Санкт-Петербургский Государственный Университет

Математико-механический факультет

Кафедра системного программирования

Поддержка избыточного кодирования в проекте «Cirrostratus». Реализация алгоритмов избыточного кодирования на уровне ядра Linux.

Курсовая работа студента 345 группы
Алеева Алексея Валерьевича

Научный руководитель ……. А.Н. Косякин
ведущий разработчик, ООО "Артек"

Санкт-Петербург
2010

Оглавление

Введение

Во многих организациях главным активом является информация, поэтому нужно обеспечивать надежное хранение информационных ресурсов и быстрый доступ к ним.

В настоящее время повышенный интерес вызывает распределенное хранение данных, обладающее рядом преимуществ перед традиционным прямым подключением дисковых массивов к серверам: сеть хранения данных представляет собой архитектурное решение для подключения внешних устройств хранения данных, таких как дисковые массивы, к серверам таким образом, чтобы операционная система распознала подключённые ресурсы как локальные.

Основные преимущества:

- Высокая масштабируемость;
- Высокая производительность и надежность;
- Простота администрирования;
- Эффективное восстановление работоспособности после сбоя;

На данный момент существует множество решений, но на небольших предприятиях они являются редкостью из-за слишком высокой стоимости. Целью проекта «Cirrostratus» является создание надежной сети хранения данных, работающей на дешевом, доступном оборудовании и обладающей всеми вышеуказанными свойствами.

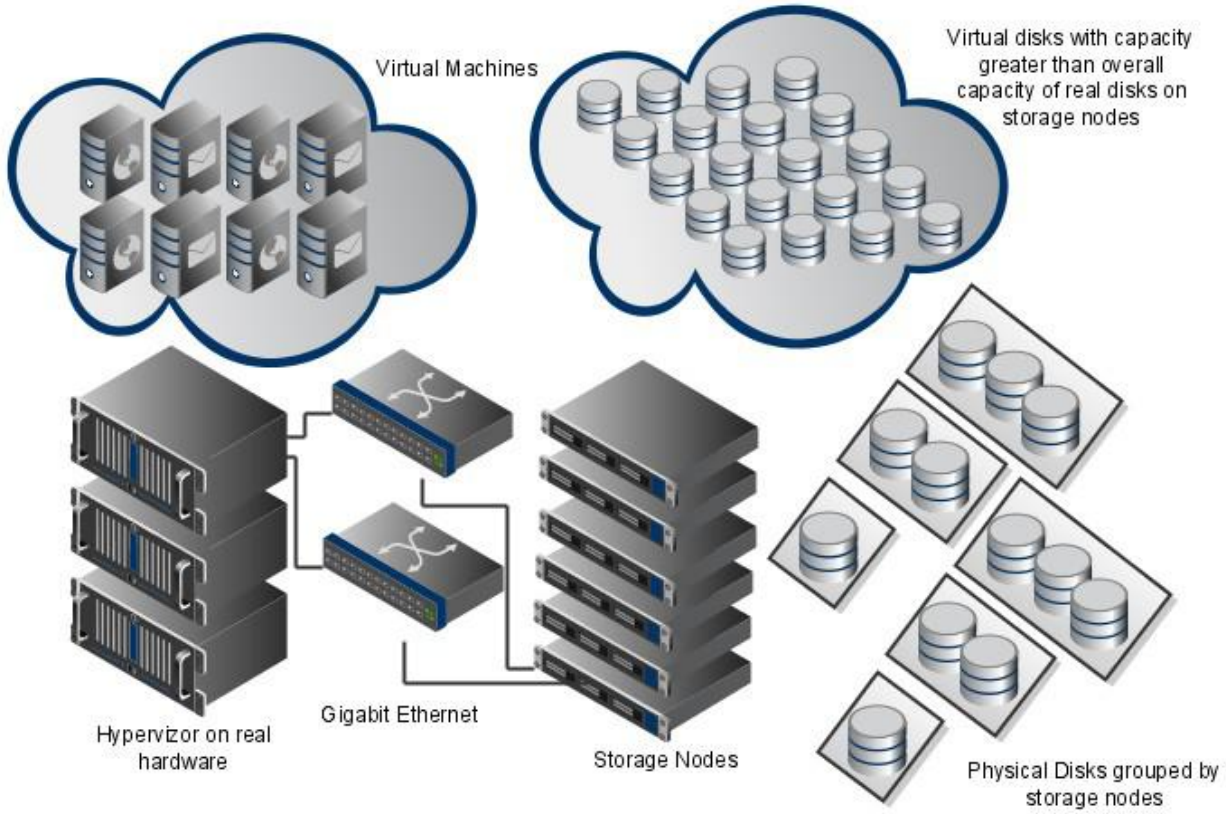

Рис.1 Распределенное хранилище

Проект используется для "Облачных вычислений", поверх некоторого числа реальных серверов имитируется в несколько раз большее число виртуальных серверов, которые внешне ничем не отличаются от реальных. Открытым стоит вопрос, как недорого имитировать жесткие диски для этих виртуальных машин (сегодня виртуальный сервер работает на сервере 1, завтра - на сервере 2; не переставлять же диски руками!), причем при сохранении приемлемой производительности и высокой надежности, ведь современные винчестеры довольно ненадежны и могут выйти из строя, либо быть бракованными.Чтобы защитить данные от отказов узлов хранения, но при этом эффективно использовать имеющееся дисковое пространство и обеспечивать приемлемую нагрузку на CPU, предлагается использовать избыточное кодирование.

Постановка задачи

В рамках проекта были проведены анализ существующих алгоритмов избыточного кодирования и выбор наиболее эффективного алгоритма (Minimal Density RAID-6 codes). Моей задачей была реализация выбранного алгоритма на уровне ядра Linux.

Данную задачу можно разбить на следующие подзадачи:

1. Овладеть принципами построения модулей к ядру Linux.
2. Реализовать выбранный алгоритм как модуль к ядру Linux.

Crypto API

Для удобной работы с различными алгоритмами кодирования в ОС Windows/Solaris/Linux/FreeBSD может быть использован интерфейс Crypto API.

Реализация всех алгоритмов (шифрования, цифровой подписи и т.п.) полностью выведена из состава самого Crypto API и реализуется в отдельных, независимых динамических модулях – «криптопровайдерах» (Cryptographic Service Provider – CSP). Сам же Crypto API просто предъявляет определенные требования к набору функций (интерфейсу) криптопровайдера и предоставляет конечному пользователю унифицированный интерфейс работы с CSP. Конечному пользователю для полноценного использования всех функций криптопровайдера достаточно знать его строковое имя и номер типа.

Как уже было сказано, криптопровайдером называют независимый модуль, обеспечивающий непосредственную работу с криптографическими алгоритмами. Каждый криптопровайдер должен обеспечивать:

- реализацию стандартного интерфейса криптопровайдера;
- работу с ключами шифрования, предназначенными для обеспечения работы алгоритмов, специфичных для данного криптопровайдера;
- невозможность вмешательства третьих лиц в схему работы алгоритмов.

Общая архитектура CryptoAPI состоит из пяти основных функциональных групп:

1) **Базовые криптографические функции**
   - Функции инициализации (работы с контекстом). Эти функции предоставляют приложению возможность выбрать определенный криптопровайдер по типу имени или по требуемой функциональности.
   - Функции генерации ключей. Эти функции предназначены для формирования и хранения криптографических ключей различных типов.
   - Функции обмена ключами. Эти функции предназначены для того, чтобы приложения могли обмениваться различными типами ключевой информации для обеспечения взаимодействия между собой.

2) **Функции кодирования/декодирования**

Данные функции предназначены для преобразования (кодирования) из внутреннего представления объектов, используемых в CryptoAPI, во внешнее представление и обратно. В

качестве внешнего представления объектов используется формат ASN.1 (Abstracy Syntax Notation One), определенный серией рекомендаций X.680.

К этой же группе функций можно отнести набор функций, позволяющих расширить функциональность CryptoAPI путем реализации и регистрации собственных типов объектов.

3) **Функции работы со справочниками сертификатов**

Эта группа функций предназначена для хранения и обработки сертификатов в различных типах справочников. Причем в качестве справочника могут использоваться самые различные типы хранилищ: от файла до LDAP.

4) **Высокоуровневые функции обработки криптографических сообщений**

Именно эта группа функций (Simplified Message Functions) в первую очередь предназначена для использования в прикладном ПО. С помощью этих функций можно:

- Зашифровать/расшифровать сообщение от одного пользователя к другому.
- Подписать данные.
- Проверить подпись данных.

5) **Низкоуровневые функции обработки криптографических сообщений**

Данная группа функций (Low Level Message Functions) предназначена для аналогичных целей, что и группа высокоуровневых функций, но обладает большей функциональностью. Вместе с тем, большая функциональность потребует от прикладного программиста более детальных знаний в области прикладной криптографии.

Crypto API в ядре Linux

Crypto API в ядре Linux реализуется на уровне модулей. Чтобы использовать собственный алгоритм кодирования для работы с Crypto API в ядре Linux, нужно написать свой собственный модуль ядра, реализующий интерфейс Crypto API (тот самый криптопровайдер, о котором было сказано выше). В «исходниках» ядра Linux можно найти файл crypto_null.c, который служит шаблоном для создания такого модуля. В нём нужно заменить «пустые» структуры и методы своими. В структуре crypto_alg, описанной в include/linux/crypto.h, есть все необходимое для корректной работы непосредственно алгоритма. В crypto_null.c присутствует такая структура, в частности, в таком виде:

```
static struct crypto_alg skcipher_null = {
      .cra_name         =     "ecb(cipher_null)",
      .cra_driver_name  =     "ecb-cipher_null",
      .cra_priority           =     100,
      .cra_flags        =     CRYPTO_ALG_TYPE_BLKCIPHER,
      .cra_blocksize          =     NULL_BLOCK_SIZE,
      .cra_type         =     &crypto_blkcipher_type,
      .cra_ctxsize            =     0,
      .cra_module       =     THIS_MODULE,
      .cra_list         =     LIST_HEAD_INIT(skcipher_null.cra_list),
      .cra_u                  =     { .blkcipher = {
      .min_keysize            =     NULL_KEY_SIZE,
      .max_keysize            =     NULL_KEY_SIZE,
      .ivsize                 =     NULL_IV_SIZE,
      .setkey                 =     null_setkey,
      .encrypt          =     skcipher_null_encrypt,
      .decrypt          =     skcipher_null_decrypt } }
};
```

В данной структуре заполняется общая информация об алгоритме (имя, размер блока входных данных, приоритет и т.д.), а также указываются функции, которые будут осуществлять кодирование и декодирование (в приведённой выше структуре это функции skcipher_null_encrypt и skcipher_null_decrypt).

Помимо этого, необходимо определить функции, которые будут выполняться при установке модуля в ядро и при его выгрузке оттуда.

Реализация Minimal Density RAID-6 кодов

В результате анализа алгоритмов избыточного кодирования было решено реализовать коды минимальной плотности (Minimal Density RAID-6 codes) как наиболее подходящие для проекта. Существуют три конструкции кодов минимальной плотности в зависимости от размера пакета входных данных w:

- Blaum-Roth codes (когда w+1 является простым числом)
- Liberation codes (когда w является простым числом)
- The Liber8tion code (когда w = 8)

Подробнее о них можно узнать из работы [1].

Для создания модуля для работы с этими алгоритмами была использована следующая структура:

```
static struct crypto_alg md_raid6_cipher = {
	.cra_name		=	"md_raid6",
	.cra_driver_name	=	"dev-md_raid6",
	.cra_priority		=	100,
	.cra_flags		=	CRYPTO_ALG_TYPE_BLKCIPHER,
	.cra_blocksize		=	MD_RAID6_BLOCK_SIZE,/* = w * packetsize
*/
	.cra_type		=	&crypto_blkcipher_type,
	.cra_ctxsize		=	MD_RAID6_STATE,
	.cra_module		=	THIS_MODULE,
	.cra_list		=	LIST_HEAD_INIT(md_raid6_cipher.cra_list),
	.cra_u			=	{ .blkcipher = {
	.min_keysize		=	8,
	.max_keysize		=	64,
	.ivsize			=	MD_RAID6_IV_SIZE,
	.setkey			=	md_setkey,
	.encrypt		=	encoder_md_raid6,
	.decrypt		=	decoder_md_raid6 } }
};
```

А так же функции, которые будут выполняться при установке модуля в ядро и при его выгрузке оттуда:

```
static int __init crypto_mod_init(void)
{
        int ret = 0;
```

```
        ret = crypto_register_alg(&md_raid6_cipher);
        if (ret < 0)
                goto out;
out:
        return ret;
}

static void __exit crypto_mod_fini(void)
{
        crypto_unregister_alg(&md_raid6_cipher);
}

module_init(crypto_mod_init);
module_exit(crypto_mod_fini);
```

Тестирование

Реализованные алгоритмы были протестированы на ноутбуке фирмы Acer с характеристиками Core 2 Duo processor P8700 (2,53 GHz), 4 GB RAM в различных условиях нагрузки на CPU.

В результате были получены следующие значения:

Кодирование небольших (около 10 MB) файлов (табл.1):

Число пакетов данных (w)	Размер пакета в байтах (packetsize)	Скорость работы алгоритма (MB/Sec)	Итоговая скорость работы программы (MB/Sec)
11	3000	1940,38	323,48
11	900	1506,56	286,3
31	900	1947	344
31	3000	1548,22	270
61	900	1709,22	306,52
8	3000	1963,9	330,0
10	3000	1909,38	323,48

Табл.1 Тестирование алгоритмов на небольших файлах.

Кодирование файлов размеров порядка 200 MB (табл.2):

Число пакетов данных (w)	Размер пакета в байтах (packetsize)	Скорость работы алгоритма (MB/Sec)	Итоговая скорость работы программы (MB/Sec)
8	1500	1939,9	288
8	3000	2087,56	346,8
11	1500	1956,33	291,3
11	900	1908	256
31	1000	1978,7	324
31	3000	1708	270
61	900	1809,22	306,52

Табл.2 Тестирование алгоритмов для файлов размера порядка 200 MB .

Кодирование файлов размеров порядка 1,5 GB (табл.3):

Число пакетов данных (w)	Размер пакета в байтах (packetsize)	Скорость работы алгоритма (MB/Sec)	Итоговая скорость работы программы (MB/Sec)
11	1000	1169,84	28,767
11	3000	1284,69	34,44
61	2000	1040,28	23,35
31	900	1390,8	43,64

Табл.3 Тестирование больших файлов.

Для тестирования были выбраны несколько значений параметра w, а так же параметра packetsize, наиболее оптимальных или неоптимальных для параметра w (см. работу [6]).

В дальнейшем планируется провести более тщательное тестирование для различных сценариев работы системы.

Заключение

По окончании работы были получены следующие результаты:

- Получен опыт написания модулей ядра Linux.
- Улучшены навыки программирования на языке C.
- Реализованы алгоритмы Minimal Density RAID-6 на уровне ядра Linux.
- Проведено тестирование работы алгоритмов.

Используемые источники

1. BLAUM, M., BRADY, J., BRUCK, J., AND MENON, J. EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures. //IEEE Transactions on Computing 44, 2 (February 1995), 192– 202.
2. CORBET, J., RUBINI, A., KROAH-HARTMAN, G.: Linux Device Drivers, Third Edition.
3. Crypto API в ядре Linux: http://diploma-thesis.siewior.net/html/diplomarbeitch4.html
4. The Linux Crypto API. A user's perspective: http://m4dch4t.effraie.org/crypto/stegano/cryptous.pdf
5. Документация Crypto API: http://www.faqs.org/rfcs/rfc2628.html
6. Описание алгоритмов: http://www.usenix.org/events/fast09/tech/full_papers/plank/plank_html/index.html