

Re: Framework 2.0 array redim unsatisfactory performance

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.vb/2005-08/msg01731.html>

- *From:* "Jay B. Harlow [MVP – Outlook]" <Jay_Harlow_MVP@xxxxxxxx>
 - *Date:* Thu, 11 Aug 2005 22:37:19 -0500
-

Tom,
| I am aware of GC issues.

IMHO No you are not aware of the GC issues per se, otherwise we would NOT be having this discussion. :-|

| However, for the purpose of simple collection test
| it simply does not matter, since no object is released for GC,
Yes it does matter! As there are a number of objects allocated in both your original sample & my modified List(Of T) sample! These objects are implementation details of Redim, Redim Preserve, and List(Of T). Redim has a significantly higher object allocated overhead then List(Of T). HashTable & Dictionary(Of K, T) would have higher overhead then List(Of T), but significantly lower then Redim. For Integers Hashtable will have significantly higher object allocated overhead then Dictionary(Of K,T) as Hashtable will box all the integers going in. Generics, such as Dictionary(Of K,T), eliminate this boxing penalty. Which is why List(Of T) should be preferred over ArrayList.

Remember that System.Array in .NET is a full fledged object. When you do a Redim Preserve a new object is allocated, the old object is copied to the new object, then the old object is discard. ReDim simply allocates a new object & discards the old object. Which means that 99,999 objects were released for GC!

In your original sample 100000 objects are allocated! You can use CLR Profiler to confirm the fact, unfortunately your sample takes a long time to run.

```
Public Sub Main()  
Dim foos() As Long
```

```
Dim n As Long  
n = 100000  
Dim i As Long  
For i = 1 To n
```

Re: Framework 2.0 array redim unsatisfactory performance

```
ReDim Preserve foos(i)
foos(i) = i
Next i
Debug.Print(foos(1000))
End Sub
```

If you change your sample to use List(Of T)

```
Public Sub Main()
Dim foos As List(Of Integer)

Dim n As Integer = 100000

For index As Integer = 1 To n
foos.Add(index)
Next

End Sub
```

Remember List(Of T) is implemented internally as an Array, not a linked list.

Then I would estimate there would only about 15 objects allocated. The List(Of T) itself, plus 14 generations of buffers. Remember the buffer (the internal array) starts out at 16 & doubles... that means that 13 objects were released for GC.

Generation – Capacity

1	–	16
2	–	32
3	–	64
4	–	128
5	–	256
6	–	512
7	–	1,024
8	–	2,048
9	–	4,096
10	–	8,192
11	–	16,384
12	–	32,768
13	–	65,536
14	–	131,072
15	–	262,144
16	–	524,288
17	–	1,048,576

Of course if you start out with a capacity of 100,000, then there would only be 2 objects allocated. The List(Of T) itself & its internal buffer.

Again one could use CLR Profiler to verify, I'll see if I have CLR Profiler loaded on my VS 2005 VPC, & report back later...

Re: Framework 2.0 array redim unsatisfactory performance

Re: Framework 2.0 array redim unsatisfactory performance

| since no object is released for GC,
I would expect that to be true if we were talking about LinkedList(Of T),
however we are talking about Array & List(Of T) which are both based on an
Array of values.

Hope this helps
Jay

"Tom Jastrzebski" <tom@xxxxxxx> wrote in message
[news:nuSKe.89\\$A%3.24@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx](mailto:news:nuSKe.89$A%3.24@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx)

| Jay,

| I am aware of GC issues. However, for the purpose of simple collection
test

| it simply does not matter, since no object is released for GC, and no
memory

| deallocated until the application exits. Of course, that is not true for
the

| "array redim" test, but I do not even attempt to test its memory
efficiency.

| There also might be an exception for Hashtable with too small initial
| capacity, but I would not bet on this.

| But, if I you would like to continue, I am wondering what memory overhead
| your tests show. Let's say List(Of Integer) with one million elements.

| My tests show that static arrays are surprisingly efficient, virtually no
| overhead. Dynamic structures, however, is the whole different story.

| Cheers,
| Tomasz

| "Jay B. Harlow [MVP – Outlook]" <Jay_Harlow_MVP@xxxxxxx> wrote in message
news:ODzLi1hnFHA.2484@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

| > Tom,

| > | Process.GetCurrentProcess().WorkingSet64

| > WorkingSet64 is the Working Set or total amount of physical memory in
use,

| > this is by no means the amount of memory the GC has allocated. In other
| > words it includes objects that have not yet been garbage collected, plus

| > it

| > includes the space for objects that were previously allocated but have
| > already been collected.

| >

| > As I stated, the GC over allocates physical memory as needed & will only
| > return it when the OS requests it as another app is asking for more

| > memory... In other words the Working Set includes both allocated memory
&

| > free memory. You need to use the .NET performance counters or CLR
Profiler

Re: Framework 2.0 array redim unsatisfactory performance

|> to see how much memory is currently allocated.
|>
|> | What I am looking for is dynamic data structure which has the least
|> memory
|> | overhead.
|> That would be List(Of T) with the "capacity" constructor set to the
|> expected
|> number of elements. For example if you know there are going to be 50 to
75
|> elements in the list, but no more then 100. I would pass 50 or 75 to the
|> constructor. With 50, there would be at most a single reallocation.
|>
|> If your not sure what the limit is going to be, but once the list is
|> filled
|> its "constant", the I would recommend using List(Of T).TrimExcess when I
|> finished filling the list.
|>
|> [http://msdn2.microsoft.com/library/ms132207\(en-us,vs.80\).aspx](http://msdn2.microsoft.com/library/ms132207(en-us,vs.80).aspx)
|>
|> Hope this helps
|> Jay
|>
|> "Tom Jastrzebski" <tom@xxxxxxx> wrote in message
|> [news:qNxKe.68\\$UA1.27@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx](news:qNxKe.68$UA1.27@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx)
|> | Jay,
|> |
|> | I just noticed something I thought was interesting and decided to
write
|> | about it, thinking that someone migrating from VB 6.0 may experience
|> this
|> | issue, regardless of what the root cause is and whether these
|> environments
|> | should be compared or not.
|> |
|> | As I said, the only reason why I tried "array redim" was because I was
|> | hoping to be nicely surprised – and I was not.
|> |
|> | What I am looking for is dynamic data structure which has the least
|> memory
|> | overhead.
|> | It seems like at some point I will have to develop my own. But that is
|> OK,
|> | it by no means diminish the value of .Net Framework.
|> |
|> | To check process memory utilization I use:
|> | Process.GetCurrentProcess().WorkingSet64
|> |
|> | Peace,
|> | Tomasz
|> |
|> |

|>
|>
|
|

• **Follow-Ups:**

- ◆ **Re: Framework 2.0 array redim unsatisfactory performance**
◇ From: Tom Jastrzebski

• **References:**

- ◆ **Framework 2.0 array redim unsatisfactory performance**
◇ From: Tom Jastrzebski
 - ◆ **Re: Framework 2.0 array redim unsatisfactory performance**
◇ From: Jay B. Harlow [MVP – Outlook]
 - ◆ **Re: Framework 2.0 array redim unsatisfactory performance**
◇ From: Tom Jastrzebski
 - ◆ **Re: Framework 2.0 array redim unsatisfactory performance**
◇ From: Jay B. Harlow [MVP – Outlook]
 - ◆ **Re: Framework 2.0 array redim unsatisfactory performance**
◇ From: Tom Jastrzebski
 - ◆ **Re: Framework 2.0 array redim unsatisfactory performance**
◇ From: Jay B. Harlow [MVP – Outlook]
 - ◆ **Re: Framework 2.0 array redim unsatisfactory performance**
◇ From: Tom Jastrzebski
 - ◆ **Re: Framework 2.0 array redim unsatisfactory performance**
◇ From: Jay B. Harlow [MVP – Outlook]
 - ◆ **Re: Framework 2.0 array redim unsatisfactory performance**
◇ From: Tom Jastrzebski
- Prev by Date: **Accessing pointers from capSetCallbackOnFrame (avicap32.dll)**
 - Next by Date: **Re: SQL query from ASP page**
 - Previous by thread: **Re: Framework 2.0 array redim unsatisfactory performance**
 - Next by thread: **Re: Framework 2.0 array redim unsatisfactory performance**
 - Index(es):
 - ◆ **Date**
 - ◆ **Thread**