Ruby - Feature #5995

calling io_advise_internal() in read_all()

02/10/2012 02:14 PM - Glass_saga (Masaki Matsushita)

Status:	Rejected
Priority:	Normal
Assignee:	mame (Yusuke Endoh)
Target version:	

Description

=begin
I propose to call io_advise_internal() in read_all().
It will increase performance.

I created a dummy file:
dd if=/dev/zero of=dummy bs=1M count=100

Then, I ran the following:

require 'benchmark'

Benchmark.bm do |x|
x.report do
f = File.open("dummy") # dummy file(about 100MB)
f.read
end
end

I freed page cache before each test:
sudo sysctl -w vm.drop_caches=1

results on Ubuntu 11.10(3.0.0-15-server):

r34462:

user system total real
0.050000 0.220000 0.270000 (0.356033)

```
   user     system      total        real
```

0.050000 0.190000 0.240000 (0.332243)

```
   user     system      total        real
```

0.060000 0.210000 0.270000 (0.347758)

patched ruby:

user system total real
0.030000 0.130000 0.160000 (0.225866)

```
   user     system      total        real
```

0.040000 0.170000 0.210000 (0.250172)

```
   user     system      total        real
```

0.040000 0.150000 0.190000 (0.254654)

It shows the patch increases performance.
=end

History

#1 - 02/14/2012 12:48 PM - Glass_saga (Masaki Matsushita)

- File patch2.diff added

I modified the patch to use do_io_advise() not io_advise_internal().

#2 - 02/14/2012 03:13 PM - kosaki (Motohiro KOSAKI)

Huh? fadvise() is a hint for *future* io access. but usually read_all() don't have any future access. I don't think this patch makes platform independent improvement. How much OSs do you tested?

#3 - 02/14/2012 10:48 PM - Glass_saga (Masaki Matsushita)

=begin

> How much OSs do you tested?

I have tested on the only platform because I don't have root authority to execute "sudo sysctl -w vm.drop_caches=1" except the one.
I'm sorry.

I ran the test again and the following is average real time of 10 times:

r34599: 0.3800858
proposal: 0.2377475

Infomation about my environment.

```
% uname -mrsvo
Linux 3.0.0-15-server #26-Ubuntu SMP Fri Jan 20 19:07:39 UTC 2012 x86_64 GNU/Linux

% lsb_release  -a
LSB Version:
core-2.0-amd64:core-2.0-noarch:core-3.0-amd64:core-3.0-noarch:core-3.1-amd64:core-3.1-noarch:core-3.2-amd64:core-3.2-noarch:core-4.0-amd64:
core-4.0-noarch
Distributor ID: Ubuntu
Description:    Ubuntu 11.10
Release:        11.10
Codename:       oneiric
```

=end

#4 - 02/14/2012 11:26 PM - mame (Yusuke Endoh)

- Status changed from Open to Assigned

- Assignee set to kosaki (Motohiro KOSAKI)

Hello,

2012/2/14 Motohiro KOSAKI kosaki.motohiro@gmail.com:

> Huh? fadvise() is a hint for *future* io access. but usually read_all() don't have any future access.

The patch calls posix_fadvise *before* the main part of read_all.
Doesn't it make sense?

Quoted from the manpage of POSIX_FADVISE(2):

> Under Linux, POSIX_FADV_NORMAL sets the readahead window to
> the default size for the backing device; POSIX_FADV_SEQUENTIAL
> doubles this size,

I don't wonder that the patch works well. Actually I can
reproduce the speed up.

```
$ sudo sysctl -w vm.drop_caches=1
vm.drop_caches = 1

$ ./ruby.old t.rb
user     system      total        real
0.100000   1.150000   1.250000 (  3.776475)
```

```
$ sudo sysctl -w vm.drop_caches=1
vm.drop_caches = 1

$ ./ruby.new t.rb
user     system      total        real
0.090000   0.750000   0.840000 (  3.766539)

$ uname -a
Linux inch 3.0.0-16-generic-pae #28-Ubuntu SMP Fri Jan 27 19:24:01 UTC 2012 i686 i686 i386 GNU/Linux
```

--
Yusuke Endoh mame@tsg.ne.jp

#5 - 02/15/2012 02:55 AM - kosaki (Motohiro KOSAKI)

- Status changed from Assigned to Rejected

> The patch calls posix_fadvise *before* the main part of read_all.
> Doesn't it make sense?

It doesn't. Because of, when we read whole file contents, we only need *one* read syscall if the file is regular. In other words, current read_all() suck. It read a few kilo bytes and append them to a string. That's said it create tons realloc. That doesn't make sense. we need fix the root cause. btw, read_all() also abuse BUFSIZ.

So, No. We are not welcome a bandaid.

#6 - 02/15/2012 12:03 PM - Glass_saga (Masaki Matsushita)

=begin

> In other words, current read_all() suck. It read a few kilo bytes and append them to a string.

I modified io.c to show how many bytes read_all() reads on each syscall.

io.c:1834

```
static long
io_bufread(char *ptr, long len, rb_io_t *fptr)
{
long offset = 0;
long n = len;
long c;

if (READ_DATA_PENDING(fptr) == 0) {
    while (n > 0) {
      again:
        c = rb_read_internal(fptr->fd, ptr+offset, n);
        if (c == 0) break;
        printf("%ld/%ld\n", c, len); /* how many bytes? */
        if (c < 0) {
            if (rb_io_wait_readable(fptr->fd))
                goto again;
            return -1;
        }
        offset += c;
        if ((n -= c) <= 0) break;
        rb_thread_wait_fd(fptr->fd);
    }
    return len - n;
}
```

io.c:2137

```
for (;;) {
READ_CHECK(fptr);
n = io_fread(str, bytes, siz - bytes, fptr);
if (n == 0 && bytes == 0) {
rb_str_set_len(str, 0);
break;
}
bytes += n;
rb_str_set_len(str, bytes);
```

```
if (cr != ENC_CODERANGE_BROKEN)
pos += rb_str_coderange_scan_restartable(RSTRING_PTR(str) + pos, RSG_PTR(str) + bytes, enc, &cr);
if (bytes < siz) break;
printf("%ld/%ld\n", n, siz); /* how many bytes? */
siz += BUFSIZ;
rb_str_modify_expand(str, BUFSIZ);
}
```

Then, I ran the test same as [ruby-core:42471] and I got:

```
user     system      total        real
102400000/102400000
102400000/102400000
0.020000   0.170000   0.190000 (  0.254729)
```

It shows current read_all() reads file at a time.
=end

#7 - 02/16/2012 03:23 AM - mame (Yusuke Endoh)

Hello,

2012/2/15 Motohiro KOSAKI kosaki.motohiro@gmail.com:

> It doesn't. Because of, when we read whole file contents, we only need *one* read syscall if the file is regular. In other words, current read_all() suck. It read a few kilo bytes and append them to a string. That's said it create tons realloc.

Really? I could be wrong, but as far as I know, IO#read first
uses fstat to estimate a buffer length enough to load the whole
content of the file. Masaki also showed the behavior.
I think there is something wrong.

--
Yusuke Endoh mame@tsg.ne.jp

#8 - 02/16/2012 09:12 PM - sorah (Sorah Fukumori)

- Status changed from Rejected to Assigned

Hi,

Yusuke Endoh wrote at [ruby-core:42663]:

> I think there is something wrong.

Agreed. Reopening this ticket.

#9 - 02/16/2012 09:55 PM - mame (Yusuke Endoh)

- Status changed from Assigned to Rejected

Hello,

There is indeed something wrong, but anyway, I agree with
kosaki's point; we cannot import the patch until we know
the exact reason why it brings performance improvement.
So please reopen this ticket if you find the reason.
(I expect kosaki-san to consider this!)

I wrote a simple C code to experiment this. The result is
as kosaki said; when calling only one read syscall, posix_
fadvise does NOT bring performance improvement (even slower).
I really wonder why File#read becomes faster.

using posix_fadvise

```
$ sudo sysctl -w vm.drop_caches=1 && time ./t dummy T
vm.drop_caches = 1
314572800

real    0m5.401s
user    0m0.000s
```

sys 0m0.740s

NOT using posix_fadvise

$ sudo sysctl -w vm.drop_caches=1 && time ./t dummy F
vm.drop_caches = 1
314572800

real 0m3.967s
user 0m0.000s
sys 0m0.896s

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>

int main(int argc, char *argv[])
{
int fd = open(argv[1], O_RDONLY);
char *buf;
struct stat s;

fstat(fd, &s);
buf = malloc(s.st_size);

if (argv[2][0] == 'T') {
    posix_fadvise(fd, 0, 0, POSIX_FADV_SEQUENTIAL);
}

printf("%d\n", read(fd, buf, s.st_size));

return 0;

}
```

--
Yusuke Endoh mame@tsg.ne.jp

#10 - 02/16/2012 10:27 PM - Glass_saga (Masaki Matsushita)

I ran Endoh-san's code.

average time of 10 times:
F: 0.2892
T(use posix_fadvise()): 0.2226

I think posix_fadvise() makes sense at least on my environment.
Can anyone reproduce this?

Anyway, I agree with Endoh-san because I have no idea about why the patch makes read_all() faster now.

#11 - 02/18/2012 02:54 AM - MartinBosslet (Martin Bosslet)

Motohiro KOSAKI wrote:

> Because of, when we read whole file contents, we only need *one* read syscall if the file is regular. In other words, current read_all() suck. It read a few kilo bytes and append them to a string. That's said it create tons realloc. That doesn't make sense. we need fix the root cause. btw, read_all() also abuse BUFSIZ.

I noticed that, too, lately, when I had to implement something similar. read_all[1] resizes the buffer in a linear fashion. This means a linear number of reallocs - wouldn't it make sense to grow the buffer exponentially by multiplying with a constant factor (1.5 or 2 maybe) instead? That way, we would only have a logarithmic number of reallocs, which would probably already give better performance. It's a bit more wasteful on memory usage, but I assume that's tolerable because in the end we would resize to the exact total size anyway, no?

[1] https://github.com/ruby/ruby/blob/trunk/io.c#L2152

#12 - 02/19/2012 11:25 AM - mame (Yusuke Endoh)

Hello, Martin

I guess that your point is off topic from this ticket.
As I and Masaki showed, in normal cases, File#read does
not reallocate a memory. (Let me know if I'm wrong)

But I think your point is valid for the general IO#read
(especially, reading from a socket). I recommend you to
create a patch and a benchmark, and discuss in another
thread.

--
Yusuke Endoh mame@tsg.ne.jp

#13 - 02/20/2012 05:52 AM - MartinBosslet (Martin Bosslet)

Yusuke Endoh wrote:

> Hello, Martin
>
> I guess that your point is off topic from this ticket.
> As I and Masaki showed, in normal cases, File#read does
> not reallocate a memory. (Let me know if I'm wrong)
>
> But I think your point is valid for the general IO#read
> (especially, reading from a socket). I recommend you to
> create a patch and a benchmark, and discuss in another
> thread.

Hi Yusuke,

you're right, my apologies! I just read 'read_all' and
'a lot of reallocs' and was immediately reminded of what
I noticed on a different note :) I opened Issue #6047 for
this separate topic!

-Martin

#14 - 02/21/2012 01:47 PM - Glass_saga (Masaki Matsushita)

Some file systems(e.g. ext3, ext4) use do_sync_read() for general read.
http://lxr.linux.no/#linux+v3.2.6/fs/ext3/file.c#L55
http://lxr.linux.no/#linux+v3.2.6/fs/ext4/file.c#L231

In read process, do_generic_file_read() is called finally.
http://lxr.linux.no/#linux+v3.2.6/fs/read_write.c#L338 (do_sync_read())
In ext3 and ext4, f_op->aio_read is generic_file_aio_read().
http://lxr.linux.no/#linux+v3.2.6/mm/filemap.c#L1395 (It calls do_generic_file_read().)

Then, do_generic_file_read() calls page_cache_sync_readahead() or page_cache_async_readahead().
http://lxr.linux.no/#linux+v3.2.6/mm/filemap.c#L1118

Both page_cache_sync_readahead() and page_cache_async_readahead() call ondemand_readahead() and its readahead size is limited by ra->ra_pages.
http://lxr.linux.no/#linux+v3.2.6/mm/readahead.c#L401

posix_fadvise() expands ra->ra_pages(http://lxr.linux.no/#linux+v3.2.6/mm/fadvise.c#L90) and it reduces the number of times of actual read.
Therefore, I think the patch makes sense on some file systems as stated above.

#15 - 02/25/2012 12:26 PM - mame (Yusuke Endoh)

- Status changed from Rejected to Assigned

Kosaki-san, can you check [ruby-core:42772]?

Matsushita-san,
I'm not sure if the mechanism you said is right because just
using posix_fadvise did not bring any speed improvement in my
experiment of [ruby-core:42683]. Did you run my program?
I'm afraid there is another reason why posix_fadvise brings
improvement to Ruby.

--
Yusuke Endoh mame@tsg.ne.jp

#16 - 02/25/2012 02:02 PM - Glass_saga (Masaki Matsushita)

Yusuke Endoh wrote:

> Did you run my program?

Yes. I ran your program in [ruby-core:42683] and I really experienced performance improvement on my environment.
Results are in [ruby-core:42684] and they can be reproduced.

I'm wondering if the fact that my environment is VPS hosted by customized KVM affects the results.
However, I have not found any grounded reasons so far.

#17 - 02/25/2012 03:23 PM - mame (Yusuke Endoh)

Hello,

2012/2/25 Masaki Matsushita glass.saga@gmail.com:

> Yusuke Endoh wrote:
>
> > Did you run my program?
>
> Yes. I ran your program in [ruby-core:42683] and I really experienced performance improvement on my environment.
> Results are in [ruby-core:42684] and they can be reproduced.

Oops. I was missing. Sorry.

--
Yusuke Endoh mame@tsg.ne.jp

#18 - 06/14/2012 04:14 PM - kosaki (Motohiro KOSAKI)

- Assignee changed from kosaki (Motohiro KOSAKI) to mame (Yusuke Endoh)

Endoh-san,

I really dislike this patch because this patch abuse fadvise() and don't guarantee to positive effect on other environment (dirrerent os, different storage type). But if you strongly prefer it, I give up to opposite.

Please check-in by yourself.

#19 - 11/20/2012 02:38 AM - mame (Yusuke Endoh)

- Target version set to 2.6

Well, I wonder what I should do.
... I procrastinate the decision to next minor.

--
Yusuke Endoh mame@tsg.ne.jp

#20 - 12/25/2017 06:15 PM - naruse (Yui NARUSE)

- Target version deleted (2.6)

#21 - 06/20/2018 04:12 PM - Glass_saga (Masaki Matsushita)

- Status changed from Assigned to Rejected

Files

patch.diff	1.25 KB	02/10/2012	Glass_saga (Masaki Matsushita)
patch2.diff	1003 Bytes	02/14/2012	Glass_saga (Masaki Matsushita)