Оптимизация LAMP-приложения на примере OpenX: разгоняемся до 1000 запросов в секунду

Александр Чистяков, alexclear@gmail.com,
http://alexclear.livejournal.com
Санкт-Петербург, компания «DataArt»

Кто я?

- alexander@chistyakov@dataart.comalexander@chistyakov@dataart.com, alexclear@gmail.com
- http://alexclear.livejournal.com
- Работаю разработчиком ПО с 1998 года
- В настоящее время – разработка высоконагруженных веб-проектов, консультации по вопросам связанным с высокими нагрузками

Почему я?

- Нагрузка 500-5000 пользователей в день есть у всех
- Нагрузка 100500 запросов в секунду – мало у кого есть, но все читают о ней доклады
- Нагрузка 500-1000 запросов в секунду – должна быть интересна слушателям, но неинтересна докладчикам, «гонка за мегагерцы»
- Переход от 1rps к 1000rps порождает ряд однотипных классов проблем

Постановка задачи

- OpenX – существующее open source веб-приложение для показа рекламы
- Linux, Apache, MySQL, PHP
- Необходимо выдержать заданные параметры производительности при заданном количестве объектов предметной области

Предметная область

- OpenX: баннеры, кампании, зоны, пользователи
- Баннеры – собственно, баннеры
- Пользователи – управляют баннерами
- Кампании – содержат баннеры, имеют приоритеты
- Зоны – группируют баннеры и кампании для расчета весов

Начало

- Одна виртуальная машина в локальной сети
- 1000 баннеров, 1 зона, 1 кампания
- ~ 5 запросов в секунду

Требования

- Заказчик – большая компания, требования расплывчаты
- 200000 баннеров, 400-700 запросов в секунду

Особенности OpenX

- Расчет весов баннеров непосредственно в PHP коде при каждом показе
- Продукт уже оптимизирован, есть рекомендации по настройке под высокую нагрузку
- Рекомендации относятся к масштабированию DB уровня
- DB уровень не участвует в расчете весов!

Расчет весов

- Несколько циклов в PHP-коде
- 200000 баннеров – 200000 повторений в циклах
- Внутренние объекты PHP кэшируются в подключаемый кэш (memcached)
- Максимальный размер объекта для memcached – 1Мб
- 200000 баннеров – объекты размером несколько мегабайт

Оптимизация

- Декомпозиция объектов до уровня отдельных полей, вынос полей в memcached
- Веса рассчитываются один раз в 10 минут и кэшируются в DB
- Алгоритм сведения любого распределения к нормальному – веса объектов на отрезке [0,1], выбор случайного числа -> удобно построить индекс и сделать SQL-запрос

Первые проблемы

- Много запросов к memcached, он почему-то не работает – переход к хранению данных в APC
- Доллго рассчитываются значения весов – необходимо версионирование
- Несколько узлов, но APC локален – у каждого узла свой кэш объектов, взаимных блокировок нет

Тестирование: средства

- Siege, JMeter
- JMeter: создание разветвленных сценариев, GUI
- Siege: URL не меняется, командная строка
- Siege: до 700 rps на одной машине (Core i7)
- JMeter: до 240 rps на Core i7

Тестирование: результаты

- От трех до семи нод, одна DB
- Проблемы: ноды перетирают данные в базе (нет синхронизации) – сделали синхронизацию через memcached
- Проблемы: APC через некоторое время перестает работать – стандартный glibc аллокатор сильно фрагментирует память (вспомните браузер FF 2.0)

Решение проблем

- Назад к memcached (slab allocator)
- php-memcached работает, php-memcache - нет
- Нет под Debian Lenny, пришлось сдлать бэкпорт пакета из Sid
- Общий кэш, синхронизация через memcached на одной ноде
- Один экземпляр memcached на всех

Новые вводные данные

- Заказчик – большая компания, нас тоже много
- Требования конкретизируются прямо на ходу
- Зон может быть до 100
- Limitations – работают при каждом запросе, кэширование всего ряда весов на 10 минут дает ошибочные результаты, так как limitations тоже кэшируются при этом

Новые проблемы

- Если веса нельзя кэшировать, нужно их пересчитывать
- Данные кэшируются для зоны, 100 зон – 100 независимых пересчетов каждые 10 минут
- 200000 баннеров – 2000 баннеров в зоне – по 2000 повторений в циклах в коде PHP при каждом запросе (limitations!)
- 100 зон – 100 наборов таблиц в базе
- Что делать?

Варианты решения

- Поменять алгоритм выбора – варианты?
- Non-uniform distribution -> uniform distribution – только через отрезок, при этом 100 пересчетов
- Хотим один пересчет
- Вариант – наименьшее общее кратное весов, один отрезок с весами, выраженными через НОК
- 200000 баннеров - ~200000 записей в базе
- Не выражать через uniform distribution, использовать веса по-другому

Компромисс

- Баннеров в зоне не более 200
- Зон по-прежнему может быть 100
- Всего три типа limitations
- 200 повторений в циклах PHP – приемлемо
- Оставляем алгоритм расчета через uniform distribution

Изменения в коде

- Объекты баннеров до применения limitations хранятся в DB
- Limitations транслируются из срокового представления в реляционное – три связи в структуре кэш-таблиц в DB
- Применение limitations к баннерам это просто SQL-запрос

Пара слов о хостинге

- Первый этап – Amazon EC2
- Миграция на Rackspace Cloud
- Проблемы: средняя нода облачного хостинга недостаточно производительна, а большая – недостаточно велика
- Недостаточно производительности для создания тестовой нагрузки, недостаточно IOPS для эффективной работы DB

Тестирование: средства

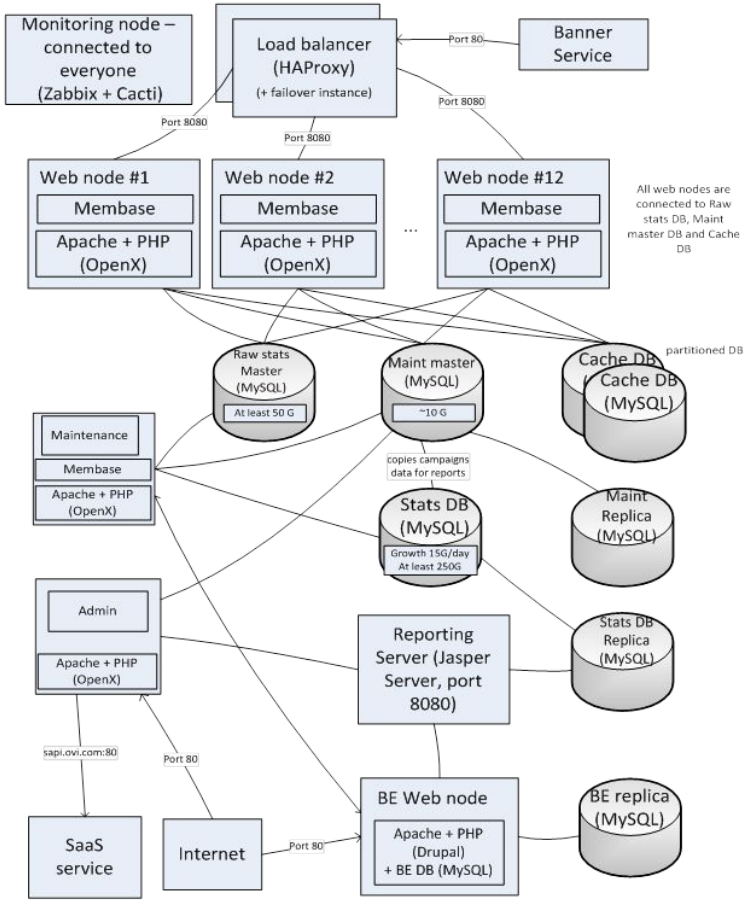
- Siege не подходит: 100 зон, около 50 параметров для каждого из лимитов – нужно менять URL
- JMeter: 240 rps на Core i7, в облаке – меньше
- Tsung

Tsung

- Tsung: написан на Erlang – распределенность на уровне VM языка
- Создавался с учетом многонодовых конфигураций
- Может генерировать необходимую нам нагрузку
- Строит отчеты в виде веб-страниц с графиками

Архитектура

- Представлена на картинке
- Картинку перерисовывали 7 раз



Распределение нагрузки

- nginx, HAProxy
- nginx – HTTP/1.0, генерирует кучу соединений, нет встроенного мониторинга состояния
- HAProxy – HTTP/1.1, мониторинг состояния на web-странице, предназначен именно для балансировки, можно задавать политику балансировки

Тестирование: проблемы 1

- 8 web-nod, одна DB, 100 rps
- Одна нода memcached не выдерживает поток запросов
- Укрупнение объектов для кэширования в memcached
- Распределенный memcached – на уровне библиотеки
- Экземпляр memcached на каждой ноде

Тестирование: проблемы 2

- 12 web-nod, одна DB, ~250 rps
- Большая нагрузка на web-ноды
- Из 4-х циклов расчета весов после применения лимитов к множеству баннеров в зоне 2 цикла можно кэшировать
- В коде осталось 2 цикла из 4-х

Тестирование: проблемы 3

- Включили maintenance скрипт в cron – перестала справляться DB
- Суть проблемы: раз в час таблица с raw logs очищается maintenance скриптом – блокировка таблиц на время удаления
- Очевидные решения: InnoDB вместо MyISAM, разбиение операции удаления на несколько мелких запросов – не помогают

Декомпозиция DB, тюнинг выделенной части

- Выделение raw logs в отдельную базу на отдельном узле
- Попытка поменять тип хранилища – MEMORY вместо InnoDB, ничего не дает, блокировки только хуже
- Мониторинг, тюнинг MySQL – добавление памяти под InnoDB buffer pool, log buffer

Варианты решения

- MariaDB vs Percona Server – разницы в производительности нет
- MySQL vs PostgreSQL
- NoSQL vs MySQL
- memcached – нет поддержки списков, Redis – есть поддержка списков, то, что нужно
- Проблема: нужно переписывать код
- Решение: никуда не мигрировать

Мониторинг

- Zabbix, Cacti
- Cacti: mysql-cacti-templates от коллег из Percona
- Zabbix: сильно нагружает сервер, data backend не в RRD, а в RDBMS, неоптимальные запросы (и неоптимизируемые)
- Zabbix: выше частота опроса, легче смотреть моментальные состояния

Тюнинг FS

- По умолчанию ext3 с data=ordered
- Перемонтировали с data=writeback, iowait вместо ~10% стал ~1.5%
- Перемонтировали FS на всех DB нодах с data=writeback

Тестирование: проблемы 4

- Теперь не справляются кэш-таблицы
- На MySQL скачки I/O
- Большое количество uncheckpointed bytes в мониторинге
- Решение: вынести кэш-таблицы в отдельную DB
- Решение: поменять тип хранилища на MEMORY
- TRUNCATE TABLE работает очень быстро

Тестирование: проблемы 5

- 12 web-нод, ~300 rps, три ноды DB
- Проблема: скачки I/O на cache DB
- Проблема: тест в Tsung бежит 6 часов, после чего падает
- Мониторинг: корреляция падений Tsung и скачков I/O на DB

Шардинг

- Мысли о шардинге были с самого начала, но по какому параметру разделять по шардам? И какие данные?
- После декомпозиции на три базы ответ стал очевиден: нужно шардить по номеру зоны данные, хранящиеся в кэш-таблицах
- Безграничные возможности для шардинга
- Проблема: не все зоны получают одинаковую нагрузку

Тестирование

- 12 web-нод, 300 rps, три ноды cache DB, одна нода raw logs DB и одна maintenance DB – тест бежит 42 часа, потом падает
- ~650 rps – тест бежит 6-7 часов
- Мониторинг: нет корреляции падений Tsung и событий в системе
- Вывод: проблемы Tsung

Предел

- ~700 rps – начинаются ошибки на балансере
- Мониторинг: нет корреляции с событиями в системе
- Что падает - непонятно
- Но задание уже выполнено – заказчик счастлив при 300 rps и пике в 600 rps в течение нескольких часов

Отказоустойчивость

- SPOF – распределенные узлы memcached
- При падении одного узла таймаут на обращении к нему – все ложится
- Варианты решения: репликация memcached, проксирование memcached
- SPOF: узлы DB

Отказоустойчивость: Moxi

- Проксирование запросов к memcached – Moxi
- Составная часть проекта Membase
- Работает на 127.0.0.1:11211
- Знает топологию узлов memcached, скрывает ее от пользователя
- Может мгновенно исключать упваший узел
- Не может перенаправлять запросы к другим узлам в standalone конфигурации

Отказоустойчивость: Membase

- Работает на порту memcached по его протоколу
- Два режима работы: с persistence и как кэш
- Web-интерфейс (не конфигурируется через текстовые файлы)
- При падении узла запросы автоматически перенаправляются на другие узлы
- Данные, бывшие на узле, теряются – приложение должно инициировать пересчет

Отказоустойчивость: MySQL

- Master-slave репликация – не средство обеспечения автоматической отказоустойчивости
- Master-master репликация – менее распространена, делается большим напряжением ума
- SLA 99.95% - достаточно, чтобы время восстановления базы после сбоя было постоянным (InnoDB, обойдемся без репликации)
- DRBD + Heartbeat + Xen

Развертывание

- Chef, Puppet
- Оба написаны на Ruby
- Про Puppet есть книга, про Chef нет
- Chef более ориентирован на развертывание Ruby-проектов
- Puppet: вся конфигурация на сервереЮ, клиент получает инструкции и разворачивает ноду
- Написаны Puppet-скрипты

Команда

- Участвовало от 5 до 8 человек
- Разработка, интеграция, тестирование, документирование, координация
- Производительностью занимались выделенные разработчики
- Начало внедрения через полгода после старта проекта

Что дальше?

- Security assesment
- Deployment
- Релиз
- Задача: распределить ввод-вывод по нескольким нодам параллельно
- Задача: обсчитывать большие объемы для получения аналитических отчетов
- Hadoop, MapReduce jobs вместо SQL

Выводы

- Времени всегда очень мало, вариантов может быть очень много
- Система не должна быть черным ящиком
- Не верьте в магию
- Прежде, чем оптимизировать, нужно измерить
- Знание высокоуровневых принципов оптимизации не спасает от огромного количества рутины – лучше иметь ответы на вопросы заранее

Вопросы?

-
-
-
-

Заказчик

- DataArt, http://www.dataart.com
- Enjoy IT!
- СПб, Большой Сампсониевский, 60А