Примеры инфраструктурных решений, применяющихся в крупных сетевых проектах

Пример реализации инфраструктуры в Google

Сегодня Google это далеко не только поисковая система. Это огромная сервисно-ориентированная платформа для построения масштабируемых приложений, позволяющая выпускать и поддерживать множество конкурентоспособных интернет-приложений, работающих на уровне всей глобальной сети. Компания ставит перед собой цель постоянно строить все более и более производительную и масштабируемую инфраструктуру для поддержки своих продуктов. Рассмотрим основные принципы и компоненты этого инфраструктурного решения. Но сначала несколько цифр, для того, чтобы представить масштабы решения.

Система включает в себя около 1 миллиона недорогих серверов (почти 2 % всех серверов в мире).

Google включает в себя более 400 GFS кластеров. Один кластер может состоять из 1000 или даже 5000 компьютеров.

Десятки и сотни тысяч компьютеров получают данные из GFS кластеров, которые насчитывают более 10 петабайт дискового пространства.

Суммарные пропускная способность операций записи и чтения между дата центрами может достигать 40 гигабайт в секунду

Система BigTable позволяет хранить миллиарды ссылок (URL), сотни терабайт снимков со спутников, а также настройки миллионов пользователей

Google визуализирует свою инфраструктуру в виде трехслойного стека:

Продукты: поиск, реклама, электронная почта, карты, видео, чат, блоги и т.п.

Распределенная инфраструктура системы: GFS, MapReduce и BigTable

Вычислительные платформы: множество компьютеров во множестве датацентров

При этом поддерживаются два принципа:

- Легкое развертывание для компании при низком уровне издержек
- Больше денег вкладывается в оборудование для исключения возможности потерь данных

С точки зрения ИТ-инфраструктуры нас интересуют в первую очередь решения, расположенные на втором слое.

Распределенная файловая система GFS (Google File System)

GFS является наиболее, наверное, известной распределенной файловой системой. Надежное масштабируемое хранение данных крайне необходимо для любого приложения, работающего с таким большим массивом данных, как все документы в интернете. GFS является основной платформой хранения информации в Google. GFS — большая распределенная файловая система, способная хранить и обрабатывать огромные объемы информации.

GFS строилась исходя из следующим критериев:

- Система строится из большого количества обыкновенного недорого оборудования, которое часто дает сбои. Должны существовать мониторинг сбоев, и возможность в случае отказа какого-либо оборудования восстановить функционирование системы.
- Система должна хранить много больших файлов. Как правило, несколько миллионов файлов, каждый от 100 Мб и больше. Также часто приходится иметь дело с многогигабайтными файлами, которые также должны эффективно храниться. Маленькие файлы тоже должны храниться, но для них не оптимизируется работа системы.
- Как правило, встречаются два вида чтения: чтение большого последовательного фрагмента данных и чтение маленького объема произвольных данных. При чтении большого потока данных обычным делом является запрос фрагмента размером в 1Мб и больше. Такие последовательные операции от одного клиента часто читают подряд идущие куски одного и того же файла. Чтение небольшого размера данных, как правило, имеет объем в несколько килобайт. Приложения, критические по времени исполнения, должны накопить определенное количество таких запросов и отсортировать их по смещению от начала файла. Это позволит избежать при чтении блужданий вида назад- вперед.
- Часто встречаются операции записи большого последовательного куска данных, который необходимо дописать в файл. Обычно, объемы данных для записи такого же порядка, что и для чтения. Записи небольших объемов, но в произвольные места файла, как правило, выполняются не эффективно.

- Система должна реализовывать строго очерченную семантику параллельной работы нескольких клиентов, в случае если они одновременно пытаются дописать данные в один и тот же файл. При этом может случиться так, что поступят одновременно сотни запросов на запись в один файл. Для того чтобы справится с этим, используется атомарность операций добавления данных в файл, с некоторой синхронизацией. То есть если поступит операция на чтение, то она будет выполняться, либо до очередной операции записи, либо после.
- Высокая пропускная способность является более предпочтительной, чем маленькая задержка. Так, большинство приложений в Google отдают предпочтение работе с большими объемами данных, на высокой скорости, а выполнение отдельно взятой операции чтения и записи, вообще говоря, может быть растянуто.

Файлы в GFS организованы иерархически, при помощи каталогов, как и в любой другой файловой системе, и идентифицируются своим путем. С файлами в GFS можно выполнять обычные операции: создание, удаление, открытие, закрытие, чтение и запись.

Более того, GFS поддерживает резервные копии, или снимки (snapshot). Можно создавать такие резервные копии для файлов или дерева директорий, причем с небольшими затратами.

Архитектура GFS

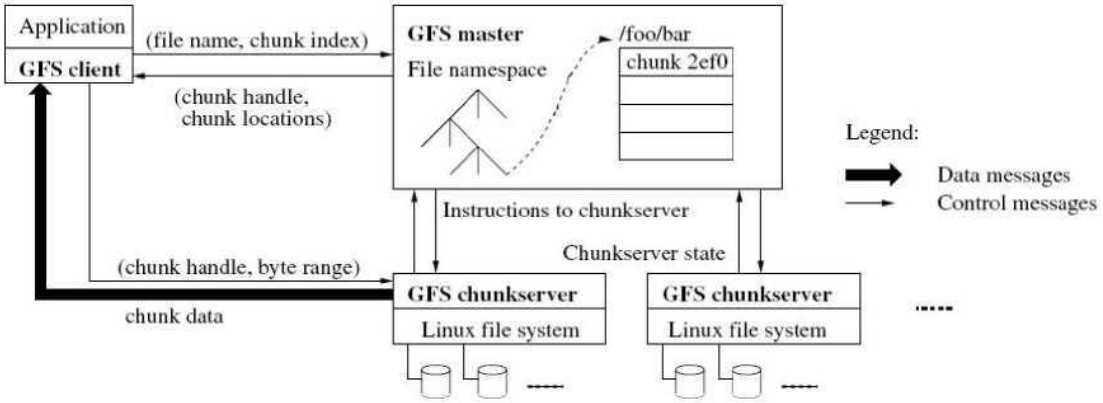

Рисунок 3.16

В системе существуют мастер-сервера и чанк-сервера, собственно, хранящие данные. Как правило, GFSknacmep состоит из одной главной машины мастера (master) и множества машин, хранящих фрагменты файлов чанк-серверы (chunkservers). Клиенты имеют доступ ко всем этим машинам. Файлы в GFS разбиваются на куски — чанки (chunk, можно сказать фрагмент). Чанк имеет фиксированный размер, который может настраиваться. Каждый такой чанк имеет уникальный и глобальный 64 — битный ключ, который выдается мастером при создании чанка. Чанк-серверы хранят чанки, как обычные Linux файлы, на локальном жестком диске. Для надежности каждый чанк может реплицироваться на другие чанк-серверы. Обычно используются три реплики.

Мастер отвечает за работу с метаданными всей файловой системы. Метаданные включают в себя пространства имен, информацию о контроле доступа к данным, отображение файлов в чанки, и текущее положение чанков. Также мастер контролирует всю глобальную деятельность системы такую, как управление свободными чанками, сборка мусора (сбор более ненужных чанков) и перемещение чанков между чанк-серверами. Мастер постоянно обменивается сообщениями (HeartBeat messages) с чанк- серверами, чтобы отдать инструкции, и определить их состояние (узнать, живы ли еще).

Клиент взаимодействует с мастером только для выполнения операций, связанных с метаданными. Все операции с самими данными производятся напрямую с чанк-серверами. GFS — система не поддерживает POSIX API, так что разработчикам не пришлось связываться с VNode уровнем Linux.

Разработчики не используют кеширование данных, правда, клиенты кешируют метаданные. На чанк-серверах операционная система Linux и так кеширует наиболее используемые блоки в памяти. Вообще, отказ от кеширования позволяет не думать о проблеме валидности кеша (cache coherence).

Мастер хранит три важных вида метаданных: пространства имен файлов и чанков, отображение файла в чанки и положение реплик чанков. Все метаданные хранятся в памяти мастера. Так как метаданные хранятся в памяти, операции мастера выполняются быстро. Состояние дел в системе мастер узнает просто и эффективно. Он выполняется сканирование чанк-серверов в фоновом режиме. Эти периодические сканирования используются для сборки мусора, дополнительных репликаций, в случае обнаружения недоступного чанк-сервера и перемещение чанков, для

балансировки нагрузки и свободного места на жестких дисках чанк-серверов.

Мастер отслеживает положение чанков. При старте чанк-сервера мастер запоминает его чанки. В процессе работы мастер контролирует все перемещения чанков и состояния чанк-серверов. Таким образом, он обладает всей информацией о положении каждого чанка.

Взаимодействия внутри системы

Каждое изменение чанка должно дублироваться на всех репликах и изменять метаданные. В GFS мастер дает чанк во владение(lease) одному из серверов, хранящих этот чанк. Такой сервер называется первичной (primary) репликой. Остальные реплики объявляются вторичными (secondary). Первичная реплика собирает последовательные изменения чанка, и все реплики следуют этой последовательности, когда эти изменения происходят.

Механизм владения чанком устроен таким образом, чтобы минимизировать нагрузку на мастера. При выделении памяти сначала выжидается 60 секунд. А затем, если потребуется первичная реплика может запросить мастера на расширение этого интервала и, как правило, получает положительный ответ. В течение этого выжидаемого периода мастер может отменить изменения.

Рассмотрим подробно процесс записи данных. Он изображен по шагам на рисунке, при этом тонким линиям соответствуют потоки управления, а жирным потоки данных.

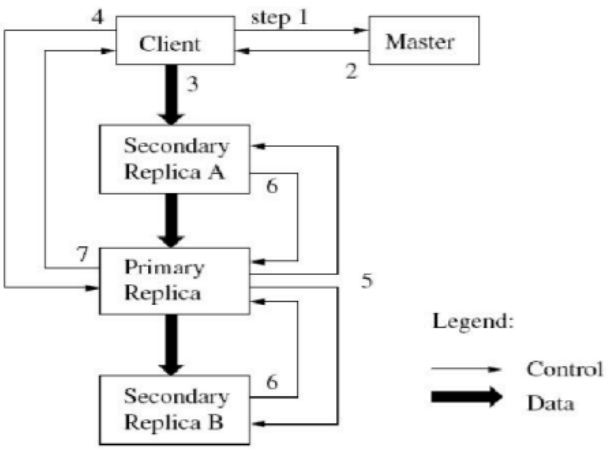

Рисунок 3.17

GFS поддерживает такую операцию, как атомарное добавление данных в файл. Обычно, при записи каких-то данных в файл, мы указываем эти данные и смещение. Если несколько клиентов производят подобную операцию, то эти операции нельзя переставлять местами (это может привести к некорректной работе). Если же мы просто хотим дописать данные в файл, то в этом случае мы указываем только сами данные.GFS добавит их атомарной операцией. Вообще говоря, если операция не выполнилась на одной из вторичных реплик, то GFS, вернет ошибку, а данные будут на разных репликах различны.

Еще одна интересная вещь в GFS — это резервные копии (еще можно сказать мгновенный снимок) файла или дерева директорий, которые создаются почти мгновенно, при этом, почти не прерывая выполняющиеся операции в системе. Это получается за счет технологии похожей на COPY on write. Пользователи используют эту возможность для создания веток данных или как промежуточную точку, для начала каких-то экспериментов.

Операции, выполняемые мастером

Мастер важное звено в системе. Он управляет репликациями чанков: принимает решения о размещении, создает новые чанки, а также координирует различную деятельность внутри системы для сохранения чанков полностью реплицированными, балансировки нагрузки на чанк-серверы и сборки неиспользуемых ресурсов.

Кластеры GFS, являются сильно распределенными и многоуровневыми. Обычно, такой кластер имеет сотни чанк-серверов, расположенных на разных стойках. Эти сервера, вообще говоря, доступны для большого количества клиентов, расположенных в той же или другой стойке. Соединения между двумя машинами из различных стоек может проходить через один или несколько свитчей. Многоуровневое распределение представляет очень сложную задачу надежного, масштабируемого и доступного распространения данных.

Когда мастер создает чанк, он выбирает где разместить реплику. Он исходит из нескольких факторов:

- Желательно поместить новую реплику на чанк-сервер с наименьшей средней загруженностью дисков. Это будет со временем выравнивать загруженность дисков на различных серверах.
- Желательно ограничить число новых создаваемых чанков на каждом чанк- сервере. Несмотря на то, что создание чанка сама по себе

быстрая операция, она подразумевает последующую запись данных в этот чанк, что уже является тяжелой операцией, и это может привести к разбалансировке объема трафика данных на разные части системы.

Во время работы мастер постоянно балансирует реплики. В зависимости от распределения реплик в системе, он перемещает реплику для выравнивания загруженности дисков и балансировки нагрузки. Также мастер должен решать, какую из реплик стоит удалить. Как правило, удаляется реплика, которая находится на чанк-сервере с наименьшим свободным местом на жестких дисках.

Еще одна важная функция, лежащая на мастере — это сборка мусора. При удалении файла, GFS не требует мгновенного возвращения освободившегося дискового пространства. Он делает это во время регулярной сборки мусора, которая происходит как на уровне чанков, так и на уровне файлов. Авторы считают, что такой подход делает систему более простой и надежной.

Мастер помимо регулярного сканирования пространства имен файлов делает аналогичное сканирование пространства имен чанков. Мастер определяет чанки, которые отсоединены от файла, удаляет их из метаданных и во время регулярных связей с чанк-серверами передает им сигнал о возможности удаления всех реплик, содержащих заданный чанк. У такого подхода к сборке мусора много преимуществ, при одном недостатке: если место в системе заканчивается, а отложенное удаление увеличивает неиспользуемое место, до момента самого физического удаления. Зато есть возможность восстановления удаленных данных, возможность гибкой балансировки нагрузки при удалении и возможность восстановления системы, в случае каких-то сбоев.

Устойчивость к сбоям и диагностика ошибок

Авторы системы считают одной из наиболее сложных проблем частые сбои работы компонентов системы. Количество и качество компонентов делают эти сбои не просто исключением, а скорее нормой. Сбой компонента может быть вызван недоступностью этого компонента или, что хуже, наличием испорченных данных. GFS поддерживает систему в рабочем виде при помощи двух простых стратегий: быстрое восстановление и репликации.

Быстрое восстановление — это, фактически, перезагрузка машины. При этом время запуска очень маленькое, что приводит к маленькой заминке, а

затем работа продолжается штатно. Про репликации чанков уже говорилось выше. Мастер реплицирует чанк, если одна из реплик стала недоступной, либо повредились данные, содержащие реплику чанка. Поврежденные чанки определяется при помощи вычисления контрольных сумм.

Важной частью системы является возможность поддерживать целостность данных. Обычный GFS кластер состоит из сотен машин, на которых расположены тысячи жестких дисков, и эти диски при работе с завидным постоянством выходят из строя, что приводит к порче данных. Система может восстановить данные с помощью репликаций, но для этого необходимо понять испортились ли данные. Простое сравнение различных реплик на разных чанк-серверах является неэффективным. Более того, может происходить несогласованность данных между различными репликами, ведущая к различию данных. Поэтому каждый чанк-сервер должен самостоятельно определять целостность данных.

Организация работы с данными при помощи MapReduce

MapReduce является программной моделью и соответствующей реализацией обработки и генерации больших наборов данных. Пользователи могут задавать функцию, обрабатывающую пары ключ/значение для генерации промежуточных аналогичных пар, и сокращающую функцию, которая объединяет все промежуточные значения, соответствующие одному и тому же ключу. Многие реальные задачи могут быть выражены с помощью этой модели. Программы, написанные в таком функциональном стиле автоматически распараллеливаются и адаптируются для выполнения на обширных кластерах. Система берет на себя детали разбиения входных данных на части, составления расписания выполнения программ на различных компьютерах, управления ошибками, и организации необходимой коммуникации между компьютерами. Это позволяет программистам, не обладающим опытом работы с параллельными и распределенными системами, легко использовать все ресурсы больших распределенных систем.

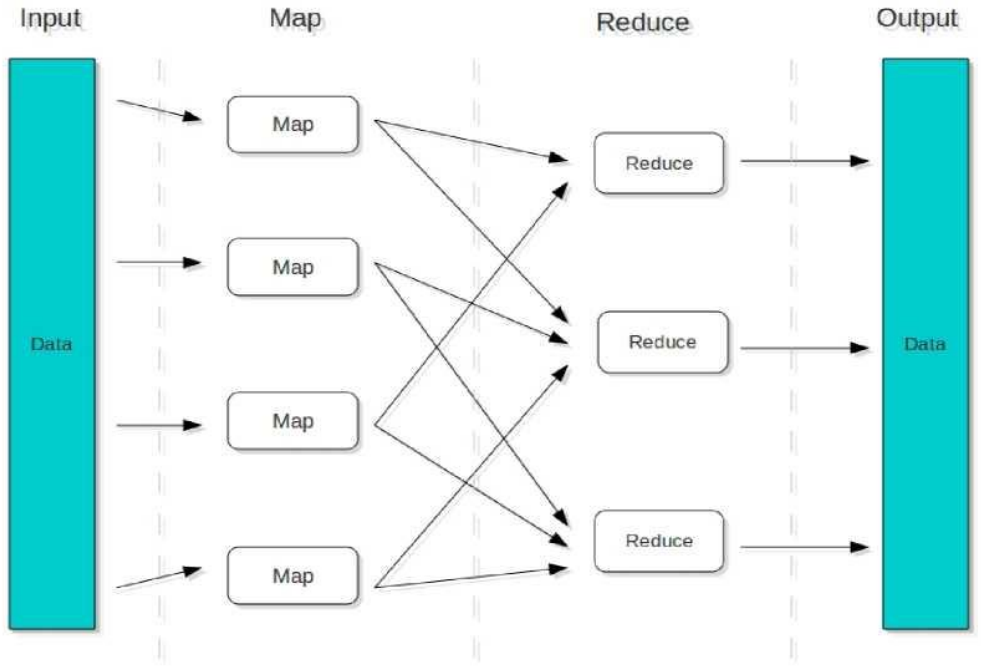

Рисунок 3.18

Преимущества использования MapReduce:

- Эффективный способ распределения задач между множеством компьютеров
- Обработка сбоев в работе
- Работа с различными типами смежных приложений, таких как поиск или реклама. Возможно предварительное вычисление и обработка данных, подсчет количества слов, сортировка терабайт данных и так далее
- Вычисления автоматически приближаются к источнику ввода-вывода

MapReduce использует три типа серверов:

- Master: назначают задания остальным типам серверов, а также следят за процессом их выполнения
- Map: принимают входные данные от пользователей и обрабатывают их, результаты записываются в промежуточные файлы
- Reduce: принимают промежуточные файлы от Map-серверов и сокращают их указанным выше способом

Технология MapReduce состоит из двух компонентов: соответственно map и reduce. Map отображает один набор данных в другой, создавая тем самым пары ключ/значение, которпыми в нашем случае являются слова и их количества. В процессе перемешивания происходит аггрегирование типов ключей. Reduction в этом случае просто суммирует все результаты и возвращает финальный результат.

В процессе индексирования Google подвергает поток данных обработке около 20 разных механизмов сокращения. Сначала идет работа над всеми записями и аггрегированными ключами, после чего результат передается следующему механизму и второй механизм уже работает с результатами работы первого, и так далее.

Транспортировка данных между серверами происходит в сжатом виде. Идея заключается в том, что ограничивающим фактором является пропускная способность канала и ввода-вывода, что делает резонным потратить часть процессорного времени на компрессию и декомпрессию данных.

Хранение структурированных данных в BigTable

BigTable является крупномасштабной, устойчивой к потенциальным ошибкам, самоуправляемой системой, которая может включать в себя терабайты памяти и петабайты данных, а также управлять миллионами операций чтения и записи в секунду. BigTable представляет собой распределенный механизм хэширования, построенный поверх GFS, а вовсе не реляционную базу данных и, как следствие, не поддерживает SQL- запросы и операции типа Join. Она предоставляет механизм просмотра данных для получения доступа к структурированным данным по имеющемуся ключу. GFS хранит данные не поддающиеся пониманию, хотя многим приложениям необходимы структурированные данные.

Коммерческие базы данных попросту не могут масштабироваться до такого уровня и, соответственно, не могут работать с тысячами машин одновременно.

С помощью контролирования своих низкоуровневых систем хранения данных, Google получает больше возможностей по управлению и модификации их системой. Например, если им понадобится функция, упрощающая координацию работы между датацентрами, они просто могут написать ее и внедрить в систему. Подключение и отключение компьютеров к функционирующей системе никак не мешает ей просто работать. Каждый блок данных хранится в ячейке, доступ к которой может

быть предоставлен как по ключу строки или столбца, так и по временной метке. Каждая строка может храниться в одной или нескольких таблицах. Таблицы реализуются в виде последовательности блоков по 64 килобайта, организованных в формате данных под названием SSTable.

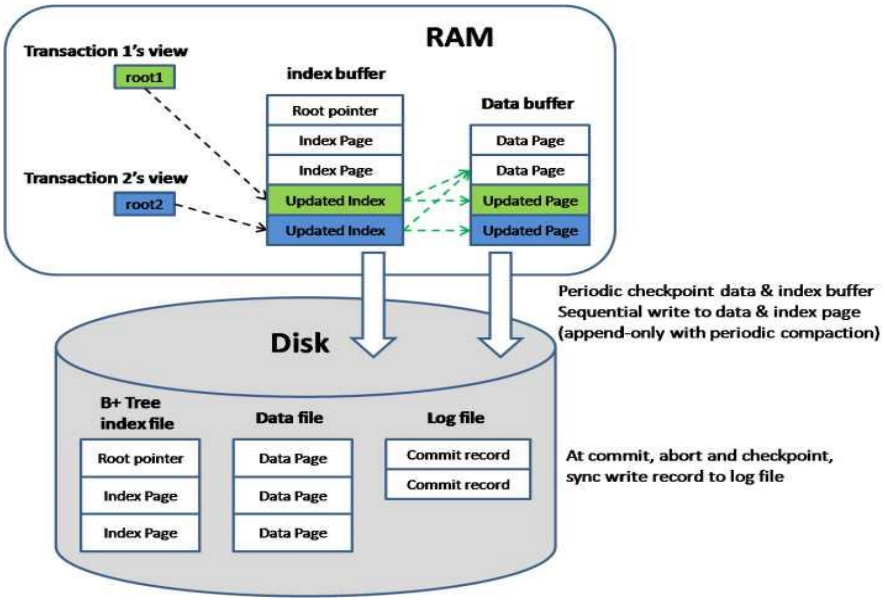

Рисунок 3.19

В BigTable тоже используется три типа серверов:

- Master: распределяют таблицы по Tablet-серверам, а также следят за расположением таблиц и перераспределяют задания в случае необходимости.
- Tablet: обрабатывают запросы чтения/записи для таблиц. Они раделяют таблицы, когда те превышают лимит размера (обычно 100-200 мегабайт). Когда такой сервер прекращает функционирование по каким-либо причинам, 100 других серверов берут на себя по одной таблице и система продолжает работать как-будто ничего не произошло.
- Lock: формируют распределенный сервис ограничения одновременного доступа. Операции открытия таблицы для записи, анализа Master-сервером или проверки доступа должны быть взаимноисключающими.

Локальная группировка может быть использована для физического хранения связанных данных вместе, чтобы обеспечить лучшую локализацию ссылок на данные. Таблицы по возможности кэшируются в оперативной памяти серверов.

Пример реализации инфраструктуры для проекта Flickr

Flickr является мировым лидером среди сайтов размещения фотографий. Перед Flickr стоит крайне непростая задача, они должны контролировать огромное количество ежесекундно обновляющегося контента, непрерывно пополняющиеся пользователи, постоянный поток новых предоставляемых пользователям возможностей, и при этом поддерживать постоянно высокий уровень производительности.

Использующиеся программные компоненты

Примечательно, что на проекте Flickr используется практически только свободное программное обеспечение.

- Платформа GNU/Linux (RedHat)
- СУБД MySQL
- Web-сервер Apache
- Скрипты программной логики, написанные на языке PHP и Perl
- Средства сегментирования (Shards) (прим.: разбиение системы на части, обслуживающие каждая свою группу пользователей; называть можно было по-разному, но давайте остановимся на этом варианте перевода)
- Memcached для кэширования часто востребованного контента
- Squid в качестве обратного прокси-сервера для html-страниц и изображений
- Шаблонизатор Smarty
- PEAR для парсинга e-mail и XML
- ImageMagick для обработки изображений
- SystemImager для развертывания элементов конфигурации
- Ganglia для мониторинга распределенных систем
- Subcon для хранения важных системных конфигурационных файлов в SVN- репозитории для легкого развертывания на машины в кластере
- Cvsup для распространения и обновления коллекций файлов по сети.

Типовое оборудование для серверов:

EMT64 под управлением RHEL 4 с 16 Gb оперативной памяти.

6 жестких дисков с 15000rpm, объединены в RAID-10.

Размер для пользовательских метаданных достигает 12 терабайт (это не включает фотографии).

Используются 2U корпуса.

Системная архитектура

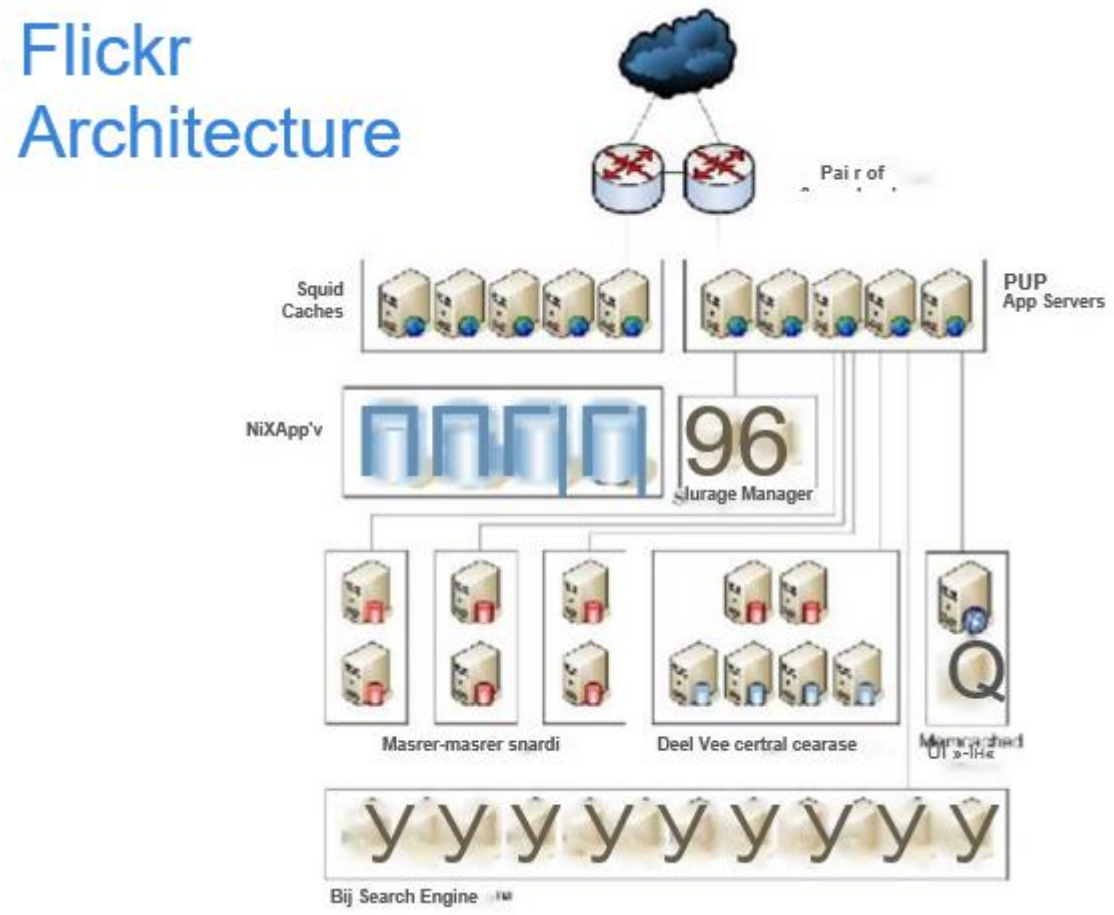

Рисунок 3.20

Рассмотрим наиболее характерные решения.

Входные запросы поступают на сдублированные контроллеры приложений Brocade ServerIron ADX. Они обеспечивают коммутацию приложений и балансировку трафика, основываясь на принципе виртуальных ферм серверов:

- Коммутатор приложений получает все клиентские запросы. Выбор “лучшего” сервера производится на основании механизма Real-Time Health и наличия требуемой производительности.
- Последовательно повышается коэффициент использования для всех серверов
- Интеллектуальное распределение загрузки осуществляется для всех доступных ресурсов
- Метод конфигурируется и выбирается пользователем

- Обеспечивается защита серверной фермы от атак и от неправильной эксплуатации
- Клиенты подсоединяются к серверам приложений используя виртуальный IP (VIP). VIP адреса настраиваются на коммутаторе приложений
- Коммутатор приложений осуществляет трансляцию адресов после выбора нужного сервера, причем сами адреса серверов скрыты.
- Обслуживание сессий ведется согласно последовательности:
- Запись о каждой пользовательской сессии создается в таблице
- Каждая сессия назначается определенному серверу
- Все сообщения в рамках сессии посылаются к одному серверу
- Таблицы сессий синхронизируются между двумя коммутаторами

За счет дублирования коммутаторов нет простоя сервиса, когда коммутатор вышел из строя: второй коммутатор обнаруживает отказ и начинает обслуживать сессии пользователей

Структура Dual Tree является индивидуальным набором модификаций для MySQL, позволяющим масштабировать систему путем добавления новых мастер-серверов без использования кольцевой архитектуры. Эта система позволяет экономить на масштабировании, так как варианты мастер-мастер требовали бы удвоенных вложений в оборудование.

Центральная база данных включает в себя таблицу пользователей, состоящую из основных ключей пользователей (несколько уникальных идентификационных номеров) и указатель на сегмент, на котором может быть найдена остальная информация о конкретном пользователе.

Все, за исключением фотографий, хранится в базе данных. Для статического контента используются выделенные сервера. Фотографии хранятся в системе хранения данных. После загрузки изображения система выдает различные его размеры, на чем ее работа заканчивается. Метаданные и ссылки на файловые системы, где расположены фотографии, хранятся в базе данных.

В основе масштабируемости лежит репликация. Для поиска по определенной части базы данных создается отдельная копия этого фрагмента. Активная репликация производится по принципу мастер-мастер. Автоматическое инкрементирование идентификационных номеров используется для поддержания системы в режиме одновременной активности обоих серверов в паре. При этом привязывание новых учетных записей к сегментам системы происходит случайным образом. Миграция

пользователей проводится время от времени для того, чтобы избавиться от проблем, связанных с излишне активными пользователями. Необходима сбалансированность в этом процессе, особенно в случаях с большим количеством фотографий.

Каждый сегмент содержит данные о более чем 400 тысячах пользователей. В системе заложены федеративные принципы сегментации: "Мои данные хранятся на моем сегменте, но запись о Вашем комментарии хранится на Вашем сегменте". При этом реализуется глобальное кольцо, принцип работы которого схож с DNS: "Необходимо знать куда Вы хотите пойти и кто контролирует то место, куда Вы собираетесь пойти". Логика, реализованная в виде PHP скриптов устанавливает соединение с сегментом и поддерживает целостность данных.