DFSORT/MVS

Recommendations for Tuning Large DFSORT Tasks

by Trevor Kingsbury
email: Trevor_Kingsbury@compuserve.com

(The author would like to thank Carrie Van Noorden, Martin Packer and Frank Yaeger for their assistance.)

Most installations have experienced explosive growth in recent years in the volume of data to be stored and processed; both in operational applications and in decision-support and data warehousing.

While RDBMS on either MVS or Unix remains the permanent data storage platform of choice, much of this data must be extracted, processed, transformed and loaded using traditional MVS batch processing. Sorting and merging of data, using DFSORT under MVS, endures as a key component of this batch processing.

In addition, utility processing in DB2/OS390 has always made extensive use of DFSORT. The introduction of LARGE tablespaces in version 5 has produced a new class of potential DFSORT challenges in this area.

This paper attempts to set out a series of recommendations for dealing specifically with **high data volume** DFSORT tasks, where high data volume could be defined (in 1999) as any of the following:

- Input file size in excess of 10Gb
- Input file record count in excess of 20 million records
- Sort key volume (that is, record count multiplied by sort key length) in excess of 2Gb

In my experience, these are the approximate data volumes at which a configuration of DFSORT that acknowledges the workload involved becomes absolutely essential. Well below these levels, a default configuration of DFSORT will be "feeling the pinch".

Generally, sorts of this nature will occur only in the more substantial installations. However, the days when a terabyte of data was considered an inconceivably large amount of storage are well and truly behind us!

The recommendations and techniques described below may be applicable to any substantial DFSORT task, perhaps well below the levels set out above. DFSORT

is often invoked hundreds or thousands of times per day in a typical MVS installation - it is perhaps the most commonly invoked batch program. Any across-the-board streamlining of DFSORT processing will, in all likelihood, lead to noticeable flow-on improvements with the remaining workload.

These recommendations stem in part from experiences with a series of very large operational and decision-support system implementations, in which tuning and re-configuration of DFSORT proved to be the single most beneficial technique for reducing overall batch processing elapsed time.

The performance comparisons shown in this paper were derived from measurements taken in a controlled stand-alone environment at a customer site using DFSORT R13. Keep in mind that performance improvements will vary depending on factors such as key size, record type, number of records, processor model, region size, and so on.

Topic List

Tuning Large DFSORT Tasks

Objectives

Generally, there will be two primary objectives with tuning of **large** DFSORT tasks:

- **Elapsed time reduction** - in general irrespective of the cost to other system resources (CPU, central and expanded storage, I/O subsystem utilization).
- **Elapsed time scalability** - that is, the ability to increase data volumes with the sole impact a near-linear increase in elapsed time. Alternately this can be defined as the ability to break down the processing into a number of smaller units, to be executed concurrently in order to maintain overall elapsed time.

Some considerations that have been the focus of DFSORT tuning in the past may be set aside in a contemporary MVS configuration. Central and expanded storage have become far more plentiful; system paging is now rarely a first-level concern. In addition, the CP usage of standard DFSORT tasks (excluding exits) is minimal, and may be effectively disregarded as a constraining factor.

Tuning Large DFSORT Tasks

Application Recommendations

Examine Application Requirements

Perhaps the most easily overlooked area is the requirements of the application itself. In particular, the following are worth investigating.

Are all fields in the sort input file actually required for the sort application or by downstream processing?

If not, performance may be improved if you can significantly reduce the length of your records with INREC. Eliminating fields with INREC reduces the amount of data that has to be sorted because INREC reformatting takes place **before** sorting. (OUTREC and OUTFIL reformatting take place **after** sorting.)

For example, a long trailing FILLER (spaces) field could be eliminated via INREC. If the FILLER field is needed for later processing, it can be reinstated via OUTREC. This change requires no application cost except a couple of additional control statements.

Is there a possibility of large data volume reduction by modifying the record format from fixed to variable?

For example, variable-length text data may be "padded" to the maximum length of the field, sorted, and then used as input to DB2 LOADs or to print utilities. The preference would be to delay this padding activity until after the sort, that is, to sort the **non-key** data in the shorter variable-length format, if this will substantially reduce the volume of data to be sorted.

Can the facilities of DFSORT be employed to both reduce data volumes and eliminate part of the application processing?

For example, data summation, reformatting, numeric and date editing can all be accomplished with standard DFSORT control statements. This is reminiscent of the SQL tuning technique of relocating as much application logic into SQL as possible.

The ICETOOL utility - a standard inclusion with the DFSORT product - provides yet more functionality. For details on ICETOOL, browse the ICETOOL mini-user guide.

Can more be done in a single pass?

For example, a sort prior to a DB2 LOAD utility may use DFSORT's OUTFIL features to split the file by the partitioning key on output, and so avoid a subsequent "split" pass of the data.

Is sorting actually necessary?

Application developers commonly confuse sort and merge. A merge of files already sorted will exhibit similar or superior performance to a DFSORT **copy** application.

Consider the following application tuning example.

```
OPTION MAINSIZE=12M,DYNALLOC=(SYSDA,12),
  SIZE=E300000,AVGRLEN=1000
INCLUDE COND=(544,1,CH,EQ,C'A')
SORT FIELDS=(96,6,PD,A,29,10,CH,A)
SUM FIELDS=(284,8,PD)
OUTFIL FNAMES=SORTOUT,CONVERT,
  OUTREC=(90,6,96,6,284,8,29,10,
       X'000000001C',102,6)
```

Figure 1 - DFSORT control statements before tuning

The first steps in tuning this application were to change the OPTION statement parameters to increase the available central storage, increase the maximum number of SORTWK files, and accurately specify the file size and average record length.

```
OPTION MAINSIZE=64M,DYNALLOC=(SYSDA,20),
  FILSZ=U25000000,AVGRLEN=621
INCLUDE COND=(544,1,CH,EQ,C'A')
SORT FIELDS=(96,6,PD,A,29,10,CH,A)
SUM FIELDS=(284,8,PD)
OUTFIL FNAMES=SORTOUT,CONVERT,
  OUTREC=(90,6,96,6,284,8,29,10,
       X'000000001C',102,6)
```

Figure 2 - DFSORT control statements after OPTION changes

Finally, by moving some of the record reformatting from the OUTFIL statement to an INREC statement, the sorted record size was reduced from the original 621 bytes to 40 bytes, resulting in a significant decrease in the amount of data sorted. We also changed the AVGRLEN value to 40 to correspond to the new sorted record size.

```
OPTION MAINSIZE=64M,DYNALLOC=(SYSDA,20),
  FILSZ=U25000000,AVGRLEN=40
INCLUDE COND=(544,1,CH,EQ,C'A')
INREC FIELDS=(1,4,90,6,96,6,284,8,29,10,102,6)
SORT FIELDS=(11,6,PD,A,25,10,CH,A)
SUM FIELDS=(17,8,PD)
OUTFIL FNAMES=SORTOUT,CONVERT,
  OUTREC=(5,6,11,6,17,8,25,10,
       X'000000001C',35,6)
```

Figure 3 - DFSORT control statements after INREC change

The performance gains from making appropriate changes to the OPTION parameters and using INREC, as shown above, were as follows.

Table 1 - DFSORT tuning example - performance results

Sort application SORTIN file size 15.1Gb	Elapsed time (sec)	CPU time (sec)	Record count (millions)	Processing rate (records sorted per second)	Percentage of base value
Before tuning	6916	169.0	29.519	4,368	100%
After OPTION changes	4740	153.5	29.519	6,228	143%
After INREC change	1680	82.0	29.519	17,571	402%

Modify Sort Granularity

Some sort tasks may always remain impossible to accomplish in a single pass. The most ambitious sort task I have encountered to date is:

- Close to half a terabyte of input data, configured as five concurrent sorts of approximately 80Gb each
- A key length of over 700 bytes, average record length of 1840 bytes, and a record count of approximately 43 million for each of the five sort tasks
- A theoretical peak SORTWK allocation requirement of nearly 700Gb

I am grateful to Frank Yaeger of the DFSORT development team for supplying the basis for the following solution.

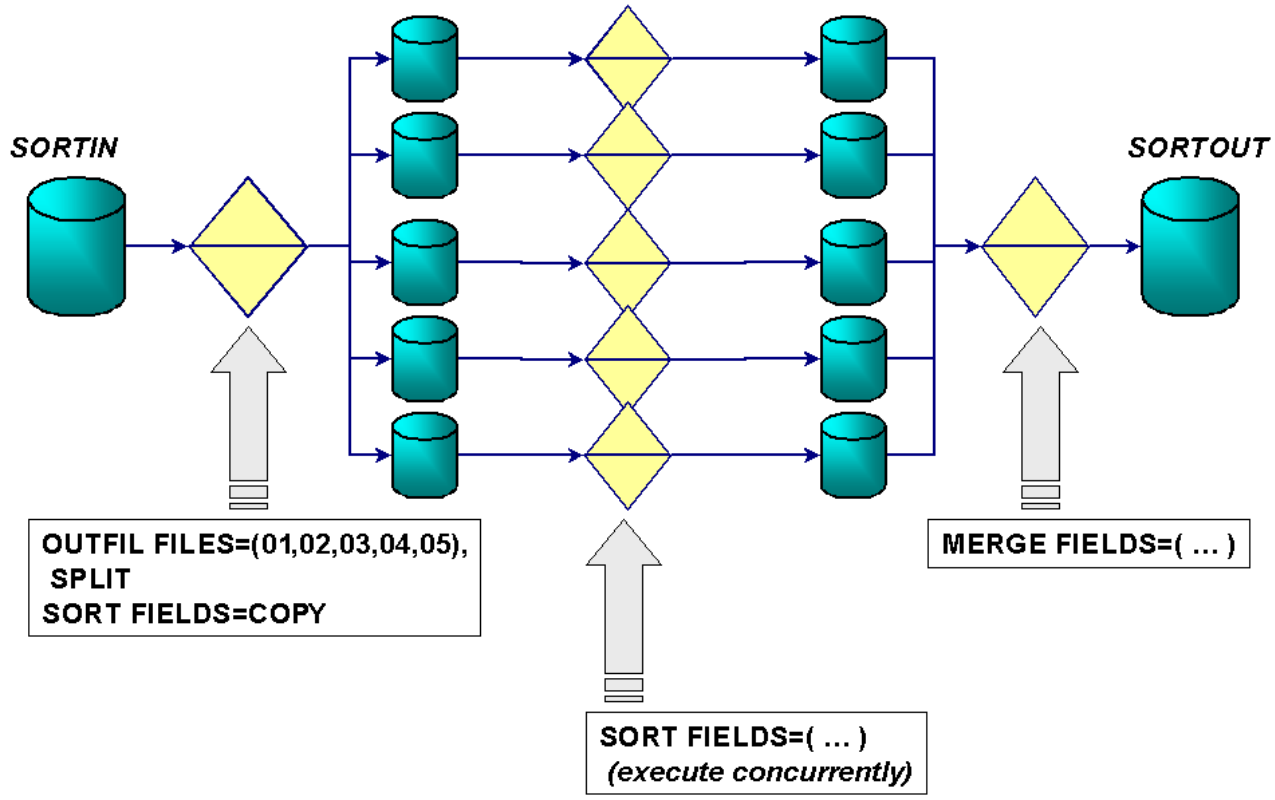

Figure 4 - Very large sort / split technique

In essence, this technique uses a simple "divide and conquer" approach.

- Records from the large SORTIN file are distributed across many sub-files in a round-robin fashion, using DFSORT's OUTFIL SPLIT feature.
- Sorts for each of these sub-files are executed concurrently.
- A final merge of the now-sorted sub-files produces the final SORTOUT file. With DFSORT R13, up to 16 files can be merged. With DFSORT R14, up to 100 files can be merged.

This technique wins on several fronts.

- Concurrent execution of the five 80Gb sorts was deemed not feasible, due to the large amount of SORTWK space required, and the consequent I/O demand. With the sort/split technique, overlap of true sort steps with copy/split and merge steps of other jobstreams is possible. Copy and merge do not use any SORTWK space.
- Despite the additional data handling required - each record is ultimately read and written an added four times - the sort/split technique proved some 27% faster in elapsed time than a standard sort.
- Smaller sorts will, as a general rule, have a faster overall processing **rate**, measured as Mb sorted per second, than larger sorts; due in part to a higher probability of required data being found in central storage, expanded storage or disk cache.
- The processing is far more scalable. Advantage can easily be taken of the expansion to 100 merge input files with DFSORT R14. Partial serialization of the sub-sorts can be contemplated to lessen the overall I/O subsystem workload. And, failure of any individual task is less of a

concern.

The following comparative elapsed times are offered as a guide to the scale of improvement possible.

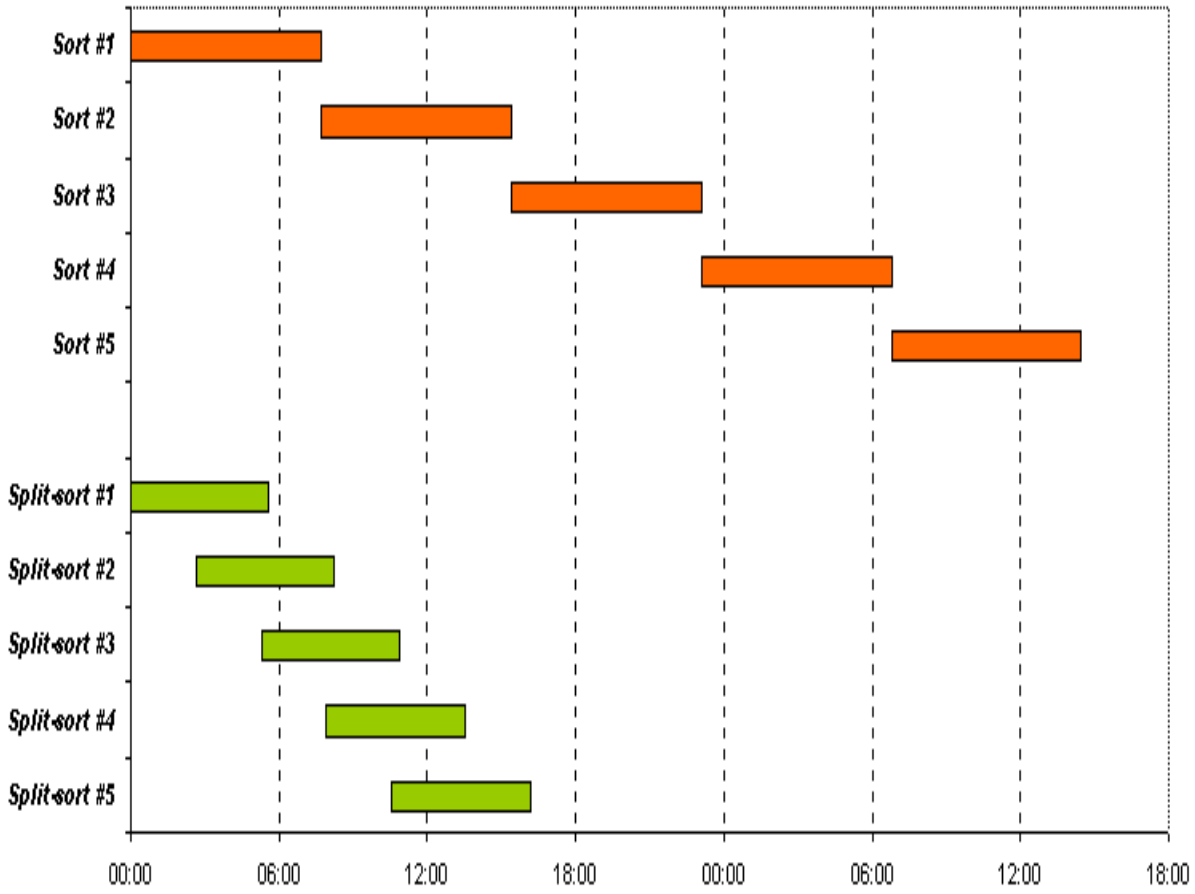

Figure 5 - Comparative elapsed times for sort/split technique vs standard sorts; 5 x 80Gb files

Sample execution JCL for this technique follows.

```
//** RUN THE JOBS IN THIS ORDER:
//** 1. CSPLIT1
//** 2. CSORT1-5 (CONCURRENTLY)
//** 3. CMERGE1
//*********************************************
//* CSPLIT1
//*********************************************
//CSPLIT1 JOB
//SPLITIT EXEC PGM=SORT
//SYSOUT DD SYSOUT=*
//SORTIN DD DSN=sortin file,DISP=SHR
```

```
//X1 DD DSN=INSORT1,...
//X2 DD DSN=INSORT2,...
//X3 DD DSN=INSORT3,...
//X4 DD DSN=INSORT4,...
//X5 DD DSN=INSORT5,...
//SYSIN DD *
 OPTION COPY
 OUTFIL FNAMES=(X1,X2,X3,X4,X5),SPLIT
//*********************************************
//* CSORT1-5
//*********************************************
//CSORT1 JOB
//SORT1 EXEC PGM=SORT
//SYSOUT DD SYSOUT=*
//SORTIN DD DSN=INSORT1,DISP=SHR
//SORTOUT DD DSN=INMRG1,...
//SYSIN DD *
 SORT FIELDS=(...)
//CSORT2 JOB
//SORT2 EXEC PGM=SORT
//SYSOUT DD SYSOUT=*
//SORTIN DD DSN=INSORT2,DISP=SHR
//SORTOUT DD DSN=INMRG2,...
//SYSIN DD *
 SORT FIELDS=(...)
//CSORT3 JOB
//SORT3 EXEC PGM=SORT
//SYSOUT DD SYSOUT=*
//SORTIN DD DSN=INSORT3,DISP=SHR
//SORTOUT DD DSN=INMRG3,...
//SYSIN DD *
 SORT FIELDS=(...)
//CSORT4 JOB
//SORT4 EXEC PGM=SORT
//SYSOUT DD SYSOUT=*
//SORTIN DD DSN=INSORT4,DISP=SHR
//SORTOUT DD DSN=INMRG4,...
//SYSIN DD *
 SORT FIELDS=(...)
//CSORT5 JOB
//SORT5 EXEC PGM=SORT
//SYSOUT DD SYSOUT=*
//SORTIN DD DSN=INSORT5,DISP=SHR
//SORTOUT DD DSN=INMRG5,...
//SYSIN DD *
 SORT FIELDS=(...)
//*********************************************
//* CMERGE1
```

```
//*********************************************
//CMERGE1 JOB
//MERGE EXEC PGM=SORT
//SYSOUT DD SYSOUT=*
//SORTIN01 DD DSN=INMRG1,DISP=SHR
//SORTIN02 DD DSN=INMRG2,DISP=SHR
//SORTIN03 DD DSN=INMRG3,DISP=SHR
//SORTIN04 DD DSN=INMRG4,DISP=SHR
//SORTIN05 DD DSN=INMRG5,DISP=SHR
//SORTOUT DD DSN=sortout file,...
//SYSIN DD *
 MERGE FIELDS=(...)
```

Figure 6 - Sample JCL for sort/split technique

Tuning Large DFSORT Tasks

DFSMS and Dataset Recommendations

SORTIN and SORTOUT Striping

All multi-Gb SORTIN and SORTOUT files should employ a DFSMS striped DATACLAS, both for ease of dataset allocation and to improve the performance of all traditional I/O bound batch processing, whether DFSORT or otherwise.

This may well entail re-allocating existing SORTIN and SORTOUT datasets from CART to disk, and performing an archive of datasets to CART at the completion of batch processing.

The competing technique of allocating datasets larger than a single 3390 volume using UNIT=(XXXXXX,nn), while similarly allocating a dataset in pieces across multiple volumes, serializes I/O at the volume level, and should be supplanted by striping.

The optimal number of stripes to be employed is a matter for experimentation, and depends largely on the bandwith available to the I/O subsystem. The following tables illustrate the scale of performance improvement possible from the introduction of striping alone - in this example, using 16 stripes.

Note that these improvements do not include additional gains in the preceding and following tasks - probably standard application batch programs - which write the SORTIN and read the SORTOUT datasets.

Table 2 - Comparative performance for sort with striping

Sort application file size 3.51Gb	Elapsed time (sec)	Processing rate (Mb/sec)	Percentage of base value
No striping	786	4.47	100.00%
Striped SORTOUT alone	516	6.81	152.33%
Striped SORTIN and SORTOUT	480	7.32	163.75%

Table 3 - Comparative performance for copy with striping

Copy application file size 3.51Gb	Elapsed time (sec)	Processing rate (Mb/sec)	Percentage of base value
No striping	456	7.70	100.00%
Striped SORTOUT alone	336	10.45	135.71%
Striped SORTIN and SORTOUT	168	20.91	271.43%

Note that striping cannot be used on SORTWK files.

Recommendations

- Activate DFSMS striping via the dataset profile masks for both SORTIN and SORTOUT datasets
- Relocate SORTIN and SORTOUT datasets from CART to striped DASD

Dataset Blocksize

It should go without saying that half-3390-track blocking (that multiple of record length closest to but not exceeding 27998) should be employed for SORTIN and SORTOUT datasets throughout.

Particular care is required when relocating these datasets from CART to disk - the optimal blocksize for the former, at 32Kb, is just larger than half-track which effectively doubles the number of tracks (and track-level EXCPs) required for the dataset.

Recommendations

- Review SORTIN and SORTOUT blocksizes, via either the DFSORT SYSOUT, or fields ICEIBLK and ICEOBLK of the SMF type 16 record
- Omit the BLKSIZE parameter altogether in dataset allocations, and permit DFSMS to select the optimal blocksize
- Ensure that the SDB DFSORT installation parameter is set to YES. This is the IBM-supplied installation default.

Storage Classes and Storage Groups

Most modern production I/O subsystems consist primarily (or entirely) of RAID devices emulating traditional 3390 disks. A typical configuration today might consist of a number of 600Gb devices, each with 4Gb to 8Gb of cache and duplexed 10Mb per second Escon channels per device.

The opportunity exists to exploit this configuration aggressively.

Isolating I/O of a certain caste to just one or two RAID devices limits the productivity of the I/O subsystem when I/O of primarily that type is in demand.

For example, if all DFSMS temporary pool volumes are allocated to a limited range of (logical) device addresses that map to only one or two physical devices, both cache and channels on those devices will at times be strained while other channels and devices lie relatively idle.

A preferable configuration is to spread this I/O across all the available devices, by dispersing logical volumes assigned to each DFSMS storage group across the physical I/O infrastructure.

Apart from dramatically increasing the maximum channel bandwidth and available disk cache, the probability of HDA contention on any one of the underlying disks at any instant in time will also decrease as a result.

This is applicable to storage groups both for permanent (SORTIN and SORTOUT) files, requiring substantial I/O bandwidth due to striping, and for temporary (SORTWK) files, where the I/O demands are for rapid read-write access to multiple large, non-striped files.

A word of warning. SORTWK EXCPs are very large - 0.5Mb to 1.0Mb - and may have a noticeable impact on measured disconnect times for other applications. Some isolation, particularly from data with highly critical response time requirements, may be warranted.

A natural corollary of this is that it is preferable to allocate a large number of smaller SORTWK files (as opposed to a small number of large files), in order to disperse SORTWK I/O as widely as possible. With DFSORT R13, the maximum number of work files is 100. With DFSORT R14, the maximum number of

work files is 255.

In some cases, all I/O for SORTWK could conceivably be satisfied entirely from the disk cache for even a very substantial sort task - but possibly at some cost to I/O subsystem service for other tasks!

However, some RAID devices permit configuration of "hot" volumes that are fully read-write cached. While often reserved for IMS WADS datasets or system paging devices, it may be worth considering these volumes as candidates for SORTWK file allocation.

One final technique is to slightly stagger the start times of substantial and concurrently-executing DFSORT tasks in order to mix the characteristics of I/O being performed at any given instant in time. For example,

- job #1 may be primarily performing parallel sequential write I/O to striped SORTOUT
- at the same time, job #2 is performing essentially random I/O to SORTWK and making heavy use of disk cache
- and job #3 is executing mostly parallel sequential read I/O to striped SORTIN.

An example of this is implied in the preceding discussion on the sort-split technique.

Recommendations

- Review DFSMS logical volume assignment and physical volume mapping algorithms for both temporary and large permanent (striping candidate) datasets; ensure that these are dispersed as widely as possible across the I/O subsystem.
- In particular, if a hierarchy of storage devices is present, ensure that SORTWK allocations are directed to those devices with the fastest available **random** I/O service times.
- Allocate a large number of SORTWK files to substantial DFSORT tasks via the DYNALLOC OPTION parameter. A minimum of 32 SORTWK files is recommended for very large sorts.

SmartBatch Pipes

SmartBatch pipes function by replacing the traditional batch processing practice of passing sequential file data as disk files between **serially executed** processes, with the passing of data through a virtual storage **pipeline** between **concurrently executing** processes.

However, latter processes must wait for the **commencement** of output by earlier processes. In the context of a large DFSORT process, any process downstream from the sort must await the completion of the sorting phase and the commencement of output to SORTOUT.

Similarly, any upstream process can eliminate only the I/O to SORTIN in the sort's input phase. Particularly in a heavily striped sequential file environment, this sequential I/O contributes only a small fraction of the overall elapsed time.

Consequently, SmartBatch pipes are less than ideally suited to the traditional batch sorting scenario, such as the following.

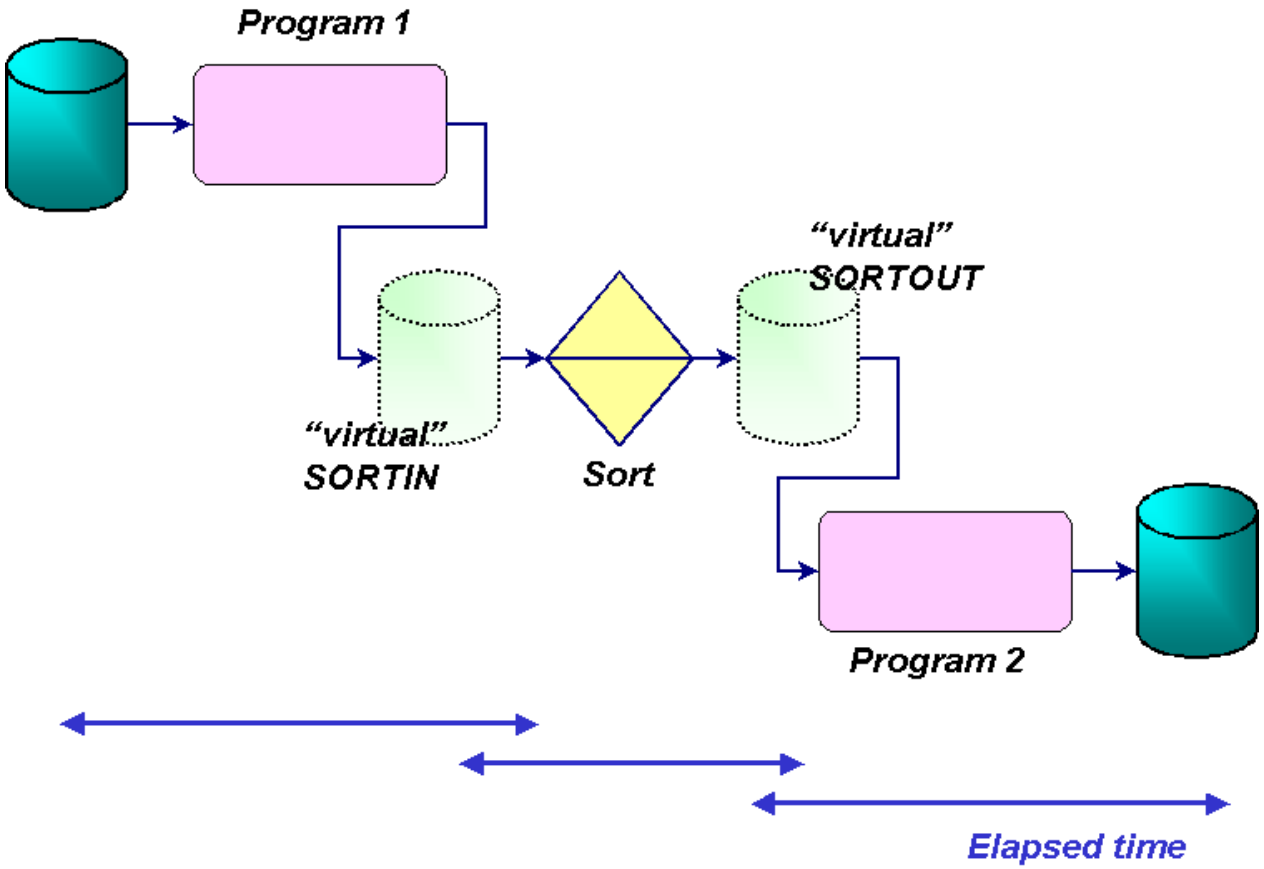

Figure 7 - Sorting with SmartBatch Pipes

Some additional points to note:

- Recoverability may be an issue, as failing processes employing SmartBatch pipes must be re-run, rather than re-started.
- Additional processes will be required to "harden" data - create a permanent disk copy - if retention of "intermediate file" data is required.

There are, however, some significant advantages for large sorts:

- Some overall elapsed time benefit, albeit **relatively** small, can be expected from the elimination of tens of gigabytes of SORTIN and SORTOUT I/O.
- The overall load on the I/O subsystem may be significantly reduced, enabling resources to be redeployed to service the main bottleneck in SORTWK file I/O.
- For DFSORT copy and merge applications, the vast majority of the total elapsed time is consumed by input and output file I/O. These will benefit very significantly.

Recommendations

- Evaluate and benchmark the use of SmartBatch pipes for the largest DFSORT applications, but with particular attention to the delta between files allocated with DFSMS striping and pipelined files

Tuning Large DFSORT Tasks

DFSORT Installation Recommendations

TMAXLIM, SIZE and DSA

These three parameters specify, respectively:

- an upper limit to the total amount of main storage available to a single DFSORT task when SIZE/MAINSIZE=MAX is in effect for a Blockset sort application
- the amount of main storage DFSORT attempts to use, defaulting to TMAXLIM if MAX is specified
- the maximum amount of storage available to DFSORT for dynamic storage adjustment of a Blockset sort application when SIZE/MAINSIZE=MAX is in effect. DFSORT will perform dynamic storage adjustment above TMAXLIM, up to the DSA limit, if it determines that additional main storage would benefit performance.

It makes little sense to attempt to micro-manage a few megabytes of main storage in these times of multi-Gb MVS processors. The following recommendations imply allocating a generous minimum of 16Mb to each DFSORT task, and allowing DFSORT to determine via DSA whether an expansion beyond that limit will benefit the task - and to allocate that optimal amount if it does.

Recommendations

- Set DSA=128Mb in ICEMAC. With DFSORT R13, the IBM-supplied installation default is 16Mb. With DFSORT R14, its 32Mb.
- Set TMAXLIM=16Mb in ICEMAC. The IBM-supplied installation default is 4Mb. If CPU time is a major concern and you have many small sorts, leave TMAXLIM at 4Mb or increase it to 8Mb.
- Set SIZE=MAX in ICEMAC . This is the IBM-supplied installation default.
- Remove MAINSIZE=n settings from DFSORT OPTION statements. Let DFSORT do the hard work.

EXPMAX, EXPOLD and EXPRES

These three parameters govern, respectively:

- the maximum total amount of available expanded storage to be used at any

one time by all Hipersorting applications

- the maximum total amount of "old" expanded storage (that is, low-usage frames) to be used at any one time by all Hipersorting applications
- the minimum amount of available expanded storage to be reserved for use by non-Hipersorting applications

Large DFSORT tasks will most commonly use SORTWK files exclusively for intermediate workfile data, and will rarely make use of either Hipersorting or dataspace sorting.

The benefit of Hipersorting lies in that it presents an opportunity to lessen the pressure from many small to medium sort tasks on the I/O subsystem, by substantially reducing or eliminating their SORTWK file I/O. In this context, Hipersorting should be permitted to flourish - and DFSORT should be free to manage the overall expanded storage allocation for Hipersorting.

Recommendations

- Set EXPMAX=MAX, EXPOLD=MAX and EXPRES=0 in ICEMAC. These are the IBM-supplied installation defaults.
- Set HIPRMAX=OPTIMAL and DSPSIZE=MAX in ICEMAC and in individual sort tasks. These are the IBM-supplied installation defaults.

Tuning Large DFSORT Tasks

Monitoring and Statistical Data Collection

Producing the DFSORT Installation Options Report

The first step in any tuning exercise is, naturally, to audit the installation options of the software. For DFSORT, this is accomplished by using the ICETOOL DEFAULTS operator with a job like this one.

```
//DEFAULTS  JOB
//SHOWDEF   EXEC PGM=ICETOOL
//TOOLMSG   DD   SYSOUT=A
//DFSMSG    DD   SYSOUT=A
//LIST1     DD   SYSOUT=A
//TOOLIN    DD   *
  DEFAULTS LIST(LIST1)
/*
```

Figure 8 - JCL to produce DFSORT installation options report

The report produced contains all ICEMAC option settings, highlighting those values which differ from the IBM-supplied installation defaults. An example of what part of the report might look like for DFSORT R14 follows.

```
DFSORT REL 14.0 INSTALLATION (ICEMAC) DEFAULTS                - 1 -

   * IBM-SUPPLIED DEFAULT (ONLY SHOWN IF DIFFERENT FROM THE
     SPECIFIED DEFAULT)

ITEM           JCL (ICEAM1)              INV (ICEAM2)              ...
----------     --------------------      --------------------      ...
RELEASE        14.0                      14.0                      ...
MODULE         ICEAM1                    ICEAM2                    ...
APAR LEVEL     BASE                      BASE                      ...
COMPILED       07/15/98                  07/23/98                  ...

ENABLE         NONE                      TD1                       ...

ABCODE         MSG                       99                        ...
                                         * MSG                     ...
ALTSEQ         SEE BELOW                 SEE BELOW                 ...
ARESALL        0                         0                         ...
ARESINV        NOT APPLICABLE            0                         ...
CFW            YES                       YES                       ...
CHALT          YES                       YES                       ...
               * NO                      * NO                      ...

...
```

```
DFSORT REL 14.0 INSTALLATION (ICEMAC) DEFAULTS                - 4 -

   * IBM-SUPPLIED DEFAULT (ONLY SHOWN IF DIFFERENT FROM THE
     SPECIFIED DEFAULT)

ITEM           TD1 (ICETD1)             TD2 (ICETD2)             ...
----------     --------------------     --------------------     ...
RELEASE        14.0                     14.0                     ...
MODULE         ICETD1                   ICETD2                   ...
APAR LEVEL     BASE                     BASE                     ...
COMPILED       07/23/98                 07/15/98                 ...

SUN            0600-2000                NONE                     ...
               * NONE                                            ...
MON            NONE                     NONE                     ...
TUE            NONE                     NONE                     ...
WED            NONE                     NONE                     ...
THU            NONE                     NONE                     ...
FRI            NONE                     NONE                     ...
SAT            0600-2000                NONE                     ...
               * NONE                                            ...

ABCODE         99                       MSG                      ...
               * MSG                                             ...
ALTSEQ         SEE BELOW                SEE BELOW                ...
ARESALL        0                        0                        ...
ARESINV        0                        0                        ...
CFW            YES                      YES                      ...
CHALT          YES                      NO                       ...
               * NO                                              ...

...
```

Figure 9 - Sample DFSORT installation options report

Descriptions of all of the ICEMAC options can be found in "DFSORT Installation and Customization".

Of course, many of these installation option values can be overridden at run-time by one or more of the following:

- An installation-written DFSORT initialization exit, ICEIEXIT
- An application EXEC statement parameter or DFSPARM statement parameter
- An application OPTION statement, supplied either via DFSPARM or SYSIN

In addition, a number of values (for example, hiperspace and dataspace pages allocated) may be determined by the availability of resources at the time of execution. Therefore, it remains necessary to examine the output (or SMF data) of each individual DFSORT task to determine the option values that may have affected the performance of a particular task.

Note that with DFSORT R14, the installation options in effect can be controlled by time-of-day.

Of particular interest to the user of large DFSORT tasks is the specification of installation limitations on the ability of DFSORT to dynamically self-optimize.

The presence of an ICEIEXIT must be viewed with suspicion, and the functions of that exit investigated. The exit may attempt to place limits on the ability of users to specify MAINSIZE, HIPRMAX and DSPSIZE, in an attempt to limit the impact of sorting applications; effectively, to set a deliberate bias against DFSORT in system resource allocation. This may have a dramatic impact on the ability of DFSORT to process data efficiently.

Note that ICEIEXIT settings cannot be overridden by run-time settings such as those in an OPTION statement.

Similarly, placing stringent limits via TMAXLIM, HIPRMAX and DSPSIZE installation option values may have a severe impact on the overall performance of DFSORT. However, a limit set in this fashion can be overridden by run-time settings.

Recommendations

- Examine closely any **downward** variation from the IBM-supplied default for numeric values in ICEMAC options. Ensure that the implications of other variations are clearly understood.
- If present, ensure that the system-wide impact of ICEIEXIT limitations is recognized and understood.

DFSORT SMF Data

Of immense value in tracking and analyzing the performance of DFSORT tasks is SMF record type 16 (hex 10).

The layout and contents of this record are described in detail in Appendix C of "DFSORT Installation and Customization", or by expanding the ICESMF macro supplied in the ICEUSER dataset at installation.

The DFSORT SMF type 16 record contains a wealth of performance data for each DFSORT task executed. The type 16 data as well as additional performance data can be obtained and analyzed by an installation-written termination exit, ICETEXIT. A useful directory of the sources and potential analysis activities for DFSORT performance data is given in "DFSORT Tuning Guide".

The following REXX routine may be used to parse basic SMF type 16 data into CSV format for downloading to a spreadsheet or database - perhaps the most amenable medium for analysis.

```
/*****************************  REXX  ********************************/
/*  CSMF16                                                           */
/*                                                                   */
/*    Sample REXX to extract selected fields from DFSORT SMF type 16 */
/*    records and format to CSV                                      */
/*                                                                   */
/*    Refer to the ICESMF macro in DFSORT library ICEUSER for the    */
/*    layout of the SMF type 16 record and a descriptions of its     */
/*    fields.                                                        */
/*******************************************************************/

ARG debug
NUMERIC DIGITS 15

IF debug = "DEBUG"
```

```
THEN
  TRACE A

ADDRESS TSO

/*********************************************************************/

DO FOREVER

  "EXECIO 1 DISKR SMF16IN"
  IF rc ¬= 0
  THEN
    SIGNAL l990
  PARSE PULL inrec
  rectype  = C2X(SUBSTR(inrec,2,1))

  IF rectype ¬= 10
  THEN
    ITERATE

  smfdate  = SUBSTR(C2X(SUBSTR(inrec,7,4)),3,5)
  smftime  = C2D(SUBSTR(inrec,3,4))
  smftime1 = smftime % 100
  smfhh    = smftime1 % 3600
  smfhh    = RIGHT("0"||smfhh,2)
  smfmm    = (smftime1 % 60) - (smfhh * 60)
  smfmm    = RIGHT("0"||smfmm,2)
  smfss    = smftime1 - (smfhh * 3600) - (smfmm * 60)
  smfss    = RIGHT("0"||smfss,2)
  smftimef = smfhh||":"||smfmm||":"||smfss

  lpar     = SUBSTR(inrec,11,4)
  jobname  = SUBSTR(inrec,15,8)
  stepno   = C2D(SUBSTR(inrec,39,1))

  datasct@ = C2D(SUBSTR(inrec,57,4)) - 4 + 1
  stepname = SUBSTR(inrec,datasct@+2,8)

  twoe32   = 4294967296
  iceexrc1 = C2D(SUBSTR(inrec,datasct@+120,4))
  iceexrc2 = C2D(SUBSTR(inrec,datasct@+124,4))
  iceexrcs = (iceexrc1 * twoe32) + iceexrc2
  iceexby1 = C2D(SUBSTR(inrec,datasct@+128,4))
  iceexby2 = C2D(SUBSTR(inrec,datasct@+132,4))
  iceexbys = (iceexby1 * twoe32) + iceexby2
  icecput  = C2D(SUBSTR(inrec,datasct@+18,4))/100
  icekeyln = C2D(SUBSTR(inrec,datasct@+28,2))
  icewblk  = C2D(SUBSTR(inrec,datasct@+30,4))
  icendyna = C2D(SUBSTR(inrec,datasct@+35,1))
```

```
icefIby2 = C2D(SUBSTR(inrec,datasct@+36,1))
SELECT
  WHEN icefIby2 = 128 THEN sorttype = "S"
  WHEN icefIby2 = 64  THEN sorttype = "M"
  WHEN icefIby2 = 32  THEN sorttype = "C"
  OTHERWISE                sorttype = "O"
  END

icetimes = C2D(SUBSTR(inrec,datasct@+40,4))
icedates = C2X(SUBSTR(inrec,datasct@+44,4))
icetimee = C2D(SUBSTR(inrec,datasct@+48,4))
icedatee = C2X(SUBSTR(inrec,datasct@+52,4))
elapstm  = (icetimee - icetimes) / 100
IF icedatee ¬= icedates
THEN
  elapstm = elapstm + 86400.00

normbyte = C2D(SUBSTR(inrec,datasct@+56,1))
IF normbyte = 0
THEN
  normterm = "Y"
ELSE
  normterm = "N"

iceavlr  = C2D(SUBSTR(inrec,datasct@+60,4))
icedsa   = C2D(SUBSTR(inrec,datasct@+64,2))
iceinio  = C2D(SUBSTR(inrec,datasct@+84,4))
iceoutio = C2D(SUBSTR(inrec,datasct@+92,4))
icewkio  = C2D(SUBSTR(inrec,datasct@+100,4))
icehspu  = C2D(SUBSTR(inrec,datasct@+154,4))
icedspu  = C2D(SUBSTR(inrec,datasct@+160,4))

IF elapstm > 60.00
THEN
  DO
  outrec = smfdate  ||";"|| smftimef ||";"||,
           lpar     ||";"|| jobname  ||";"||,
           stepno   ||";"|| stepname ||";"||,
           normterm ||";"|| sorttype ||";"||,
           elapstm  ||";"|| icecput  ||";"||,
           iceexrcs ||";"|| iceavlr  ||";"||,
           icekeyln ||";"|| iceexbys ||";"||,
           icendyna ||";"|| icewblk  ||";"||,
           iceinio  ||";"|| iceoutio ||";"||,
           icewkio  ||";"|| icehspu  ||";"||,
           icedspu
  PUSH outrec
  "EXECIO * DISKW SMF16OUT"
  END
END
```

```
L990:

"EXECIO 0 DISKR SMF16IN  (FINIS"
"EXECIO * DISKW SMF16OUT (FINIS"

EXIT
```

Figure 10 - Sample REXX routine to extract SMF type 16 data

Sample execution JCL for this routine follows.

```
//   EXEC PGM=IKJEFT01,DYNAMNBR=20,PARM='%CSMF16'
//SYSEXEC DD DSN=userid.TEST.ISPCLIB,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD DUMMY
//SMF16IN DD DSN=userid.SMFDUMP.D98350.T16,DISP=SHR
//SMF16OUT DD DSN=userid.SMF.TYPE16,DISP=(,CATLG),
//   SPACE=(TRK,(1,1)),RECFM=VB,LRECL=255
```

Figure 11 - Sample JCL to execute REXX SMF type 16 data extract

---

Recommendations

- Ensure that DFSORT's SMF installation option is set to FULL (NO is the IBM-supplied default), and that type 16 collection is specified in the MVS SYS1.PARMLIB(SMFPRM00) parameters
- Collect and retain SMF type 16 data for all substantial DFSORT tasks. Consider investing in a simple spreadsheet or database-based data storage and statistical analysis application. However, you may need to dust off your Statistics 101 texts from college!

Rules of Thumb

Rule of thumb estimates are notoriously unreliable - especially given the wide range of potential installation and execution environments. They are often rapidly dated by the fast pace of change of both software and hardware.

The following formulae were derived from standard regression analysis of a large volume of SMF type 16 data, and relate to sorts with an input file size in excess of 3.5Gb.

Work file size = 0.93 * input file size + 1.77 * key size

(all expressed in Mb)

{key size = record count * key length}

Work I/O volume = 4.98 * work file size

(both expressed in Mb)

Elapsed time = input file size / 2.70

(elapsed time in seconds; input file size in Mb)

Recommendations

- Use the above formulae with caution, and as a guideline only. In particular, the overall sort rate figure of 2.70Mb/sec is an average, and may vary widely depending on the characteristics of the environment and on the nature and size of the sort task involved.
- Collect SMF data, and perform regression analysis against your own environment to determine more appropriate factors.

Tuning Large DFSORT Tasks

DFSORT and DB2 Utilities

DB2 utilities have progressed greatly since the days when an entire suite of vendor replacement utilities was a "must-have" for any substantial DB2 implementation. Intelligent interfacing of the utilities with DFSORT has been a key component of this improvement.

Similar general recommendations apply for DFSORT with large utility operation as with any other batch sort task; however, some special considerations warrant mentioning.

REORG TABLESPACE Utility

Three parameters for this utility relate to DFSORT performance.

SORTDATA

When specified, this parameter directs the utility to perform unload via a tablespace or partition scan and batch sort to clustering index sequence - as opposed to unloading using the clustering index, the equivalent of a SQL **SELECT * ... ORDER BY ...** statement.

Except where the underlying data clustering is very near to perfect, SORTDATA will provide a substantial - often radical - improvement in elapsed time. DB2 makes no decision to use SORTDATA dynamically, even if the CLUSTERRATIO column of the index in the DB2 catalog (SYSIBM.SYSINDEXES or SYSIBM.SYSINDEXSTATS) contains a low value.

Note, however, that if a clustering index is not **explicitly** defined on the table - as opposed to implicitly, by having the lowest OBID - DB2 will ignore the parameter if specified, and **always** unload via the **implicit** clustering index.

NOSYSREC

Ordinarily, REORG SORTDATA will unload the tablespace or partition to SYSREC, use DFSORT to perform a batch sort of this dataset, and reload the data by reading the sorted SYSREC.

NOSYSREC eliminates the I/O to SYSREC, by passing data directly to DFSORT in the UNLOAD phase of the utility, and retrieving data directly from DFSORT in the RELOAD phase.

One word of caution. If this parameter is used in a traditional (SHRLEVEL NONE) REORG execution, a SYSREC dataset is no longer available to re-start the utility should it fail during the RELOAD phase. The tablespace or partition must be RECOVERed - hence, a QUIESCE to establish a recovery point just prior to the REORG is mandatory. **SHRLEVEL NONE is the default.**

SORTKEYS

This parameter is specified without a key count estimate for REORG - this can be accurately determined by the utility itself. It directs REORG to pass **non-clustering index** key values directly from the RELOAD phase to a DFSORT subtask separate from that used for SORTDATA, rather than the traditional method of writing the key values to SYSUT1 for batch sorting into SORTOUT.

The index BUILD phase then retrieves the sorted key values directly from DFSORT, eliminating all I/O to SYSUT1 and SORTOUT. SYSUT1 and SORTOUT must still be present in the utility JCL, but may have minimal space allocation.

From Version 6 of DB2 on, this index building is accomplished by multiple concurrent sort tasks.

Note that this parameter is only of benefit for indexes **additional to** the partitioning or (explicit or implicit) clustering index - the UNLOAD phase has already ordered the key values for these indexes, and an index SORT phase is not executed.

Recommendations

- Always define the clustering index **explicitly** for any table of substantial size. Note, however, that this is always implied for the partitioning index of a partitioned tablespace.
- Where appropriate parameters are not specified to eliminate I/O altogether, or are inappropriate, specify DFSMS striping for SYSREC, SYSUT1 and SORTOUT. For REORG INDEX, always specify striping for SYSUT1.
- When the tablespace or partition contains variable-length columns, use the DFSPARM data set to specify an accurate tablespace or partition row count via the OPTION parameter FILSZ=Un. (DB2 assumes in its estimate to DFSORT that all variable-length columns are at maximum length.)
- Execute both LOAD and REORG at the partition level to reduce the granularity of the DFSORT tasks. Concurrent partition-level utility tasks can, however, present contention problems when NPIs are defined on the

tablespace.

LOAD Utility

The LOAD utility has a SORTKEYS parameter similar to that for REORG. Again, the parameter requests LOAD to pass index key and foreign key values directly from the RELOAD phase to a DFSORT subtask - rather than using SYSUT1/SORTOUT - and then directly to the BUILD and ENFORCE phases.

The following points are worth noting:

- If SORTKEYS is specified with no key count estimate, or with an estimate of zero, sorting takes place via SYSUT1 and SORTOUT as usual.
- If the tablespace or partition data to be loaded is presented to LOAD in pre-sorted key sequence, and only a single index exists on the table, the SORT phase is eliminated altogether.

Recommendations

- When LOADing large partitioned tables with only a partitioning index:
  - Pre-sort the input file using DFSORT
  - Use DFSORT's multiple output function (OUTFIL) to split the DFSORT output by partitioning key
  - Execute multiple concurrent partition-level LOAD utility tasks

  For large segmented tablespaces with only a single index, the input file should be similarly pre-sorted.
  - Specify SORTKEYS nnnnnnnn, calculating nnnnnnnn using the formulae stipulated in the DB2 Utilities Guide and Reference
  - If sorting using WORKDDN files cannot be avoided, specify DFSMS striping for SYSREC, SYSUT1 and SORTOUT.

DSNTIAUL Utility

DB2 (up to version 5) does not provide a standard "unload" utility; facilities are limited to sample program DSNTIAUL which simply executes dynamic SQL to produce a default-format (or user-specified) unload dataset.

Note that with version 6, the REORG utility provides this facility via the FORMAT EXTERNAL option, but without the full power of SQL.

As a general rule, executing a table unload via a SQL **SELECT ... ORDER BY** has always been preferable to unloading via a tablespace scan and post-sorting the unload dataset. Two exceptions are notable, however:

- For very large tablespaces, DSNDB07 may well be overwhelmed by a very large ORDER BY or GROUP BY key set. At the minimum, the sorting process may not scale well.
- Access paths may be a concern. A non-matching index scan may be selected, instead of a tablespace scan, or the degree of parallelism selected by DB2 may be less than that desired or possible.

One possible improvement is to:

- Unload the tablespace at partition level using DSNTIAUL, selecting and ordering or aggregating data by partition. A partition scan access path should be engineered by specifying partitioning key column values explicitly as literals.
- Use DFSORT MERGE to produce the final ordered dataset from the partition-level unloads, or use SUM to produce a finally aggregated file.

Note that parallel unload of data need not be restricted to partitioned tablespaces; the unload of any substantial segmented tablespace may be divided in this manner.

Tuning Large DFSORT Tasks

Conclusion

In essence, the battle to reduce the elapsed times of large DFSORT tasks reduces to an I/O contest. Put simply, the options available are to:

- **Reduce** the I/O
- **Accelerate** the I/O
- **Convert** the I/O from random to sequential, or to in-storage processing
- **Parallelize** the I/O

All of the recommendations in this paper fall under one or more of the above four headings.

Based on a substantial amount of experience, I would rate the following as the five most productive initiatives in improving the performance of large sorts. There is no particular order to the effectiveness of these initiatives - they should be considered as a group of options to be implemented in unison.

1. Allocate additional central storage via SIZE/MAINSIZE, TMAXLIM and DSA
2. Stripe SORTIN and SORTOUT datasets
3. Re-configure dataset allocations to disperse I/O - SORTWK I/O in particular
4. Increase the number of SORTWK files allocated
5. Allocate additional expanded storage via HIPRMAX and DSPSIZE

For the very largest sorts, deploy the sort/split technique.