Ruby trunk - Feature #10714

Array#reject! nonlinear performance problem

01/08/2015 02:24 AM - akr (Akira Tanaka)

Status:	Closed
Priority:	Normal
Assignee:	akr (Akira Tanaka)
Target version:	

Description

I found Array#reject! is too slow.

I measured it and it seems the performance is nonlinear.

```
% ./ruby -v -e '
20.times {|i|
  a = [nil]*i*10000;
  t1 = Time.now
  a.reject! { true }
  t2 = Time.now
  t = t2 - t1
  p ["*" * (t * 20).to_i , t]
}
'
ruby 2.3.0dev (2015-01-08 trunk 49175) [x86_64-linux]
["", 3.683e-06]
["", 0.019059723]
["*", 0.052964771]
["**", 0.1177318]
["****", 0.208824818]
["******", 0.334757354]
["*********", 0.482717139]
["************", 0.669606441]
["*****************", 0.866588588]
["***********************", 1.116195389]
["****************************", 1.392828177]
["*********************************", 1.701906753]
["****************************************", 2.013290644]
["************************************************", 2.415258165]
["***********************************************************", 2.783918449]
["*****************************************************************", 3.27417584]
["***************************************************************************", 3.724958298]
["************************************************************************************", 4.30726
3787]
["*******************************************************************************************
**", 4.922179118]
["*******************************************************************************************
***********", 5.403641168]
```

Ruby 2.2, 2.1, 2.0, 1.9.3 also have the problem but Ruby 1.9.2 works well.

```
% ruby-1.9.2-p330 -v -e '
20.times {|i|
  a = [nil]*i*10000;
  t1 = Time.now
  a.reject! { true }
  t2 = Time.now
  t = t2 - t1
  p ["*" * (t * 20).to_i , t]
}
'
ruby 1.9.2p330 (2014-08-07 revision 47094) [x86_64-linux]
["", 2.111e-06]
["", 0.000798623]
```

```
["", 0.001441408]
["", 0.00155386]
["", 0.001656242]
["", 0.002166389]
["", 0.002355492]
["", 0.002703977]
["", 0.003123692]
["", 0.00348722]
["", 0.003884792]
["", 0.004300034]
["", 0.004701378]
["", 0.006854893]
["", 0.005485207]
["", 0.005972309]
["", 0.006298597]
["", 0.006901775]
["", 0.007216343]
["", 0.007373332]
```

Related issues:

Related to Ruby trunk - Bug #2545: Array#delete_if is borked if user calls 'b...	**Closed**	**01/02/2010**
Related to Ruby trunk - Bug #5752: Array#delete_if side effects due to change...	**Closed**	**12/13/2011**
Related to Ruby trunk - Bug #10722: Array#keep_if is borked if user calls 'br...	**Closed**	**01/09/2015**

Associated revisions

Revision 5ec029d1 - 01/15/2015 01:44 AM - nobu (Nobuyoshi Nakada)

array.c: linear performance

- array.c (rb_ary_select_bang, ary_reject_bang): linear performance. [ruby-core:67418] [Feature #10714]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@49255 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 49255 - 01/15/2015 01:44 AM - nobu (Nobuyoshi Nakada)

array.c: linear performance

- array.c (rb_ary_select_bang, ary_reject_bang): linear performance. [ruby-core:67418] [Feature #10714]

Revision 49255 - 01/15/2015 01:44 AM - nobu (Nobuyoshi Nakada)

array.c: linear performance

- array.c (rb_ary_select_bang, ary_reject_bang): linear performance. [ruby-core:67418] [Feature #10714]

Revision 49255 - 01/15/2015 01:44 AM - nobu (Nobuyoshi Nakada)

array.c: linear performance

- array.c (rb_ary_select_bang, ary_reject_bang): linear performance. [ruby-core:67418] [Feature #10714]

Revision 49255 - 01/15/2015 01:44 AM - nobu (Nobuyoshi Nakada)

array.c: linear performance

- array.c (rb_ary_select_bang, ary_reject_bang): linear performance. [ruby-core:67418] [Feature #10714]

Revision 49255 - 01/15/2015 01:44 AM - nobu (Nobuyoshi Nakada)

array.c: linear performance

- array.c (rb_ary_select_bang, ary_reject_bang): linear performance. [ruby-core:67418] [Feature #10714]

Revision c7ba10cc - 01/21/2015 12:22 AM - akr (Akira Tanaka)

#10714 is a feature.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@49361 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 49361 - 01/21/2015 12:22 AM - akr (Akira Tanaka)

#10714 is a feature.

Revision 49361 - 01/21/2015 12:22 AM - akr (Akira Tanaka)

#10714 is a feature.

Revision 49361 - 01/21/2015 12:22 AM - akr (Akira Tanaka)

#10714 is a feature.

Revision 49361 - 01/21/2015 12:22 AM - akr (Akira Tanaka)

#10714 is a feature.

Revision 49361 - 01/21/2015 12:22 AM - akr (Akira Tanaka)

#10714 is a feature.

History

#1 - 01/08/2015 08:55 AM - wanabe (_ wanabe)

According to git bisect, it caused by r32373 (related to [Bug #2545]) .

#2 - 01/09/2015 11:30 PM - nobu (Nobuyoshi Nakada)

- Related to Bug #2545: Array#delete_if is borked if user calls 'break' added

#3 - 01/10/2015 12:48 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Rejected

The target of Array#reject! is the receiver itself, so the modification can be observed from the given block, or the block can exit by break.
Therefore the compaction is necessary each times.

I can't think of the way to avoid it, unless we have a way to guarantee that an object is never referred from a block, directly or indirectly.

#4 - 01/10/2015 03:51 AM - akr (Akira Tanaka)

I don't think the modification must be observable from the given block.

[Bug #2545] doesn't discuss the observability in the block.
It discusses the receiver after "break".

If the modification is not required to be observable from the block,
it is possible to implement with linear performance.

```
% ruby -e '
class Array
  def my_reject!
    i = 0
    j = 0
    while i < self.length
      v = self[i]
      unless yield v
        self[j] = v
        j += 1
      end
      i += 1
    end
    i != j ? self : nil
  ensure
    if i != j
      self[j, i-j] = []
    end
  end
end
a = [1,2,3]; a.my_reject! {|v| v == 2 }; p a
a = [5,6,7,8,9,10]; a.my_reject! {|i| break i if i > 8; i < 7}; p a
'
[1, 3]
[7, 8, 9, 10]
```

So the problem is which is important between the linear performance and the observability.

I vote the linear performance because:

- the nonlinear performance makes reject! useless for long array and
- [Bug #2545] is not a issue for the observability.

#5 - 01/10/2015 04:04 AM - nobu (Nobuyoshi Nakada)

I know, and make test-all failed.

#6 - 01/10/2015 04:15 AM - akr (Akira Tanaka)

- Related to Bug #5752: Array#delete_if side effects due to changeset r32360 added

#7 - 01/10/2015 04:30 AM - akr (Akira Tanaka)

Hm.
The failed test is introduced by you after [Bug #5752].

The bug report describes as "If this is indeed the intended behaviour, ...".
It means that the reporter is not sure that the current behavior is intended or not.

Why current behavior is desired over linear performance behavior?

#8 - 01/10/2015 06:45 AM - nobu (Nobuyoshi Nakada)

Ok, then try if something dies?

```
diff --git a/array.c b/array.c
index 0de7231..f2f7352 100644
--- a/array.c
+++ b/array.c
@@ -2824,6 +2824,48 @@ rb_ary_select(VALUE ary)
     return result;
 }

+struct select_bang_arg {
+    VALUE ary;
+    long len[2];
+};
+
+static VALUE
+select_bang_i(VALUE a)
+{
+    volatile struct select_bang_arg *arg = (void *)a;
+    VALUE ary = arg->ary;
+    long i1, i2;
+
+    for (i1 = i2 = 0; i1 < RARRAY_LEN(ary); arg->len[0] = ++i1) {
+   VALUE v = RARRAY_AREF(ary, i1);
+   if (!RTEST(rb_yield(v))) continue;
+   if (i1 != i2) {
+       rb_ary_store(ary, i2, v);
+   }
+   arg->len[1] = ++i2;
+    }
+    return (i1 == i2) ? Qnil : ary;
+}
+
+static VALUE
+select_bang_ensure(VALUE a)
+{
+    volatile struct select_bang_arg *arg = (void *)a;
+    VALUE ary = arg->ary;
+    long len = RARRAY_LEN(ary);
+    long i1 = arg->len[0], i2 = arg->len[1];
+
+    if (i2 < i1) {
+   if (i1 < len) {
+       RARRAY_PTR_USE(ary, ptr, {
+           MEMMOVE(ptr + i2, ptr + i1, VALUE, len - i1);
+       });
+   }
+   ARY_SET_LEN(ary, len - i1 + i2);
+    }
+    return ary;
+}
+
```

```
 /*
  *  call-seq:
  *     ary.select!  {|item| block } -> ary or nil
@@ -2832,6 +2874,8 @@ rb_ary_select(VALUE ary)
  *  Invokes the given block passing in successive elements from +self+,
  *  deleting elements for which the block returns a +false+ value.
  *
+ *  The array may not be changed instantly every time the block is called.
+ *
  *  If changes were made, it will return +self+, otherwise it returns +nil+.
  *
  *  See also Array#keep_if
@@ -2843,22 +2887,14 @@ rb_ary_select(VALUE ary)
 static VALUE
 rb_ary_select_bang(VALUE ary)
 {
-    long i;
-    VALUE result = Qnil;
+    struct select_bang_arg args;

     RETURN_SIZED_ENUMERATOR(ary, 0, 0, ary_enum_length);
     rb_ary_modify(ary);
-    for (i = 0; i < RARRAY_LEN(ary); ) {
-   VALUE v = RARRAY_AREF(ary, i);
-   if (!RTEST(rb_yield(v))) {
-       rb_ary_delete_at(ary, i);
-       result = ary;
-   }
-   else {
-       i++;
-   }
-    }
-    return result;
+
+    args.ary = ary;
+    args.len[0] = args.len[1] = 0;
+    return rb_ensure(select_bang_i, (VALUE)&args, select_bang_ensure, (VALUE)&args);
 }

 /*
@@ -3101,23 +3137,32 @@ ary_reject(VALUE orig, VALUE result)
 }

 static VALUE
-ary_reject_bang(VALUE ary)
+reject_bang_i(VALUE a)
 {
-    long i;
-    VALUE result = Qnil;
+    volatile struct select_bang_arg *arg = (void *)a;
+    VALUE ary = arg->ary;
+    long i1, i2;

-    rb_ary_modify_check(ary);
-    for (i = 0; i < RARRAY_LEN(ary); ) {
-   VALUE v = RARRAY_AREF(ary, i);
-   if (RTEST(rb_yield(v))) {
-       rb_ary_delete_at(ary, i);
-       result = ary;
-   }
-   else {
-       i++;
+    for (i1 = i2 = 0; i1 < RARRAY_LEN(ary); arg->len[0] = ++i1) {
+   VALUE v = RARRAY_AREF(ary, i1);
+   if (RTEST(rb_yield(v))) continue;
+   if (i1 != i2) {
+       rb_ary_store(ary, i2, v);
    }
+   arg->len[1] = ++i2;
     }
-    return result;
+    return (i1 == i2) ? Qnil : ary;
+}
+
+static VALUE
```

```
+ary_reject_bang(VALUE ary)
+{
+    struct select_bang_arg args;
+
+    rb_ary_modify_check(ary);
+    args.ary = ary;
+    args.len[0] = args.len[1] = 0;
+    return rb_ensure(reject_bang_i, (VALUE)&args, select_bang_ensure, (VALUE)&args);
 }

 /*
@@ -3128,8 +3173,7 @@ ary_reject_bang(VALUE ary)
  *  Equivalent to Array#delete_if, deleting elements from +self+ for which the
  *  block evaluates to +true+, but returns +nil+ if no changes were made.
  *
- *  The array is changed instantly every time the block is called, not after
- *  the iteration is over.
+ *  The array may not be changed instantly every time the block is called.
  *
  *  See also Enumerable#reject and Array#delete_if.
  *
```

#9 - 01/10/2015 06:47 AM - nobu (Nobuyoshi Nakada)

Forgot a patch of the tests

```
diff --git a/test/ruby/test_array.rb b/test/ruby/test_array.rb
index 31f33dd..33fc5d6 100644
--- a/test/ruby/test_array.rb
+++ b/test/ruby/test_array.rb
@@ -661,7 +661,7 @@ class TestArray < Test::Unit::TestCase

     bug2545 = '[ruby-core:27366]'
     a = @cls[ 5, 6, 7, 8, 9, 10 ]
-    assert_equal(9, a.delete_if {|i| break i if i > 8; assert_equal(a[0], i) || true if i < 7})
+    assert_equal(9, a.delete_if {|i| break i if i > 8; i < 7})
     assert_equal(@cls[7, 8, 9, 10], a, bug2545)
   end

@@ -1160,7 +1160,7 @@ class TestArray < Test::Unit::TestCase

     bug2545 = '[ruby-core:27366]'
     a = @cls[ 5, 6, 7, 8, 9, 10 ]
-    assert_equal(9, a.reject! {|i| break i if i > 8; assert_equal(a[0], i) || true if i < 7})
+    assert_equal(9, a.reject! {|i| break i if i > 8; i < 7})
     assert_equal(@cls[7, 8, 9, 10], a, bug2545)
   end
```

#10 - 01/10/2015 07:22 AM - akr (Akira Tanaka)

> Ok, then try if something dies?

I think we should try.

We can explain the reason of this change.

It may be a good idea to describe this issue in NEWS because a documented behavior changed, though.

#11 - 01/10/2015 07:54 AM - akr (Akira Tanaka)

- Related to Bug #10722: Array#keep_if is borked if user calls 'break' added

#12 - 01/10/2015 08:39 AM - akr (Akira Tanaka)

It seems that several people found this problem.

- http://qiita.com/Nabetani/items/623df6f738864b5ed005
- https://twitter.com/grafi_tt/status/263097832158924800
- http://d.hatena.ne.jp/plonk123/20140603/1401832299

#13 - 01/10/2015 08:44 AM - duerst (Martin Dürst)

Akira Tanaka wrote:

> So the problem is which is important between the linear performance and the observability.
>
> I vote the linear performance because:
>
> - the nonlinear performance makes reject! useless for long array and
> - [Bug #2545] is not a issue for the observability.

I fully agree. Nonlinear performance is a killer. Observability and ability to break are just nice-to-have. I think we should reopen the bug.

#14 - 01/10/2015 09:36 AM - akr (Akira Tanaka)

- Tracker changed from Bug to Feature

Now, I think this issue is a feature instead of a bug because it changes a documented behavior.

#15 - 01/10/2015 02:22 PM - nobu (Nobuyoshi Nakada)

https://github.com/nobu/ruby/compare/Feature%2310714-Array-linear-performance

#16 - 01/10/2015 06:40 PM - zzak (Zachary Scott)

Using the following benchmark I compared nobu's patch vs. 2.2:

```
require 'derailed_benchmarks'
require 'derailed_benchmarks/tasks'

namespace :perf do
  desc "Array#select! and friends"
  task :array_select => [:setup] do
    Benchmark.ips do |x|
      [10, 100, 1000].map do |i|
        a = []
        i.times { |z| a = [nil]*z*10000 }

        x.report("Array#select! * #{i}") { a.select! { true } }
      end

      x.compare!
    end
  end
end
```

Here are the results:

2.2

```
Calculating -------------------------------------
  Array#select! * 10    24.000  i/100ms
 Array#select! * 100     2.000  i/100ms
Array#select! * 1000     1.000  i/100ms
-------------------------------------------------
  Array#select! * 10    243.090  (± 5.8%) i/s -      1.224k
 Array#select! * 100     22.046  (± 4.5%) i/s -    110.000
Array#select! * 1000      2.200  (± 0.0%) i/s -     11.000

Comparison:
  Array#select! * 10:      243.1 i/s
 Array#select! * 100:       22.0 i/s - 11.03x slower
Array#select! * 1000:        2.2 i/s - 110.48x slower
```

Patched

```
Calculating -------------------------------------
  Array#select! * 10    23.000  i/100ms
 Array#select! * 100     2.000  i/100ms
Array#select! * 1000     1.000  i/100ms
-------------------------------------------------
  Array#select! * 10    234.708  (± 6.0%) i/s -      1.173k
 Array#select! * 100     21.559  (± 4.6%) i/s -    108.000
Array#select! * 1000      2.129  (± 0.0%) i/s -     11.000

Comparison:
```

```
  Array#select! * 10:      234.7 i/s
 Array#select! * 100:       21.6 i/s - 10.89x slower
Array#select! * 1000:        2.1 i/s - 110.25x slower
```

#17 - 01/10/2015 06:56 PM - funny_falcon (Yura Sokolov)

Zachary, patch fixes reject! and you test select!, ie patch fixes case when most items were deleted, and you test case when no item is deleted.

Based on numbers you present, patched version is so close to unpatched so I could not tell real difference between.

#18 - 01/11/2015 01:14 AM - zzak (Zachary Scott)

Ooops, there was a bug in my test, and after fixing it and updating the results above it seems that nobu's patch improves things slightly.

I'm going to post the results for both select! and reject! soon.

@Yura, you can see that nobu's patch also to #select! here:
https://github.com/ruby/ruby/pull/811/files#diff-24411fb2a634ee46e29925b04767d15eR2851

#19 - 01/11/2015 01:14 AM - zzak (Zachary Scott)

- Status changed from Rejected to Open

#20 - 01/11/2015 02:05 AM - zzak (Zachary Scott)

Here are the results for reject! using this benchmark:

```
require 'derailed_benchmarks'
require 'derailed_benchmarks/tasks'

namespace :perf do
  desc "Array#reject!"
  task :array_reject => [:setup] do
    Benchmark.ips do |x|
      20.times do |i|
        a = []
        i.times { |z| a = [nil]*z*10000 }

        x.report("Array#reject! * #{i}") { a.reject! { true } }
      end
    end
  end
end
```

2.2

```
   Array#reject! * 0      8.236M (± 6.4%) i/s -     40.928M
   Array#reject! * 1      8.255M (± 5.9%) i/s -     41.049M
   Array#reject! * 2      8.298M (± 5.5%) i/s -     41.297M
   Array#reject! * 3      8.291M (± 5.7%) i/s -     41.243M
   Array#reject! * 4      8.244M (± 7.0%) i/s -     40.901M
   Array#reject! * 5      8.148M (± 8.0%) i/s -     40.375M
   Array#reject! * 6      8.247M (± 6.6%) i/s -     40.939M
   Array#reject! * 7      8.309M (± 5.3%) i/s -     41.319M
   Array#reject! * 8      8.313M (± 5.1%) i/s -     41.369M
   Array#reject! * 9      8.332M (± 4.0%) i/s -     41.522M
  Array#reject! * 10      8.331M (± 4.0%) i/s -     41.491M
  Array#reject! * 11      8.281M (± 5.6%) i/s -     41.106M
  Array#reject! * 12      8.162M (± 8.2%) i/s -     39.586M
  Array#reject! * 13      1.006M (±17.3%) i/s -    743.452k
  Array#reject! * 14      1.007M (±17.0%) i/s -    743.165k
  Array#reject! * 15      1.007M (±17.0%) i/s -    746.140k
  Array#reject! * 16      1.006M (±17.2%) i/s -    747.039k
  Array#reject! * 17      1.007M (±17.2%) i/s -    752.532k
  Array#reject! * 18      1.006M (±17.2%) i/s -    741.489k
  Array#reject! * 19      1.006M (±17.3%) i/s -    745.300k
```

Patched

```
   Array#reject! * 0      7.285M (± 5.8%) i/s -     36.227M
   Array#reject! * 1      7.294M (± 6.5%) i/s -     36.274M
   Array#reject! * 2      7.378M (± 4.1%) i/s -     36.831M
   Array#reject! * 3      7.260M (± 7.8%) i/s -     35.974M
   Array#reject! * 4      7.385M (± 4.3%) i/s -     36.849M
   Array#reject! * 5      7.376M (± 5.5%) i/s -     36.735M
```

```
  Array#reject! * 6      7.145M (±10.2%) i/s -     35.136M
  Array#reject! * 7      6.974M (± 9.6%) i/s -     34.385M
  Array#reject! * 8      7.279M (± 5.7%) i/s -     36.259M
  Array#reject! * 9      7.333M (± 5.3%) i/s -     36.532M
 Array#reject! * 10      7.345M (± 5.5%) i/s -     36.554M
 Array#reject! * 11      7.328M (± 5.3%) i/s -     36.481M
 Array#reject! * 12      7.325M (± 3.2%) i/s -     36.579M
 Array#reject! * 13      7.319M (± 4.1%) i/s -     36.522M
 Array#reject! * 14      7.335M (± 5.5%) i/s -     36.527M
 Array#reject! * 15      7.338M (± 6.0%) i/s -     36.480M
 Array#reject! * 16      7.405M (± 5.2%) i/s -     36.901M
 Array#reject! * 17      7.151M (± 4.8%) i/s -     35.624M
 Array#reject! * 18      7.378M (± 5.7%) i/s -     36.714M
 Array#reject! * 19      7.414M (± 4.4%) i/s -     36.952M
```

Seems to have fixed the non-linear regression by this benchmark.

#21 - 01/11/2015 12:50 PM - akr (Akira Tanaka)

I think nobu has a right to accept this feature because the documented bahavior which will be changed was implemented and committed by nobu and others don't requested.

If nobu don't think so, we need approval of matz or naruse.

#22 - 01/11/2015 03:18 PM - nobu (Nobuyoshi Nakada)

I negate the part "others don't requested", ... somebody requested it, ... perhaps.

#23 - 01/14/2015 08:59 AM - matz (Yukihiro Matsumoto)

- Assignee set to akr (Akira Tanaka)

I agree having linear performance is crucial, and the proposed new behavior is acceptable.
Go ahead, and make the change.

Matz.

#24 - 01/14/2015 09:06 AM - akr (Akira Tanaka)

Thank you, matz.

Please commit, nobu.

#25 - 01/15/2015 01:45 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Closed

- % Done changed from 0 to 100

Applied in changeset r49255.

---

array.c: linear performance

- array.c (rb_ary_select_bang, ary_reject_bang): linear performance. [ruby-core:67418] [Feature #10714]