Making the Most of the Client Cache Engine

A White Paper From

GOLDSTAR SOFTWARE

www.GoldstarSoftware.com

For more information, see our web site at
http://www.goldstarsoftware.com

Making the Most of the Client Cache Engine

Last Updated: 03/07/2016

Newer versions of PSQL (and Btrieve v12) include a little-used component known as the Client Cache Engine, or CCE. In this white paper, we are going to see how to configure the CCE for use in your environment, so that you can evaluate whether it is appropriate for your users or not.

What is the Client Cache Engine?

So, what is the Client Cache Engine, anyway? As its name implies, the Client Cache Engine is a database *engine* that runs in memory on the *client* side of a client-server environment, *caching* data for use by the applications running on the workstation. The CCE uses the exact same code as the Workgroup Engine, and it also uses the same executable name (W3DBSMGR.EXE). However, the CCE works VERY differently from the Workgroup Engine and Server Engine.

How Does the Client Cache Engine Work?

Internally, the PSQL engine stores database records inside database files on "data pages", and it keeps index information on "index pages". (The actual page size is determined by the application developer.) Under normal database operation, requests to read or write records are sent across the network from the client workstation to the server, they are processed by the server, and then the response is returned over the network wire back to the client computer. This mechanism allows for the centralized database engine to keep track of all reads and writes from each user, resolving multi-user access issues, caching data to reduce disk accesses, and so on.

However, what happens if the application on a workstation reads the SAME record over and over again? Each time the record is read, the client must send the request to the server and wait for a reply. Wouldn't it be great if the client could just keep a copy of the record in memory for future reference?

The CCE does this – but it even goes one step further. Instead of just reading one record at a time, it actually reads an entire data file page, which could be 1K, 4K, or even as much as 16K at a time. With larger pages, this one network request may actually provide a data page with numerous database records on it. The entire page is then kept in local memory (i.e. cache) for future use. Subsequent reads that need to access this same page are then handled locally – by the CCE – which eliminates the need to send another network request and wait for a reply.

Database writes are handled a bit differently. When the application attempts to insert, update, or delete a record, or if a record is being locked as part of a specific operation or within a transaction, the CCE is bypassed. The request is always transferred to the server for processing so that the data can be written to the common database file, and allow other users to see the newly-written data. (As you would expect, this is why write-intensive applications are not expected to see substantial benefits with the CCE.)

One caveat – the CCE will only provide a cache for the Btrieve (Transactional) interface database calls. If your application is making SQL calls to retrieve data, then these are always handled by the engine on the database server instead, and the CCE will not be used.

Advantages to Using the Client Cache Engine

The biggest advantage to the CCE is that it can eliminate round trips across the network. If an application reads the same common record (or records) over and over again via the Btrieve (Transactional) interface, it needs to only go across the wire ONE time, and the rest of the times the CCE is able to satisfy that request from the local cache. For "chatty" applications that keep reading the same configuration record over and over again, this local cache can reduce network and system workload considerably.

Further, let's look at an application that is reading database records in order, one after the next. If the data file page size is 4K and each data page contains 10 records, then only one network round trip is required to read the 4K block -- and all 10 records come over at the same time. Subsequent records are satisfied from the cache, eliminating 90% of the networking overhead for those 10 record reads! [Astute readers may notice that the total amount of data may not have reduced by much, but because each individual network request has overhead and incurs a round trip delay, the net result is still better performance overall.]

Disadvantages to Using the Client Cache Engine

With data in memory on both the server AND on the workstation, the CCE and the PSQL engine on the server must keep track of what data is in memory on each machine at all times. If the data is already on the workstation, then the CCE can serve up the data locally without involving the server. Because of this extra effort, both the client AND the server will incur extra expense in tracking which pages are in the cache, and this can increase the amount of memory needed on the server accordingly.

Another potential issue is cache consistency. As noted previously, database write operations are sent back to the database server for processing. The server than keeps track of the changed pages, and notifies each active CCE of pages in memory that are now "invalid". Each CCE must then re-read these pages to get current information. But, what happens during the small time window (which can be as much as 5-10 seconds) where the CCE cache still contains old data, and the server hasn't yet notified the CCE that the page is invalid? The answer is simple – the CCE still returns the OLD data! In most cases, a small timing issue won't have any real impact on the environment, and viewing old data will not impact most applications. (Because record locks and all writes are handled by the server, applications using active locking are usually immune to this issue.)

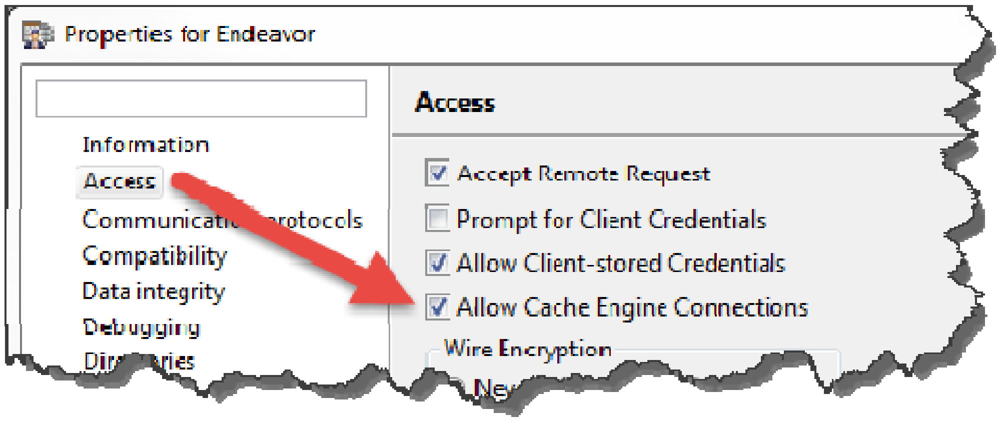
However, it is certainly possible that the use of old data can cause issues for certain applications. Imagine an application that is trying to save an invoice number. It queries the Invoice table to find the last invoice, and it finds Invoice 17555. So, it decides to add one to that number and create Invoice 17556. However, someone else has already created 17556 and written it back to the server, so the attempted insert (which is processed on the server) returns a Status 5, *Duplicate Key Error*. Since the application may be expecting this duplication (as most multi-user applications can get into such a state), it loops around to re-read the last invoice number again. Because this data is still in the cache, it again gets 17555, adds 1, and attempts to save 17556 again, which also returns an error. Maybe it loops two more times, getting errors both times, before it just gives up, returning an error to the user and quitting with some cryptic error message.

The solution, of course, is to allow additional retries, or to allow for a different mechanism for handling invoice numbers (such as an AutoInc key). However, this would require code changes by the developer, and this may be outside of what you can get accomplished if you are merely the end user of the application. In this case, your application may be **incompatible with the CCE**, and you will not be able to reliably use this feature. If you find this to be the case, you should disable the CCE on your environment completely.

Since Actian has no idea which applications are compatible (or not compatible) with the CCE, they opt to leave the CCE **disabled** by default, as this is the safest option. It is up to you, the end user, to determine if your application is compatible and to enable this feature if you want to use it.

Enabling the Client Cache Engine on the Server

By default, the CCE is **enabled** on the PSQL database engine. You can confirm this by going to the server, launching the *PSQL Control Center*, and clicking on the **Configure Local Engine** link in the **Welcome** screen. Open the **Access** tab, and you should see the **Allow Cache Engine Connections** item checked, as shown here.

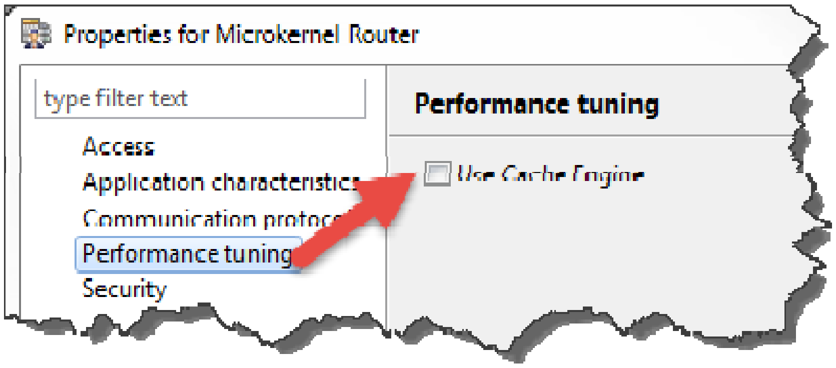


[✓] Accept Remote Request
[ ] Prompt for Client Credentials
[✓] Allow Client-stored Credentials
[✓] Allow Cache Engine Connections

If the box is unchecked, check it and restart your engine to enable the CCE.

Enabling the Client Cache Engine on the Client

By default, the CCE is **disabled** on most PSQL client workstations, which is done purposefully by Actian as a safety measure (as discussed above). You can change this setting by going to the client workstation, launching the *PSQL Control Center*, and clicking on the **Configure Microkernel Router** link in the **Welcome** screen. Open the **Performance Tuning** tab and look for the **Use Cache Engine** item.



If the box is unchecked, check it and click OK to close. While this setting does not require a restart, it is recommended that you reboot the machine anyway, especially if you are working on a terminal server, as if the client modules are already loaded, they may not see your new setting without a restart. With both the Server and Client configured to allow the CCE, it should now be active in your environment.

Important Note: If you do want to use the CCE on a Terminal Services or Citrix Server, it is imperative that the PSQL Client be installed as a *Service*, NOT as an *Application*. With the PSQL v12 Client, the default is to install as a Service, so this should already be working for you. However, the older PSQL v11 Client installed as an Application by default, and therefore you may need to fix this. If you find that the Client is installed as an Application, you must uninstall the PSQL Client and reinstall it as a Service.

Confirming that the Client Cache Engine Is Working

Once you enable the CCE, you can use Task Manager to monitor it as it starts to use up more and more memory. The CCE itself is the process names "w3dbsmgr.exe", and will be visible as shown here in yellow. (You may need to add or remove columns on your own system to see these same values.)

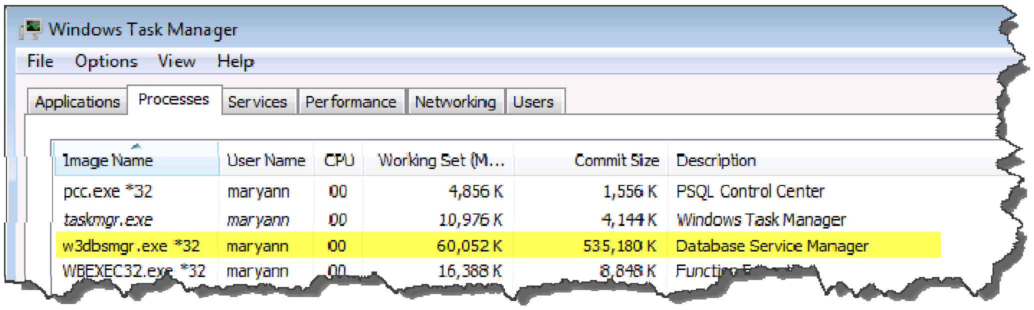


Image Name	User Name	CPU	Working Set (M...	Commit Size	Description
pcc.exe *32	maryann	00	4,856 K	1,556 K	PSQL Control Center
taskmgr.exe	maryann	00	10,976 K	4,144 K	Windows Task Manager
w3dbsmgr.exe *32	maryann	00	60,052 K	535,180 K	Database Service Manager
WBEXEC32.exe *32	maryann	00	16,388 K	8,848 K	Functi

In this case, the CCE has been initialized, and the **Commit Size** column shows us how much memory has been allocated for the cache and its tracking information. In this case, this is a machine with a 500MB L1 cache, and it is going to grab 535MB of memory. However, at this time, the CCE has not been loaded up with data, as the **Working Set** for this process shows only 60MB. If you now carefully monitor the Working Set as the application fires up and starts reading data from the server, you will see this memory counter increase, indicating that data is starting to get saved in the local cache by the CCE.

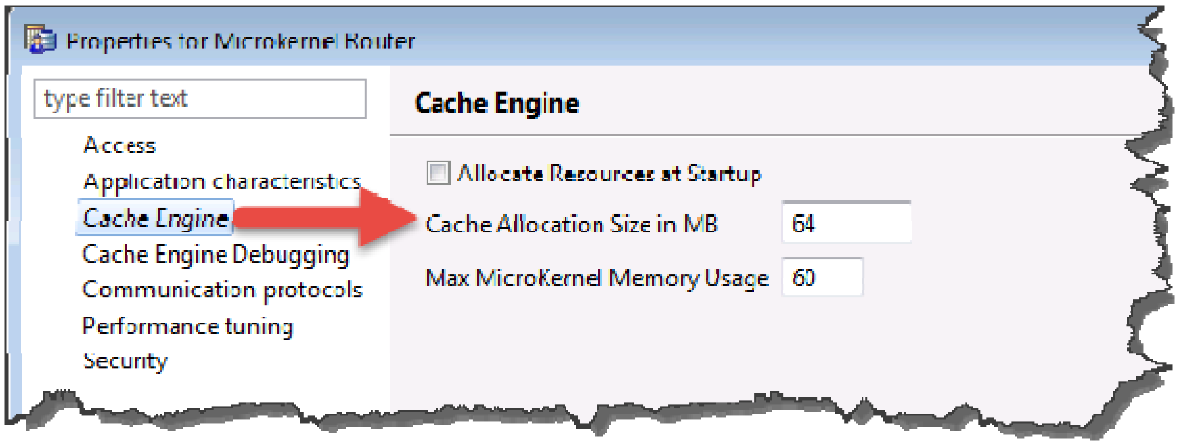
Improving Performance by Using a Larger Cache

Like with the database engine on the server, the CCE can benefit from a much larger database cache. The default value of 64MB will work, but this is really kind of small. We expect that Actian selected this number to provide some benefits to applications without using up *too much* memory on workstations. Before making any changes, however, you should first evaluate the resources on computer and answer a few basic questions:

- How much *free* memory is available on the computer?
- What operating system bit level is currently in use, 32-bit or 64-bit?

Once you know how much free memory that you have, you can now determine how much you can allocate to the CCE. If you have a large database, and if you have many users sharing access to that database from this computer (most important on Citrix or Terminal Server computers), then you will definitely want to increase the size of the CCE cache to get better performance.

You can find the CCE configuration screen by going to the client workstation, launching the *PSQL Control Center*, and clicking on the **Configure Microkernel Router** link in the **Welcome** screen, and open the **Cache Engine** tab:


Properties for Microkernel Router

type filter text

- Access
- Application characteristics
- Cache Engine
- Cache Engine Debugging
- Communication protocols
- Performance tuning
- Security

Cache Engine

☐ Allocate Resources at Startup

Cache Allocation Size in MB	64
Max MicroKernel Memory Usage	60

The **Cache Allocation Size in MB** setting is the L1 database cache memory to be allocated on the workstation (for the CCE). Increase this up to the size of your database (or to the amount of available memory you can spare). The **Max Microkernel Memory Usage** is the L2 cache, and we currently recommend that this be disabled (set to 0) for best results. (We may change this recommendation in the future as further analysis is completed.)

If you are running a 32-bit operating system, then you will be limited to a total addressing space of 2GB for the CCE process. You can safely allocate up to approximately 1.4GB of memory to the L1 cache. If you are running a 64-bit operating system, then the 32-bit CCE process will be limited to a maximum of 4GB of addressing space, and you can safely increase the L1 cache to as high as 3GB at most.

These numbers are designed to be “safe” starting points. Whether you are using a 32-bit or 64-bit system, special care must be taken to ensure that you **never** exceed the addressing space for the 32-bit CCE process, as doing so can cause the CCE to stop working or can cause it to crash outright, causing an outage for users. (See our white paper on **Understanding Server Memory Counters** for more details on how to monitor the actual addressing space of the W3DBSMGR process.)

Improving Performance by Holding Files Open

One of the problems with the Client Cache Engine is that it will ONLY keep data in the client-side cache *while the database files are open by that client computer*. We suspect that this was originally implemented to help streamline system operation for situations where the data files are getting replaced on the server. However, this retention policy has a downside – if a user runs a large report and populates 1GB of data into the CCE cache, then closes it and runs the report a second time, the second run will be forced to re-read all of the data again. Re-reading the data nullifies the substantial benefits of the CCE!

Luckily, there is an easy way to address this issue. If you have a set of database files that will be frequently accessed from the workstation, you can use some other process to sit on the computer and hold the files open. Some applications will do this for you automatically, but many will not. For those that do not keep files open, you can do this manually with the Function Executor (which is handy for testing purposes), or you can use a special tool like Goldstar Software's *HoldOpen* utility, which can hold files open on the workstation so that the cached data stays valid. You can get more information about this tool from www.goldstarsoftware.com/tools.asp.

For a typical end-user workstation, this is probably overkill and can lead to more problems if users forget to shut down the tool at the end of the day, impacting backups. However, for a Citrix or Terminal Server with many users sharing the same cache engine, holding the database files open for everyone using that terminal server can provide some substantial performance benefits.

Conclusions

The Client Cache Engine (CCE) can be a bit complicated to get started with, and it can definitely cause problems for some applications which have a design that makes them incompatible with the CCE. However, if your application supports the use of the CCE, then it can also provide some great performance gains for systems used for extensive reporting or users of terminal servers, and it may be well worth the effort to configure it properly.

If you still can't get it to work, contact Goldstar Software and let us work with you to help! Please note that this may be a billable support call if you have already used up your free support time.