Защо инфраструктурата не ви обича

понякога и аз...

Васил Колев <vasil@ludost.net>

Кой съм аз

- Системен администратор
- Правил инфраструктурата на малки и големи системи
- от 1 сървър, 9600bps свързаност и 40 потребителя
- до 750 сървъра, 350gbps свързаност и десетки милиони потребители
- ISP, хостинги, телефония, криптография, streaming
- Човекът, който будят в 4 сутринта, ако нещо не работи

За какво ще ви говоря

- Основно за проблеми от моята практика
- CAP теорема
- Single points of failure
- Хоризонтален scaling
- Мрежови проблеми
- Линейни и комплексни системи
- ... и други неща, които пречат на здравия сън

За какво няма да говоря

- Storage
- Cloud (облак)
  - 50мл. мента, 100мл мастика, нищо сложно

Какво имам в предвид под “state”

- Бази данни, сесии, статус на мрежата...
- В общи линии всичко без хардуера и кода
- Всичките данни в една система

Какво е инфраструктура?

- Това, което е под вашия софтуер
- От оптика до бази данни и web сървъри
- Всичко, за което отговаря някой друг :)

CAP теорема

- Consistency, Availability, Partition tolerance - pick two
- ... или защо не можем просто да добавяме нови машини към базата данни

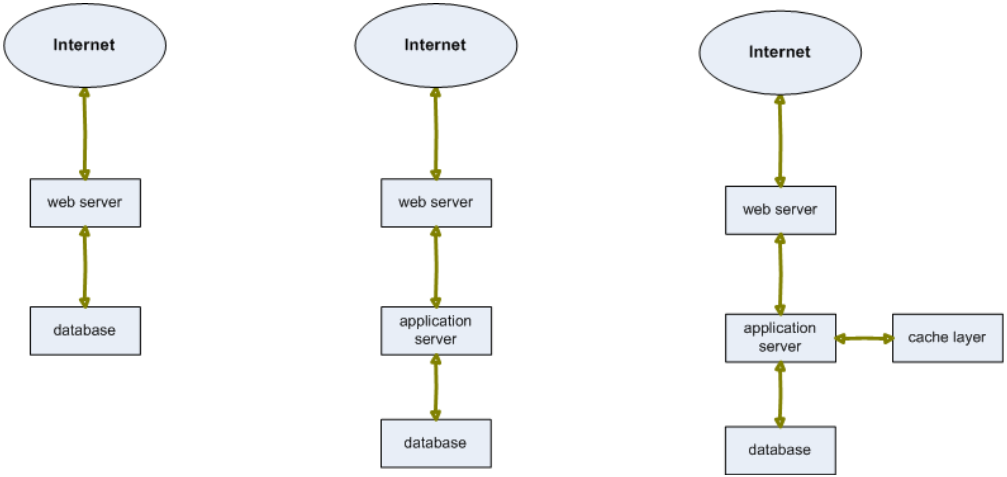


Рис.: Сферичен кон

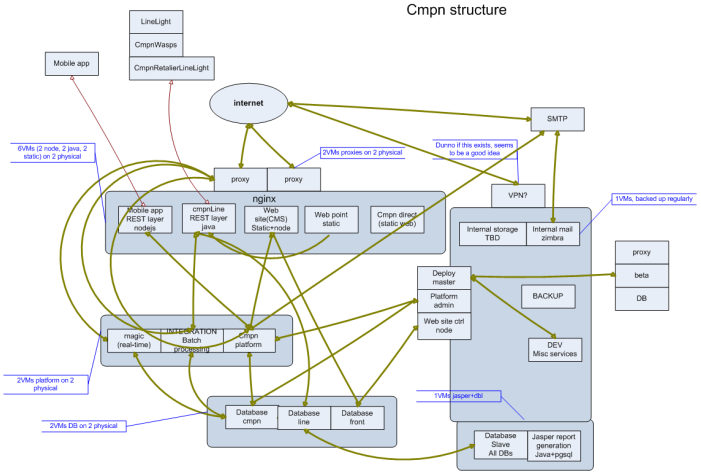


Рис.: Реалният свят

Каква е моята работа

- Да накарам софтуера да работи
- Да преча на услугата да умре поради проблеми на инфраструктурата

Single points of failure

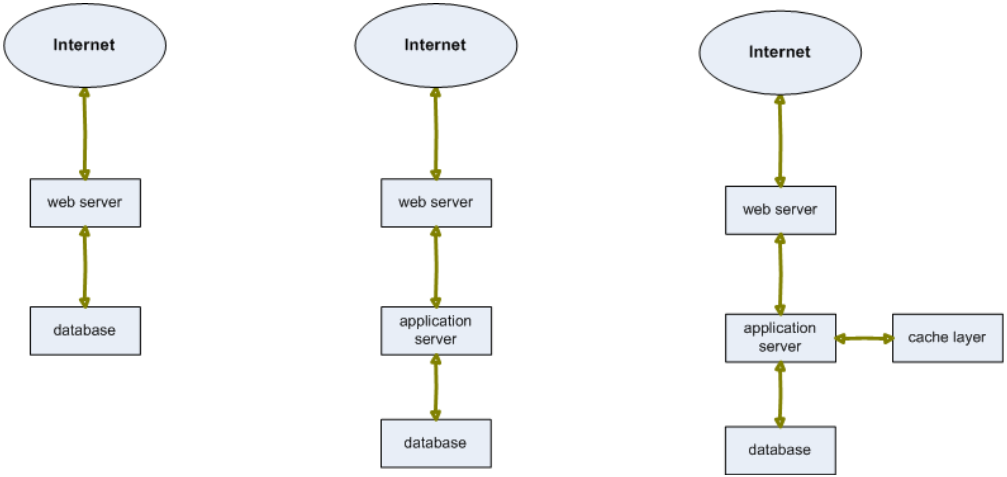


Рис.: Failures everywhere

Как се създават в софтуера

- Единственото, което пречи на дублирането на даден компонент е local state
  - Махнете го и може да се скалирате
- Всичко останало е въпрос на малко работа и разбиране с инфраструктурните хора

Как се избягват

- Прочита се CAP теоремата
- Използва се дистрибутиран/репликиран метод за съхраняване на state
- В днешните времена е трудно да не се избегне

Дистрибутиран/подсигурен state (1/2)

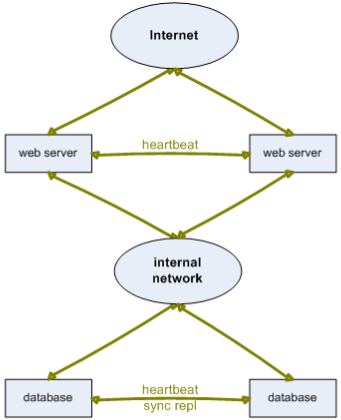
- Вариант, който винаги работи - синхронно репликиран втори сървър, който да се пусне ако главния падне
  - Скалируемост никаква

Дистрибутиран/подсигурен state (2/2)

- База данни/подобна система, която поддържа дистрибутиране
- В последните 10 години има много много много разработки по въпроса
- Повечето такива системи имат ограничения, които трябва да се вземат в предвид
- Никой не е свикнал с тези ограничения
- На всички им се иска безкрайна ACID база със SQL интерфейс
  - включително и на мен

Всичко останало

- ... се скалира с тривиални средства като:
  - load balancing
  - дублиране на компонент



rs

rs

Рис.: не-толкова-сферичен кон

TL;DR

- Дръжте си сесиите, cookie-тата и т.н. в shared storage
- Дайте си шанс да ползвате друг тип база
- Не искайте невъзможното

Хоризонтален scaling

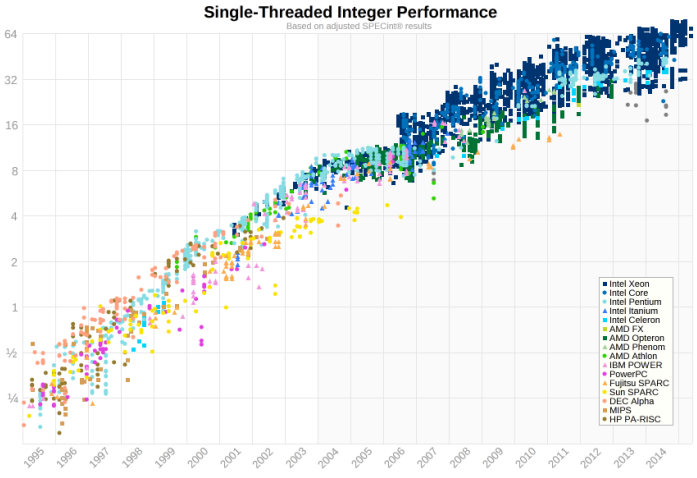


Рис.: Single core performance

Накъде вървим

- Големи клъстери от машини
  - без споделена памет
  - или с бавна такава
- State е изнесен (външен storage, бази данни)
- Паралелизация до смърт
- Бъдещето на доста неща - co-routine модел

Какво следва

- Да оптимизираме за латентност
- Да премахваме зависимостите доколкото можем
- Да избягваме единични компоненти, през които да минава всичко

Мрежи

Всичко е мрежа

- Архитектура на дънните платки
- SATA/SAS
- Почти на никого базата данни не е локално
- Storage вече е на друго място
- Всички сме в облаците

List of network fallacies

- The network is reliable.
- Latency is zero.
- Bandwidth is infinite.
- The network is secure.
- Topology doesn't change.
- There is one administrator.
- Transport cost is zero.
- The network is homogeneous.

Грешки от практиката

Internet и мрежите не са reliable

- Да се използва internet за reliable комуникация
  - синхронизация на бази данни
  - каквито и да е синхронни операции

Да забравим латентността

- Все още няма по-бърза светлина
- Да се работи синхронно с някаква услуга без да има смисъл
- Да се използват бавни протоколи
- Да не се оптимизират съществуващите
  - Малко хора се замислят, че handshake на SSL сесия може да бъде забързан двойно

Твърде много данни по твърде малка тръба

- Мрежите не могат да настигнат вътрешната комуникация на един сървър
- “RDMA”-тип действия
  - По-лесно е да кажем на две системи да си копират информацията м/у тях, вместо да я прекарваме през нас
  - Погледнете BitTorrent

Голи данни по пробита мрежа

- Данните, които са важни, трябва да се предават криптирано
- Много трудно се дефинира кои данни са важни
  - Направо е страшно колко информация може да се извади дори от трафични данни
- Всички данни са равни
  - Няма по-равни :)
- Извод: криптирайте всичко, ПРАВИЛНО

Мрежата ще отговаря на всичките ви изисквания

- Рядко цялата мрежа, от която зависите е под единен административен контрол
- Дори да е, тоя административен контрол не е магически
  - или просто не сте единствения, който поставя изисквания
- Извод: запознайте се с възможностите на мрежата ВНИМАТЕЛНО

. . . и т.н.

- Мрежата може много да ви помогне
- Може и да ви простреля в крака през главата

Store & Forward срещу pass-through

Дефиниция

- Приемане на цялата заявка и след това обработката и
- Или предаване на заявката директно нататъка

Примерна система

- Система, която обработва писма, с два компонента - mail сървър и app сървър
- Pass-through - mail сървъра за всяко получено писмо се свързва с app сървъра, дава му го да го обработи и чака отговор
- Store & Forward - mail сървъра съхранява получените писма и app сървъра си ги взима, за да ги обработи

Предимства и недостатъци

- Pass-through е с по-ниска латентност от S&F
  - Преди да се претовари
- Pass-through е по-чупливо от S&F
- Pass-through е малко по-сложно за дистрибутиране
- ...и много други, зависими от конкретната реализация/ситуация

Извод

- Ползвайте pass-through само ако знаете какво правите и трябва да ускорявате нещо
- По-маловажните компоненти трябва да са на store&forward и да може да се живее и без тях
- Failfast!

Линейни и комплексни системи

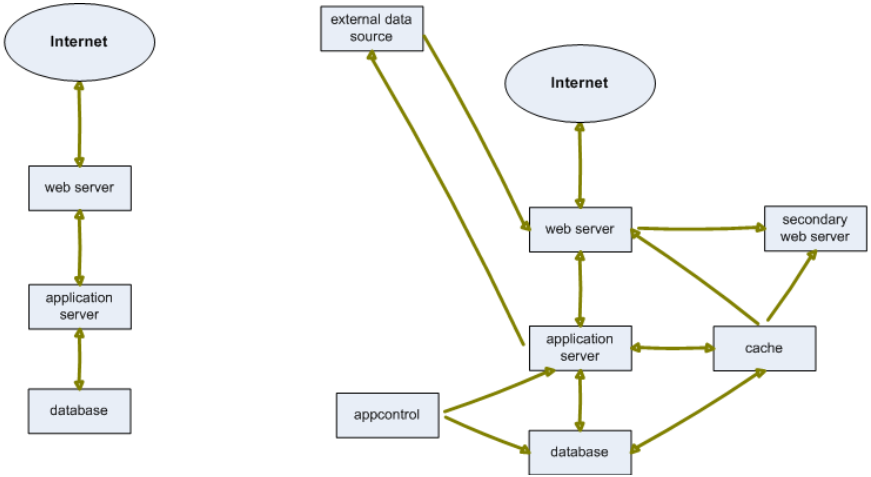


Рис.: Линейна и комплексна системи

(почти) Всичките ни системи са комплексни

- Airbus A380 има около 4 милиона компонента
- Само linux kernel-а в момента е над 12 милиона реда код
  - 1,179,146 условни прехода в компилиран вид
- Ако хванем целия си софтуерен стек, от който зависим и нашия собствен код, спокойно можем да стигнем 500 милиона
- На практика нямаме толкова голяма работеща система в реалния свят

Тъжни факти

- Трудно е да си направим работещ модел на комплексна система
  - съответно да разберем как може да се счупи
  - такава с над 25,000 компонента е направо магическа
- Нямаме начин да кажем какво ще се случи, ако повече от един компонент откаже
  - Пример - ядрени реактори
- Средно имаме по 15-50 бъга на 1000 реда production код.

Може да се мине със следния цитат

> *"Don't be fooled by the many books on complexity or by the many complex and arcane algorithms you find in this book or elsewhere. Although there are no textbooks on simplicity, simple systems work and complex don't"*

Jim Gray, "Transaction Processing"

Изроди

- Избягвайте комплексните системи, доколкото е възможно
- Правете ги МАЛКИ
- Търсете инструменти, с които да можете да ги докажете формално
- Failfast!

Bitrot

Какво е bitrot

- Записвате нещо и то се повреждa, без да ви върне грешка
  - hard disk-ове
  - памет
  - eeprom

Статистика от wikipedia

- 1.5 милиона диска за 41 месеца - 300,000 undetected disk corruptions
- от $2.5*10^{-11}$ до $7*10^{-11}$ шанс за грешка в паметите за всеки бит
  - над 8% от DIMM-овете дават поне една грешка годишно

Случки от реалния свят

- от 395 милиона файла (около 1PB, в/у около 100 сървъра), ~300 грешки в рамките на месец и половина
  - В рамките на последната седмица - 3 грешки
  - файловете са в/у raid масиви
- Тестов lab - редовно сървърите без ECC crash-ват софтуера
  - свестни сървърни машини
  - открили след известно време липсата на ECC
  - 16-32GB памет

row-hammer exploit

*"Just by reading from the same address >100K times, it is possible to corrupt data in nearby addresses."

`https://github.com/CMU-SAFARI/rowhammer`

Инструкции за цивилното население

- “Завийте се в бял чаршаф и пълзете бавно към гробищата”
- Прости системи
- Failfast
- Run-time проверки за коректност

Благодаря за вниманието!

Въпроси?

Книги

- “The Architecture of Open-source Applications”, vol. 1 и 2
- “The Performance of Open-source Applications”
- “Beautiful Data”
- “Normal Accidents: Living with High-Risk Technologies”, Charles Perrow
- “Cryptography Engineering”, Schneier, Ferguson & Konho