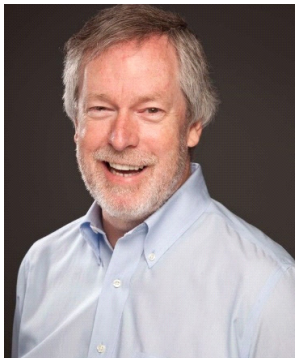
A PRACTICAL JOURNAL OF z/OS TUNING AND MEASUREMENT ADVICE

Cheryl Watson's Tuning Letter

REPRINT



This document is a reprint of a *Cheryl Watson's Tuning Letter 2022 No. 2* article titled **'Optimizing DFSORT Use of Z Sort Accelerator**'.

One of the enhancements on the IBM z15 is the IBM Integrated Accelerator for Z Sort. DFSORT exploitation of this capability is disabled by default, meaning that customers must explicitly enable it. This can be done at the system or individual sort level.

Readers have been asking about the best way to track exploitation of this new capability, and if there are things they can do to increase the number of sorts that can benefit from it. The relevant information is in the DFSORT SMF records, but not everyone has tools to format those records. To assist DFSORT customers quickly and easily get the required insights, we worked with IBM's **Sri Hari Kolusu** to create three sample ICETOOL reports and the information in this article.

In addition to providing information about Z Sort exploitation, the sample reports illustrate valuable techniques that could be applied if you would like to use ICETOOL to report on *other* SMF record types. Additionally, the article contains information to help readers that are not familiar with ICETOOL to customize the sample report jobs to meet your unique needs.

We hope this article and the corresponding report jobs will prove valuable. Please let us know if you have questions or suggestions. And THANK YOU to Kolusu for his invaluable help and suggestions in creating this article.

Optimizing DFSORT Use of Z Sort Accelerator

Regular Tuning Letter readers will know that we've taken a keen interest in the IBM Integrated Accelerator for Z Sort (hereafter simply referred to as 'Z Sort') capability that was delivered on the z15 generation of CPCs. Every z15 (or later) *core* includes a Sort Accelerator function at no additional charge. IBM benchmarks indicate sorts that are able to exploit the Sort Accelerator can achieve elapsed time savings of between 20% and 30%, and CPU time savings of up to 40%. Accordingly, Z Sort would seem to provide something for everyone - financial and capacity savings for the execs and accountants, and performance improvements for the techies and system users.

However, not every DFSORT 'sort' will benefit from Z Sort. For example, if you use DFSORT to simply copy the contents of one file to another, no sorting is involved, so that job step obviously would not use Z Sort. On the other hand, some sorts might not use Z Sort because of an environmental restriction, such as the step's MEMLIMIT being too low. In the latter case, adjusting the MEMLIMIT settings (an infrastructure-level change) might result in more sort job steps being able to benefit from Z Sort.

The objective of this article is to show how you can use information in the DFSORT type 16 SMF records to better understand:

- How many of your sort job steps are using Z Sort?
- What benefits are your sort steps achieving from the use of Z Sort?
- What are the most common reasons for sort job steps *not* using Z Sort, and are there actions you can take at the infrastructure level to increase the number of sorts that benefit from Z Sort?

Before we proceed, we want to thank **Sri Hari Kolusu** from the IBM DFSORT development team for his outstanding help, enthusiasm, and patience, *and* for providing the ICETOOL report programs this article is based around.

Target Audience

This article is appropriate for system programmers, performance and capacity analysts, production control staff, and application developers that use DFSORT. While the article uses DFSORT SMF records, you do not need to be an SMF expert to benefit from it. The sample DFSORT jobs we use to illustrate the examples in this article are available on the IBM DFSORT website and can be used with minimal JCL changes. It is possible to make relatively minor changes to the samples without having ICETOOL expertise, however more extensive changes *would* require some level of ICETOOL experience.

Introduction

We have already covered the Z Sort concepts and latest status information in previous Tuning Letter articles, so we will not go through all that information again. While it is not necessary to have that information before reading this article, it *would* be helpful if you decide to continue your Z Sort tuning exercise beyond the scope of this article. The articles in question are:

- z/OS Sort Accelerator in *Tuning Letter 2020 No. 3*.
- z15 Sort Accelerator Latest Status in *Tuning Letter 2021 No. 4*.
- There is also information in our SHARE in Dallas 2022 *Watson & Walker zRoadshow* .

We also want to remind you that IBM's zBNA tool was enhanced to provide additional reports and capabilities in support of Z Sort. One of the advantages of using zBNA is that it includes information from SMF type 30 records as well as the type 16 records that are the focus of this article. You can find more information in this excellent video: *Putting the New Z SORT Named Favorite into Practice* by **John Burg** and **Joel Moss** (while it might appear that you need a Box userid to download this file, if you just click on 'Download' you can download the file without having to logon to Box). We hope the information in this article will help you get even more value out of the Z Sort support in zBNA. However, if you are unable to install zBNA on your work computer, or if pulling the required SMF data down to your PC is not permitted or simply too much trouble, or if you want to view fields in the SMF records that are not provided by zBNA, the sample reports described in this article should help you.

Reminder: A typical sort consists of three parts[a]:

1. Read the data into memory.
2. Sort it into the new desired sequence.
3. Write the sorted data to the output data set.

The Sort Accelerator addresses the second of those parts - sorting the data. When predicting the benefits Z Sort might provide, remember that the elapsed time of the first and last parts will be unaffected by whether the actual sorting used the accelerator or not.

The IBM benchmark numbers mentioned above are based on the *entire job step's* elapsed and CPU times. For example, if the step elapsed time prior to enabling Z Sort was 100 seconds (that includes reading the file, sorting it, and writing the sorted data to the output file) and IBM claim a 20% improvement, that means that the total elapsed time dropped to 80 seconds. When you consider that Z Sort doesn't help the first or third items in the list above, that means that the elapsed time of the actual sort processing dropped by *more* than 20%. That's especially impressive when you consider that vendors have been tweaking and fine tuning sort algorithms for over 40 years now ☺.

a. These parts equate roughly to the description of DFSORT processing shown in https://www.ibm.com/docs/en/zos/2.4.0?topic=works-dfsort-processing#idg7073__stmtseq, with the first part including everything up to and including the INREC box, the second part being the SORT/SUM box, and the third part being everything including and after OUTREC processing.

For more information about general DFSORT tuning recommendations and best practices, refer to the 'Sorting Out Your Sort Performance' article in *Tuning Letter 2021 No. 1*.

Extracting Z Sort Insights from DFSORT SMF Records

The DFSORT exploitation of Z Sort is intended to be relatively transparent. You can turn Z Sort exploitation ON or OFF at the system or individual job step level. The default is for Z Sort to be *disabled*.

If the enabling PTF has been installed, DFSORT will examine the sort statements and the execution environment and determine if the Z Sort restrictions have been met. If they *have*, and the required resources are available, it will automatically use Z Sort. If they have *not*, or if the required resources are not available, it will not.

Note: Z Sort does *not* need to be enabled for a job step in order for DFSORT to provide you with a reason code indicating why Z Sort would not be used. This allows you (and zBNA) to evaluate sorts in advance of enabling Z Sort, so that you can get a feel for the potential benefits or the restrictions that will limit how many sorts can use Z Sort.

Looking at the sort at the job step level (in the SMF type 30 records, for example), it should be transparent to you whether Z Sort was used or not. This is all goodness - the easier it is to implement something, the more value it will deliver. And especially in these times, when most z Support teams have too much work and too few people, ease-of-installation and ease-of-use are even more important.

However, what about if you specifically *want* more information about how much value you are getting from Z Sort? IBM benchmarks are all well and fine, but it means much more when you can show your execs how much CPU time (and, potentially, money) Z Sort is saving *you*.

The ICE267I message in the sort step job log will tell you whether Z Sort was used, and if not, why not. That is very helpful information if you are investigating a particular sort. But what if you would like an overview of Z Sort exploitation across *all* your sorts? For that, we need our friend, SMF.

DFSORT writes a wealth of interesting information to its type 16 SMF records. However, if you want the records to contain information about the use of Z Sort (including whether Z Sort was even used or not), you *must* be using the SMF=FULL installation setting in DFSORT. (You can find more information about the SMF options in the 'Collecting statistical data' chapter of the *z/OS DFSORT Installation and Customization* manual).

SMF=FULL must be specified in order to get Z Sort information in DFSORT SMF records.

Note: The DFSORT manual warns against the use of SMF=FULL for sorts of variable length records, however we are assured that is *obsolete information* and can be ignored.

There are a number of key fields in the type 16 records that help you identify which sorts used Z Sort, or if they didn't use Z Sort, *why* they didn't:

ICEZSRT / ICEZSRTL / ICEZSRTN

These fields in the Header section provide the offset to, length of, and number of Z Sort sections. *Note that it is possible for a type 16 record to contain a Z Sort section even if Z Sort was not actually used for the sort*.

Note: This is a key point, so it is worth repeating. You will almost certainly have type 16 records that contain a Z Sort section even though the sort did *not* use Z Sort. However, not *all* sorts will contain a Z Sort section. This makes the processing of type 16 records a little more complex because your report program needs to cater for records that *do* have a Z Sort section, as well as records that do *not*.

ICEFLBY5 Flag bit indicating if Z Sort was used for this sort - if the bit is turned ON, Z Sort *was* used.

ICEZSRNU Byte containing the reason code describing why Z Sort was *not* used. The meanings of the reason code are described in the text for the ICE267I message.

The ICEFLYB5 and ICEXSRNU fields are both contained in the Data section of the type 16 records.

ICEZSFLG This flag byte is in the Z Sort section. If bit 0 is turned on, the contents of the Z Sort section are valid. Remembering that it is possible to get a Z Sort section even if Z Sort was not used, you should check the setting of this flag before assuming that the data in the Z Sort section represents an actual use of Z Sort.

The settings of the ICEFLBY5 and ICEZSFLG bits *should* be the same.

When the Z Sort-enabling DFSORT PTFs are installed on a system, *every* sort step will be evaluated to determine if it meets the criteria to be eligible to use Z Sort. This means that every SMF=FULL type 16 record *must* fall into one of two categories:

- The ICEFLBY5 flag is turned ON, indicating that the sort used Z Sort, or,
- The ICEZSRNU byte is non-zero, indicating that Z Sort was *not* used, and why not.

Sample Type 16 Record Analysis Reports

It is beyond the scope of this article to make you a DFSORT or ICETOOL expert. Fortunately, you do not *need* to be a DFSORT or ICETOOL expert to use the sample reports. Nevertheless, we felt that it would be helpful to spend a *little* time to explain the logic of the sample job that creates the detailed, job step-level, reports - this information might help if you want to change the job, or if your changes are not delivering the expected results.

Get the Sample Now: If you have not already done so, you should download the sample DFSORT jobs and associated symbol definition members from the DFSORT website now. This will enable you to step through the examples and try the jobs on your system as we go along.

Note that you should also have at least a few hours' worth of SMF type 16 records available - and don't forget that the records must have been produced after the SMF=FULL option was enabled.

Three sample report jobs are provided:

ZSDETRPT This job provides a report that shows detailed information about every job step that used Z Sort. It includes information from the Z Sort section of the type 16 record.

NZDETRPT This job provides a report that shows detailed information about every sort job step that did *not* use Z Sort.

SUMRPT This job provides a report showing a summary of the count and CPU time for all the jobs that *did* use Z Sort, and the same information for each of the reason codes for *not* using Z Sort.

Because the ZSDETRPT report contains information from the Z Sort section, it is the most complex of the three sample jobs. It consists of the following steps:

1. Delete the report data set from a previous run of the job.
2. Analyze the SMF records to determine the offsets to the Z Sort sections in the records.

3. A report step that shows information about each sort step that *did* use Z Sort.

DFSORT SMF records are structured like many other SMF records in that they have a Header section that is a fixed length, followed by a variable number of other sections. Each section is defined in a 'triplet' near the end of the Header section. A single Product section (also fixed length) follows the Header section. The Product section is followed by a single Data section. Depending on which functions are invoked in the sort, there may be a variety of sections following the Data section - for example, if there were multiple input files to the sort, then there will be multiple Input Data Set sections.

This variability means that the offset to the Z Sort section can vary. As you know, sort control statements are based on offsets in the record being processed, so this variability presents something of a challenge when processing these varying-length records with sort or ICETOOL. To get around this, Kolusu created a step that reads the input SMF file, determines the potential offsets to the Z Sort sections, and creates a set of DFSORT IFTHEN statements that are then input to the report step of the job.

See the ZSDETRPT report for an interesting technique to handle SMF records.

The ZSDETRPT report (described in 'Which Sort Job Steps Used Z Sort?' on page 8) displays fields from the Header, Product, Data, and Z Sort sections. Because the offset of the Z Sort section is variable, the report uses the IFTHEN statements to locate the Z Sort section and then remap the record in memory to move the Z Sort section to immediately after the Data section. His technique for reformatting SMF records into a format that is more ICETOOL-friendly might be of interest if you have struggled with trying to use ICETOOL with SMF records.

The NZDETRPT report (described in 'Which Sort Job Steps Did Not Use Z Sort?' on page 11) is similar to the ZSDETRPT report. However, because it focuses on the sorts that did *not* use Z Sort, it does not use any of the fields in the Z Sort section. This makes the ICETOOL statements for this report a little simpler. This job produces a report with information about each sort step that did *not* use Z Sort, along with an English description of *why* Z Sort was not used.

The other sample job (described in 'Summary of Overall Z Sort Usage' on page 18) is called SUMRPT. Rather than providing detailed, job step-level reports, it provides a summary of the various reason codes encountered by job steps that were not able to use Z Sort. It doesn't need information from the Z Sort section, so like the NZDETRPT report, its ICETOOL statements are a little simpler.

Note that Kolusu's samples rely heavily on the use of DFSORT symbols to relate field names to offsets. The use of symbols mean that you can use statements like this:

```
INCLUDE COND=(ICERTYP,EQ,16)
```

rather than like this:

```
INCLUDE COND=(6,1,BI,EQ,16)
```

The use of symbols also eliminates the need to manually calculate the offset for every field you might be interested in. The DFSORT website includes an XMIT-format PDS containing all the symbol definitions, with one member for each section in the FULL type 16 record.

Helpful documentation: All the information in the sample reports comes from the type 16 SMF records. You can easily add or remove fields to the reports (as we will see in 'Customizing the Sample Job' on page 20), so you might find it helpful to have the type 16 record layout to hand while reading the following sections. You can find the information online here: https://www.ibm.com/docs/en/zos/2.5.0?topic=customization-smf-type-16-record. If you want to see the information for *your* exact z/OS release and service level, the mapping macro for the type 16 records is contained in the ICESMF member of the SYS1.SICEUSER data set.

Which Sort Job Steps Used Z Sort?

The ZSDETRPT report provides information about every job step that *did* use Z Sort. You might want to use such a report as a quick, single-point-of-reference, to see if a given sort step used Z Sort. You *could* also find this information in the job's log, however, support staff don't always have security access to read production job logs. Also, if you want to check on more than one job, it would be a lot faster to find the information in one data set, rather than having to go in and out of multiple job logs.

Figure 1 - Overview section of ZSDETRPT report

IZSORT USAGE REPORT 07/08/22 05:47:00 - 1 -

JOB Name	STEP Name	STEP Num	Step End Date	Step End Time	SYSID	TCB Time	SRB Time	Num Recs Sorted
FPK01XPA	STEP0001	1	2022-05-25	22:28:13	FPKB	26.06	5.01	30,671,598
FPK01PNC	STEP0001	1	2022-05-25	23:25:10	FPKB	70.02	4.69	202,700,913
FPK0124C	STEP0003	3	2022-05-25	07:38:55	FPKB	21.10	4.77	8,325,581
FPK01DPA	STEP0004	3	2022-05-25	23:33:07	FPKA	19.40	4.41	90,133,849
FPK01DTD	STEP0001	2	2022-05-26	02:49:45	FPKA	31.21	3.37	36,771,536
FPK01EMH	STEP0001	1	2022-05-26	00:11:05	FPKB	19.16	3.33	69,488,148
FPK01CJA	STEP0001	1	2022-05-26	00:11:07	FPKB	18.50	3.26	69,488,148
FPK01EMF	STEP0001	1	2022-05-26	00:11:06	FPKB	19.17	3.13	79,831,759
FPK01CJB	STEP0001	1	2022-05-26	00:11:11	FPKB	24.89	3.08	79,831,759
FPK01DLC	STEP0003	3	2022-05-25	06:13:24	FPKB	14.43	2.92	5,274,572
FPK01DDB	STEP0003	3	2022-05-26	02:01:29	FPKB	33.78	2.47	134,077,045
FPK01EBC	STEP0001	1	2022-05-26	00:28:51	FPKB	19.70	2.43	27,419,379
FPK0124C	STEP0003	3	2022-05-25	09:16:27	FPKB	12.51	3.17	4,913,308
FPK01RDC	STEP0001	1	2022-05-26	00:32:59	FPKB	19.02	2.51	27,419,379

An example of the left-hand side of the report is shown in Figure 1. This section of the report provides the job name, step name, step number, and the step end date and time (unfortunately the type 16 record doesn't contain the job step start time). It would be nice to also be able to search using the job number, however that information is currently not contained in the type 16 record.

This report also shows the total TCB time (to an accuracy of hundredths of a second), SRB time (also accurate to hundredths of a second), and the number of records sorted. The SRB time field is interesting because it illustrates how you can easily perform calculations on the

raw data that is read by ICETOOL. The type 16 record contains the SRB time at the start of sort processing (ICESRBTS) and at the end of the sort (ICESRBTE), but not the difference between the two. The sample report subtracts the ICESRBTS value from the ICESRBTE value and provides the difference in the SRB Time column.

Figure 2 - Memory-related section of ZSDETRPT report

Megabytes Sorted	MOSIZE (MB)	Mem Obj Used (MB)	Main Stor X in KB	Main Stor Y in KB	Main Stor Z in KB
25565	2,147,483,647	34865	8192	60968	60955
26960	2,147,483,647	32463	8192	67357	67344
24372	2,147,483,647	24595	8192	61958	61945
19340	2,147,483,647	20120	8192	53321	53309
16832	2,147,483,647	18283	8192	50634	50617
16302	2,147,483,647	17140	8192	48828	48815
16302	2,147,483,647	17140	8192	48828	48815
15683	2,147,483,647	16583	8192	48389	48376
15683	2,147,483,647	16549	8192	47930	47918
15179	2,147,483,647	16478	8192	48895	48882
14576	2,147,483,647	15397	8192	46385	46368
13885	2,147,483,647	14692	8192	45258	45245
14417	2,147,483,647	14571	8192	47653	47640
13885	2,147,483,647	14495	8192	44921	44909

The next section over to the right (shown in Figure 2) shows the number of MBs sorted, the DFSORT MOSIZE value in effect for the step, and the number of memory objects actually used. Remember that sorts that use memory objects are more efficient than those that use sort work data sets, so these fields give you an idea of how much memory the sorts are currently using, and how close to the MOSIZE limit the jobs are. If you use the default MOSIZE, it is *extremely* unlikely that any jobs will get anywhere close to the limit, but if you override the MOSIZE value, then it *is* possible that your MOSIZE could become a constraint on the use of Z Sort.

If you are not familiar with the many storage-related DFSORT parameters, we highly recommend reading the 'DFSORT Storage Considerations' section of the *DFSORT Installation and Customization Guide.* There is also a very helpful SHARE presentation - *DFSMS DFSORT: Resource Usage and Understanding*, by **Vicky Vezinaw**.

Figure 3 - ZSDETRPT information about sort work data sets

Num SORTWK EXCPs	SORTWK Trks Alloc	SORTWK Trks Used	Num dyn alc wkds
32	0	0	35
32	0	0	35
32	0	0	35
32	0	0	35
32	0	0	35
32	0	0	35
32	0	0	35
32	0	0	35
32	0	0	35
32	0	0	35
32	0	0	35
32	0	0	35
32	0	0	35
32	0	0	35
32	0	0	35

The next section over to the right again (shown in Figure 3) provides information about sort work data sets. Two interesting metrics are the number of EXCPs to SORTWK data sets[1] and the number of allocated SORTWK tracks. Ideally, a sort that uses Z Sort will not use

[1] Note that DFSORT issues 1 'test' EXCP for each possible SORTWK data set, to verify that the sort work data sets, if allocated, would support zHPF.

SORTWK data sets *at all*, so it might be worthwhile to review the report for sort steps with more than zero sort work data set tracks (ICEWALLE). In our sample data, there were steps that sorted less than 700 **M**B that had over 100,000 tracks of sort work data sets, while other steps that sorted more than 4 **G**B with 0 tracks of sort work data sets. A little JCL modernization to remove //SORTWKnn DD statements might eliminate the unnecessary allocation of those sort work data sets.

Sample Job to Print DFSORT Default Values

As you review which jobs did or did not use Z Sort, it might be helpful to have a list of the default DFSORT values that are in use on your system. You can print those values using this JCL:

```
//ICEDFLT  JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//SHOWDEF EXEC PGM=ICETOOL,REGION=1024K
//TOOLMSG DD SYSOUT=*
//DFSMSG  DD SYSOUT=*
//LIST1   DD SYSOUT=*
//TOOLIN DD *
DEFAULTS LIST(LIST1)
/*
```

The final section (shown in Figure 4), over on the far right of the report, provides information that is only available for sorts that used Z Sort. The 'zSORT Flag' column interprets the ICEZSFLG flag byte. The zSORT Phase 1 Elapsed and TCB time columns report the time that DFSORT spends processing the data after it has been read into memory but before it is sent to the Sort Accelerator for sorting. And the Phase 3 represents the cleanup work at the end of the sort.

Figure 4 - Z Sort-unique fields in ZSDETRPT report

zSORT Flag	zSORT Ph1 ELP Time	zSORT Ph1 TCB Time	zSORT Ph3 ELP Time	zSORT Ph3 TCB Time
ZSORT statistics available and valid	72.25	16.36	222.66	0.30
ZSORT statistics available and valid	80.18	45.63	214.41	7.46
ZSORT statistics available and valid	66.32	10.19	203.15	0.13
ZSORT statistics available and valid	51.71	8.39	161.78	2.22
ZSORT statistics available and valid	45.66	22.45	121.03	0.86
ZSORT statistics available and valid	54.03	8.38	160.69	3.22
ZSORT statistics available and valid	53.98	7.61	163.51	3.36
ZSORT statistics available and valid	53.49	8.16	162.97	3.10
ZSORT statistics available and valid	57.15	14.91	163.47	2.53
ZSORT statistics available and valid	40.50	7.52	114.70	0.10
ZSORT statistics available and valid	42.58	21.28	112.99	2.28
ZSORT statistics available and valid	50.83	11.41	131.18	0.32
ZSORT statistics available and valid	39.01	5.99	109.94	0.08
ZSORT statistics available and valid	40.18	11.92	122.19	0.31

If there are other fields that you believe would be valuable, check the provided ZSDETRPT job. The ICETOOL statements include a number of fields that we commented out - you might find that you can get the field you are interested in by simply uncommenting that statement. For more information about modifying the supplied sample jobs, refer to 'Customizing the Sample Job' on page 20.

Which Sort Job Steps Did *Not* Use Z Sort?

The NZDETRPT report shows the sort steps that did *not* use Z Sort, and *why* they didn't use Z Sort. It is important to remember that it simply wouldn't make sense for some 'sorts' to use the Sort Accelerator. One example we mentioned previously is where you use DFSORT to simply copy or merge a data set - such operations do not sort the data, so the Sort Accelerator has no role in that operation. There are other scenarios where DFSORT currently doesn't use the Sort Accelerator, but might do so in the future. And, finally, there are situations where Z Sort is not used for environmental reasons - for example, if there is insufficient real or virtual storage available to the sort. This latter group are the most interesting ones in the context of this article.

A snippet from the leftmost section of the NZDETRPT report is shown in Figure 5. You can see this information is very similar to the ZSDETRPT report.

Figure 5 - Overview section of the NZDETRPT report

ZSORT NON-USAGE REPORT 07/07/22 13:09:35 - 1 -

JOB Name	STEP Name	STEP Num	Job Run Date	Job Run Time	SYSID	TCB Time	SRB Time	Num Recs Sorted
FPK01BAC	STEP0001	1	2022-05-25	05:01:34	FPKA	0.32	0.00	129,817
FPK01PAI	STEP0001	1	2022-05-25	05:04:45	FPKA	2.87	0.16	4,607,224
FPK01DFF	STEP0001	2	2022-05-25	06:32:20	FPKA	0.01	0.00	18
FPK01DFF	STEP0001	2	2022-05-25	08:17:03	FPKA	0.01	0.00	992
FPK01DFF	STEP0001	2	2022-05-25	08:19:12	FPKA	0.01	0.00	62
FPK01DFF	STEP0001	2	2022-05-25	08:21:48	FPKA	0.01	0.00	11
FPK01DFF	STEP0001	2	2022-05-25	08:25:44	FPKA	0.01	0.00	3,816
FPK01DFF	STEP0001	2	2022-05-25	08:38:58	FPKA	0.01	0.00	1,230
FPK01DFF	STEP0001	2	2022-05-25	08:39:30	FPKA	0.01	0.00	33
FPK01DFF	STEP0001	2	2022-05-25	08:41:26	FPKA	0.01	0.00	48
FPK01DFF	STEP0001	2	2022-05-25	08:46:48	FPKA	0.01	0.00	3
FPK01DFF	STEP0001	2	2022-05-25	08:47:06	FPKA	0.01	0.00	3
FPK01DFF	STEP0001	2	2022-05-25	08:59:56	FPKA	0.01	0.00	15
FPK01DFF	STEP0001	2	2022-05-25	09:06:12	FPKA	0.27	0.00	54,312

One of the differences between the NZDETRPT and the ZSDETRPT reports is that the NZDETRPT report shows the reason why Z Sort was not used if you scroll to the right (in plain English, as shown in Figure 6). Also, as you can see, the NZDETRPT report is sorted on the reason code, so all the job steps that had the same reason code are grouped together. Of course, you can easily change the sort order to be based on any field you wish.

Figure 6 - Z Sort reason code for each job step

ZSORT Non-usage	Description
194	zSORT is not enabled/activated
194	zSORT is not enabled/activated
194	zSORT is not enabled/activated
194	zSORT is not enabled/activated
194	zSORT is not enabled/activated
194	zSORT is not enabled/activated
194	zSORT is not enabled/activated
194	zSORT is not enabled/activated
194	zSORT is not enabled/activated
194	zSORT is not enabled/activated
194	zSORT is not enabled/activated
194	zSORT is not enabled/activated
194	zSORT is not enabled/activated
194	zSORT is not enabled/activated

At the time of writing this article, there are nearly 30 documented reasons for DFSORT not using Z Sort. Table 1 on page 12 shows all the reasons provided in the ICE267I message, a brief description, an indication of whether each one can be addressed at the infrastructure level, and a brief comment with a little more information. As you will see in the table, many of

the reasons for a sort not using Z Sort are outside your control. Therefore, the NZDETRPT report focuses on aspects of each sort that *are* under your control.

Table 1 - Reasons Why Z Sort Was Not Used

Reason Code	Description	Addressable at the infrastructure level?[a]	Comment
193	NOT RUNNING ON A Z15 AND HIGHER PROCESSOR.	**Possibly**	Direct job to a system running on a z15 or later if possible.
194	ZSORT IS NOT ENABLED/ACTIVATED	**Possibly**	Enable Z Sort in ICEPRMxx member or by specifying OPTION ZSORT via //DFSPARM DD *
195	SORT FIELDS=COPY OR OPTION COPY IS USED	NO	Not supported for use with Z Sort.
196	MERGE FIELDS= IS SPECIFIED	NO	Not supported for use with Z Sort.
197	INREC IS SPECIFIED	NO	Not supported for use with Z Sort.
198	OUTREC IS SPECIFIED	NO	Not supported for use with Z Sort.
199	OUTFIL IS SPECIFIED	NO	Not supported for use with Z Sort.
200	PROGRAM INVOKED SORT	NO	Not supported for use with Z Sort.
201	FAILED TO GET THE REQUIRED MEMORY OBJECT	**Possibly**	See the ICE267I message description for list of ways to address this.
209	MINIMUM MEMORY REQUIREMENTS ARE NOT MET	**Possibly**	See the ICE267I message description for list of ways to address this.
210	UNABLE TO OBTAIN REQUIRED MEMORY OBJECT	**Possibly**	See the ICE267I message description for list of ways to address this.
211	UNSUPPORTED SORT FIELDS	NO	Not supported for use with Z Sort.
212	SUM FIELDS= IS SPECIFIED	NO	Not supported for use with Z Sort.
213	JOINKEYS MAINTASK, SUBTASK1 OR SUBTASK2	NO	Not supported for use with Z Sort.
214	MODS STATEMENT IS SPECIFIED	NO	Not supported for use with Z Sort.
215	UNKNOWN FILE SIZE OR FILSZ=0	**Possibly**	See 'Reason 215, 233 - FILSZ Considerations for Z Sort' on page 16.
216	VARIABLE LENGTH INPUT FILE LRECL <= 24	NO	Not supported for use with Z Sort.

Reason Code	Description	Addressable at the infrastructure level?[a]	Comment
217	INPUT OR OUTPUT IS A VSAM CLUSTER	NO	Not supported for use with Z Sort.
226	EXCP ACCESS METHOD WAS FORCED TO BE USED	NO	Not supported for use with Z Sort.
227	NOT ENOUGH BELOW-THE-BAR STORAGE	**Possibly**	Specify SIZE=MAX in ICEPRMxx or OPTION MAINSIZE=MAX as a runtime option. Also, refer to the section titled 'Required main storage' in the *DFSORT Installation and Customization Guide*.
228	SORT FIELDS BEYOND 4092 FOR VARIABLE RECORDS	NO	Not supported for use with Z Sort.
229	SORT KEY LENGTH > 4080 OR 4088	NO	Not supported for use with Z Sort.
230	INTERNAL CONDITION THAT PREVENTS ZSORT	NO	Not supported for use with Z Sort.
231	SIZE OF THE FILE TO SORT < 32 MB	NO	Not supported for use with Z Sort.
232	ZHPF DISABLED FOR SORTWK DATASETS	**Possibly**	Verify that your disk subsystem supports zHPF, then enable zHPF in IECIOSxx member of Parmlib.
233	SIZE/FILSZ=UXXXXXX IS SPECIFIED	**Possibly**	See 'Reason 215, 233 - FILSZ Considerations for Z Sort' on page 16.
240	FAILED TO GET MEMORY OBJ OF 75% OF FILE SIZE	**Possibly**	See the ICE267I message description for list of ways to address this.
241	INSUFFICIENT DISK WORK SPACE	**Possibly**	Remove SORTWKnn DD statements and let DFSORT allocate them dynamically.

a. NOTE. This reflects DFSORT exploitation of Z Sort in June 2022. It is possible that some of these restrictions might be addressed in future PTFs or DFSORT releases.

As you can imagine, there could be multiple reasons why a given sort doesn't use Z Sort; for example, NOZSORT was specified, AND the job is running on a z14, AND it is a COPY rather than a SORT. The reason that is presented in the ICE267I message (and in the ICEZSRNU field in the type 16 SMF record) depends on the sequence in which DFSORT code evaluates the sort. If DFSORT detects a condition that means that it can't use Z Sort for that sort, it will take a note of that condition and then revert to a traditional sort. As a result, it is possible that you might address the reason shown in the report and find that the sort still doesn't use Z Sort, but for a different reason. Nevertheless, by addressing that reason you will have moved a step closer to being able to benefit from Z Sort. And if you

refer to the list of reasons shown in Table 1 on page 12 while you are reviewing a particular sort, you might be able to use that opportunity to address any other potential reasons for the sort not being able to use Z Sort.

We hope that you can easily combine the information in the NZDETRPT report with that provided in the table above and the following sections to determine if you can take action to increase a sort's chances of being eligible to use Z Sort.

Reason 193 - Sort Not Run on z15

The Sort Accelerator is available on all z15 (both business class and enterprise class) and later CPCs at no additional charge. If a sort is run on a system that has the DFSORT Z Sort-enabling PTFs, but that system is running on a CPC older than z15, reason code 193 will be reported. Obviously your IBM team would love you to run out and upgrade your CPC to a z15 (or, even better, a z16!). However, in the interim, if you have at least one z15 in your installation, you might be able to route that job to a system running on the z15, for example by adding a WLM Scheduling Environment that is enabled on systems running on a z15, and adding a SCHENV statement to the job that points at that Scheduling Environment.

Reason 194 - Z Sort Not Enabled

If your system has the Z Sort PTFs for DFSORT and it is running on a z15 or later CPC, the sort will produce reason 194 if Z Sort is not enabled. You can control whether or not Z Sort is enabled by default by updating the ZSORT parm in your ICEPRMxx member of Parmlib.

If Z Sort IS enabled at the system level, but the sort is still producing reason code 194, that means that the system-level setting is being overridden at the step level through the use of the OPTION NOZSORT statement in the //DFSPARM DD * JCL statement. If that is the case, it is possible that someone explicitly disabled Z Sort for that sort because of a previous problem encountered when that sort tried to use Z Sort. You should investigate this before removing the NOZSORT setting.

Reasons 201, 209, 210, and 240 - MEMLIMIT-Related Issues

DFSORT use of memory, and the many DFSORT parameters related to controlling that use, is a complex topic that deserves an article all of its own. It is certainly more than we can cover in this article. However, there are just a few things to keep in mind:

- The objective of the Sort Accelerator is to reduce the time to perform a given sort. To achieve this, you really don't want to be shuffling data back and forth to sort work data sets during the sort process. Achieving this means giving the LPAR sufficient memory, and giving each sort enough below-the-bar virtual storage and (64-bit) memory objects to perform the sort in memory.

  - Don't forget the potential knock-on benefits of more efficient sorts. For example, if more efficient sorting allows you to reorganize your databases more frequently, that might result in reduced CPU consumption in the tasks that access that data.

- At the start of each sort, DFSORT uses the size of the input file(s) to determine how much memory it will need to perform an in-memory sort. The sort must be able to allocate enough memory objects (pageable 1MB page frames) to hold *at least* 75% of the input files in order to be eligible to use Z Sort. Ideally, there will be enough memory to allocate memory objects that sum to *twice* the size of the input files. (Refer to the 'z/OS Sort Accelerator' article in *Tuning Letter 2020 No. 3* for a simplified description of how a sort is performed).

- There are DFSORT options to limit the amount of memory that can be used by an individual sort and also by *all* concurrent sorts.
  - The *recommended* individual-sort-level values (MOSIZE=MAX and HIPRMAX=OPTIMAL) are also the defaults.
  - The *recommended* system-level limit for sort memory use (EXPMAX=MAX) is also the default.
  - As a result, if you don't change any of the defaults you should be well positioned from *a DFSORT perspective.*

- However, regardless of what DFSORT might *like* to do, each sort might also be limited by the amount of above-the-bar memory available to the address space. This is controlled by the MEMLIMIT parameter. The MEMLIMIT value that controls a given job step can come from a number of places:
  - The system-level default value. This is set using the MEMLIMIT parm in the SMFPRMxx Parmlib member. You can determine the current value by using the `D SMF,O` command.
  - The MEMLIMIT keyword on the sort's JOB or EXEC JCL statement.
    - Specifying REGION=0 results in an unlimited MEMLIMIT for that step.
  - The SMFLIMIT member of Parmlib.
  - An IEFUSI exit.

  The *actual* MEMLIMIT for the step is shown in SMF type 30 field SMF30MEM. To determine which of these places set the MEMLIMIT for the sort step, check the SMF30MES and SMF30SLM fields in the type 30 SMF record for that step.

If Z Sort is not used for a sort, and the reason is one of 201, 209, 210, or 240, refer to https://www.ibm.com/docs/en/zos/2.5.0?topic=messages-ice267i for guidance about possible ways to address the restriction.

Reason 215, 233 - FILSZ Considerations for Z Sort

The FILSZ or SIZE sort parms can be used to provide information to DFSORT about the number of records to be sorted. DFSORT can perform a sort more efficiently if it knows roughly how much data it is going to sort. Generally speaking, DFSORT is able to determine this information dynamically, so there is usually no need to specify a FILSZ or SIZE value. In fact, even in the situation where the data set to be sorted resides on tape, DFSORT is usually able to determine its size from your tape management product.

However, there *are* some situations where DFSORT is unable to dynamically determine the file size:

- When records are inserted via an E15 exit.
- When DFSORT is invoked via a program.
  - **NOTE**: There are *two* exceptions to this rule. DFSORT CAN use Z Sort for the following program-invoked sorts:
    - If DFSORT is able to read the SORTIN file (in this case, bit 5 of field ICEFLBYT and bit 3 of field ICEIOTYP in the type 16 record will be set).
    - If DFSORT was called by Db2 *and* the data to be sorted has variable length records.
- A tape input does NOT have standard label information.

DFSORT will consider such sorts to be ineligible for Z Sort *unless* the file size is provided on a FILSZ=nnnn or FILSZ=Ennnn statement.

Specifying FILSZ=Unnn forces DFSORT to use the provided file size, which might or might not be accurate. If the values are inaccurate, DFSORT might allocate too few or too many resources for the sort. As a result, sorts containing FILSZ=Unnn statements are also considered ineligible for Z Sort. Additionally, specifying FILSZ=0 disables DFSORT's ability to dynamically determine the input file size, so DFSORT will not try to use Z Sort for such sorts.

Apart from the three scenarios listed above, DFSORT generally runs more efficiently if FILSZ is *not* specified. For this reason, we recommend avoiding the use of the FILSZ parameter for sorts unless you have determined that you *really do* need to use it.

Bit 2 of the ICEFSZFL field in the SMF type 16 record will be turned ON if the Unnnn form of the FILSZ parameter was specified for the sort. The FILSZ value is contained in the ICEFILSZ field - if that field is zero and bits 0 or 1 in ICEFSZFL are turned ON, that indicates that FILSZ=0 was specified. Either of these conditions will stop the sort from being able to use Z Sort. If you check the NZDETRPT report, you will see that it reports if FILSZ was specified in the sort control statements, and if so, it shows the value specified on the FILSZ= statement.

Regardless of Z Sort support, the use of FILSZ might be an interesting topic to review anyway. If the use of FILSZ can negatively impact sort performance, it might be interesting to see how widespread its use is - if you have many sorts using it where it is not required, you might improve overall sort performance if you can eliminate unnecessary uses of this parameter.

Tuning opportunity - eliminate unnecessary use of FILSZ parm.

You can find more information about FILSZ in the *OPTION control statement* section of the DFSORT Application Programming Guide. Also refer to the section *File Size and Dynamic Allocation* in that same manual for information on situations where DFSORT cannot determine the file size accurately, and what to do about it.

Reason 227 - Insufficient Below-the-Bar Storage

This reason is related to 31-bit storage, as opposed to 64-bit memory object storage. It can be caused by a too-small REGION value in the JCL, or too-low DFSORT memory parms. If the REGION size is reasonable, you could try specifying MAINSIZE=MAX if that is not your default. If that doesn't help, review your MAXLIM and TMAXLIM values. Also, refer to the section titled 'Required main storage' in the *DFSORT Installation and Customization Guide*. In particular, there is a very helpful subsection called 'Storage Considerations' in that manual that shows the impact of various combinations of these parameters.

Reason 232 - zHPF Not Enabled for Sort Work Data Sets

DFSORT requires that the sort work data sets are allocated on disk volumes that support zHPF. In our experience, nearly all customers have enabled zHPF by now. But if you are one of the holdouts, check with your disk vendor to ensure that your subsystem(s) support zHPF, and then enable zHPF by updating the IECIOSxx member to say ZHPF=YES (the default is NO, so you must explicitly specify this parameter).

Reason 241 - Insufficient Disk Work Space

The recommended approach to allocating SORTWKn data sets is to remove those DD statements from the sort jobstep and allow DFSORT to dynamically determine how many sort work data sets are needed, and their size, based on the size of the input files. However, it is still very common to find jobs that still contain //SORTWKn DD statements, so it is possible that they simply have too-small SPACE values. Unless the sort is one of those unusual ones where DFSORT is unable to dynamically determine the size of the input data set(s), we recommend removing the //SORTWKn DD statements and let DFSORT dynamically allocate the required data sets.

Note: The default maximum number of dynamically allocated sort work data sets is 4. If you have very large sorts, you might want to increase that maximum to a larger value - this can be done using the DYNALOC installation parameter.

Summary of Overall Z Sort Usage

The third sample report is called SUMRPT. This report has one line for each reason code encountered in the type 16 records you provided, sorted by SYSID. The line shows the reason code, the SYSID, an English description of the meaning of that code, a count of the number of job steps that encountered that reason code, and the total sort CPU time (broken out into TCB time and SRB time) for those job steps, with totals at the system level. An example is shown in Figure 7.

Figure 7 - Z Sort summary report

Summary report of zsort usage/nonusage 2022/07/10 10:00:47

ZSORT Code	System-id	Description	Num Steps	Total CPU(HS)	Total SRB(HS)
000	FPKA	zSORT is used	953	1,768.56	224.41
194	FPKA	zSORT is not enabled/activated	184	32.89	0.78
195	FPKA	SORT FIELDS=COPY or OPTION COPY is used	21,895	5,108.40	235.13
196	FPKA	MERGE FIELDS= is specified	428	3.82	0.00
197	FPKA	INREC is specified	15	6.13	0.12
198	FPKA	OUTREC is specified	27	78.99	9.45
199	FPKA	OUTFIL is specified	1,199	407.67	6.17
200	FPKA	Program Invoked Sort or Db2 REORGs FLR Sort	7,037	1,466.88	20.93
201	FPKA	Failed to get the required memory object	1	2.78	0.59
209	FPKA	Minimum memory requirements are NOT met	2	0.69	0.02
211	FPKA	Unsupported sort fields	1	0.01	0.00
212	FPKA	SUM FIELDS= is specified	1,417	580.90	20.24
213	FPKA	JOINKEYS Maintask, Subtask1 or Subtask2	380	236.39	5.22
214	FPKA	MODS Statement is specified	351	87.95	0.01
231	FPKA	Size of the file to sort < 32 MB	10,860	579.43	0.00
		Totals :	44,750	10,361.49	523.07
000	FPKB	zSORT is used	689	1,028.30	119.89
194	FPKB	zSORT is not enabled/activated	2	2.26	0.07
195	FPKB	SORT FIELDS=COPY or OPTION COPY is used	3,024	352.31	43.91
196	FPKB	MERGE FIELDS= is specified	15	0.10	0.01
197	FPKB	INREC is specified	6	3.02	0.06
198	FPKB	OUTREC is specified	13	9.15	0.20
199	FPKB	OUTFIL is specified	10	25.77	1.07
200	FPKB	Program Invoked Sort or Db2 REORGs FLR Sort	985	4,246.78	298.59
201	FPKB	Failed to get the required memory object	1	0.31	0.01
211	FPKB	Unsupported sort fields	4	1.89	0.34
212	FPKB	SUM FIELDS= is specified	512	493.28	8.36
213	FPKB	JOINKEYS Maintask, Subtask1 or Subtask2	542	227.93	2.92
214	FPKB	MODS Statement is specified	39	94.38	2.29
217	FPKB	Input or output is a VSAM cluster	1	0.35	0.01
231	FPKB	Size of the file to sort < 32 MB	5,918	516.64	0.25
240	FPKB	Failed to get memory obj of 75% of file size	6	2.12	0.09
		Totals :	11,767	7,004.59	478.07

This example shows information about two systems, called FPKA and FPKB, sorted in order of reason code within system. You will notice that the first line is for reason code 000 - those are the job steps that *did* use Z Sort. Looking at system FPKA, you will see that the group of sorts that successfully used Z Sort was the second-highest group on that system in terms of CPU consumption, even though the actual number of job steps was relatively small - just 953 out of a total of about 45,000. This is good - it shows that the sorts that are benefiting from Z Sort are the more CPU-intensive ones.

Overall, you can see the report gives you a very quick way to identify the reason codes that are impacting the largest number of job steps. For example, you can see that the most common reason code on system FPKA was 195 - sorts that were actually just copies. Those ‘sorts’ will never use Z Sort, so they can be ignored from the perspective of trying to optimize the use of Z Sort.

After sorts that did use Z Sort, the next largest group in terms of CPU consumption are the ones that had reason code 200 - these are program-invoked sorts or Db2 reorgs. DFSORT currently doesn't support Z Sort for program-invoked sorts unless the size of the input files is available to the sort or if DFSORT was called by a Db2 utility, and the data to be sorted consists of variable length records.

Moving down the list, we see that there were nearly 11,000 sorts on FPKA that didn't use Z Sort because of the relatively small amount of data to be sorted. The threshold is hard-coded in DFSORT, and was set based on the cost of setting up the environment to use Z Sort - for very small sorts, the potential savings from Z Sort would not be large enough to offset the setup cost. While you have no control over the threshold, this is still useful information. *The* most efficient sort is one that is carried out using just the 31-bit memory in the address space. Based on your knowledge of the threshold for sending a sort to Z Sort, you might be able to optimize your DFSORT memory-related parms to ensure that smaller sorts (those of up to 32MB of data) can be completed using memory in the address space.

In fact, looking through the remaining reason codes, there were actually very few sorts on FPKA that could potentially use Z Sort but that were artificially constrained. This was a particularly well set-up system, with lots of memory, so you should not necessarily expect to see such a positive result if you run this report for *your* system. Regardless, we still believe this report will be very helpful if you are trying to determine the scope for greater use of Z Sort, and the information provided should help you identify where to concentrate your efforts.

You can also use the report to track your progress on the road to maximum Z Sort exploitation. It might also be interesting to monitor the percent of sorts that did not use Z Sort because of memory-related limitations. If you see one system that consistently has a higher than average percentage of such sorts, that might help you justify adding memory to that LPAR.

As we discussed previously, some sorts might not use Z Sort because of restrictions that *might* be addressed in future DFSORT releases or PTFs. Should that happen, the summary report could help you decide how much benefit you might get from applying the associated PTF. For example, in the sample report shown in Figure 7 on page 18, there were nearly 8000 sorts that did not use Z Sort because they were invoked by another program, or they were part of a Db2 reorg. If IBM released a PTF that removed this restriction, you could quickly review the summary report to determine if it would be worth accelerating the application of that PTF to your systems.

For a definitive list of the potential reasons for DFSORT not using Z Sort for a given sort, refer to the description of the ICE267I message in the *z/OS DFSORT Messages, Codes and Diagnosis Guide*.

Customizing the Sample Job

If you are interested in tracking the CPU time of one or more jobs before and after they start exploiting Z Sort, you need information from records that have ICEFLBY5 set to 0 (not using Z Sort) *and* ones that have the flag set to 1 (Z Sort *was* used). The sample job doesn't provide such a report, but it should be easy to create one. The fields you would be interested in are probably:

ICEJOBNM	Job name
ICESTPNM	Step name
ICESTN	Step number
ICEBDATE	Date sort step ended
ICEBTIME	Time sort step ended
ICESID	SYSID
ICEFLBY5	Z Sort usage flag
ICEZSRNU	Z Sort non-usage reason code
ICEEXBYS	Number of bytes sorted
ICECPUT	TCB time
ICESRBTS	SRB time at start of sort
ICESRBTE	SRB time at end of sort

While it would be nice to also have the elapsed time, the type 16 records don't contain the job step start time. That information *is* contained in the type 30 records, but in this article we are sticking with just the type 16 records, so it is not available to you.

All these fields are in the Header or Data sections of the type 16 records, so you can use the NZDETRPT job as a model. If you go to the //TOOLIN DD statement, you will find statements similar to the following:

```
SORT FROM(SMFIN) TO(SMFNZ) USING(CTL1)
DISPLAY FROM(SMFNZ) LIST(NZSRTDET)                                  -
TITLE('ZSORT NON-USAGE REPORT') TFIRST DATE  TIME PAGE              -
BLANK                                                               -
HEADER('JOB Name')           ON(ICEJOBNM)                           -
HEADER('STEP Name')          ON(ICESTPNM)                           -
HEADER('STEP Num')           ON(ICESTN)                             -
HEADER('Step End Date')      ON(ICEBDATE,DT1,E'9999-99-99')         -
HEADER('Step End Time')      ON(ICEBTIME,TM1,E'99:99:99')           -
HEADER('SYSID')              ON(ICESID)                             -
```

Each HEADER line defines a column in the report, with the value in the 'ON' statement indicating the SMF field that contains the value you want to place in that column. For example, the job name is in the field called ICEJOBNM (and ICEJOBNM is defined as a DFSORT symbol in the symbol definitions file that you used for the three sample reports). The only field in our list above that is not already included in the NZDETRPT job is the Z Sort flag (ICEFLBY5), so you would add a line like the following at the appropriate point to insert that column into the report:

```
HEADER('ZSORT Flag')         ON(ICEFLBY5)
```

There are probably some columns in the NZDETRPT report that are not really needed if you just want to see the CPU and elapsed times before and after the job starts using Z Sort. To make it easier to add those columns back in at a later stage, you can simply comment them out by adding '**' in the first two positions of those HEADER lines.

There are two other changes we want to make:

- Add an INCLUDE statement specifying the job name that you want to report on.

  You would do this by changing the INCLUDE COND= statement to look like this:

```
INCLUDE COND=(ICERTYP,EQ,16,AND,          # Type 16
              ICEJOBNM,EQ,C'FPK01DFF',AND, # This job only
              ICERSUB2,EQ,ICERSUBF,AND,   # Successful job
              ICEDATA,GT,0,AND,           # Data sect OFFSET > 0
              ICEDATAL,GT,0,AND,          # Data sect LENGTH > 0
              ICEDATAN,GT,0)              # Data sect NUM    > 0
```

- Because we want to see the job before *and* after it used Z Sort, you will need to remove the INCLUDE statement that selects only jobs that did not use Z Sort.

  The INCLUDE statement that we are referring to is the following:

```
INCLUDE=(ICEZSRNU,GT,0),                  # ZSORT usage code > 0
```

  To include records for both sorts that used Z Sort and ones that did not, simply delete that line.

If you want to see the information graphically, you could drop the resulting report into a spreadsheet. You might also use such a report to spot job steps that go back and forth between using Z Sort and not using it. A possible reason for such behavior might be that the system has more available memory at some times than others. If that is the case, you might consider adding a little more memory to that LPAR, so the system nearly always has enough available memory to support the use of Z Sort.

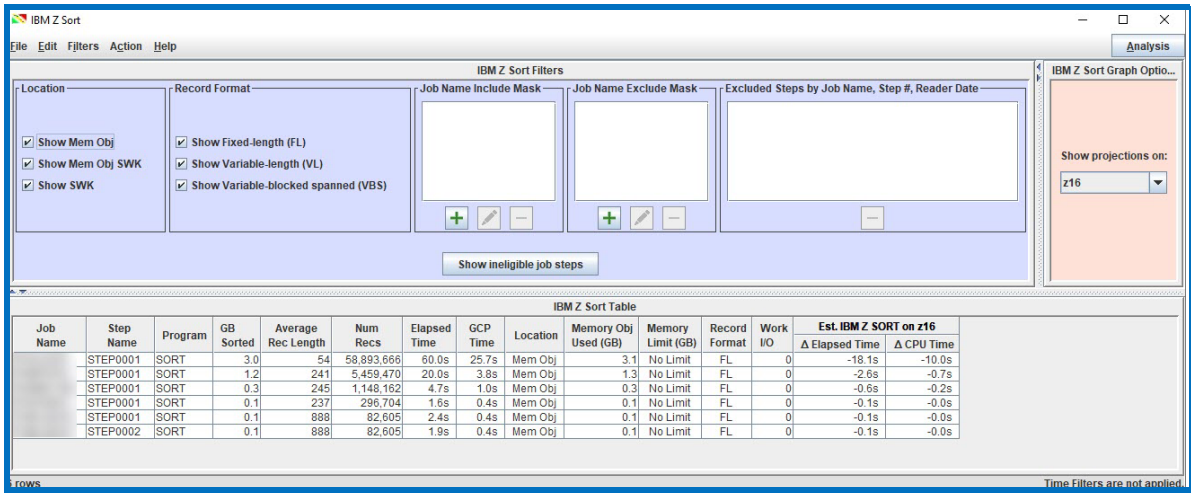
This information certainly doesn't make you a DFSORT expert, but hopefully it will be enough to get you started. If you want to make more fundamental changes to the provided samples, refer to the *z/OS DFSORT Application Programming Guide*.

zBNA Z Sort Support

Support for estimating the benefit of Z Sort was added to zBNA V2.2.4. zBNA has an advantage over the sample programs discussed in this article because zBNA uses data from more than one SMF record type - in particular, it uses information from the type 71 records to identify how much available memory exists on your system. There is also Z Sort-relevant information in the SMF type 30 and type 113 records that can be input to CP3KEXTR.

An advantage of zBNA compared to the ICE267I message is that zBNA is able to inspect *all* the information in the type 16 record. Whereas ICE267I reports the *first* reason it finds for why a given sort can't use Z Sort, zBNA is able to report on *all* the reasons it can find for why a job might not be able to use Z Sort. This can be a big time-saver if you have multiple jobs with multiple reasons for not using Z Sort.

Figure 8 - Sample zBNA tabular report



One usage note - when you load up your Z Sort info into zBNA and select the DFSORT icon, the initial report (as shown in Figure 8) currently gives you a list of sort job steps that could *potentially* use Z Sort. If you have not implemented Z Sort yet, that list is likely to be quite long. However, if you have already enabled Z Sort, that list will only contain the job steps that could potentially use Z Sort, but are not doing so at the moment. We've been looking at this and can't find a way in zBNA to determine why those jobs did not use Z Sort. We even tried clicking on the Show ineligible job steps button and scrolling through the list to find the job steps in question, but those jobs are not included in that list. In the absence of any other way to do it, you could take the information from zBNA and use that to create a NZDETRPT report for just those jobs. It is not ideal, but that is the best we can offer at the moment. But watch this space - you never know what exciting enhancements the IBM CPS tools group have in store for us.

Tip: zBNA actually contains a little more information than it displays on the report (in the interest of avoiding you having to scroll back and forth to see all columns). You can see those additional columns (System ID, Job Step Start date and time, and Job Step End date and time), by clicking on File and then Save CSV.

You can also save the list of Ineligible Job Steps as a CSV if you would like to do filtering, or count the number of jobs steps, or any other manipulation you would like to do with the information provided in that report.

Something to consider is that zBNA must be installed on your PC, and many companies lock down their PCs, making it impossible, or at least very painful, to install additional software.

And while zBNA's graphics are very consumable, the price you pay for that is that it is more work to create the input files needed by zBNA. As we all know, there is no such thing as a free lunch ☺.

There are numerous IBM documents that provide excellent information about how to use zBNA. There are also zBNA labs that can be downloaded so you can play around with zBNA without having to go to the trouble of running your own SMF data through CP3KEXTR. Unfortunately, the current labs were created before the Z Sort support was added, but hopefully this will be addressed in the near future.

For more information about zBNA, refer to the zBNA home page and the zBNA enablement website.

References

If you would like to know more about the topics covered in this article, the following materials might be helpful:

- SHARE Summer 2021, *Making the Most of zSORT* presentation by **Ryan Bouchard** and **Sam Smith**.
- SHARE Winter 2021, *Z Sort on z15 + Large Memory = Big performance!* presentation by **Joe Gentile**.
- SHARE Winter 2021, *Accelerate Your Sorts with DFSORT Exploitation of the Integrated Accelerator for Z Sort*, presentation by **Jeff Suarez**.
- SHARE Summer 2020, *z/OS Large Memory Considerations* presentation by **Dave Betten**.
- IBM video *Putting the New Z SORT Named Favorite into Practice* by **John Burg** and **Joel Moss**. And you can find the corresponding PDF here.
- IBM white paper '*IBM Z Sort and DFSORT Considerations*' by **John Burg** and **Dave Betten**.
- 'Sorting Out Your Sort Performance' article in *Tuning Letter 2021 No. 1*.
- IBM manual *z/OS DFSORT Application Programming Guide*, SC23-6878.
- IBM manual *z/OS DFSORT Installation and Customization*, SC23-6881.
- IBM manual *z/OS DFSORT Tuning Guide*, SC23-6882.

If you have any questions or problems with the sample jobs, use the Feedback button that exists on nearly every DFSORT webpage. This will ensure that your questions are directed to the right person.

Summary

When you consider the price of a modern mainframe, the Sort Accelerator might not quite qualify as ‘something for nothing’, however it is fair to say that it is ‘something for no additional charge’ - if you have a z15 or later CPC, every core has a Sort Accelerator on it. And there is no question that, for a subset of your sorts, Z Sort can deliver real elapsed time and CPU time benefits.

However, the level of benefit you observe will depend to some extent on how much time you are willing to invest in setting it up and maximizing its exploitation. Remember that DFSORT use of Z Sort is *disabled* by default, so the absolute minimum that you must do is update your DFSORT installation parms to enable it.

Beyond that, there might be sorts that potentially could benefit from Z Sort, but are not doing so because of some environmental constraint. If you are aware of those constraints and are willing to invest a little time to address them, you can potentially increase the number of sorts that see reduced CPU and elapsed times as a result of using Z Sort.

We believe the sample reports described in this article, and available on the IBM DFSORT website, will make it significantly easier for you to get the maximum benefit from Z Sort, with the minimal amount of your time. Kolusu’s reports help you understand whether a particular sort job step is using Z Sort or not, and if not, why not. They help you track the exploitation of Z Sort in your systems, and enable you to quickly home in on environmental constraints that are impacting the largest number of Z Sort candidates. We want to thank Kolusu for his outstanding support and enthusiasm - this article would have been a lot less valuable without all the help he kindly gave us.