САНКТ - ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет
Кафедра системного программирования

Колянов Дмитрий

Оптимизация обработки AoE — запросов для встроенных систем с многоядерными процессорами

Выпускная квалификационная работа

Допустить к защите.
Зав. Кафедрой
д. ф.-м. н., профессор …… А. Н. Терехов

Научный руководитель …… С. В. Богатырев

Рецензент …… А. Н. Косякин

Санкт-Петербург
2011

St. Petersburg State University
Faculty of Mathematics and Mechanics
Chair of Software Engineering

Optimization of processing AoE-requests for embedded systems with multicore processors

Graduate paper by

Kolyanov Dmitry

"Admitted to proof"
(Head of the chair, Dr. of Phys. and Math. Sci.)
/Signature/_____/ A.N. Terekhov

Scientific advisor
/Signature/_____/ S. V. Bogatyrev

Reviewer
/Signature/_____/ A. N. Kosyakin

St. Petersburg, 2011

Содержание

1 Введение

Организация систем хранения данных является важной задачей для многих сфер бизнеса, по результатам исследования компании IDC общий объем цифровой информации в мире с каждым годом вырастает приблизительно в полтора раза [1].

Большое значение имеет степень надежности систем хранения информации — потеря данных в ряде случаев может принести большие убытки. Соответственно, значительная часть проблем связана с обеспечением защиты от сбоев: отключение питания, выход определенной части оборудования из строя или обрыв проводов не должны приводить к утрате работоспособности системы в целом.

Важно не только гарантировать сохранность и конфиденциальность данных, но и обеспечивать удобный доступ к ним. В связи с этим возникла идея сетевого хранения, разработаны соответствующие протоколы [2] и решения, которые их поддерживают. Сетевое хранение позволяет экономить на дисковых ресурсах, поскольку одни и те же диски могут использовать несколько клиентов, а так же хранилище гораздо легче масштабировать и обслуживать.

Пользователям можно предоставлять возможность создавать свои виртуальные машины в сети — это позволит значительно уменьшить затраты дисковых и процессорных ресурсов, а также оперативной памяти сервера в сравнении со случаем, когда каждому клиенту выделяется отдельное оборудование. Такой подход называется облачными вычислениями — идея заключается в том, что клиенты редко используют все предоставленные им ресурсы — например, суммарная емкость виртуальных дисков может быть больше, чем суммарная емкость настоящих дисков, что позволит увеличивать объем хранилища только в случае действительной необходимости. Также виртуальные машины могут мигрировать с сервера на сервер, занимая таким образом минимальное число серверов, что приводит к потенциальной экономии электроэнергии.

Для сетевого хранилища важен выбор аппаратуры и протоколов для передачи данных— некоторые решения являются дорогостоящими и требуют специального оборудования — например, в случае использования технологии Fibre Channel предполагается построение отдельной сети, преимущественно на основе оптико-волоконных кабелей, приобретение специализированных коммутаторов, адаптеров для серверов и систем хранения данных с соответствующими внешними интерфейсами [3].

Эта работа является частью проекта по созданию распределенной системы хранения данных, которая будет использоваться совместно с облаком виртуальных машин. Проект ориентирован на малые и средние инфраструктуры, главные требования — эффективное функционирование на стандартных аппаратных средствах, надежность и максимально возможная производительность. Цель работы — распараллелить обработку запросов со стороны серверов, равномерно распределить нагрузку на ядра процессора и как следствие получить прирост в производительности.

2 Постановка задачи

В рамках данной работы рассматривается упрощенная топология сети проекта — один сервер с дисками и некоторое число клиентов. Разрабатываемая сеть находится в пределах одного Ethernet—сегмента, поэтому в качестве протокола для доступа к дисковым ресурсам выбран ATA over Ethernet.

На сервере может быть установлено несколько сетевых карт — таким образом экспортируемый диск может быть доступен на нескольких сетевых интерфейсах[1]. Для экономии электроэнергии на сервере используются маломощные многоядерные процессоры, предназначенные для встроенных систем.

Задачами данной дипломной работы являются:

- проведение тестов, которые показывают, что выбранная для реализации сервера программа полностью использует одно ядро процессора;
- реализация схемы обработки запросов: очереди запросов должны обрабатываться несколькими процессами или потоками;
- отладка работы программы на виртуальных машинах;
- тестирование на системе, близкой по характеристикам к целевой;

1 Multipath routing – использование нескольких альтернативных путей в сети. В данном случае это означает что запрос к диску может прийти на разные сетевые интерфейсы.

3 Обзор

3.1 *Обзор сетевых протоколов*

Для сетей хранения данных нужны специальные протоколы, которые позволяют осуществлять доступ к удаленным дискам. Далее будут рассмотрены три наиболее используемых протокола.

3.1.1 *Internet Small Computer System Interface*

Название протокола произошло от SCSI – интерфейса, который позволяет устанавливать на шину различные по своему назначению устройства, в частности жесткие диски.

В терминологии SCSI взаимодействие происходит между инициатором и целевым устройством. Если речь идет о сети хранения данных, то инициатор может находиться далеко от целевого устройства и задачей iSCSI является надежная доставка SCSI команд по сети. Для адресации целевые устройства имеют уникальное имя, которое составляется по правилам, принятым для названия узлов в сети Internet.

iSCSI базируется на TCP/IP, поэтому в сетях, которые построены на основе данного протокола возможны значительные задержки при обработке пакетов. Задержки могут быть связаны не только с особенностью стека TCP/IP, в сети могут происходить различные сбои, приводящие к потере данных, также важно обеспечивать безопасность соединения из-за возможности несанкционированного доступа, что в свою очередь требует дополнительной затраты ресурсов.

Названные причины приводят к тому, что при скорости передачи данных, близкой к Gigabit Ethernet, может происходить почти полная загрузка современных процессоров [4]. Хотя протокол может работать и с обычными сетевыми картами, для эффективного использования требуются специальные сетевые адаптеры. У таких адаптеров есть есть своя память и процессор, что позволяет обрабатывать iSCSI-пакеты на аппаратном уровне.

С точки зрения проекта этот протокол избыточен, поскольку в рассматриваемой сети не требуется маршрутизация.

3.1.2 *Fibre Channel*

Основной протокол, используемый для сетей хранения данных [5]. Fibre Channel – это единственный протокол, каждый уровень которого спроектирован в расчете на сети хранения, поэтому он чаще всего используется как наиболее технологически совершенный. Раньше подсистемы Fibre Channel работали на скорости 250Mbits, но современные сети могут использовать скорости 1Gbit, 2Gbits, 4Gbits, 8Gbits или 10Gbits.

Fibre Channel состоит из пяти уровней:

- FC-0 физический уровень: описывает среду передачи, коннекторы и типы используемых кабелей, включает определение электрических и оптических характеристик, скоростей передачи данных и других физических компонентов.
- FC-1 уровень кодирования: описывает процесс кодирования.
- FC-2 сетевой уровень: описывает сигнальные протоколы. На этом уровне происходит определение слов, разбиение потока данных на кадры.
- FC-3 уровень общего обслуживания: определяет базовые и расширенные службы для транспортного уровня, а также такие особенности, как возможность передачи потока данных через несколько соединений и отображение множества портов на одно устройство.
- FC-4 уровень отображения протоколов: предоставляет возможность инкапсуляции других протоколов.

Существует разновидность этого протокола, специализированная для сетей Ethernet - Fibre Channel over Ethernet, спецификация протокола FCoE заменяет уровни FC0 и FC1 стэка Fibre Channel на Ethernet стэк.

Применение FCoE требует от Ethernet 3-х дополнительных возможностей

для передачи пакетов Fibre Channel в сетях хранения данных:

- инкапсуляция пакета Fibre Channel в пакет Ethernet;
- расширение возможностей протокола Ethernet для обеспечения передачи пакетов без потерь (lossless Ethernet);
- замена Fibre Channel соединения MAC адресом в lossless Ethernet.

Несмотря на то, что это лучший протокол для построения сетей хранения данных, он неприменим к проекту, поскольку требует специального дорогостоящего оборудования.

3.1.3 *ATA over Ethernet*

ATA over Ethernet (AoE) – сетевой протокол, разработанный компанией Coraid, предназначен для высокоскоростного доступа к SATA - устройствам хранения данных через сеть Ethernet. Спецификация данного протокола состоит из одиннадцати страниц [6], протокол является весьма легковесным, для сравнения спецификация протокола iSCSI составляет порядка 250 страниц [7]. Простота протокола позволяет использовать самое недорогое сетевое оборудование.

Протокол является немаршрутизируемым, для доступа к AoE–устройствам не нужны ip–адреса, AoE не оперирует с таким понятием, как соединение, каждое сообщение, отправленное на сервер считается уникальным и ненадежным.

В сравнении с протоколом iSCSI можно отметить следующие недостатки и преимущества:

- преимущества:
  - низкая стоимость;
  - высокая производительность;
  - простота.
- недостатки:

- низкая масштабируемость, возможность использования только в пределах одной Ethernet-сети — в данном случае нет необходимости экспортировать устройства дальше одного Ethernet-сегмента;
- отсутствие механизмов обеспечения безопасности — в данном случае считается, что сеть не может быть подвержена несанкционированному доступу;
- отсутствие механизмов контроля ошибок передачи данных, за исключением тех, что есть у самой сети Ethernet.

Клиентская часть AoE (AoE initiator) представляет из себя драйвер, позволяющий импортировать доступные в Ethernet — сегменете диски. Драйвер AoE включен в основную ветку ядра Linux с февраля 2005 года, разработан и поддерживается компанией Coraid. Драйвер позволяет системе воспринимать удаленный диск как обычное блочное устройство, импортированные диски видны в системе в директории /dev/etherd, например как /dev/etherd/e1.0.

Серверная часть AoE представляет из себя программу, которая экспортирует заданное блочное устройство в сегмент Ethernet. При настройках экспорта указываются номера «Shelf» и «Slot». Shelf — порядковый номер компьютера, на котором находится блочное устройство, Slot — порядковый номер блочного устройства. Например, номер AoE устройства 1.2 говорит о том, что блочное устройство располагается на сервере с порядковым номером 1 под порядковым номером 2.

3.1.3.1 *Используемая адресация*

Минимальной адресуемой единицей на жестком диске является сектор. В спецификации ATA размер сектора указан равным 512 байтам [6]. Существует несколько видов адресации — основные: CSH, Large и LBA [8].

CSH - система адресации сектора дисковых накопителей, основанная на использовании физических адресов геометрии диска. Блок на жестком диске

адресуется тремя числами: цилиндр, головка и сектор. CSH является самым старым видом адресации и не предоставляет доступ к данным за порогом в 8 Гб. В связи с этим появились новые системы адресации — Large, а затем LBA.

Протокол ATA over Ethernet использует LBA – адресацию (Logical block addressing) [6]. В LBA каждый блок, адресуемый на жёстком диске имеет свой номер - целое число, начиная с нуля. LBA не зависит от физических особенностей диска, в отличие от адресаций Large и CSH, и в настоящее время для задания номера блока используется 48 бит, что дает возможность адресовать 2^{48} блоков.

Нулевой блок в LBA задается следующим образом [9]:

```
LBA 0 = cylinder 0 / head 0 / sector 1
```

Формула для получения номеров остальных блоков [9]:

```
LBA = [ (cylinder * heads_num + head) * sectors / track]
+ (sector -1)
```

- `cylinder` – номер цилиндра
- `heads_num` – число головок
- `head` – номер выбранной головки
- `sectors` – число секторов на одной дорожке
- `track` – номер дорожки
- `sector` – номер сектора

3.1.3.2 *Регистры ATA устройств*

Рассмотрим основные регистры, которые принимают участие в выполнении ATA – команд [9]:

- `Data register` (регистр данных) - используется для передачи информации между программой и устройством.
- `Error register` (регистр ошибки) - регистр содержит информацию о ошибках в выполнении последней команды. Должен использоваться

только в том случае, если флаг ошибки ERR регистра Status содержит единицу.

- `Features` (Регистр возможностей) - назначение регистра зависит от производителя устройства. Многие диски игнорируют этот регистр.
- `Sector Count` (Регистр количества секторов) - в этот регистр заносится количество секторов, участвующих в операции. Если он содержит нуль, то под этим подразумевается число 256. После выполнения команды в нем содержится информация о количестве секторов, которые принимали участие в операции.
- `Sector Number` (Регистр номера сектора) - в режиме CHS в этот регистр заносится номер сектора, то есть одна из трех координат, необходимых для индексирования сектора. В LBA режиме в этот регистр после выполнения команды заносятся биты 0-7 линейного 28-разрядного адреса
- `Cylinder Low` (Регистр младшей части цилиндра) - в режиме CHS регистр содержит младшие 8 бит номера цилиндра. В режиме LBA регистр содержит биты 8-15 линейного 28-разрядного адреса.
- `Cylinder High` (Регистр старшей части цилиндра) - в режиме CHS регистр содержит старшую часть номера цилиндра. В режиме LBA регистр содержит биты 16-23 линейного 28-разрядного адреса.
- `Status` (Регистр состояния) - регистр содержит флаги, отображающие информацию о состоянии диска после выполнения команды.
- `Command` (Регистр команд) - в этот регистр заносится код команды, которую необходимо выполнить. Выполнение команды начинается сразу же после записи кода команды в этот регистр.

3.1.3.3 *Обзор спецификации протокола*

Рассмотрим подробнее сам протокол: в каждом AoE — пакете находится команда для ATA — диска или ответ от ATA — диска . В дополнение к ATA командам, у AoE есть простая возможность идентификации доступных устройств с помощью конфигурационных опросных пакетов. Все пакеты можно классифицировать следующим образом [6]:

1. Пакеты с ATA – командами.

В данный тип входят команды на чтение и запись. Инициатор должен указать номер стартового логического блока и количество секторов для чтения или записи.

Поле с данными выглядит следующим образом:

```
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 24 |    AFlags     |  Err/Feature  |  Sector Count |   Cmd/Status  |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 28 |     lba0      |     lba1      |     lba2      |     lba3      |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 32 |     lba4      |     lba5      |           Reserved            |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 36 |                               Data                            |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Сервер при разборе пакета с данной командой помещает значения адреса и количество блоков в соответствующие регистры устройства:

```
Device        <- (AFlags & 0x50 | 0xA0)
LBA Low       <- lba3
LBA Low       <- lba0
LBA Mid       <- lba4
LBA Mid       <- lba1
LBA High      <- lba5
LBA High      <- lba2
Sector Count  <- 0
Sector Count  <- Sector Count
Err/Feature   <- Err/Feature
Cmd/Status    <- Cmd/Status
```

При ответе происходит обратное: сервер отправляет состояние регистров устройства:

```
Device        <- lba3
LBA Low       <- lba0
LBA Mid       <- lba1
LBA High      <- lba2
Sector Count  <- Sector Count
Err/Feature   <- Err/Feature
Cmd/Status    <- Cmd/Status
```

В случае асинхронного ввода-вывода сервер может кэшировать запросы в оперативной памяти и отправлять все поля не измененными.

2. Конфигурационные пакеты.

   При запросе конфигурации сервер отправляет следующие сведения:

- максимальное число запросов, которые сервер может поставить в очередь;
- версию программного обеспечения;
- максимальное число секторов, которое сервер может обрабатывать с одной командой;
- версию протокола, которую поддерживает сервер;
- конфигурационную строку сервера.

  С конфигурационной строкой могут выполняться следующие операции:

- чтение;
- тестирование — строка или часть строки сервера должна совпадать со строкой клиента;
- установка строки — если строка пустая, то клиент может установить ее, или заменить на новую.

3. Пакеты с запросами списка доступных мак — адресов.

   Сервер хранит список мак адресов клиентов с которыми он общается. Сервер будет отвечать на клиентские запросы с ATA - командами только в том

случае, если список мак — адресов пуст или адрес клиента уже содержится в списке. Поэтому клиент в начале должен сообщить серверу свой мак — адрес.

Доступны следующие команды:

- прочитать список мак — адресов;
- добавить адрес в список;
- удалить адрес из списка.

Клиент может иметь несколько мак-адресов и сервер может общаться со многими клиентами, поэтому в пакете предусмотрено поле, в котором содержится количество добавляемых или запрашиваемых мак — адресов.

4. Пакеты с запросами на резервацию устройств.

Протокол предоставляет интерфейс для блокировки доступа к ATA – устройству. Блокировка осуществляется с использованием списка мак-адресов. Когда устройство зарезервировано, оно принимает команды на чтение и запись только от тех клиентов, чьи мак — адреса внесены в соответствующий список.

Доступны следующие команды:

- прочитать список зарезервированных мак — адресов;
- установить список зарезервированных мак — адресов.

В зависимости от приоритета команды, установка списка мак — адресов может быть проигнорирована, если мак — адрес клиента, отправившего такой запрос, не содержится в текущем списке.

3.2 Обзор программ

В этой части обзора рассматриваются open-source программы, которые могут быть установлены на AoE — сервере.

3.2.1 *Vblade*

Базовая программа [10], предоставленная компанией Coraid, написана предельно просто, позволяет экспортировать устройство одной командой и допускает только минимальные настройки:

```
[user@server:~$]vblade -m 10:20:AD:BF:11:FF 1 0 eth0 /data/testdev
```

- `eth0` – интерфейс, на котором будет доступно устройство;
- `/data/testdev` – файл устройства;
- `1.0` – номер, присваиваемый устройству;
- `10:20:AD:BF:11:FF` – список мак–адресов клиентов, для которых будет доступно соответствующее устройство.

Команда будет выполняться в терминале, но есть возможность запуска в качестве демона[1] при помощи скрипта *vbladed.*

Главным недостатком можно назвать отсутствие возможности экспорта устройства на несколько сетевых интерфейсов.

3.2.2 *Aoeserver*

Aoeserver [11] в отличие от других рассматриваемых программ является драйвером ядра. Управление программой осуществляется через файл /proc/aoeserver:

- добавление устройства:

  ```
  echo add /dev/testdev 1 3 eth1 > /proc/aoeserver
  ```

  Команда проэкспортирует устройство с именем testdev, присвоив ему номер 1.3, устройство будет доступно на интерфейсе eth1. В случае, если интерфейс не будет указан, устройство проэкспортируется на все

1 Демон (англ. daemon) – служба, работающая в фоновом режиме без прямого общения с пользователем.

доступные интерфейсы.

- удаление интерфейса:

  ```
  echo del /dev/testdev 1 3 eth1 > /proc/aoeserver
  ```

  для удаления соответствующего интерфейса требуется указывать те же параметры, что и при добавлении устройства.

- ограничение доступа к устройствам:

  ```
  echo hostmask 1 3 00:01:02:03:04:05 > /proc/aoeserver
  ```

  команда запретит устройству с номером 1.3 принимать запросы от клиентов с мак – адресом, отличающимся от указанного.

  ```
  echo rmmask 1 3 00:01:02:03:04:05 > /proc/aoeserver
  ```

  команда удалит ограничение.

По сравнению с vblade, данная программа немного выигрывает в производительности при экспортировании одного устройства и может существенно лучше работать при экспортировании нескольких устройств за счет многопоточности.

Недостатки программы:

- программа является драйвером, ее трудно поддерживать и отлаживать;
- синхронный ввод-вывод может приводить к лишним затратам времени.

3.2.3 *Qaoed*

Программа [12] настраивается при помощи конфигурационного файла, в котором можно указывать следующие параметры:

- экспортируемые устройства:

```
device
{
     shelf = 0;                    // номер сервера
     slot = 2;                     // номер диска на сервере
     target = /root/data;          //   экспортируемый   файл   или
устройство
     log-level = info;
     // интерфейс для доступа к устройству
     interface = eth0;
```

```
    // список мак-адресов для доступа к устройству
    acl
    {
        cfgread = acl1;
        cfgset = acl1;
        read = acl1;
        write = acl1;
    }
}
```

- настройки сетевых интерфейсов:

```
interface
{
      interface = "eth0";          // название сетевого интерфейса
      mtu = auto;                  // максимальный размер блока
      log = default;
      log-level = info;
}
```

- настройки списков доступа:

```
access-list
{
    name = acl1;                   // название списка
    policy = accept;               // правило по умолчанию
    // пакеты, пришедшие с этого мак-адреса должны быть
обработаны.
    accept = "00:12:23:45:66:FF";
}
```

Для начала работы нужно заполнить конфигурационный файл, а затем выполнить следующую команду:

```
[user@server:~/qaoed-src/$] ./qaoed -c server.conf
```

после этого программа экспортирует диски и работает в режиме демона.

Главным недостатком qaoed является отсутствие возможности экспортировать устройство на несколько сетевых интерфейсов.

3.2.4 *Ggaoed*

Ggaoed [13] является наиболее удачной реализацией AoE — сервера по сравнению с ранее рассмотренными. Некоторые идеи при написании программы заимствованы у qaoed и vblade.

Основные особенности:

- все запросы к дискам, поступающие с разных сетевых интерфейсов

обрабатываются при помощи одного процесса;

- использование Linux AIO;
- запросы для соседних блоков данных при чтении и записи объединяются в один запрос;
- для множества запросов может быть использован один системный вызов;
- поддержка подключения и отключения сетевых интерфейсов во время работы программы;
- использование общей памяти для доступа к сетевым сокетам;
- использование интерфейса epoll;
- использование интерфейса eventfd для получения уведомления о завершении ввода-вывода;
- возможность экспортирования диска на несколько интерфейсов.

Для доступа к статистике можно использовать утилиту ggaoectl, которая выдаст данные по текущему состоянию:

```
[user@server:~/ggaoed-src/$] ./ggaoectl -c server.conf
```

активность сетевых интерфейсов:

net	rrqm/s	rkB/s	wrqm/s	wkB/s	drp	avrun
eth0	11828.7	47730.75	11828.7	415.85	0	5,77
eth1	11862.6	47867.53	11861.6	417.01	0	5,78

Таблица 1 — статистика сетевых интерфейсов

активность устройств:

dev	rrqm/s	rkB/s	wrqm/s	wkB/s	oth	avgqsz	qs	qf	ae	pe	svctm
dev1	93.8	375.06	6686.2	26744	2	10,69	0.00	1424	0.00	0.00	0.80
dev2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
dev3	93.8	375.06	6685.2	26740	2	10,43	0.00	1360	0.00	0.00	0.80
dev4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
dev5	0.00	0.00	8956.4	35825	0.00	10,11	0.00	1569	0.00	0.00	0.73
dev6	0.00	0.00	0.0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Таблица 2 — статистика устройств

Для настройки программы используется конфигурационный файл,

подобный файлу для qaoed. Для запуска программы достаточно указать в конфигурационном файле настройки экспортируемых дисков и интерфейсов на которых они могут быть доступны, а затем выполнить:

```
[user@server:~/ggaoed-src/$] ./ggaoed -c server.conf
```

после этого программа запустится в режиме демона.

В дальнейшем архитектура ggaoed будет рассмотрена более подробно.

3.2.5 *Aoetools*

Aoetools — утилиты для различных настроек со стороны клиента. Они взаимодействуют с драйвером aoe ядер Linux версий 2.6. Рекомендуется использовать версию не ниже 2.6.14, так как в них добавлена поддержка до 16 адресов slot на один адрес shelf.

Наиболее важные утилиты, которые входят в пакет:

- aoeping - утилита для связи между устройствами AoE

```
[root@client:~#] aoeping -v 1 10 eth0
tag: 80000000
eth: eth0
shelf: 1
slot: 10
config query response:
00 21 91 0b ff 9a 00 21 91 0b 70 51 88 a2 18 00
00 01 0a 01 80 00 00 00 00 10 01 00 0d 10 00 00
found e1.10 with mac 0021910b7051
```

- aoe-stat - вывод информации о состоянии AoE-устройства

```
 [root@client:~#] aoe-stat
e1.1         0.104GB  eth2,eth0 6656  up
e1.2         0.104GB  eth2,eth0 6656  up
e1.3         0.104GB  eth2,eth0 6656  up
```

- aoe-version - вывод информации о версии ПО для устройств AoE

```
[root@client:~#] aoe-version
aoetools:               32
```

```
installed aoe driver:    76
running aoe driver:      76
```

оставшиеся утилиты:

- aoecfg - настройка параметров AoE
- aoe-discover - включение/выключение поиска устройств ATA через Ethernet
- aoe-flush - сброс состояния устройств в драйвере aoe
- aoe-interfaces - ограничение интерфейсов доступных для AoE
- aoe-mkdevs - создание файлов символьных и блочный устройств
- aoe-mkshelf - создание символьного устройства для адреса полки
- aoe-revalidate - перепроверка размера диска AoE-устройства

3.2.6 *Заключение*

Для использования со стороны сервера была выбрана программа ggaoed, как наиболее производительная и многофункциональная из всех рассмотренных, со стороны клиента будет использоваться стандартный драйвер aoe, а также набор утилит aoetools для настройки доступа к удаленным устройствам.

4 Архитектура ggaoed

В этой главе рассматриваются основные особенности программы ggaoed.

4.1 *Очереди запросов*

При обработке запросы на сервере могут выстраиваться в очереди, поскольку их количество может быть достаточно большим и не все запросы могут быть обработаны сразу.

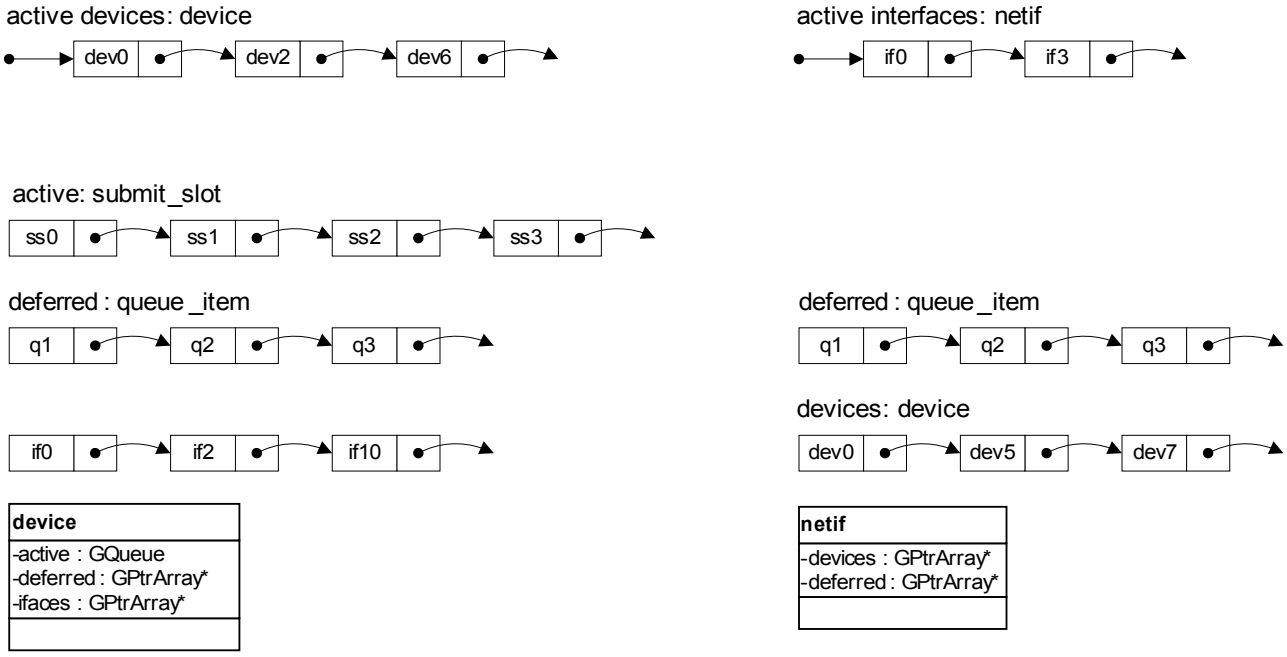

рис.1 схема очередей ggaoed

В программе используется 3 вида очередей:

- *очередь входящих запросов*: соответствует каждому сэкспортированному устройству, как только запрос приходит на соответствующий интерфейс, производится поиск среди всех устройств, доступных на данном интерфейсе, а затем запрос ставится в очередь к найденному устройству;
- *очередь обрабатываемых запросов:* соответствует устройству — очередь входящих запросов на устройстве сортируется, затем запросы к соседним секторам диска объединяются в один и ставятся в очередь для асинхронного ввода-вывода;

- *очередь обработанных запросов для отправки:* соответствует каждому сконфигурированному сетевому интерфейсу, ответ на запрос попадает в эту очередь только в случае, когда в файле сокета не остается места для нового пакета.

Ниже представлен конечный автомат, который демонстрирует схему обработки запроса:

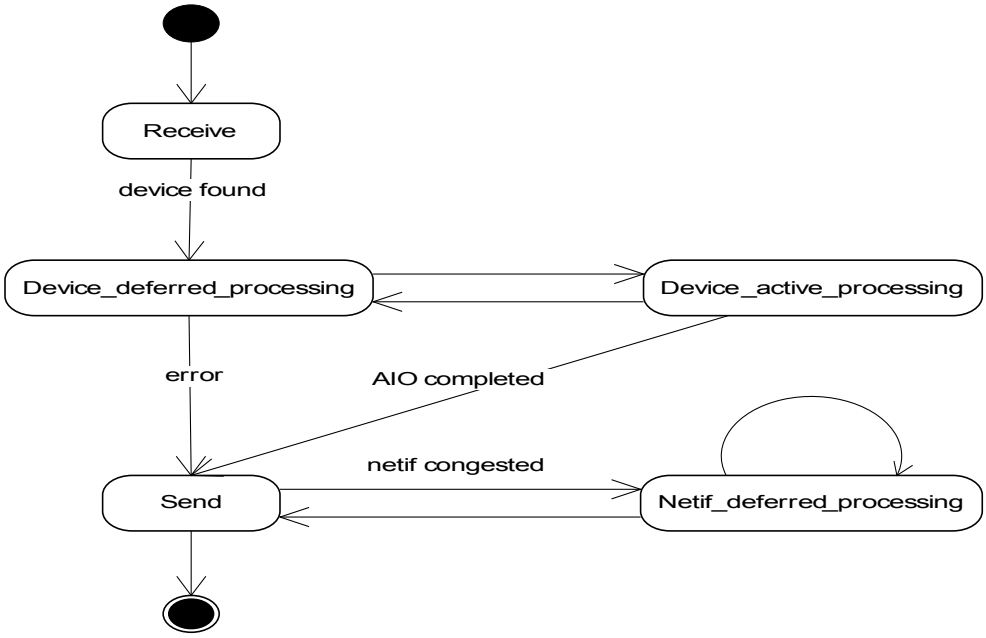

Рис.2 схема обработки одного запроса

Состояния Device_deferred_processing, Device_active_processing и Netif_deferred_processing отражают обработку соответствующих очередей. Состояния Receive и Send означают прием и отправку пакетов соответственно.

4.2 *Механизм получения событий*

Программа использует интерфейс epoll для отслеживания следующих видов событий:

- получение запроса на сетевом сокете;
- завершение операций ввода / вывода;
- включение / выключение сетевых интерфейсов.

Данные события отслеживаются при помощи файловых дескрипторов, ассоциированных с соответствующими сущностями:

- для отслеживания приходящих запросов на каждом сетевом интерфейсе используется дескриптор сокета:

```
iface->fd = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_AOE));
```

- для каждого устройства создается отдельный дескриптор:

```
dev->event_fd = eventfd(0, EFD_NONBLOCK);
```

когда в ядро ставится запрос на асинхронный ввод-вывод, вызывается функция, которая указывает, что уведомление о завершении будет отправлено на выбранный дескриптор:

```
io_set_eventfd(&s->iocb, s->dev->event_fd);
```

- для отслеживания состояния интерфейсов используется netlink — сокет :

```
nl_fd = socket(PF_NETLINK, SOCK_DGRAM, NETLINK_ROUTE);
```

такой вид сокетов используются для обмена данными между ядром и уровнем пользователя, в данном случае указываются следующие параметры:

```
addr.nl_family = AF_NETLINK;
addr.nl_groups = RTMGRP_LINK;
ret = bind(nl_fd, (struct sockaddr *)&addr, sizeof(addr));
```

`RTMGRP_LINK` – флаг, который указывает, что необходимо отслеживать создание, удаление, включение и выключение сетевых интерфейсов.

В начале работы программы создается дескриптор для epoll:

```
efd = epoll_create(32),
```

а затем все описанные выше дескрипторы добавляются для прослушивания следующим образом:

```
struct event_ctx *ctx;
event.events = EPOLLIN | EPOLLOUT;
event.data.ptr = ctx;
epoll_ctl(efd, EPOLL_CTL_ADD, fd, &event);
```

структура event_ctx содержит соответствующие флаги, которые определяют события, на которые следует реагировать:

- `EPOLLIN` – файл доступен для операций чтения
- `EPOLLOUT` – файл доступен для операции записи

и контекст, который будет передан программе при получении соответствующего события.

После того, как все подготовительные действия завершены, запускается цикл обработки запросов, в котором периодически запускается функция epoll_wait и обрабатываются все полученные события.

4.3 *Прием запросов*

Для получения запросов ggaoed использует кольцевые буферы, размещенные в общей памяти с ядром, данные буферы являются отображением файлов сокетов.

Для каждого сетевого интерфейса создается два таких буфера — для приема и для отправки. Данный подход применяется для снижения числа системных вызовов при операциях с пакетами, что приводит к повышению производительности.

Настройки размеров колец являются важными параметрами системы и задаются через конфигурационный файл. От MTU, размера колец и блоков зависит процент фрагментированной памяти:

MTU	Block size	Frames/block	Wasted
9000	64k	7	3.3%
9000	32k	3	17.1%
9000	16k	1	44.7%
4608	64k	14	0.6%
4608	32k	7	0.6%
4608	16k	3	14.8%
4608	8k	1	43

Таблица 3 — фрагментация памяти в зависимости от размера MTU.

5 Реализация

Для выполнения задачи была создана минимально возможная сеть хранения данных - из одного сервера и одного клиента, на обоих машинах было установлено по две сетевые карты, соединение обеспечивалось при помощи кросс-кабелей, без дополнительного сетевого оборудования.

На сервере был установлен процессор Intel Atom 330. Процессор является двухъядерным и для каждого ядра присутствует поддержка технологии Hyper-threading. Цель использования такого процессора — низкая стоимость и низкое энергопотребление.

Для каждого сетевого интерфейса была измерена пропускная способность. Для встроенной сетевой карты:

```
[root@server:~#] iperf -s
[ 4] 0.0-10.0 sec 987 MBytes 826 Mbits/sec
```

и для дополнительной сетевой карты:

```
[root@server:~#] iperf -s
[ 4] 0.0-10.0 sec 991 MBytes 840 Mbits/sec
```

Таким образом максимальная суммарная пропускная способность сети составила около 200 Мб/сек.

Требовалось определить, какие части программы в первую очередь нужно распараллеливать, для этого был использован профайлер gprof:

Название функции	% от общего времени работы программы	Число вызовов функции	Время выполнения функции
process_request	15,15	47347	0,05
net_io	12,25	918	0,04
finish_request	10,61	47303	0,04
queue_compare	9,09	59321	0,03
new_request	6,06	47367	0,02
send_response	6,06	47303	0,02
free_packet	6,06	47283	0,02
process_packet	3,03	48252	0,01

ata_rw	3,03	47190	0,01
run_queue	3,03	5935	0,01
dev_io	3,03	2956	0,01
run_ifaces	3,03	958	0,01

Таблица 4 — результаты профилирования.

Из результатов видно, что большую часть времени занимают функции, которые связаны с приемом и разбором пакетов, а также существенное время тратится на сортировку входящих очередей. Отсутствие больших задержек при обработке входных очередей объясняется использованием асинхронного ввода-вывода.

Существует несколько общепринятых схем обработки запросов:

- обработка одним потоком:

Все действия производятся последовательно одним процессом, не требуется тратить ресурсы на синхронизацию.

Недостатки:

- клиенты могут достаточно долго ожидать ответа на запрос;
- программе доступен только один процессор.

- обработка в несколько потоков:

Такой подход позволяет ускорить обработку запросов в случае, когда на системе установлено несколько процессоров. В случае, когда процессор один, данный подход позволяет уменьшить время отклика для каждого клиента в отдельности. На каждый запрос создается отдельный поток, что приводит к следующим проблемам:

- затраты ресурсов на создание и уничтожение потоков;
- малое время работы одного потока;
- большое число потоков;
- большая затрата ресурсов на переключение между потоками.

- обработка пулом потоков:

В этом случае заранее создается некоторое число потоков. Главный поток принимает клиентские запросы и каждому запросу назначает поток из пула для обработки. Если свободного потока нет, то запрос ставится в очередь. Данный способ решает проблемы, свойственные предыдущим двум, поэтому он был выбран для реализации.

На целевой системе установлен двухъядерный процессор с поддержкой двух потоков на каждом ядре, соответственно оптимальную работу можно ожидать при двух или четырех запущенных потоках. В реализации обработка может происходить любым фиксированным числом потоков, но создавать имеет смысл либо два либо четыре по указанным выше причинам.

При реализации был сохранен механизм epoll, поскольку для поиска по списку прослушиваемых дескрипторов epoll использует алгоритм с трудоемкостью O(1) и может выдавать много событий за один раз, что обеспечивает хорошую масштабируемость.

Для каждого потока из пула была создана очередь событий, в которую они ставятся по мере поступления главным потоком. Каждый поток может находиться в двух состояниях — либо он последовательно выполняет задания из своей очереди, либо ожидает поступления новых заданий. Нагрузка на потоки распределяется равномерно — выбор потока для определенного задания производится алгоритмом, подобным round robin.

При организации обработки запросов немаловажно снижать возможные пересечения по данным между потоками, так как большое количество блокировок может привести к существенному снижению производительности. Поскольку в программе поддерживается технология multipath, устройства и интерфейсы относятся как многие ко многим, что увеличивает потенциальное число блокировок - например в случае, когда разные потоки могут добавлять или удалять запрос для одной и той же очереди часто требуется дополнительная синхронизация. В связи с этим было принято решение сначала обрабатывать события, приходящие от сетевых интерфейсов, а затем события, приходящие от

устройств. То есть уменьшение числа пересечений по общим для потоков очередям достигается за счет того, что на первом этапе приходящие запросы ставятся в очереди на соответствующие устройства и запросы в очередях на интерфейсах копируются для отправки, на втором этапе происходит обработка входящих очередей — завершение ввода-вывода и постановка новых запросов на ввод-вывод.

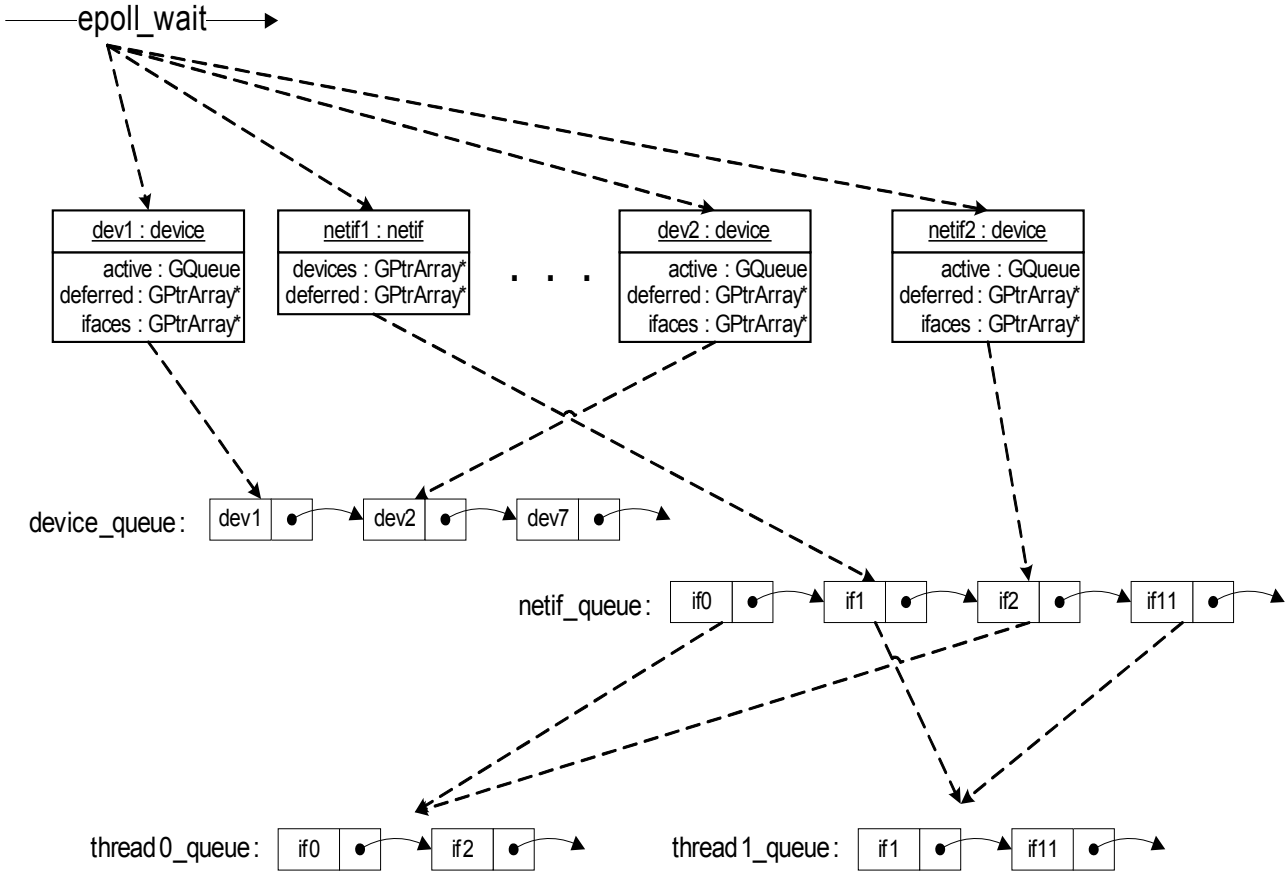

Рис. 3 установка заданий для потоков

Для отладки программы были использованы виртуальные машины VirtualBox, объединенные в одну общую сеть. Одна машина выступала в качестве сервера, остальные выполняли роли клиентов. Система тестировалась для следующих параметров:

- число клиентов: 1 - 9;
- число сетевых интерфейсов на сервере: 1 - 4;
- число экспортируемых дисков: 1 - 15;

- размер MTU: 1500 - 9000 байт;
- число потоков: 1 - 15.

Тестирование на виртуальных машинах подтверждает работоспособность программы и целостность данных при передаче. Виртуальные машины имитируют сетевые карты и процессоры, а их производительность и доступная им операционная память зависят от их количества и числа других приложений, запущенных в системе, поэтому производительность замерялась на конфигурации, близкой к целевой.

6 Результаты реализации

В данной работе были получены следующие результаты:

1. При помощи собранной сети было показано, что выбранная программа может полностью использовать ресурс одного ядра процессора при определенных нагрузках.

Скорость передачи данных может быть ограничена несколькими параметрами: скоростью записи на диск, пропускной способностью сети и производительностью процессора. На тестируемой конфигурации был установлен только один винчестер, поэтому, для того чтобы исключить ограничение в скорости, связанное с ним, тестирование проводилось на устройствах, размещенных в оперативной памяти, также, для увеличения скорости при передаче данных, в параметрах программы был отключен режим direct-io.

Со стороны сети наибольшая производительность достигается при использовании максимально возможного числа сетевых интерфейсов и максимального MTU, доступного сетевым картам.

При скорости передачи данных, близкой к скорости Gigabit Ethernet и записи только на одно устройство, ядро целевого процессора было загружено на 90-100%, второе ядро при этом оставалось полностью свободным. Таким образом, было показано, что в данном случае производительность процессора является одним из определяющих факторов скорости передачи данных.

2. Обработка запросов может производиться при помощи нескольких потоков.

Цикл обработки запросов делится на несколько последовательных этапов, на которых в свою очередь может выполняться параллельный код:

- получение новых запросов;
- обработка текущих запросов;
- отправка обработанных запросов;

На каждом этапе потоки имеют минимальное пересечение по общим

данным и при разном количестве таких этапов производительность не ухудшалась сколько-нибудь значительным образом, из чего следует вывод о том, что реализация не приводит к значительным затратам на синхронизацию потоков.

Оптимальный результат был достигнут при использовании двух потоков, без включенной поддержки технологии Hyper-threading.

3. При использовании нескольких потоков нагрузка на процессор распределяется между ядрами.

Основную часть ресурса процессоров занимает обработка входящих очередей запросов, то есть основной прирост в производительности получается за счет распараллеливания этой части программы.

Программа использует события для уведомления о пришедших на сетевой интерфейс пакетах, за один шаг цикла обрабатывается определенное число событий, каждому событию назначается отдельный поток. Каждому интерфейсу может соответствовать только одно событие, поэтому с данной схемой реализации будет невозможно получить значительный выигрыш в производительности при использовании одного сетевого интерфейса.

Еще одним минусом данного подхода является то, что даже из двух сетевых интерфейсов, в некоторых случаях одновременно может обрабатываться только один, так как программа может получить в цикле обработки событие только от одного сетевого интерфейса.

4. Для тестируемой конфигурации было получено до 12% прироста производительности.

7 Использованные источники

[1] New Study Forecasts Explosive Growth Of The Digital Universe,
(http://www.emc.com/about/news/press/2008/20080311-01.htm)

[2] Протоколы сетей хранения: настоящее и перспективы,
(www.storagenews.ru/34/SAN-FC_34.pdf)

[3] Сети хранения данных на базе интерфейса Fibre Channel,
(http://www.williamspublishing.com/PDF/5-8459-0746-2/part.pdf)

[4] iSCSI и другие,
(http://www.ixbt.com/storage/iscsi.shtml)

[5] Протоколы SAN,
(http://www.fcoe.ru)

[6] AoE protocol specification,
(http://support.coraid.com/documents/AoEr11.txt)

[7] iSCSI protocol specification,
(http://tools.ietf.org/html/rfc3720)

[8] Cylinder-head-sector,
(http://en.wikipedia.org/wiki/Cylinder-head-sector)

[9] Information technology - AT Attachment Interface for Disk Drives,
(http://www.t10.org/t13/project/d0791r4c-ATA-1.pdf)

[10] vblade sources,
(http://aoetools.sourceforge.net/)

[11] Aoeserver sources,
(http://code.google.com/p/aoeserver/)

[12] Qaoed sources,
(http://code.google.com/p/qaoed/)

[13] Ggaoed sources,
(http://code.google.com/p/ggaoed/)