МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Государственное образовательное учреждение

высшего профессионального образования

«Донской государственный технический университет»

ДГТУ

Кафедра «ПОВТ и АС»

УТВЕРЖДАЮ

Зав.каф._____ Нейдорф Р.А.

"_____"_____2011г.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к преддипломной практике на тему:

Распределенная файловая система

Практикант Лубягов Николай Александрович_____ Группа __ВИ–51_____

Специальность __230105 Программное обеспечение вычислительной техники и автоматизированных систем_____

Руководитель работы _____ / Клубков Иван Михайлович_____/

Нормоконтролер _____ / Котельникова Ирина Владимировна /

Ростов-на-Дону

2011г.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Государственное образовательное учреждение

высшего профессионального образования

«Донской государственный технический университет»

ДГТУ

Кафедра «ПОВТ и АС»

УТВЕРЖДАЮ

Зав.каф._____ Нейдорф Р.А.

"_____"_____2011г.

ЗАДАНИЕ

на преддипломную практику

Практикант Лубягов Николай Александрович_____ Группа__ВИ-51____

Тема _Распределенная Файловая Система_____

Исходные данные для преддипломной практики: _Параллельная кластерная файловая система для высокопроизводительных вычислительных систем [Электронный ресурс] -– http://www.swsys.ru/index.php?page=article&id=375___

Руководитель работы _____ /Клубков Иван Михайлович_____/

Задание принял к исполнению _____ /Лубягов Николай Александрович/

РЕФЕРАТ

Отчет содержит: листов - 62, рисунков - 4, приложений – 1.

Ключевые слова: ФАЙЛОВАЯ СИСТЕМА, ПРОТОКОЛ, ОБМЕН ФАЙЛАМИ, КЛИЕНТ, СЕРВЕР, ПАРСЕР, XML, КЛАСТЕР, ИНФОРМАЦИЯ.

В данном отчете по преддипломной практике «Распределенная файловая система» рассматривается:

- современные распределенные файловые систем;
- современные инструменты построения масштабируемых файловых систем;
- разработка схемы построения распределенной файловой системы;
- разработка протокола обмена данными файловой системы;
- разработка протокола обмена запросами на операции с файлами.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Сетевая файловая система является программным продуктом, позволяющим производить монтирование каталогов в сети и трактовать удаленные файлы как локальные. Идея сетевых файловых систем далеко ненова. Это значит, что первые реализации файловых систем, внешних по отношению к физическим ЭВМ, появились в начале 80х - это файловые системы для VAX-VMS кластеров. Эту идею развивали многие компании, в результате чего мы по сути имеем несколько фундаментальных концептов разделяемых файловых систем, не говоря уже о количестве конкретных реализаций данных концептов. Фундаментальные концепты включают сетевые файловые системы (NFS), параллельные файловые системы (Lustre) и симметричные блочно-разделяемые файловые системы (GFS).

Распределенная сеть машин, может обеспечить более агрегированную производительность вычислительного оборудования по сравнению с большой ЭВМ. Сетевая обработка данных вообще более эффективна, нежели централизованная обработка.[1]

1 ОБЗОР СЕТЕВЫХ ФАЙЛОВЫХ СИСТЕМ

1.1 Текущее состояние вопроса

Уже прошли те времена, когда сайтов было мало и пользователи были согласны ждать, пока восстановят функционирование ненадежного сайта. В Интернете при столь высокой конкуренции даже пятиминутный сбой в работе сервиса вызывает отток пользователей к конкуренту. Поэтому каждый, кто хочет построить устойчивый сервис, должен начать с ее основ – отказоустойчивой файловой системы. Кластерные файловые системы еще не достаточно приспособлены для использования на крупных предприятиях: обычно процесс их развертывания и поддержания в работающем состоянии не так уж прост. Но зато они отлично масштабируются и достаточно дешевы, ведь для них достаточно самого простого серверного оборудования и свободных операционных систем и программного обеспечения.

1.2 Обзор существующих аналогов

Существует достаточно большое число файловых систем работающих по сети и организующих доступ к файлам на удаленных компьютерах. Среди таких файловых систем:

- SSHFS — предоставляет доступ к удалённой ФС через SSH;
- GmailFS — файловая система, которая хранит данные как почту в Gmail;
- WikipediaFS — просмотр и редактирование статей Википедии так, как будто они являются файлами;
- CurlFtpFS — предоставляет доступ к удалённой ФС через libcurl;
- LftpFS — предоставляет доступ к удалённой ФС через lftp;
- Andrew File System (AFS) — Сетевая кластерная файловая система;

- GlusterFS — высокопроизводительная кластерная ФС;
- GoogleFS — кластерная система;
- Lustre — кластерная файловая система.

Наиболее близкими файловыми системами к поставленной задаче являются AFS, GlusterFS и Lustre.

1.2.1 Google File System

Кластерная система оптимизированная для работы с большими блоками данных по 64 Мб (chunk), а также обладающая повышенной защитой от сбоев. Вся информация копируется и хранится в трёх (или более) местах одновременно, при этом система способна очень быстро находить реплицированные копии, если какая-то машина вышла из строя. Задачи автоматического восстановления после сбоя решаются с помощью программ, созданных по модели MapReduce. Является коммерческой тайной компании Google. Несовместима с POSIX и создавалась Google для своих внутренних потребностей[2].

1.2.2 SSHFS (Secure SHell FileSystem)

SSHFS (Secure SHell FileSystem) это файловая система для Linux (и других операционных систем, для которых существует реализация FUSE (Filesystem in Userspace), например Mac OS X), используемая для удаленного управления файлами по протоколу SSH (точнее, его расширению SFTP) таким образом, как будто они находятся на локальном компьютере. Администратор может настроить ограниченный аккаунт на сервере для обеспечения большей безопасности и пользователь сможет видеть только выделенную ему область в системе. [3]

1.2.3 GmailFS

Google бесплатно позволяет создавать почтовые ящики размером 6Гб. Достаточно интересным является использование GMail в качестве хранилища файлов. GmailFS позволяет пользователю смонтировать удаленную файловую систему, на которой он может использовать до 6 Гб дискового пространства.[4]

1.2.4 WikipediaFS

Это монтируемая Linux виртуальная файловая система позволяющая читать и редактировать статьи из Википедии и других сайтов основанных на Mediawiki так, как будто это обычные файлы. Что позволяет просматривать и редактировать статьи, используя обычный текстовый редактор. Текстовые редакторы, как правило, более удобны, чем просто формы браузера, когда речь идет о редактировании больших текстов, и они, как правило, включают такие полезные функции, как подсветка синтаксиса Mediawiki и проверка орфографии[5].

1.2.5 CurlFtpFS

Это файловая система для доступа к FTP хостам основанная на FUSE и libcurl, в отличии от других FTP файловых систем она имеет следующие функции: поддержка SSLv3 и TLSv1 шифрования, подключение через HTTP прокси, автоматическое переподключение в случае разрыва соединения, преобразовывает абсолютные ссылки в точку возврата на FTP-файловой системе[6].

1.2.6 LftpFS

Сетевая файловая система с доступом только для чтения, интеллектуально кэширующая зеркала(mirrors) сайтов. Используется для

создания копий Linux репозиториев (repositories). Она основана на FUSE и LFTP клиенте и поддерживает FTP, HTTP, FISH, SFTP, HTTPS, FTPS протоколы, позволяет работать с ней через прокси[7].

1.2.7 Andrew File System (AFS)

Продукт AFS представляет собой распределенную файловую систему, изначально разработанную в университете Карнеги-Меллона, она поддерживается и развивается как продукт корпорации Transarc (в настоящее время IBM Pittsburgh Labs). Она предлагает архитектуру клиент-сервер для обмена файлами, распределенного доступа содержимого только для чтения, распространения содержимого, предоставляя возможности независимости, масштабируемости, безопасности и свободной миграции. AFS доступна для широкого диапазона операционных систем, включая UNIX, Linux, MacOS X и Microsoft Windows. Она имеет несколько преимуществ по сравнению с традиционными сетевыми файловыми системами, в частности в области безопасности и масштабируемости. Секция AFS может поддерживать более двадцати пяти тысяч клиентов. AFS использует Kerberos (компьютерный сетевой протокол аутентификации, позволяющий отдельным пользователям общаться через незащищённые сети для безопасной идентификации. Так же является набор бесплатного ПО от Массачусетского Технологического Института (Massachusetts Institute of Technology (MIT)), разработавшего этот протокол. Его организация направлена в первую очередь на клиент-серверную модель и обеспечивает взаимную аутентификацию - оба пользователя через сервер подтверждают личности друг друга. Сообщения, отправляемые через протокол Kerberos, защищены от прослушивания и атак.) для аутентификации и осуществляет списки контроля доступа на директории для пользователей и групп. Каждый клиент кэширует файлы на локальную файловую систему, что

увеличивает скорость при последующих запросах на тот же файл. Это также позволяет иметь ограниченный доступа к файловой системы в случае сбоя сервера или коммерческих сетей[8].

IBM выделил из исходного продукта AFS, и сделал копию исходного кода для сообщества разработчиков и технического обслуживания. Она получила название Open AFS[9].

1.2.8 Lustre

Lustre представляет собой кластерную файловую систему, основными особенностями которой являются превосходные надежность и масштабируемость. Производительность также более чем высока — скорость передачи данных может достигать сотен гигабит в секунду, а теоретический максимум доступного дискового пространства измеряется петабайтами. Эта файловая система может использоваться как на скромных рабочих группах из нескольких компьютеров, так и на огромных кластерах, насчитывающих десятки тысяч машин[10].

Помимо этого поддерживаются все возможности, который должна иметь любая уважающая себя кластерная файловая система:

- поддержка широкого ассортимента типов высокоскоростных сетевых соединений;
- надежная система «замков» для обеспечения параллельного доступа к файлам;
- возможность автоматического самовосстановления в случае падения любого из узлов;
- распределенное управление файловыми объектами для предоставления масштабируемого доступа к файлам.

Изначально архитектура этой файловой системы была разработана просто

в рамках исследовательского проекта Петера Браама в 1999, но он решил не останавливаться на достигнутом и основал Cluster File Systems, Inc., в которой уже и велась основная разработка самой файловой системы. Первый релиз Lustre 1.0 был выпущен в 2003 году. Спустя четыре года компания была приобретена Sun Microsystems в октябре 2007 года, но это лишь способствовало дальнейшему развитию проекта. Программное обеспечение, входящее в состав проекта, выпускается под лицензией GPL, что также сыграло немаловажную роль в его жизни.

Архитектура

Каждый компьютер, входящий состав кластера Lustre, выполняет свою четко определенную функцию:

- MDS — сервер метаданных предназначен для хранения всей служебной информации о системе: названия файлов, директорий, прав доступа и так далее. Достаточно наличие одного такого сервера в системе, но для обеспечения надежности на случай каких-либо сбоев, обычно его дублируют. Возможно использование внешнего хранилища данных (MDT), которое может быть общим для двух дублирующих друг друга MDS;
- OSS — компьютеры для хранения самих данных. Каждый из них работает с 2-8 OST, в их роли могут выступать практически любые средства хранения данных, начиная от просто жестких дисков или RAID массивов внутри OSS, заканчивая внешними системами хранения данных enterprise-класса. Сумма дискового пространства всех OST и является размером доступного дискового пространства всей файловой системы Lustre;
- клиент — компьютеры, непосредственно использующие файловую

систему. Им предоставляется полный параллельный доступ, полностью соответствующий стандарту POSIX.

Один и тот же компьютер теоретически может совмещать в себе несколько функций, но в большинстве случаев это нецелесообразно (за исключением совмещения клиентов с OST и, возможно, случаев, когда количество узлов кластера очень мало). Написанное выше наглядно можно представить схемой архитектуры системы (рис. 1).

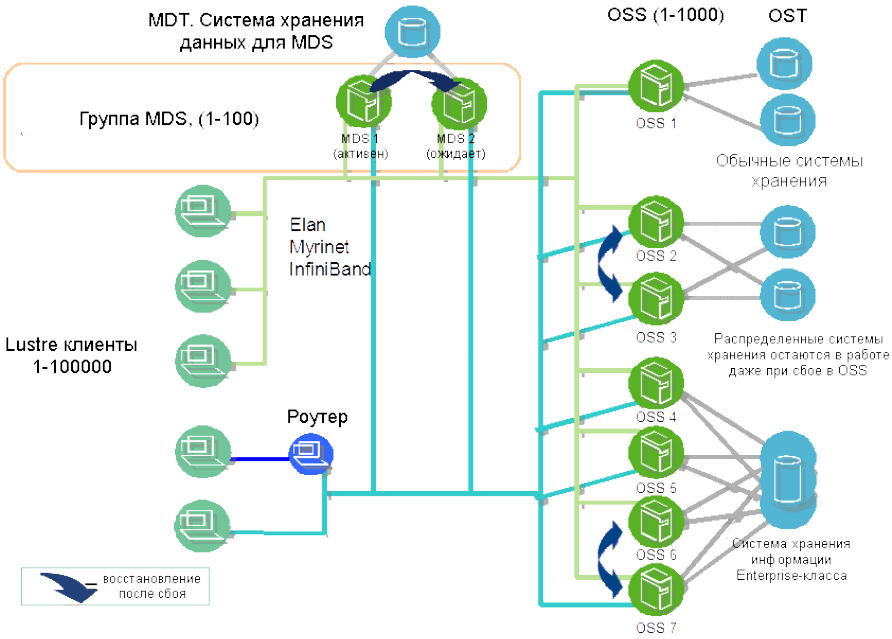


Рисунок 1 – Схема архитектуры системы Lustre

Помимо этого для функционирования системы необходим еще один компонент, по большому счету не являющийся ее частью — MGS. Его роль заключается в предоставлении конфигурационной информации всем компонентам одной или нескольким файловым системам Lustre. Он также нуждается в отдельном хранилище данных, но теоретически он может быть совмещен с одним из компонентов файловой системы.

Функционирование

Основным толчком для выполнения каких-либо действий в рамках всей файловой системы обычно является запрос с одного из клиентов. Программное обеспечение для клиентов представляет по сути интерфейс между виртуальной файловой системой Linux и серверами Lustre. Каждому типу серверов соответствует своя часть клиентского ПО: MDC, OSC, MGT. В отличии от Hadoop и GFS файловая система Lustre должна быть примонтирована к локальной системе клиентов для полноценного их функционирования.

Для осуществления коммуникации между клиентами и серверами используется собственный API, известный как LNET. Он поддерживает множество сетевых протоколов с помощью NAL.

В системе отсутствуют незаменимые компоненты, это является залогом отказоустойчивости системы. В случае возникновения какие-либо неполадок или сбоев в работе оборудования, работу вышедших из строя компонентов системы перехватят другие ее компоненты, что сделает сбой незаметным для пользователей системы. Это достигается за счет дублирования серверов, выполняющих одинаковые функции, а также наличие налаженных алгоритмов действий, направленных на автоматическое восстановление полноценного функционирования системы в случае возникновения чрезвычайных ситуаций. Но этого конечно же не достаточно для абсолютной надежности системы, в дополнение должна быть предоставлена как минимум система бесперебойного питания для всех компонентов кластера на случай проблем с электроэнергией в датацентре (для России более чем актуально).

В списке дополнительных возможностей, предоставляемых файловой системой, можно назвать возможность выделения квот на дисковое пространство для каждого пользователя системы, аутентификацию пользователей с помощью механизма Kerberos, повышение физической

пропускной способности сетевого соединения путем агрегирования физических сетевых соединений в одно логическое виртуальное сетевое соединение (достаточно интересная возможность, способная при выполнении определенных условий существенно повлиять на быстродействие системы). Помимо этого предоставляется целый ряд возможностей по созданию резервных копий данных на уровне файловой системы в целом, отдельных устройств или же файлов.

Эта файловая система нашла свое применение во множестве крупнейших кластеров и суперкомпьютеров по всему миру, но это не мешает ей с тем же успехом демонстрировать и на кластерах существенно меньшего масштаба. Около половины из самых производительных суперкомпьютеров во всем мире используют Lustre в качестве файловой системы. Помимо этого многие компании предоставляют ее в качестве основы для Linux кластеров (например HP StorageWorks SFS, Cray XT3, Cray XD1).

1.2.9 GlusterFS

Кластерная файловая система, способная хранить до нескольких пета-байт информации. Она позволяет соединять в одну большую параллельную сетевую файловую систему несколько хранилищ данных, посредством Infiniband RDMA или TCP / IP соединения. GlusterFS основана на создании стека дисковых пространств пользователей без ущерба для производительности.[11]

Особенности

GlusterFS представляет собой кластерную файловую систему, способную масштабироваться для хранения до нескольких петабайта данных. Как и многие другие кластерные файловые системы, GlusterFS агрегирует дисковое пространство большого количества машин в одну общую параллельную сетевую файловую систему через Infiniband RDMA или TCP/IP соединение. Обычно в

качестве аппаратной основы для этой файловой системы используется недорогое серверное оборудование, в полной мере реализуя принцип программного построения стабильности при использовании на ненадежном оборудовании[12]. Список основных ее особенностей по большей части мало чем отличается от других кластерных файловых систем:

- ФС состоит из клиентской и серверной частей. Клиентская часть позволяет монтировать файловую систему, а серверная — glusterfsd — экспортировать в нее локальное дисковое пространство;
- масштабируемость близка к O (1);
- широкий спектр возможностей за счет использования модульной архитектуры;
- имеется возможность восстановления файлов и директорий из файловой системы даже без ее инициализации;
- отсутствие централизованного сервера метаданных, что делает ее более устойчивой к потенциальным сбоям;
- расширяемый интерфейс выполнения задач, с поддержкой загрузки модулей в зависимости от особенностей выполнения пользователями операций по работе с данными;
- расширяющий функциональность механизм трансляторов;
- поддержка Infiniband RDMA и TCP/IP;
- возможность автоматического восстановления в случае сбоев;
- полностью реализована на уровне приложений, что упрощает ее поддержание в рабочем состоянии, портирование и дальнейшую разработку.

Но некоторые моменты все же заслуживают отдельного внимания.

Совместимость

Как уже упоминалось, файловая система реализована полностью на уровне пользовательских приложений, что делает возможным ее монтирование без каких-либо дополнительных патчей в ядре операционной системы, единственное требование к нему: поддержка FUSE. Серверная часть GlusterFS может функционировать на любой POSIX-совместимой операционной системе и протестирована на Linux, FreeBSD, OpenSolaris, в отличии от клиентской части, которая может работать только в Linux.

Модули

В виде модулей реализованы различные варианты выполнения основополагающих операций: передачи данных и балансировки нагрузки в рамках кластера. Транспортные модули обеспечивают передачу данных по различным типам соединений:

- TCP/IP;
- infiniband-verbs;
- infiniband-SDP.

Балансировка нагрузки может выполняться по следующим алгоритмам:

- ALU — использует целый ряд факторов, включающий объем свободного локального дискового пространства, активность операций чтения и записи, количество одновременно открытых файлов, скорость физического вращения дисков. Значимость, придаваемая каждому из показателей, может достаточно гибко настраиваться;
- RR — по очереди размещает файлы последовательно на каждом узле, после чего начинает процесс заново, образуя своеобразный цикл. Этот метод эффективен если файлы имеют примерно одинаковый размер, а узлы кластера — одинаковый размер экспортированного локального

дискового пространства;

- Random — распределяют файлы случайным образом;
- NUFA — приоритет отдается созданию файлов локально, а не на других узлах кластера;
- Switch — располагает файлы по определенным указанным особенностям имен файлов, по аналогии со switch (filename) в программировании, обычно в качестве критерия распределения файлов имеет смысл использовать их расширение.

Трансляторы

Они представляют собой очень мощный механизм для расширения возможностей GlusterFS, сама идея трансляторов бала позаимствована у GNU/Hurd и заключается она в загрузке бинарных библиотек (.so) в процессе работы системы в зависимости от использованных настроек и использовании их в виде своеобразной цепочки обработчиков при работе с файлами как на серверной, так и на клиентской стороне. В GlusterFS практически все дополнительные возможности реализованы именно виде трансляторов, начиная от дополнений, увеличивающих производительность, заканчивая средствами отладки. Вкратце перечислю основные из них:

- AFR — автоматическая репликация файлов;
- Stripe — разбивает файлы на блоки фиксированного размера;
- Unify — объединяет несколько узлов кластера в один большой виртуальный узел, один узел выделяется для обеспечения внутреннего namespace. Директории создаются на всех узлах, составляющих unify, а каждый файл — лишь на одном (если не используется AFR);
- Trace — предоставляют информацию для отладки в виде дополнительных записей в лог;

- Filter — фильтрация файлов на основании их имен и/или атрибутов;
- Posix-locks — обеспечивает POSIX блокировку записей независимую от используемой системы хранения;
- Trash — предоставляет функциональность сопоставимую с libtrash (или «корзиной» — если так понятнее);
- Fixed-id — обеспечивает доступ только для пользователей с определенными UID и GUID;
- Posix — соединяет GlusterFS с низлежащей локальной файловой системой;
- rot-13 — транслятор обеспечивает возможность шифрования и дешифрования данных по примитивному одноименному алгоритму.

Благодаря возможности работы через Infiniband (высокоскоростная коммутируемая последовательная шина, применяющаяся как для внутренних (внутрисистемных), так и для межсистемных соединений[13]) производительность передачи данных также достаточно высока — она может достигать десятков гигабит в секунду. Обработка сбоев в отдельных узлах также осуществляется достаточно эффективно, так как может быть автоматизирована. Из потенциальных недостатков можно назвать некоторое количество редко проявляющих себя багов в коде, а также достаточно большой размер заголовков в используемом протоколе (несколько сотен байт). В целом эта система вполне работоспособна и полноценно выдерживает конкуренцию со стороны своих opensource «коллег».

1.2.10 Hadoop и HDFS

Hadoop представляет собой платформу для построения приложений, способных обрабатывать огромные объемы данных[14]. Система основывается на распределенном подходе к вычислениям и хранению информации, основными ее особенностями являются:

- масштабируемость: с помощью Hadoop возможно надежное хранение и обработка огромных объемов данных, которые могут измеряться петабайтами;
- экономичность: информация и вычисления распределяются по кластеру, построенному на самом обыкновенном оборудовании. Такой кластер может состоять из тысяч узлов;
- эффективность: распределение данных позволяет выполнять их обработку параллельно на множестве компьютеров, что существенно ускоряет этот процесс;
- надежность: при хранении данных возможно предоставление избыточности, благодаря хранению нескольких копий. Такой подход позволяет гарантировать отсутствие потерь информации в случае сбоев в работе системы;
- кроссплатформенность: так как основным языком программирования, используемым в этой системе является Java, развернуть ее можно на базе любой операционной системы, имеющей JVM.

HDFS

В основе всей системы лежит распределенная файловая система под незамысловатым названием Hadoop Distributed File System. Представляет она собой вполне стандартную распределенную файловую систему, но все же она обладает рядом особенностей:

- устойчивость к сбоям, разработчики рассматривали сбои в оборудовании скорее как норму, чем как исключение;
- приспособленность к развертке на самом обыкновенном ненадежном оборудовании;
- предоставление высокоскоростного потокового доступа ко всем данным;
- настроена для работы с большими файлами и наборами файлов;
- простая модель работы с данными: один раз записали — много раз прочли;
- следование принципу: переместить вычисления проще, чем переместить данные.

Архитектура HDFS

Проще всего ее демонстрирует схема, позаимствованная с официального сайта проекта и переведенная на русский язык (рис. 2).

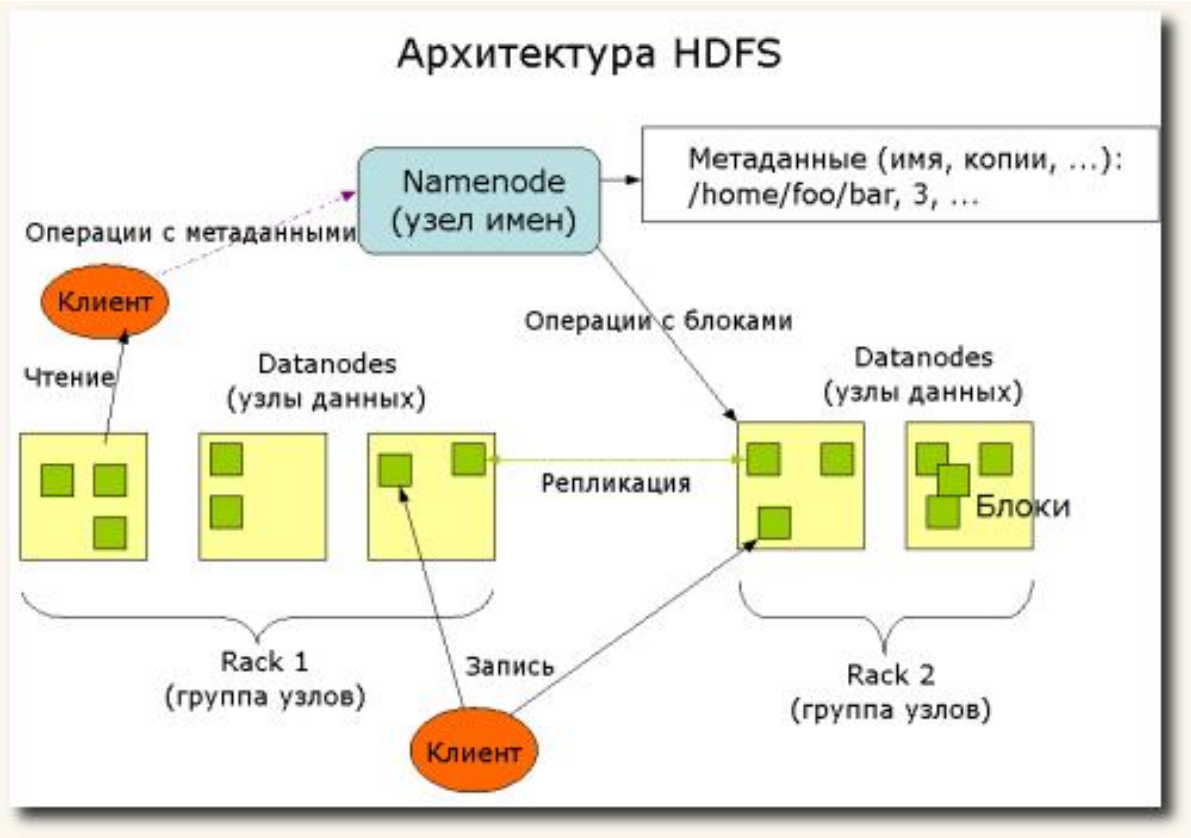


Рисунок 2 – Схема архитектуры HDFS

Компоненты HDFS:

- namenode — этот компонент системы осуществляет всю работу с метаданными. Он должен быть запущен только на одном компьютере в кластере. Именно он управляет размещением информации и доступом ко всем данным, расположенным на ресурсах кластера. Сами данные проходят с остальных машин кластера к клиенту мимо него;
- datanode — на всех остальных компьютерах системы работает именно этот компонент. Он располагает сами блоки данных в локальной файловой системе для последующей передачи или обработки их по запросу клиента. Группы узлов данных принято называть Rack, они используются, например, в схемах репликации данных;
- клиент — приложение или пользователь, работающий с файловой системой. В его роли может выступать практически что угодно.

Пространство имен HDFS имеет классическую иерархическую структуру: пользователи и приложения имеют возможность создавать директории и файлы. Файлы хранятся в виде блоков данных произвольной (но одинаковой, за исключением последнего; по-умолчанию 64 mb) длины, размещенных на Datanode'ax. Для обеспечения отказоустойчивости блоки хранятся в нескольких экземплярах на разных узлах, имеется возможность настройки количества копий и алгоритма их распределения по системе. Удаление файлов происходит не сразу, а через какое-то время после соответствующего запроса, так как после получения запроса файл перемещается в директорию /trash и хранится там определенный период времени на случай, если пользователь или приложение передумают о своем решении. В этом случае информацию можно будет восстановить, в противном случае — физически удалить.

Для обнаружения возникновения каких-либо неисправностей, Datanode периодически отправляют Namenode'у сигналы о своей работоспособности. При прекращении получения таких сигналов от одного из узлов, Namenode помечает его как «мертвый» и прекращает какое-либо с ним взаимодействие, до возвращения его работоспособности. Данные, хранившиеся на «умершем» узле реплицируются дополнительный раз из оставшихся «в живых» копий, и система продолжает свое функционирование, как ни в чем не бывало.

Все коммуникации между компонентами файловой системы проходят по специальным протоколам, основывающимся на стандартном TCP/IP. Клиенты работают с Namenode с помощью так называемого ClientProtocol, а передача данных происходит по DatanodeProtocol, оба они обернуты в Remote Procedure Call (RPC).

Система предоставляет несколько интерфейсов, среди которых командная оболочка DFSShell, набор ПО для администрирования DFSAdmin, а также простой, но эффективный веб-интерфейс. Помимо этого существуют

несколько API для языков программирования: Java API, C pipeline, WebDAV и так далее.

MapReduce

Помимо файловой системы, Hadoop включает в себя framework для проведения масштабных вычислений, обрабатывающих огромные объемы данных. Каждое такое вычисление называется Job (задание) и состоит оно, как видно из названия, из двух этапов:

- map — целью этого этапа является представление произвольных данных (на практике чаще всего просто пары ключ-значение) в виде промежуточных пар ключ-значение. Результаты сортируются и групируются по ключу и передаются на следующий этап;
- reduce — полученные после map значения используются для финального вычисления требуемых данных. Практически любые данные могут быть получены таким образом, все зависит от требований и функционала приложения.

Задания выполняются, подобно файловой системе, на всех машинах в кластере (чаще всего одних и тех же). Одна из них выполняет роль управления работой остальных — JobTracker, остальные же ее беспрекословно слушаются — TaskTracker. В задачи JobTracker'a входит составление расписания выполняемых работ, наблюдение за ходом выполнения, и перераспределение в случае возникновения сбоев.

В общем случае каждое приложение, работающее с этим framework'ом, предоставляет методы для осуществления этапов map и reduce, а также указывает расположения входных и выходных данных. После получения этих данных JobTracker распределяет задание между остальными машинами и предоставляет клиенту полную информацию о ходе работ.

Помимо основных вычислений могут выполняться вспомогательные

процессы, такие как составление отчетов о ходе работы, кэширование, сортировка и так далее.

1.3 Выбор библиотеки для реализации файловой системы

1.3.1 FUSE

Filesystem in Userspace (FUSE) (Файловая система в пользовательском пространстве) — это модуль для ядер Unix-подобных ОС, с открытым исходным кодом и относящийся к свободному программному обеспечению. Модуль распространяется под лицензиями GNU GPL и GNU LGPL. Он позволяет пользователям без привилегий создавать их собственные файловые системы без необходимости переписывать код ядра. Это достигается за счёт запуска кода файловой системы в пространстве пользователя, в то время как модуль FUSE только предоставляет «мост» для актуальных интерфейсов ядра. FUSE была официально включена (слита) в главное дерево кода Linux в версии 2.6.14[15]. С помощью FUSE можно разработать файловую систему в пространстве пользователя без знания внутреннего устройства файловой системы или изучения программирования модулей ядра. FUSE особенно полезна для написания виртуальных файловых систем. В отличие от традиционных файловых систем, которые по существу сохраняют информацию для восстановления данных с диска, виртуальные файловые системы не хранят данные непосредственно. Они действуют как представление, трансляция (перевод) существующей файловой системы или устройства хранения. В принципе, любой ресурс, доступный для использования FUSE, может быть экспортирован в файловую систему. Сама система FUSE была частью проекта A Virtual Filesystem (AVFS), но потом AVFS выделился в собственный проект на SourceForge.net [8].

FUSE позволяет разрабатывать полностью функциональную файловую

систему, которая имеет простую библиотеку API, может использоваться непривилегированными пользователями и обеспечивает защищенную реализацию. К тому-же FUSE обладает доказанной стабильностью.

При помощи FUSE вы можете разработать файловую систему в виде исполняемых двоичных файлов, связанных с библиотеками FUSE; другими словами, эта интегрированная файловая система не требует от вас изучения внутреннего устройства файловой системы или программирования модулей ядра[16].

Что касается файловых систем, то файловая система в пространстве пользователя не является чем-то новым. Приведем несколько примеров коммерческих и академических реализаций таких систем:

- LUFS — гибридная файловая система в пространстве пользователя, поддерживающая неопределенное число файловых систем прозрачно для любого приложения. Она состоит из модуля ядра и демона, работающего в пространстве пользователя. По существу, он делегирует большинство VFS-вызовов специализированному демону, который их обрабатывает;
- UserFS позволяет пользовательским процессам быть смонтированными как нормальные файловые системы. Этот экспериментальный прототип предоставляет программу ftpfs, которая организует анонимный FTP с интерфейсом файловой системы;
- Ufo Project — глобальная файловая система для Solaris, которая позволяет пользователям работать с удаленными файлами так, как будто они являются локальными;
- OpenAFS — это версия с открытыми исходными кодами файловой системы Andrew FileSystem;
- CIFS — Common Internet FileSystem.

Так же с помощью FUSE разработаны уже рассмотренные GlusterFS, GmailFS, CurlFtpFS, SSHFS, WikipediaFS, LftpFS. В отличие от этих коммерческих и академических примеров FUSE переносит возможности этих проектов файловых систем в Linux. Поскольку FUSE использует исполняемые файлы (вместо, например, разделяемых объектов, используемых в LUFS), облегчается отладка и разработка. FUSE работает с обеими версиями ядра (2.4.x и 2.6.x) и теперь поддерживает Java™-связывание, т.е. вы не ограничены программированием файловой системы в C и C++

1.4 Вывод

В следствии исследования предметной области были выявлены положительные особенности файловых систем, которые можно использовать в разрабатываемой системе. Архитектура программы реализующей файловую систему, должна является клиент-серверной, без наличия сервера транзакций, и метаданных, как в GlusterFS в отличии от Lustre и HDFS, это упростит ее функционирование, хотя может вызвать проблемы с конфигурированием нескольких клиентов. Файловая система должна кэшировать файлы на стороне клиента, благодаря чему повышается надежность системы, даже при сбое в сети она продолжит частично функционировать, что так-же упростит функции сохранения данных на другие компьютеры при выключении или выходе из строя компьютера на который производится запись, подобно AFS. Идея трансляторов и модулей, позволит достичь более структурированный код, и даст возможность расширения файловой системы по мере необходимости без модификации самого ядра системы, таким образом как это сделано в GlusterFS. Использовать минимально возможный размер заголовка в пакетах при передаче содержимого файлов, для того чтобы свести к минимуму объем передаваемых по сети данных на заголовках пакетов, в отличии от GlusterFS. Реализовать несколько

алгоритмов выбора компьютера на который производится запись как отдельные модули, реализовать обработку запросов от клиентов отдельными модулями, в том числе аутентификацию и файловые операции, подобно GlusterFS. В протоколе должны быть ping пакеты подобно системе HDFS, которые будут отправляться к клиентам от серверов, и если клиент не получает информацию, в течении некоторого промежутка времени, то сервер будет считаться «умершим» и удален из списка серверов внутри клиента. Реализовать два отдельных независимых протокола для данных, которые передают содержимое файлов и для метаданных, как в Hadoop и HDFS.

1.5 Постановка задачи

Целью данной преддипломной работы является создание распределенной файловой системы. Программа должна создавать единое дисковое пространство для нескольких объединенных в сеть компьютеров, при этом каждый компьютер позволяет записать в данное дисковое пространство некоторые данные, после чего система должна выбрать наиболее подходящий из соединенных компьютеров. Затем будет произведена запись данных на диск данного компьютера. В случае отключения компьютера данные не дублируются и файлы находящиеся в дисковом пространстве данного компьютера будут недоступны, до его включения.

Решение данной задачи разбивается на следующие подзадачи:

- создание протокола обмена данными между клиентом и сервером;
- разработка алгоритмов надежного обмена данными, устойчивого к аппаратным сбоям, и сбоям в сети;
- создание программы клиента, монтирующего файловую систему;
- создание программы сервера, реализующего операции с файлами по запросам клиента.

2 АЛГОРИТМИЧЕСКОЕ КОНСТРУИРОВАНИЕ ФАЙЛОВОЙ СИСТЕМЫ

При построении файловой системы можно выделить две независимых программы, сервер и клиент. Сервер создает соединение и ожидает подключения клиентов, для работы по сети можно использовать соединение по TCP-IP. Клиент подключившись к серверу посылает команды для работы с файлами, сервер выполняет, те или иные действия и возвращает ответы клиенту. Клиент должен одновременно подключаться к группе серверов. Со стороны клиента должен стоять FUSE модуль который позволяет монтировать файловую систему на клиентскую машину, серверная же сторона просто выполняет действия над файлами используя системные вызовы ядра UNIX. В качестве протокола обмена можно разработать протокол на основе XML (eXtensible Markup Language) и обрабатывать его XML парсером по технологии SAX (Simple API for XML). Формат XML позволяет создавать легко расширяемый интуитивно понятный протокол разметки, таким образом при внесении модификаций в новых версиях файловой системы можно добиться, того что старые версии клиентов будут работать с новыми серверами и наоборот без внесения значительных изменений в код программ. Однако XML не позволяет передавать внутри себя поток данных таких как файлы. Поскольку протокол является текстовым, он не может обеспечить передачу символов, которые являются частью его разметки. Для решения данной проблемы обычно использую кодировку формата MIME (Multipurpose Internet Mail Extensions — многоцелевые расширения почты интернета) — стандарт, описывающий передачу различных типов данных по электронной почте, а также, спецификация для кодирования информации и форматирования сообщений таким образом, чтобы их можно было пересылать по Интернету. Однако данный формат вызывает значительный прирост к пересылаемому трафику от сервера к клиенту, что в результате может повлечь значительное замедление функционирования системы. Для решения

поставленной задачи можно использовать второй двоичный протокол, через который будет производиться передача содержимого файлов. В таком случае клиент и сервер буде иметь два постоянно открытых соединения. Через первое соединение будет производиться передача запросов на файловые операции (чтение содержимого директорий, переименование удаление, получение и изменение атрибутов файлов и прочие операции), а по второму протоколу будет производиться непосредственно сама передача данных содержимого файлов.

2.1 Требования к разрабатываемой ФС

При построении файловой системы следует учесть следующие требования:

- в случае ошибки нехватки дискового пространства, данные должны быть переписаны на другую машину, и произведена до-запись. Если подобных машин не найдено, тогда вызывать ошибку;
- в случае отключения компьютера в момент записи должен быть выбран новый компьютер и данные перезаписаны на него, таким образом программа на стороне клиента должна кэшировать данные и производить их запись на выбранный компьютер параллельно;
- программа должна функционировать в операционной системе UNIX и ее разновидностях;
- разрабатываемое приложение должно состоять из двух программ: клиента который подключается к набору серверов и выполняет операции над файлами, и сервера, который реализует файловую систему. Клиент подключается к серверу и производит операции над файлами. Любой клиент может подключаться к одному или более серверов одновременно, любые сервера могут быть подключены к нескольким клиентам;

- файловая система должна поддерживать расширяемые модули для сервера и клиента, благодаря которым будут осуществляться операции с файлами, выбор компьютеров для записи фалов, аутентификация и прочие действия;
- для реализации передачи данных разработать два собственных протокола обмена информацией между сервером и клиентом, один для передачи метаданных, второй для передачи содержимого файлов, благодаря которым будут производиться все поставленные задачи;
- внутри протокола должны быть пакеты поддержания соединения для проверки работоспособности сервера.

2.2 Структура разрабатываемой файловой системы

В результате анализа полученной информации можно выделить основные составляющие части разрабатываемой программы и проиллюстрировать взаимодействие с другими компонентами системы (рис. 3).

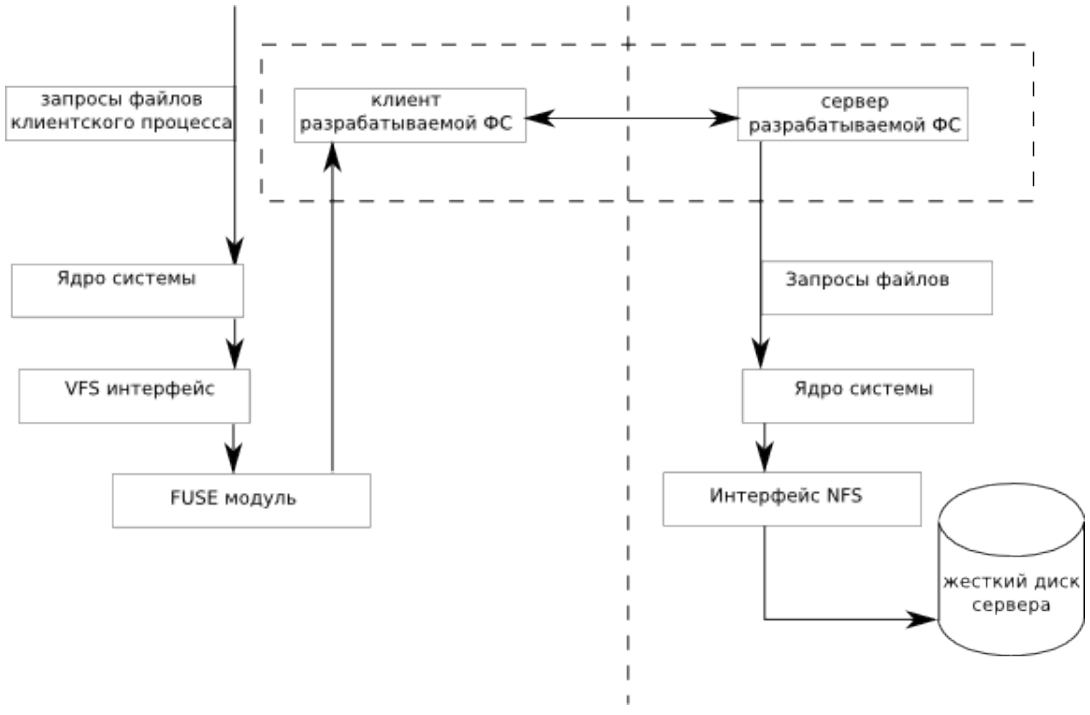


Рисунок 3 – Схема взаимодействие разрабатываемой программы с компонентами системы

Здесь пунктиром выделена та часть которую требуется разработать. Связь соединяющая сервер и клиент является связью многие к многим, т.е. один клиент может подключаться к нескольким серверам и один сервер может быть подключен к нескольким клиентам одновременно.

Структуру создаваемой программы, тогда можно описать следующим образом, (рис. 4).

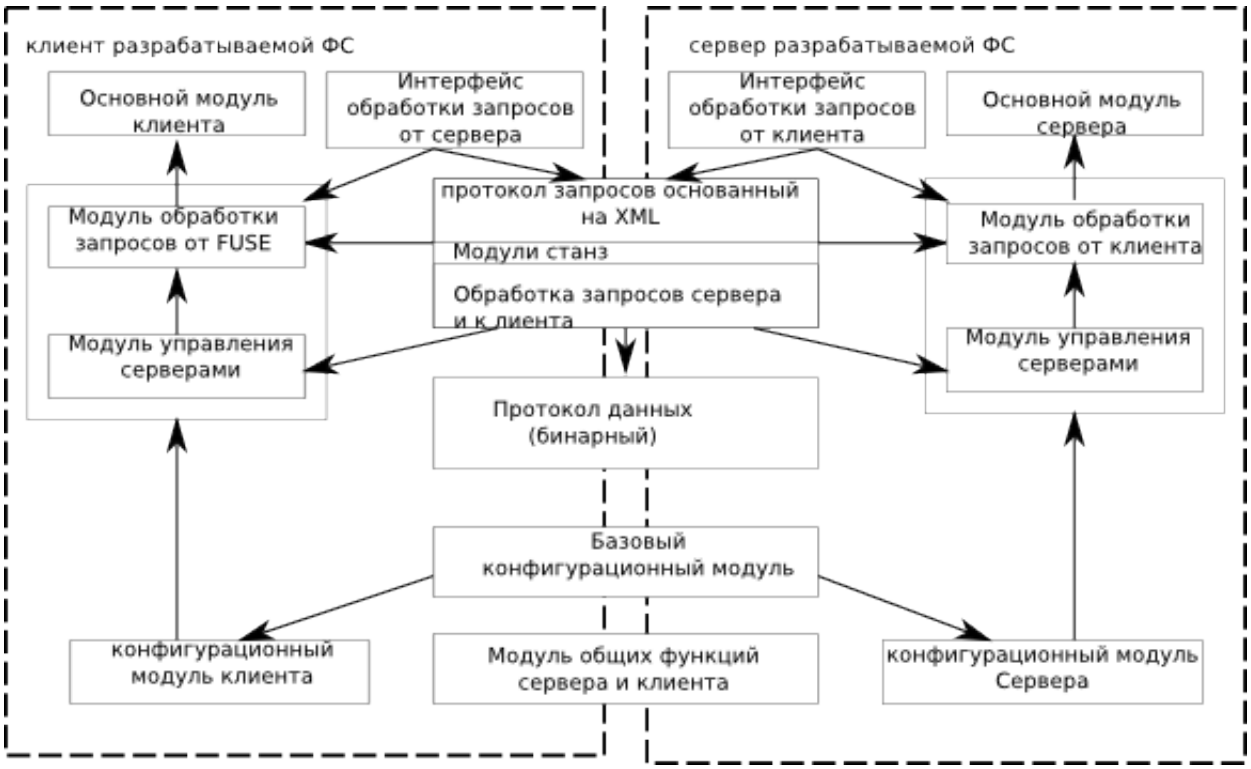


Рисунок 4 – Структурная схема разрабатываемой программы

Таким образом можно выделить около шестнадцати взаимосвязанных модулей.

Общие модули для сервера и клиента:

- протокол запросов основанный на XML — описывает сам протокол работы клиента и сервера;
- модуль станз — в нем описываются станзы которыми обменивается сервер и клиент;
- обработка запросов сервера и клиента — описывает протокол, который выполняет запросы на прием и передачу станз, данных

файлов, чтения содержимого и отправка содержимого файлов, объединяя в себе бинарный протокол;

- бинарный протокол — модуль описывает сам протокол через который будет производиться передача данных, константы идентификаторы пакетов, структуры передаваемых данных и методы с помощью которых будет передаваться информация в бинарном протоколе. А так же интерфейс событий приходящих от бинарного протокола;
- модуль общих функций сервера и клиента — содержат общие глобальные функции и типы данных, которые используются как сервером так и клиентом;
- базовый конфигурационный модуль — в нем описаны свойственные как для клиента так и для сервера данные из конфигурационного файла.

Модули клиента:

- основной модуль — содержит метод main;
- модуль обработки запросов FUSE — Содержит саму виртуальную файловую систему, в которой находятся файлы хранящиеся на серверах;
- модуль управления серверами — содержит список экземпляров серверов, в нем описаны сами сервера, содержаться экземпляры их протоколов, через которые к ним выполняются запросы;
- конфигурационный модуль клиента — в нем находится производный от базового конфигурационного модуля модуль, в нем описывается все свойственные для клиента данные из конфигурационного файла;
- интерфейс обработки запросов от сервера — содержит интерфейс в котором описаны методы вызываемые при тех или иных событиях происходящих на сервере и возврата от него команд.

Модули сервера:

- основной модуль — содержит метод main;
- модуль обработки запросов клиента — содержит методы выполняющиеся при происхождении событий на стороне клиента, запросов на чтение файлов и каталогов;
- модуль управления клиентами — содержит список экземпляров клиентов, в нем описаны сами клиенты, содержаться экземпляры их протоколов. От них сервер получает запросы на те или иные действия выполняет какие либо события связанные с реальной файловой системой;
- конфигурационный модуль сервера — в нем находится производный от базового конфигурационного модуля модуль, в нем описывается все свойственные для сервера данные из конфигурационного файла;
- интерфейс обработки запросов от клиента — содержит интерфейс в котором описаны методы вызываемые при тех или иных событиях происходящих на стороне клиента и возврата от него команд.

2.3 Конструирование протоколов обмена

2.3.1 Протокол управления файлами

Для построения протокола можно выбрать следующие конструкции языка XML, по аналогии с уже существующим и работающим в сфере IM систем протоколом JABBER[17].

Установка соединения

После установления соединения первым начинает общение клиент отправляя серверу версию поддерживаемого протокола XML, после чего открывается тег strem, внутри которого будут находится все передаваемые

клиентом запросы.

<?xml version="1.0"?>

<stream:stream xmlns:stream="http://etherx.jabber.org/streams" version="1.0" xmlns="mydfs:client" to="доменное имя сервера" xml:lang="en" xmlns:xml="http://www.w3.org/XML/1998/namespace" >

Затем, сервер возвращает свою версию и stream, и те возможности, которые он поддерживает, возможности планируется расширять с новыми версиям, к ним относятся метод аутентификации и поддержка сжатия и шифрования в файловой системе, которая пока не планируется.

<?xml version='1.0'?>

<stream:stream xmlns='mydfs:client' xmlns:stream='http://etherx.jabber.org/streams' id='3722602580' from='доменное имя сервера' version='1.0' xml:lang='ru'>

<stream:features>

</stream:features>

В случае появления ошибок в ходе подключения, обнаружившая ошибку сторона должна отправить станзу следующего формата:

<stream:error>

<xml-not-well-formed xmlns='urn:ietf:params:xml:ns:xmpp-streams'/>

</stream:error>

Например сервер обнаружил ошибку разбора.

Клиент:

<message xml:lang='en'>

<body>Ошибка XML не закрыт тег body!

</message>

Сервер:

<stream:error>

<xml-not-well-formed xmlns='urn:ietf:params:xml:ns:xmpp-streams'/>

</stream:error>

Сервер:

</stream:stream>

Аутентификация

После чего производится аутентификация клиента, если она требуется, и начинается обмен сообщениями между сервером и клиентом. Закрытие соединения производится закрытием тега stream затем противоположная сторона, должна отправить свой закрывающийся тег stream и разорвать соединение.

Клиент:

</stream:stream>

Сервер:

</stream:stream>

На данный момент тип аутентификации поддерживаемый сервером будет иметь только одно значение:

<mechanisms xmlns="urn:ietf:params:xml:ns:xmpp-sasl">

<mechanism>NONE</mechanism>

</mechanisms>

После чего клиент выбирает механизм аутентификациии отправлеет его серверу:

<auth xmlns="urn:ietf:params:xml:ns:xmpp-sasl" mechanism="NONE" />

При неудачном идентифицировании клиента возвращается станза следующего вида:

<failure xmlns="urn:ietf:params:xml:ns:xmpp-sasl">

<not-authorized/>

</failure>

Обмен данными между сервером и клиентом

После аутентификации и установки соединения начинается обмен данными между сервером и клиентом. К этим данным относятся все операции с файлами в файловой системе. Они могут повторятся и выполняться сколько угодно раз и идти в любой последовательности. Формат запросов имеет следующий вид:

<имя_тега id='уникальный идентификатор' from='ip адрес или отправителя' to='ip адрес получателя' type='тип запроса'/>

Тип запроса может принимать следующие значения:

- set при установке каких-либо параметров, например записи данных в файл или создания директории;
- get получении каких-либо данных у противолежащей стороны, например запрос списка файлов внутри директории;
- result при возврате ответов на set или get.

Запрос содержимого директории

Клиент запрашивает у сервера содержимое директории, для этого отправляется тег следующего содержания:

Запрос:

```
<iq id='идентификатор' xml:lang='ru-RU' type='get' to='ip сервера'>
    <readdir xmlns='readdir-data' dir="путь к директории"/>
</iq>
```

Ответ:

```
<iq from='ip сервера' to='ip клиента' type='result' id='идентификатор'>
    <readdir xmlns='readdir-data' dir="путь к директории">
        <file filename='имя файла 1'/>
        <file filename='имя файла 2'/>
```

<file filename='имя файла ..'/>

<file filename='имя файла N'/>

</readdir>

</iq>

Запрос атрибутов файлов

После запроса содержимого директории клиент запрашивает у сервера содержимое атрибутов и типов файлов, для этого отправляется тег следующего содержания.

Запрос:

<iq id='идентификатор' xml:lang='ru-RU' type='get' to='ip сервера'>

<getattr xmlns='getattr-data' dir="путь к файлу или папке"/>

</iq>

Ответ:

<iq from='ip сервера' to='ip клиента' type='result' id='идентификатор'>

< getattr xmlns='getattr-data'/>

<mode>XXXX</mode>

<nlink>XXXX</nlink>

</getattr>

</iq>

- mode — атрибуты файла
- nlink — число ссылок на файлы

Открытие файла

Запрос:

<iq id='идентификатор' xml:lang='ru-RU' type='get' to='ip сервера'>

<open xmlns='open-data' dir="путь к файлу или папке">

<mode>XXXX</mode>

<desc>XXXX</desc>

</open>

</iq>

- mode режим открытия файла принимает числовые значения например O_RDONLY;
- desc дискриптор файла.

Ответ:

<iq from='ip сервера' to='ip клиента' type='result' id='идентификатор'>

< read xmlns='open-data' result='XXXX'/>

</iq>

- result содержит в себе ответ об ошибке или успешном открытии. 0 успешное открытие.

Чтение файла

Запрос:

<iq id='идентификатор' xml:lang='ru-RU' type='get' to='ip сервера'>

<read xmlns='read-data' type='get' dir="путь к файлу или папке">

<offset>XXXXX</offset>

<size>XXXXX</size>

</read>

</iq>

Ответ:

<iq from='ip сервера' to='ip клиента' type='result' id='идентификатор'>

< read xmlns='read-data' type='result' result='XXXX'>

<patckets_id>XXXX</patckets_id>

<size>XXXXX</size>

</read>
</iq>

- result принимает значение 0 в случае успеха, иначе номер ошибки;
- offset — смещение с которого будет производиться чтение;
- size в запросе: размер сколько требуется прочитать, в ответе, сколько реально будет прочитано;
- packets_id принимает номер потока, через который по бинарному протоколу будут передаваться данные.

В случае успеха клиент отправляет сообщение подтверждающее отправку, если оно получено, то начинается чтение по бинарному протоколу, иначе по истечении тайм аута файл должен быть закрыт.

<iq from='ip сервера' to='ip клиента' type='result' id='идентификатор'>
< read xmlns='read-data' type='sublmit' id='идентификатор потока'/>
</iq>

Запись в файл

Запрос:

<iq id='идентификатор' xml:lang='ru-RU' type='get' to='ip сервера'>
<write xmlns='write-data' type='get' dir="путь к файлу или папке">
<offset>XXXXX</offset>
<size>XXXXX</size>
</write>
</iq>

Ответ:

<iq from='ip сервера' to='ip клиента' type='result' id='идентификатор'>
< write xmlns='write-data' type='result' result='XXXX'>
<patckets_id>XXXX</patckets_id>

</write>

</iq>

- result принимает значение 0 в случае успеха, иначе номер ошибки;
- offset — смещение с которого будет производиться чтение;
- size — размер сколько требуется записать;
- packets_id — номер потока через который по бинарному протоколу будут передаваться данные.

В случае успеха клиент отправляет сообщение подтверждающее отправку, если оно получено, то начинается чтение по бинарному протоколу, иначе по истечении тайм аута файл должен быть закрыт.

<iq from='ip сервера' to='ip клиента' type='result' id='идентификатор'>

< write xmlns='write-data' type='sublmit' id='идентификатор пакетов'/>

</iq>

Закрытие файла

<iq from='ip сервера' to='ip клиента' type='set' id='идентификатор'>

< release xmlns='release-data' dir='путь к файлу' desc='XXXX' >

</iq>

- desc дескриптор файла.

Переименование фала

Запрос:

<iq from='ip сервера' to='ip клиента' type='set' id='идентификатор'>

< rename xmlns='rename-data' dirfrom='путь к файлу на данный момент' dirto='новый путь к файлу' >

</iq>

Ответ:

<iq from='ip сервера' to='ip клиента' type='result' id='идентификатор'>

< rename xmlns='rename-data' result='ответ'>

</iq>

- result номер ошибки операции.

Удаление файла

Запрос:

<iq from='ip сервера' to='ip клиента' type='set' id='идентификатор'>

< unlink xmlns='unlink-data' dir='путь к файлу на данный момент' >

</iq>

Ответ:

<iq from='ip сервера' to='ip клиента' type='result' id='идентификатор'>

< unlink xmlns='unlink-data' result='ответ'>

</iq>

- result номер ошибки операции.

Удаление директории

Запрос:

<iq from='ip сервера' to='ip клиента' type='set' id='идентификатор'>

< rmdir xmlns='rmdir-data' dir='путь к файлу на данный момент' >

</iq>

Ответ:

<iq from='ip сервера' to='ip клиента' type='result' id='идентификатор'>

< rmdir xmlns='unlink-data' result='ответ'>

</iq>

- result номер ошибки операции.

2.3.2 Бинарный протокол, обмена содержимым файлов

Бинарный протокол используется для чтения и записи файлов. Он содержит заголовок и тело. В заголовке находятся информация о содержимом. В теле само содержимое файла. Нужно сделать учет того, что будут новые поля, таким образом имеем следующую структуру:

- ID заголовка 16Bit, беззнаковое целое;
- размер заголовка 16Bit, беззнаковое целое;
- номер заголовка в последовательности 32Bit, беззнаковое целое;
- размер тела 32Bit, беззнаковое целое;
- тело.

Данные файлов передаются блоками, заранее заданного размера.

2.4 Построение конфигурационного файла

В результате анализа полученной информации можно использовать следующую структуру конфигурационного файла.

2.4.1 Конфигурационный файл сервера

```
conf{
* сеция глобальных данных как сервера так и клиента
  global_data{
    extension_dir="<путь к папке с расширениями>";
    module_dir="<путь к папке с модулями>";
  }
* секция данных индивидуальных для сервера
 server_data{
   server_ip=<IP на котором ожидается подключение клиентов>;
   bibary_protocol_ip=<предпочтительный IP для бинарного протокола>;
```

```
    server_port=<Порт для подключения конечных клиентов>;
    shared_dir="<путь к папке с общедоступными данными>"
  }
* секция в которой будут храниться данные расширений и модулей
  module_data{
      <название модуля>{
         <название параметра>=значение параметра;
      }
  }
}
```

2.4.2 Конфигурационный файл клиента

```
conf{
* сеция глобальных данных как сервера так и клиента
   global_data{
      extension_dir="<путь к папке с расширениями>";
      module_dir="<путь к папке с модулями>";
  }
* секция данных индивидуальных для клиента
  client_data{
* секции серверов к которым подключается данный клиент.
    servers{
       server{
          server_ip=<"IP адрес сервера">;
          server_port=<"порт сервера">;
       }
    }
```

```
  }
* секция в которой будут храниться данные расширений и модулей
  module_data{
     <название модуля>{
        <название параметра>=значение параметра;
     }
  }
}
```

2.5 Вывод

В результате алгоритмического конструирования, решено выбрать в качестве базового протокола для обмена метаданными протокол XML и расширить его таким образом, чтобы можно было реализовать функции файловой системы. Разработаны принципы обмена данными между сервером и клиентом, протокол обмена метаданными и построены конструкции XML. Описан бинарный протокол для передачи содержимого файлов. Разработана структура создаваемой системы, и связи между программными модулями. Так же разработана структура конфигурационного файла для сервера и клиента.

3 ПРОГРАММНОЕ КОНСТРУИРОВАНИЕ ФАЙЛОВОЙ СИСТЕМЫ

3.1 Выбор XML парсера

Так как решено использование формата представления данных XML в качестве базового формата для обмена запросами операций между сервером и клиентом, то необходимо выбрать подходящий XML анализатор. Для этого рассмотрены следующие анализаторы:

3.1.1 LibXML2

Libxml2 это анализатор XML на языке C, разработанный для проекта Gnome (но используется вне платформы Gnome), это бесплатное программное обеспечение доступно в соответствии с лицензией MIT.

Libxml2 это очень легко переносимая библиотека, ее можно собрать и использовать без особого труда, на большом числе вариантов операционных систем. К ним относятся: Linux, Unix, Windows, CygWin, MacOS, MacOS X, RISC Os, OS/2, VMS, QNX, MVS, VxWorks, и.т.д. Библиотека используется в модуле Jabber клиента Pidgin[18].

3.1.2 QtXML

Модуль QtXml обеспечивает работу с потоками чтения и записи XML документов и реализацию их в форме SAX и DOM. Однако, требует наличие библиотеки QT для создания приложений использующих его.

3.1.3 Expat

Expat это библиотека для синтаксического анализа XML, написанные на языке C. Это потоко-ориентированный анализатор, в котором приложение регистрирует обработчики событий для элементов, которые анализатор может найти в XML документе (например начало тега). Данный анализатор

поддерживается во многих языках, таких как Erlang, Ocaml, Objective-C, Python, Simkin, Ruby и др. С применением данного анализатора написаны такие проекты как Ejabberd - jabber сервер, и xmpppy - библиотека протокола XMPP для языка Python[19].

3.1.4 Решение

Наиболее подходящим для данной задачи является анализатор Expat, он успешно применяется в таком крупном проекте как сервер Ejabberd компании ProcessOne, для анализа потокового протокола XMPP системы Jabber. Протокол разрабатываемой файловой системы схож по своей структуре с протоколом xmpp, и применение данного анализатора вполне оправдано.

3.2 Выбор библиотеки для построения конфигурационного файла

Для сохранения конфигурации программы требуется создать конфигурационные файлы. Для этой цели целесообразно применить библиотеку XFlib[20]. XF (eXchange Format) - это универсальный, легкий и переносимый формат представления данных в текстовом виде, который просто воспринимается человеком и обрабатывается программами. XF значительно более гибок и лаконичен, чем XML. Он полностью поддерживает стандарт Unicode и может применяться для создания приложений на различных языках.

3.3 Общие модули для клиента и сервера

3.3.1 Бинарный протокол обмена содержимым файлов BibaryProtocol.h

Типы данных

Заголовок пакета бинарного протокола.

```
#pragma pack(push,1)
struct BibaryPacketHeader{
```

```
	unsigned short pktID;
```

Идентификатор передаваемого пакета.

```
	unsigned short headSize;
```

Размер заголовка.

```
	unsigned int bodySize;
```

Размер тела.

```
};
#pragma pack(pop)
```

Структура описывающая бинарный пакет.

```
struct BibaryPacket{
	BibaryPacketHeader header;
	void * body;
};
```

Класс описывает сам протокол обмена между клиентом и сервером в оба направления. Его экземпляр создается для каждого подключенного к серверу клиента, и в каждом подключенном к серверу клиенту, он передается во все модули внутри сервера, после чего, модули по мере необходимости регистрируют в нем требующиеся функции на прием передачу данных файлов.

```
template <class Listener_T> class BinaryProtocol{
```

Параметр шаблона — класс внутри которого находится метод, который будет вызван при приеме пакета данных.

```
public:
	typedef void (Listener_T::*packetRecerver)(BibaryPacket);
```

Тип данных — указатель на функцию, которая будет вызываться при передаче данных.

```
	BinaryProtocol(){};
```

Метод подключения. Передается IP адрес и порт.

```
	void connect(unsigned int ip, unsigned short port){};
```

Производится подключение к противолежащей стороне, ip — адресс к которому подключаемся, port — порт через который производится соединение.

```
	void disconnect(){};
```

Разрыв соединения.

```
	void setRecervedPacketFunc(packetRecerver func,unsigned short pktID){};
```

Принимает указатель на метод который будет вызван при приеме данных, а так же идентификатор потока данных, который конкретно будет приниматься и отправляться к конкретной функции.

```
	void sendPacket(int size, void *data){};
```

Отправка пакета данных через бинарный протокол, size — размер пакета, data — указатель на передаваемые данные.

```
	virtual ~BinaryProtocol(){};
};
```

3.3.2 Чтение конфигурационного файла

Базовый модуль конфигурационного файла BaseConfigFile.h

Конфигурационный файл, построен с использованием библиотеки Xflib. Базовый класс, открывает файл и читает данные совпадающие с конфигурационным фалом сервера и клиента. Клиентский, и серверный класс наследуются от него, и читают свои части. Класс состоит из следующих функций:

```
class BaseConfigFile{
public:
	BaseConfigFile(){};
	BaseConfigFile(char* filename);
```

Конструктор, в качестве параметра принимает имя конфигурационного файла.

```
	char* getModulesDir();
```

Возвращает директорию с модулями.

```
	char* getExtendsDir();
```

Возвращает директорию с расширениями.

```
	xfNode* getModulesSect();
```

Возвращает указатель на ноду в конфигурационном файле, в которую модули могут сохранять свои данные.

```
unsigned int getHostIpByName(xfChar* address, int af);
```

Получение ip адреса сервера по доменному имени или строке с IP адресом, address — адрес, ip которого получаем, af — тип адреса.

```
unsigned int unicodeToIP(xfChar* cip);
```

Получение адреса по строке с адресом.

```
virtual ~BaseConfigFile();
```

Деструктор закрывает конфигурационный файл

```
};
```

3.4 Модули сервера

3.4.1 Модуль конфигурационного файла сервера ServerConfigFile.h

Этот модуль содержит возможности получения уникальных для сервера данных из конфигурационного файла.

```
class ServerConfig : public BaseConfigFile{
public:
    char* getSharedDir();
```

Директория в которой хранятся файлы доступные через данный сервер ФС

```
    unsigned int getServerPort();
```

Получение порта через который сервер будет ожидать подключения.

```
    list<int> *getServerIps();
```

Список IP адресов через которые сервер будет ожидать подключения клиентов.

```
    ServerConfig();
    virtual ~ServerConfig();
};
```

3.4.2 Модуль взаимодействия с модулями расширений и клиентами сервера ClientServerDataTransfer.h

Модуль обработки запросов между сервером и клиентом, он вызывает определенные модули для обработки каждой из пришедших станз, которые

будут выполнять действия над файлами, и возвращать клиенту ответы об операциях, а также выполнение процесса аутентификации авторизации сжатия и шифрования.

Форматы функций чтения отправки данных для расширений.

```
typedef void* funcGetData(int datacount);
typedef void funcSendData(void* data, int datacount);
```

Указатель такого типа обрабатывает вызовы всех станз. Модуль сам должен их заполнить в список и после прихода станзы вызывается каждый раз последовательно до тех пор, пока не будет возвращено значение true.

```
typedef bool moduleFunc(void* stanza, funcSendData *sendd,funcGetData *getd);
typedef list<moduleFunc*> moduleFuncListType;
```

Тип вызывается при нахождении файла при поиске модулей и расширений.

```
typedef void findedFileT(char * filename);
```

Функция расширений принимает данные, проводит их через расширение, передает дальше.

```
struct ExtendInfo{
	int number;
```

Порядковый номер расширения, в последовательности как они обрабатываются.

```
};
```

Структура передается в каждый поток где обрабатывается каждый подключеный клиент.

```
struct ThreadStruct{
	ClientServerDataTransfer* self;
```

Указатель на класс внутри которого передача данных.

```
	int clientSocket;
```

Сокет через который конкретный клиент будет совершать подключения.

```
	unsigned int clientIP;
```

IP адрес клиента с которым в данный момент соединены.

```
};
```

Непосредственно сам класс который будет производить цикл обработки.

```
class ClientServerDataTransfer{
```

```
protected:
```

Расширения которые идут поверх протокола.

```
map<String stanza,moduleFunc*> moduleFuncMap;
```

Название расширения, функция которая вызывается для передачи в него данных.

Список функций обрабатывающих станзы.

```
moduleFuncListType moduleFuncList;
ServerConfig conf;
```

Конфигурационный файл сервера, он будет передаваться во все расширения, чтобы они могли прочесть необходимые им данные из конфигурационного файла.

```
void findModules(){};
```

Поиск модулей.

```
void findExtends(){};
```

Поиск расширений.

Метод для поиска файлов и добавления их в списки модулей и в списки расширений.

```
void findFiles(char* path, findedFileT * file);
public:
```

Выполняет поиск модулей и регистрирует вызовы их для обработки станз, ожидает подключение клиентов. Модули передают в класс указатель на функцию, которая обрабатывает станзу, ей передается станза и два метода один для чтения данных другой для записи ответов, и она возвращает логическое значение принимает ли модуль данную станзу. В которую передается, если не принимает то производится поиск следующего модуля.

```
ClientServerDataTransfer(ServerConfig c);
void waiteClentConnections();
```

Метод, который ожидает подключения клиентов, и создает новый поток для каждого из них, выполняет последовательно чтение целой станзы и вызывает каждый модуль до тех пор, пока станза не будет отвергнута всеми, либо не

найдется такой модуль, который ее примет.

```
        void moduleProcess();
        virtual ~ClientServerDataTransfer();
};
```

3.5 Модули динамически подключаемых библиотек расширений сервера

Библиотека FileOperationModule выполняет базовые операции с файлами, такие как открытие, чтение, запись, просмотр содержимого каталога.

3.6 Выводы

В процессе программного конструирования был выбран в качестве XML анализатора Expat. Разработана структура программного кода сервера, а так-же интерфейсные части динамически подгружаемых модулей. Для построения конфигурационных файлов решено использовать библиотеку XFLib, как наиболее простой и легко читаемый и анализируемый формат представления данных.

ЗАКЛЮЧЕНИЕ

В данной работе были проанализированы аналоги разрабатываемой программы, найдены их уникальные особенности, построена структура программы, разработаны используемые алгоритмы. В следствии исследования предметной области выявлены следующие положительные особенности файловых систем, которые решено использовать в разрабатываемой системе, к ним относятся:

- архитектура файловой системы является клиент-серверной, без наличия сервера транзакций, и метаданных, как в GlusterFS в отличии от Lustre и HDFS это упростит ее функционирование, хотя может вызвать проблемы с сконфигурированным нескольких клиентов;
- файловая система кэширует файлы на стороне клиента, благодаря чему повышается надежность системы. Даже при сбое в сети она продолжает частично функционировать, что так-же упростит функции сохранения данных на другие компьютеры при включении или выходе из строя компьютера на который производится запись подобно AFS;
- идея трансляторов и модулей, позволит достичь более структурированный код, и даст возможность расширения файловой системы по мере необходимости без модификации самого ядра системы, таким образом как это сделано в GlusterFS;
- использовать минимально возможный размер заголовка в пакетах при передаче содержимого файлов, для того, чтобы достичь минимальные потери в скорости на заголовках пакетов, в отличии от GlusterFS;
- реализовать несколько алгоритмов выбора компьютера, на который производится запись как отдельные модули, реализовать обработку запросов от клиентов отдельными модулями, в том числе аутентификацию и файловые операции, подобно GlusterFS;

- в протоколе должны быть ping пакеты подобно системе HDFS, которые будут отправляться к клиентам от серверов, и если клиент не получает информацию, в течении некоторого промежутка времени, то сервер будет считаться «умершим» и удален из списка серверов внутри клиента;
- реализовать два отдельных независимых протокола для данных, которые предают содержимое файлов и для метаданных, как в Hadoop и HDFS;
- в качестве библиотеки для построения файловой системы выбрать FUSE.

Так-же в процессе алгоритмического конструирования разработаны два протокола обмена данными между клиентом и сервером, которые использованы в разрабатываемой программе, они имеют следующие особенности:

- решено выбрать в качестве базового протокола для обмена метаданными протокол XML и расширить его таким образом чтобы можно было реализовать функции файловой системы. А в качестве XML анализатора выбрать Expat;
- разработан бинарный протокол для передачи содержимого файлов.

Для конфигурации программы были проанализированы библиотеки, позволяющие создавать файлы конфигурации и принято решение использовать библиотеку XFLib, как наиболее простой и легко читаемый и анализируемый формат представления данных. Коме того была разработана структура конфигурационного файла для сервера и клиента

В результате проделанной работы подготовлена необходимая информация для создания файловой системы, разработаны алгоритмы и структура модулей программы.

4 СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Обзор сетевых файловыз систем nfs, lustre и gfs (nfs lustre gfs) [Электронный ресурс] -– URL: http://www.opennet.ru/base/net/network_fs_review.txt.html

2. Google File System [Электронный ресурс] -– URL: http://ru.wikipedia.org/wiki/Google_File_System

3. SSHFS [Электронный ресурс] -– URL: http://ru.wikipedia.org/wiki/SSHFS

4. GmailFS - делаем хранилище файлов из почты Gmail [Электронный ресурс] -– URL: http://easylinux.ru/node/288

5. Сайт WikipediaFS [Электронный ресурс] -– URL: http://wikipediafs.sourceforge.net/

6. A FTP filesystem based on cURL and FUSE [Электронный ресурс] -– URL: http://curlftpfs.sourceforge.net/

7. Сайт проекта LTFP [Электронный ресурс] -– URL:http://lftpfs.sourceforge.net/

8. Разработка собственной файловой системы с помощью FUSE [Электронный ресурс] -– URL: http://www.ibm.com/developerworks/ru/library/l-fuse/index.html

9. Сайт проекта OpenAFS [Электронный ресурс] -– URL: http://www.openafs.org/

10. LUSTRE [Электронный ресурс] -– URL: http://www.insight-it.ru/net/scalability/lustre/

11. GlusterFS [Электронный ресурс] -– URL: http://www.gluster.com

12. GlusterFS [Электронный ресурс] -– URL: http://www.insight-it.ru/net/scalability/glusterfs/

13. InfiniBand [Электронный ресурс] -– URL:

http://ru.wikipedia.org/wiki/InfiniBand

14. Hadoop [Электронный ресурс] -– URL: http://www.insight-it.ru/net/scalability/hadoop/

15. FUSE [Электронный ресурс] -–URL: http://ru.wikipedia.org/wiki/Filesystem_in_Userspace

16. FUSE [Электронный ресурс] -– URL: http://fuse.sourceforge.net

17. XMPP [Электронный ресурс] -– URL: http://xmpp.org

18. LibXML2 [Электронный ресурс] -– URL: http://xmlsoft.org

19. Expat [Электронный ресурс] -– URL: http://expat.sourceforge.net

20. XF [Электронный ресурс] -– URL: http://xfhome.org

5 ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ РАСПРЕДЕЛЕННОЙ ФАЙЛОВОЙ СИСТЕМЫ

5.1 Резюме

Предлагается разработка программного продукта (ПП) предназначенного для монтирования каталогов удаленных ЭВМ в компьютерной сети и использования файлов в них как локальные.

Использование данного ПП возможно в разных организациях, где необходимо организовать единое дисковое пространство между ЭВМ.

Целью намеченного бизнеса является получение прибыли за счет предложению рынку конкурентоспособного продукта.

Для разработки и создания ПП необходимы капитальные вложения составляют 121960 руб., капитальные вложения будут погашены через 6 месяцев.

Критический объем продаж программного продукта составляет 31 копию. Запас финансовой прочности, за этот же период, составляет 73 копии ПП, коэффициент финансовой прочности равен 70%.

Цель данного бизнес-плана – выработка стратегических решений путём рассмотрения данного бизнеса с позиции маркетингового синтеза.

5.2 Характеристика ПП

Программный продукт позволяет монтировать каталоги на удаленных ЭВМ создавая единое дисковое пространство для нескольких объединенных в сеть компьютеров. Каждый компьютер может записать в это дисковое пространство некоторые данные, и использовать их совместно с другими компьютерами в сети, как будто это локальные файлы.

Разрабатываемый программный продукт может применяться для эффективной организации дискового пространства вычислительных машин и

повышения надежности хранимых данных в сети, путем дублирования их на несколько ЭВМ. ПП должен реализовывать следующие функции:

- позволяет монтировать каталоги на удаленных ЭВМ как локальные создавая единое дисковое пространство для нескольких объединенных в сеть компьютеров;
- в случае ошибки нехватки дискового пространства, данные должны быть переписаны на другую машину, и произведена до-запись. Если подобных машин не найдено, тогда вызывать ошибку;
- в случае отключения компьютера в момент записи должен быть выбран новый компьютер и данные перезаписаны на него, таким образом программа на стороне клиента должна кэшировать данные и производить их запись на выбранный компьютер параллельно;
- программа должна функционировать в операционной системе UNIX и ее разновидностях;
- разрабатываемое приложение должно состоять из двух программ: клиента который подключается к набору серверов и выполняет операции над файлами, и сервера, который реализует файловую систему. Клиент подключается к серверу и производит операции над файлами. Любой клиент может подключаться к одному или более серверов одновременно, любые сервера могут быть подключены к нескольким клиентам;
- файловая система должна поддерживать расширяемые модули для сервера и клиента, благодаря которым будут осуществляться операции с файлами, выбор компьютеров для записи фалов, аутентификация и прочие действия;

Охарактеризуем рассматриваемый проект с позиции маркетинга по параметрам замысел, реальное исполнение, область применения, преимущества

у пользователя:

- Замысел. Организация единого дискового пространства;
- Реальное исполнение. Распределенная файловая система
- Область применения. ПП может применяться в различных вычислительных центрах, ;
- Преимущества у пользователя. Использование стандартного интерфейса ФС для работы с данными на удаленных ЭВМ, повышенная надежность целостности данных, распределение сетевого трафика при обращении к данным на удаленных ЭВМ позволяющее уменьшить нагрузку на ЛВС, использовать данные совместно с другими пользователями, пользоваться своими данными на других ЭВМ в ЛВС.

(Если это не преимущества то что же преимущество? Конечный пользователь не видит разницы, но системный администратор может применяя распределенную ФС повысить надежность, производительность, предоставить использовать одни данные находящиеся на разных ЭВМ пользователям. Программы, же которыми пользователь производит операции над ФС «подмены» жесткого диска на виртуальный, не замечают, как сервер так и клиент это программы демоны, которые работают в фоновом режиме и их не видит пользователь, интерфейса у программы нет как такового, только текстовый файл где прописаны настройки)

Получает возможность используя стандартный интерфейс для работы с файлами, работать с данными находящимися на удаленных ЭВМ.

- Конкуренты. Andrew File System, Lustre, Gluster FS, HDFS.

5.3 Исследование и анализ рынка

В качестве основных потребителей рассматриваются учебные заведения Ростова-на-Дону. Процесс сегментирования представлен в ниже приведенной

таблице.

Таблица №__ «Ориентировочная сегментация потенциальных потребителей»

Сегменты рынка	Планируемый объем продаж по годам					
	2011г	2012г		2013г		Всего
	2 полугодие	1 полугодие	2 полугодие	1 полугодие	2 полугодие	
Интернет провайдеры	5	7	9	11	12	44
Хостинг компании	3	4	6	7	10	30
Коммерческие организации	2	3	6	8	11	30
Итого	10	14	21	26	33	104

5.4 Производственный план

5.4.1 Расчёт единовременных затрат

В структуре единовременных затрат ($З_к$) выделяют капитальные затраты (К), включающие затраты на приобретение или дооборудование вычислительной техники (Квт), приобретении пакета прикладных программ (Кппп) и операционных систем (Кос).

Зк = Квт+Кос+Кппп.

В состав затрат на приобретение ВТ, ЛВС, ППП, ОС включаются транспортные расходы и затраты на установку. Единовременные затраты приведены в таблице №__.

№	Наименование тех. средства	Тип или модель	Кол-во	Поставщик	Стоимость	Сумма

	или ПО		(шт.)		одного изделия	(руб.)
1	Системный блок	Granda I3 2100	1	Solwin	12990	12990
2	Монитор	SAMSUNG B1940R BMB [TFT]	1	Solwin	5990	5990
3	Клавиатура	Genius KB110, Black [PS/2]	1	Solwin	203	203
4	Мышь	Genius Navigator 535 Laser 1600dpi [USB]	1	Solwin	690	690
5	Операционная система	Debian Squeeze GNU/Linux	1	knoppix.ru	359	359
6	Прикладное ПО	Eclipse IDE	1	The Eclipse Foundation	0	0
7	Прикладное ПО	GNU Compiler Collection	1	Free Software Foundation, Inc.	0	0
Итого (руб.):						18683

Зк=18683

5.4.2 Расчёт текущих затрат на разработку ПП

Для расчета текущих затрат определим состав персонала, участвующего в разработке программного продукта. Расчет зарплаты персонала приведен в таблице №__

Категория персонала	Количество сотрудников	Оплата за 1 час (руб.)	Время, необходимое для разработки программного продукта (час)	Общая зарплата сотрудников данной категории (руб.)
Инженер-программист	1	110	288	31680
Итого (руб.)				31680

Оплата труда производится на основе повременной оплаты, исходя из часовой ставки ($C_ч$) и времени на разработку ПП ($T_{пр}$). В этом случае заработная плата рассчитывается по формуле:

$$З_{пр} = \sum (C_ч) * T_{пр}$$

Потребность в капиталовложениях на разработку проекта представлена в таблице №__.

Наименование статей затрат	Формула для расчета	Сумма
1.Единовременные затраты (Зк)	$З_К = К + З_{ПК}$	20232
1.1. Затраты на приобретение ВТ (Квт)	$К_{ВТ}$	19873
1.2. Затраты на приобретениепакетов прикладных программ и операционных систем (Кппп)	$К_{ППП}$	359
2. Текущие затраты (С)	$С = З_{ПР} + Н_З + З_Н$	104860
2.1. Затраты на заработную плату (Зпр)	$З_{ПР} = \sum (С_ч) * Т_{ПР}$	31680
2.2. Страховые взносы ($Н_з = 26\%\ от\ З_р$)	$Н_З = 26\ от З_{ПР}$	9821
2.3. Накладные расходы (Зн)	$З_Н = 200\ от З_{ПР}$	63360
Итого затрат (З)	$З = К + С$	125093

5.4.3 Определение цены ПП

Рассчитав текущие затраты на разработку ПП (С), можно определить себестоимость одной копии ПП по формуле:

$$C_1 = \frac{C}{N} + З_{тир} + З_{сер}$$

где N- объем продаж в натуральном выражении;

$З_{тир}$ – затраты на тиражирование в расчёте на одну копию;

$З_{сер}$ – затраты на сервисное обслуживание в расчете на одну копию.

$$З_{тир} = 650\ (руб.)$$

$$З_{сер} = 300\ (руб.)$$

$$C_1 = \frac{104860}{104} + 650 + 300 = 1958\ (руб.)$$

Определяем оптовую цену одной копии по следующей формуле:

$$Ц_{о1} = С_1 + П_1,$$

где: C_1 – себестоимость одной копии, руб.;

$П_1$ – прибыль на одну копию, руб.;

Определяем прибыль:

$$П_1 = C_1 * Р/100,$$

где Р – процент предполагаемой рентабельности (Р=20%);

$$П_1 = \frac{1958 \cdot 20}{100} = 392\ (руб.)$$

$$Ц_{01} = 1943 + 397 = 2340 (руб.)$$

Цена продажи программного продукта:

$$Ц_{пр} = Ц_{о1} + НДС,$$

где НДС – налог на добавленную стоимость в соответствии действующей ставкой на данный вид продукции (18%),

$$Ц_{пр} = 2340 + 421 = 2761 (руб.)$$

5.5 План маркетинговых действий

План маркетинга - это план мероприятий по достижению намечаемого объема продаж и получению максимальной прибыли путем удовлетворения рыночных потребностей. В данном разделе отражаются: товарная, ценовая, сбытовая политика и сервисное обслуживание.

Товарная политика предполагает постоянную модификацию ПП, учитывая требования пользователей, бесплатные обновления, сервисное обслуживание в течение всего жизненного цикла ПП, а также on-line консультации.

Ценовая политика предполагает установление цены, ориентируясь на цены мирового рынка, а также ее изменение в зависимости от области предпринимательской деятельности.

Сбытовая политика предполагает:

- создание и регулирование коммерческих связей через посредников, дилеров, агентов и пр;
- организация и участие в ярмарках, выставках;
- использование кредита в различных формах, продажа в рассрочку, лизинг;
- презентацию продукции специально для потенциальных потребителей.

Сервисное обслуживание предполагает предпродажный и послепродажный сервис.

Предпродажный сервис ориентирован на постоянное изучение и анализ требований потребителей с целью совершенствования качественных параметров предлагаемой продукции.

Послепродажный сервис предусматривает комплекс работ по обслуживанию (комплекс работ по установке ПП и обучению пользователя).

5.6 Потенциальные риски

В рыночных условиях этот раздел особенно важен, и от глубины его проработки в значительной степени зависит доверие потенциальных инвесторов, кредиторов и партнеров по бизнесу.

Следует учитывать следующие виды рисков: производственные, коммерческие, финансовые и риски, связанные с форс–мажорными обстоятельствами.

Производственные риски связаны с различными нарушениями в производственном процессе:

- выход из строя комплектующих ПК;
- потеря информации из-за выхода из строя носителей информации;
- потеря информации из-за аварийного отключения электропитания;
- обрыв линии передачи информации между сервером и пользователями рабочих станций.

Производственные риски можно снизить с помощью дублирования информации, копирования информации на разные носители, создание архивов программных версий, использования средств защиты.

Коммерческие риски связаны с реализацией продукции на товарном рынке:

- уменьшение размеров и емкости рынков;
- снижение платежеспособного спроса;
- появление новых конкурентов;
- отсутствие рекламы;
- отсутствие консультационного центра.

Коммерческие риски можно снизить, если создать отдел маркетинга для рекламирования ПП, а также создание консультационного центра для

поддержки пользователей.

Финансовые риски вызываются:

- инфляционными и девальвационными процессами;
- отплатой по безналичному расчету.

Финансовые риски можно снизить, если предусмотрены условия по реализации: объем реализации по безналичному расчету должен быть не менее 10000 руб.

Риски, связанные с форс-мажорными обстоятельствами - чрезвычайное событие, которое невозможно было предвидеть и предотвратить:

- стихийные бедствия.

Мерой по предотвращению рисков, связанных с форс-мажорными обстоятельствами, является работа предприятия с достаточным запасом финансовой прочности.

Для снижения общего влияния рисков предусмотрено коммерческое страхование: страхование имущества, транспортных перевозок, перестрахование и т.д.

5.7 Финансовый план

Финансовый план является заключительным разделом бизнес-плана и содержит обоснование экономической эффективности затрат, произведенных в связи с разработкой и реализацией ПП.

Предполагаемые доходы от продаж (Qпр) определяются по формуле:

$$Qпр = Цпр*N; \quad (5.7)$$

где Цпр – цена продажи ПП, руб.; N - объем продаж по периодам в соответствии с исследованиями рынка, шт.

Издержки производства (И) включают, кроме текущих затрат, расходы на тиражирование, сервисное обслуживание, маркетинг, рекламу и некоторые

виды налогов (на имущество, местные налоги и т.п.):

И=С1*N+Зм+Н; (5.8)

где С1 - себестоимость копии ПП, Зм - затраты на маркетинг и коммерческие расходы (Зм = от 5 до 20% от Qпр соответствующего периода); Н - налоги. Налог на имущество НИ определяется в процентах от стоимости ВТ и ЛВС по действующей ставке (Ни = 1,1% от стоимости ВТ и ЛВС). Расчеты производятся по годам и сводятся в таблице №__:

Таблица №__ - Финансовый план

Показатели	2011г	2012г		2013г	
	2 полугодие	1 полугодие	2 полугодие	1 полугодие	2 полугодие
1 Доходы от продаж – (Qпр)	27610	38654	57981	71786	91113
2 Издержки производства (И)	26672,9	36653,7	54960,1	67636,1	84062,5
3 Прибыль от реализации	937,1	2000,3	3020,9	4149,9	7050,5
4. Налог на прибыль 20%	187,42	400	604,18	829,98	1410,1
5. Чистая прибыль	749,68	1600,3	3048,4	4102	5577,04
Справочно: Планируемый объем продаж ПП (шт.)	10	14	21	26	33

В данном случае разработчик принимает решение осуществить разработку ПП за счет собственных средств без привлечения коммерческого кредита, поэтому определим срок окупаемости ПП, который определяется сроком окупаемости дополнительных капитальных вложений по формуле:

$$T_{ок} = \frac{K}{П_1 * N};$$

где К – необходимые капитальные вложения, руб.; $П_1$ – прогнозная чистая прибыль от реализации одного ПП, руб.; N- прогнозный годовой объём продаж ПП, шт.

$$Т_{ок} = \frac{18683}{104 * 389} = 0{,}46 \approx 0{,}5 \text{ год} = 6 \text{ месяцев}$$

5.8 Расчёт безубыточности

Под безубыточностью в разработанном бизнес-плане понимается объем продаж ПП в натуральном выражении, при котором возможно покрытие всех расходов без получения прибыли.

Расчет достижения безубыточности производится по следующей формуле:

$$Q_{кр} = \frac{ФЗ}{Ц_{пр} - ПЗ_1},$$ где:

- $Q_{кр}$ – критический объем продаж программного продукта;
- ФЗ – сумма условно-постянных (фиксированных) затрат;
- $ПЗ_1$ – сумма условно переменных затрат для одного программного продукта;
- $Ц_{пп}$ – цена продажи одного программного продкта.

$ФЗ = З_н + З_м + Н$

ФЗ = 63360+7730,8+1720,93 = 72811,73 руб.

$ПЗ = З_{пр1} + Н_{з1}$

ПЗ = 31680+9672=41352 руб.

$$ПЗ_1 = \frac{41352}{104} = 397{,}6\,руб.$$

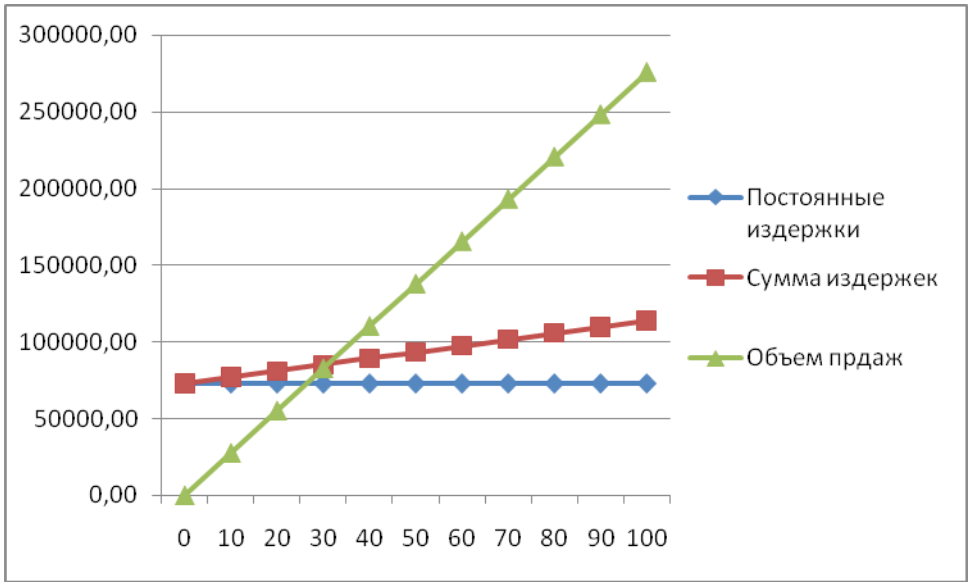
$$Q_{кр} = ФЗ/(Ц_{пр} - ПЗ_1)$$

$Q_{кр}$ =72811,73/$(2761 - 397{,}6) = 30{,}8 \approx 31 шт.$

Расчет постоянных и переменных издержек приведен в таблице №__

Кол-во (шт.)	Постоянные издержки (руб.)	Переменные издержки (руб.)	Сумма издержек (руб.)	Доходы от продаж (руб.)
0	72811,73	0	72811,73	0
10	72811,73	4135,2	76946,93	27610
20	72811,73	8270,4	81082,13	55220
30	72811,73	12405,6	85217,33	82830
40	72811,73	16540,8	89352,53	110440
50	72811,73	20676	93487,73	138050
60	72811,73	24811,2	97622,93	165660
70	72811,73	28946,4	101758,13	193270
80	72811,73	33081,6	105893,33	220880
90	72811,73	37216,8	110028,53	248490
100	72811,73	41352	114163,73	276100

После расчета критического объема продаж строится график безубыточности. (Рисунок 5.1):



Влево от точки безубыточности лежит область убытков, вправо – область прибыли.

Запас финансовой прочности (по расчёту за первый год производства) определяется по следующей формуле:

$$ЗФП = Q_{пр} - Q_{кр} = 104 - 31 = 73$$

Коэффициент запаса финансовой прочности $К_{зпф}$ определяется

отношением величины запаса финансовой прочности к объему продаж. Он характеризует степень финансовой устойчивости, рекомендуемая нижняя граница 30% к объему продаж.

$$K_{ЗФП} = \frac{ЗФП}{Q_{пр}} \cdot 100 = \frac{73}{104} \cdot 100 \approx 70$$

Таким образом, можно констатировать, что разработка, при данном объеме продаж, имеет достаточный запас финансовой прочности и является финансово устойчивой.

6 БЕЗОПАСНОСТЬ И ЭКОЛОГИЧНОСТЬ ПРОЕКТА

6.1 Введение

Работа с программой «Распределённая файловая система» осуществляется людьми, имеющими профессию связанную использованием ПК: пользователи, программисты. Следовательно, необходимо рассмотреть рабочее место программиста.

Особенности организации рабочего места этих профессий осуществляется в соответствии с их видом деятельности и по стандартизации предприятия.

Главными элементами рабочего места программиста являются письменный стол и кресло. Основным рабочим положением является положение сидя. Рабочее место для выполнения работ в положении сидя организуется в соответствии с ГОСТ 12.2.032-78.

Рабочая поза сидя вызывает минимальное утомление программиста. Рациональная планировка рабочего места предусматривает четкий порядок и постоянство размещения предметов, средств труда и документации. То, что требуется для выполнения работ чаще, расположено в зоне легкой досягаемости рабочего пространства.

Моторное поле - пространство рабочего места, в котором могут осуществляться двигательные действия человека.

Максимальная зона досягаемости рук - это часть моторного поля рабочего места, ограниченного дугами, описываемыми максимально вытянутыми руками при движении их в плечевом суставе.

Оптимальная зона - часть моторного поля рабочего места, ограниченного дугами, описываемыми предплечьями при движении в локтевых суставах с опорой в точке локтя и с относительно неподвижным плечом.

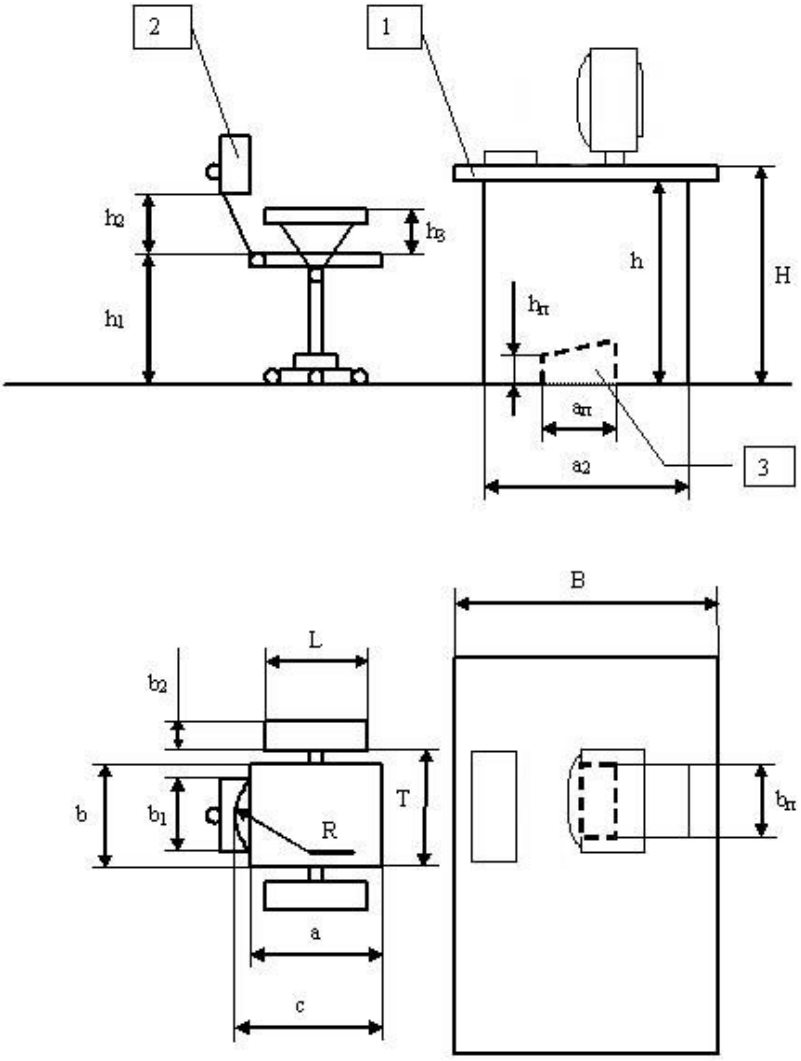


Рисунок 5 – Схема рабочего места

Рассмотрим оптимальное размещение предметов труда и документации в зонах досягаемости рук:

- Дисплей размещается в зоне а (в центре);
- Клавиатура - в зоне г/д;
- Системный блок размещается в зоне б (слева);
- Принтер находится в зоне а (справа);
- Документация располагается в зоне легкой досягаемости ладони - в (слева) - литература и документация, необходимая при работе или в

выдвижных ящиках стола - литература, неиспользуемая постоянно.

При проектировании письменного стола следует учитывать следующее:

- высота стола должна быть выбрана с учетом возможности сидеть свободно, в удобной позе, при необходимости опираясь на подлокотники;
- нижняя часть стола должна быть сконструирована так, чтобы программист мог удобно сидеть, не был вынужден поджимать ноги;
- поверхность стола должна обладать свойствами, исключающими появление бликов в поле зрения программиста;
- конструкция стола должна предусматривать наличие выдвижных ящиков (не менее 3 для хранения документации, листингов, канцелярских принадлежностей, личных вещей).

Параметры рабочего места выбираются в соответствии с антропометрическими характеристиками. При работе в положении сидя рекомендуются следующие параметры рабочего пространства:

- ширина не менее 700 мм;
- глубина не менее 400 мм;
- высота рабочей поверхности стола над полом 700-750 мм.

Оптимальными размерами стола являются:

- высота 710 мм;
- длина стола 1300 мм;
- ширина стола 650 мм.

Поверхность для письма должна иметь не менее 40 мм в глубину и не менее 600 мм в ширину. Под рабочей поверхностью должно быть предусмотрено пространство для ног:

- высота не менее 600 мм;
- ширина не менее 500 мм;

- глубина не менее 400 мм.

Важным элементом рабочего места программиста является кресло. Оно выполняется в соответствии с ГОСТ 21.889-76. При проектировании кресла исходят из того, что при любом рабочем положении программиста его поза должна быть физиологически правильно обоснованной, т.е. положение частей тела должно быть оптимальным. Для удовлетворения требований физиологии, вытекающих из анализа положения тела человека, в положении сидя, конструкция рабочего сидения должна удовлетворять следующим основным требованиям:

- допускать возможность изменения положения тела, т.е. обеспечивать свободное перемещение корпуса и конечностей тела друг относительно друга;
- допускать регулирование высоты в зависимости от роста работающего человека (в пределах от 400 до 550 мм);
- иметь слегка вогнутую поверхность;
- иметь небольшой наклон назад.

6.2 Анализ вредных и опасных производственных факторов

Вредный производственный фактор - фактор среды и трудового процесса, который может вызвать снижение работоспособности, патологию (профессиональное заболевание), привести к нарушению здоровья потомства.

Вредными могут быть:

физические факторы: температура, влажность и подвижность воздуха, неионизирующие и ионизирующие излучения, шум, вибрация, недостаточная освещенность;

- химические факторы: загазованность и запыленность воздуха;
- биологические факторы: болезнетворные микроорганизмы;

- факторы тяжести труда: физическая статическая и динамическая нагрузка; большое количество стереотипных рабочих движений, большое число наклонов корпуса, неудобная рабочая поза;
- факторы напряженности труда: интеллектуальные, сенсорные, эмоциональные нагрузки, монотонность и продолжительность работы.

Опасный производственный фактор - фактор среды и трудового процесса, который может вызвать резкое ухудшение здоровья, травму, смерть. Это: электрический ток, огонь, нагретая поверхность, движущиеся части оборудования, избыточное давление, острые кромки предметов, высота и.т.п.).

Основными факторами вредного и опасного влияния компьютера и другой офисной техники на организм человека являются:

- ионизирующее излучение;
- электростатическое поле;
- электромагнитное поле;
- высокий уровень шума;
- видимое излучение экрана;
- плохой микроклимат помещения;
- неправильное освещение рабочего места, блики и мерцания;
- нарушение эргономических норм при работе с компьютером.

6.3 Расчётная часть необходимого заземления и вентиляции.

//==========

6.3.1 Расчет защитного заземления

Цель расчета: определение основных параметров заземляющего устройства: числа, размеров и порядка размещения одиночных заземлителей и заземляющих проводников, при которых напряжение прикосновения и шага в

период замыкания фаз на заземленный корпус не превышает допустимых значений.

//==========

Цель расчета: определить число и размер вертикальных заземлителей (стержней), длину горизонтальных элементов и выполнить схему защитного заземления.

//==========

Исходные данные:

- Линейное напряжение сети $U_л = 220\,В$;
- Заземляющее устройство состоит из вертикальных стержней длинной $l = 2000\,мм$ и диаметром $d = 10\,мм$;
- Стержни размещаются в ряд $P = 14\,м$;
- Общая длина подключенных воздушных линий $l_В = ¿$;
- Общая длина подключенных к сети кабельных линий $l_К = ¿$;
- Приближенное значение удельного сопротивления грунта (смешанный грунт) $\rho_{изм} = 40\,Ом \cdot м$ ρ изм = 40 Ом · м (см. таблицу 2);
- Расстояние между стержнями а, м (при этом а/l = 1).

Расчетный ток замыкания со стороны подстанции

$$I_з = \frac{U_л \cdot (35 \cdot l_к + l_в)}{350}$$

Сопротивление заземляющего устройства нейтрали трансформатора на стороне 220В по данным таблицы 1 Rз ≤ 4Ом. В дальнейших расчётах Rз принимаем равным 4 Ом.

По формуле определяем расчёт удельного сопротивление грунта

$$\rho_\rho = \rho_{изм} \cdot \psi$$

Сопротивление одиночного вертикального стержневого заземлителя, заглублённого ниже уровня земли на t0 = 0,7м определяется по формуле:

$$R_{овс}=\frac{\rho_{\rho}}{2\pi\cdot l}\left(\ln\frac{2l}{d}+\frac{1}{2}\cdot\ln\frac{4t+l}{4t-l}\right)$$

Приближённое количество заземлителей.

$$n\approx\frac{R_{овс}}{R_{з}}$$

Коэффициент использования заземлителей.

$$a=\frac{P}{n_{приб}}$$

Предварительное количество заземлителей.

$$n_{з}=\frac{R_{овс}}{\eta R_{з}}$$

Сопротивление соединительной полосы (без учёта коэффициента использования полосы)соединяющей одиночные вертикальные стержни.

$$R_{пол}=\frac{\rho_{\rho}}{2\pi\cdot l_{1}}\cdot\ln\frac{2l_{1}^{2}}{bh_{0}}\qquad l_{1}=a\cdot(n_{з}-1)$$

Сопротивление соединительной полосы (с учётом коэффициента использования полосы):

$$R_{пол}^{'}=\frac{R_{пол}}{\eta_{ип}}$$

$\eta_{ип}=$¿

Для 4-ёх заземлителей расстояние между стержнями:

$$a=\frac{P}{n}\qquad R_{овс}^{пол}=\frac{R_{пол}^{'}\cdot R_{з}}{R_{пол}^{'}-R_{з}}$$

Уточняем количество заземлителей с учётом коэффициента:

$$n_{з}^{'}=\frac{Rовс}{R_{пол}^{овс}\cdot\eta_{из}}$$

Схема заземлителя.

==========

6.4 Действия персонала при пожароопасной ситуации

Инструкция (типовая), определяющая действия обслуживающего персонала по обеспечению безопасной эвакуации людей, (к плану эвакуации)

№ п/п	Наименование действий	Порядок и последовательность действий	Ф.И.О., должность исполнителя
1	Сообщение о пожаре	При обнаружении пожара необходимо немедленно вызвать пожарную помощь по тел. 01	Сотрудник, первый обнаруживший пожар
2	Извещение о пожаре	Дать условный сигнал, включить систему оповещения. Открыть основные и запасные выходы.	Вахтер
3	Эвакуация людей из здания	Вывести людей организовано через коридоры, лестничные клетки, выходы немедленно после получения сообщения о пожаре.	Сотрудники хорошо знающие помещение.
4	Тушение возникшего пожара до прибытия пожарной помощи	Тушение пожара организуется немедленно с момента его обнаружения при помощи пожарных кранов, огнетушителей, а также подручных средств, в том числе водой.	Персонал, не занятый эвакуацией людей (поименно)
5	Встретить прибывшие пожарные подразделе	Встретить прибывшие пожарные машины, доложить руководителю тушения пожара об обстановке в здании, что и где горит, есть ли опасность людям, о нахождении	Директор, завхоз

	ния	водоисточников на территории, вручить поэтажные планы эвакуации.	

Таблица 5 – Порядок эвакуации при пожаре

Примечания:

- Пути следования детей во время эвакуации не должны пересекаться;
- В зимнее время следует предусмотреть организацию пункта размещения эвакуированных детей (школах - интернатах и в ночное время);
- Отработку плана эвакуации с действиями обслуживающего персонала при возникновении пожара осуществляют сразу же после его составления и затем периодически, не реже 2 раза в год. Занятия должны быть практическими по отработке каждых конкретных действий;
- С планом эвакуации и распределением обязанностей должен быть ознакомлен весь обслуживающий персонал под роспис

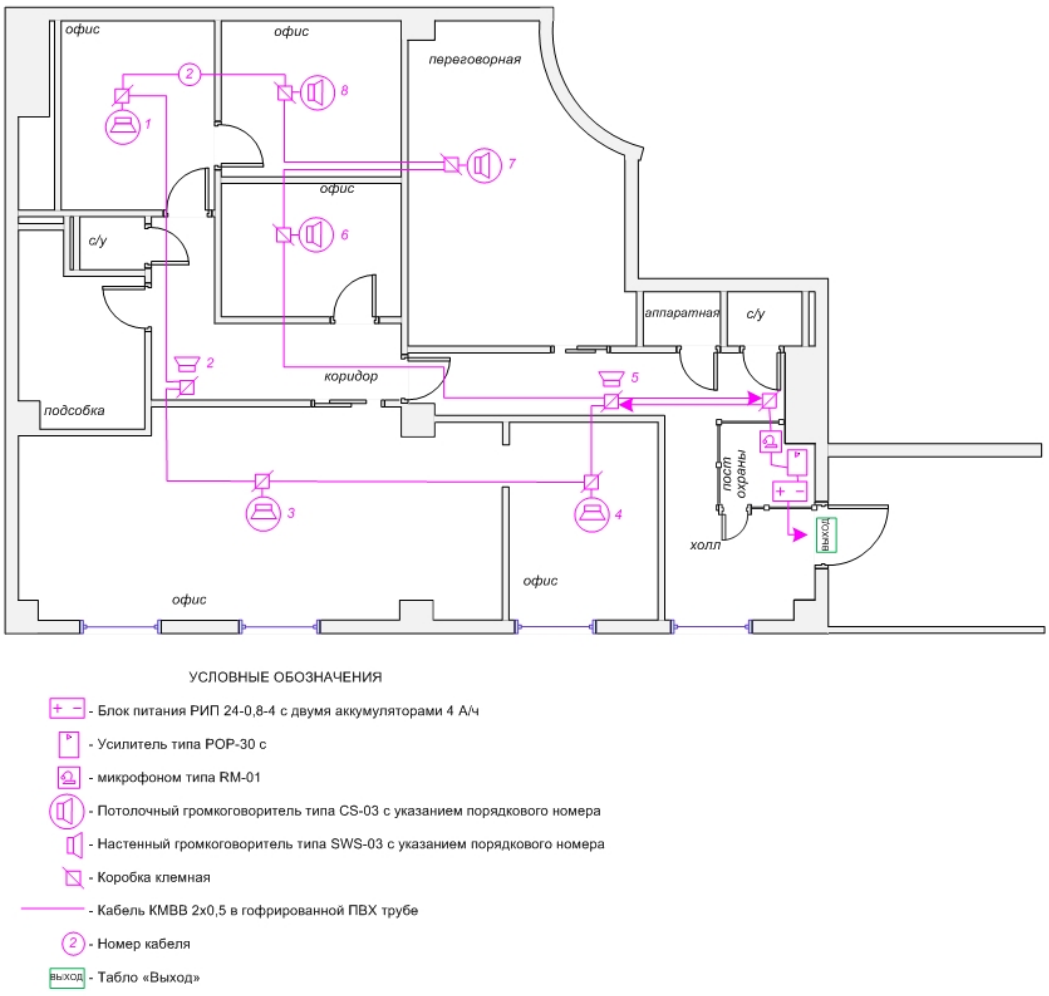


Рисунок 6 – Схема пожарной эвакуации

ПРИЛОЖЕНИЕ А ТЕХНИЧЕСКОЕ ЗАДАНИЕ

«СОГЛАСОВАНО»	«УТВЕРЖДЕНО»
Руководитель дип. проекта:	зав. кафедрой «ПОВТ и АС»
Клубков И.М. _____	Нейдорф Р.А. _____
« ___ » _____ 2011 г.	« ___ » _____ 2011 г.

П.А.1 Наименование

Наименование программного средства — «Распределенная файловая система».

П.А.2 Область применения

Разрабатываемое программное средство может применяться в сфере информационных технологий.

П.А.3 Основание для разработки

Разработка ведется на основании документа «Учебный план для студентов ВУЗа», факультета «Информатика и вычислительная техника» (ИиВТ) направления 230000 «Информатика и вычислительная техника» специальность 230105 «Программное обеспечение вычислительной техники и автоматизированных систем» Донского Государственного Технического Университета (ДГТУ), утвержденного министерством общего и профессионального образования 22.09.2002г., в соответствии с которым студенты, заканчивающие ВУЗ, должны предоставить к защите выпускную квалификационную работу, подтверждающую присвоение им квалификации «Инженер». Предметным основанием является задание на дипломную работу.

П.А.4 Назначение разработки

П.А.4.1 Функциональное назначение

Функциональное назначение разрабатываемой программы заключается в создании общего дискового пространства для группы компьютеров, соединенных в сеть, каждый из которых имеет доступ к этому пространству.

П.А.4.2 Эксплуатационное назначение

ПО предназначено для эксплуатации на серверах и/или рабочих станциях, с целью образования единого дискового пространства.

П.А.5 Технические требования к программе или программному изделию

П.А.5.1 Требования к функциональным характеристикам

Программа должна организовывать единое дисковое пространство для групп серверов, которые позволят сохранять в него данные пользователей. Дисковое пространство должно поддерживать:

- создание файлов;
- запись данных в файлы;
- чтение файлов;
- изменение и чтение прав доступа на файлы;
- просмотр и создание директорий.

П.А.5.2 Требования к надежности

Надежность программного средства должна обеспечиваться:

- обработкой фатальных ошибок;
- обработкой всех исключительных ситуаций в процессе работы программы;

- обработкой ошибок связанных с недостаточностью дискового пространства при записи на диск, и производить выбор нового компьютера;
- обработка ошибок связанных с записью данных на диск в момент выключения компьютера на который она производится;
- производить кэширование данных на стороне клиента для повышения надежности;
- обработка ошибок связанных с разрывом сетевого соединения в момент сеанса работы.

П.А.5.3 Условия эксплуатации.

Для функционирования программы необходимо выполнять условия эксплуатации ЭВМ и носителей информации.

Климатические условия эксплуатации: условия, при которых должны обеспечиваться заданные характеристики, должны удовлетворять требованиям, предъявляемым к техническим средствам в части условий их эксплуатации.

Требования к видам обслуживания: программа не требует проведения каких-либо видов обслуживания.

Требования к квалификации персонала: для конфигурирования и начальной установки программного продукта необходим специалист с квалификацией “инженер-программист”.

Для эксплуатации данной программы достаточно одного человека, получившего квалификацию оператора ПК или более высокую квалификацию.

П.А.5.4 Требования к составу и параметрам технических средств.

Для функционирования программы необходимо выполнить системные требования:

- два или более ПК с UNIX совместимыми архитекторами (alpha, amd64, arm, armel, hppa, i386, ia64, mips, mipsel, powerpc, s390, sparc);
- объем ОЗУ не менее 32 МБ в зависимости от платформы;
- свободное место на жестком диске не менее 10 МБ для программы, а также некоторое количество места для кэша и распределенных данных;
- наличие сетевого интерфейса.

П.А.5.5 Требования к информационной и программной совместимости

Требования к операционной системе, наличию библиотек и языкам программирования:

- UNIX совместимая ОС (Linux с версией ядра 2.6, либо либо NetBSD, FreeBSD, OpenSolaris);
- наличие Expat — динамической библиотеки времени выполнения expat, библиотеки для работы с XML;
- наличие динамической библиотеки XFLib — анализатора формата XF;
- наличие библиотеки FUSE для клиентской части — интерфейс файловой системы в пространстве пользователя;
- компилятор C++.

П.А.5.6 Требования к программной документации

Документация должна включать:

- техническое задание (ГОСТ 19.201);
- руководство системного программиста (ГОСТ 19.503);
- руководство программиста (ГОСТ 19.504);
- руководство оператора (ГОСТ 19.505).

П.А.6 Технико-экономические показатели

К программе не предъявляется экономических требований.

П.А.7 Стадии и этапы разработки

Стадии и этапы разработки программного средства:

- сформулирована постановка задачи (с 01.02.2011 по 05.02.2011);
- изучение предметной области (с 06.02.1011 по 10.02.2011);
- разработка технического задания (с 11.02.11 по 14.02.11);
- согласование и утверждения технического задания (с 14.02.2011 по 15.02.2011);
- разработка методов решения поставленной задачи (с 15.02.2011 по 18.02.2011);
- алгоритмическое конструирование (с 15.02.2011 по 20.02.2011);
- разработка интерфейса программы (с 17.02.2011 по 18.02.2011);
- написание текста программных модулей (с 21.02.2011 по 07.03.2011);
- отладка модулей программы (с 06.03.2011 по 12.03.2011);
- разработка и испытание контрольных тестовых примеров (с 23.02.2011 по 5.03.2011);
- разработка руководств системного программиста, программиста, оператора (с 25.02.2011 по 7.03.2011);
- разработка пояснительной записки к данному программному продукту (с 22.02.2011 по 07.03.2011).

П.А.8 Порядок контроля и приемки

Порядок и контроль приёмки определяются заведующим кафедрой «ПОВТ и АС» и основан на демонстрации знаний технологии и умении создавать

программные средства для различных предметных областей. Главным требованием к приемке является наличие правильно работающего программного модуля, иллюстрируемого тестовым примером, и отчета, представленного в печатном виде.

Разработал

студент гр. ВИ-51 «___» _____ 2011 год

Лубягов Н.А. _____