성균관대학교
SUNG KYUN KWAN UNIVERSITY(SKKU)

[REPORT]

■과 목 명: 운영체제 (SWE3004-42)

■담 당 교 수: 신동군 교수님

■제 출 일: 2022년 5월 23일

■학　　과: 수학과

■학　　번: 2016314786

■성　　명: 김호진

Chap 41. FFS

수학과 김호진 (2016314786)

1. Examine the file in.largefile**, and then run the simulator with flag** -f in.largefile **and** -L 4**. The latter sets the large-file exception to 4 blocks. What will the resulting allocation look like? Run with** -c **to check.**

in.largefile에서는 총 40개의 블록이 있는 파일을 할당하는데, 실행 결과에서 확인할 수 있듯이 각 10개의 그룹은 30개의 블록으로 구성되어 있다. 그렇기 때문에 flag -f in.largefile만을 사용하여 시뮬레이터를 실행하면 앞선 group부터 차례대로 블록을 할당하게 되고, -L 4를 추가로 사용했을 때는 large-file exception을 4개의 블록으로 설정하여 10개의 그룹에 블록을 4개씩 할당하게 된다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch41$ python3 ffs.py -f in.largefile -c

num_groups:       10
inodes_per_group: 10
blocks_per_group: 30

free data blocks: 259 (of 300)
free inodes:      98 (of 100)

spread inodes?    False
spread data?      False
contig alloc:     1

     00000000000000000000 1111111111 2222222222
     01234567890123456789 0123456789 0123456789

group inodes    data
    0 /a-------- /aaaaaaaaa aaaaaaaaaa aaaaaaaaaa
    1 ---------- aaaaaaaaaa a--------- ----------
    2 ---------- ---------- ---------- ----------
    3 ---------- ---------- ---------- ----------
    4 ---------- ---------- ---------- ----------
    5 ---------- ---------- ---------- ----------
    6 ---------- ---------- ---------- ----------
    7 ---------- ---------- ---------- ----------
    8 ---------- ---------- ---------- ----------
    9 ---------- ---------- ---------- ----------

borussen@DESKTOP-L834KLC:~/ostep_ch41$ python3 ffs.py -f in.largefile -L 4 -c

num_groups:       10
inodes_per_group: 10
blocks_per_group: 30

free data blocks: 259 (of 300)
free inodes:      98 (of 100)

spread inodes?    False
spread data?      False
contig alloc:     1

     00000000000000000000 1111111111 2222222222
     01234567890123456789 0123456789 0123456789

group inodes    data
    0 /a-------- /aaaa----- ---------- ----------
    1 ---------- aaaa------ ---------- ----------
    2 ---------- aaaa------ ---------- ----------
    3 ---------- aaaa------ ---------- ----------
    4 ---------- aaaa------ ---------- ----------
    5 ---------- aaaa------ ---------- ----------
    6 ---------- aaaa------ ---------- ----------
    7 ---------- aaaa------ ---------- ----------
    8 ---------- aaaa------ ---------- ----------
    9 ---------- aaaa------ ---------- ----------
```

2. Now run with -L 30**. What do you expect to see? Once again, turn on** -c **to see if you were right. You can also use** -S **to see exactly which blocks were allocated to the file** /a**.**

각 그룹은 최대 30개의 데이터 블록을 가지기 때문에 -L 30으로 시뮬레이터를 실행하게 되면 Large-file exception(-L)을 설정하지 않았을 때와 동일한 결과를 얻게 된다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch41$ python3 ffs.py -f in.largefile -L 30 -S -c

num_groups:       10
inodes_per_group: 10
blocks_per_group: 30

free data blocks: 259 (of 300)
free inodes:      98 (of 100)

spread inodes?    False
spread data?      False
contig alloc:     1

     00000000000000000000 1111111111 2222222222
     01234567890123456789 0123456789 0123456789

group inodes    data
    0 /a-------- /aaaaaaaaa aaaaaaaaa aaaaaaaaaa
    1 ---------- aaaaaaaaaa a--------- ----------
    2 ---------- ---------- ---------- ----------
    3 ---------- ---------- ---------- ----------
    4 ---------- ---------- ---------- ----------
    5 ---------- ---------- ---------- ----------
    6 ---------- ---------- ---------- ----------
    7 ---------- ---------- ---------- ----------
    8 ---------- ---------- ---------- ----------
    9 ---------- ---------- ---------- ----------

symbol  inode#  filename     filetype   block_addresses
/           0  /            directory  0
a           1  /a           regular    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
```

3. Now we will compute some statistics about the file. The first is something we call *filespan***, which is the max distance between any two data blocks of the file or between the inode and any data block. Calculate the filespan of** /a**. Run** ffs.py -f in.largefile -L 4 -T -c **to see what it is. Do the same with** -L 100**. What difference do you expect in filespan as the large-file exception parameter changes from low values to high values?**

Large-file exception parameter가 커질수록 많은 그룹으로 분산되지 않기 때문에 -L 값이 증가할수록 파일의 두 데이터 블록 사이의 최대 거리 또는 inode와 데이터 블록 사이의 최대 거리를 나타내는 filespan은 다음과 같이 감소한다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch41$ python3 ffs.py -f in.largefile -L 4 -T -c

num_groups:       10
inodes_per_group: 10
blocks_per_group: 30

free data blocks: 259 (of 300)
free inodes:      98 (of 100)

spread inodes?    False
spread data?      False
contig alloc:     1

     00000000000000000000 1111111111 2222222222
     01234567890123456789 0123456789 0123456789

group inodes    data
    0 /a-------- /aaaa----- ---------- ----------
    1 ---------- aaaa------ ---------- ----------
    2 ---------- aaaa------ ---------- ----------
    3 ---------- aaaa------ ---------- ----------
    4 ---------- aaaa------ ---------- ----------
    5 ---------- aaaa------ ---------- ----------
    6 ---------- aaaa------ ---------- ----------
    7 ---------- aaaa------ ---------- ----------
    8 ---------- aaaa------ ---------- ----------
    9 ---------- aaaa------ ---------- ----------

span: files
  file:         /a  filespan: 372
               avg  filespan: 372.00

span: directories
  dir:           /  dirspan: 373
               avg  dirspan: 373.00
```

```
borussen@DESKTOP-L834KLC:~/ostep_ch41$ python3 ffs.py -f in.largefile -L 100 -T -c

num_groups:       10
inodes_per_group: 10
blocks_per_group: 30

free data blocks: 259 (of 300)
free inodes:      98 (of 100)

spread inodes?    False
spread data?      False
contig alloc:     1

     00000000000000000000 1111111111 2222222222
     01234567890123456789 0123456789 0123456789

group inodes    data
    0 /a-------- /aaaaaaaaa aaaaaaaaaa aaaaaaaaaa
    1 ---------- aaaaaaaaa a--------- ----------
    2 ---------- ---------- ---------- ----------
    3 ---------- ---------- ---------- ----------
    4 ---------- ---------- ---------- ----------
    5 ---------- ---------- ---------- ----------
    6 ---------- ---------- ---------- ----------
    7 ---------- ---------- ---------- ----------
    8 ---------- ---------- ---------- ----------
    9 ---------- ---------- ---------- ----------

span: files
  file:         /a  filespan:  59
               avg  filespan:  59.00

span: directories
  dir:           /  dirspan:  60
               avg  dirspan:  60.00
```

4. Now let's look at a new input file, in.manyfiles**. How do you think the FFS policy will lay these files out across groups? (you can run with** -v **to see what files and directories are created, or just** cat in.manyfiles**). Run the simulator with** -c **to see if you were right.**

해당 FFS 정책에서는 동일한 디렉토리에 있는 파일의 inode 및 데이터 블록을 동일한 그룹에 두어, 3개의 그룹에 파일과 디렉토리를 각각 배치한다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch41$ cat in.manyfiles
file /a 2
file /b 2
file /c 2
file /d 2
file /e 2
file /f 2
file /g 2
file /h 2
file /i 2

dir /j
dir /t

file /t/u 3
file /j/l 1
file /t/v 3
file /j/m 1
file /t/w 3
file /j/n 1
file /t/x 3
file /j/o 1
file /t/y 3
file /j/p 1
file /t/z 3
file /j/q 1
file /t/A 3
file /j/r 1
file /t/B 3
file /j/C 3
```

```
borussen@DESKTOP-L834KLC:~/ostep_ch41$ python3 ffs.py -f in.manyfiles -c

num_groups:       10
inodes_per_group: 10
blocks_per_group: 30

free data blocks: 245 (of 300)
free inodes:      72 (of 100)

spread inodes?    False
spread data?      False
contig alloc:     1

     00000000000000000000 1111111111 2222222222
     01234567890123456789 0123456789 0123456789

group inodes    data
    0 /abcdefghi /aabbccdde effgghhii- ----------
    1 jlmnopqrC- jlmnopqrCC C--------- ----------
    2 tuvwxyzAB- tuuuvvvwww xxxyyyzzzA AABBB-----
    3 ---------- ---------- ---------- ----------
    4 ---------- ---------- ---------- ----------
    5 ---------- ---------- ---------- ----------
    6 ---------- ---------- ---------- ----------
    7 ---------- ---------- ---------- ----------
    8 ---------- ---------- ---------- ----------
    9 ---------- ---------- ---------- ----------
```

5. A metric to evaluate FFS is called *dirspan*. This metric calculates the spread of files within a particular directory, specifically the max distance between the inodes and data blocks of all files in the directory and the inode and data block of the directory itself. Run with in.manyfiles **and the** -T **flag, and calculate the dirspan of the three directories. Run with** -c **to check. How good of a job does FFS do in minimizing dirspan?**

dirspan은 inode block과 data block 사이의 span을 구성하는 inode region의 크기에 큰 영향을 받는다. 해당 FFS의 경우, 파일이 특정 그룹을 overflow 하기에 충분한 공간을 차지하지 않기 때문에 dirspan을 최소화하는 데 도움이 된다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch41$ python3 ffs.py -f in.manyfiles -T -c

num_groups:       10
inodes_per_group: 10
blocks_per_group: 30

free data blocks: 245 (of 300)
free inodes:      72 (of 100)

spread inodes?    False
spread data?      False
contig alloc:     1

     00000000000000000000 1111111111 2222222222
     01234567890123456789 0123456789 0123456789

group inodes    data
    0 /abcdefghi /aabbccdde effgghhii- ----------
    1 jlmnopqrC- jlmnopqrCC C--------- ----------
    2 tuvwxyzAB- tuuuvvvwww xxxyyyzzzA AABBB-----
    3 ---------- ---------- ---------- ----------
    4 ---------- ---------- ---------- ----------
    5 ---------- ---------- ---------- ----------
    6 ---------- ---------- ---------- ----------
    7 ---------- ---------- ---------- ----------
    8 ---------- ---------- ---------- ----------
    9 ---------- ---------- ---------- ----------

span: files
  file:         /a  filespan:  11
  file:         /b  filespan:  12
  file:         /c  filespan:  13
  file:         /d  filespan:  14
  file:         /e  filespan:  15
  file:         /f  filespan:  16
  file:         /g  filespan:  17
  file:         /h  filespan:  18
  file:         /i  filespan:  19
  file:       /t/u  filespan:  12
  file:       /j/l  filespan:  10
  file:       /t/v  filespan:  14
  file:       /j/m  filespan:  10
  file:       /t/w  filespan:  16
  file:       /j/n  filespan:  10
  file:       /t/x  filespan:  18
  file:       /j/o  filespan:  10
  file:       /t/y  filespan:  20
  file:       /j/p  filespan:  10
  file:       /t/z  filespan:  22
  file:       /j/q  filespan:  10
  file:       /t/A  filespan:  24
  file:       /j/r  filespan:  10
  file:       /t/B  filespan:  26
  file:       /j/C  filespan:  12
               avg  filespan:  14.76

span: directories
  dir:          /  dirspan:  28
  dir:         /j  dirspan:  20
  dir:         /t  dirspan:  34
              avg  dirspan:  27.33
```

6. Now change the size of the inode table per group to 5 (-I 5). How do you think this will change the layout of the files? Run with -c to see if you were right. How does it affect the dirspan?

그룹당 inode table의 크기를 5(-I 5)로 변경하면, 같은 디렉토리의 파일이 여러 그룹으로 분산되어 dirspan이 증가하게 된다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch41$ python3 ffs.py -f in.manyfiles -I 5  -T -c

num_groups:       10
inodes_per_group: 10
blocks_per_group: 30

free data blocks: 245 (of 300)
free inodes:      72 (of 100)

spread inodes?    True
spread data?      False
contig alloc:     1

     00000000000000000000 1111111111 2222222222
     01234567890123456789 0123456789 0123456789

group inodes    data
    0 /jy------- /jyyy----- ---------- ----------
    1 atp------- aatp------ ---------- ----------
    2 buz------- bbuuuzzz-- ---------- ----------
    3 clq------- cclq------ ---------- ----------
    4 dvA------- ddvvvAAA-- ---------- ----------
    5 emr------- eemr------ ---------- ----------
    6 fwB------- ffwwwBBB-- ---------- ----------
    7 gnC------- ggnCCC---- ---------- ----------
    8 hx-------- hhxxx----- ---------- ----------
    9 io-------- iio------- ---------- ----------

span: files
  file:         /a  filespan:  11
  file:         /b  filespan:  11
  file:         /c  filespan:  11
  file:         /d  filespan:  11
  file:         /e  filespan:  11
  file:         /f  filespan:  11
  file:         /g  filespan:  11
  file:         /h  filespan:  11
  file:         /i  filespan:  11
  file:       /t/u  filespan:  13
  file:       /j/l  filespan:  11
  file:       /t/v  filespan:  13
  file:       /j/m  filespan:  11
  file:       /t/w  filespan:  13
  file:       /j/n  filespan:  11
  file:       /t/x  filespan:  13
  file:       /j/o  filespan:  11
  file:       /t/y  filespan:  12
  file:       /j/p  filespan:  11
  file:       /t/z  filespan:  15
  file:       /j/q  filespan:  11
  file:       /t/A  filespan:  15
  file:       /j/r  filespan:  11
  file:       /t/B  filespan:  15
  file:       /j/C  filespan:  13
               avg  filespan:  11.92

span: directories
  dir:          /  dirspan: 371
  dir:         /j  dirspan: 371
  dir:         /t  dirspan: 332
              avg  dirspan: 358.00
```

7. Which group should FFS place inode of a new directory in? The default (simulator) policy looks for the group with the most free inodes. A different policy looks for a set of groups with the most free inodes. For example, if you run with -A 2**, when allocating a new directory, the simulator will look at groups in pairs and pick the best pair for the allocation. Run** ./ffs.py -f in.manyfiles -I 5 -A 2 -c **to see how allocation changes with this strategy. How does it affect dirspan? Why might this policy be good?**

-A 2로 실행하는 경우, 해당 정책에 의해서 디렉토리와 해당 파일의 inode 사이에 다른 디렉토리가 들어가지 않기 때문에 dirspan이 감소한다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch41$ python3 ffs.py -fin.manyfiles -I 5 -A 2 -T -c

num_groups:       10
inodes_per_group: 10
blocks_per_group: 30

free data blocks: 245 (of 300)
free inodes:      72 (of 100)

spread inodes?    True
spread data?      False
contig alloc:     1

     0000000000000000000 1111111111 2222222222
     01234567890123456789 0123456789 0123456789

group inodes    data
    0 /ejmyr---- /eejmyyyr- ---------- ----------
    1 ---------- ---------- ---------- ----------
    2 aftwpB---- aafftwwwpB BB-------- ----------
    3 ---------- ---------- ---------- ----------
    4 bgunzC---- bbgguuunzz zCCC------ ----------
    5 ---------- ---------- ---------- ----------
    6 chlxq----- cchhlxxxq- ---------- ----------
    7 ---------- ---------- ---------- ----------
    8 divoA----- ddiivvvoAA A--------- ----------
    9 ---------- ---------- ---------- ----------

span: files
  file:         /a  filespan:  11
  file:         /b  filespan:  11
  file:         /c  filespan:  11
  file:         /d  filespan:  11
  file:         /e  filespan:  11
  file:         /f  filespan:  12
  file:         /g  filespan:  12
  file:         /h  filespan:  12
  file:         /i  filespan:  12
  file:       /t/u  filespan:  14
  file:       /j/l  filespan:  12
  file:       /t/v  filespan:  14
  file:       /j/m  filespan:  11
  file:       /t/w  filespan:  14
  file:       /j/n  filespan:  14
  file:       /t/x  filespan:  14
  file:       /j/o  filespan:  14
  file:       /t/y  filespan:  13
  file:       /j/p  filespan:  14
  file:       /t/z  filespan:  16
  file:       /j/q  filespan:  14
  file:       /t/A  filespan:  16
  file:       /j/r  filespan:  13
  file:       /t/B  filespan:  16
  file:       /j/C  filespan:  18
               avg  filespan:  13.20

span: directories
  dir:           /  dirspan: 333
  dir:          /j  dirspan: 335
  dir:          /t  dirspan: 336
               avg  dirspan: 334.67
```

8. One last policy change we will explore relates to file fragmentation. Run ./ffs.py -f in.fragmented -v **and see if you can predict how the files that remain are allocated. Run with** -c **to confirm your answer. What is interesting about the data layout of file** /i**? Why is it problematic?**

./ffs.py -f in.fragmented -v를 실행했을 때, 파일 /i의 data layout은 fragmented 하고 non-contiguous 하다. 이럴 경우, 파일을 read 하거나 write 하기 위해서 disk가 더 많은 검색 및 회전을 해야 하기 때문에 속도가 느려지는 문제가 발생한다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch41$ python3 ffs.py -f in.fragmented -T -v -c
op create /a [size:1] ->success
op create /b [size:1] ->success
op create /c [size:1] ->success
op create /d [size:1] ->success
op create /e [size:1] ->success
op create /f [size:1] ->success
op create /g [size:1] ->success
op create /h [size:1] ->success
op delete /a ->success
op delete /c ->success
op delete /e ->success
op delete /g ->success
op create /i [size:8] ->success

num_groups:       10
inodes_per_group: 10
blocks_per_group: 30

free data blocks: 287 (of 300)
free inodes:      94 (of 100)

spread inodes?    False
spread data?      False
contig alloc:     1

     00000000000000000000 1111111111 2222222222
     01234567890123456789 0123456789 0123456789

group inodes    data
    0 /ib-d-f-h- /ibidifihi iii------- ----------
    1 ---------- ---------- ---------- ----------
    2 ---------- ---------- ---------- ----------
    3 ---------- ---------- ---------- ----------
    4 ---------- ---------- ---------- ----------
    5 ---------- ---------- ---------- ----------
    6 ---------- ---------- ---------- ----------
    7 ---------- ---------- ---------- ----------
    8 ---------- ---------- ---------- ----------
    9 ---------- ---------- ---------- ----------

span: files
  file:         /b  filespan:  10
  file:         /d  filespan:  10
  file:         /f  filespan:  10
  file:         /h  filespan:  10
  file:         /i  filespan:  21
               avg  filespan:  12.20

span: directories
  dir:           /  dirspan:  22
               avg  dirspan:  22.00
```

9. A new policy, which we call *contiguous allocation* (-C), tries to ensure that each file is allocated contiguously. Specifically, with -C n**, the file system tries to ensure that** n **contiguous blocks are free within a group before allocating a block. Run** ./ffs.py -f in.fragmented -v -C 2 -c **to see the difference. How does layout change as the parameter passed to** -C **increases? Finally, how does** -C **affect filespan and dirspan?**

./ffs.py -f in.fragmented -v -C 2 -c를 실행하면, 파일 /i의 data layout이 contiguous하게 할당된다. 만약 -C로 전달된 파라미터가 증가한다면 -C보다 작은 파일이 데이터 블록 영역에서 external fragmentation을 유발하게 된다. 그러므로 -C는 filespan과 dirspan을 증가시킨다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch41$ python3 ffs.py -f in.fragmented -T -v -C 2 -c
op create /a [size:1] ->success
op create /b [size:1] ->success
op create /c [size:1] ->success
op create /d [size:1] ->success
op create /e [size:1] ->success
op create /f [size:1] ->success
op create /g [size:1] ->success
op create /h [size:1] ->success
테스트 lete /a ->success
op delete /c ->success
op delete /e ->success
op delete /g ->success
op create /i [size:8] ->success

num_groups:       10
inodes_per_group: 10
blocks_per_group: 30

free data blocks: 287 (of 300)
free inodes:      94 (of 100)

spread inodes?    False
spread data?      False
contig alloc:     2

     00000000000000000000 1111111111 2222222222
     01234567890123456789 0123456789 0123456789

group inodes    data
    0 /ib-d-f-h- /-b-d-f-hi iiiiiii--- ----------
    1 ---------- ---------- ---------- ----------
    2 ---------- ---------- ---------- ----------
    3 ---------- ---------- ---------- ----------
    4 ---------- ---------- ---------- ----------
    5 ---------- ---------- ---------- ----------
    6 ---------- ---------- ---------- ----------
    7 ---------- ---------- ---------- ----------
    8 ---------- ---------- ---------- ----------
    9 ---------- ---------- ---------- ----------

span: files
  file:         /b  filespan:  10
  file:         /d  filespan:  10
  file:         /f  filespan:  10
  file:         /h  filespan:  10
  file:         /i  filespan:  25
               avg  filespan:  13.00

span: directories
  dir:           /  dirspan:  26
               avg  dirspan:  26.00
```