Основы построения файловых систем

Сегодня мы поговорим про NFS

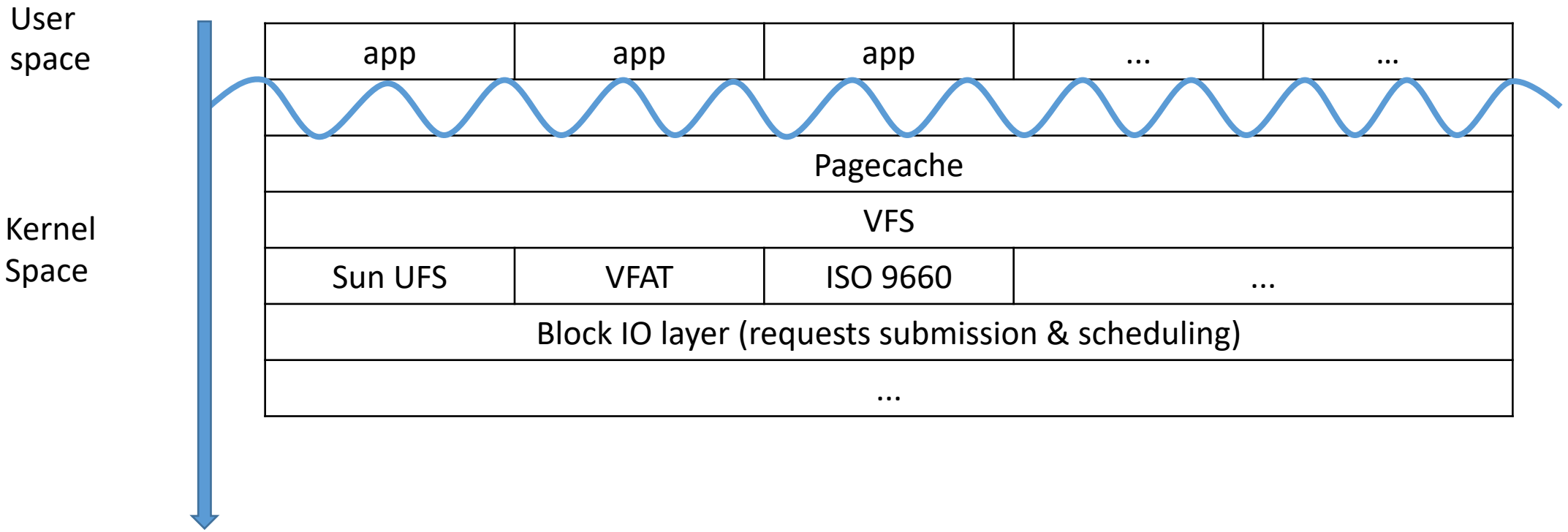
NFS (Network File System) – механизм для того, чтобы локальную ФС одного компьютера сделать доступной по сети.

Основные цели

- Доступ к ФС по сети должен быть таким же, как доступ к локальной ФС, притом даже на уровне ядра.

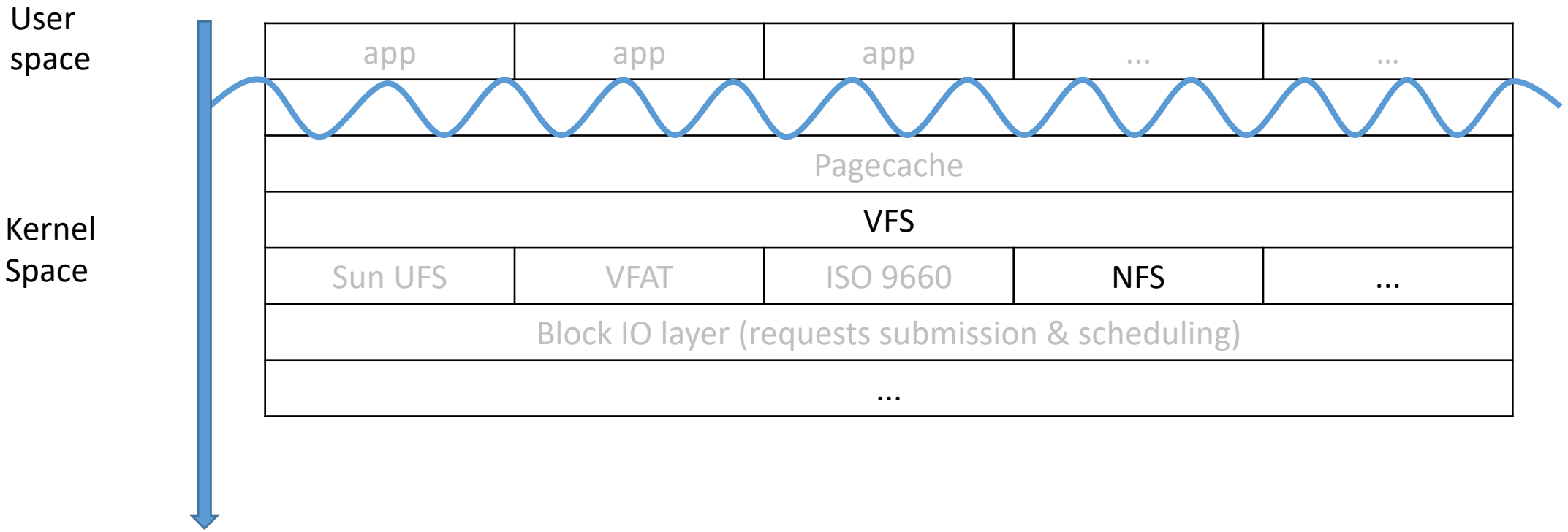
Основные цели

- Доступ к ФС по сети должен быть таким же, как доступ к локальной ФС, притом даже на уровне ядра.



Основные цели

- Доступ к ФС по сети должен быть таким же, как доступ к локальной ФС, притом даже на уровне ядра: **NFS должен предоставлять те же колбеки для VFS, что и локальные файловые системы**.



POSIX API vs. колбеки для VFS

POSIX API	VFS
Вызовы, которые работают с путями, обрабатывают путь целиком: • `open(“./dev/pstorage-fes/main.c”, O_RDONLY)`	

POSIX API vs. колбеки для VFS

POSIX API	VFS
Вызовы, которые работают с путями, обрабатывают путь целиком: • `open(“./dev/pstorage-fes/main.c”, O_RDONLY)`	Ядро проходит по пути по одной компоненте за итерацию: • `t0 = openat(AT_FDCWD, “.”),` • `t1 = openat(t0, “dev”),` • `t2 = openat(t1, “pstorage-fes”),` • `t3 = openat(t2, “main.c”).` Зачем это надо?

POSIX API vs. колбеки для VFS

POSIX API	VFS
Вызовы, которые работают с путями, обрабатывают путь целиком: • `open(“./dev/pstorage-fes/main.c”, O_RDONLY)`	Ядро проходит по пути по одной компоненте за итерацию: • `t0 = openat(AT_FDCWD, “.”),` • `t1 = openat(t0, “dev”),` • `t2 = openat(t1, “pstorage-fes”),` • `t3 = openat(t2, “main.c”).` Зачем это надо? На каждом шаге ядро должно проверить, что ему встретилось, и соответствующим образом обработать следующие случаи: • каталог, • символическая ссылка, • точка монтирования.

POSIX API vs. колбеки для VFS

POSIX API	VFS
Вызовы, которые работают с путями, обрабатывают путь целиком: • `open(“./dev/pstorage-fes/main.c”, O_RDONLY)`	Ядро проходит по пути по одной компоненте за итерацию: • `t0 = openat(AT_FDCWD, “.”),` • `t1 = openat(t0, “dev”),` • `t2 = openat(t1, “pstorage-fes”),` • `t3 = openat(t2, “main.c”).` Зачем это надо? На каждом шаге ядро должно проверить, что ему встретилось, и соответствующим образом обработать следующие случаи: • каталог, • символическая ссылка, • точка монтирования.

Вопрос: какой каталог откроет вызов open(**“./dev/pstorage-fes/..”**, O_DIRECTORY), когда
- ”dev” и “pstorage-fes” – это каталоги,
- “dev” – это каталог, а “pstorage-fes” – символическая ссылка?

POSIX API vs. колбеки для VFS

Клиент	Сервер
/	/
|- /home	|- /home
| |	| |
| |- /artem	| |- /artem
| |	| |
| |- hello.txt	| |- hello.txt
|	|
|- /mnt <--- сюда смонтирован /exports	|- /exports <--- смонтирован в /mnt на клиенте
	|
	|- /home
	|
	|- /artem
	|
	|- hello.txt
	|- symlink <-- указывает на “/home/artem/hello.txt”

Который из файлов hello.txt прочтёт следующая команда на клиенте:

```
# cat /mnt/home/artem/symlink
```

Протокол NFSv2

- Запросы, связанные с проходом по пути:
  - lookup(dirfh, name) -> (fh, attr),
  - getattr(dirfh) -> attr,
  - readdir(dirfh, cookie, count) -> [direntry],

Аргументы fh (File Handle) можно представлять себе как файловые дескрипторы.

Attr – это аналог struct stat. Он описывает свойства файла или каталога (тип, размер, владелец, права доступа, etc.)

Протокол NFSv2

- Запросы, связанные с проходом по пути:
  - lookup(dirfh, name) -> (fh, attr),
  - getattr(dirfh) -> attr,
  - readdir(dirfh, cookie, count) -> [direntry],

- Работа с каталогами и символическими ссылками:
  - mkdir(dirfh, name, attr) -> (fh, newattr),
  - rmdir(dirfh, name) -> status,
  - symlink(dirfh, name, string) -> status,
  - readlink(fh) -> string,
  - link(dirfh, name, tofh, toname) -> status,
  - rename(dirfh, name, tofh, toname) -> status,

Протокол NFSv2

- Запросы, связанные с проходом по пути:
  - lookup(dirfh, name) -> (fh, attr),
  - getattr(dirfh) -> attr,
  - readdir(dirfh, cookie, count) -> [direntry],

- Работа с каталогами и символическими ссылками:
  - mkdir(dirfh, name, attr) -> (fh, newattr),
  - rmdir(dirfh, name) -> status,
  - symlink(dirfh, name, string) -> status,
  - readlink(fh) -> string,
  - link(dirfh, name, tofh, toname) -> status,
  - rename(dirfh, name, tofh, toname) -> status,

- Создание/удаление файлов:
  - create(dirfh, name, attr, flags) -> (newfh, attr),
  - remove(dirfh, name) -> status,

Протокол NFSv2

- Запросы, связанные с проходом по пути:
  - lookup(dirfh, name) -> (fh, attr),
  - getattr(dirfh) -> attr,
  - readdir(dirfh, cookie, count) -> [direntry],

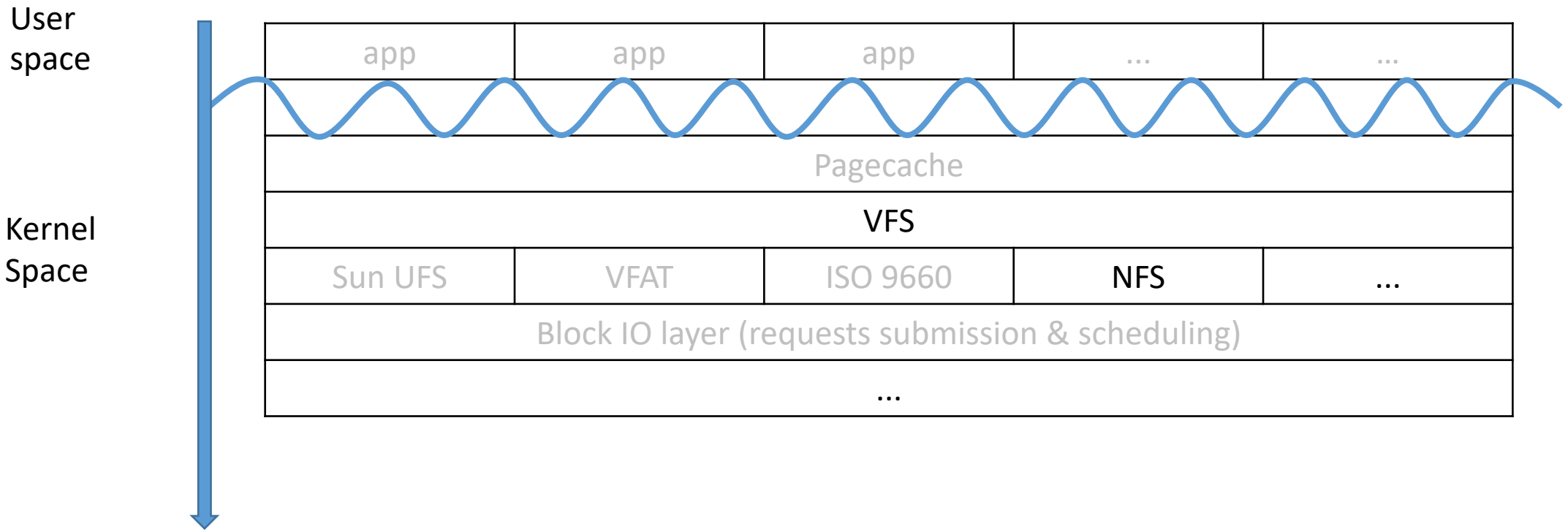
- Работа с каталогами и символическими ссылками:
  - mkdir(dirfh, name, attr) -> (fh, newattr),
  - rmdir(dirfh, name) -> status,
  - symlink(dirfh, name, string) -> status,
  - readlink(fh) -> string,
  - link(dirfh, name, tofh, toname) -> status,
  - rename(dirfh, name, tofh, toname) -> status,

- Создание/удаление файлов:
  - create(dirfh, name, attr, flags) -> (newfh, attr),
  - remove(dirfh, name) -> status,

- Файловый ввод-вывод:
  - read(fh, offset, count) -> (attr, data),
  - write(fh, offset, count, data) -> attr.

Основные цели

- Доступ к ФС по сети должен быть таким же, как доступ к локальной ФС, притом даже на уровне ядра: **NFS должен предоставлять те же колбеки для VFS, что и локальные файловые системы**.



Основные цели

- Доступ к ФС по сети должен быть таким же, как доступ к локальной ФС, притом даже на уровне ядра:
  NFS должен предоставлять те же колбеки для VFS, что и локальные файловые системы.
- NFS-сервер может перезагрузиться в любой момент времени, притом ядро на клиенте должно быть в состоянии продолжить использовать те же file handles.

```
• t0 = openat(AT_FDCWD, “.”),
• t1 = openat(t0, “dev”),
• t2 = openat(t1, “pstorage-fes”),
• t3 = openat(t2, “main.c”).
```

Сервер перезапустился, но t2 должен остаться валидным

Основные цели

- Доступ к ФС по сети должен быть таким же, как доступ к локальной ФС, притом даже на уровне ядра:
  NFS должен предоставлять те же колбеки для VFS, что и локальные файловые системы.

- NFS-сервер может перезагрузиться в любой момент времени, притом ядро на клиенте должно быть в состоянии продолжить использовать те же file handles.:
  File handles должны переживать остановку/запуск NFS-сервера.
  Многие операции вроде create() можно подтверждать клиенту только после syncfs().

- `t0 = openat(AT_FDCWD, “.”),`
- `t1 = openat(t0, “dev”),`
- `t2 = openat(t1, “pstorage-fes”),`
- `t3 = openat(t2, “main.c”).`

Сервер перезапустился, но t2 должен остаться валидным

Stateless server

File handles должны переживать остановку/запуск NFS-сервера.

Один из способов сделать так, чтобы состояние сервера сохранялось при перезапуске – это не хранить никакого состояния.

Stateless server

File handles должны переживать остановку/запуск NFS-сервера.

Один из способов сделать так, чтобы состояние сервера сохранялось при перезапуске – это не хранить никакого состояния.

Семантика Unix File System позволяет этого добиться: в качестве file handle достаточно использовать номер иноды*.

** Как быть со сценарием удаления/создания файла, которые переиспользуют номер иноды?*

Stateless server

File handles должны переживать остановку/запуск NFS-сервера.

Один из способов сделать так, чтобы состояние сервера сохранялось при перезапуске – это не хранить никакого состояния.

Семантика Unix File System позволяет этого добиться: в качестве file handle достаточно использовать номер иноды*.

Проблема: доступ к файлам по номеру иноды напрямую доступен только ядру и изначально NFS-сервер в Linux был реализован в ядре. Можно ли сделать NFS-сервер пользовательским приложением?

** Как быть со сценарием удаления/создания файла, которые переиспользуют номер иноды?*

Stateless server

File handles должны переживать остановку/запуск NFS-сервера.

Один из способов сделать так, чтобы состояние сервера сохранялось при перезапуске – это не хранить никакого состояния.

Семантика Unix File System позволяет этого добиться: в качестве file handle достаточно использовать номер иноды*.

Проблема: доступ к файлам по номеру иноды напрямую доступен только ядру и изначально NFS-сервер в Linux был реализован в ядре. Можно ли сделать NFS-сервер пользовательским приложением?

- man name_to_handle_at
- man open_by_handle_at

** Как быть со сценарием удаления/создания файла, которые переиспользуют номер иноды?*

Stateless server

Многие операции вроде create() можно подтверждать клиенту только после syncfs().

Для редких операций это нестрашно. Но делать fsync() после каждого write() непозволительно дорого.

Stateless server

Многие операции вроде create() можно подтверждать клиенту только после syncfs().

Для редких операций это нестрашно. Но делать fsync() после каждого write() непозволительно дорого.

Решение в NFSv3:

- при запуске NFS-сервера генерируется случайное число,
- это число сообщается клиенту в ответ на запрос write(),
- добавляется операция commit() – аналог fsync(), которому клиент передаёт число-ответ от write(). Если между write() и commit() был перезапуск NFS-сервера, то числа на клиенте и на сервере не совпадут, и клиент может узнать, что ему надо перепослать все неподтверждённые запросы write().

Stateless server, проблемы с POSIX-семантикой

POSIX разрешает файлы без имени: они существуют, пока у них имеется открытый файловый дескриптор.

Stateless server, проблемы с POSIX-семантикой

POSIX разрешает файлы без имени: они существуют, пока у них имеется открытый файловый дескриптор.

Проблема: при перезагрузке NFS-сервера такие файловые дескрипторы исчезнут и безымянные файлы будут удалены.

Stateless server, проблемы с POSIX-семантикой

POSIX разрешает файлы без имени: они существуют, пока у них имеется открытый файловый дескриптор.

Проблема: при перезагрузке NFS-сервера такие файловые дескрипторы исчезнут и безымянные файлы будут удалены.

Она решается на клиентской стороне (silly rename):

- при вызове unlink() на открытый файл NFS-клиент переименовывает файл в имя вида “.nfs_случайная_строчка”,
- клиент удаляет эти файлы, когда закрывает все файловые дескрипторы к ним.

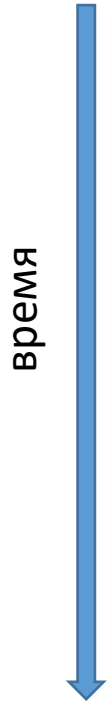
Если клиент аварийно перезагружается или теряет связь с NFS-сервером, то за ним остаётся мусор.

Проблемы с неидемпотентными запросами

Клиент	Сеть	Сервер
rmdir(“/path/to/dir”)	req ---->	

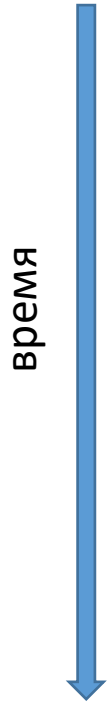
Проблемы с неидемпотентными запросами

Клиент	Сеть	Сервер
rmdir(“/path/to/dir”)	req ---->	
		rmdir(“/prefix/path/to/dir”)
	<---- resp(ok)	



Проблемы с неидемпотентными запросами

Клиент	Сеть	Сервер
rmdir(“/path/to/dir”)	req ---->	
		rmdir(“/prefix/path/to/dir”)
	<---- resp(ok)	
	connection lost	



Проблемы с неидемпотентными запросами

Клиент	Сеть	Сервер
rmdir(“/path/to/dir”)	req ---->	
		rmdir(“/prefix/path/to/dir”)
	<---- resp(ok)	
	connection lost	
timeout		

время

Проблемы с неидемпотентными запросами

Клиент	Сеть	Сервер
rmdir(“/path/to/dir”)	req ---->	
		rmdir(“/prefix/path/to/dir”)
	<---- resp(ok)	
	connection lost	
timeout		
retry rmdir(“/path/to/dir”)	---->	
		rmdir(“/prefix/path/to/dir”)

время

Проблемы с неидемпотентными запросами

Клиент	Сеть	Сервер
rmdir(“/path/to/dir”)	req ---->	
		rmdir(“/prefix/path/to/dir”)
	<---- resp(ok)	
	connection lost	
timeout		
retry rmdir(“/path/to/dir”)	---->	
		rmdir(“/prefix/path/to/dir”)
	<---- resp(not found)	

время

Проблемы с сетью в общем

С какой скоростью выполняется операция lookup() локально
и на NFS?

Проблемы с сетью в общем

С какой скоростью выполняется операция lookup() локально и на NFS?

lookup() для NFS не меньше, чем RTT между клиентом и сервером.

Проблемы с сетью в общем

С какой скоростью выполняется операция lookup() локально и на NFS?

lookup() для NFS не меньше, чем RTT между клиентом и сервером.

Для сравнения:

- прочесть 6KB из 2ch DDR4-памяти @ 3.6Ghz – 1μs,
- RTT между машинами, подключенными в один 10G ethernet-свитч – 50μs,
- RTT между машинами в проводном сегменте сети московского офиса Акронис -- 700μs,
- RTT между Москвой и Кёльном – около 50ms.

Проблемы с сетью в общем

С какой скоростью выполняется операция lookup() локально и на NFS?

lookup() для NFS не меньше, чем RTT между клиентом и сервером.

Для сравнения:

- прочесть 6KB из 2ch DDR4-памяти @ 3.6Ghz – 1μs,
- RTT между машинами, подключенными в один 10G ethernet-свитч – 50μs,
- RTT между машинами в проводном сегменте сети московского офиса Акронис -- 700μs,
- RTT между Москвой и Кёльном – около 50ms.

sunrpc portmap service использует UDP и его ответ может быть на порядок длиннее запроса.

Проблемы с сетью в общем

С какой скоростью выполняется операция lookup() локально и на NFS?	lookup() для NFS не меньше, чем RTT между клиентом и сервером. Для сравнения: • прочесть 6KB из 2ch DDR4-памяти @ 3.6Ghz – 1μs, • RTT между машинами, подключенными в один 10G ethernet-свитч – 50μs, • RTT между машинами в проводном сегменте сети московского офиса Акронис -- 700μs, • RTT между Москвой и Кёльном – около 50ms.
sunrpc portmap service использует UDP и его ответ может быть на порядок длиннее запроса.	Он может использоваться для “traffic amplification” атак.

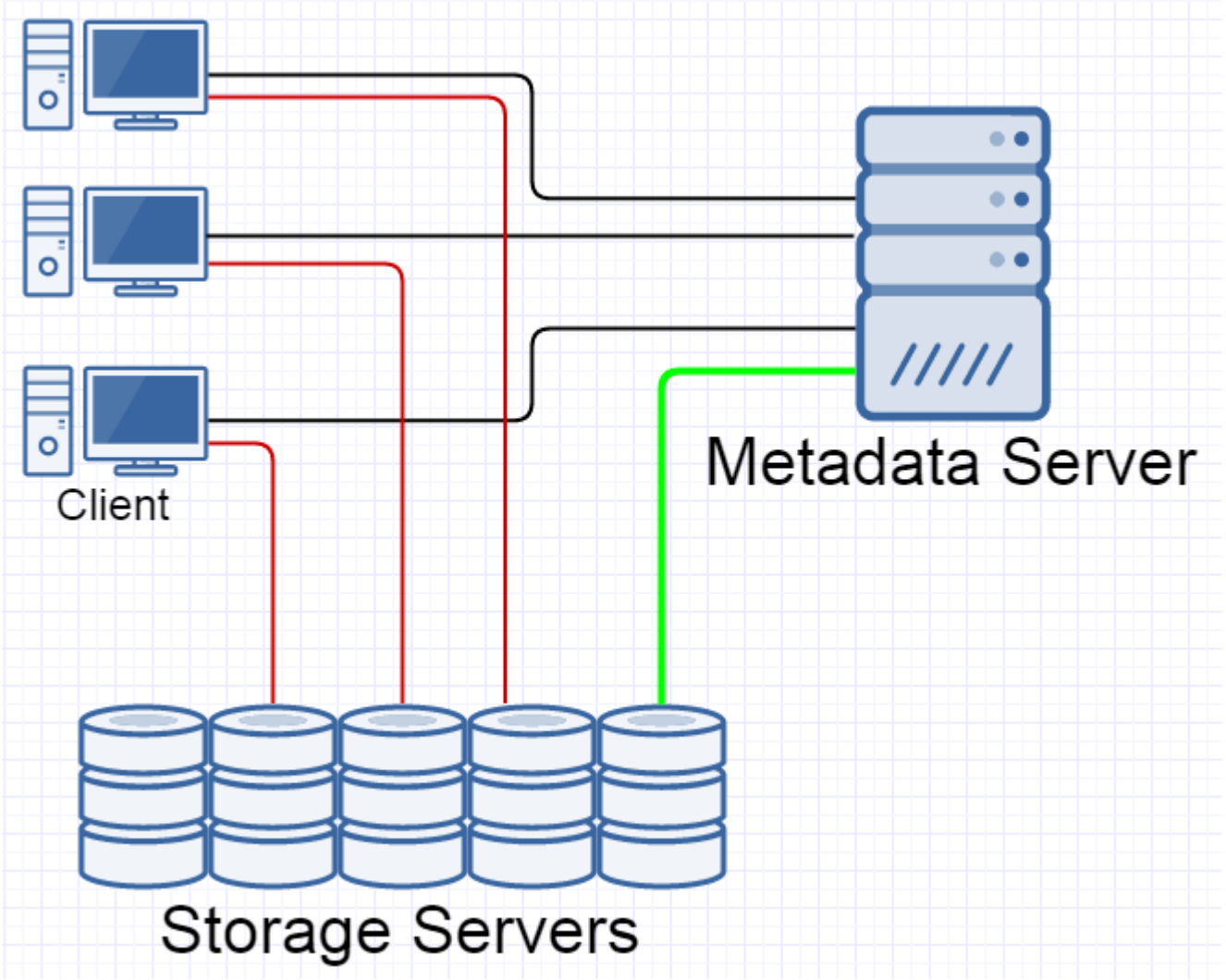
Одна точка отказа и узкое место

NFS-сервер оказывается узким местом, поскольку ресурсы одной машины приходится делить между многими клиентами.

Одна точка отказа и узкое место

NFS-сервер оказывается узким местом, поскольку ресурсы одной машины приходится делить между многими клиентами.

Parallel NFS в NFSv4.1:



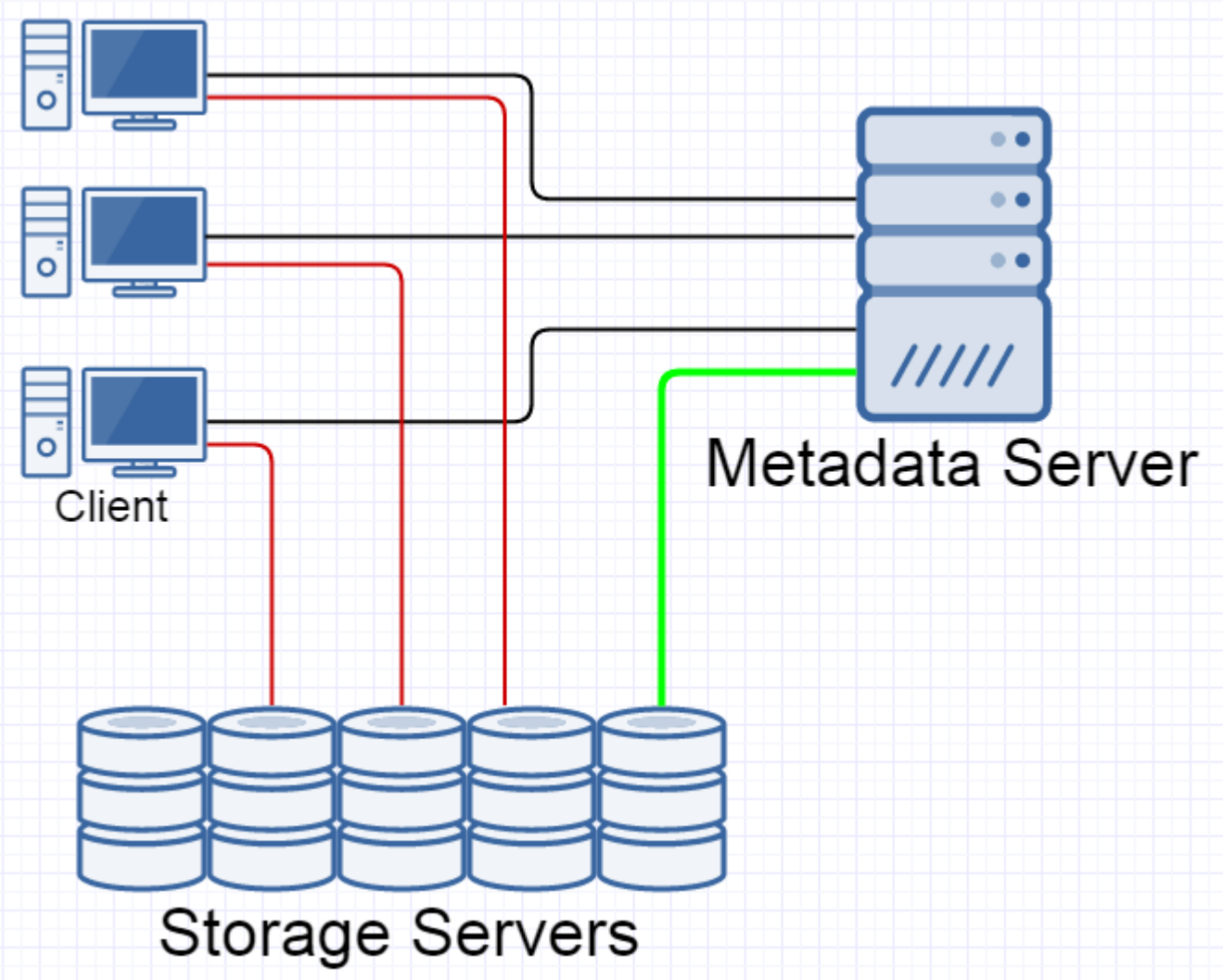
Одна точка отказа и узкое место

NFS-сервер оказывается узким местом, поскольку ресурсы одной машины приходится делить между многими клиентами.

Parallel NFS в NFSv4.1:

Привносит свою долю интересных сценариев:

- Что будет, если клиент потеряет связь с metadata server'ом, но сохранит соединения к storage-серверам?



The Fallacies of Networked Computing

1. Сеть надёжно доставляет сообщения,
2. Задержки в сети нулевые,
3. Пропускная способность не ограничена,
4. Сеть безопасна.

The Fallacies of Networked Computing

1. Сеть надёжно доставляет сообщения,
2. Задержки в сети нулевые,
3. Пропускная способность не ограничена,
4. Сеть безопасна.

По сравнению с приложением, работающем в одном процессе, вызов функции приобретает новый неудачный сценарий: вызов функции что-то сделал, но мы об этом не узнали.

Протоколы клиент-серверного взаимодействия должны быть спроектированы с учётом того, что каждый запрос может быть не доставлен или доставлен несколько раз.

Порвавшееся соединение – норма, а не исключение из правил.

The Fallacies of Networked Computing

1. Сеть надёжно доставляет сообщения,
2. Задержки в сети нулевые,
3. Пропускная способность не ограничена,
4. Сеть безопасна.

Вызовы в RPC качественно отличаются от вызовов функций. Call + сохранение/восстановление регистров – это единицы или десятки наносекунд. RPC-вызов в 10G ethernet сети – десятки микросекунд только на RTT в сети.

The Fallacies of Networked Computing

1. Сеть надёжно доставляет сообщения,
2. Задержки в сети нулевые,
3. Пропускная способность не ограничена,
4. Сеть безопасна.

Вернуть сколь угодно большой объект из локальной функции стоит столько же, сколько вернуть указатель.

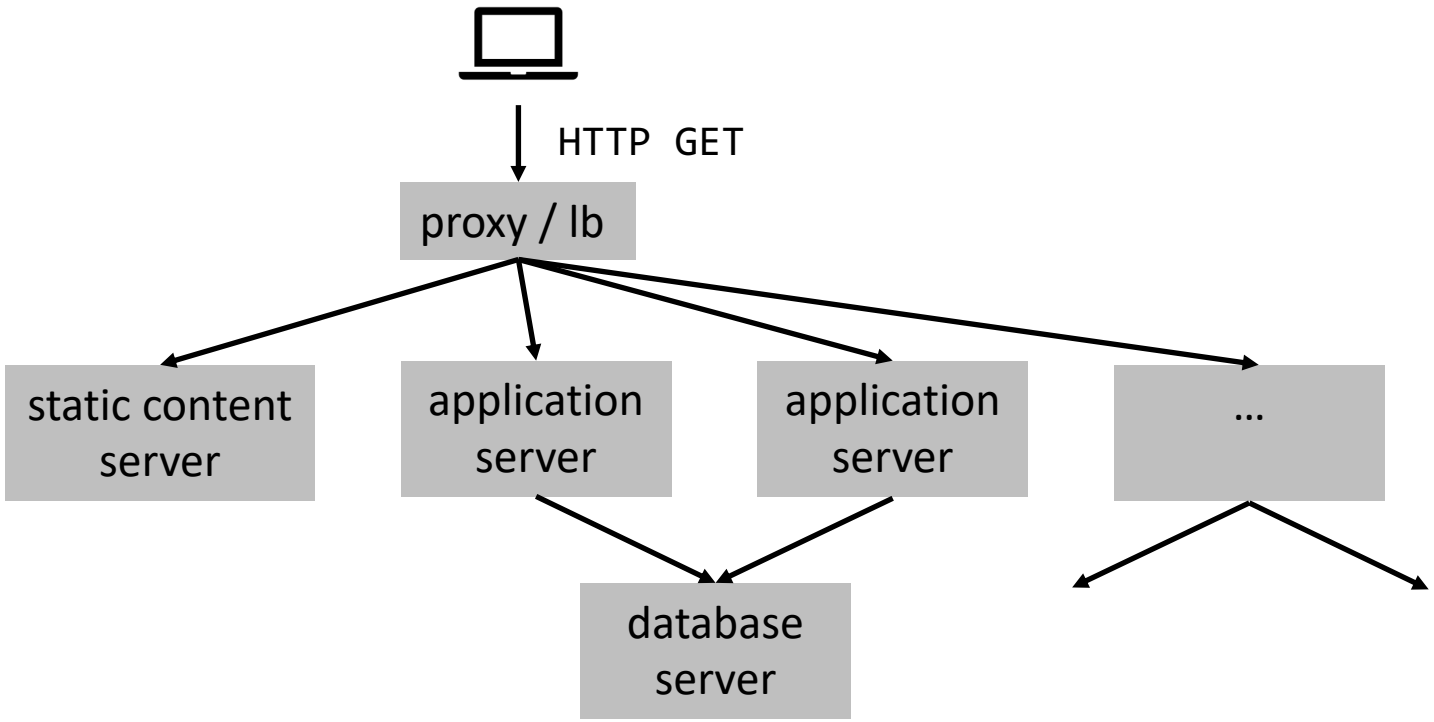
Для удалённых вызовов необходимо учитывать время на передачу ответа. Для Web-сервисов, зачастую, ещё необходимо учитывать время на сериализацию и десериализацию XML, JSON и прочего.

Ещё несколько трудностей с сетью

Зачастую сервер, принявший запрос клиента, разбивает его на подзапросы и передаёт дальше:

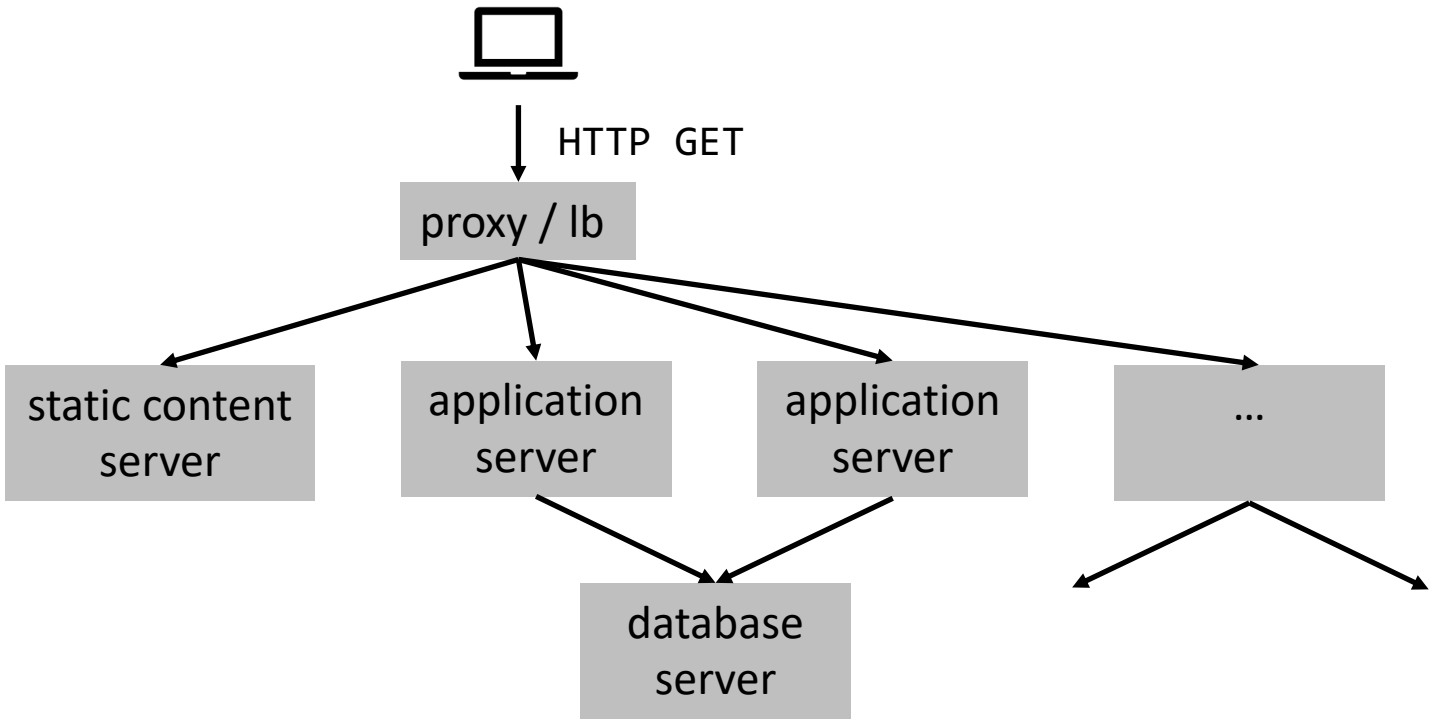
Ещё несколько трудностей с сетью

Зачастую сервер, принявший запрос клиента, разбивает его на подзапросы и передаёт дальше:



Ещё несколько трудностей с сетью

Зачастую сервер, принявший запрос клиента, разбивает его на подзапросы и передаёт дальше:

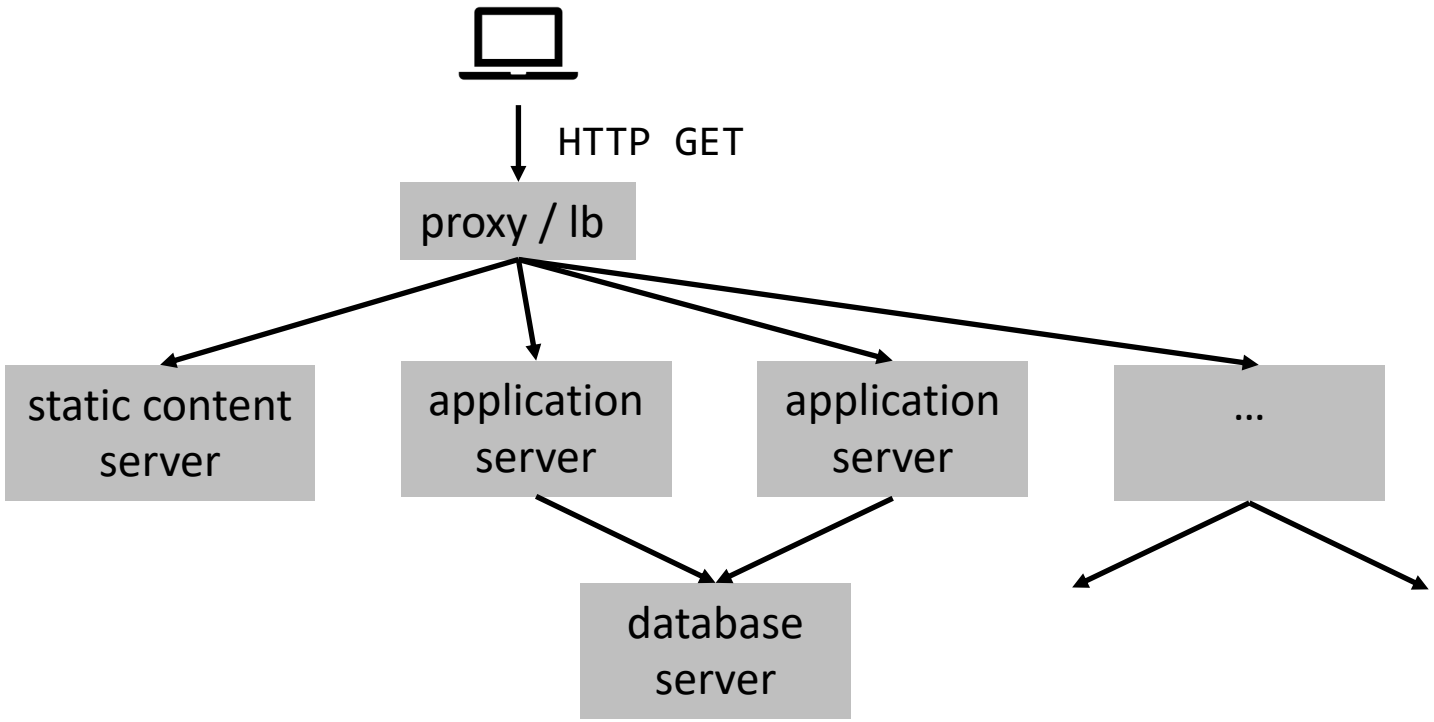


1. Какое распределение задержек будет у сервиса, который для построения ответа должен опросить 10 серверов, каждый из которых гарантирует 1ms задержки в 99-м персентиле, но 1с задержки в общем случае?

   Как контролировать хвосты в распределении задержек?

Ещё несколько трудностей с сетью

Зачастую сервер, принявший запрос клиента, разбивает его на подзапросы и передаёт дальше:



1. Какое распределение задержек будет у сервиса, который для построения ответа должен опросить 10 серверов, каждый из которых гарантирует 1ms задержки в 99-м персентиле, но 1с задержки в общем случае?

   Как контролировать хвосты в распределении задержек?

2. Как исследовать путь обработки одного клиентского запроса? Как собрать логи с отдельных серверов в общую картину?

Логи в распределённой системе и задача о днях рождения

Как собрать логи с отдельных серверов в общую картину?

Логи в распределённой системе и задача о днях рождения

Как собрать логи с отдельных серверов в общую картину? – Присвоить каждому запросу уникальный идентификатор, который передавать в каждый подзапрос. Лог обработки запроса получится как объединение логов подзапросов с общим идентификатором запроса.

Логи в распределённой системе и задача о днях рождения

Как собрать логи с отдельных серверов в общую картину? – Присвоить каждому запросу уникальный идентификатор, который передавать в каждый подзапрос. Лог обработки запроса получится как объединение логов подзапросов с общим идентификатором запроса.

Как эффективно генерировать уникальные идентификаторы в распределённой системе? Этот процесс не должен требовать синхронизации между узлами системы.

Логи в распределённой системе и задача о днях рождения

Как собрать логи с отдельных серверов в общую картину? – Присвоить каждому запросу уникальный идентификатор, который передавать в каждый подзапрос. Лог обработки запроса получится как объединение логов подзапросов с общим идентификатором запроса.

Как эффективно генерировать уникальные идентификаторы в распределённой системе? Этот процесс не должен требовать синхронизации между узлами системы.

Можно ли выбирать идентификаторы случайно?

Логи в распределённой системе и задача о днях рождения

Как собрать логи с отдельных серверов в общую картину? – Присвоить каждому запросу уникальный идентификатор, который передавать в каждый подзапрос. Лог обработки запроса получится как объединение логов подзапросов с общим идентификатором запроса.

Как эффективно генерировать уникальные идентификаторы в распределённой системе? Этот процесс не должен требовать синхронизации между узлами системы.

Можно ли выбирать идентификаторы случайно?

Задача о днях рождения: пусть даны n значений случайной целочисленной величины, равномерно распределённой на интервале [1, d]. Чему равна вероятность p(n, d) того, что среди этих чисел есть хотя бы два совпадающих?

Логи в распределённой системе и задача о днях рождения

Как собрать логи с отдельных серверов в общую картину? – Присвоить каждому запросу уникальный идентификатор, который передавать в каждый подзапрос. Лог обработки запроса получится как объединение логов подзапросов с общим идентификатором запроса.

Как эффективно генерировать уникальные идентификаторы в распределённой системе? Этот процесс не должен требовать синхронизации между узлами системы.

Можно ли выбирать идентификаторы случайно?

Задача о днях рождения: пусть даны n значений случайной целочисленной величины, равномерно распределённой на интервале [1, d]. Чему равна вероятность $p(n, d)$ того, что среди этих чисел есть хотя бы два совпадающих?

Ответ:

$$p(n,d) = f(x) = \begin{cases} 1 - \prod_{k=1}^{n-1}\left(1 - \frac{k}{d}\right), & n \le d \\ 1, & n > d \end{cases}$$

Для $n \ll d$ с хорошей точностью выполняется

$$p(n,d) \approx 1 - \exp\left(-\frac{n(n-1)}{2d}\right)$$

Логи в распределённой системе и задача о днях рождения

Как собрать логи с отдельных серверов в общую картину? – Присвоить каждому запросу уникальный идентификатор, который передавать в каждый подзапрос. Лог обработки запроса получится как объединение логов подзапросов с общим идентификатором запроса.

Как эффективно генерировать уникальные идентификаторы в распределённой системе? Этот процесс не должен требовать синхронизации между узлами системы.

Можно ли выбирать идентификаторы случайно?

Если сгенерировать 2^{32} случайных 128-битных идентификаторов, то вероятность получить совпадения будет

$$p(2^{32}, 2^{128}) \approx 1 - \exp\left(-\frac{2^{32} \cdot 2^{32}}{2 \cdot 2^{128}}\right) =$$

$$= 1 - \exp(-2^{-65}) \approx 2^{-65}$$

Задача о днях рождения: пусть даны n значений случайной целочисленной величины, равномерно распределённой на интервале [1, d]. Чему равна вероятность p(n, d) того, что среди этих чисел есть хотя бы два совпадающих?

Ответ:

$$p(n,d) = f(x) = \begin{cases} 1 - \prod_{k=1}^{n-1}\left(1 - \frac{k}{d}\right), & n \le d \\ 1, & n > d \end{cases}$$

Для $n \ll d$ с хорошей точностью выполняется

$$p(n,d) \approx 1 - \exp\left(-\frac{n(n-1)}{2d}\right)$$

Домашнее задание

1. Напишите gRPC-сервер (https://grpc.io), который в ответ на каждый запрос генерирует случайную строчку длиной 128 байт. Вставьте случайные паузы так, чтобы 99 из 100 запросов обрабатывались очень быстро, а 1 запрос из 100 занимал 1с.
2. Напишите gRPC-сервер, который в ответ на каждый запрос делает 16 запросов к разным экземплярам сервиса из задачи №1 и в качестве ответа возвращает объединение строк от подзапросов. Подзапросы должны исполняться параллельно.
3. (*) Установите Prometheus (https://prometheus.io). Добавьте в сервисы из задач №1 и №2 мониторинг задержек, который отправляет данные в Prometheus. Установите Grafana (https://grafana.com) и создайте дешборд с графиком 90-го, 95-го и 99-го персентилей задержек в сервисе из задачи №2 и каждого из 16 экземпляров сервиса из задачи №1.
4. (*) Установите Zipkin (https://zipkin.io). Добавьте в сервисы №1 и №2 поддержку opentracing (https://opentracing.io) и подключите их к своему экземпляру Zipkin.