Классификация систем хранения и обработки данных

Климов Евгений aka Slach
www.l-jet.ru
http://slach.livejournal.com

Что общего в архитектуре любого web-приложения среднего уровня?

у нас ~2.2 миллиона DAU, ~40kk php req, avg req time 0.03 ;-)

- Обработка соединений (nginx, apache, node.js, tornado и т.п.)
- Application Layer (php,python,v8, jvm, asp.net и т.п.)
- СИСТЕМЫ ХРАНЕНИЯ ДАННЫХ (filesystem, memory, sql, nosql)

- Наш (и не только) опыт показывает, что практически не важно какую технологию вы используете для Application Logic, узким местом является система хранения, причем зачастую не потому что «плохая», а потому что используется не правильно.

Как мы классифицируем системы хранения 1) где и сколько храним данных, протокол

Тип хранения

- Где именно система хранит данные
- RAM (быстро), SSD, SAS, HDD во всех проявлениях (медленнее)
- Гибрид RAM+Диск (IMHO оптимально)

Доступность по сети (TCP\UDP)

Ограничение на размер данных

- **Соотношение размер\скорость работы**. На текущий момент все рассматриваемые продукты имеют достаточно большие лимиты на максимальный размер данных, но для всех систем хранения есть предел, после которого «активная часть данных» начинает тормозить

Как мы классифицируем системы хранения 2) что и как пишем

- Надежность записи (защита от сбоев, допустимая величина потерь, ACID)
- Сложность структур данных, доступных для записи (таблицы, объекты, списки, массивы, хеши, деревья и т.п.)
- Соотношение Размер структуры и Скорость записи в req/sec
- Конкурентность и масштабируемость записи (горизонтальная желательно)
- Возможность проверки консистентности данных на стороне системы хранения
- Возможность BULK (BATCHING) записи
- Возможность асинхронной (DELAYED) записи

Как мы классифицируем системы хранения 3) что и как читаем

- Надежность чтения (актуальность данных на момент чтения, допустимые потери актуальности)
- Сложность языка (api) запросов и структур данных, доступных для чтения (SQL, XQuery, REST, GET\SET)
- Соотношение Размер «порции данных» (recordset, nodeset) и скорость чтения в req/sec
- масштабируемость чтения (горизонтальная желательно) и конкурентность (где происходит блокировка, buzy lock и т.п.)

Как мы классифицируем системы хранения 4) как этим управлять

- Переносимость (доступность для альтернативной win32 платформы ;) и легкость развертывания (пакеты, порты и т.п.)
- Простота, гибкость и глубина конфигурирования
- Управление масштабированием (из приложения или «коробочно» на уровне системы хранения)
- Легкость (скорость и простота) backup\restore
- Легкость операций по изменению структуры хранения данных
- Доступность и глубина «мониторинга» (готовые шаблоны для cacti, nagios, munin, zabbix что можно мониторить и т.п.)
- Возможности устранения failover

Как мы классифицируем «данные» 5) какой характер работы с данными

- Соотношение чтение\запись?
- Сложность выборок
- Оперативные данные или «аналитика» (OLTP \ OLAP)?
- Размер «активной части данных» (на запрос, на все приложение)
- Легкость изменения структур данных (schema-lock, schema-less) и легкость (прозрачность) «re-sharding» в случае горизонтального масштабирования

Классификация на практике
Хранение

Фактор\Продукт	MySQL (InnoDB)	Memcache	APC	Redis	FileSystem(ext, reiserfs, NTFS)	GlusterFS
ХРАНЕНИЕ						
Где храним	RAM+диск	RAM	RAM	RAM + диски	диск + RAM (кеш)	диск + RAM (кеш)
Сетевой протокол	TCP	TCP + UDP	нет	TCP	нет	Infiniband или TCP
соотношение скорость\размер (на один box)	сотни Gb	десятки Gb	единицы Gb	десятки Gb	единицы Tb	десятки Tb

Классификация на практике
Запись (часть 1)

Фактор\Продукт	MySQL (InnoDB)	Memcache	APC	Redis	FileSystem (ext, reiserfs, NTFS)	GlusterFS
ЗАПИСЬ						
Надежность записи и хранения	Высокая	низкая (потеря данных после рестарта)	низкая (потеря после ребута)	Высокая	Высокая	Высокая
Защита от сбоев	Commit	нет	полагается на OS и железо RAM	полагается на OS и железо RAM + HDD	зависит от типа, полагается на драйвер блочного устройства =)	множество вариантов AFR, Stripe
Сложность структур данных	Таблицы и скалярные типы стобцов	Key-value	Key-value	Key-value, LIST, SET, ZSET, HASH	Key-value, TREE (много накладных расходов)	Key-value, TREE (много накладных расходов)
Соотношение размер структуры - скорость записи req\sec (на один box)	row < 10kb, rps <2.5k	value < 10kb rps ~5-10k valye <100kb rps ~1k	value = 1-100kb rps 50-100k	key, list < 10kb, rps ~90k set, hash< RAM, rps - 3-5k	size от 50kb до 10Mb, rps от 0.1 до 2k	size от 50kb до 10Mb, rps от 0.1 до 2k

Классификация на практике
Запись (часть 2)

Фактор\Продукт	MySQL (InnoDB)	Memcache	APC	Redis	FileSystem(ext, reiserfs, NTFS)	GlusterFS
ЗАПИСЬ						
Конкурентность	тред на коннект + innodb треды, до 1-2k коннектов на бокс	треды, настраиваются для 1.2+, есть глобальные локи, до 5-10k коннектов на box	shared memory (семафоры?), сегементы на процесс, коннектов нет	single thread, собственная библиотека для работы с event pool (epoll для Linux) до 10k коннектов на box	блокировки на уровне I\O операций файловой подсистемы OS, кол-во обращений от процессов ограничено CPU, размером данных и пропускной способностью блочных устройств	блокировки на уровне процессов gluster-client и gluster-server, плюс локальные блокировки на уровне локальных FS
Масштабируемость записи	Вертикальная (лучше диски, больше памяти) и горизонтальная Sharding, MySQL Cluster, много outbox решений	Вертикальная - больше RAM, больше тредов, больше ядер в CPU) горизонтальная Sharding (хеш на клиентских библиотеках)	слабая вертикальная (поставить более быструю память)	Вертикальная - несколько экземпляров Reids на Multi Core CPU машинах, Горизонтальная - Sharding (persistent hash на клиентских библиотеках) + Репликация,	Вертикальная - более быстрые диски, RAID	Горизонтальная - но кол-во узлов не бесконечно

Классификация на практике
Запись (часть 3)

Фактор\Продукт	MySQL (InnoDB)	Memcache	APC	Redis	FileSystem(ext, reiserfs, NTFS)	GlusterFS
ЗАПИСЬ						
Консистентность	Триггеры, Транзакции	нет	нет	**нет**	нет, файловая подсистема OS проверяет только контрольные суммы, что именно лежит в файле она не знает	нет, файловая подсистема OS проверяет только контрольные суммы, что именно лежит в файле она не знает
BULK запись	LOAD DATA	есть для binary протокола	нет	есть MSET	сама запись в файл, это в некотором роде BULK операция	см. FileSystem
DELAYED запись	INSERT DELAYED	нет	нет	нет	FLUSH из файлового кеша OS	FLUSH из файлового кеша OS

Классификация на практике
Чтение

Фактор\Продукт	MySQL (InnoDB)	Memcache	APC	Redis	FileSystem(ext, reiserfs, NTFS)	GlusterFS
			ЧТЕНИЕ			
Надежность чтения	Высокая	Высокая	Высокая	Высокая	Высокая	Высокая
Сложность API\языка запроса	SQL - высокая	GET\SET низкая	FETCH\\STORE низкая	средняя за счет SET, LIST, HASH структур	fread, fwrite - файловое API	fread, fwrite - файловое API
Скорость чтения	row < 1kb, rps <5k сильно зависит от сложности SELECT	value < 10kb rps ~15-20k valye <100kb rps ~0.7-1k -	value = 1-100kb rps 50-100k	key, list < 10kb, rps ~70k set, hash< RAM, rps - 3-5k	size от 50kb до 10Mb, rps от 0.1 до 2k	size от 50kb до 10Mb, rps от 0.1 до 2k
Масштабируемость чтения	вертикальная - сеть, диск, CPU горизонтальная Multi-slave репликация, Шардинг + Sphinx	горизонтальная на клиенте через pesistent hash	слабая вертикальная (поставить более быструю память)	горизонтальная на клиенте через pesistent hash + Muli-Slave репликация	Вертикальная - RAID 10, SSD и т.п.	Слабая горизонтальная в конечном итоге все упирается в пропускную способность каналов связи в ДЦ
Конкурентность чтения	тред на коннект, до 1-2k коннектов на box	до 5-10k коннектов на box	ограничено кол-вом скриптов работающих с данным shared memeory сегментом	однотредовый, до 10k коннектов на box	ограничено пропускной способностью дисков и кол-вом файловых дескрипторов, также возможны блокировки по flock и семафорам	ограничено пропускной способностью дисков кластера и кол-вом файловых НОД в кластере + блокировки gluster + блокировки локальной FS

Классификация на практике Управляемость (часть 1)

Фактор\Продукт	MySQL (InnoDB)	Memcache	APC	Redis	FileSystem (ext, reiserfs, NTFS)	GlusterFS
УПРАВЛЯЕМОСТЬ						
Переносимость	Очень высокая	Высокая (есть win32 порт)	Очень высокая (везде, где есть PHP)	Высокая (есть win32 сборки)	Средняя (некоторые типы FS есть на одной платформе но нет на другой примеры ZFS, NTFS, при этом для приложения все прозрачно	Низкая (под FreeBSD еще допиливать, под win32 только клиент через CIFS, сервера видимо никогда не будет)
Простота конфигурирования	Просто	Просто	Очень просто	Очень просто	Не всегда просто	Необходимо понимание работы
Гибкость конфигурирования	Высокая	Высокая	Достаточная	Достаточная	Не всегда достаточная	Высокая
Глубина конфигурирования	Очень высокая	Достаточная	Высокая	Достаточная	Не всегда достаточная	Высокая

Классификация на практике Управляемость (часть 2)

Фактор\Продукт	MySQL (InnoDB)	Memcache	APC	Redis	FileSystem (ext, reiserfs, NTFS)	GlusterFS
УПРАВЛЯЕМОСТЬ						
Управление масштабированием	MySQL Cluster из "коробки", mmm-monitor, maakit много прикладных решений	Внешнее с помощью puppet и на уровне приложение	Не нуждается в управлении, поскольку нет горизонтального масштабирования	Внешнее с помощью puppet и на уровне приложения	нет возможности, необходимо проверять некоторые FS по расписанию на "целостность"	реконфигурируется на лету, есть встроенные средства
Простота\скорость backup\restore	Много способов - LVM, ZFS snapshot, mysqldump, xtrabackup, HOT BACKUP для MySQL6	напрямую backup\restore нет, есть repcached (multi-master-repllication)	нет	есть .rdb и .log файлы, которые можно бекапить через lvm или обычным tar, потому что новый дамп делается "рядом"	любые доступные методы файлогового бекапа (rsync, lvm и т.п.) или зеркалирование в RAID	возможность репликации файлов на несколько узлов одновременно
Изменение структуры данных	ALTER TABLE - головная боль, Re-sharding - тоже не сахар, все более или менее прилично только в MySQL Cluster	просто перегружаем сервер и пишем в другой ключ либо GET+SET+DEL	FETCH old_key STORE new_key DEL old_key	автоматического решардинга нет =(RENAME, GET+SET+DEL	mv, cp, rm etc	mv, cp, rm etc

Классификация на практике Управляемость (часть 3)

Фактор\Продукт	MySQL (InnoDB)	Memcache	APC	Redis	FileSystem (ext, reiserfs, NTFS)	GlusterFS
УПРАВЛЯЕМОСТЬ						
Доступность мониторинга	cacti, zabbix, nagios	cacti, zabbix, nagios	cacti	cacti, zabbix, nagios	cacti, zabbix (SMART)	см. FileSystem - но нет заточки под специфические показатели Gluster
Глубина мониторинга	Очень глубоко, буквально каждый аспект	достаточно глубоко memcache_info - распределение slub, LRU, req\sec, available memory. Кроме того есть debug режим -vv	достаточно глубоко	не так глубоко как хотелось бы (нет распределения по командам, есть только req\con sec)	весьма глубоко iostat, lsof	только общие показатели у узлов по I/O, CPU, RAM нагрузкам
Устранение failover	MySQL Cluster, HAProxy, Multi-Master + mmm-monitor	restart демона	только рестарт php (см. Php-fpm например)	master-slave репликация или HAProxy	RAID 10	есть из "коробки", но мы еще до этого не доходили

Ок, а теперь «грабли» ;-)
MySQL InnoDB

- Все просто замечательно, пока какой то тип нагрузки преобладает (чтение для классических сайтов, запись для логов или аналитики)
- Как только надо много read+write из одного и того же места и нет времени на Replication Lag … после определенной concurency все равно наступает «жопа», 99% процентов выбирают Memcache для того чтобы упаковать в него «активную часть данных» и использует его как pesistent storage, а не как кеш ;-)
- Также весьма популярен sharding, основная проблема в нем правильный выбор ключа для хеширования, при этом решардинг (ребалансировка) и schema change – тоже головная боль

Ок, а теперь «грабли» ;-) memcache

- Dog-pile эффекты (lock через add при записи)
- Размер value одних ключей больше чем других. Следите на LRU, slubs и evicted
- Некоторые ключи читаются\пишутся чаще чем остальные, не допускайте чтобы этих ключей было МАЛО (один) и они после хеширования ложились на ОДИН сервер =)
- Пишите код с учетом того, что может навернуться канал связи в ДЦ (read\write\connect timeout) или вы можете просто упереться в потолок сетевухи
- Память дешевая, но не бесконечно дешевая, не храните в кеше ЛИШНИХ данных =)

Ок, а теперь «грабли» ;-)
APC

- ОЧЕНЬ быстрый, но Dog-pile эффекты никуда не делись (lock через apc_add есть забавный баг)
- Shared memory сегмент «на процесс», кеш не общий, может получиться дублирование данных
- МНОГО данных не положишь (гигабайты, сотни мегабайт на приложение) максимум десятки Mb
- Нельзя использовать как pesistent storage потому что горизонтально не масштабируется
- IMHO идеален для кеша read-only «справочных» данных

Ок, а теперь «грабли» ;-) FileSystem, GlusterFS

- Если «активная часть данных» умещается на SSD и есть деньги тащите туда. За минимизацией random seek будующее =)
- csv, grep, sed, awk + pipes никто не отменял
- GlusterFS непонятно еще как «мониторить», пока нет кластерных реализаций lsof и iostat и т.п.
- IMHO идеально для UGC (не видео) + метаданные в более «быстром» хранилище
- IMHO хранить (монтировать) лучше на Application серверах (запись+чтение) + Frontend (чтение)

Ок, а теперь «грабли» ;-)
Redis

- Все что справедливо для Memcache
- Single thread (пока еще) в век Multi Cure CPU и даже без worker pool management ;)
- Дамп отдельным тредом (за сколько времени ваши диски зальют 8Gb ?)
- Не устоявшийся набор команд и их поведение (пример сочетание SETEX + INCR)
- Осторожнее с maxmemory
- KEYS такой соблазнительный и такой «блокирующий» (RTFM юзайте SETS ;)
- Дублирование данных и не всегда эффективное хранение в памяти (мониторинга распределения ключей нет)

Ответы аудитории

- Серебренной пули нет ;)
- Из представленных систем хранения, по теореме CAP, MySQL это CA система, Redis, Memcache – AP
- Все что я сказал банально? Пожалуйста пройдемте к кулуары, я давно хотел поговорить с умным человеком ;-)