EXT4 filesystem

Nekriach Vladyslav, TK-41

EXT2

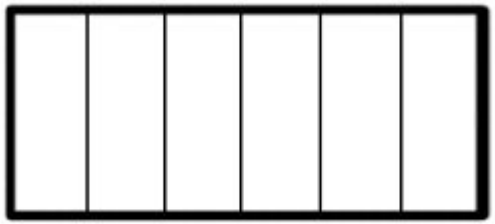
- Випущена в 1993
- Замінила EXT1 через свій дизайн “з задумкою на розширення в майбутньому” та меншу кількість багів

Історія файлової системи

- Чому назва - EXT4?
- Були попередні версії
- Інформації про EXT1 не дуже багато (скоріш за все через велику кількість проблем з імплементацією), але про EXT2 та EXT3 інформації значно більше

Блоки

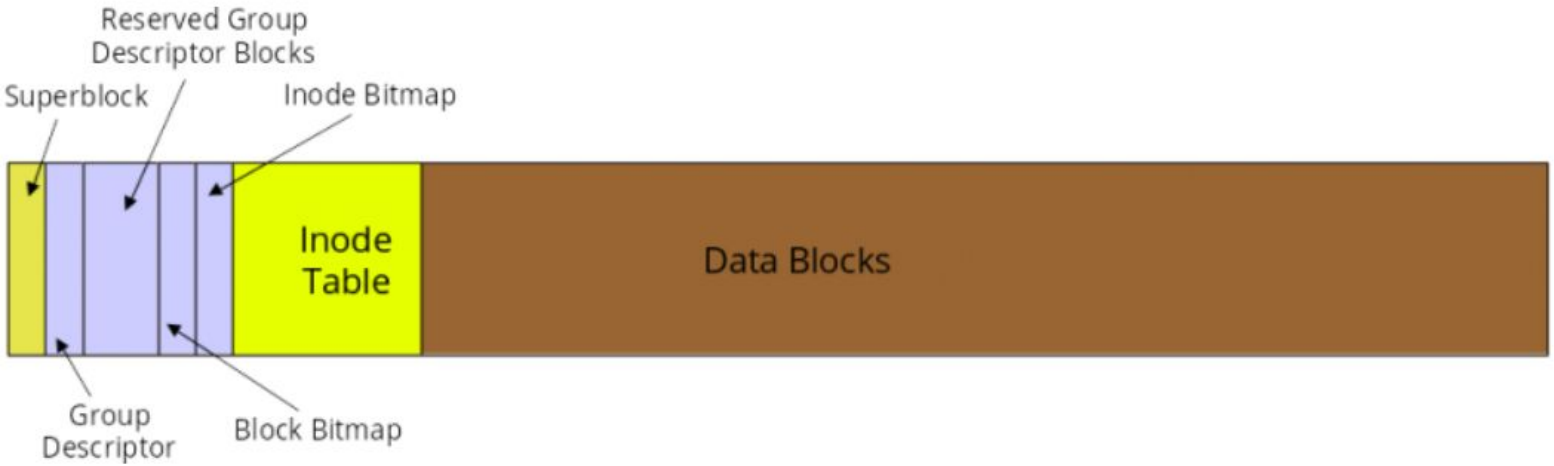
- Найменша одиниця виміру в контексті файлової системи становить блок.
- Блок - комірка пам’яті **файлової системи*** розміром в 4 кілобайти (можна налаштувати під час створення файлової системи на розділі жорсткого диску)
  Під коміркою файлової системи мається на увазі той факт, що жорсткий диск розбивки на подібні блоки не має. Жорсткий диск зазвичай розбитий на ***сектори (також мають назву логічні блоки)****.*



Групи блоків

- Блоки, в свою чергу, зібрані в групи. Файлова система намагається зібрати блоки, які відносяться до одного файлу, в одну групу, для зменшення кількості пошуків відповідної комірки пам’яті на диску.
- В EXT2 група блоків може максимум вміщати в себе 8 МБ даних.
- Також в групі блоків мають зберігатися метадані - де шукати конкретні файли (inode структури), розміри/права/назви файлів і т.п.

Структура групи блоків



EXT3 та журнал

- Основна ціль 3 версії - покращити час відновлення системи при аварійному виключенні
- В EXT2 цей процес (команда ***$ fsck***) міг займати години, а іноді і дні...
- Для більш швидкого відновлення був доданий так званий **журнал**

EXT3 та журнал

- Всі дані, які мають потрапити до файлової системи, спочатку потрапляють до журналу, який знаходиться на спеціально виділеній частині жорсткого диску.
- Після запису до журналу, дані переносяться з нього власне до файлової системи.
- Робиться це особливим чином, щоб якщо аварійне відключення відбудеться під час переносу даних з журналу, “пошкоджені” блоки були виправлені під час наступного запуску системи.
- Додання журналу зменшує швидкість запису в систему, але підвищує безпеку

Види ведення журналу

- Журнал має 3 режими, в залежності від того наскільки сильно ви хочете захистити дані на диску від подібної ситуації.
- **Journal -** в журнал пишуться як метадані, так і дані. Зменшується швидкість запису до файлової системи через 2 копіювання, але система більш стійка до аварійних відключень
- **Ordered** - дефолтний режим в більшості дистрибутивів Linux. Строгий порядок операцій - метадані в журнал, дані в файлову систему, метадані в систему. Файли з даними, в які йшов запис, можуть залишитися пошкодженими, але файли в які запис не йшов та метадані залишаться цілими, що означає, що система залишиться цілою
- **Writeback** - коли важлива ефективність, операції виконуються в тому порядку, який гарантує найбільшу швидкість запису до файлової системи. Метадані все ще записуються в журнал - система буде працювати навіть після збою. Тим не менш, нема гарантії, що дані, записані **після** або навіть **до** збою будуть цілі.

Проблеми EXT3

Головні проблеми EXT3:

- Індекс блоку ФС мав обмеження в 32 біти. Враховуючи розмір блоку в 4КБ, отримуємо, що EXT3 може підтримувати **лише 16 ТБ** даних (в 2004 році вже можна було вільно купити HDD на 1 ТБ, що казати про enterprise-level сервери).
- Обмеження в 32000 піддиректорій
- Виділених inode-структур недостатньо, хоча місце під дані на диску все ще є
- One-by-one-block-allocation -> fragmentation
- And others… (немає на все часу :( )

EXT4 to rescue!

- Головна проблема EXT3 - 32-бітові номери блоків. В EXT4 номер блоку став 48-бітним
- Але чому не 64 біти?

Степенева функція наносить удар у відповідь

Навіть з 48 бітами, EXT4 може підтримувати 1EB (2^{60} байтів) даних.

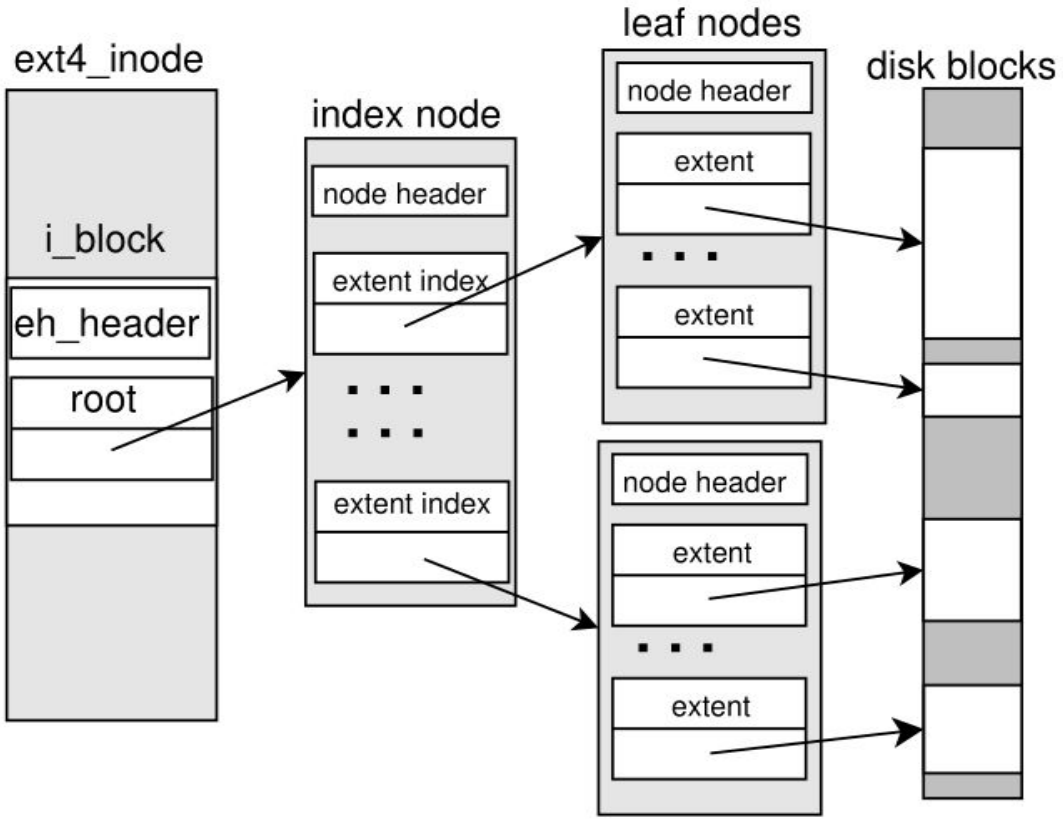
Один запуск fsck для такої системи (навіть з оптимізаціями EXT4) займе 119 років. Тепер помножте це ще на 65536 якщо використовувати 64-бітний індекс блоку…

Біда не приходить одна

Inode-структури, котрі виконують маппінг назви файлу до його місця на диску. в EXT3 використовували наївну імплементацію, яка зберігала номери блоків де зберігаються дані.

Замість цього, в EXT4 використовуються так звані extents, котрі дозволяють зберігати перший та останній блок даних, які зберігають вміст файлу (якщо файл фрагментований або більше 128МБ, то, звісно, треба буде більше ніж 1 extent),

Якщо в EXT3 використовувалися дерева блоків (trie глибиною 3), то в EXT4 використовуються дерева extents.


ext4_inode
i_block
eh_header
root
index node
node header
extent index
extent index
leaf nodes
node header
extent
extent
node header
extent
extent
disk blocks

Багато піддиректорій

В EXT4 прибрали обмеження в 32000 піддиректорій - тепер їх можна створювати скільки забажаєш. Для підтримки великої кількості директорій було імплементовано H-дерево - підвид B-дерева, який використовує 32-бітні хеші. Результат - пришвидшення роботи від 50 до 100 разів порівняно з списком з EXT3.

Inode-проблема

На жаль, EXT4 не розв’язує проблему динамічного генерування inode-структур, тобто можливий варіант коли місце на диску ще є, але всі структури вже зайняті (наприклад, коли система забита невеликими файлами, бо кожна inode відповідає за 1 файл).

В науковій роботі по EXT4 пропонується декілька підходів щодо вирішення цієї проблеми, і ми закликаємо глядачів прочитати цю роботу для кращого розуміння файлової системи.

Алгоритми виділення блоків в EXT4

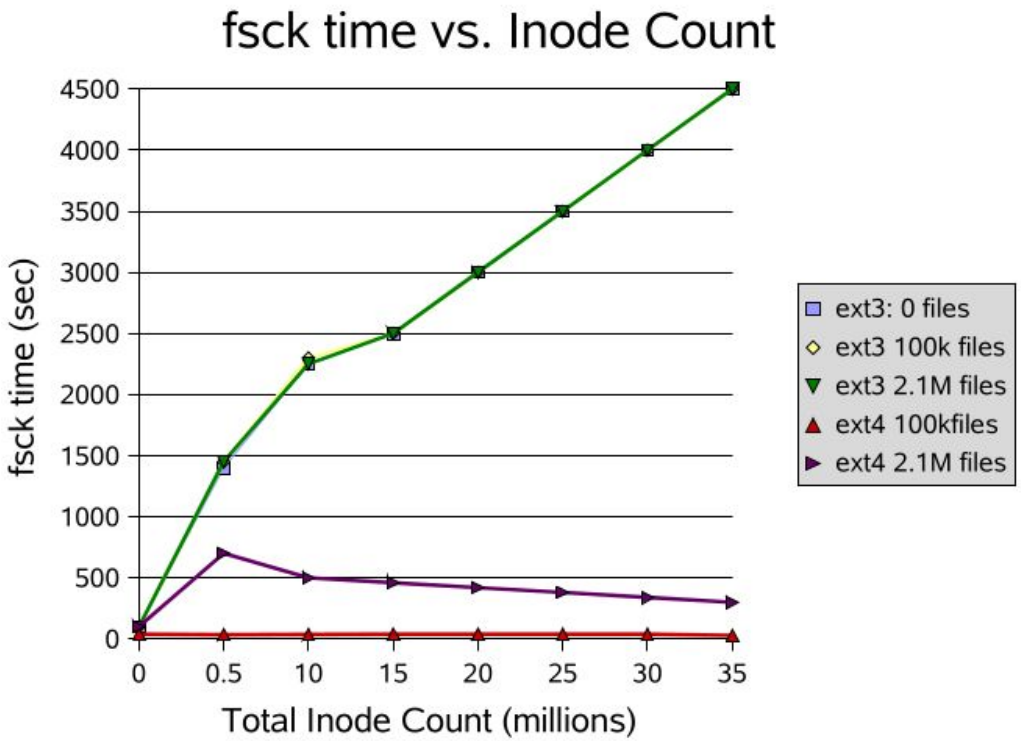
- Для кожного файлу виділяється додатковий простір “для дописування” щоб зменшити можливу фрагментацію в майбутньому
- В EXT3, блоки для дописування виділялися по одному, що було затратно в плані CPU та фрагментації файлів. EXT4 об’єднує багато запитів на виділення блоків під запис в один, що дозволяє оптимально записати їх на жорсткий диск в плані лінійності пам’яті, зменшити навантаження на CPU та не виділяти блоки під недовговічні файли, котрі можуть жити в RAM.

fsck стає ще швидше

Для перевірки того, чи справно працює система, треба обійти всі inode структури. Але єдине місце, де зберігається інформація про те, чи використана inode - це бітмапа. Лінійна перевірка - це занадто повільно для великої кількості inode (їх дійсно може бути багато).

EXT4 зберігає всі невикористані inode в кінці таблиці inode, таким чином, під час перевірки, можна не питати в бітмапи, чи використовується inode. Замість цього, можна просто зупинитися, як тільки натрапляєш на першу невикористану inode. Для додаткової безпеки робиться checksum таблиці.

Пришвидшення оцінюють від 2 до 20 разів.


fsck time vs. Inode Count
fsck time (sec)
4500
4000
3500
3000
2500
2000
1500
1000
500
0
0
0.5
10
15
20
25
30
35
Total Inode Count (millions)
ext3: 0 files
ext3 100k files
ext3 2.1M files
ext4 100kfiles
ext4 2.1M files

Ще трошки нового...

- Збільшення розміру inode до 256 байт (бо 128 вже майже повністю використовувалися в EXT3)
- Checksums майже усюди
- e4defrag
- Файли до 16 ТБ
- Мета-блок-групи

Міграція

EXT4 також підтримує міграцію з EXT3.

Міграція проходить в 2 етапи - зміна маппінгу з індексу блоків на extents та збільшення розміру inode до 256 байтів. Зміна на 48-бітові номери блоків була представлена як патч до EXT3.

Фінал

EXT4 задумувалася як “рішення тут і зараз до існуючих проблем”. Автор EXT4 вважає, що в майбутньому мають з’явитися більш сучасні файлові системи на заміну EXT4. Але, як бачимо, навіть в 2023 році ця ФС використовується як основна для більшості дистрибутивів Linux :)