Empowered by Innovation NEC

Outline of Ext4 File System & Ext4 Online Defragmentation Foresight

LinuxCon Japan/Tokyo 2010

September 28, 2010

Akira Fujita <a-fujita@rs.jp.nec.com>

NEC Software Tohoku, Ltd.

Self Introduction

- Name: Akira Fujita
- Company: NEC Software Tohoku, Ltd. in Sendai, Japan.
- Since 2004, I have been working at NEC Software Tohoku developing Linux file system, mainly ext3 and ext4 filesystems.

  Currently, I work on the quality evaluation of ext4 for enterprise use, and also develop the ext4 online defragmentation.

Outline

- What is ext4
- Ext4 features
- Compatibility
- Performance measurement
- Recent ext4 topics

- What is ext4 online defrag
- Relevant file defragmentation
- Current status / future plan

What is ext4

- Ext4 is the successor of ext3 which is developed to solve performance issues and scalability bottleneck on ext3 and also provide backward compatibility with ext3.

- Ext4 development began in 2006.
  Included in stable kernel 2.6.19 as EXPERIMENTAL (ext4dev).
  Since kernel 2.6.28, ext4 has been released as stable
  (Renamed from ext4dev to ext4 in kernel 2.6.28).

- Maintainers
  Theodore Ts'o tytso@mit.edu , Andreas Dilger adilger.kernel@dilger.ca
- ML
  linux-ext4@vger.kernel.org
- Ext4 Wiki
  http://ext4.wiki.kernel.org

Ext4 features

Ext4 features

Bigger file/filesystem size support.

Compared to ext3, ext4 is:
8 times larger in file size, 65536 times(!) larger in filesystem size.

Filesystem	Max. file size	Max. filesystem size
ext3	2TB	16TB
ext4	16TB	1EB

- I/O performance improvement: delayed allocation, multi block allocator extent map and persistent preallocation.
- Fast fsck: flex_bg and uninit_bg
- Reliability: journal checksumming
- Maintenance: online defrag
- Misc: backward compatibility with ext2/ext3, nanosec timestamps, subdir scalability, etc.

Feature flags

Feature flags of ext filesystems

	Ext2 features (common)	Ext3 features	Ext4 features
Ext4	ext_attr resize_inode dir_index filetype sparse_super	has_journal	huge_file uninit_bg dir_nlink extra_isize **extent**[1] **flex_bg**[1]

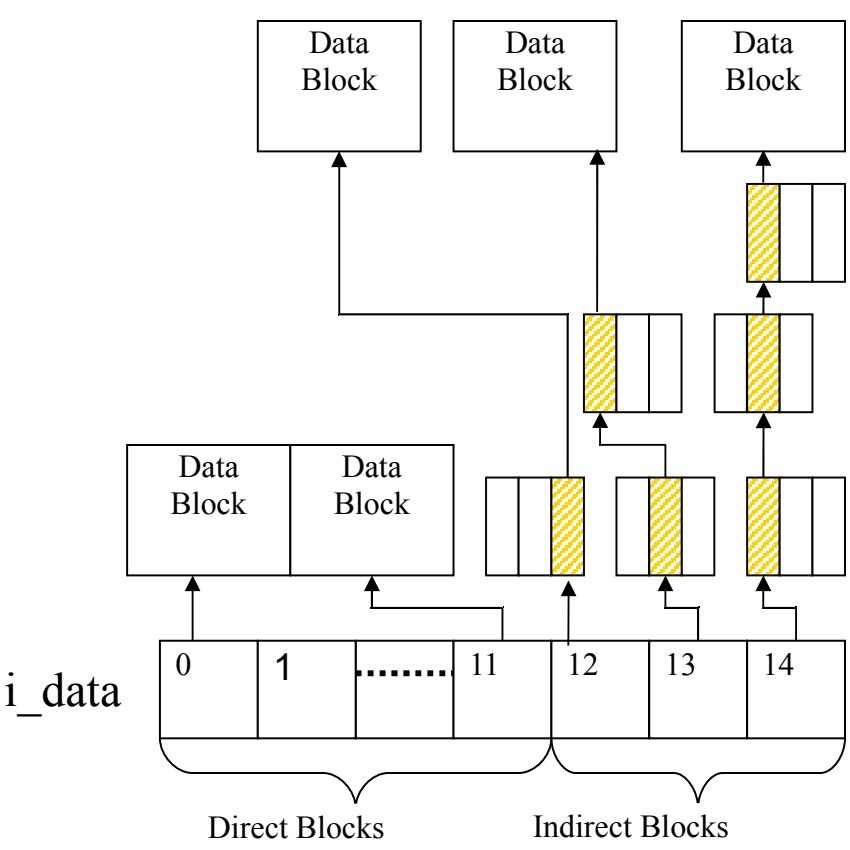
[1] unremovable feature flag

Note: Feature flags are enabled/disabled by tune2fs, except for flex_bg (only by mkfs).

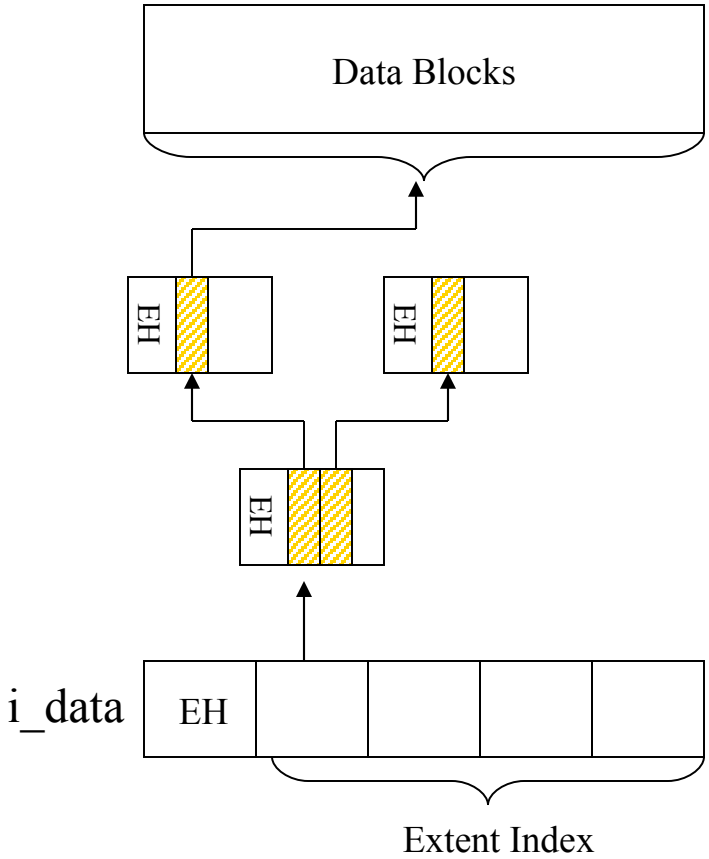
If you set uninit_bg and dir_index feature flags, you need to run e2fsck with –pD options to enable them.

Indirect Block Map and Extent Map

- Ext4 supports two block maps. Extent map is more efficient and can handle large file in comparison with indirect block map.



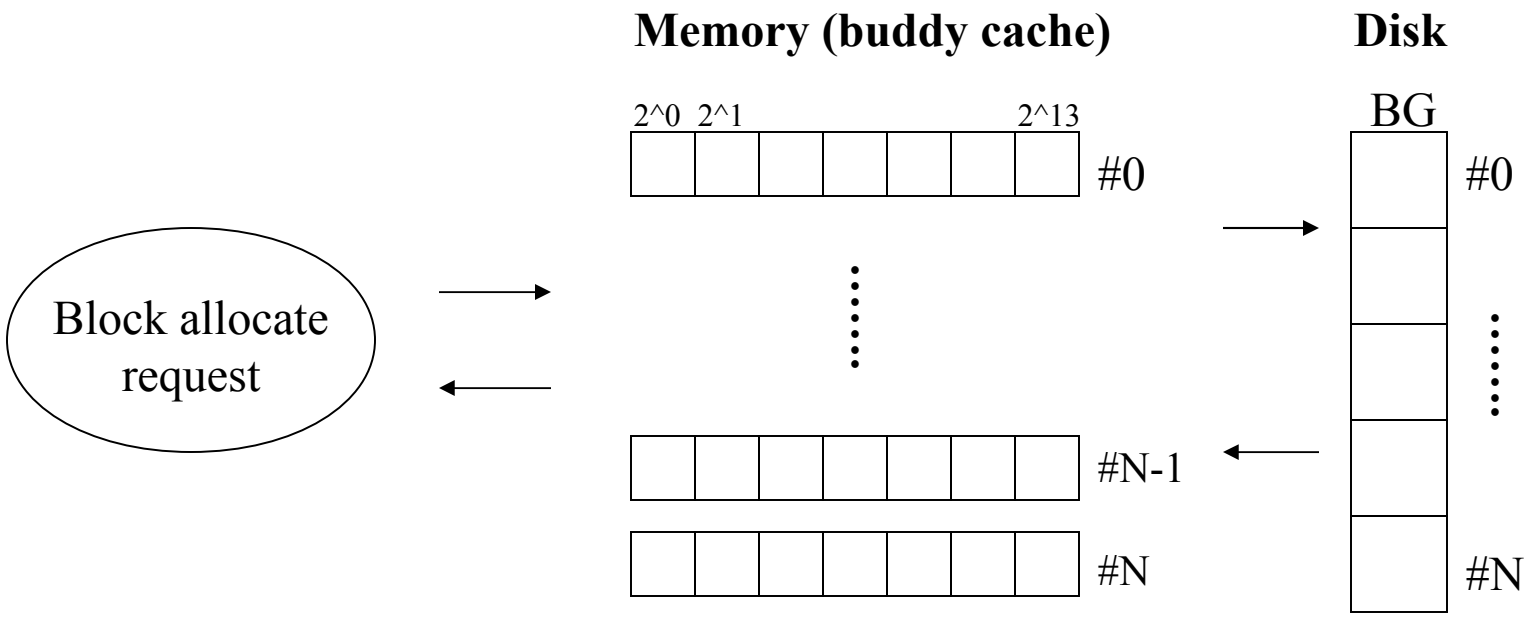
Indirect Block Map



Extent Map

Multiple block allocator

- Contiguous multiple blocks are allocated at once to prevent file fragmentation. This decreases CPU utilization and seek time.
- Contiguous free blocks of block group are managed by the buddy system in memory (2^0-2^13).

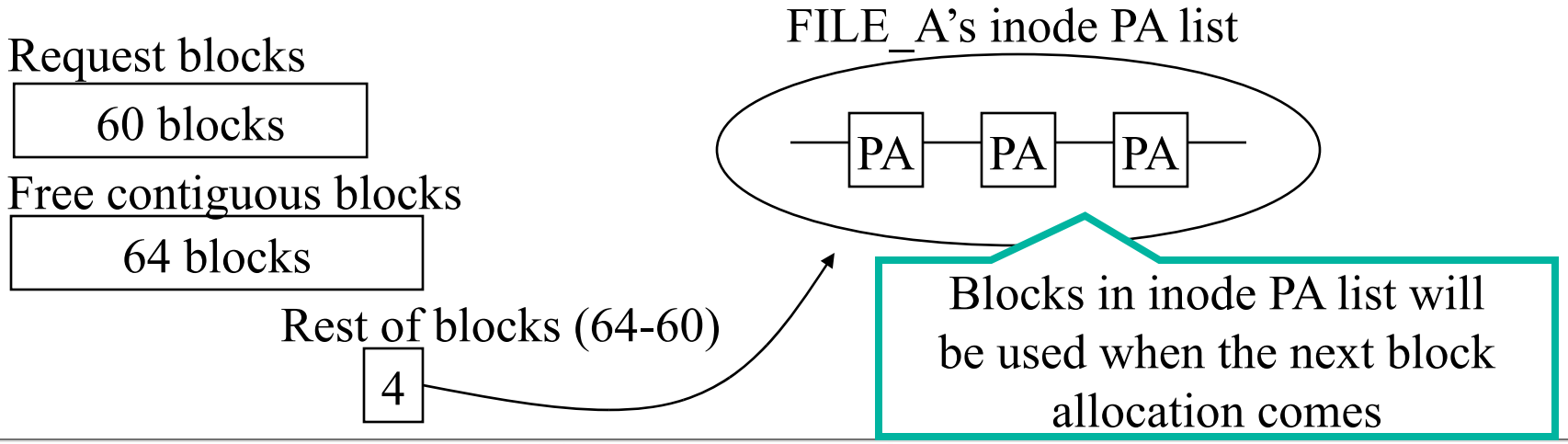


Multiple block allocator (cont.)

- Blocks unused by the current allocation are added to inode preallocation.
- Inode preallocation enables blocks will be assigned preferentially when the next block allocation comes. Consequently contiguous multiple blocks are used.
- For a file smaller than 16 blocks is added to the locality group[2] instead.

[2] Locality group is defined by a CPU the allocation is running on to pack small files together

e.g. Allocate 60 blocks to FILE_A, rest of 4 blocks are added to inode preallocation(PA) list.



Delayed allocation

- Delayed allocation is used by write system call. It delays real block allocation until written data is flushed from memory to disk.
- Prevent file fragmentation and decrease CPU utilization.
- Suppress writes to temporary files which exist for short period.

- There is a trade-off between performance and reliability.

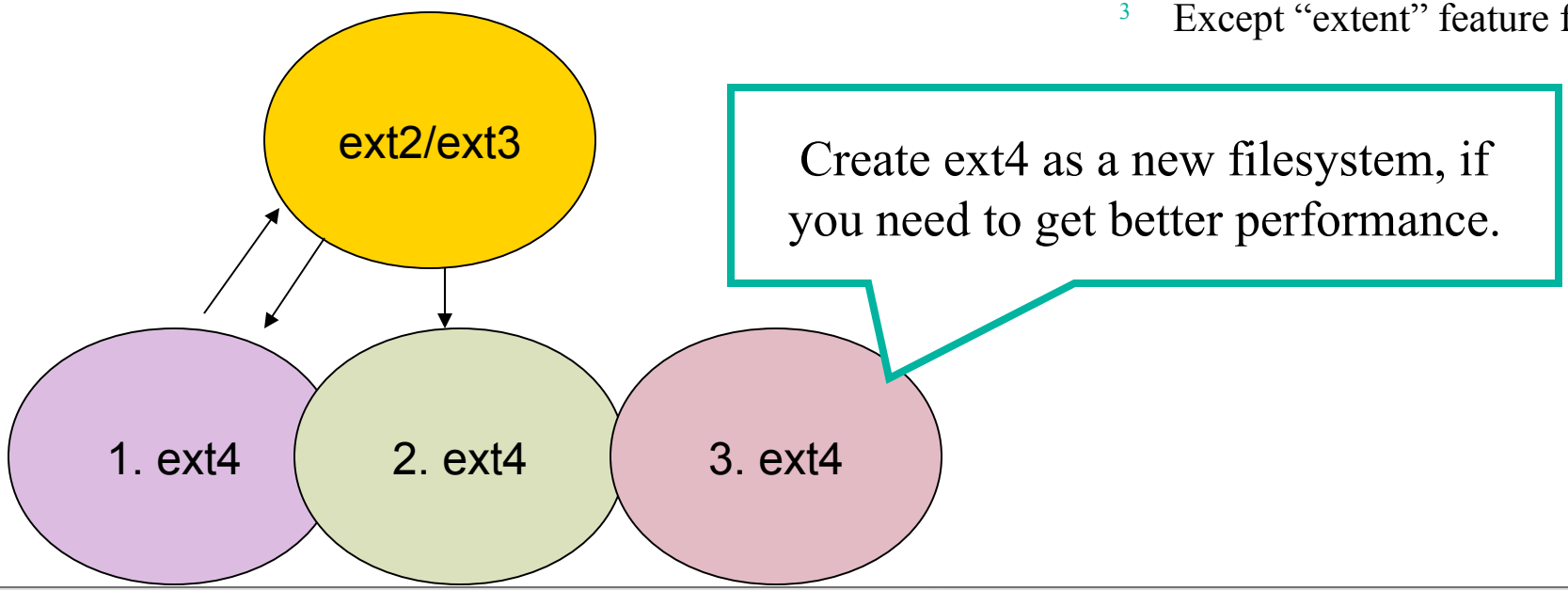
There is a known-issue on data loss. If a crash occurs when created or truncated file is closed, or a file is renamed to replace the previous file, data may be lost. To avoid the issue, mount ext4 filesystem with “noauto_da_alloc” option.

Compatibility

There are 3 types of ext4. Compatibility differences of them are shown below:

Type	Method to mount ext4 filesystem	Compatibility
1	Mount ext2/3 as ext4 (just do mount –t ext4)	○
2	Mount ext2/3 as ext4 with turning on feature flags	○ [3]
3	Create ext4 with mke2fs as new filesystem	×

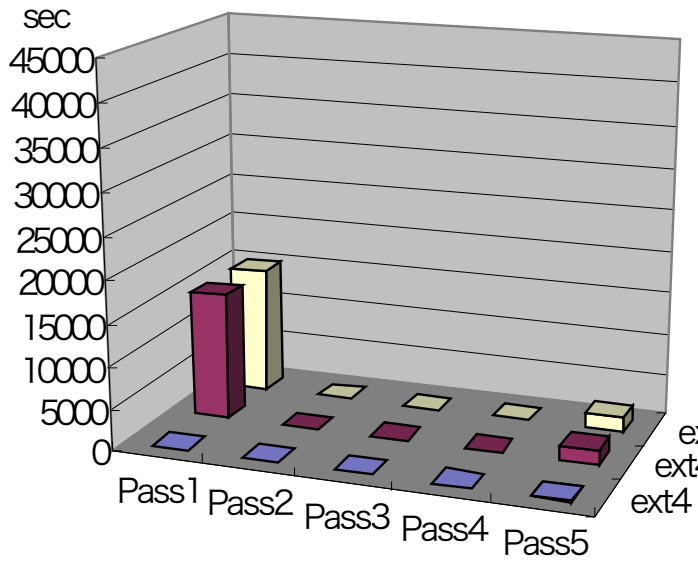
[3] Except “extent” feature flag



Performance measurement

mkfs / fsck time with large filesystem

e2fsck time on empty 16TB filesystem



e2fsck for ext3 and ext4(^extent, ^uninit_bg, ^flex_bg) takes about 4.5 hours.

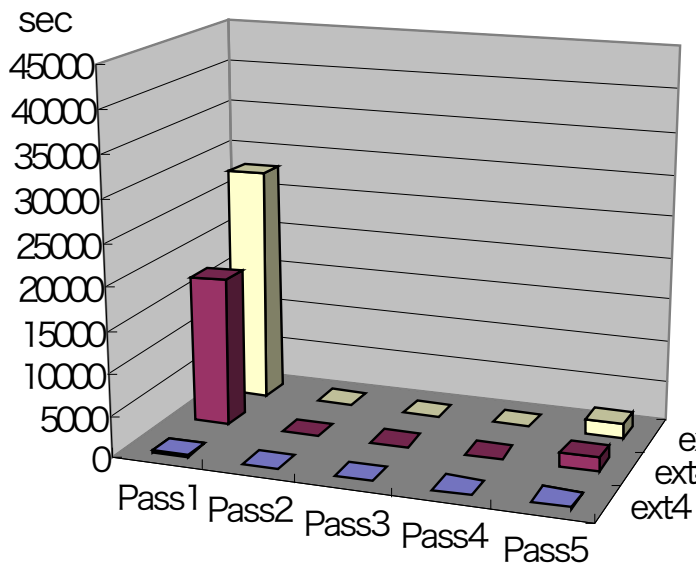
	Pass1	Pass2	Pass3	Pass4	Pass5
ext4	1.82	0.25	0.08	9.17	83.51
ext4(^extent,^unint_bg,^flex_bg)	15125.72	1.33	0.07	9.17	169.71
ext3	1506.03	0.82	0	18.35	168.29

Kernel: 2.6.35, e2fsprogs: 1.41.12, Arch: x86_64, CPU: Xeon 3.00GHz, Memory: 3.5GB

“^” means without feature flag

mkfs / fsck time with large filesystem

e2fsck time on 16TB filesystem using 4TB (10MB x 4096000files)



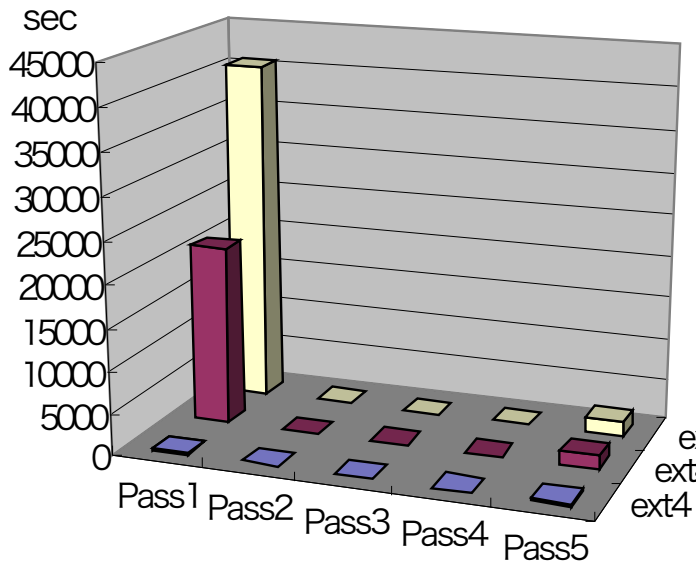
As file size gets bigger, ext3 fsck takes longer

	Pass1	Pass2	Pass3	Pass4	Pass5
ext4	59.57	40.89	0.08	9.19	149.15
ext4(^extent,^unint_bg,^flex_bg)	17741.86	16.04	0.07	9.18	1659
ext3	28020.18	46.42	0.15	18.36	1690.71

Kernel: 2.6.35, e2fsprogs: 1.41.12, Arch: x86_64, CPU: Xeon 3.00GHz, Memory: 3.5GB

mkfs / fsck time with large filesystem

e2fsck time on 16TB filesystem using 8TB (10MB x 8129000files)



Ext3 fsck takes about 12 hours!!

Ext4 is extremely faster than others.

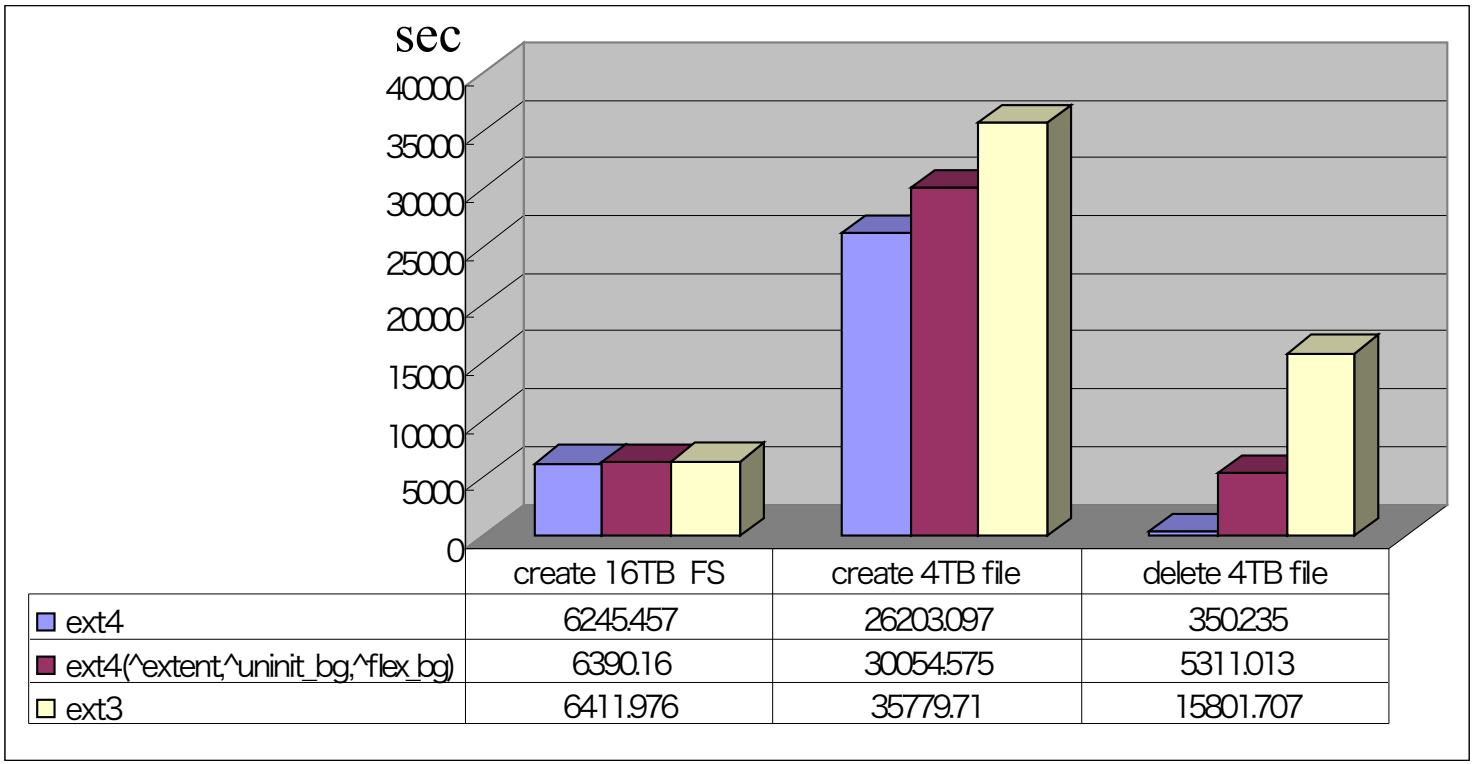
	Pass1	Pass2	Pass3	Pass4	Pass5
ext4	117.74	78.06	0.08	9.2	215.58
ext4(^extent,^unint_bg,^flex_bg)	21230.18	31.69	0.07	9.19	1665.94
ext3	4074.056	91.98	0.15	18.38	1743.44

Kernel: 2.6.35, e2fsprogs: 1.41.12, Arch: x86_64, CPU: Xeon 3.00GHz, Memory: 3.5GB

uninit_bg feature influences fsck performance a lot.

Misc operation time with large filesystem

- File create: dd if=/dev/zero of=fille bs=1048576 count=10
- File delete: rm –rf *



	create 16TB FS	create 4TB file	delete 4TB file
ext4	6245.457	26203.097	350.235
ext4(^extent,^uninit_bg,^flex_bg)	6390.16	30054.575	5311.013
ext3	6411.976	35779.71	15801.707

- There is no difference on filesystem creation time (1h)
- Ext4 gets better performance on file creation/deletion than ext3

FFSB Benchmark Result

Test Environment:

Arch: x86_64

Kernel: 2.6.35 (default mount option)

CPU: Intel(R) Core(TM)2 CPU 6320 @ 1.86GHz

Memory: 2GB

IO scheduler: CFQ

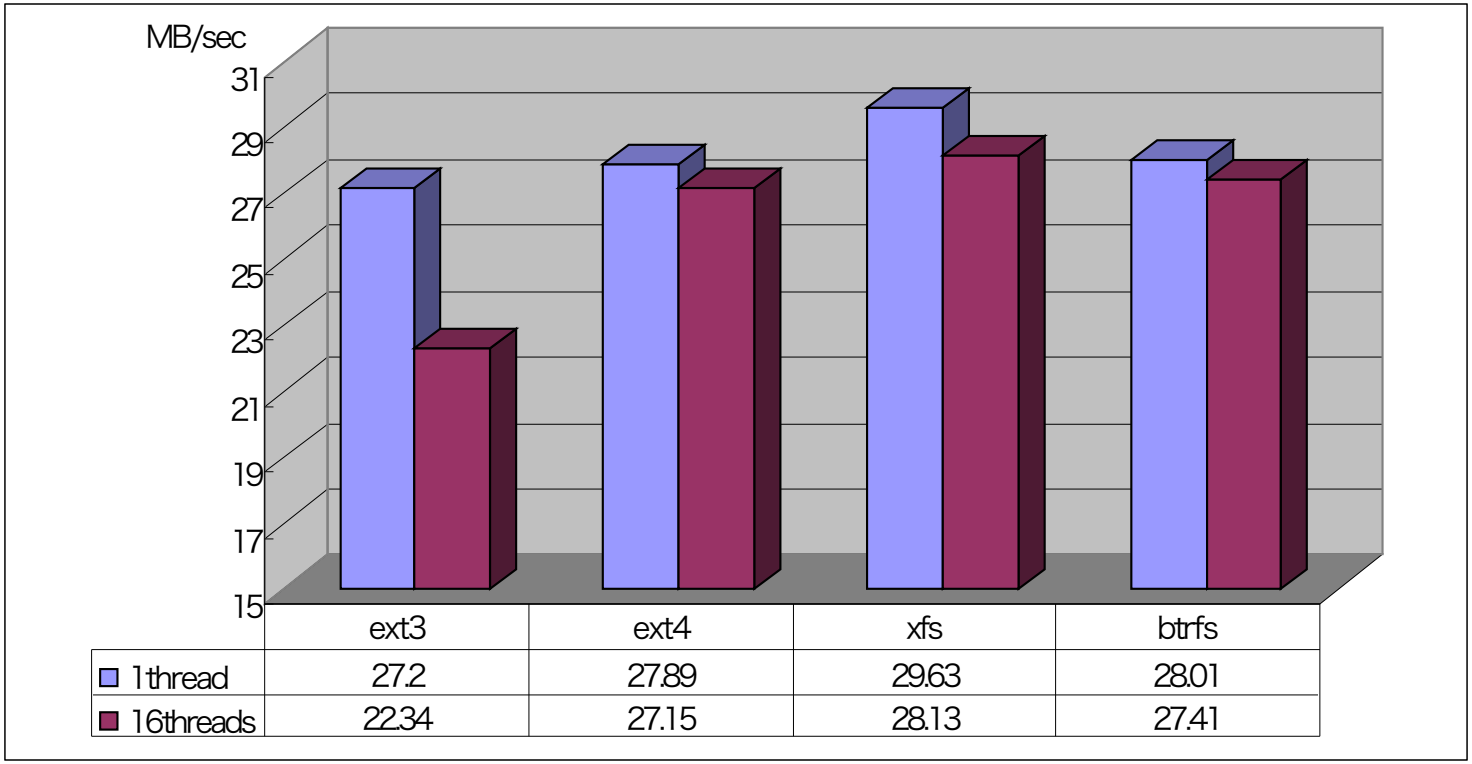
Device: ST3500641AS(500GB), 3.AAB, Serial ATA 2

TP: The Flexible Filesystem Benchmark (FFSB) 6.0-rc2

Target filesystem: ext3, ext4, xfs and btrfs (xfs and btrfs are for reference).

Large file sequential write using FFSB

- 1GB sequential write with1thread and 16threads (average of 10 trials)

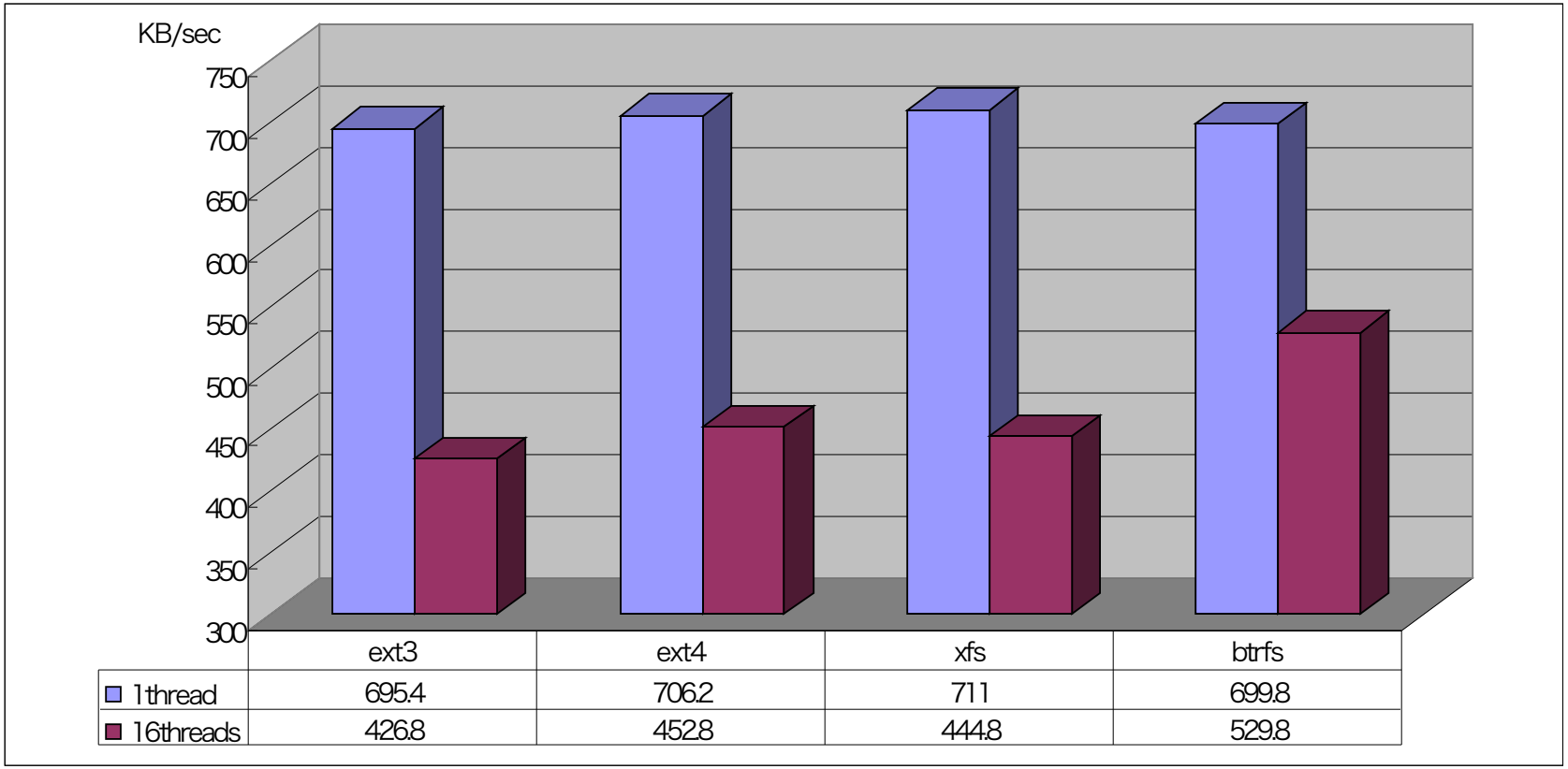


	ext3	ext4	xfs	btrfs
1thread	27.2	27.89	29.63	28.01
16threads	22.34	27.15	28.13	27.41

Write performance on ext4 is 21.5% greater than ext3 in case of 16thread.

Large file random read using FFSB

- 100MB x 1024files random read with1thread and 16threads (average of 10 trials)



	ext3	ext4	xfs	btrfs
1thread	695.4	706.2	711	699.8
16threads	426.8	452.8	444.8	529.8

Read performance of ext4 is slightly better than ext3

Write performance with typical ext4 options

Large file sequential write using FFSB with typical ext4 options

1GB sequential write with 1thread (average of 3 trials)
Each measuring time is 5 minutes.

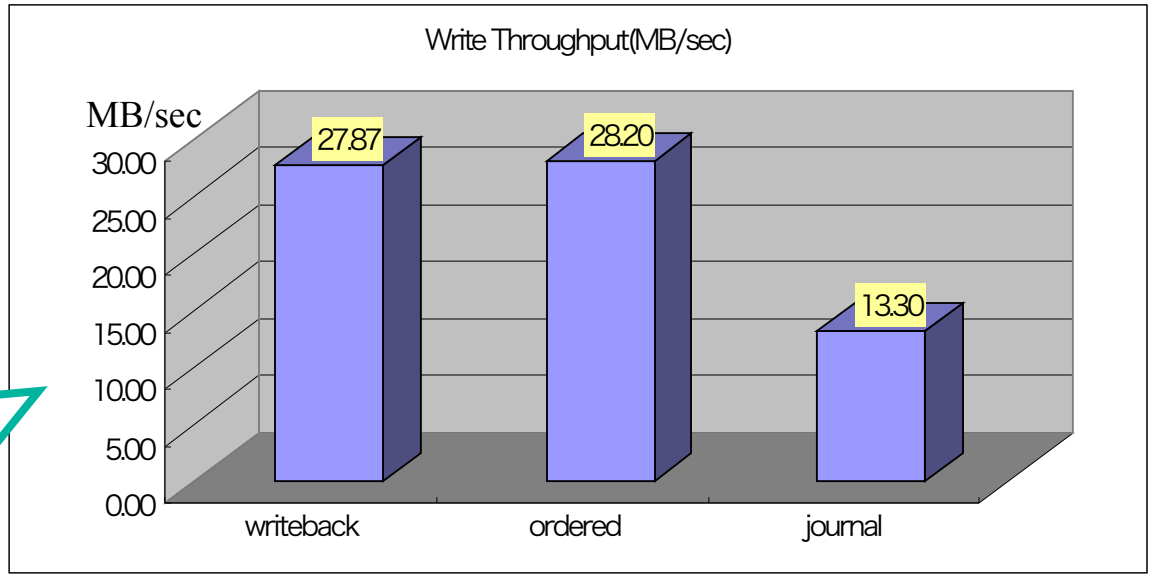
	Mount option (except for block map)
Journal mode	writeback, ordered (default), journal
Delayed allocation	delalloc (default), nodelalloc, noauto_da_alloc
Barrier	barrier=1 (default), barrier=0
Block map [4]	extent (default), indirect

[4] Block map is not tuned with mount option

Write performance with typical ext4 options

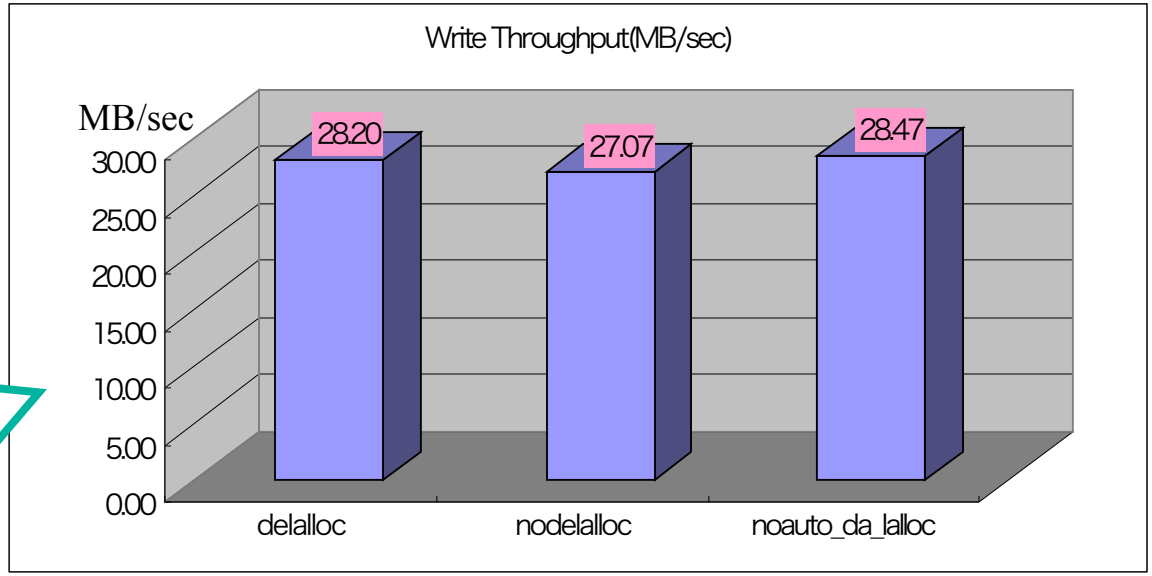
Journal mode

When “journal” mode, is used, twice amount of data (metadata + data) is written.



Delayed allocation

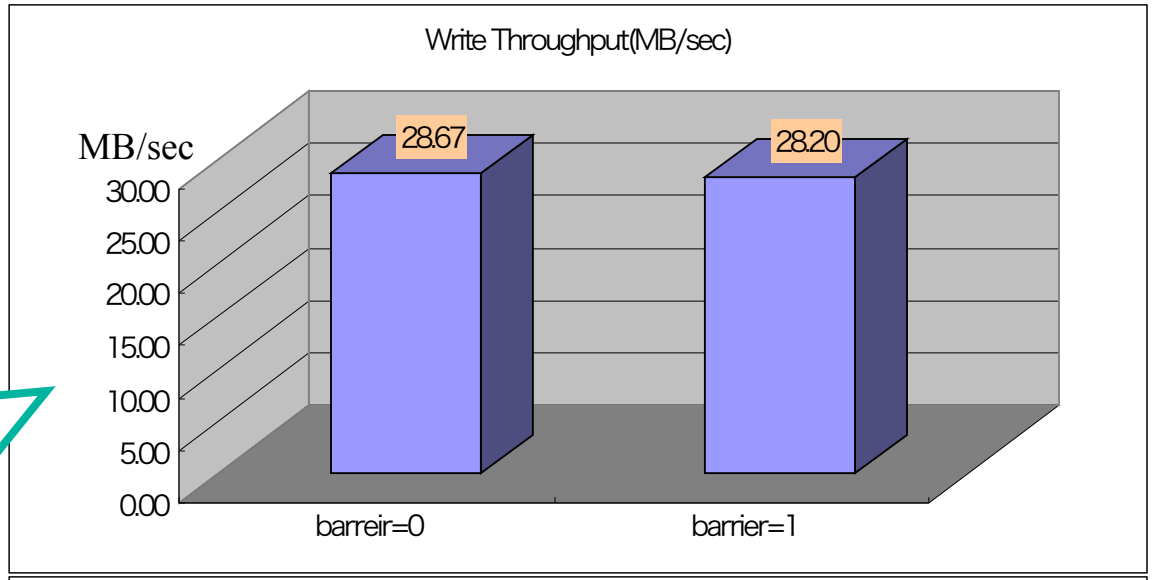
Write throughput performance with dealloc is 4.1% greater than nodelalloc.



Write performance with typical ext4 options

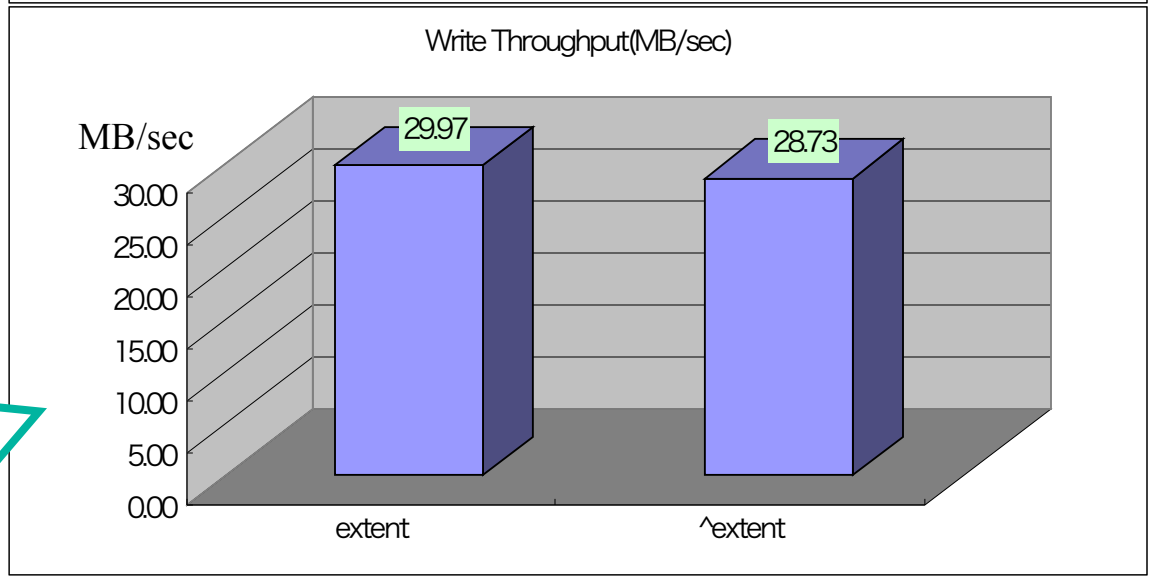
Barrier

Using barrier makes write performance slower. In this case, it's comparable.



Block map

Using extent map is 4.2% faster than using indirect map.

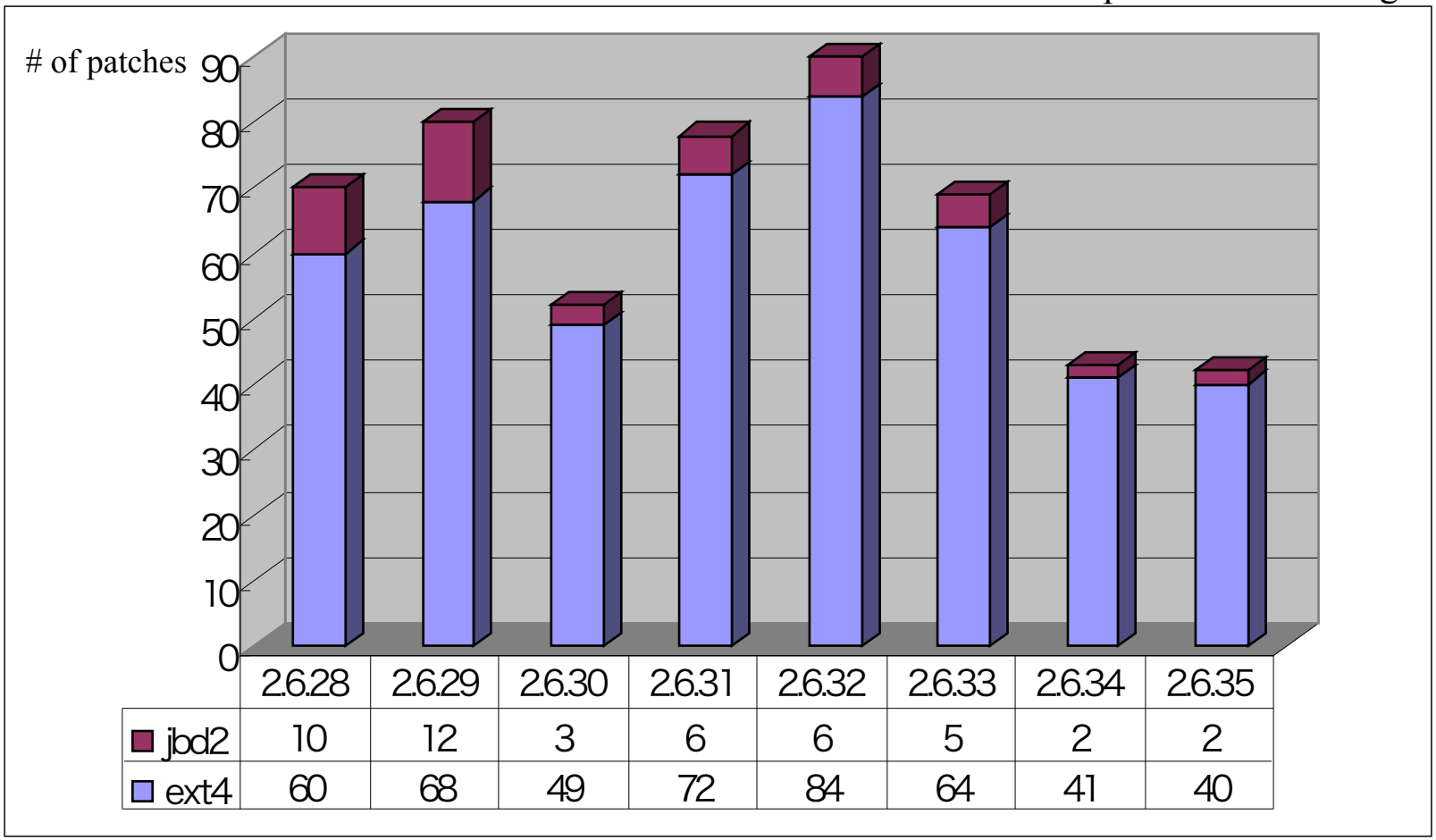


Ext4 performance conclusion

- Read and write performance of ext4 is faster than ext3, especially in the case of using multi thread operation.
- Ext3 fsck time on the large filesystem takes a so long time that it is useless. Ext4 solves this issue. (ext3: 12h ext4: 7min)
- There is no difference on filesystem creation time between ext3 and ext4. (1.5h)
- Default mount option of ext4 gives better performance. In addition, there is room for the consideration on trade-off between performance and reliability.

Transition of ext4 patches

The Linux Kernel Archives: http://www.kernel.org/



	2.6.28	2.6.29	2.6.30	2.6.31	2.6.32	2.6.33	2.6.34	2.6.35
jbd2	10	12	3	6	6	5	2	2
ext4	60	68	49	72	84	64	41	40

In the point of view of the transition, many bug fixes and feature additions have been merged so far. Recently it shows tendency that number of patches become slightly a few.

Recent ext4 topics

- SSDs
- Reduce mkfs time
- Snapshot for ext4

etc.

ext4 TODO list: https://ext4.wiki.kernel.org/index.php/TODO_list

Ext4 online defragmentation

What is ext4 online defragmentation

- Ext4 online defrag solves fragmentation on mounted ext4 filesystem.

- EXT4_IOC_MOVE_EXT ioctl which exchanges blocks between two inodes and e4defrag command have been merged into Linux kernel and e2fsprogs each so far.

Therefore we can solve single file fragmentation on current ext4 with following steps :

1. Create a donor file
2. Allocate contiguous blocks to donor file
3. Call EXT4_IOC_MOVE_EXT ioctl to exchange data blocks between target file and donor_file
4. Remove donor file

Relevant file defragmentation

- e4defrag -r solves relevant file fragmentation.

  Usage: e4defrag -r directory...| device...

  Relevant file defrag is solved by moving related files under specified directory closer together on ext4 filesystem.

- e4defrag -r uses new ioctl EXT4_IOC_COTROL_PA[5] to set inode preallocation (PA) to specified inode.
- If inode has inode PA, block allocator tries to use blocks from there.

[5] EXT4_IOC_COTROL_PA is new ioctl which gets free blocks from specified physical offset and sets these blocks to the inode.

Types of fragmentation

There are three types of fragmentation.

1. Single file fragmentation

   Single file fragmentation occurs when a single file is broken into multiple pieces. This decreases the performance of accessing a single file.

2. Relevant file fragmentation

   Files frequently accessed by an application at the same time, are stored to different physical locations individually on ext4. It makes application performance slower especially in case of small file access.

3. Free space fragmentation

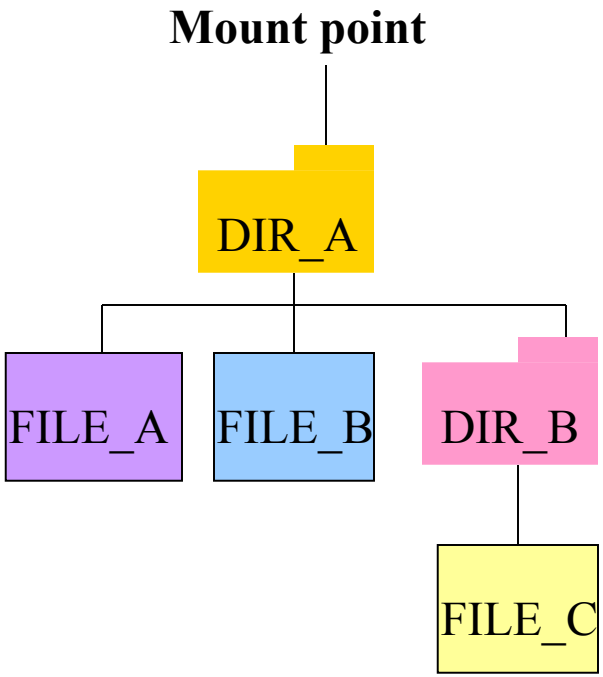
   Free space fragmentation occurs when filesystem has many small free areas and there is no large free area which consists of contiguous blocks.

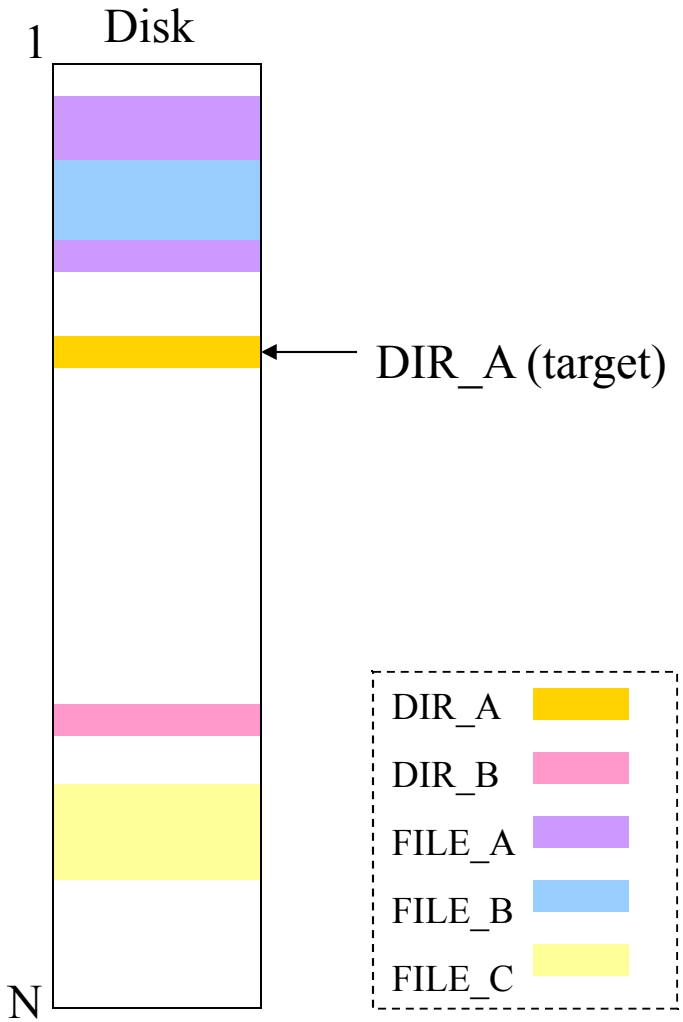
Detail of relevant file defragmentation

1. Execute relevant file defrag (e4defrag –r) to DIR_A. by issuing the following command.

   # e4defrag -r DIR_A

   Files under DIR_A are defraged in order (FILE_A, FILE_B, FILE_C).

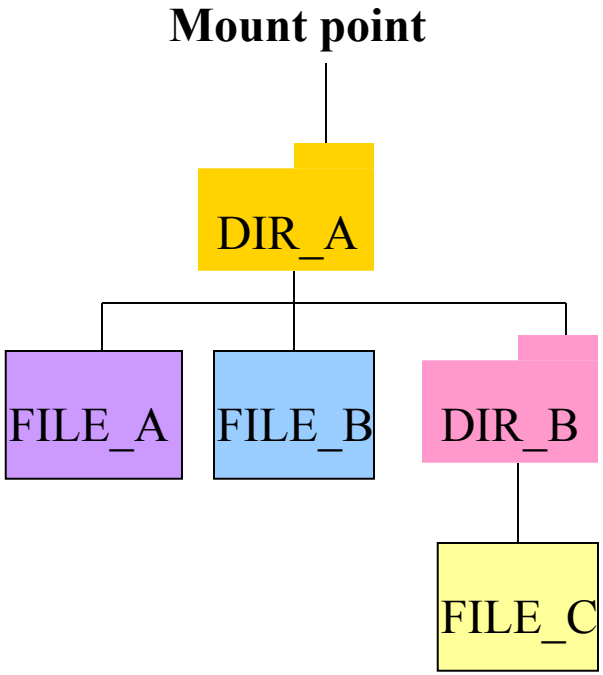


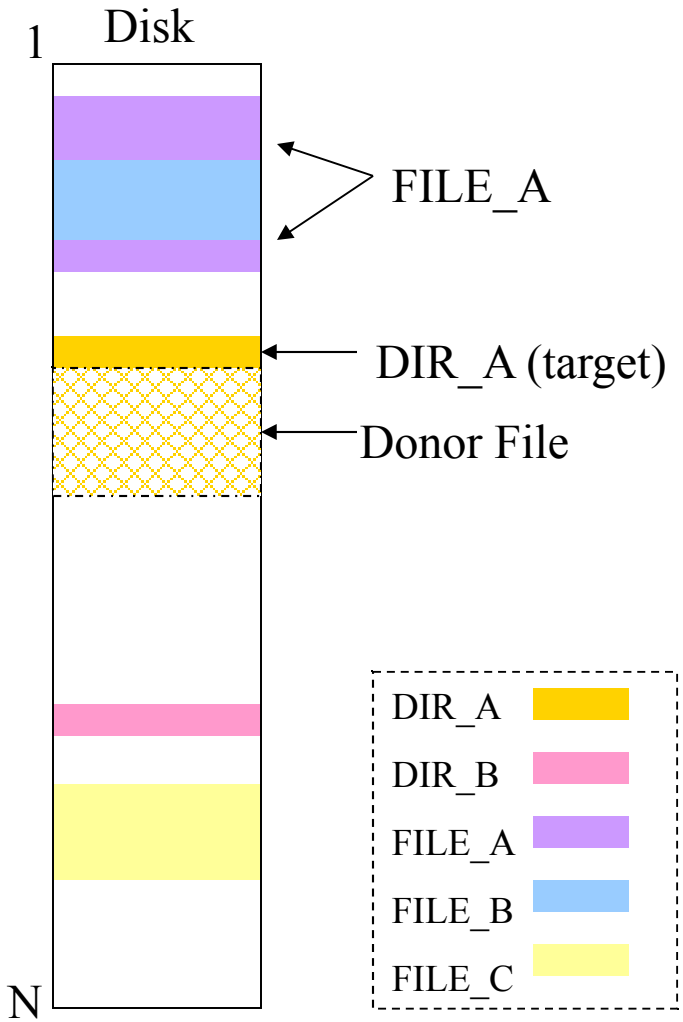


Detail of relevant file defragmentation (cont.)

For example, target file is FILE_A.

2. e4defrag creates a donor file and allocate inode PA based on physical offset of DIR_A with EXT4_IOC_CONTROL_PA to the donor file. If it is succeed, it calls fallocate to assign inode PA to the blocks from the next offset of DIR_A.

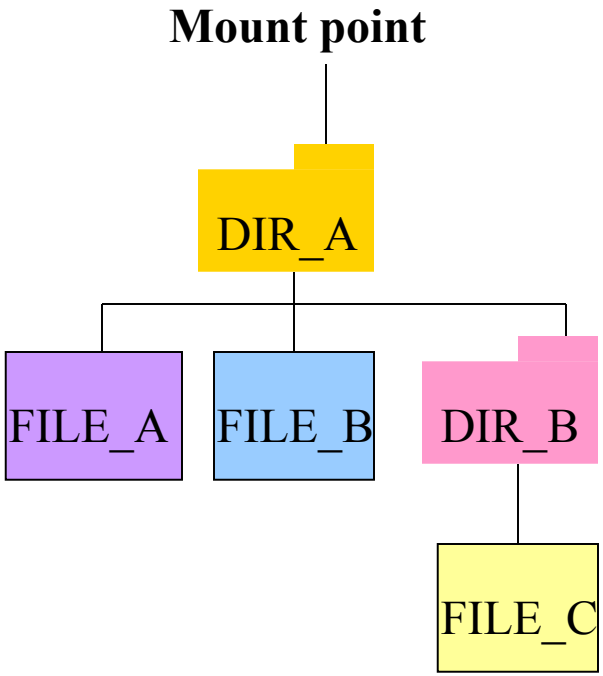


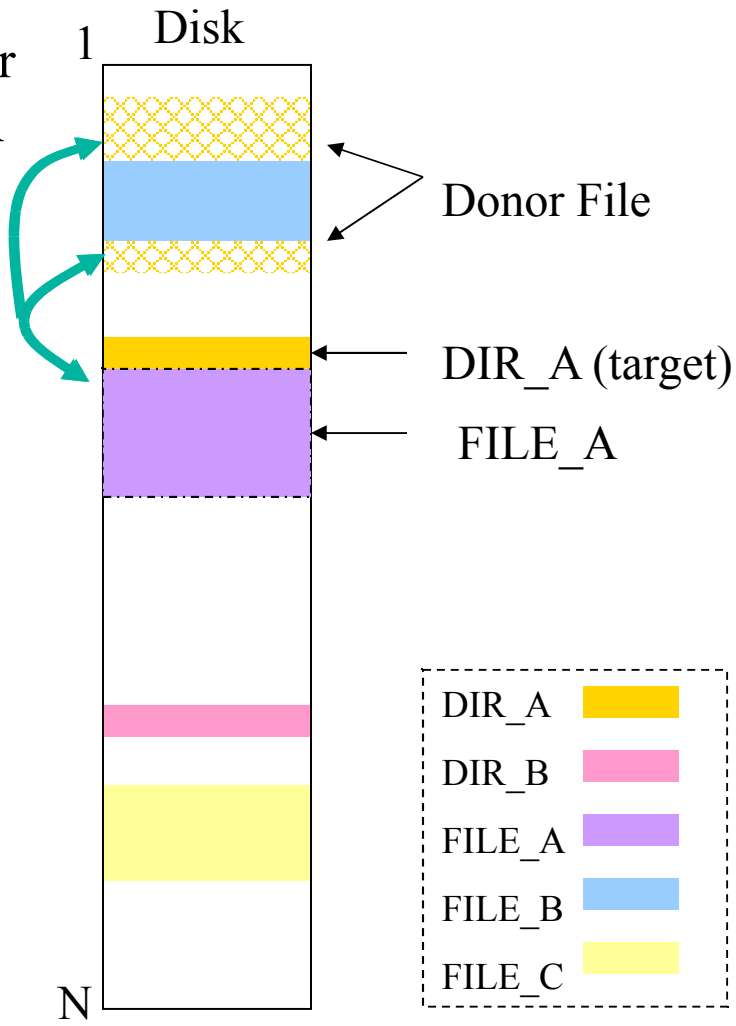


Detail of relevant file defragmentation (cont.)

3. It calls EXT4_IOC_MOVE_EXT[6] to exchange blocks between FILE_A and Donor File. As the result, data blocks of FILE_A are moved close to DIR_A.

[6] Block exchange ioctl which is used to solve single file defragmentation by e4defrag.

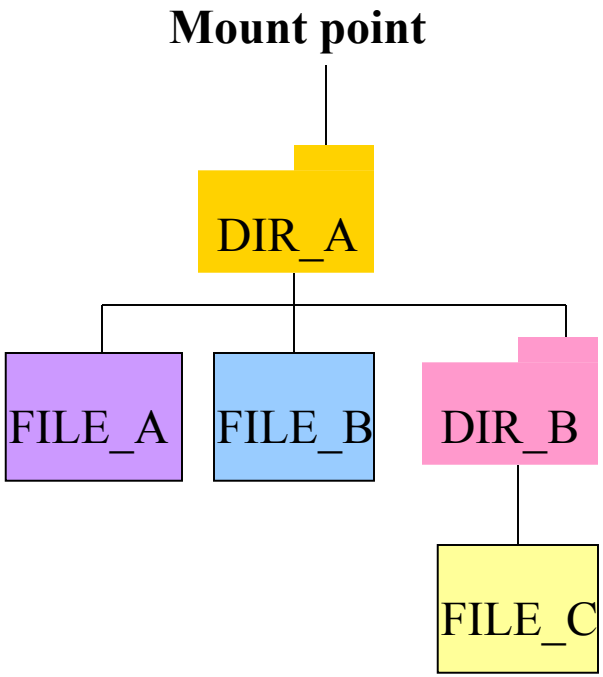


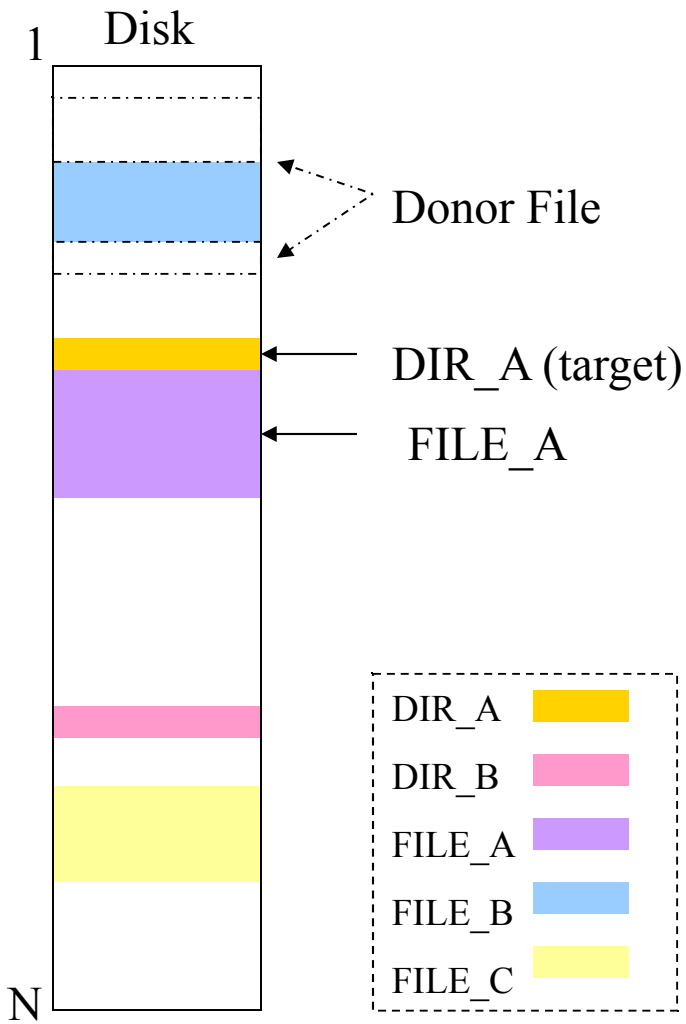


Detail of relevant file defrag (cont.)

4. Remove Donor file.

   Continue doing steps from 2 to 4 for the rest of files (FILE_B and FILE_C).



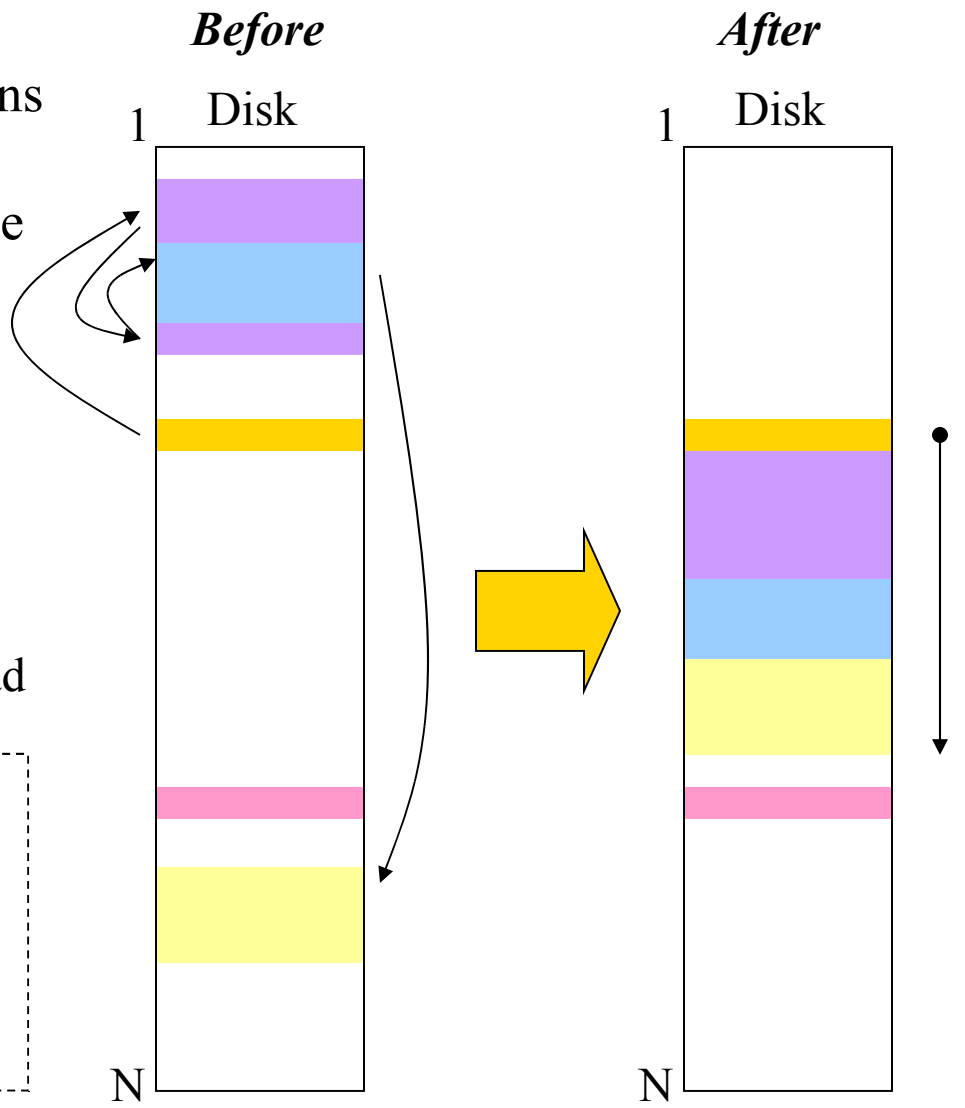


Detail of relevant file defragmentation (cont.)

5. Compare the previous physical locations of data blocks of each file with the current physical locations. Relevant file fragmentation is resolved.

   As the result, read performance of DIR_A gets improved due to enabling sequential read.

"→" is movement of disk head



Result of relevant file defragmentation

Result of relevant file defrag to kernel tree

e4defrag -r linux-2.6.35

	Before	After
Extent count	33342	33342
Read time (sec)	204.8	**164.3**
Make time (sec)	379.5	374.4

Extent count is not changed, but resolved relevant file defrag brings 25% improved read performance.

Current status / future plan

- Patches related to relevant file defrag (e4defrag -r) are under review.
- Accelerate e4defrag patches (include bug fixes) to be merged into e2fsprogs.
- e4defrag needs more feedbacks !

- Implement free space defragmentation (e4defrag -f)

  We might implement this feature when the relevant defrag development is completed.

How to start ext4 online defragmentation

- Kernel: 2.6.31 or later (EXT4_IOC_MOVE_EXT of ioctl has been merged)

- Command: e4defrag (e2fsprogs-1.41.8 or later)

  To use the latest e4defrag command, apply the following patches to the e2fsprogs git tree (http://git.kernel.org/pub/scm/fs/ext2/e2fsprogs.git).

  http://marc.info/?l=linux-ext4&m=128272678710600&w=2

- Usage: e4defrag [-v] file...| directory...| device...

  e4defrag -c file...| directory...| device...

  e4defrag -r directory...| device...

- **Any feedbacks of ext4 online defrag are welcome.**

Appendix

EXT4_IOC_CONTROL_PA (new ioctl)

EXT4_IOC_CONTROL_PA ioctl sets/discards inode PA to specified inode.

```
#define EXT4_IOC_CONTROL_PA _IOWR('f', 16, struct ext4_prealloc_info)
struct ext4_prealloc_info {
        __u64 pi_pstart;      /* physical offset for the start of the PA from
                               * the beginning of the file (in/out) */
        __u32 pi_lstart;      /* logical offset for the start of the PA from
                               * the beginning of the disk (in/out) */
        __u32 pi_len;         /* length for this PA (in/out) */
        __u32 pi_free;        /* the number of free blocks in this PA (out) */
        __u16 pi_flags;       /* flags for the inode PA setting ioctl (in) */
};
```

There are three flags for pi_flags entry

- EXT4_MB_MANDATORY — Get blocks for inode PA from specified range, if not returns error.
- EXT4_MB_ADVISORY — Get blocks from arbitrary range.
- EXT4_MB_DISCARD_PA — Discard inode PA

EXT4_IOC_GET_PA (new ioctl)

- EXT4_IOC_GET_PA ioctl gets inode PA of specified inode.

  This ioctl is used for debug.

```
#define EXT4_IOC_GET_PA _IOWR('f', 17, struct ext4_prealloc_list)

struct ext4_prealloc_list {
        __u32 pl_count;                 /* size of pl_space array (in) */
        __u32 pl_mapped;                /* number of PAs that were mapped (out) */
        __u32 pl_entries;                       /* number of PAs the inode has (out) */
        struct ext4_prealloc_info pl_space[0];      /* array of mapped PAs (out) */
};
```

Thank you for listening !