# 如何在 VSE/ESA 系统平台获取更高的批量程序效率

## How To Enhance Batch – Programs' Performance In VSE/ESA System

马宇航　（清华大学软件学院 100084）

**摘要**：本文介绍一种在金融系统批量状态下，通过利用 LE 的功能，使批量程序突破 COBOL 语言本身的限制，充分利用系统内存资源而提高处理效率的方法。这种方法对于批量状态下对于数据量适中但需频繁访问的数据集的操作提供了一种新的思路。

**关键词**：VSE/ESA LE LINKAGE SECTION 数组 内存

## 1 目前 VSE/ESA 的系统状况及限制

在目前中国银行各分行的主机上存在着两套 VSE/ESA 版本，其中个别分行使用的 VSE/ ESA V1.2 版本为 24 位（24 – bit）寻址方式，其最大寻址空间只能达到 16MB，所以其分配的每个分区空间最大为 16MB。保留 SUPVIOR 所必须占用的 5MB，分区上程序实际可用空间最大只能取 11MB，没有太多的性能提升空间。

而大部分 VSE 主机行使用的 VSE/ ESA V2.3 版本为 31 位（31 – bit）寻址方式，其最大寻址空间可达 2GB，因而其运行的分区空间不受 16MB 的限制。在分行应用配置中一般批量运行的分区分配的空间都较大（有些使用固定的空间较大的静态分区，有的直接放到动态分区上），具有较大的性能空间，本文也针对这种系统潜力进行分析和利用。

在批量作业运行中，可以在作业中指定程序运行分区，并在 EXEC 语句中指定程序所使用内存的大小，其定义语句为：

EXEC 程序名，SIZE = nnn K 或

EXEC 程序名，SIZE = AUTO

该 SIZE 含义是指程序运行空间，系统以此处定义为最终值。其中 SIZE = AUTO 是可以使用分区上的最大空间。在作业中指定好 EXEC 的执行 SIZE 后，GETVIS 域的内存空间 = 分区可用空间 – EXEC SIZE。这部分空间也是我们要加以利用的空间。

我们知道 COBOL 语言在进行编译时仅将定义的变量说明和程序语句等生成到执行码 PHASE 中，而对于数组的使用则是在程序运行时从程序运行的分区上获取一块内存用来完成数组的存放。所以在进行批量处理的优化上我们首先想到了使用数组，并取得了不错的效果，但是在测试实践的过程中我们也发现了 COBOL 语言的弱点。

## 2 COBOL 语言编程的限制

在目前中行主机系统的程序设计中一般使用 COBOL/VSE 语言，在针对大型金融产品的开发中逐渐引入了面向对象的设计思想，主要体现在设计时封装功能模块、建立公共的变量定义、常量定义，在程序开发时统一进行调用（COPY 或 INCLUDE），如数据存储对应的结构描述，通用的数据值定义等。但 COBOL 语言本身却存在着一定的限制：COBOL 程序中变量定义统一定义在 WORKING STORAGE 中，一般 01 层变量定义和 77 层变量定义空间仅为 16M，所以在程序开发上一方面要保证标准化的引用，使程序设计规范，一方面还要有充足的空间开辟数组来提高程序效率，这本身就是一对矛盾。在测试中我们通过实验发现在程序中开辟的数组空间一般最多达到 4M 左右。这么小的空间对于许多中型的表均显现出不足，象一般分段计息帐户的利率历史都不能完整的 LOAD 到数组中。

这种现状意味着虽然各分行使用是 VSE/ ESA V2.3 环境，能够支持大量的分区内存空间（最大寻址空间可达 2GB），但却因为 COBOL 语言本身的限制而不能得到很好的利用。怎样才能更有效的利用这一部分资源呢？下面介绍解决该矛盾的手段。

## 3　动态使用分区内存方法介绍

在程序中嵌套使用 LE 的功能调用，并利用 COBOL 语言中 LINKAGE SECTION 段在编译中没有空间的限制，来完成在程序运行中动态获取和使用内存空间，获得较高的运行效率。具体步骤如下：

### 3.1　在程序 WORKINGSTORAGE 中定义如下信息

```
01 WK – HEAPID PIC S9(09) BINARY.（堆标识，自定义变量，获得内存时使用）
01 WK – STORAGE – SIZE PIC S9(09) BINARY.（内存空间大小，自定义变量，获得内存时使用）
01 WK – ADDRESS USAGE IS POINTER.（内存地址，自定义变量获得内存时返回值）
01 WK – INDEX1 PIC S9(09) BINARY.（应用使用数组下标）
01 WK – FC.（使用 LE 调用的返回）
  02 Condition – Token – Value.
  COPY CEEIGZCT.（LE 环境提供包含 LE 访问返回值）
    03 Case – 1 – Condition – ID.
      04 Severity PIC S9(4) BINARY.
      04 Msg – No PIC S9(4) BINARY.
    03 Case – 2 – Condition – ID
        REDEFINES Case – 1 – Condition – ID.
      04 Class – Code PIC S9(4) BINARY.
      04 Cause – Code PIC S9(4) BINARY.
    03 Case – Sev – Ctl PIC X.
    03 Facility – ID PIC XXX.
  02 I – S – Info PIC S9(9) BINARY.
```

### 3.2　在程序中定义 LINKAGE 信息

```
LINKAGE SECTION.
01 LK – RECORDS.（应用使用空间定义，大小要与 GET 的内存匹配）
    05 LK – RECORD OCCURS 1 TO 100000 TIMES DEPENDING ON WK – RECNUM
        ASCENDING KEY IS LK – KEY INDEXED BY LK – INDEX1.
        10 LK – KEY PIC X(13).
        10 FILLER PIC X(220).
```

### 3.3　在程序中获取内存

在程序中每次进行内存获取前均可以根据数据量和每条数据的 LENGTH 来计算实际要使用的内存大小，并以该大小进行内存的申请：

```
（堆标记赋 0 即可，如果要同时使用多个堆需使用 CEECRHP 的函数调用）
MOVE 0 TO WK – HEAPID（使用单个堆）
COMPUTE WK – STORAGE – SIZE = （可事先计算使用内存大小）
    WK – RECNUM * LENGTH OF BI – LIM – LIMREC
END – COMPUTE
（如果使用多个堆的话可以使用如下语句获取另外的堆 – ID：WK – HEAPID 为返回的堆 ID，WK – INIT – STORAGE – SIZE 为分配空间大小应为 4096 的整数倍，0 表示当内存空间不足时每次的增量也为 4096 的整数倍，79 或 80 表示在批量模式下）
CALL ´CEECRHP´ USING WK – HEAPID, WK – INIT – STORAGE – SIZE,
                0, 79.
DISPLAY ´Storage size is: ´ WK – STORAGE – SIZE
（调用 LE 功能 CEEGTST 按照 WK – STORAGE – SIZE 申请内存，调用结束将返回地址到 WK – ADDRESS，返回码返回到 WK – FC）
CALL ´CEEGTST´ USING WK – HEAPID, WK – STORAGE – SIZE,
                WK – ADDRESS, WK – FC.
IF NOT CEE000 of WK – FC（判断返回码的正确与否）
    DISPLAY ´CEEGTST failed with msg ´ Msg – No of WK – FC
    MOVE 8 TO RETURN – CODE
     STOP RUN
END – IF
DISPLAY ´CEEGTST Ok! ´
SET ADDRESS OF LK – RECORDS TO WK – ADDRESS
（如正确获得内存则将地址赋给 LINK 区）
```

需要注意的是如申请失败，则可能程序运行的分区分配的空间太小，换一个大些的分区即可。

### 3.4　将应用数据写入内存

应用数据写入内存一般是将数据库数据或 VSAM 数据写入内存，实际就是将应用数据写入 LINKAGE

SECTION 区中自行定义的 LK – RECORDS。

我们使用了循环写入的方式

```
MOVE 1 TO WK – INDEX1
PERFORM UNTIL SQLCODE NOT = 0
  EXEC SQL FETCH C – LIM INTO
       :BI – LIM – DEALID, :BI – LIM – SOURCE, :
        BI – LIM – SYSNUM,
       :BI – LIM – OPTSEQ, :BI – LIM – REPCYC, :
        BI – LIM – SEQIDE,
       :BI – LIM – STARNU, :BI – LIM – LENNUM,
        :BI – LIM – OPTIDE,
       :BI – LIM – LIMIT1, :BI – LIM – LIMIT2, :BI –
        LIM – LIMIT3,
       :BI – LIM – LIMIT4, :BI – LIM – LIMIT5, :BI
        – LIM – LIMIT6,
       :BI – LIM – LIMIT7, :BI – LIM – LIMIT8, :BI –
        LIM – LIMIT9,
       :BI – LIM – LIMIT0
  END – EXEC
  IF SQLCODE NOT = 0 AND SQLCODE NOT = 100
    DISPLAY 'Fetch C – LIM error:' SQLCODE
    MOVE 8 TO RETURN – CODE
    STOP RUN
  END – IF
  IF SQLCODE = 0
  MOVE BI – LIM – LIMREC TO LK – RECORD(WK –
INDEX1)
```

(注意不要超过自己定义的数组大小)

```
  ADD 1 TO WK – INDEX1
END – IF
END – PERFORM。
```

### 3.5 应用数据的使用

数据获取完毕后,其使用与一般数组使用一致,可以用 SEARCH 等方式进行数据的查找,使用记数器下标进行循环等。

### 3.6 内存的释放

注意如程序是在循环中使用内存,需在使用完内存后对内存进行释放,否则再进行获取时必须重新获取堆的标记。释放内存语句如下:

(使用 CEEFRST 函数对内存进行释放,其中 WK – ADDRESS 为 CEEGTST 函数分配的内存地址,WK – FC 为返回码)

```
CALL 'CEEFRST' USING WK – ADDRESS, WK – FC
```

(注意如返回码如不正确,则再次使用内存时需重新使用 CEECRHP 获取新的堆 ID)

## 4 使用该方式的效率比对

我们进行了效率方面的测试。

测试操作系统环境为 VSE/ ESA V2.3,数据库使用 DB2 5.1。

测试数据为一个仅有 8000 条记录,每条记录 500 字节的小表。

在批量程序中按照一般应用程序方式进行按键值 SELECT,共进行 215 万次访问,程序运行消耗 36'15"。

采用在程序 WORKING STORAGE 中直接建立数组方式,程序不能正常编译。

采用本文介绍的获取内存预先存储方式后,程序能够正常编译,生成 PHASE,执行时程序先建立该表的 CURSOR,再将数据 FETCH 到申请的数组空间中,耗费 2'28",再对数组进行 215 万次的读取,消耗 1'24",共耗时 3'52",效率提升相当明显。

## 5 结束语

本文介绍了一种在主机批量模式下突破 COBOL 语言的限制,充分利用系统内存而达到提高应用处理效率的方法。这种方法只是从程序优化的角度提出的一点批量优化的建议,对于未来的数据集中模式下的系统程序设计开发也有一定的指导意义。

## 参考文献


1 IBM COBOL for MVS & VM Version 1 Release 2 Performance Tuning – – R. J. Arellanes IBM Corporation Software Solutions January 26, 1996.

2 24 Hour Continuous Process, IBM,2003.11

3 CICS(ESA) VSAM FILE PERFORMANCE – – Bob Archambeault R. A. Solutions International Inc., 1998.

4 IBM Storage Networking Solutions, IBM,2004.01.

5 ICBC Technical Account Plan, 工商银行电脑部, 2002.09。