Санкт-Петербургский государственный университет

Кафедра системного программирования

Морозов Сергей Валерьевич

Многоярусная файловая система: оптимизация архитектуры и алгоритмов управления

Курсовая работа

Научный руководитель:
д. ф.-м. н., профессор Нестеров В. М.

Санкт-Петербург
2016

Оглавление

Введение

Большинство современных компаний ежедневно генерируют большой объем информации. Возникает необходимость сохранить эту информацию, чтобы в будущем иметь возможность ее получить и обработать. Проблема хранения может быть решена разными способами. Выбор того или иного подхода к хранению данных зависит от многих причин, в том числе и от финансовых возможностей организации. С конца прошлого десятилетия некоторыми IT компаниями осуществляется разработка решений, позволяющих осуществить переход организаций на *третью платформу* [15], одной из составляющих которой является использование облачных сервисов. Облачные хранилища данных – один из видов облачных технологий. Облачные хранилища данных можно разделить на публичные, частные, общественные и гибридные [8]. Выбор вида облачной системы часто обусловлен финансовыми возможностями организации, требованиями к производительности и регуляторными нормами.

Для корпоративного сегмента гибридные облачные системы хранения очень важны, так как позволяют организациям хранить менее важные данные в дешевом публичном облачном хранилище, используя возможности масштабирования и экономии средств, предоставляемые провайдером публичной облачной системы. Данные, важные для деятельности организации, остаются при этом в частном облачном хранилище, которое предоставляет повышенные меры безопасности [8].

Современные пользователи ожидают от облачного хранилища практически бесконечного запаса свободного места. Количество объектов в облачных объектных системах хранения уже достигает триллионов. Многим пользователям (приложениям) необходим POSIX интерфейс для работы с привычными абстракциями – файлами и папками. Появляется необходимость организовать объектов в иерархические структуры на пространствах имен в облачном объектном хранилище. Объем метаданных пространств имен с иерархической структурой пропорционален объему хранимых в облачном объектном хранилище данных.

Скрытие интерфейсов облачной системы хранения POSIX абстракциями задача сложная, имеющая малый потенциал использования в реальных системах из-за сильных задержек, вызванных архитектурной неприспособленностью облачных систем к POSIX стандарту и географической удаленностью облачных систем от пользователей. Пользователи и приложения, пользующиеся POSIX интерфейсами, ожидают более быстрого отклика.

Поддержка POSIX интерфейсов в кластерных файловых системах также является непростой задачей (все оборудование расположено в локальной вычислительной сети). Тем не менее есть достаточно много файловых систем, как комерческих, так и некоммерческих, с открытым исходным кодом, успешно с ней справляющихся. Такие файловые системы предоставляют либо полную, либо близкую к POSIX семантику работы с файлами, папками и другими привычными пользователю сущностями. Кластерные файловые системы скрывают распределенную топологию размещения ресурсов хранения, создавая иллюзию, что пользователь работает на отдельно взятом сервере. К сожалению, количество свободного места для хранения данных и метаданных ограничено совокупным объемом ресурсов хранения кластера.

На основе накопленных статистических данных можно утверждать, что шаблон обращения к данным подчиняется распределению Парето [7]. К 20% хранимых данных обращаются часто («горячие» данные), к остальным 80% обращения происходят значительно реже («холодные» данные). Доступ к «горячим» данным должен происходить очень быстро. Время получения «холодных» данных может быть достаточно большим.

Пользователю хочется иметь много свободного места, владельцу оборудования хочется минимизировать расходы на установку и поддержку устройств хранения информации. Хорошим компромисом между пользователями и владельцем оборудования может стать файловая система, объединяющая устройства хранения владельца с облачной системой хранения, обеспечивающая пользователей POSIX семантикой и позволяющая хранить практически неограниченный объем информа-

ции за счет интеллектуальной выгрузки «холодных» данных в облачное хранилище и предиктивного перевода данных из разряда «холодных» в разряд «горячих» с помощью загрузки данных из облачного хранилища на локальное оборудование. Файловые системы, осуществляющие синхронизацию данных между устройствами хранения различных классов будем называть *многоярусными* файловыми системами.

Многоярусные файловые системы, осуществляющие синхронизацию данных между различными классами дисков (твердотельными накопителями, жесткими магнитными дисками, ленточными накопителями и прочими), существуют, причем выбор достаточно велик, особенно в корпоративном сегменте. Многоярусных файловых систем с ярусом в виде кластера в локальной вычислительной сети и ярусом в виде облачного объектного хранилища единицы. Автору известно одно коммерческое решение – EMC Cloud Array[1] и одно open-source решение – EMC 2 TIERS[2] на основе параллельной файловой системы OrangeFS.

Целью научной работы в области многоярусных файловых систем является создание и оптимизация существующих интеллектуальных алгоритмов синхронизации данных между ярусами, в особенности файловых систем, где ярусами являются ресурсы локальной вычислительной сети и облачное объектное хранилище.

Целью текущей курсовой работы является разработка архитектуры многоярусной файловой системы вышеупомянутого типа, а также создание прототипа, демонстрирующего работоспособность концепции. Поставлены следующие задачи:

- сделать обзор существующих кластерных файловых систем;
- выбрать существующую файловую систему, подходящую для внедрения многоярусности;
- определить архитектуру многоярусной файловой системы;
- создать прототип, демонстрирующий возможость синхронизации данных между кластером в локальной вычислительной сети и облачным объектным хранилищем.

[1] http://russia.emc.com/storage/cloudarray/index.htm

[2] http://www.orangefs.org/

1. Определения

Файловая система. Иерархическая структура на файлах. Обеспечивает простой доступ к файлам, размещенным на жестком диске, разделе диска или логическом разделе. Файловая система также осуществляет контроль доступа к файлам пользователей [8].

Распределенная (кластерная) файловая система. Файловая система, предоставляющая доступ к постоянному хранилищу, в котором множество объектов существует с момента их явного создания до момента их явного удаления, и устойчивая к отказам, как к программным, так и к аппаратным. Постоянное хранилище охватывает несколько федераций ресурсов хранения, в которых клиенты могут создавать, удалять, читать и писать файлы. В отличие от локальных файловых систем, ресурсы хранения и клиенты файловой системы распределены по сети. Распределенная файловая система должна удовлетворять свойствам [5]:

- *прозрачности*: пользователи имеют доступ к системе вне зависимости от места, где они выполнили вход в систему, способны выполнять операции в распределенной файловой системе как в локальной, а также не должны заботиться о возможных отказах оборудования;
- *отказоустойчивости*: система не должна прекращать работу в случае отказа части оборудования;
- *расширяемости*: способности эффективно использовать много серверов, которые могут динамически добавляться в систему.

Параллельная файловая система. Тип распределенных файловых систем. Предоставляется доступ к разделяемому пространству имен. Параллельная файловая система изначально ориентирована на параллелизм, конкурентный (часто скоординированный) доступ многих клиентов. Такие системы работают через высокоскоростные сети и имеют сильно оптимизированные потоки ввода-вывода, чтобы

обеспечивать максимальную пропускную способность. В параллельных файловых системах происходит распределение «кусочков» файла по нескольким узлам. Предполагается, что такие файловые системы устанавливаются на корпоративные системы хранения данных [14].

Ярусность. Установление иерархии на системах хранения, основывающейся на требованиях к предоставляемому сервису, таких как производительность, непрерывность бизнеса, безопасность, стоимость и другое. [16].

2. Обзор

Строить свою распределенную многоярусную файловую систему с нуля неразумно. В этом разделе будут приведены примеры распределенных файловых систем, которые рассматривались, как основа для добавления многоярусности.

MooseFS[3]. Отказоустойчивая сетевая распределенная файловая система, предоставляющая пользователям общее пространство имен. Данные распределяются по нескольким физическим серверам, которые для пользователя представляются одним ресурсом. При работе со стандартными операциями с файлами, MooseFS ведет себя также, как и UNIX-подобные файловые системы. Некоторые свойства MooseFS представлены ниже:

- иерархическая структура директорий;
- хранение атрибутов POSIX-файлов;
- поддержка символьных и блоковых устройств, трубопроводов и сокетов;
- поддержка символьных и жестких ссылок;
- отказоустойчивость;
- масштабируемость (динамическое расширение объемов хранения за счет присоединения новых компьтеров или дисков).

Файловая система имеет открытый исходный код и распространяется под лицензией GPL.

CephFS[4]. Распределенная фаловая система с открытым исходным кодом, распространяемая под лицензией LGPL. CephFS динамически управляет распределенными метаданными с помощью *кластера метаданных (MDS)* и хранит данные и метаданные в *объектном хранилище (OSD)*.

CephFS имеет несколько интерфейсов взаимодействия с пользовате-

[3] **http://www.moosefs.org**
[4] **http://ceph.com**

лями. Благодаря своей архитектуре, файловая система предоставляет семантику близкую к POSIX [2]. Поддерживается специальный механизм блокировок. Уровень согласованности данных может быть понижен выставлением специльного флага C_LAZY, который разрешает чтение файла в момент его перезаписи [5].

Файловая система CephFS обладает высокой отказоустойчивостью, а также поддерживает многоярусность. Выделяют два яруса – cache tier (для горячих данных) и storage tier (для холодных данных) [1].

GlusterFS[5]. Распределенная файловая система с открытым исходным кодом, выпускаемая под лицензией GPL. Имеет клиент-серверную архитектуру, в которой нет отдельного сервера для хранения метаданных. Данные и метаданные хранятся на нескольких устройствах, присоединенных к разным серверам. GlusterFS полностью поддерживает семантику POSIX [9].

Местонахождение файла определяется с помощью алгоритма *EHA (Elastic Hashing Algorithm)* [3], поэтому от сущности сервера метаданных в этой файловой системе отказались.

В случае отказа одного из серверов, на котором установлена GlusterFS, он автоматически удаляется из системы и операции ввода-вывода на этом узле блокируются [5]. Поддерживает несколько ярусов [10].

OrangeFS[6]. Параллельная файловая система, являющаяся ответвлением параллельной файловой системы PVFS[7]. Предоставляет очень быстрый доступ к данным для параллельных приложений. OrangeFS отличается от своего родителя наличием функциональности, которой нет в PVFS. PVFS предназначена для очень больших кластеров. Следующими характеристиками обладает как OrangeFS, так и PVFS: производительность, надежность, независимость от аппаратной платформы, быстрое развертывание.

[5] **https://www.gluster.org**
[6] **http://www.orangefs.org**
[7] http://www.pvfs.org

OrangeFS также обладает некоторыми достоинствами, которых нет в PVFS.

- *Операции над метаданными.* Используется взаимодействие сервер-сервер, чтобы повысить производительность выполнения операций над метаданными. В OrangeFS реализован механизм распределенных директорий [6]. Также в одном из подпроектов OrangeFS реализуется улучшенный механизм поиска по метаданным.
- *Избыточность данных.* В OrangeFS реализован конфигурируемый механизм избыточности данных, а также механизм восстановления после сбоев. Имеется возможность задавать свой уровень избыточности отдельному файлу.
- *Контроль доступа.* В OrangeFS реализована поддержка контроля доступа пользователей к файловой системе. Эта одно из основных направлений, над которыми велась работа в последних релизах.

OrangeFS поддерживает семантику, близкую к POSIX. OrangeFS является проектом с открытым исходным кодом и распространяется под лицензией LGPL.

MarFS[8]. Распределенная файловая система. Реализует с помощью одной или более POSIX файловых систем расширяемый компонент метаданных. Также реализует с помощью одного или более хранилища данных расширяемый компонент данных. В качестве хранилища данных может использоваться, например, EMC ECS[9].

MarFS поддерживает работу с несколькими ярусами, одним из которых может являться облачное объектное хранилище данных.

К сожалению, MarFS обладает существенными для разрабатываемой многоярусной системы хранения ограничениями. Авторами MarFS утверждается, что файловая система поддерживает семантику, близкую к POSIX. Тем не менее в документации указаны ограничения на POSIX-семантику более жесткие, чем в других рассматриваемых в об-

[8] **https://github.com/mar-file-system/marfs**

[9] http://www.emc.com/storage/ecs/index.htm

зоре файловых системах. К примеру, MarFS не производит никаких проверок в случае, когда несколько клиентов пытаются писать в один файл, также MarFS не предоставляет возможность обновления файла, находящегося в облачном хранилище.

MarFS имеет открытый исходный код и распространяется под лицензией BSD.

Сравнение файловых систем. В таблице 1 приведено сравнение рассматриваемых в обзоре файловых систем.

Таблица 1: Сравнение файловых систем. Вопросы в некоторых клетках означают отсутствие явного указания на наличие выбранной характеристики в документации к файловой системе. В большинстве систем семантика POSIX полностью не соблюдается, но в некоторых системах POSIX-семантика ослаблена сильнее, чем в остальных. Такие системы отмечены +-. Системы, послностью поддерживающие POSIX-семантику, отмечены ++.

	POSIX-совмест.	Ярусность	Отказоуст.	Лицензия
MooseFS	+	-	+	GPL
CephFS	+	+	+	LGPL
GlusterFS	++	+	+	GPL
OrangeFS	+	-(planned for 3.0 rel.)	+	LGPL
MarFS	+-	+	?	BSD

Выбор файловой системы для внедрения многоярусности. В качестве базовой системы для внедрения многоярусности была выбрана OrangeFS. Эта параллельная файловая система сейчас является стабильной, готовой к использованию на «боевых» серверах. Ядерный модуль для OrangeFS был включен в недавно вышедшее ядро Linux 4.6 [11]. К разработке файловой системы присоединилось несколько крупный компаний[10], занимающихся системами хранения данных. OrangeFS

[10]http://www.orangefs.org/community/

обладает хорошей производительностью при работе с файлами от 1 мегабайта. На более маленьких файлах производительность тоже неплохая. OrangeFS выбрана, как базовая система для внедрения мгногоярусности в проекте EMC 2 Tiers[11]. В [4] представлено развернутое сравнение нескольких упомянутых в обзоре файловых систем. Показано, что OrangeFS их превосходит при определенном сценарии использования.

[11]EMC 2 Tiers предназначена только для высокопроизводительных вычислений.

3. Архитектура многоярусной файловой системы

Шаблон использования проектируемой файловой системы – разделяемая между пользователями директория. Пользователями могут быть как люди, так и приложения.

В дальнейшем будем придерживаться следующей терминологии:

- *Локальный ярус.* Ярус, состоящий из ресурсов хранения локальной вычислительной сети.
- *Расширенный ярус.* Ярус, являющийся облачным объектным хранилищем.

3.1. Ожидания пользователей

Работая с абстракциями многоярусной файловой системы, пользователь ожидает такого же поведения, как и при работе с локальной файловой системой. Существует вероятность одновременного доступа на запись к файлу нескольких пользователей и система должна корректно обрабатывать такие ситуации, поддерживать блокировки.

При работе с данными на локальном ярусе задержка получения данных будет небольшой, возможно незаметной для пользователя, так как OrangeFS является сильно оптимизированной параллельной файловой системой. При работе с данными на расширенном ярусе могут быть задержки при первом открытии/чтении/модификации файла, если алгоритм предвыборки данных не предугадал скорого обращения к данным этого файла.

И, конечно, пользователь ожидает увидеть почти неограниченный объем свободного места.

3.2. Ожидания системных администраторов

Системному администратору нужна возможность конфигурирования файловой системы под конкретные шаблоны использования. У мно-

гоярусной файловой системы должно быть как минимум два режима работы: выгрузка «холодных» данных с локального яруса на расширенный с последующим стиранием этих данных с локального яруса и режим резервного копирования, когда после записи на расширенный ярус стирания данных с локального не происходит. Режим резервного копирования полезен в случаях, когда неожиданно безвозвратно выходят из строя все устройства хранения в локальной сети организации и восстановление хоть какой-то версии файлов может съэкономить миллионы долларов.

Режим работы, при котором происходит выгрузка «холодных» данных с локального яруса на расширенный с последующим стиранием этих данных с локального яруса, тоже должен быть настраеваемым. Необходимо иметь возможность выбирать алгоритмы синхронизации данных между уровнями от самого простого – выгрузка данных при долгом неиспользовании и загрузка по требованию, до самых сложных алгоритмов, предугадывающих какие файлы подгрузить на локальный ярус еще до обращения к ним. Ниже приведены примеры характеристик, которые могут влиять на решения определенных алгоритмов о выгрузке/загрузке файла:

- *Тип пользователя.* Пользователь с идентификатором *user1* чаще обращается к файлам и поддиректориям директории *dir1*. Пользователь с идентификатором *user2* чаще обращается к файлам и поддиректориям директории *dir2*. *dir1* и *dir2* являются поддиректориями директории *dirRoot*. Тогда при обращении пользователя *user1* к файлам директории *dirRoot* разумно заранее подгрузить содержимое директории *dir1* и наоборот. Директорию, к которой у пользователя нет доступа ни на чтение, ни на запись, вообще нужно исключить из рассмотрения.
- *Приоретизация пользователей.* Для пользователя с ролью ”бухгалтер” следует в первую очередь подкачивать на локальный ярус файлы из директории *accounting*, а файлы из директории *hr* подкачивать с низким приоритетом.
- И другие...

3.3. Компоненты файловой системы и их взаимодействие

В этом разделе будет дано высокоуровневое описание компонентов системы и определены механизмы их взаимодействия.

Новые компоненты многоярусной файловой системы должны быть максимально независимы от существующих компонентов файловой системы OrangeFS, чтобы минимизировать усилия по их интеграции в файловую систему и отладке. Предлагается добавить к OrangeFS два дополнительных модуля:

- ***tier-engine*** – модуль, отвечающий за перенос данных с яруса на ярус. Он ответает за поддержание связи с облачным хранилищем, за сканирование файловой системы (поиск файлов-кандидатов на перенос на расширенный ярус), а также предоставляет API для других модулей OrangeFS для проверки местонахождения файла и, при необходимости, его переноса с расширенного на локальный ярус.
- ***rule-engine*** – модуль, отвечает за определение политик синхронизации файлов между ярусами, располагает набором интеллектуальных алгоритмов для принятия решения о выгрузке/загрузке того или иного файла. В нем реализован парсер нового предметно-ориентированого языка[12] (DSL), предназначенного для описания работы механизмов синхронизации. Предоставляет API для компонента tier-engine, позволяющего ему узнать дальнейшие действия с файлом.

На сегодняшний день существует несколько языков для описания политик и правил в системах хранения данных, но синтаксис этих языков сложен как для восприятия, так и для написания самих правил. Примерами таких систем являются Robinhood Policy Engine[13] и iRODS Rule Engine[14].

[12]Создание DSL языка – возможное продолжение научной работы.

[13]https://github.com/cea-hpc/robinhood/wiki

[14]http://wiki.irods.org/index.php/Rule_Engine, https://docs.irods.org/master/manual/rule_language

OrangeFS имеет несколько значимых преимуществ по сравнению с другими кластерными файловыми системами: данные и метаданные равномерно распределяются по всем узлам; объем метаданных на файл меньше, чем у других систем.

OrangeFS поддерживает POSIX семантику на уровне двух интерфейсов: ядерного модуля и интерфейса прямого доступа [13]. Ядерный модуль позволяет использовать файловую систему OrangeFS в родном для Linux виде, как монтируемую файловую систему, где можно использовать стандартные Linux-утилиты: *ls*, *cp*, *rm* и другие. Интерфейс прямого доступа позволяет работать с файловой системой OrangeFS в обход ядра Linux. Интерфейс обладает похожей на POSIX семантикой.

У OrangeFS есть два основных компонента - сервер и клиент. Данные на уровне сервера хранятся во внутреннем формате OrangeFS. Файл разбивается на несколько объектов, и объекты раскладываются по нескольким серверам. Без глубокой интеграции с внутренним недокументированным протоколом OrangeFS невозможно определить какой объект к какому файлу относится. Экземпляры серверов OrangeFS работают в пространстве пользователя.

Для работы клиенткой части OrangeFS необходимо иметь загруженным модуль ядра Linux, называемый *orangefs*. Когда пользователь отдает POSIX команду, вызов уходит в ядерный модуль, откуда транслируется обратно в пространство пользователя в экземпляр клиента OrangeFS, где обрабатывается. Удобнее всего сделать интеграцию компонента tier-engine с клиентской реализацией OrgangeFS, добавив вызовы функций из API tier-engine в реализацию POSIX функций клиента OrangeFS.

Метаданные о файлах, выгруженных на расширенный ярус можно хранить различными способами. Первым вариантом является создание сервера метаданных, хранящего информацию о файлах, выгруженных в облачное объектное хранилище. Вторым, более предпочтительным вариантом, является хранение новых метаданных в расширенных атрибутах каждого выгруженного файл. На месте выгруженного файла следует оставлять файл-пустышку с тем же набором атрибутов. Это

позволит задействовать собственные сервера метаданных OrangeFS.

На рисунке 1 представлена вышеописанная архитектура многоярусной файловой системы.

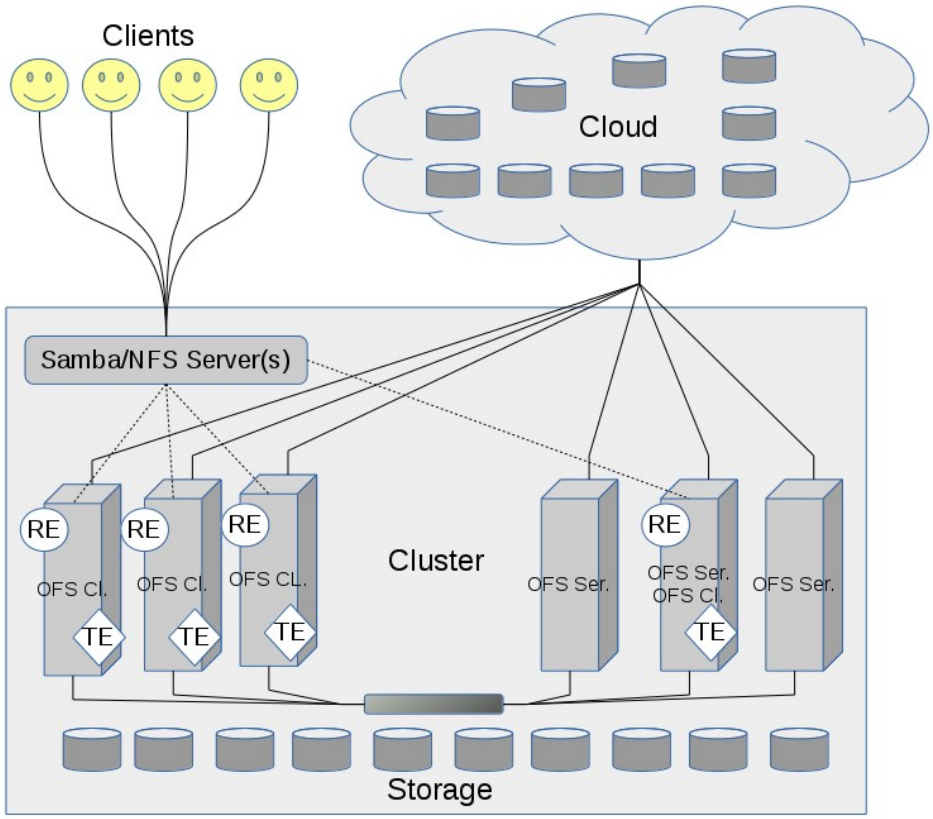

Рис. 1: Архитектура многоярусной файловой системы. Распределенная файловая система установлена на оборудовании организации (где-то только клиент OFS, где-то только сервер OFS, где-то обе сущности). На оборудовании с клиентами OFS установлены также компоненты tier-engine и rule-engine. Пользователи работают с Samba/NFS директориями, которые привязаны к директориям многоярусной файловой системы. «Горячие данные» хранятся в локальном хранилище (Storage) на локальром ярусе, а «холодные» в облачном объектном хранилище (Cloud) на расширенном ярусе.

При программировании предлагаемой архитерктуры стоит обращать много внимания на согласованность данных и, вероятно, поддерживать транзакции.

4. Реализация

На текущий момент создан прототип многоярусной файловой системы с двумя ярусами – локальным и расширенным. Он реализует лишь некоторые идеи, описанные в разделе 3. За основу была взята параллельная файловая система OrangeFS. Разработан отдельный компонент, называемый *tier-engine*. Он отвечает за синхронизацию между ярусами. Компонент не связан с реализацией OrangeFS и может быть использован для внедрения многоярусности в другие файловые системы. Компонент tier-engine предоставляет API для интеграции со сторонними файловыми системами (набор функций). Чаще всего вызов этих функций необходимо добавить в реализацию определенных POSIX функций, выбор POSIX функций зависит от реализации конкретной файловой системы. На текущий момент прототип поддерживает не все POSIX операции, а лишь некоторое их подмножество. При использовании POSIX операций, в которые была внедрена поддержка многоярусности, прототип работает стабильно.

4.1. Описание компонента tier-engine

tier-engine – это компонент, отвечающий за синхронизацию данных между локальным и расширенным ярусами. tier-engine выполняет две основные функции: перенос данных, к которым не было обращений достаточно долгое время, на расширенный ярус и перенос данных с расширенного яруса на локальный при обращении.

tier-engine запускается в отдельном фоновом потоке и выполняет обход дерева файловой системы в глубину, добавляя в очередь регулярные файлы, удовлетворяющие условиям переноса на расширенный ярус. Другой фоновый поток ждет появления файлов в очереди и начинает перенос данных на расширенный ярус.

После того, как данные регулярного файла успешно перенесены на расширенный ярус, выполняется еще одна проверка на соответствие условиям переноса: проверяется время последнего обращения к файлу. Если по прежнему обращений к файлу не было долгое время, файл

заменяется на файл-пустышку с таким же набором атрибутов, как у исходного файла, а также с несколькими расширенными атрибутами [12], с помощью которых устанавливается индикатор нахождения файла на расширенном ярусе и указываются метаданные, необходимые для переноса содержимого файла с расширенного яруса на локальный. Перенос данных с локального яруса на расширенный осуществляется при вызове функций из API tier-engine.

4.2. API компонента tier-engine

API tier-engine состоит всего из двух функций:

- int load_file_data(const char *path);
- int load_file_meta(const char *path, void *meta);

Функция "load_file_data" должна вызываться для любого файла при вызове определенных POSIX функций, которые могут открывать файл впервые (fopen, fread и другие). Если файл находится на локальном ярусе, сразу же происходит выход из функции, никаких действий не выполняется. Если данные файла находятся на расширенном ярусе, то происходит перенос данных на локальный.

Функция "load_file_meta" должна вызываться при получении атрибутов файла, в особенности его размера. В структуре "struct stat" заменяется значение размера файла на значение реального размера, записанного ранее в расширенные атрибуты.

Заключение

В рамках курсовой работы были выполнены следующие задачи:

- сделан обзор существующих кластерных файловых систем;
- выбрана файловая система, подходящая для внедрения многоярусности – OrangeFS;
- определа архитектура многоярусной файловой системы;
- создан прототип, демонстрирующий возможность синхронизации данных между кластером в локальной вычислительной сети и облачным объектным хранилищем.

Продолжением этой работы будет расширение поддержки POSIX операций в прототипе, а затем создание и оптимизация существующих интеллектуальных алгоритмов синхронизации данных между ярусами для случая, когда ярусами являются ресурсы локальной вычислительной сети и облачное объектное хранилище. Одним из возможных продолжений работы также является создание языка описания алгоритмов синхронизации данных между ярусами.

Список литературы

[1] Ceph Storage. Cache tiering. — URL: http://docs.ceph.com/docs/master/rados/operations/cache-tiering/ (online; accessed: 22.12.2015).

[2] CephFS. Differences from POSIX. — URL: http://docs.ceph.com/docs/hammer/dev/differences-from-posix/ (online; accessed: 22.12.2015).

[3] Cloud Storage for Modern Data Center. An Introduction to Gluster Architecture. — Version 3.1.x.

[4] Cluster filesystem for HPC. — URL: https://forums.gentoo.org/viewtopic.php?p=7006816#6996204 (online; accessed: 26.05.2016).

[5] Depardon Benjamin, Le Mahec Gaël, Séguin Cyril. Analysis of Six Distributed File Systems. — 2013.

[6] Distributed Directories. — URL: https://www-01.ibm.com/support/knowledgecenter/ssw_ibm_i_61/rzahy/rzahydistdir.htm (online; accessed: 22.12.2015).

[7] Dror Feitelson. Workload Modeling for Computer Systems Performance Evaluation. — 2011.

[8] EMC Education Services. Information Storage and Management: Storing, Managing and Protecting Digital Information in Classic, Virtualized and Cloud Environments / Ed. by Somasundaram Gnanasundaram, Alok Shrivastava. — Second edition. — John Wiley & Sons, Inc., 2012. — P. 528.

[9] GlusterFS General FAQ. — URL: http://www.gluster.org/community/documentation/index.php/GlusterFS_General_FAQ (online; accessed: 22.12.2015).

[10] GlusterFS. Tier. — URL: http://gluster.readthedocs.org/en/release-3.7.0/Features/tier/ (online; accessed: 22.12.2015).

[11] Linux 4.6-rc1. — URL: http://lkml.iu.edu/hypermail/linux/kernel/1603.3/01187.html (online; accessed: 26.05.2016).

[12] Linux Programmer's Manual. — URL: http://man7.org/linux/man-pages/man7/xattr.7.html (online; accessed: 26.05.2016).

[13] OrangeFS Installation Instructions. — Version 2.9.

[14] Parallel File Systems. — URL: http://www.cs.iit.edu/~iraicu/teaching/CS554-F13/lecture17-pfs-sam-lang.pdf (online; accessed: 22.12.2015).

[15] 'Third platform' shift triggers enterprise software evolution. — URL: http://www.zdnet.com/article/third-platform-shift-triggers-enterprise-software-evolution/ (online; accessed: 25.05.2016).

[16] Tiered File System without Tiers. — URL: http://www.snia.org/sites/default/education/tutorials/2011/spring/file/ShepardLaura_Tiered_FileSystem_Without_Tiers_FINAL.pdf (online; accessed: 21.12.2015).