science + computing

A Bull Group Company

Preventing File System Fragmentation
samba xp 2010, Göttingen

Vortrag von Dr. Olaf Flebbe

science + computing ag

IT-Dienstleistungen und Software für anspruchsvolle Rechnernetze

Tübingen | München | Berlin | Düsseldorf

Begin of story

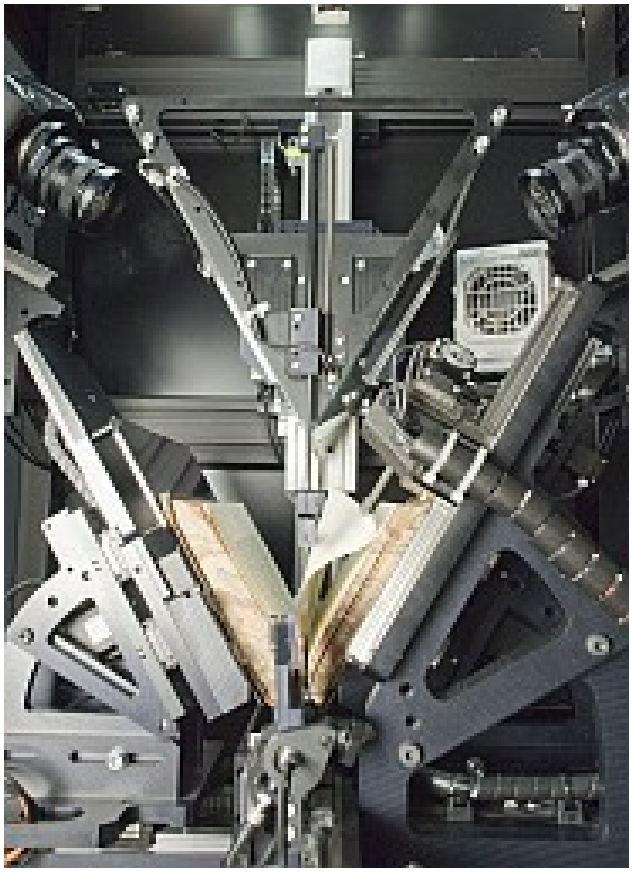
science + computing
A Bull Group Company

A large AIX HSM/TSM installation for semi-automatic scanning of antique books.

Problem with connection resets while unstaging files solved with Async I/O. Patches already upstream.

The file service performance – both read and write – is much slower than expected.

Findings: The files of the file services are highly fragmented on disk.



File System Internals

science + computing
A Bull Group Company

Tools to analyse file system layout

Linux: filefrag (-v) , debugfs status subcommand

AIX: fileplacej2

Sample output from an CD ripped with iTunes onto a samba share (ubuntu default):

```
$ ls -l 01\ Track\ 01.wav
-rwxr--r-- 1 nobody nogroup 58016828 2010-05-02
17:37 01 Track 01.wav
$ filefrag  01\ Track\ 01.wav
01 Track 01.wav: 722 extents found
```

File System Internals

File: f1 Size: 45615000 bytes Vol: /dev/hd3

Blk Size: 4096 Frag Size: 4096 Nfrags: 11137

Physical Addresses (mirror copy 1)				Logical Extent
01850430-01850431	hdisk0	2 frags	8192 Bytes, 0.0%	00014878-00014879
01845425	hdisk0	1 frags	4096 Bytes, 0.0%	00009873
01850432	hdisk0	1 frags	4096 Bytes, 0.0%	00014880
01845426	hdisk0	1 frags	4096 Bytes, 0.0%	00009874
01850433	hdisk0	1 frags	4096 Bytes, 0.0%	00014881
01845427	hdisk0	1 frags	4096 Bytes, 0.0%	00009875
01850434	hdisk0	1 frags	4096 Bytes, 0.0%	00014882
01845432	hdisk0	1 frags	4096 Bytes, 0.0%	00009880
01850435-01850436	hdisk0	2 frags	8192 Bytes, 0.0%	00014883-00014884

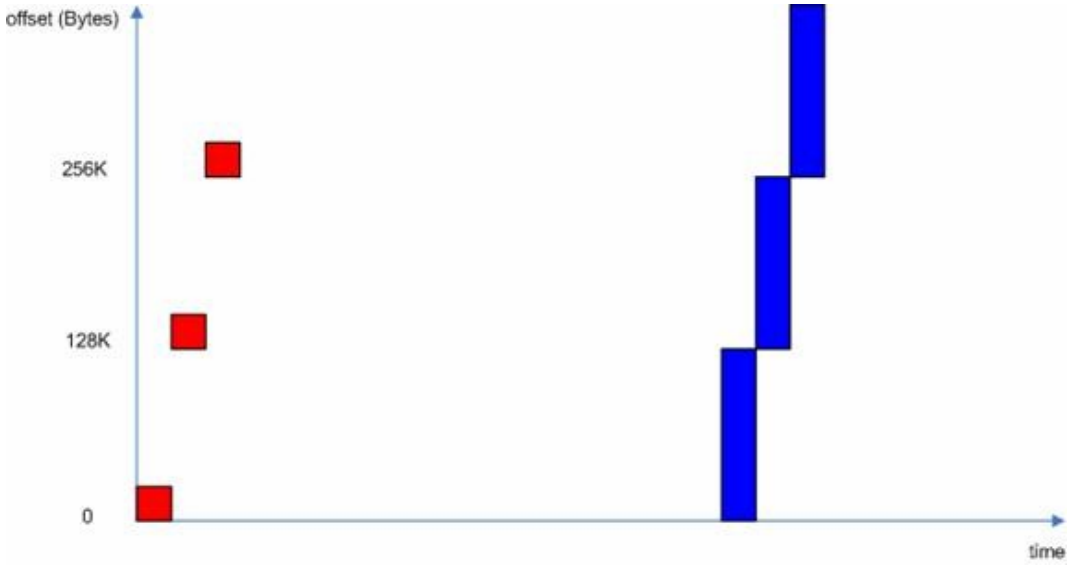
science + computing
A Bull Group Company

Windows CIFS Client Implementation

Exhibits „preallocation requests“:

Writes of a single null bytes beyond end of file.

Clustered writes filling up the „allocated“ disk space afterwards.



http://software.intel.com/en-us/articles/windows-client-cifs-behavior-can-slow-linux-nas-performance/

Write Pattern

science + computing
A Bull Group Company

The write pattern of the CIFS Client can be reproduced by this simple Windows Client „Unbuffered“

```
#define BLKSZ 9123

  for ( i = 0; i < 10000; i++) {
                if (BLKSZ != _write( fd, buf, BLKSZ)) {
                        fprintf( stderr, "disk full\n");
                        exit(1);
                }
        }
```

Samba default Semantics

science + computing
A Bull Group Company

Write pattern on UNIX Filesystem induced by the observed CIFS write patterns:

Prealloc:	Buffer Flush:
Seek(end of file + offset)	Seek(pos)
Write(1 byte)	Write(64k)
Seek(end of file + offset)	Write(64k)
Write (1 byte)	Write(64k)

NTFS/CIFS semantics of write beyond end of files is allocating space!

Impedance Mismatch

science + computing
A Bull Group Company

Unix Semantics of the preallocate requests (i.e. Write Beyond EOF) is creating a „sparse file“ (an partially allocated file).

The buffer flush will allocate disk space in the holes.

The result is highly dependent on file system!

Workaround

science + computing
A Bull Group Company

Samba smb.conf option

Strict allocate = yes

strict allocate (S)

```
In UNIX terminology this means that Samba will
stop creating sparse files. This can be slow on
some systems.
When strict allocate is no the server does sparse
disk block allocation when a file is extended.
```

Default: strict allocate = no

Lies, Damn Lies and Benchmarks

science + computing
A Bull Group Company

Running Unbuffered application on samba on

Ext3, ext4 with and without strict allocate and resulting fragmentation

Write performance

Ext3	13.1 sec	(4949 extents)
Ext4	9.7 sec	(1 extent)
Ext3 strict alloc	2.6 sec	(182 extents)
Ext4 strict alloc	2.45 sec	(1 extent)

FileSystems

science + computing
A Bull Group Company

Ext3 on Linux and JFS2 on AIX

Default setup leads to heavy file fragmentation!

Read performance drops by more an factor of 2 and more when reading fragmented data from disk! Avoid fragmentation !!

But it has an impact on **Write** too.

Ext4 uses a delayed allocation strategy,

So fragmentation is barely visible and no problem with read performance.

Customer needs Async I/O together with strict allocate!

science + computing
A Bull Group Company

Async implementation does not honor strict allocate = yes

Simple fallback to synchronous if prealloc pattern detected:
Fall back to synchronous.

Bugzilla ID #6942

Fallback no problem here since unstaging of files will be triggered before. (but IMHO we need aio write size = 1)

Async Benchmark

science + computing
A Bull Group Company

Running „unbuffered“ application on samba share to ext3, ext4 file system with and without strict allocate

Write performance

Ext3 13 sec (sync 13.1 sec) (5300 extents)
Ext4 13 sec (sync 9.7 sec) (1 extent)
Ext3 strict alloc 2.6 sec (sync 2.8 sec) (120 extents)
(w/o patch 13 sec 5000 extents!)
Ext4 strict alloc 2.6 sec (sync 2.45 sec) (1 extent)
(w/o patch 13 sec 1 extents!)

BREAK!

The saga continues... why don't do it better?

posix_fallocate()

science + computing
A Bull Group Company

Can we implement the allocating write ahead patterns better?

The posix_fallocate() system call looks promising:

Asking the file system to allocate space for a file on disk. Part of the POSIX Real Time Specifications.

libc is emulating it if the underlying file systems does not support it.

ext4 is supporting it natively.
It is supported since ages on AIX JFS2.

True lies

Wrote testbench emulating the write patterns of the samba file services with and w/o posix_fallocate():

On Ubuntu 9.10 (Linux Kernel 2.6.31) I got highly irregular timing behaviour: Not suitable for production.

What about a current distro ?

Ubuntu 10.04 (2.6.32)

Testbench

science + computing
A Bull Group Company

Synthetic Benchmark using the write pattern of samba when preallocating CIFS messages

-s Skip (Current implementation of strict allocate = no)

```
pwrite( fh, buf, 1, pos + blksz-1);
pos += blksz;
```

Testbench

science + computing
A Bull Group Company

Synthetic Benchmark using the write pattern of samba when preallocating CIFS messages

-c Clear (Current implementation of strict allocate = yes)

```
lseek( fh, pos, SEEK_SET);
write( fh, buf, blksz-1);
pwrite( fh, buf, 1, pos + blksz-1);
pos += blksz;
```

Testbench

science + computing
A Bull Group Company

Synthetic Benchmark using the write pattern of samba when preallocating CIFS messages

-f fallocate (optimization?)

```
posix_fallocate( fh, pos, blksz-1);
pwrite( fh, buf, 1, pos + blksz-1);
pos += blksz;
```

Results on ext4

science + computing
A Bull Group Company

skip	19 sec	(strict allocate off)
clear	25 sec	(strict allocate on)
fallocate	42 sec	(OUCH!!!)

Runtimes not comparable to samba because of network overhead!

Testbench

science + computing
A Bull Group Company

Synthetic Benchmark using the write pattern of samba when preallocating CIFS messages

-F allocate+1 (optimization)

```
posix_fallocate( fh, pos, blksz);
pos += blksz;
```

Results on ext3/ext4 compared

science + computing
A Bull Group Company

ext4: (flush time 18 sec : 50 MB/sec)

skip	20 sec	(strict allocate off)	7 extents
clear	25 sec	(strict allocate on)	
fallocate	42 sec	(OUCH 5000 extents!)	
fallocate+1	18 sec	(Uff …)	8 extents

ext3: (flush time 18 sec : 50 MB/sec)

skip	44 sec	(strict allocate off)	201106 extents
clear	36 sec	(strict allocate on)	227 extents
fallocate	37 sec		
fallocate+1	36 sec		

Results on JFS2 AIX 6.1

science + computing
A Bull Group Company

smaller size, LPAR .., absolute times not comparable

Buffer flush time is ~1sec

JFS2

skip	7sec	(10000 Fragments)	
clear	6sec	(12 Fragments)	
fallocate	25 sec	OUUUUUHHHH!!!	(2200 Fragments)
fallocate+1	61 sec	OUUUUUHHHH!!!	(2300 Fragments)

samba strict allocate optimazation

science + computing
A Bull Group Company

Patch to implement strict allocate with fallocate+1 strategy

I can post to samba-technical, if anyone interested...

Almost no speedup implementing strict allocate = yes when writing zeros compared to using posix_fallocate() on Linux ext4.

Conclusion

science + computing
A Bull Group Company

File fragmentation is an performance issue.

strict allocate = yes

in the current implementation seems to have an overall positive effect, tested on ext3, ext4 and jfs2.

Please make it default!

Educate users / Distribution maintainers / developers of embedded devices to check the fragmentation effect on file systems.

Conclusion 2

science + computing
A Bull Group Company

Avoiding fragmentation together with AIO not possible until now.

since the code path to handle

strict allocate = yes

is not easy to implement for Async I/O.

Please include simple patch with fallback from bugzilla #6942.

Conclusion3

science + computing
A Bull Group Company

Performance data are entirely inconsistent.

A patch shows no speed up when using posix_fallocate().

IMHO it is more likely to trigger fragmentation effects than a positive speed ups. So I decided to drop this proposal.

Currently there are some loose ends which may have to be discussed with the ext4 develepers first.

Final Conclusion

science + computing
A Bull Group Company

Thanks for samba!

Thanks for Listening!

science + computing

A Bull Group Company

Vielen Dank für Ihre Aufmerksamkeit.

Dr. Olaf Flebbe
science + computing ag
www.science-computing.de

Telefon 07071 9457-0
o dot flebbe ät science-computing.de