



Setting the ObjectStore Client Cache Size (OS_CACHE_SIZE)

By Laurent Poulain

PROGRESS
SOFTWARE

Table of Contents

Introduction	3
The tradeoff	3
Cache and Address Space	3
ObjectStore Optimum Cache Size	4
An example	4
Scalability	6
Operating System Optimum Cache Size	6
Checking the amount of available physical memory	6
Checking the amount of memory swapped (Windows)	7
Checking the amount of memory swapped (Unix)	7
Conclusion	8
About Progress Software Corporation	8



Introduction

One of the big advantages of ObjectStore is its patented Cache-Forward Architecture®. The purpose of the ObjectStore cache is to hold database pages in physical memory while they are actively used. This provides very fast access to the objects on those pages, as they are read from memory instead of the disk. The cache also holds recently used pages, so that they can get quickly accessed when the ObjectStore client needs them again. This eliminates the need to fetch the page from the ObjectStore Server, saving a page read from the disk and possibly from the network as well; net result, better performance.

Now, the question that a lot of customers ask is: what is the optimum cache size? Even though it highly depends on the ObjectStore application and the system on which it runs, there are some ways to determine tune the cache size.

In the rest of this white paper, the following terms will be used:

- 1) Cache: the ObjectStore cache used by the client
- 2) Client: the ObjectStore client application
- 3) Physical memory: RAM
- 4) Swap space or page file: part of the hard disk the operating system is using to complement physical memory
- 5) Virtual memory (or memory): any amount of memory used by a process or the system, whether physical or swap space

The tradeoff

Setting the right cache size is about a tradeoff between what ObjectStore wants and what the operating system can provide. Set too small a cache, and ObjectStore will need to fetch the same pages multiple times to perform a single task. Fetching a cache page from disk will be slower than accessing it in physical memory. Set it too large and performance will suffer due to system load. Indeed, if the system runs out of physical memory, the operating system might swap part of the cache to disk, which negates the advantage of a cache.

Cache and Address Space

Customers who are using a large PSR (Persistent Storage Region) might wonder if a large cache consumes address space, thus leaving less address space for the PSR.

Fortunately, the size of the cache has no impact on the amount of PSR available. The cache is only mapped inside the PSR. It consumes physical and virtual memory, but does not consume address space.

Note that on Windows, the cache cannot be larger than the PSR (doing so would lead to an error, e.g. “assertion 0 was violated in file \os\top\nclient\nfmap_ctl.cc, at line 425.”)

ObjectStore Optimum Cache Size

The optimum cache size for ObjectStore is the cache size which doesn't require a client to fetch the same page twice within its lifetime.

The best way to determine an optimal cache size is to try different cache sizes and compare how much data is read using ObjectStore utility `ossvstat` (ObjectStore Server Statistics). When measuring that data, it is important to make sure you are always looking at the same thing. So when trying several runs with different cache sizes, the client should always perform the same operation (e.g. go through the same collection). The `ossvstat` tool provides information about the server and individual clients. It is recommended that you look at the statistics for individual clients.

It is also important to remember that `ossvstat` provides information *from the ObjectStore Server*. The numbers reported by the tool are the number of pages that the server received from and sent to the client. Pages that the client updated but were discarded because the transaction was aborted are not necessarily sent to the server. Therefore those types of pages might not show up in `ossvstat` statistics.

An example

Let's examine the following test-case, composed of a loader and an updater

- ⇒ The loader creates a database, creates a collection and inserts 200,000 elements in that collection. Everything is done in one transaction, and generates a 16 Mb database.
- ⇒ The updater, using one single transaction, browses through the entire collection and updates each and every element.

Both clients pause after the transaction is complete and before exiting, allowing a chance to look at the client statistics section of the `ossvstat` output. Here is the relevant output for the loader with the default cache size of 8 Mb:

```
Client #3 (coll_put.exe/Global Session)
    557 messages received           0 callback messages sent
      0 callback sectors           0 succeeded sectors
263 KB read                     14474 KB written
      2 commits                     0 readonly commits
      0 aborts                       0 two phase transactions
      0 lock timeouts                0 lock waits
      0 deadlocks
```

Not surprisingly, not much data is read, 263 Kb (mostly the application schema), however the number of Kb written is significantly higher, 14 Mb. One might wonder why the database is 16 Mb large when only 14 Mb were sent to the server. The remaining 2 Mb is composed of database free space as well as ObjectStore metadata. These items are not reported by `ossvstat`.

When the updater is run with the default cache size of 8 Mb, the `ossvstat` output is quite surprising:

Client #3 (update.exe/Global Session)

51113 messages received	8 callback messages sent
0 callback sectors	0 succeeded sectors
174469 KB read	173234 KB written
1 commits	0 readonly commits
0 aborts	0 two phase transactions
0 lock timeouts	0 lock waits
0 deadlocks	

The updater read and wrote 12 times as much data (171 Mb) as was written during the whole load process! How can this happen? It happens because the cache is too small, and so it filled up.

When the cache is full, it needs to evict some pages before being able to fetch more pages. If a page has been modified, it needs to be sent to the server to be held there until the transaction commits. This would not be a problem if the client did not need to read those pages later in time. In the above example, each page is on average read and sent back to the server 12 times!

When the cache size is increased to 14 Mb, we see a much more reasonable output:

Client #3 (update.exe/Global Session)

9250 messages received	0 callback messages sent
0 callback sectors	0 succeeded sectors
22713 KB read	21650 KB written
1 commits	0 readonly commits
0 aborts	0 two phase transactions
0 lock timeouts	0 lock waits
0 deadlocks	

The amount read and written is still larger than the whole database size itself. If we increase the cache size further to 16 Mb:

Client #3 (coll_get.exe/Global Session)

6923 messages received	0 callback messages sent
0 callback sectors	0 succeeded sectors
14345 KB read	13286 KB written
1 commits	0 readonly commits
0 aborts	0 two phase transactions
0 lock timeouts	0 lock waits
0 deadlocks	

The amount of data read and written is now smaller than the database size. Increasing the cache size further will not change the result. This implies that the updater is using no more than 16 Mb of cache. Thus the optimum cache size for this application is approximately 15-16 Mb.

Scalability

Sometimes increasing the cache size is not a scalable solution. The above example indeed requires a cache large enough to fit the whole collection. What if the collection size keeps increasing? This is why a large cache does not remove the need to have a good application design. Applications will get their best performance if their working set of database pages is as small as feasible. A small working set size maximizes both raw speed and concurrency when multiple clients are working in the same database.

Increasing the cache size can be the most expedient solution. However, there are times when the cache size is not the problem. Rather, it is necessary to tune the application in order to reduce the working set size, or possibly redesign the database layout for how objects are clustered/partitioned. Once this redesign is finished, an optimum cache size will again bring its benefits.

For further information about application tuning, please refer to the “Coding Applications on Cache-Forward Architecture” white paper on our Web site (<http://www.progress.com/realtime/publications/index.ssp>).

Operating System Optimum Cache Size

Using a cache large enough to contain a whole database does not sound like a bad idea, but it can negatively impact performance if the operating system cannot handle it. Indeed, if the operating system runs out of available physical memory, it might move the cache to its swap space, which is counter-productive. Fetching cache pages from a local disk drive is much slower than reading them from physical memory, even if it might be a little faster than fetching them from an ObjectStore server.

An associated risk with too large a cache is virtual memory thrashing, where processes have so little physical memory available that they can only swap in pages by swapping out other pages that will need to be retrieved later.

The ObjectStore cache is only consuming memory when it is actually being used. For example, an ObjectStore client with a 300 Mb cache will only consume 10 Mb of cache memory if only 10 Mb of the cache is being used. A cache can nevertheless fill up over time. Pages indeed stay indefinitely in the cache unless the client exits or explicitly clears all or parts of its cache using `objectstore::return_memory()` or `objectstore::return_all_pages()`.

Although there is no silver bullet to determine when a cache is too large for the operating system, there are some diagnostic tools available. Even with these tools, the approach is, once again, trial and error. Try a given value for the cache size and monitor the result.

It is important to remember that the swapping behavior of the operating system is highly dependent on all the other processes running on the system. When doing some measurement about swap space, one must always keep in mind that there will always be fluctuations because background processes seldom consume a constant amount of memory over time, particularly on a live system.

Checking the amount of available physical memory

The first thing to look at is the decrease of available physical memory when the ObjectStore application starts and fills up the cache. Compare that to the actual amount of memory consumed by the application.

To check the memory of a process on Windows, use the Task Manager (Ctrl-Shift Escape), click on the “Processes” tab, and look at the “Mem Usage” column. On Unix, run command line ‘top’ and look at the SIZE column.

For instance, if the host has 400 Mb of physical memory available before starting the application and 200 Mb of physical memory left after, this means that the application is using 200 Mb of physical memory. Now, if the application is consuming 400 Mb of virtual memory, we know that 200 Mb is being put into the swap file. If on top of that the

ObjectStore cache is 300 Mb, we *definitely* know it does not entirely stay in physical memory.

This method has however a big caveat. If other processes happen to consume more memory during the test (like the ObjectStore server if it runs on the same host than the client), then looking at the amount of physical memory available on the system can be misleading.

Checking the amount of memory swapped (Windows)

On Windows, it is possible to know the amount of memory in the swap (also known as page file) for a given process. To determine this:

- ⇒ Start the Windows Task Manager (Ctrl-Shift Escape)
- ⇒ Click to the “Processes” tab
- ⇒ On the Task Manager menu, select View / Select Columns... and make sure that “Mem Usage” and “VM Size” are checked
- ⇒ Look at those two columns for your ObjectStore client

“VM Size” indicates the complete amount of virtual memory that the process is consuming (RAM + page file). “Mem Usage” indicates the amount of RAM used by the process. Therefore, the difference between the two is the amount of process memory which is paged.

If it is not possible to determine whether it is the cache part of the process memory which is in the page file, a large ObjectStore cache should represent a large portion of the process total memory. If too much of the memory is paged, chances are that part of it is the ObjectStore cache.

Checking the amount of memory swapped (Unix)

On Unix, the way to determine the amount of memory in the swap is by using the command ‘top’.

In the ‘top’ output, the SIZE column indicates the amount of virtual memory consumed by each process. The RES column indicates the amount of resident memory (or physical memory) consumed by each process.

So if the RES value is pretty much the same as the SIZE value, it means that the whole application (ObjectStore cache included) sits mostly in physical memory.

Just like in the Windows world, it is not possible to determine whether it is the cache part of the process memory which is in the swap file. But in the case of a large cache, too much of the process virtual memory in the swap isn’t a good sign.



Conclusion

There are several ways to determine if the cache size should be increased or decreased. But one should never forget that, in a real world environment, things constantly change. The database grows larger so more data needs to be processed. More clients are added. Or some third party software is installed on the host, consuming more resources. This is why all the methods described above should be considered as guidelines and not hard rules.

About Progress Software Corporation

Progress Software Corporation (Nasdaq:PRGS) is a global industry leader providing application infrastructure software for all aspects of the development, deployment, integration and management of business applications through its operating units: Progress OpenEdge Division, Sonic Software, DataDirect Technologies, and Progress Real Time Division. Headquartered in Bedford, Mass., Progress can be reached at www.progress.com/realtime or +1-781-280-4000.

PROGRESS
SOFTWARE

www.progress.com/realtime

Worldwide and North American Headquarters
Progress Real Time Division, 14 Oak Park, Bedford, MA 01730 USA Tel: +1 781 280 4000

UK and Northern Ireland
Progress Real Time Division, 210 Bath Road, Slough, Berkshire, SL1 3XE England Tel: +44 1753 216 300

Central Europe
Progress Real Time Division, Konrad-Adenauer-Str. 13, 50996 Köln, Germany Tel: +49 6171 981 127

France
Progress Real Time Division, 3 Place de Saverne, Les Renardières B, 92901 Paris la Défense Tel: +33 1 41 16 16 56

© 2006 Progress Software Corporation. All rights reserved. Progress, Real Time Division and Cache-Forward are trademarks or registered trademarks of Progress Software Corporation, or any of its affiliates or subsidiaries, in the U.S. and other countries. Any other trademarks or service marks contained herein are the property of their respective owners. Specifications subject to change without notice. Visit www.progress.com/realtime for more information.