The z/OS® Language Environment® (LE) component provides a common runtime environment for the IBM version of certain high-level languages. LE provides runtime options that can be customized according to the programs behaviors to achieve better execution performance. This tutorial describes an LE heap storage tuning method for IBM's InfoSphere® Data Replication for DB2® for z/OS (Q Replication). The tuning reduces contentions of concurrent heap storage allocation requests among multiple threads of the Q Capture and Q Apply programs of Q Replication for z/OS, while keeping the heap storage overall allocation to a minimum.
After applying the heap tuning techniques outlined here, a notable 13-percent throughput performance was achieved for OLTP type workloads and CPU reduction was noted for all workload types.

### Introduction

#### LE heap storage

Language Environment (LE) is an IBM solution that provides a common environment for all LE-conforming high-level language (HLL) products. LE establishes a common runtime environment for all participating HLLs by combining essential and commonly used runtime services, such as routines for runtime message handling, condition handling, storage management, date and time services, math functions, and making these services available through a set of consistent interfaces to all supported programming languages. With LE, a single runtime environment can be used for multiple applications, regardless of their programming languages, as long as they are all LE-conforming.

LE provides services that control the stack and heap storage used at runtime. LE-conforming HLLs and assembler language routines use these services for all storage requests. Heap storage is used to allocate storage that has a lifetime not related to the execution of the current routine; it remains allocated until explicitly freed or until the enclave terminates. An application's heap storage can be tuned using the LE runtime options HEAP, THREADHEAP, and HEAPPOOLS.

### Q Replication

Q Replication is a high-volume low-latency replication solution that uses WebSphere® MQ message queues to transmit transactions between source and target DB2 databases or subsystems. Q Replication has the following two main processing components:

- **Q Capture** reads the DB2 recovery log for changes that occur in source tables, turns the changes into messages, and sends the messages over WebSphere MQ queues for processing by a Q Apply program.
- **Q Apply** receives messages sent by Q Capture from WebSphere MQ queues and rebuilds the transactions that the messages contain and applies the transactions to target DB2 tables.

DB2 logs row changes (inserts, updates, or deletes) in the recovery log. The Q Capture program calls the DB2 IFI306 interface to read log records from the recovery log. If a log record is for a table that is part of an active Q subscription (a subscription is the user's means of indicating an interest in replicating the changes for a table), the Q Capture program creates a list of rows in heap storage areas for the transaction. If the transaction terminates with an abort log record, the transaction is discarded. After reading a commit log record, the Q Capture program builds an MQ message with the rows in the transaction and puts the messages onto the MQ send queues under syncpoint. Each captured DB2 transaction from the log is transmitted using one or more MQ messages.

After a period of time specified by the `COMMIT_INTERVAL` parameter, the Q Capture program issues an MQ commit call to the WebSphere MQ queue manager, instructing it to transmit the messages on the send queues to the receive queue of the Q Apply program or user applications.
The storage areas used by Q Capture to hold changed row information reside in LE-managed heap storage, which is part of the address space's private region area. Q Capture has a MEMORY_LIMIT configuration parameter, which governs the maximum amount of memory the Q Capture program can allocate for the changed row information. If not specified by the user, Q Capture determines the MEMORY_LIMIT based on the amount of storage available in the address space's private region area at startup.

In large enterprises, billions of rows might be replicated each day, the nature of the replication task, the high variance in transaction and row sizes, as well as variations in throughput demand a heavy and dynamic usage of heap storage.

This tutorial focuses on LE heap storage tuning for Q Capture. Although the same tuning method applies to the Q Apply program, Q Apply program gained less performance improvement compared to Q Capture program. Unlike Q Capture, Q Apply ships with a default HEAPPOOLS LE runtime option, and the introduction of a HEAPPOOLS runtime option to Q Capture with our tuning methodology is the main source for Q Capture's performance boost.

The first part introduces the basic concepts of LE's heap management services. Based on these concepts, the second part describes a Q Capture-specific LE heap storage tuning method involving heap and HEAPPOOLS runtime options that would improve Q Capture's executing efficiency. In the final part, a result of the tuning effort of our simulated workload is laid out to demonstrate the performance improvement.

**Basic concepts of heap storage**

Heap storage is a collection of one or more heap segments comprising an initial heap segment, which is dynamically allocated at the first request for heap storage, and one or more heap increments allocated as additional storage is required.

Each heap segment is subdivided into individual heap elements and unallocated space. Heap elements are obtained by calling one of the heap allocation functions and are allocated within the initial heap segment by z/OS LE storage management routines. When the initial heap segment becomes full, LE gets another segment or increment from the OS. Given that elements vary in size, segments can become fragmented over time. All segments are located in the user private region of the address space that the application is running in.

There are 24-bit below-the-line heaps, 31-bit above-the-line heaps, and 64-bit above-the-bar heaps. Q Replication only uses 31-bit storage. It is possible to have multiple heaps from different areas within the same address space as they are requested by and allocated to specific enclaves. An enclave is a z/OS logical grouping of related threads that share resources.

Heap storage is used to allocate storage that has a lifetime not related to the execution of the current routine; it remains allocated until freed or its owning enclave terminates. Heap storage can be allocated or freed with any of the HLL storage facilities, such as malloc(), calloc(), or allocate, along with other LE storage services. Heap storage usage can be tuned to fit an application's specific needs by using some LE runtime options.
The initial heap is dynamically allocated by LE and does not require an explicit LE service call for its allocation. The initial heap segment is identified by heap_id=0. It is also known as the user heap. See Figure 1 for an illustration of LE heap storage. Heap storage is shared among all program units and all threads in an enclave. Any thread can free heap storage. One element can be freed at a time with CEEFRST callable service. The heap can also be freed at once using CEEDSHP. However, the initial heap cannot be discarded.

**Figure 1. Illustration of LE heap storage**

---

**Methodology of LE heap tuning**

The z/OS Language Environment Programming Guide provides a well-documented tuning methodology and was a major source of material for this tutorial. As it is a living document, it should be referenced for any changes that may have occurred since the writing of it.

As heap extents are not contracted, allocating them will cause runtime overhand and heap fragmentation; for best performance, the initial heap segment should be large enough to satisfy all requests for heap storage of the application. The LE storage report generated by the RPTSTG (ON) runtime option shows how much heap storage is being used, how many heap segments are allocated, how the heap pools (HEAPPOOL) of different cell sizes are being used by a program, and the report also shows the LE-recommended values for the heap, ANYHEAP, BELOWHEAP, BELOWHEAP, and HEAPPOOLS runtime options. The heap pool algorithm is used to significantly increase the performance of heap storage allocations, especially in a multi-threaded application that experiences contentions for heap storage.
Steps for tuning Q capture

1. Generate an options report and a storage report for the Q Capture main program by specifying LE runtime options RPTOPTS(ON) and RPTSTG(ON) with a representative workload. The options report prints out current LE runtime options in effect for Q Capture main enclave, while the storage report of the Q Capture program provides detailed storage usage information for various LE storage classes, such as STACK, THREADSTACK, LIBSTACK, THREADHEAP, HEAP, HEAPPOOL, and BELOWHEAP.

2. Examine the storage generated report. For basic heap storage tuning, we will be focusing on sections in the report related to heap storage usage for HEAP, ANYHEAP, and BELOWHEAP. The total amount of heap storage allocated, and the number of heap segments allocated for each type of heap storage will be looked at and analyzed. The major goal for tuning these heap options is to have "Number of segments allocated" to be as close to 1 as possible. If storage consumption is a concern, make sure initial and incremental sizes of these options are not set too big to avoid heap storage over-allocation. The values listed as suggested in the storage report are the LE recommended values based on the storage usage statistics it gathered in real time.

3. Change the HEAP/ANYHEAP/BELOWHEAP runtime options to use the LE-suggested values recorded by means of overriding the LE runtime options. Re-run the Q Capture program with the same workload and reprint the storage report to determine whether further adjustments to the heap storage options are needed. If so, repeat this step until you're satisfied. Different runs of the same application may have slight differences on heap storage usage. How Q Capture uses heap storage depends highly on the workload being replicated. If you have different sets of workload with very different heap usage patterns, consider the possibility of having matched sets of tuned LE runtime options for each workload.

4. After a satisfying period of testing on test environment, the new LE runtime options can be put into production environment. Note that the test environment should be as close to the production environment as possible. Make sure that no options report or storage report is being generated in production. The runtime option RPTSTG(ON) with which the storage reports are generated can have a negative impact on the application performance because as the application runs, statistics are recorded on storage requests.

Using HEAPPOOLS to improve performance

HEAPPOOLS is an optional heap storage allocation algorithm for C/C++ applications that is much faster than the normal malloc()/free() algorithm in most circumstances. The algorithm is designed to avoid contention for heap storage in a multi-threaded application. The heap pool algorithm allows for between one and 12 sizes of storage cells allocated from pools out of the heap. For each size, 1-255 pools can be created, where each pool is used by a portion of the threads for allocating storage. The HEAPPOOLS runtime option is used to specify the sizes of the cells, the number of pools for each cell size, and the size of cell pool extents for storage between the 16MB line and 2GB bar. The HEAPPOOLS option is also used to enable the heap pools algorithm, and HEAPPOOLS64 is used for storage above the 2GB bar as HEAPPOOLS. Since Q Replication programs currently don't support usage of storage above 2GB bar, we will be discussing HEAPPOOLS only here.
Why Q Capture should use HEAPPOOLS

The LE programming guide explains applications that should use heap pools:

1. Multi-threaded applications: although single-threaded applications can benefit from the heap pools algorithm, multi-threaded applications can get the most benefit because the proper use of heap pools virtually eliminates contention for heap storage.
   The Q Capture program has multiple threads working together to capture DB2 transactions. The two main threads are a LogReader thread that reads the DB2 logs and copies the records into memory; a Publisher thread that decodes the log records, rebuilds transactions, and constructs MQ messages. This structure of multi-threading makes Q Capture a good candidate to benefit from heap pools.

2. Applications that issue many storage requests with a malloc() of 65536 bytes or less because the heap pools algorithm is not used in a malloc() greater than 65536 bytes.
   Q Capture deals mainly with the DB2 log records, many of them at the same time, reading them from the DB2 logs and putting them together into transactions, etc. Since most DB2 log records tend to be quite small, the storage used by Q Capture to hold the DB2 log records and handle them are always small, too. This characteristic of Q Capture again makes it a good candidate to use heap pools.

3. Applications that are not storage-constrained; the heap pools algorithm gives up storage for speed. Without tuning, the heap pools algorithm uses much more storage than the normal malloc()/free() algorithm; when properly tuned, it uses only slightly more. Therefore, storage constrained applications should try heap pools, but only if the cell sizes and cell pool percentages are carefully tuned.
   The Q Capture program that we run in our lab environment, which has a high capture rate, has a defined 2GB of region size. Taking common storage into consideration, the available region private storage is around 1.6GB. The total storage used is far smaller than this. Because replication is moving changes as fast as possible, only enough memory needs to be used at any given time to keep the replication process moving along. The total memory required is generally quite modest, well below 1GB, because DB2 applications following best practices for performance commit changes frequently e.g. every few hundred or thousand rows during batch processes. OLTP transactions generally commit a few rows at a time. If Q Capture is constantly processing very big transactions or handling LOB objects, the storage consumption should not be too high to cause a virtual storage concern. And as stated above, if tuned properly only slightly more storage should be required by using heap pools.
   Q Replication programs use multiple threads to do work and both the capture and apply programs make lots of malloc()/free() to acquire heap storage. So Q Replication applications should be gaining benefits with the introduction of tuned HEAPPOOLS option.

Steps of HEAPPOOLS option tuning

1. Generate an options report and a storage report for Q Capture main program by specifying LE runtime options RPTOPTS(ON), RPTSTG(ON) and HEAPPOOLS(ON) with a representative workload on a test system. HEAPPOOLS(ON) option here is equal to HEAPPOOLS(On,8,10,32,10,128,10,256,10,1024,10,2048, 10,0,10,0,10,0,10,0,10,0,10,0,10,0,10,0,10,0,10).
   This makes LE to start using heap pool algorithm to manage heap storage. With the
HEAPPOOLS statistics and summary information that produced in the storage report, we now have a starting point of tuning heap pools.

2. Read and analyze the HEAPPOOLS statistics and HEAPPOOLS summary sections in the LE runtime storage reports. Analyze and adjust HEAPPOOLS options according to the following rules:

1. Choose good cell sizes based on the distribution of the `get` requests of different cell sizes and the LE-suggested HEAPPOOLS option. If a heap storage request doesn't have an exact cell size to match it, the heap storage will be allocated from the first cell pool to fit the request with cell pools sorted by sizes in ascending order.

2. Keep extents allocated for each cell size in the HEAPPOOLS summary section close to 1 by either adjusting percentage numbers of the cell sizes or adjusting heap initial sizes.

3. Keep cells per extent greater than maximum cells used by a reasonable margin in the HEAPPOOLS summary section. Keeping in mind that:

\[
\text{Cells Per Extent} = \text{Max}(4, \frac{(\text{Initial HEAP Size} \times (\text{Extent Percent}/100))}{\text{Element Size}})
\]

If current `Cells Per Extent` is smaller than `Maximum Cells Used` and heap usage report indicates that it has many segments allocated, consider raise heap initial size first. If heap storage is large enough without multiple segments allocated, consider raising the `Extent Percent` to a bigger number. Vice versa, if current `Cells Per Extent` is over 50% greater than `Maximum Cells Used`, either reduce the heap initial size or change `Extent Percent` to a smaller number, or do both to waste less storage.

4. If Maximum Cells Used in the HEAPPOOLS summary section appears to be high for a certain cell size, consider using multiple cell pools for that cell size to spread the heap storage requests to reduce potential contentions.

5. For multi-thread application, additional benefit will be gained by using ALIGN mode instead of ON mode HEAPPOOLS option. In addition to avoiding contention during storage allocation and release, the goal of ALIGN mode is to reduce cache contention when two adjacent cells are being updated at same time. Although, ALIGN mode uses more storage compared to ON mode.

3. Repeatedly adjust the HEAPPOOLS option, heap initial size, and increment sizes for a few times with Q Capture running under a representative workload. A couple of tips here if you are to try HEAPPOOLS out for your Q Capture program: measure your performance improvement in a test environment and decide if you want to implement the new option in production or not. If yes, keep in mind to remove RPTOPTS(ON) and RPTSTG(ON) options in your production when activating heap pools.

### Q Replication LE heap tuning result

#### Environment system hardware configuration

The test environment hardware configurations are shown in Table 1.

#### Table 1. Hardware configuration

<table>
<thead>
<tr>
<th>System name</th>
<th>Function</th>
<th>CP#</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM zEnterprise EC12 2827</td>
<td>System_1</td>
<td>20</td>
</tr>
</tbody>
</table>
Software build level and configurations

The software code level and configuration details are shown in Table 2.

Table 2. Software code level and configuration details

<table>
<thead>
<tr>
<th>Software</th>
<th>Code level</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2</td>
<td>V10</td>
<td>DBCL as non-datasharing source database at System_1, which hosts the workload and generate log record. DZM1 as non-datasharing target database at System_2</td>
</tr>
<tr>
<td>MQ</td>
<td>V7.1</td>
<td>QCL1 service Q capture at System_1 to transmit messages QCL2 service Q apply at System_2 to receive messages</td>
</tr>
<tr>
<td>Queue Replication</td>
<td>V10.0.1</td>
<td>Q capture read DBCL log and publish to transmit queue Q apply reply the transactions BATCH_SIZE 4 MEMORY_LIMIT 500M</td>
</tr>
</tbody>
</table>

Workload characteristics

The IBM-created OLTP RTW workload was used to drive the replication. Tests with different application transaction sizes were done to simulate different workload environments.

RTW is a standard workload used by the CICS Hursley Performance team to assess changes in performance characteristics within new releases of CICS® code when running DB2 applications. In these applications, the presentation logic is separated from the business logic by EXEC CICS LINK calls. The front-end presentation logic is simple. It receives data from the terminal, passes it to the back-end business logic, and sends a response to the terminal when it returns control.

The workload has the following characteristics:

- All COBOL programs
- Seven unique CICS transactions consisting of
  - 54% Select, 1% insert, 1% update, 1% delete, 8% open cursor, 27% fetch cursor, 8% close cursor
  - 20 database tables with
  - an average table row size of 125 bytes
  - column types: CHAR, INTEGER, DECIMAL, SMALLINT

Execution scenario

Phase 1: Preparation

Increase throughput with z/OS Language Environment heap storage tuning method
1. Modify the workload to adjust the transaction size for the test.
2. Clean out Q Replication and MQ monitor tables.
3. Either enable or disable the heap pool tuning.

Phase 2: Start replication and data measurement

1. Start Q Replication on both sides.
2. Run workload.
3. After 20 minutes of warm-up, take 10 minutes of measurement data, including QREP control tables and MQ monitor data. Note the publication rate.

Procedure of LE heap tuning for Q Replication

1. Add the following JCL statements to the Q capture startup JCL.

```
//CEEopts DD *
RPTopts(ON)
RPTSTG(ON)
```

If using end-to-end replication to tune the LE options, restart both Q Capture and Q Apply, and run the representative workload for a while with reasonable intensity. After stopping the workload, stop Q Capture and Q Apply (optional). For RTW workload, we used end-to-end replication for tuning. The LE reports will be printed with the termination of Q Capture program, and the reports are directed to SYSOUT DD by default.

1. Initiate tuning analysis based on LE reports produced above.

Fill Table 3 with data shown in the LE options and storage usage reports. Table 3 has been filled with sample data for reference purposes.

1. Tune the HEAPPOOLS runtime option

The Q Capture program is not using HEAPPOOLS by default. To enable HEAPPOOLS and tune the option properly, turn on HEAPPOOLS option by specifying HEAPPOOLS(ON) in CEEOPTS DD input. With RPTSTG(ON), LE will collect HEAPPOOLS statistics. Follow the HEAPPOOLS tuning rules put forward in last part of this tutorial to guide the tuning process.

1. Repeat the preceding steps a few times to tune and verify the LE runtime options.
2. Implement the tuned LE options

After gaining satisfaction with the new tuned LE runtime options in the test environment, change RPTOPTS(ON) and RPTSTG(ON) both to (OFF). Then restart Q Replication address spaces and let them run with the tuned LE options in test environment. The final tuned LE heap storage-related runtime option overrides for the RTW workload are:

```
HEAP(2097152,2097152,ANYWHERE,KEEP,8192,4096)
```
Table 3. Information retrieved from LE options and storage usage

Result highlights

1. Figure 2 shows about 13-percent improvement of capture rate was observed with transaction size of 4808 bytes.

Figure 2. Capture rate improvement

1. Other than the throughput improvements, we also noticed a noticeable drop of TCB CPU busy percentage for the QREP Publisher thread and LogReader thread.

2. To further test Q Capture rate improvements introduced by the tuned HEAPPOOLS option for different transaction sizes, we made more runs with different RTW transaction sizes by using the "preload DB2 log" test method with "fake apply." For details on "preload DB2 log" method, "fake apply," and how we controlled transaction sizes, refer to Appendix A. About a 7- to 13-percent improvement of capture rate was observed with transaction sizes less than 16,200 bytes. As the transaction size increased, the amount of improvement in the capture rate decreased. Only about a 1- to 2-percent improvement of the capture rate was seen with transaction sizes larger than 16,200 bytes. From our customer experience, OLTP transaction sizes are usually in the 1,000- to 4,000-byte range. The max transaction sizes shown in the chart below were captured by Q replication monitor. In our workload, various transaction sizes are distributed in a certain range. The max transaction size is the peak value. When changing the workload logic to change transaction, all kinds of transaction sizes are increased. In other words, the transaction size range is increased. In the figure below, the max transaction size is the representation of the range.
Conclusion

By following the LE-documented LE heap tuning procedures in the z/OS V1R13.0 Language Environment Programming Guide, replicating an OLTP workload with Q Replication gained a significant performance improvement. For transaction sizes <= 16,200 bytes, the capture rate improvement was much more significant than for transaction sizes larger than 16,200 bytes. Monitored CPU time spent in LE modules that handle heap storage showed decreases for all transaction sizes. This means with the tuned heap-related LE runtime options, the heap storage processing became more efficient and this is probably the primary contributor to the Q Replication's throughput improvement. Our LE tuning yielded less capture rate improvement for larger transactions because the benefits introduced by using heap pools may have been outweighed by the time Q Capture spent in handling/composing the bigger transactions.

Appendix A

• "Preload DB2 log" test method run steps

  1. Fill DB2 log by running RTW workload for a period of time.
  2. Start Q Replication programs with Q Capture program having its LSN (Log Sequence Number) set to the starting point of the previously filled DB2 log.

This method was always used together with "fake apply" when testing Q Capture performance with the exact same DB2 log and running Q Capture multiple times with different settings.

• "Fake apply"

"Fake apply" means the Q Apply program is not started on the target system. Instead, a simple high-speed MQGET-only batch program is reading and discarding the MQ messages sent by Q Capture on the target system.

• Adjusted RTW programs producing different-sized transactions

Adjusted versions of the original RTW programs were created by changing the frequencies at which SYNCPOINTs were taken to produce varying-sized transactions. The general sequences of the DB2 requests of these changed RTW programs were kept close to the original RTW programs.
Resources

Learn

- Read Language Environment Programming Guide (z/OS V1R13), SA22-7561-08 (GA).
- Read z/OS Language Environment Debugging Guide (z/OS V1R13), GA22-7560-08 (GA).
- See "Understanding and Using Q Replication for High Availability Solutions on the IBM z/OS Platform" (IBM Redbooks SG24-8154-00).
- z/OS Language Environment Customization (z/OS V1R13), SA22-7564-13 (GA).
- Visit the developerWorks Information Management zone to find more resources for DB2 developers and administrators.
- Stay current with developerWorks technical events and webcasts focused on a variety of IBM products and IT industry topics.
- Follow developerWorks on Twitter.
- Watch developerWorks on-demand demos ranging from product installation and setup demos for beginners, to advanced functionality for experienced developers.

Get products and technologies

- Evaluate IBM products in the way that suits you best: Download a product trial, try a product online, or use a product in a cloud environment.

Discuss

- Get involved in the developerWorks community. Connect with other developerWorks users while exploring the developer-driven blogs, forums, groups, and wikis.
About the authors

Terry Borden

Terry Borden has been working with IBM for 44 years, most of which in System z and software architecture and design. Borden has spent the past four years in the IBM Hardware Acceleration Lab working on z/OS solution scaling performance.

Greg Dyck

Greg Dyck has been working with IBM for 32 years. The first 20 years he was in Poughkeepsie, working in processor support and z/OS development. He then spent nine years working in DB2 development and the past three years working in the IBM Hardware Acceleration Lab working on z/OS solution performance.

Nicholas Matsakis

Nicholas Matsakis has been working with IBM for 30 years, of which 25 were in the z/OS development area and three years in the IBM Hardware Acceleration Lab working on z/OS solution performance.

Zhuang Lei

Zhuang Lei has been an IBM intern for two years.

Wu Hao

Wu Hao joined IBM in 2005 and has worked in DB2 for z/OS testing team for seven years. Hao joined HAL in 2012 working on z/OS solution performance and managing the HAL China team.
Chunxin Yang

Chunxin Yang has been with IBM for seven years, of which five were in the DB2 for z/OS QA and development area, and two in the Hardware Acceleration Lab working on z/OS solution performance.

Xuelian Zhang

Xuelian Zhang has seven years of experience with System z technologies. Zhang started as a CICS system programmer for State Street Corp, joined IBM in 2012, and is currently working in the STG Lab Services team.

© Copyright IBM Corporation 2014
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)