Тестирование производительности

1 Конфигурация

Тестирование выполнялось на двух серверах.

Сервер 1:

- nginx 1.0.6, CT++ модуль (http://ngx-ctpp.vbart.ru/),
- php-fpm 5.3 + APC (http://php.net/manual/en/book.apc.php)
- Yii 1.1.8 (http://www.yiiframework.com/)
- Node.js 0.4.11 (http://nodejs.org/)

Сервер 2:

- nginx 1.0.6, CT++ модуль (http://ngx-ctpp.vbart.ru/),
- php-fpm 5.3 + APC
- Node.js 0.4.11 (http://nodejs.org/)
- MongoDB 2.0 (http://www.mongodb.org/downloads)
- Redis 2.2.12 (http://redis.io/)
- PostgreSQL 9.1 (http://www.postgresql.org/)

1. Для CT++ необходимо указать:
   - templates_root - директорию, откуда он будет подгружать шаблоны (локально)
   - ctpp2_data_buffer 512K
3. В nginx подключить php-fpm для тестового сайта, например /var/www/yii/site
4. APC отдать под кеш не менее 200Mb, apc.stat=0
5. Yii положить на 1 уровень выше doc_root тестового сайта. Например /var/www/yii/framework и /var/www/yii/site
6. Под скрипты node.js выделить директорию по аналогии с Yii. Например /var/www/node/site
7. Через npm под node.js установить модули:
   - Socket.io (https://github.com/learnboost/socket.io)
   - Redis + hiredis
   - MongoDB
8. В PostgreSQL завести БД, для тестирования можно использовать root-доступ

2 Результаты тестирования

Инструментарий:

1. apache benchmark (ab):
   запускается с параметрами -n 5000 -c 100
2. siege:
   запускается с параметрами -d0 -r5000 -c100
3. httperf:
   запускается с параметрами --hog --wsess 1000,1000,0.01 --rate 50 --timeout 5 --method GET и --hog --wsess 1000,1000,0.1 --rate 50 --timeout 5 --method GET

2.1 Сравнительный анализ времени инициализации основных фреймворков

- Yii (1.1.8): время обработки минимального запроса от 27 до 47ms, avg: 34ms
- Symfony (2.0.1): время обработки минимального запроса от 46 до 92ms, avg: 72ms
- ZF (оптимизированная сборка): время обработки минимального запроса от 114 до 201ms, avg: 132ms

Вывод: Yii имеет минимальное время инициализации и на запросах выдачи из кеша или обработки мелких запросов показывает более чем приемлимое время ответа

2.2 Сравнительный анализ основных шаблонизаторов

Шаблонизаторы	CT++ (2.7.1)	Smarty (3.10)	Twig (1.1.2)
Простой вывод 3-х переменных			
одиночный вызов без фреймворка	4-5ms	6-7ms	6-7ms
- при первом обращении	4-5ms	14ms	12ms
Вывод таблицы в 1000 строк и 3 столбца			
при первом обращении	33ms	112ms	110ms
при последующих обращениях	33ms	79ms	82ms
Lebowski benchmark (построение типовой страницы с хидером, колонкой навигации слева, анотациями статей, ссылками на полную версию, облаком тегов и футером)			
одиночный запрос при первом обращении	16ms	77ms	82ms
одиночный запрос под последующих обращениях		27ms	28ms
под нагрузкой	27ms	79-101ms	82-104ms
запросов в секунду	1531	1213	1191
расход памяти при полной нагрузке	324Mb	387Mb	372Mb

Впечатления от использования шаблонизаторов:

- CT++ - шаблоны полностью отделены от приложения, делается предварительная компиляция всех шаблонов, компиляцию легко включить в деплой-срипт, язык шаблонов smarty-подобный по возможностям, но чуть более громоздкий из-за конструкций вида <TMPL_var total>, <TMPL_if sections> - смарти это {$total}, {if $sections}
- Smarty 3 - привычный в использовании, расширился синтаксис, удобно использовать, интегрируется в фреймворк
- Twig - схож со смарти 3, но еще более лаконичный и удобный с точки зрения конструкций, интегрируется в фреймворк

Вывод: CT++ наиболее быстр, потребляет меньше ресурсов, лучше держит нагрузку. позволяет полностью отказаться от логики представления в приложении. позволяет унифицировать выдачу приложения. Так же важным преимуществом является не зависимость от языка разработки: используя CT++ можно шаблонизировать выдачу php-приложения и, например, node.js Twig и Smarty - более привычные и, возможно, более удобные в использовании.

2.3 Проверка типовой схемы построения ответа

Сценарий:

1. HTTP GET запрос от клиента
2. передача Yii на исполнение
3. выборка нужных блоков из кеша
4. отдача ct++
5. возврат результата клиенту

Инструментарий:

apache benchmark (ab): запускается с параметрами -n 5000 -c 100

- выборка данных одним запросом к кешу: полное время прохождения запроса - 14ms, под нагрузкой - 66ms
- выборка данных блоками и слияние в итоговый json (5 блоков): 14ms, под нагрузкой - 72ms, 1222 запроса/сек
- выборка данных блоками и слияние в итоговый json (45 блоков): 14ms, под нагрузкой - 79ms, 1151 запрос/сек

Вывод: по результатам тестирования видно, что при типичной модели построения страницы время формирования страницы не превышает 80мс при наличии в кеше всех нужных данных. в секунду такая система потенциально способна обработать 1200 запросов или 4.3 млн в час, что позволяет отказаться от кеширования html и опираться на кеширование данных.

2.4 Тестирование отказоустойчивости mongodb replica set

Сценарий:

Для теста поднято 3 инстанса mongodb:

- /data/service/mongodb/bin/mongod --rest --replSet ntv --port 27017 --dbpath /data/r0
- /data/service/mongodb/bin/mongod --replSet ntv --port 27018 --dbpath /data/r1
- /data/service/mongodb/bin/mongod --replSet ntv --port 27019 --dbpath /data/r2

После конфигурирования реплики, был проведен последовательное отключение на 27017 и 27019 портах. После каждого отключения инстанса mongodb был проверен статус и доступ из PHP к оставшимся серверам. Статус мастера оперативно менялся, данные, полученные из БД - актуальны. После падения мастера php-клиент автоматически подключается к новому мастеру в реплике. Полная строка подключения выглядит так:

```
$m = new Mongo(
"mongodb://193.232.148.35:27017,193.232.148.35:27018,193.23
2.148.35:27019", array("replicaSet" => true));
```

При такой строке подключения падение серверов на 27017 и 27019 портах не влияет на работу клиента.

Возможна такая строка подключения

(master - 193.232.148.35:27018):

```
$m = new Mongo("mongodb://193.232.148.35:27017",
array("replicaSet" => true));
```

При таком подключении обращение идет к мастеру (определение происходит автоматисчески), но в случае отключения 193.232.148.35:27017 при попытке коннекта происходит ошибка "Transport endpoint is not connected".

Минусы решения:

1. oplog имеет ограничение на размер коллекции, соответственно если мастер долго лежал, то самостоятельно он не сможет синхронизироваться - старые записи могут быть вытеснены из коллекции oplog более новыми.Это не проблема - есть команда синхронизации, которая ве это делает не опираясь на оплог и при этом делает это с балансировкой нагрузки - необходимо выполнить на слейве команду {resync:1} или запустить слейв с ключом --autoresync
2. При ручном останове одного из инстансев наблюдается лаг

драйверов php - появляется ошибка "couldn't send command", которая после перезагрузки страницы пропадает. При этом если остановить инстанс и перегрузить php-fpm, то ошибка не возникает. Видимо ошибка появляется в следствии кеширования php адреса мастер-сервера. Коннект через консоль монги к реплике происходит полностью прозрачно и отстрела инстанса вообще не замечаешь. Возможно, такое поведение php есть следствие разнесения инстасев mongodb по портам, а не по хостам. В текущей конфигурации проверить это не представляется возможным.

2.5 Тестирование mongodb

1. Вставка 10 000 000 записей вида Array ([_id] => MongoId Object ([$id] => 4e8d2fe42b6ac6d41b0000a3) [i] => 163 [status] => 1 [x] => 8973 [y] => 20433) поштучно - 592 сек
2. Вставка 10 000 000 записей вида Array ([_id] => MongoId Object ([$id] => 4e8d2fe42b6ac6d41b0000a3) [i] => 163 [status] => 1 [x] => 8973 [y] => 20433) блоками по 10 000 - 392 сек
3. update всех записей коллекции с инкрементом "y" на 1 - 927 сек
4. в момент выполнения update в эту же коллекцию было вставлено поштучно 10 записей - запись прошла успешно, счетчики обновились
5. выборка в момент выполнения update: из 3-х попыток 2 успешные, 1 отвалилась по таймауту.