соответствуют параметрам действия ModAuth.

3.3.3.3. Операции работы с DNS-данными

Все операции добавления и удаления DNS-данных обязательно требуют параметров, соответствующих действий. Например, для операции AddArenaOp требуются все параметры действия AddArena и т. д. Также операции могут принимать дополнительные параметры.

Любая операция для работы с DNS-данными удовлетворяет следующим свойствам:

1. Выполнение внутри сессии:
    1. Если сессия была создана ранее, то ее идентификатор передается параметром (sessid, см. п. 5.3.3.1).
    2. Если идентификатор не указан, требуется передать дополнительные параметры, соответствующие операции создания сессии (auth_arena, auth_key, см п. 5.3.3.1). В этом случае требуемая операция будет выполнена в рамках автоматически созданной сессии. Дальнейшее использование данной сессии невозможно.
2. Установка соответствующей блокировки.
    1. При работе на уровне арен – глобальная блокировка.
    2. При работе на уровне сегментов – блокировка соответствующей арены.
    3. При работе на уровне зон – блокировка соответствующего сегмента.
    4. При работе на уровне записей зоны – блокировка соответствующей зоны.
3. При невозможности установить блокировку (в случае, если ресурс уже заблокирован в другой сессии) происходит ожидание в течение

определенного времени и повторная попытка. Если все попытки были исчерпаны действие завершается неудачно.

4. При использовании пользовательского логина, в качестве арены используется арена, указанная в создании сессии.
5. При использовании логина администратора необходимо указывать арену при помощи параметра arena.

Перечень поддерживаемых операций работы с DNS-данными включает в себя:

- AddArenaOp – добавление арены.
- DelArenaOp – удаление арены.
- GetArenasOp – получение списка арен. Доступна только для логина администратора.
- AddSegmentOp – добавление сегмента.
- DelSegmentOp – удаление сегмента.
- GetSegmentsOp – получение списка сегментов.

  Параметры:

  - Имя арены берется из сессии или указывается явно (см. свойства 4, 5).

  Возвращаемое значение:

  - Список имен сегментов.
- AddZoneOp – добавление зоны.

  Параметры:

  - Имя арены берется из сессии или указывается явно (см. свойства 4, 5).
  - initial_records – список начальных записей зоны. Необязательный параметр. Формат определения каждой из записей совпадает с форматом параметров операции добавления записи.

- DelZoneOp – удаление зоны.
- GetZonesOp – получение списка зон.

  Параметры:

  - Имя арены берется из сессии или указывается явно (см. свойства 4, 5). В случае логина администратора и отсутствии параметра арены будет получен список всех зон.
  - segment – имя целевого сегмента, необязательный параметр. В случае отсутствия будет получен список всех зон арены, иначе – список зон указанной арены и указанного сегмента.
- AddRecordOp – добавление записи.

  Параметры:

  - type – тип добавляемой записи.
  - rec_spec – набор параметров соответствующего действия по добавлению указанного типа записи.
- DelRecordOp – удаление записи.

  Параметры:

  - type – тип удаляемой записи.
  - rec_spec – набор параметров соответствующего действия по удалению указанного типа записи.
- GetRecordsOp – получение списка записей.

  Параметры:

  - zone – имя целевой зоны.

  Возвращаемое значение:

  - список записей, каждая из которых представлена в формате

хэш-таблицы соотвествия имени параметра его значению.

3.3.4. Взаимодействие с сервером синхронизации

Для возможности реализации концепции активного источника, сервер синхронизации должен иметь возможность следить за изменениями в данных.

Изменение данных происходит централизованно через сервер управления БД. После того, как любое изменение вступает в силу (в результате фиксации сессии), сервер управления БД посылает сигнал (средствами операционной системы) серверу синхронизации БД, по которому последний осуществляет обновление активных приемников (см. п. 5.2.1.4).

3.3.5. Сетевой протокол

Взаимодействие пользователя с системой управления БД осуществляется по протоколу HTTP или, в случае использования шифрования транспортного уровня, HTTPS. В качестве формата передаваемых данных поверх протокола HTTP используется формат YAML.

Каждой существующей операции поставлен в соответствие URL:

- GetArenasOp – get_arenas;
- GetSegmentsOp – get_segments;
- GetZonesOp – get_zones;
- GetRecordsOp – get_records;
- SessionBeginOp – begin_session;
- SessionKeepaliveOp – keepalive_session;
- SessionCommitOp – commit_session;
- SessionRollbackOp – rollback_session;
- ModAuthOp – mod_auth;

- AddArenaOp – add_arena;
- DelArenaOp – del_arena;
- AddSegmentOp – add_segment;
- DelSegmentOp – del_segment;
- AddZoneOp – add_zone;
- DelZoneOp – del_zone;
- AddRecordOp – add_record;
- DelRecordOp – del_record.

3.3.6. Конфигурация

Конфигурация сервера управления БД описывается в формате YAML.

Настройка секции базы данных совпадает с соответствующей настройкой в файле конфигурации сервера синхронизации.

Основные настройки сервера:

```
interface: 192.168.1.1
port: 2100
syncd_pid_path: /run/syncd.pid
```

- interface – сетевой интерфейс, на котором доступен сервер.
- port – сетевой порт (протокола TCP), на котором доступен сервер.
- syncd_pid_path – путь к файлу, содержащему идентификатор процесса (Process Id) сервера синхронизации syncd. Используется для передачи сигнала серверу синхронизации при обновлении данных (см. п. 5.3.4).

3.4. Схема БД

Используемая встраиваемая база данных Berkeley DB предоставляет возможность создавать таблицы для хранения данных в виде ключ-значение. Система использует единую базу данных для всех компонентов, схема которой представлена на рисунке 7, стр.72.

Таблицы dns_zone, dns_client, dns_data и dns_xfr используются DNS-сервером BIND с расширением DLZ для хранения DNS-данных. Для реализации схемы разделения данных на арены и сегменты и организации доступа используются дополнительные таблицы: arena, arena_auth, arena_segment, segment_zone, zone_dns_data.

Таблицы dbstate, peer и action используются сервером синхронизации БД для работы механизма проверки состояний, организации репликации и ведения журнала действий (см. п. 5.2).

Таблицы session, action_journal, session_action, lock, session_lock, lock_hier, lock_wait используются сервером управления БД для реализации механизмов сессий и блокировок (см. п. 5.3).

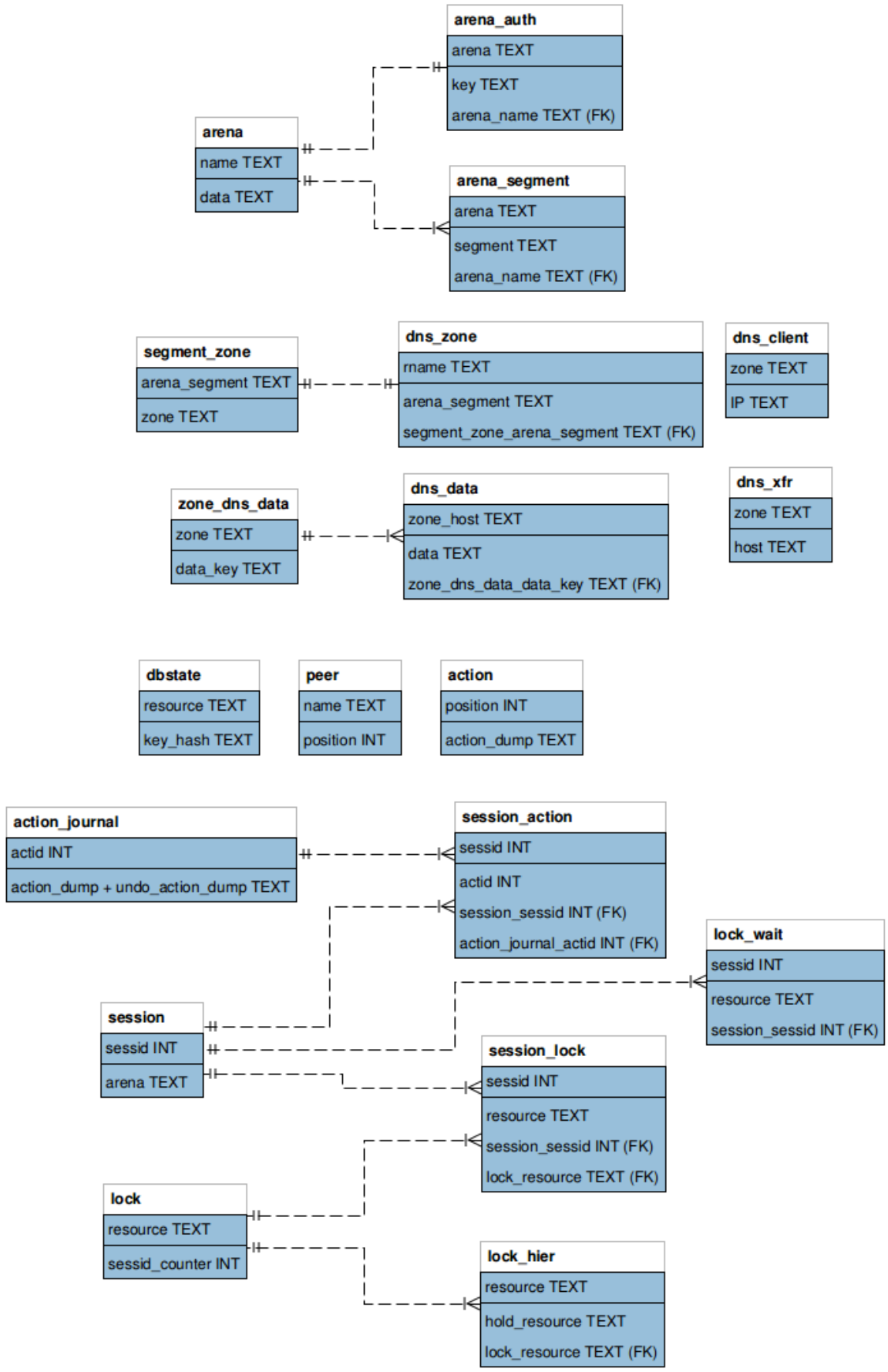

Рисунок 7: схема БД

3.5. Функциональное тестирование

Для осуществления функционального тестирования системы был разработан комплекс тестов, покрывающих основные функции сервера управления БД и сервера синхронизации БД и соответствующих выявленным требованиям к тестам (см. п. 2.3).

Во всех созданных тестах производится тестирование кластера, состоящего из трех узлов: alpha, beta и gamma. Конфигурация серверов управления и синхронизации БД каждого из этих узлов приведена далее.

Система управления БД alpha:

```
database:
    dbenv_homedir: ./tests/databases/alpha
    dbfile: dlz.db
interface: 127.0.0.1
port: 2100
syncd_pid_path: /run/alpha.pid
```

Система синхронизации БД alpha:

```
transport-encrypt: true
accept-auth: chap
server:
    name: alpha
    interface: 127.0.0.1
    port: 1100
    private-key: ./tests/configs/syncd/cert/alpha.key
    cert: ./tests/configs/syncd/cert/alpha.pem
database:
    dbenv_homedir: ./tests/databases/alpha
    dbfile: dlz.db
peers:
    beta:
        key: "topsecret"
        host: 127.0.0.1
        port: 1101

    gamma:
        key: "topsecret2"
        host: 127.0.0.1
        port: 1102
```

Система управления БД beta:

```
database:
    dbenv_homedir: ./tests/databases/beta
    dbfile: dlz.db

interface: 127.0.0.1
port: 2101
syncd_pid_path: /run/beta.pid
```

Система синхронизации БД beta:

```
transport-encrypt: true
accept-auth: chap
server:
    name: beta
    interface: 127.0.0.1
    port: 1101
    private-key: ./tests/configs/syncd/cert/beta.key
    cert: ./tests/configs/syncd/cert/beta.pem
database:
    dbenv_homedir: ./tests/databases/beta
    dbfile: dlz.db
peers:
    alpha:
        key: "topsecret"
        host: 127.0.0.1
        port: 1100
    gamma:
        key: "topsecret3"
        host: 127.0.0.1
        port: 1102
```

Система управления БД gamma:

```
database:
    dbenv_homedir: ./tests/databases/gamma
    dbfile: dlz.db
interface: 127.0.0.1
port: 2102
syncd_pid_path: /run/gamma.pid
```

Система синхронизации БД gamma:

```
transport-encrypt: true
accept-auth: chap
server:
    name: gamma
    interface: 127.0.0.1
    port: 1102
    private-key: ./tests/configs/syncd/cert/gamma.key
    cert: ./tests/configs/syncd/cert/gamma.pem
database:
    dbenv_homedir: ./tests/databases/gamma
    dbfile: dlz.db
peers:
    alpha:
        key: "topsecret2"
        host: 127.0.0.1
        port: 1100
    beta:
        key: "topsecret3"
        host: 127.0.0.1
        port: 1101
```

3.6. Выводы

В ходе работы система были определены компоненты системы в соответствии с выполняемой задачей: DNS-сервер, БД, сервер управления БД, сервер синхронизации БД. Сервер управления БД был реализован как web-сервис, построенный по принципам REST. Сервер синхронизации БД реализует механизм асинхронной мульти-мастерной репликации. Были разработаны функциональные тесты, позволяющие проверять базовую работоспособность серверов управления и синхронизации БД. Разработанная система полностью удовлетворяет поставленным требованиям и готова к эксплуатации.

В перспективах проекта планируется:

- Дорабатывать механизм разрешения конфликтов.
- Добавить поддержку операций рекурсивного удаления объектов вместе со всем их содержимым (например, удаление арены со всеми связанными сегментами, зонами и записями).
- Добавить поддержку операций перемещения объектов в рамках иерархии

(например, перемещение некоторых сегментов арены в другую арену).

- Добавить поддержку операций переименования объектов.
- Добавить поддержку операций рекурсивной деактивации/активации объектов без их удаления (например, деактивация сегмента, всех связанных зон и записей).
- Разработать приложение-консоль для подключения к базе данных системы и выполнения служебных действий.
- Проработать механизм ротации, очистки и архивирования журнала действий.
- Проработать механизм трансфера всех данных уже работающего кластера на вновь внесенный новый узел в случае, если на старых узлах имеется неполный журнал действий.

4. Экспериментальная часть

4.1. Оценка соответствия разработанной системы техническому заданию

Разработанная система полностью удовлетворяет всем требованиям технического задания. Возможности реализованной системы:

- создание отказоустойчивых кластеров серверов доменных имен без единой точки отказа;
- наличие web-сервиса для управления DNS-данными;
- использование высокопроизводительного DNS-сервера ISC BIND;
- поддержка потенциально неограниченного горизонтального масштабирования в системе синхронизации БД.

4.2. Внедрение системы в рабочий процесс

Разработанная система была внедрена в рабочий процесс компании ЗАО «Флант». В ходе эксплуатации система доказала свою работоспособность и практическую пользу.

5. Охрана труда

5.1. Исследование возможных опасных и вредных факторов при эксплуатации ЭВМ и их влияния на пользователей

5.1.1. Введение

Охрана труда – это система законодательных актов, социально-экономических, организационных, технических, гигиенических и лечебно-профилактических мероприятий и средств, обеспечивающих безопасность, сохранение здоровья и работоспособности человека в процессе труда.

Основной задачей охраны труда является сведение к минимуму вероятности поражения или заболевания работающего с одновременным обеспечением комфорта при максимальной производительности труда.

Основным источником проблем, связанных с охраной здоровья людей, использующих в своей работе автоматизированные информационные системы на основе персональных компьютеров, являются дисплеи (мониторы), а так же другие средства отображения информации индивидуального пользования. Особенно стоит отметить дисплеи с электронно-лучевыми трубками. Они представляют собой источники наиболее вредных излучений, неблагоприятно влияющих на здоровье операторов и пользователей.

Конфигурация компьютеризированного рабочего места для работы над дипломом:

- Ноутбук на основе процессора Intel(R) Core(TM) i5-2520M CPU @ 2.50GHz с необходимым набором устройств ввода-вывода и хранения информации;
- TFT-дисплей с LED подсветкой:
  - диагональ 15.6 дюймов;
  - разрешение HD+ 1600x900;

- легко регулируемые контрастность и яркость;
- частота кадровой развертки при максимальном разрешении: 56-75 Гц;
- частота строчной развертки при максимальном разрешении: 30-83 кГц.

Питание ПЭВМ производится от сети 220В. Так как безопасным для человека напряжением является напряжение 40В, то при работе на ПЭВМ опасным фактором является поражение электрическим током.
В дисплее ПЭВМ высоковольтный блок строчной развертки и выходного строчного трансформатора вырабатывает высокое напряжение до 25кВ для второго анода электронно — лучевой трубки. А при напряжении от 5 до 300 кВ возникает рентгеновское излучение различной жесткости, которое является вредным фактором при работе с ПЭВМ (при 15-25 кВ возникает мягкое рентгеновское излучение).

Изображение на ЭЛТ создается благодаря кадрово-частотной развертке с частотой:

- 85 Гц (кадровая развертка);
- 42 кГц (строчная развертка).

Следовательно, пользователь попадает в зону электромагнитного излучения низкой частоты, которая является вредным фактором.

Во время работы компьютера дисплей создает ультрафиолетовое излучение, при повышении плотности которого > 10 Вт/м2, оно становиться для человека вредным фактором. Его воздействие особенно сказывается при длительной работе с компьютером.

Любые электронно-лучевые устройства, в том числе и электронно-вычислительные машины во время работы компьютера вследствие явления статического электричества происходит электризация пыли и мелких частиц, которые притягивается к экрану. Собравшаяся на экране

электризованная пыль ухудшает видимость, а при повышении подвижности воздуха, попадает на лицо и в легкие человека, вызывает заболевания кожи и дыхательных путей.

5.1.2. Выводы

При эксплуатации перечисленных элементов вычислительной техники могут возникнуть следующие опасные и вредные факторы:

1. Поражение электрическим током.
2. Электромагнитное излучение.
3. Ультрафиолетовое излучение.
4. Статическое электричество.

5.2. Методы измерений и оценки эргономических параметров и параметров безопасности

5.2.1. Средства измерения

Логично предположить, что для подобных методов измерений потребуются соответствующие средства. Их можно разделить на две независимые группы: средства предназначенные для измерения визуальных эргономических параметров и средства измерения параметров полей.

5.2.2. Средства измерения визуальных эргономических параметров

1. Яркометр – предназначен для измерения яркости участков рабочего поля экрана.
2. Люксометр – предназначен для измерения осещённости
3. Источник света – используется при измерении коэффициента диффузного отражения экрана дисплея (тип А по ГОСТ 7721).
4. Матовая поверочная пластина – используется при измерении коэффициента диффузного отражения экрана.

5. Микроскоп – используется для оценки пространственной нестабильности изображения.
6. Линейка беспараллаксная – предназначена для определения линейных размеров изображения на экране.

5.2.3. Средства измерения параметров полей

1. Измеритель электростатического потенциала поверхности экрана – предназначен для непосредственного измерения электростатического потенциала поверхности экрана.
2. Измеритель напряжённости электростатического поля – предназначен для непосредственного измерения электростатического потенциала поверхности экрана.
3. Измеритель напряжённости переменного электрического поля – предназначен для измерения среднеквадратичного значения напряжённости переменного электрического поля в двух диапазонах частот.
4. Измеритель плотности магнитного потока – предназначен для измерения среднеквадратичного значения плотности потока в двух диапазонах частот.

5.2.4. Подготовка, обработка и оценка результатов измерений

Перед началом измерений необходимо подготовить дисплей и средства измерения в соответствии с их эксплуатационной документацией.

Измерения проводят в нормальных климатических условиях, если другого не требуют нормативные документы на дисплей (по ГОСТ 21552).

Измерения проводят не ранее чем 20 минут после включения дисплея, если не предусмотрено других условий.

Измерения параметров изображения проводят в пяти участках экрана,

указанных на рисунке 8, стр. 84.

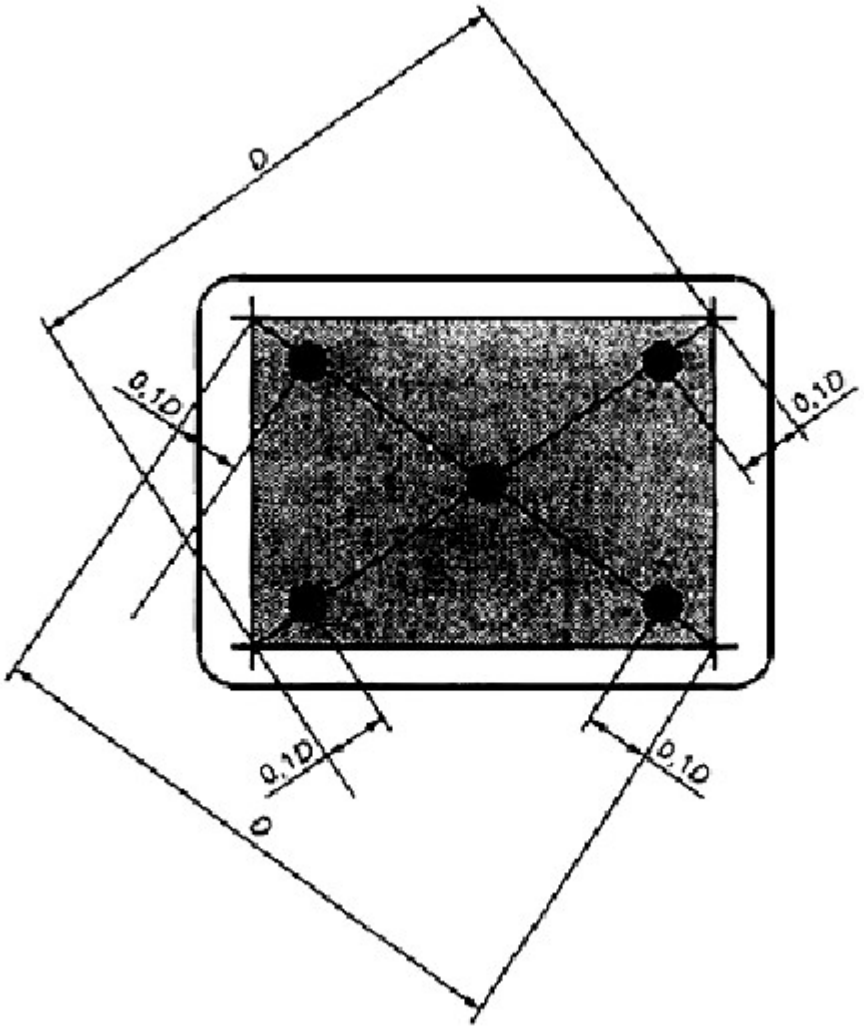

Рисунок 8: измерение параметров изображения

Измерения параметров изображения проводят в затемнённом помещении или при наличии искусственного внешнего освещения. Освещение экрана должно быть диффузным, или угол падения света должен быть равен или более 45 градусов относительно нормали к плоскости, касательной к поверхности экрана в его центре.

Яркость изображения L_u на экране складывается из двух составляющих:

- Яркость излучения $L_{изл}$.
- Отражённая яркость, обусловленная внешним освещением $L_{отр}$.

Яркость изображения в результате рассчитывается по формуле:

$$L_и = L_{изл} + L_{отр}$$

Измерение яркости излучения в затемнённом помещении проводят при освещённости экрана, не превышающей 5 лк.

Отражённую яркость измеряют при выключенном дисплее по следующей формуле:

$$L_{отр} = \frac{E\rho_Д}{\pi}$$, где:

- E – освещённость экрана, лк.
- $\rho_Д$ – коэффициент диффузного отражения экрана дисплея.

Измерение визуальных эргономических параметров проводят с использованием специальных тестовых изображений. Разрешение тестируемого дисплея должно соответствовать рекомендуемому разрешению, указанному в нормативных документах на дисплей.

5.3. Методы и средства защиты пользователей от воздействия на них опасных и вредных факторов

5.3.1. Методы и средства защиты от поражения электрическим током

Зануление — преднамеренное соединение нетоковедущих частей с нулевым защитным проводником.

Защитное зануление применяется в трехфазных сетях с глухо заземленной нейтралью, в установках до 1000В и является основным средством обеспечения электро безопасности.

Принцип защиты пользователей при занулении заключается в отключении сети за счет тока короткого замыкания, который вызывает отключение ПЭВМ от сети.

Для отключения ПЭВМ от сети в случае короткого замыкания или других неисправностей в цепь питания ПЭВМ необходимо ставить автомат с Jном = 8.

5.3.2. Методы и средства защиты от ультрафиолетового излучения

Энергетической характеристикой является плотность потока мощности [Вт/м2]. Биологический эффект воздействия определяется внесистемной единицей эр: 1 эр — это поток (280-315 нм), который соответствует потоку мощностью 1 Вт.

Воздействие ультрафиолетового излучения сказывается при длительной работе за компьютером.

Максимальная доза облучения:

- 7.5 мэр*ч/ м2 за рабочую смену;
- 60 мэр*ч/м2 в сутки.

Для защиты от ультрафиолетового излучения применяют:

- защитный фильтр или специальные очки (толщина стекол 2мм, насыщенных свинцом);
- одежда из фланели и поплина;
- побелка стен и потолка (ослабляет на 45-50%).

5.3.3. Методы и средства защиты от статического электричества

Защита от статического электричества и вызванных им явлений осуществляется следующими способами:

- проветривание без присутствия пользователя;
- влажная уборка;
- отсутствие синтетических покрытий;
- нейтрализаторы статического электричества;
- подвижность воздуха в помещении не более 0.2 м/с;
- иметь контурное заземление.

Для защиты от статического электричества предусмотрены специальные шнуры

питания с встроенным заземлением. Там, где это не используется (отсутствует розетка) необходимо заземлять корпуса оборудования.

Также для защиты от воздействия электрического тока все корпуса оборудования, клавиатура, защелки дисководов и кнопки управления выполнены из изоляционного материала.

Для уменьшения влияния статического электричества необходимо пользоваться рабочей одеждой из малоэлектризующихся материалов, например халатами из хлопчатобумажной ткани, обувью на кожаной подошве. Не рекомендуется применять одежду из шелка, капрона, лавсана.

5.3.4. Методы и средства защиты от электромагнитных полей низкой частоты

Защита от электромагнитных излучений осуществляется следующими способами:

- время непрерывной работы — не более 4 часов в сутки, суммарное время работы за неделю — не более 20 часов;
- расстояние — не менее 50 см от источника;
- экранирование экрана монитора, поверхность экрана покрывается слоем оксида олова, либо в стекло ЭЛТ добавляется оксид свинца;
- расстояние между мониторами — не менее 1,5 м;
- не работать слева от монитора ближе 1.2 м, сзади — 1 м.

5.3.5. Общие рекомендации при работе с вычислительной техникой

Для защиты от вредных факторов имеющих место при эксплуатации ЭВМ необходимо придерживаться следующих рекомендаций:

- правильно организовывать рабочие места;
- правильно организовать рабочее время оператора, соблюдая ограничения

при работе с вычислительной техникой.

5.4. Выводы

Выбранные методы и способы защиты пользователей от воздействии на них опасных и вредных факторов, при соблюдении эргономических требований, позволяют обеспечить безопасную работу и здоровье.

Заключение

Итоги

В результате проделанной работы были выполнены следующие задачи:

- сформированы требования к создаваемой системе;
- проанализированы существующие технологические решения и подходы к организации отказоустойчивых кластеров серверов доменных имен на соответствие сформированным требованиям;
- принято решение о создании новой системы управления и репликации базы данных сервера доменных имен на основе использования сервера доменных имен ISC BIND с расширением DLZ;
- произведен выбор подхода для реализации сервиса управления данными;
- произведен выбор модели репликации данных;
- произведен выбор инструментария для реализации системы;
- спроектирована общая архитектура системы, предполагающая ее разделение на 2 приложения: система синхронизации БД и система управления БД;
- спроектирован и разработан web-сервис для удаленного чтения/записи в базу данных (далее БД) с учетом специфики хранимых данных;
- спроектирована и разработана система репликации удаленных БД, позволяющая объединять несколько (2 и более) инсталляций системы в единый кластер;
- разработан комплекс функциональных тестов, покрывающих основные функции для всех компонентов системы;
- произведено внедрение системы в рабочий процесс компании

ЗАО «Флант».

В результате выполнения данных задач была повышена эффективность работы со службой доменных имен и ее отказоустойчивость.

Выводы

- Служба доменных имен является важнейшим механизмом создания иерархичных пространств имен для разделения глобальной сети на домены. При этом на рынке решений с открытым исходным кодом отсутствуют решения для построения отказоустойчивой инфраструктуры системы доменных имен с потенциально неограниченным горизонтальным масштабированием в виде сетевого сервиса, с возможностью простой интеграции в существующую сетевую инфраструктуру.
- Разработанная система представляет собой универсальное решение для управления системой доменных имен, которое может быть использовано как для внутренних нужд компании, так и для таких услуг, как хостинг.
- Использование Open Source технологий позволяет создать решение, независимое от сторонних вендоров, легко расширяемое и поддерживаемое. Открытость исходного кода самой системы позволит в дальнейшем привлечь к системе Open Source-сообщество и поддерживать высокое качество продукта.

Список литературы

1. W3Schools, SOAP Tutorial,

   http://www.w3schools.com/soap/default.asp

2. Shawn Mehan, Сравнение SOAP и REST, 2011,

   http://seanmehan.globat.com/blog/2011/06/17/soap-vs-rest/

3. Habrahabr, сборник новостей и аналитических статей на тему IT-технологий, Сравнение SOAP и REST, 2011,

   http://habrahabr.ru/post/131343/

4. Habrahabr, сборник новостей и аналитических статей на тему IT-технологий, Веб-сервисы, 2009,

   http://habrahabr.ru/post/75248/

5. Habrahabr, сборник новостей и аналитических статей на тему IT-технологий, RESTful API для сервера, 2012,

   http://habrahabr.ru/post/144011/

6. Habrahabr, сборник новостей и аналитических статей на тему IT-технологий, Асинхронное программирование: концепция Deferred, 2009, http://habrahabr.ru/post/51762/

7. CITforum, Коллекция статей, посвященная Web-сервисным технологиям, CORBA - Архитектура распределенных объектов, 1998,

   http://citforum.ru/database/articles/corba.shtml

8. CITforum, Коллекция статей, посвященная Web-сервисным технологиям, Database Replication, 1997,

   http://citforum.ru/database/digest/dig_1606.shtml

9. Corba, Официальный сайт Corba, 2013, http://corba.org/

10. PSDN Software Developers Network, The advantages of CORBA, 2013, http://documentation.progress.com/output/Iona/e2a/asp/5.1/mainframe/ConceptsGuide/cgCORBAConcepts7.html
11. Washington University, Techniques for Enhancing Real-time CORBA Quality of Service, 2002, http://www.cs.wustl.edu/~schmidt/PDF/IEEE-proc.pdf
12. IBM, Официальный сайт, RESTful Web services, 2008, https://www.ibm.com/developerworks/webservices/library/ws-restful/
13. IBM, Официальный сайт, Репликация в PostgreSQL, 2012, http://www.ibm.com/developerworks/ru/library/os-07_sql/
14. CoderShip Galera Cluster for MySQL, Galera clustering, 2013, http://www.codership.com/wiki/doku.php?id=info#galera_clustering
15. PostgreSQL книга, 2012, https://github.com/le0pard/postgresql_book
16. Brown University, Introduction to Asynchronous Programming, 2013, http://cs.brown.edu/courses/cs168/f12/handouts/async.pdf
17. Boost, The Proactor Design Pattern: Concurrency Without Threads, 2013, http://www.boost.org/doc/libs/1_53_0/doc/html/boost_asio/overview/core/async.html
18. pts.blog, Feature comparison of Python non-blocking I/O libraries, 2013, http://ptspts.blogspot.ru/2010/05/feature-comparison-of-python-non.html
19. Git Wiki, Проекты использующие Git, 2013, https://git.wiki.kernel.org/index.php/GitProjects
20. Python package index, BSDDB3, 2013, https://pypi.python.org/pypi/bsddb3/
21. Язык программирования Python, Официальный сайт, 2013,

http://www.python.org/

22. ISC BIND, Официальный сайт, 2013, https://www.isc.org/software/bind
23. BIND DLZ, Официальный сайт, 2013, http://bind-dlz.sourceforge.net
24. Berkeley DB, Официальный сайт, 2013, http://www.oracle.com/technetwork/products/berkeleydb/overview/index.html
25. Twisted Matrix Labs, Фреймворк Twisted, 2013, http://twistedmatrix.com/
26. ГОСТ, 12.0.003-86, Опасные и вредные производственные факторы. Классификация, 1986
27. ГОСТ, Р 50949-96, Средства отображения информации индивидуального пользования. Методы измерения и оценки эргономических параметров и параметров безопасности, 1996
28. ГОСТ, 12.1.030-81 Электробезопасность. Защитное заземление, зануление, 1981
29. САНП, САНП и Н 1340-03 Гигиенические требования к персональным ЭВМ и организация работы, 20-3
30. ГОСТ, ССБТ 12.1. 124-84 Средства защиты от статического электричества, 1984
31. ГОСТ, 12.0.003-86 Опасные и вредные производственные факторы. Классификация, 1986
32. ГОСТ, ССБТ 1212.1.045-84 Электростатические поля. Допустимые условия на рабочем месте., 1984

Приложения

Приложение 1. Акт о внедрении