НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Современные компьютерные технологии НИЧ НГУ
НОВОСИБИРСК 2014

Разработка модуля синхронизации метаданных для распределенного сетевого хранилища

Выполнила:
Маллаева Ю.М.

Научный руководитель:
Мигинский Д.С.
Иртегов Д.В.

Введение в проблему

Веб-кластера с балансировкой нагрузки:

- Клиентские приложения хранятся на локальных дисках.
  - Перенос файлов между локальными дисками серверов кластера трудноосуществим.

- Клиентские приложения находятся на сетевом хранилище.
  - Большие задержки при открытии файлов на сетевых файловых системах. (NFS, SMB)
  - Невозможность параллельного доступа на запись для сетей доступа к дискам. (iSCSI, AoE, Parallels Cloud Storage)

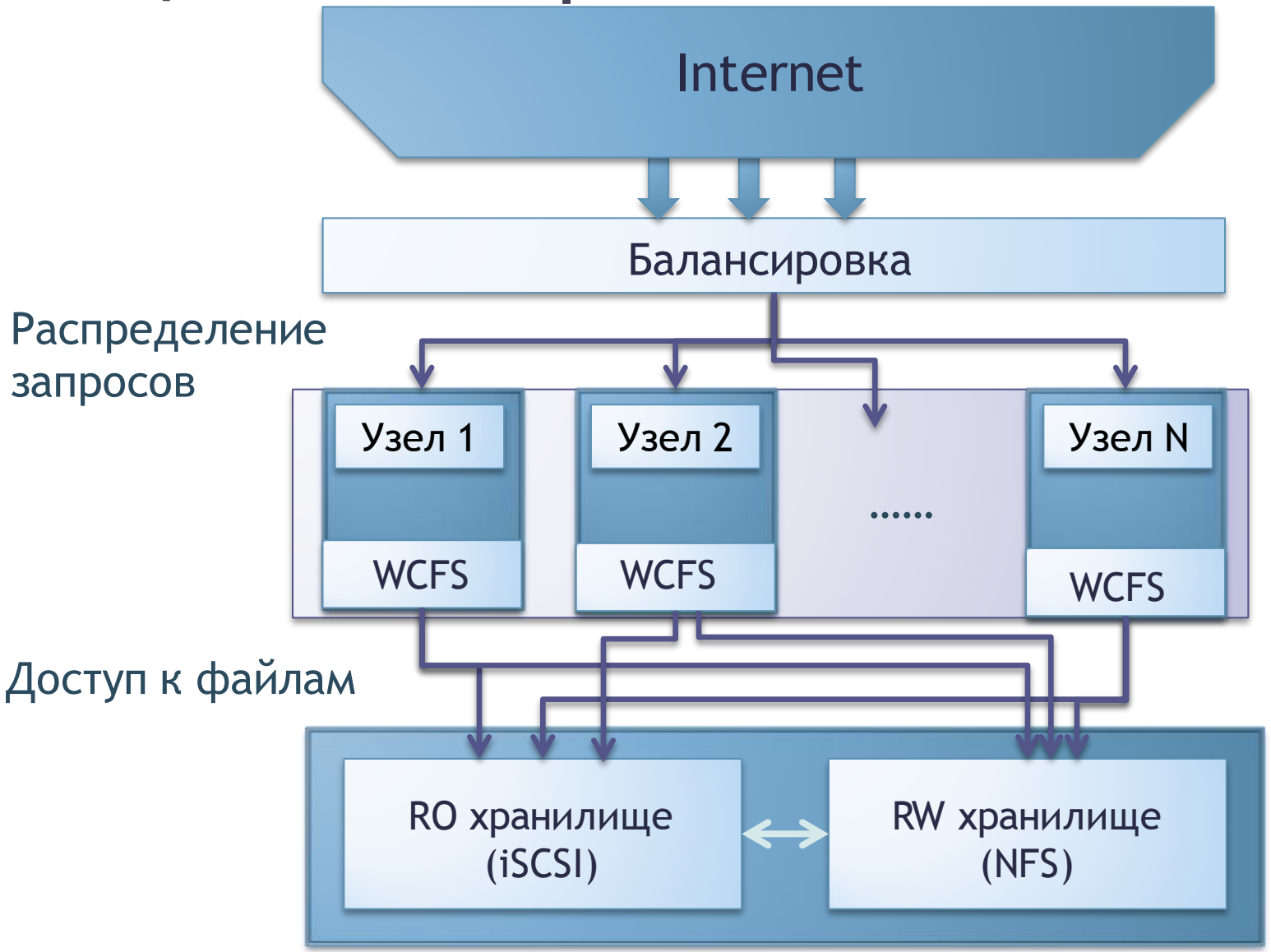
Особенности использования ФС

- При открытии каждой страницы сервер открывает от десятков до сотен файлов.
- Большая часть из них открывается на чтение (read-only)(сценарии, картинки, шаблоны).
- Есть способы снизить задержки открытия файлов при работе с read-only хранилищем.
- Сделать разделяемое хранилище полностью read-only нельзя:
  - Обновления сайта
  - Кэши, сессия, пользовательский контент

Предлагаемое решение

- Хранилище с каскадно-объединенным монтированием двух файловых систем (примеры в Linux: Union FS, AUFS)
- Доступ к RO файлам осуществляется по iSCSI, что обеспечивает малую задержку открытия за счет агрессивного кэширования
- Доступ к RW файлам осуществляется по NFS

Общая схема работы WCFS



Постановка задачи

- Проект заключается в разработке гибридного хранилища:
  - Доступ на чтение (RO) - с сетевого диска
  - Параллельный доступ на чтение-запись (RW) - с сетевой ФС

- Чтобы знать, с какой из веток хранилища (RO/RW) нужно брать каждый конкретный файл, нужны метаданные.
- Нужна синхронизация метаданных между узлами кластера.

Цель работы:

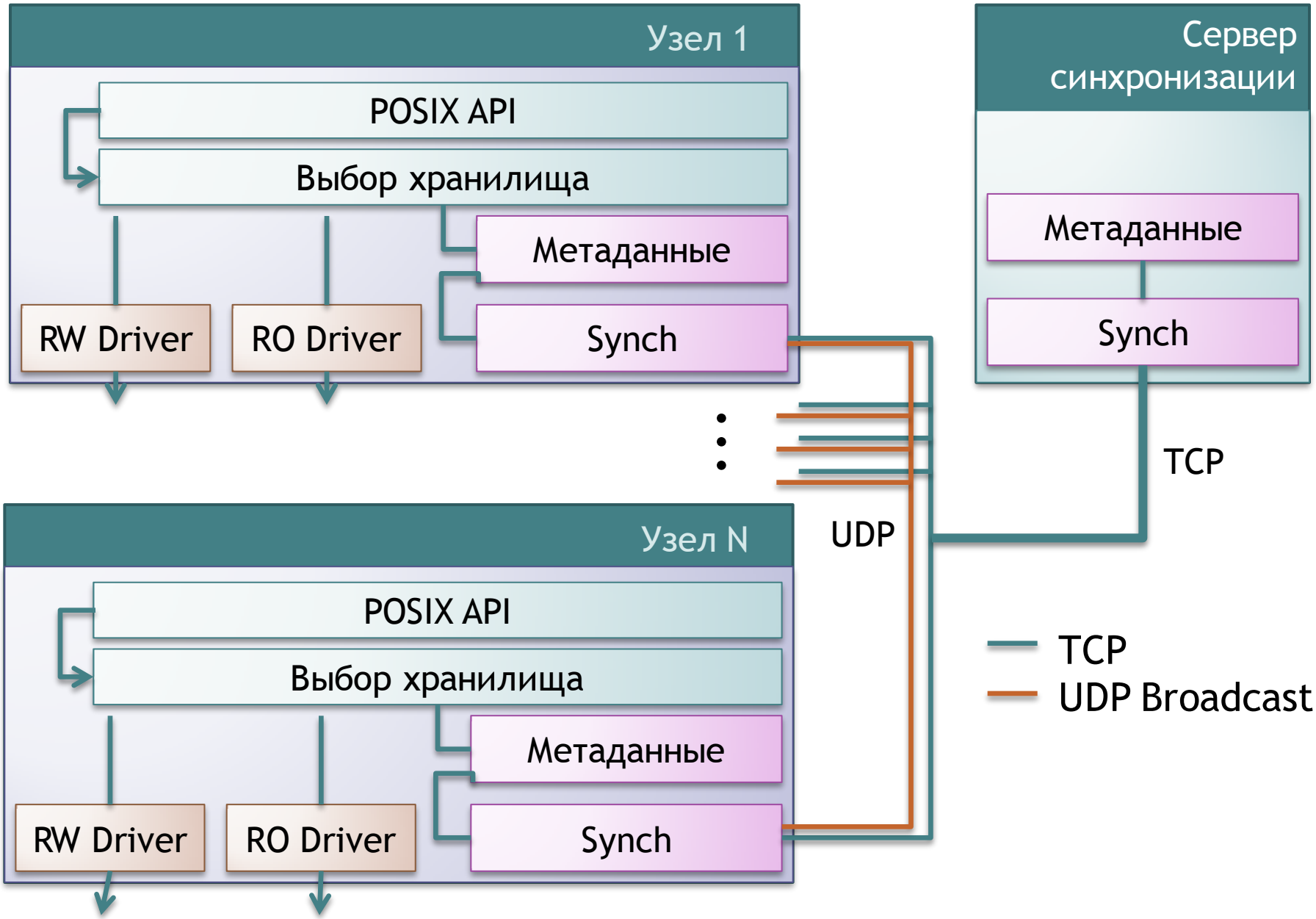
Разработка и реализация модуля синхронизации метаданных для распределенного сетевого хранилища Web Cluster File System (WCFS).

Метаданные

- Существующие каскадно-объединенные ФС (UnionFS, aufs) сначала ищут файл в RW ветке, потом в RO
  - Задержки определяются задержками операции lookup на RW ветке, в нашем случае – NFS
- Чтобы избежать этого, мы храним таблицу трансляции: список файлов, размещенных на RW
- Эта таблица обновляется при каждом
  - Первом открытии файла на RW
  - Создании файла
  - Удалении файла

Модуль синхронизации: требования

- Синхронизация метаданных между узлами кластера
- Восстановление метаданных на узле кластера после его перезагрузки (плановой или аварийной)
- Минимизация средних задержек синхронизации
- Корректная работа при перегрузке сети или отдельных узлов


Узел 1
POSIX API
Выбор хранилища
Метаданные
RW Driver
RO Driver
Synch
Сервер синхронизации
Метаданные
Synch
TCP
UDP
Узел N
POSIX API
Выбор хранилища
Метаданные
RW Driver
RO Driver
Synch
TCP
UDP Broadcast

Возможные подходы

- Драйвер WCFS – это модуль ядра Linux
- Таблица трансляции хранится в ядре

- Модуль синхронизации можно реализовать
  - Как модуль ядра
  - Как процесс в userspace

Процесс в userspace

- Использовался в прототипе первого этапа
  - Драйвер WCFS имеет дополнительный интерфейс: символьное устройство, через которое связывается с модулем синхронизации
  - Через это устройство передаются обновления таблицы в обе стороны

- Как минимум, одна лишняя сущность (символьное устройство)
- Копирование kernel space <-> userspace
- Дополнительный источник отказов и задержек (нужен контроль за состоянием процесса)

Клиент синхронизации: модуль ядра

- Увеличивается сложность разработки (изучение API, отладка, ошибки приводят к неработоспособности ядра)

- Производительность:
  - Приоритет ядра
  - Нет избыточного копирования
- Легкость сопровождения
  - Относительная простота управления (взаимодействие только с драйвером WCFS)
  - Относительная простота структуры и логики работы системы (нет доп. компонентов)

Проделанная работа:

- Реализован прототип клиента синхронизации в виде userspace-утилиты
- Реализован модуль клиента синхронизации в виде модуля ядра Linux, работающего в составе драйвера хранилища.
- Осуществлена миграция клиента синхронизации на ядро 2.6.32 для использования в CentOS 6.5

Kernel client: реализация

- Kernel thread для каждого запущенного клиента (каждого монтирования WCFS)
- Поток ожидает определенных событий:
  - Изменение метаданных на стороне клиента
  - Получение метаданных
    - от сервера синхронизации
    - От других узлов

- Проблема: нужно опрашивать несколько сокетов, но нет возможности использовать системный вызов select({sock1,sock2,…})

Socket API

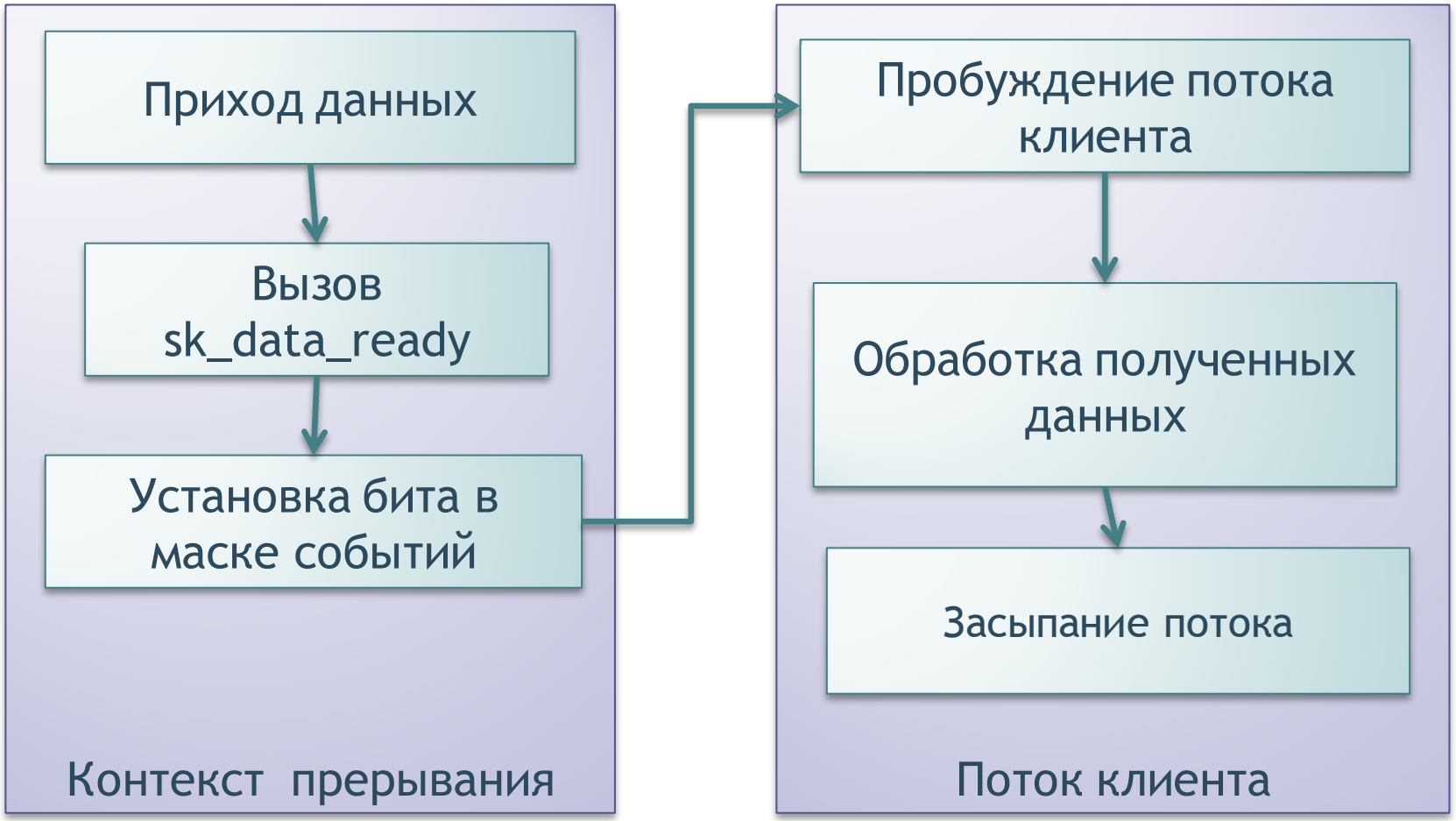
```
struct sock {
[…]
void (*sk_data_ready)(struct sock *sk, int bytes);
void (*sk_write_space)(struct sock *sk);
[…]
}
```

Решение: переопределить обработчики использующихся сокетов по умолчанию.

sk_data_ready – появление данных в буфере приема,

sk_write_space - освобождение места в буфере отправки.

Пример: получение данных от сервера синхронизации



В дальнейшем

- Доработка и расширение функциональности клиента и сервера
- Интеграция клиента синхронизации с драйвером хранилища
- Разработка набора тестов для проверки корректной работы клиента и сервера синхронизации
- Тестирование (в тестовом окружении и на стенде из нескольких узлов)

Спасибо за внимание.

Работа выполнена при финансовой поддержке Минобрнауки РФ (договор № 02.G25.31.0054).