Basic and Advanced Course

(May be used as self-study material for hands-on training)

2017-05-17

GiAPA_Course_Exercises.docx

Table of contents:

In case of self-study training: Spend at least half an hour on trying to solve the exercise before turning to the answer – whilst you may learn from reading the solutions, you remember much better what you learn from trying out different possibilities!

The data used in these course examples was compiled from collections originating from many different installations. The examples serve well for training purpose, but the data does not "add up" correctly, i.e. summaries of all resources used do not show the correct total for the jobs contained in this collection.

To protect the installations from where the data was copied, most job, user, program and file names were changed. IBM names were left unchanged.

Course prerequisites:

1. Since this course is almost entirely based on hands-on practice, participants must have attended Tutorials 1 – 10 from www.gipapa.com before the class. The remaining tutorials are needed before attending the advanced part at the end of the course.

2. During the course each participant must have access to an IBM System i ("AS/400") server with GiAPA incl. graphics installed. (Only exercises # 20, 21 and 23 requires a valid GiAPA software security code.)

3. Test data library for the GiAPA course must be available on the server:
   a. Restore library GIAPATEST from the save file received.
   b. Call program GIAPATEST/CRTFILES

Add GIAPALIB to your library list, and reach the GiAPA Menu simply by using command GIAPA.

Observe: "GIAPATEST" must be specified as data library during the course.

"Hands-on" exercises in performance analysis using GiAPA

1. Where would you tell the programmer to look for possible speed up of job DCD.TMSFPM?

2. Do you see any optimization potential for jobs LTAMW*, and if so, how much?

3. The Job Name Summary report (GiAPA Menu option 17) shows a job that ran 96 times, using a total of 376.478.330 logical I/Os. In the test data only one of these jobs was kept as an example. The main problem of this very frequently running small job is a rather long elapsed run time. How can these jobs be speeded up with at least 50 %?

4. How can runtime for batch job LTEF430 apparently be cut in half? Illustrate the answer using a pie chart.

5. Which performance collection interval (identified by time of day) showed the highest CPU usage for Job PRD? (Additional question only for installations using M3: What is the descriptive text for the M3 Movex class that was active at that time?)

6. Sometimes use of storage for temporary objects (in WRKSYSSTS) gets unusually high, causing overall paging to increase noticeably. In our test data one job used an impressive 7.596.746 pages of memory (= ~31 GB). Which job was that, when was it and what was running causing so much memory to be allocated? Also find the user program and statement number causing this.

7. User name P59020 complains that his batch job, which normally completes within one hour, sometimes takes 2½ hours. The test data contains such an example of such a delay. What is the cause of the delay?

8. Three completely unrelated questions:
   a. What is the total CPU-time used by all LTAMW* jobs?
   b. If a pool shows heavy paging, can GiAPA tell which job(s) are paging, and if so, how?
   c. Find some basic spec's for the LPAR where this test data was collected: Number of processors, main memory size, disk storage size, and % ASP used.

9. Find performance optimization potential for job V4RPE488 and give estimates for each of the saving possibilities.

10. GiAPA's global "File analysis based on HotSpots" suggests that more than 2 billion I/Os might be saved for one file. Looking at the jobs behind this file, it appears that one job alone needs to be changed to obtain this. Explain how CPU usage may be decreased by around 75 % for the job, which seems to be using 3 hours of CPU time.

11. The TESTDATA shows that the server reached 100 % CPU utilization around 6 a.m. on March 1st, 2016. Which job name used the most resources when that happened, and which user program caused this high CPU usage. Do you see any optimization potential for this job?

12. Job FBEJZT68 only runs 4 minutes, but if used once every half hour, it might be worthwhile looking for optimization potential:
    a. Are records read in blocks or one by one?
    b. For the output file, this job writes several records per logical I/O, i.e. blocking is used to improve performance. Calculate the blocking factor used for output.
    c. The number of reads and writes are the same in this job – what is most time consuming, the reads or the writes?
    d. What might cause this difference in time used between reads and writes?
    e. Which operating system function is used to access the files, and (how) could we make it run faster?

13. Looking at the HotSpot Count Summaries for Active Program/Class we see very many HotSpots for "Retrieve Data Area". A follow up on this reveals that they all origin from one daily running job. How could probably at least 5 hours CPU time be saved here?

14. Which file has 86 % of all reads and 96 % of all updates in job REABHMW82, and what might be done to cut the elapsed run time?

15. How do you assess the possibilities for optimizing job GPFSDX030?

16. Do you see any optimization possibilities for job E044502?

17. Estimate possible savings in % by improving I/O for most read file in job HCEB400.

18. What is using 23 % of the elapsed time in job EGRFRHK46B, and do you see any possibilities for speeding up the job?

19. To avoid GiAPA HotSpot data collection using too much CPU, installation parameters limit the number of threads for which call stack and open file information should be collected, and also the number of levels of call stacks to be kept. A user wants for a special analysis to focus on one JAVA job, where call stack data in 30 levels for the 10 threads using the most CPU should be collected every 5 seconds for half an hour. File usage data is not wanted, and during this special data collection the user accepts not

to get HotSpot data for any other jobs. Explain how that special data collection can be specified (do not start such a collection).

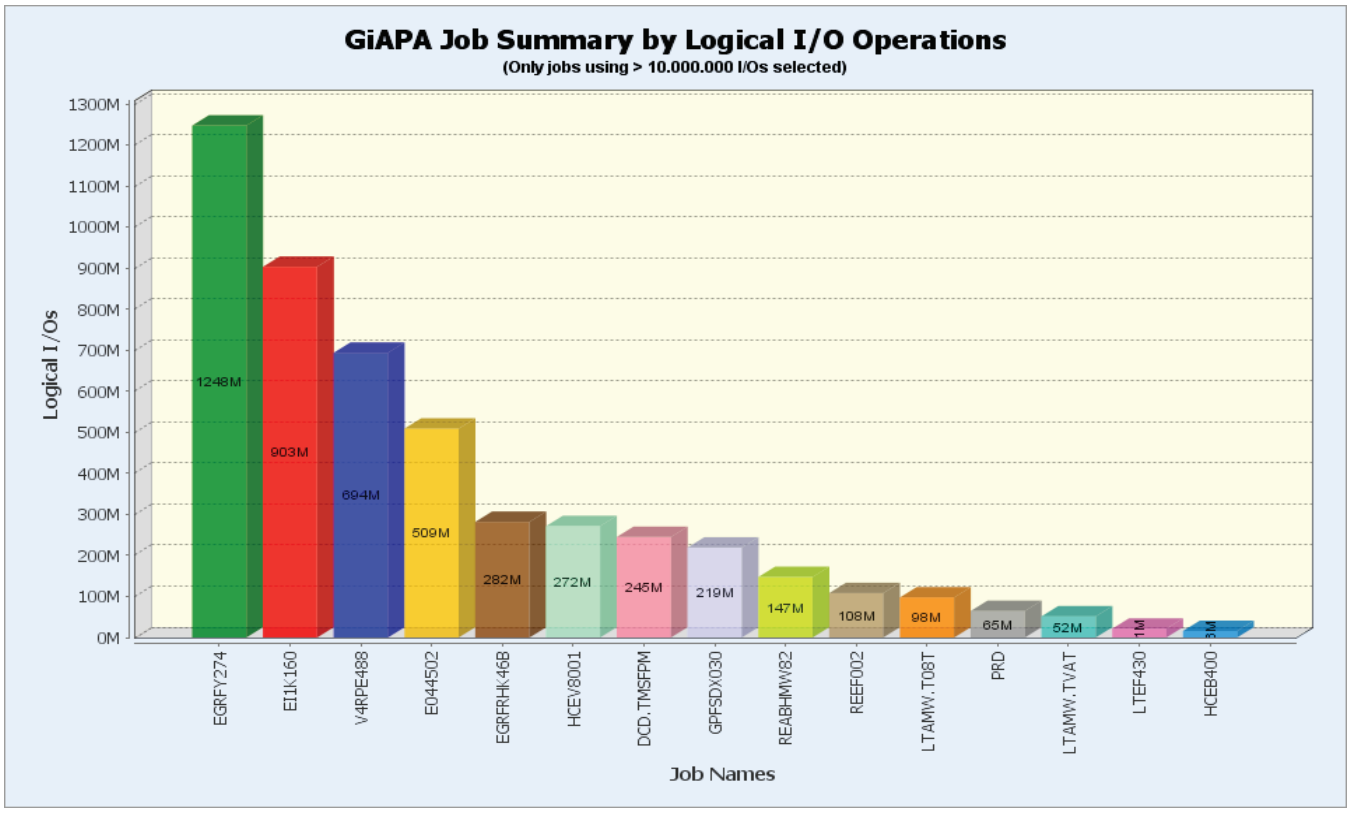
20. GiAPA's "File Check" data collection (Menu option 51) has been used on library MYTESTLIB.

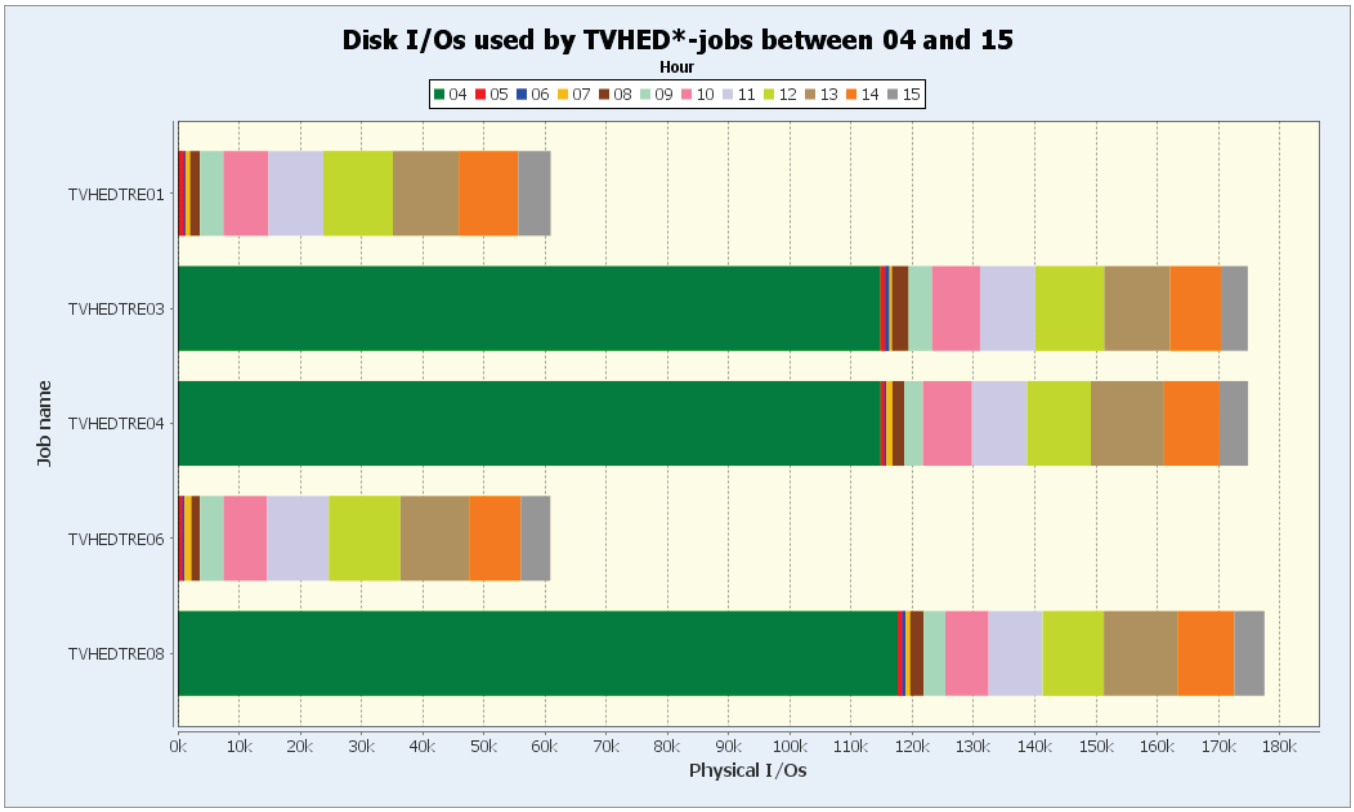
Using this collection you should

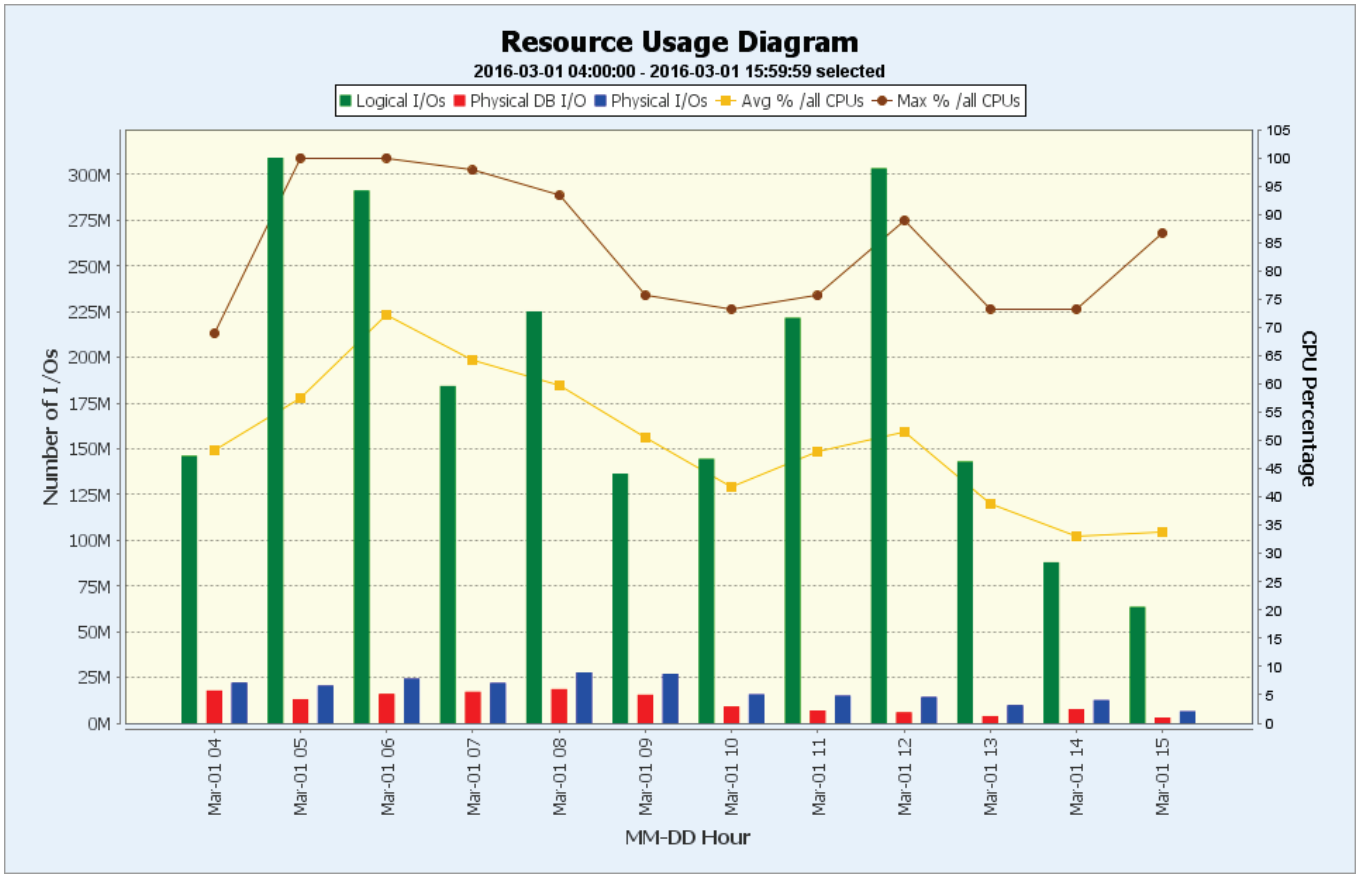
a. Find how many GB of disk space that could be saved by deleting files which have not been used the last two years.
b. Find the names of the two files where a reorganize (removing deleted records) can save a total of one gigabyte of disk storage.
c. Find how many files in MYTESTLIB that have the old index type seizing the entire index during an update of a single entry.
d. Explain how you make GIAPA generate the CL statements needed to
   - run the reorganize, and
   - change the old indexes to the newer type.

21. Data base indexes generated by query functions are not kept, and if the same index is generated repeatedly, resources could be saved by creating a permanent index. Find the file name for which most time was used on index generation, and find the program and statement number calling this repeated incident.

22. Produce the following three charts:




Disk I/Os used by TVHED*-jobs between 04 and 15
Hour
04 05 06 07 08 09 10 11 12 13 14 15
Job name
TVHEDTRE01
TVHEDTRE03
TVHEDTRE04
TVHEDTRE06
TVHEDTRE08
0k 10k 20k 30k 40k 50k 60k 70k 80k 90k 100k 110k 120k 130k 140k 150k 160k 170k 180k
Physical I/Os


Resource Usage Diagram
2016-03-01 04:00:00 - 2016-03-01 15:59:59 selected
Logical I/Os
Physical DB I/O
Physical I/Os
Avg % /all CPUs
Max % /all CPUs
Number of I/Os
0M 25M 50M 75M 100M 125M 150M 175M 200M 225M 250M 275M 300M
CPU Percentage
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 105
Mar-01 04
Mar-01 05
Mar-01 06
Mar-01 07
Mar-01 08
Mar-01 09
Mar-01 10
Mar-01 11
Mar-01 12
Mar-01 13
Mar-01 14
Mar-01 15
MM-DD Hour

23. **ADVANCED / Performance Expert Level Exercise:** Make sure GiAPA data collection is active. From your interactive session run application “Display orders” (= prompt for command GIAPATEST/CUSTORDERS – the Jobq used must not cause waits). Display the orders for different customer numbers between 00001 and 99999. The response time will mostly be OK, but significant delays may occur – your job seems to be held back by something, and while it is in wait state and not using resources, no GiAPA HotSpots will be triggered to tell what the problem is. Therefore you should (while testing CUSTORDERS again) try finding the cause of the delays by using the following GiAPA Menu options for special problem investigations:

    a. 12 GiAPA’s HotSpot watch of your own job once per second

    b. 31–32 IBMs PEX “Statistics” (running under GiAPA) for your own job

    c. 36–37 IBMs PEX “Profile” (running under GiAPA) for program CUSTORDERR

    d. 41–43 IBMs TRCJOB (running under GiAPA) for your own job

Observe: Watching tutorials 15, 16, and 17 is a prerequisite for solving this exercise!

Where would you tell the programmer to look for possible speed up of job DCD.TMSFPM?

Starting from the Job Performance Summary report you position the cursor on the job, which used more than one hour CPU and 244 million logical I/Os. Using F11 we can see that the job ran 4+ hours.

This one is easy, and there seems to be a very large potential for optimization.

Using F9 to get the call stack we see that 46 % of the run time is used on keyed reads, closely followed by 42 % used in user program A1OPPDBW.

Try positioning the cursor on QDBGETKEY (in the "Active program" column or in the red line at the bottom) and use F4, and you will get the text telling the function of the program. Try the same for program A10PPDBW, and you don't get the text – simply because we are not on the customer machine from where this example originates, and therefore the program cannot be found → we cannot retrieve the text.

Let us deal with the user program first, analyzing the call stack (* in input field in top of the screen + "Enter"). We see A1OPPDBW active 265 times, and 28 times further down. The program is obviously called from more than one program (or from more locations within one program), because we see the name more than once, even though the active statement number is not shown.

Hitting F5 adds statement numbers to the active A1OPPDBW program. We now see statement 171 being active in 149 and in 10 samples, and statement 179 was found in 115 and in 18 samples. So we should tell the programmer to check the routine in statements 171 + 179 in program A1OPPDBW.

Hitting F8 once or twice (twice gives summaries on a higher level) we can see that files A1ODQA04 and A1DVPO04 in library R1NPFEDP are opened several times for input only, and use an impressive total of 175 million reads on reading a few hundred records. So maybe we also should suggest the programmer to keep these small files in tables within the program and thereby save 85 % of all the reads.

(In the relatively rare cases where a user program appears very frequently as the last called in the stack, it is quite often possible to re-think that function and improve the performance. One good example is calculations on dates and time stamps to get number of days, e.g. to calculate interest. It may be easy to code, but the machine internal time-of-day clock is based on a 64-bit binary field which increments by 4096 every microsecond and has its starting point at 12:03:06.3148 p.m. on August 23rd, 1928. So just to get todays date and time is not all that cheap. Used once per job, it doesn't matter at all. Used within a loop for all transactions it might well show to be a problem. If interest must be calculated for millions of transaction from some earlier date until today, it is much more efficient to create a small table containing the result of that calculation for all days e.g. the last 2-3 years, and then use a binary table lookup instead of recalculating the number of days for each transaction.)

Do you see any optimization potential for jobs LTAMW*, and if so, how much?

Start with the selection screen for the Job Performance Summary report, enter LTAMW* in the job name selection field, and hit “Enter” to get the report. There are 3 jobs listed, and they seem to behave similarly. Go back to the panel “Selection of Job Summary Report” and hit F9 to get the cumulative Call Stack report for the generic job name. You will see that around 2/3 of the run time is used on unblocked writes and reads.

To analyze how these I/Os are used you should return to the “Selection of Job Summary Report” and once again hit “Enter” to get the “Job Performance Summary” report. Hitting F8 with the cursor on one of the jobs tells that many of the files show around the same number of I/Os as the RRN-span (difference between highest and lowest relative record number seen). Positioning the cursor on one of the input files and hitting F7 will in most cases confirm unblocked sequential read – number of reads and RRN are the same, and a message in the top of the screen tells “Relative record numbers are ascending --> Access seems to be arrival sequence reads only”.

All in all we could probably cut 50+ % of the run time by reading and writing the majority of these files using blocked access. It could be interesting to see in the source code how the reads are made – because when reading files opened for input only, or when writing to output only files, data management will normally automatically use blocked access, if straight forward read and write operations are used.

The Job Name Summary report (GiAPA Menu option 17) shows a job that ran 96 times, using a total of 376.478.330 logical I/Os. In the test data only one of these jobs was kept as an example. The main problem of this very frequently running small job is a rather long elapsed run time. How can these jobs be speeded up with at least 50 %?

Using F9 with the cursor positioned on this job (Job name **HCEB05402**) from the Job Performance Summary Report will show the call stack, where the pink statistics line in the bottom of the screen will show that all resources are used on unblocked reads. Entering * (asterisk) in the input field in the top of the screen to analyze the call stacks + hit “Enter” will sort all call stacks on all levels. Since the result is only one line, we can see that all the reads collected were for one file, read in statement 819 of program WBOD08402. Then we use F7 to look at how the files of this job are moving – it appears that the job indeed only uses one file. We see that the file is opened for I/O, although only read instructions are used. The file is apparently read in arriving sequence only: We see that RRN = Relative Record Number is increasing all the time.

An additional hint: Use F8 to reach the File Analysis report, then position the cursor on the file and hit F7. The same report as before appears, but when called with the cursor on a file, GiAPA selects only that file. This allows GiAPA to check the read sequence and give the informative message “(Relative record numbers are ascending --> Access seems to be arrival sequence reads only)”, displayed in read in the report headings.

To speed up this job there can only be one suggestion: The input reads should be blocked. But blocking is denied by data management because the file is opened for I/O. Data management will not allow blocking for update files, because that would mean that an entire block of records all should be locked. The solution is therefore simply to open the file for input only, since no updates or writes seems to be needed. That would automatically cause data management to use blocking for the file, when the file is read not using keyed access – we can see that is the case, because the call stack showed QDBGETSQ = Data base get sequential unblocked. (If read-by-key had been used, the I/O routine called would have been QDBGETKY.)

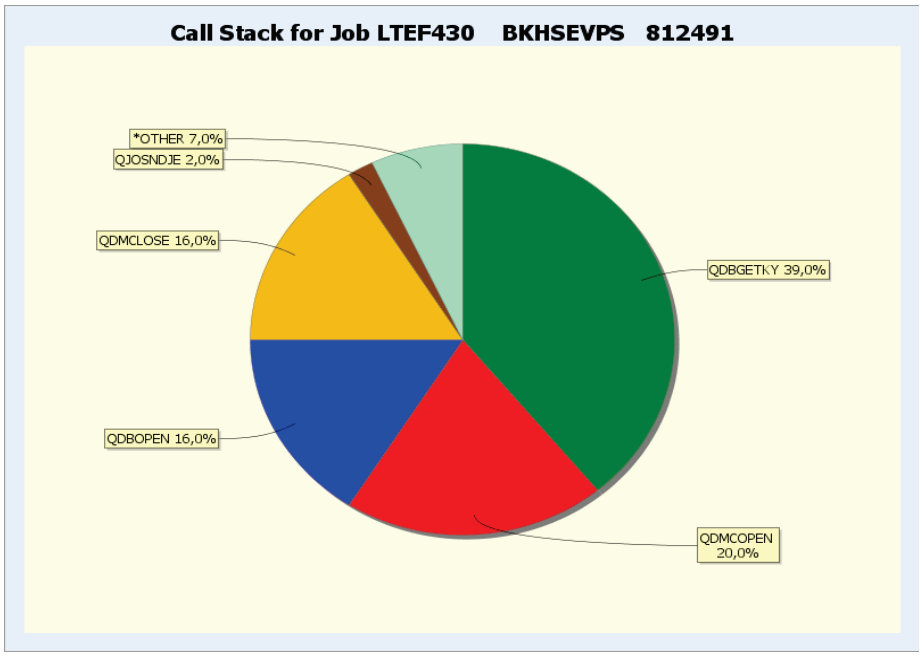
How can runtime for batch job LTEF430 apparently be cut in half? Illustrate the answer using a pie chart.

The question is very easily answered from the Job Performance Summary report by positioning the cursor on the job and hitting F9 to get the Call Stack report.

The bottom pink line containing statistics of active programs shows use of 52 % of the run time on opening and closing files. The job is apparently – maybe for each transaction processed – calling a subprogram which is using a file, but closing the file at each return from the subprogram.

If we from the Call Stack report enter * (asterisk) in the input field in the top of the screen and hit "Enter", we can see that there is only one occurrence of the three open/close routines QDMCOPEN, QDBOPEN and QDMCLOSE, and we can see the name of the program not closing the file: ROON027. Positioning the cursor on these three lines (one by one of course) and hitting F11 will give us the entire call stack. The frequent open/close can also be seen is we use F6, through which we get an overview over the total file usage per interval.

The statement numbers used within ROON027 for open and close can also be seen. For open we see however the rather strange looking statement number 4000004, which we do not see in our source code. This must be a so-called OPM program – we do not have this hurdle in ILE programs. To locate the real source statement number, you must recompile the program using GENOPT(*LIST). This will cause the compilation listing to include the MI-code generated as an intermediate step during program creation. In the MI code you can find the statement number shown in the call stack, and comments in the MI code listing will tell you which file it is and/or which RPG, COBOL or CL code line number generated that MI code.



To produce the chart, simply hit F14 when the call stack report with the pink line at the bottom is displayed. Then use GiAPA Menu option 27 to start GiAPA Graphics in your browser, select the generated file and click on "Show graph".

Which performance collection interval (identified by time of day) showed the highest CPU usage for Job PRD? (Additional question only for installations using M3: What is the descriptive text for the M3 Movex class that was active at that time?)

Use F11 on the "Job Performance Summary" report to see the interval maximum values showed in the 2nd line for the job. The job peaked at 91,3 % CPU.

Using F10 with the cursor on the job you can see the intervals for the job (only including intervals having a higher CPU usage than the limit specified in the Installation parameters, GiAPA Menu option 78). You could page down and look for the interval having the peak value, but a more convenient way would be to use F22 (= "Upper case" + F10) through which you get a "Work with spooled files" display showing the printed version of the report and allowing "find", so you could simply search for the constant 91,3. The answer to the first question appears to be the interval ending at 20:49:45.

Now use F9 to get call stack analysis. The installation parameters caused in this case only stacks for two threads to be collected. Use C to request call stack list in chronological order, and page down to 20:49:45.

The second thread had no call stacks retrieved for that point of time, but the first shows an entry received at 20:29:46.

If you have installed M3 Movex software: Position the cursor on the class name (MMS200MI) and use F4, and you will be told that the text was "Movex item. Open Toolbox ".

You get the text although this class probably is not found on the server used to run these course examples. The Movex classes are not shipped with texts anyway, but GiAPA development has been allowed to receive and store the description for 2000+ M3 classes.

Sometimes use of storage for temporary objects (in WRKSYSSTS) gets unusually high, causing overall paging to increase noticeably. In our test data one job used an impressive 7.596.746 pages of memory (= ~31 GB). Which job was that, when was it and what did the job run that caused so much memory to be allocated? Also find the user program and statement number causing this.

Use sort criteria 16 = “Maximum pages used” on the “Selection of Job Summary Report Panel” (GiAPA Menu option 15). This will show the report sorted descending by maximum usage of memory.

The “which job” question is then already answered. Now Position the cursor on that job (**ESDIFWF**) and hit F10. That will result in the details per interval, where “Pages Used” can be found in the 3rd last column. We could then page down until we found it.

But there might be an easier way: Go back to the Job Performance Summary, keep the cursor on the job, but use F22 instead of F10 to get the report. In this way you will get a “Display Spooled File” showing the report, and here you have a search function. A search for the value 7596746 will immediately position you to the interval at 11:34:30, where the memory usage peaked.

Now return to Job Performance Summary and hit F9 to get the call stack. Roll down see what the job was doing in the minutes leading up to the time wanted, 11:34. The answer is there: A query was running, apparently sorting a lot of data.

And should we so want, it is as usual very easy to see from where the query was called. Position the cursor on one of the call stack lines before 11:34 and hit F11. The query or SQL was called in source code line 407 of program FLALXH4.

User name P59020 complains that his batch job, which normally completes within one hour, sometimes takes 2½ hours. The test data contains such an example of very long run time. What is the cause of the delay?

Enter the user name in the selection panel for the Job Performance Summary report (GiAPA Menu option 15). This will show the name of the job, **EHYCRPKDY**. Use F11 to see the total elapsed run time. The user complains that the job took 2½ hours that day – that does not agree with the run time shown!?

But – the second last column has the heading “Transaction or Job Queue Time”. Transaction time is shown for interactive jobs. For batch jobs the column is used to show the time the job has been waiting on the job queue – here more than one hour.

Use F9 on the job to see the call stack, which shows the remaining part of the delay. The job has very frequently been in record lock wait – some other job was apparently disturbing this run.

Enter * in the input field on the call stack report and hit enter, then use F5 to show status and statement number, and you see which programs and statements had the lock waits, after which it should be easy for the programmer to determine which file(s) are involved.

Observe that the longest job queue wait times and interactive jobs response times can be seen by selecting sort sequence 11 for the “Job Performance Summary” report. If you only want to see either job queue time or response time on that report, enter “B” or “I” in the job type selection field.

Three completely unrelated questions:

a. **What is the total CPU-time used by all jobs names prefixed by LTAMW* ?**

b. **If a pool shows heavy paging, can GiAPA tell which job(s) are paging, and if so, how?**

c. **Find some basic spec's for this LPAR: Number of processors, main memory size, disk storage size, and % ASP used.**

a. Select GiAPA Menu option 16, specify selection on generic job name LTAMW.T*, and select output type T for Totals only. Then use selection 1, 2 or 3 in front of the member name – since we do not select on time, and since none of these jobs were active when the data collection started, the three selections will in this case give the same result.

b. The easiest way to find the answer is to look in the GiAPA manual for the word "paging" – either in the index, if you have a printed copy, or simply use a "find" if you have the manual available on-line. The section on non-permanent writes on manual page 11 mentions paging, and on the top of the next page you can see which report can be used to see names of jobs paging.

c. GiAPA Menu option 15 "Selection of Job Summary report" allows F4=Member prompt. From the list of members use selection 5=Display analysis and expansion statistics – that panel will in the rightmost column amongst others also show the values wanted here.

Find performance optimization potential for job V4RPE488 and give estimates for each of the saving possibilities.

Hit F11 on the Job Performance Summary report. We see that the job ran for exactly 5 hours, and that we have 1184 HotSpots. The data collection interval of 15 seconds times 1184 is 4 hours and 56 minutes, i.e. we have HotSpots for almost the entire job – we can follow everything that happened – not surprising with an average CPU usage of 37 %.

Now use the cursor to select the job and hit F9 to get the call stack. We see that more than half the run time is used on keyed reads. But also the next two entries on the pink line are interesting: The user program A2SRF599 uses 17 % of the elapsed time – that is unusual. And 16 % of the elapsed time was used sending status messages (which cannot be seen in a batch job) – what is going on here?

Let us handle the QMHSNSTA first: OPM program code reports “No record found” condition back to the I/O routine of keyed reads through using an internal kind of status messages. We can prove this by analyzing the call stack (* in the input field, hit “Enter”). You will see that program A2SRF599 called the I/O routine in MI-statement 80B. The line showing 15 QMHSNSTA also points back to 80B within A2SRF599 – it is caused by the same source statement. And we can pair the other send-status-message lines with the other read-by-keys. So what is the solution here: Convert the program to ILE – that should be easy – then that problem disappears, we saved 16 % run time.

Now use F5 to get statement numbers even for the active programs. F5 is a flip/flop turning statement number on/off for the active program. We can see that the statistics change somewhat – for analyzing an active user program we want to see the statement number. We can ask the programmer to check code line 6C7 of program A2SRF599 in library A2TSCPGM, because it was active in 195 out of 1184 call stack samples = 16,4 % of the job run time.

We should also address all the keyed reads shown when hitting F8 once or twice (Twice gives the summary with only one line per file). The input file number 5 used 250.792.549 I/Os to read <= 1665 records, and file number 4 used 40 million reads on 36 records. You can verify this by positioning the cursor on the file and use F7. These files are no doubt accessed with read-by-key operations. It seems we could save around 45 % QDBGETKY by keeping the few records read frequently from these two files in a table in memory.

Details how to do that can be found in the GiAPA tutorial number 14 on www.giapa.com .

In addition it appears that file number 2 is accessed in arrival sequence (use F7 on the file to verify this), but the 250 million reads are unblocked. Only 9 % of the run time is used by QDBGETSEQ, but if this is used on that file, most of the 9 % could be saved just by forcing blocked access – absolutely worthwhile doing.

One more question: If that file (MMS2RDET) really is read sequentially from beginning to end, what may then explain that the I/O-count does not increase at the same pace as the relative record number? (Hint: The probable answer is very simple.)

GiAPA's global "File analysis based on HotSpots" suggests that more than 2 billion I/Os might be saved for one file. Looking at the jobs behind this file, it appears that one job alone needs to be changed to obtain this. Explain how CPU usage may be decreased by around 75 % for the job, which seems to be using 3 hours of CPU time.

The report generated by GiAPA Menu option 22, selection 1, includes a column with the heading "Estimated Record Reuse", which shows the number of I/Os that maybe could be saved. Positioning the cursor on a file and hitting F10 shows the jobs that made the many I/Os.

Job **EGRFY274** clearly uses the CPU time on the many logical I/Os – the Job Performance Summary report says 1.247.503.513, but the asterisk after the start time indicates that the job started before this data collection. Using F8 to get the File Analysis report tells that the I/O total for the job is much higher – the total number of reads for just one file reaches 2.147.483.647 (Remember that this TESTDATA only contains data for (part of) one day). Since the job on the day shown used 1 hour 33 minutes CPU, the total CPU usage must have been around 3 hours.

Observe that the **PERFORMANCE DATA** received every 15 seconds from the performance collector API and reported on the Job Performance Summary report gives the exact number of I/Os for the job within the time reported. The **HOTSPOT DATA** shows the values received from the API corresponding to what can be seen using DSPJOB option 14 = Opened files, showing the accumulated number of I/Os from when the file was opened. If the job was running since yesterday, the HotSpot accumulated values since job start might show higher numbers than the statistics for the job on the day shown by the Job Performance Summary report.

The File Analysis report also indicates that file only contains around 4½ million records, so the obvious suggestion is to read those into memory and keep them in a table – savings would be most significant. The object type "User Index" would be suited for this purpose.

Details on how to replace a file access with a user index can be found in the GiAPA tutorial number 14 on www.giapa.com, and example source code is available within GIAPALIB in source file GIAPAEXAMP.

If we use F9 to look at the call stack, our assumption is confirmed: The job uses 98 % of the run time on reading.

This file is apparently positioned by key (RPG SETLL or COBOL START) and then read sequentially over and over again. If another job has made updates to the file in between each read, all these reads could make some sense – but that is probably not the case.

Very strange is also the fact that the call stack shows the rather inefficient unblocked read being used. If the file is read sequentially, i.e. not read by key, why is blocked access not used? Just that change would maybe cut the run time in half.

The TESTDATA shows that the server reached 100 % CPU utilization around 6 a.m. on March 1st, 2016. Which job name used the most resources when that happened, and which user program caused this high CPU usage? Do you see any optimization potential for this job?

Select GiAPA Menu option 21 (Collection interval summaries). You could request the report just by entering a "1" in front of the TESTDATA member and then page down to 6 a.m. But it is probably more convenient to limit the report by specifying a selection criterion. In this case we have two possibilities: We could use

- date/time selection to specify a start time of 160301 055800 (YYMMDD hhmmss)
- "Select only if CPU % exceeds" to specify that we only want intervals using more than e.g. 95 % CPU.

(Try both possibilities, just to see how it works!)

You will see that the 100 % CPU utilization occurred in the collection interval ending at 06:02:15.

Now hit F7 to see the jobs using most CPU at that point of time. The job using most CPU was **LTAMW.T08T**.

Copy the job name, return to GiAPA Menu option 15 ("Selection of Job Summary Report") and copy the job name into the job name selection field in the middle of the screen. Hit "Enter", and the job is shown.

To find which user program was active at 06:02:15 use F9 to get the call stack. You may now roll all the way down to 06:02:15, but there is an easier way. Use F21 to get a time selection window, enter 060000 in the first selection field and hit Enter.

The data will now be positioned so you can see that the job at the wanted time was reading data (QDBGETSQ), and the user program doing that was DADWDA (shown in column "Non-Q Program")

But looking at this job anyway we couldn't resist the temptation to look for optimization potential. The pink line in the bottom of the call stack report tells that the job in 36 % of the HotSpots was doing unblocked writes. Looking at the files being written we see that they apparently are "new" files, i.e. starting empty – it is not the question of adding more data to files already containing records. In that case we should force blocking and save the lion's share of the 36 %. Blocking is probably denied by the operating system because the file has keyed access path(s) defined. When writing to an empty file it would be quicker to deactivate such indexes, use blocked writing, and then generate the indexes afterwards.

Job FBEJZT68 only runs 4 minutes, but if used once every half hour, it might be worthwhile looking for optimization potential:

a. **Are records read in blocks or one by one?**
b. **For the output file, this job writes several records per logical I/O, i.e. blocking is used to improve performance. Calculate the blocking factor used for output.**
c. **The number of reads and writes are the same in this job – what is most time consuming, the reads or the writes?**
d. **What might cause this difference between reads and writes?**
e. **Which operating system function is used to access the files, and (how) could we make it run faster?**

a. Set the cursor on the job on the Job Summary Report and hit F7 to get the file statistics per file and interval. You can see that the value in the column for number of reads is exactly the half of the RRN (Relative Record Number), i.e. records are transferred in blocks of 2 records from data base management to the program work area. You could also simply use F9 to get the call stack, where you can see, that reads are made using QDBGETM = DATA BASE MULTIPLE GET (Blocked).

b. The output file obviously already contained over 16 million records when this job started. To calculate the output blocking factor you subtract the highest value for both writes and RRN from the lowest, and divide the last result by the first: (18.020.610 – 16.904.072) / (581,893 - 23,624) = 2.

c. The pink statistics in the bottom of the call stack display (F9) clearly shows that reads are the most time consuming.

d. The disks must have disk cache. When reading, we need to locate the records on the disks – that takes a little time. When writing, we just send the data to the cache, and the actual writing is then made later asynchronously.

e. CPYF (CL-command copy file) is used here. You may see this by hitting F4 after positioning the cursor on the QCPGENIO program shown in the call stacks. A simple program to read and write (i.e., copy the records) could be defined to use a much higher blocking factor.

Looking at the HotSpot Count Summaries for Active Program/Class we see very many HotSpots for "Retrieve Data Area". A follow up on this reveals that they all origin from one daily running job. How could probably at least 5 hours CPU time be saved here?

GiAPA Menu option 18, selection 5, results in a report, where we see the wanted information in the third line. Hitting F6 with the cursor positioned on that line will save the program or class name.

Now turn to the "Selection of Job Performance Report" (Menu option 15) and enter "*YES" for "Only show jobs using selected object?" before hitting enter. This will cause selection of the job(s) that used the program selected – here only one job, **EI1K160**.

F11 on the Job Performance Summary report shows that the job is active throughout the day, peaking at 69,8 % CPU, but on average using 34 % CPU, probably mainly on the 903 million logical I/Os. Any significant optimization here could have a major impact!

Using F9 with the cursor on the job to get the call stack we see that 45 % of the run time is used by program QCLRTVDA. If we do not know what this is, position the cursor on the program name and use F4 to see the program text in the message line in the bottom of the screen.

Data areas are convenient for many purposes, but should never be used e.g. for each transaction within a batch job, which probably is the case here. All data area updates are written to disk, and data areas are fetched from disk – not a problem if fetched only a few times, but adding up rapidly if used continuously.

We can from this call stack screen, or after having analyzed the call stack (asterisk * in input field and "Enter") by positioning the cursor on the line and hitting F11 see the entire call stack, i.e. see the program and statement number(s) using the QCLRTVDA command.

Instead of intensive use of a data area we can recommend a space in memory, e.g. a user space. Details on how to replace a data area with a user space can be found in the GiAPA tutorial number 14 on www.giapa.com, and example source code is available within GIAPALIB in source file GIAPAEXAMP.

The pink line of call stack statistics also showed 7 % QMHSNSTA (Send Status Message), which are used to signalize "Record not found" back to "Read-by-key" instructions. These messages could be avoided simply by converting the program from OPM to ILE.

Using F8 on the job reveals a very unusual access of file MYCAB41 in library ANVAMY1. The file is written and read very many times, in some cases opened for I/O (which prohibits blocked access) although no updates are appearing. Of course we could be wrong, but our guess is that this rather small file with only around 4 million records ought to be kept within the program work area, in a user space, or in a user index (See Tutorial 14 on www.giapa.com + source code examples in GIAPALIB).

If these three changes are possible, the total savings would probably be around 6 hours CPU time per day.

Which file has 89 % of all reads and 96 % of all updates in job REABHMW82, and what might be done to cut the elapsed run time?

Hit F8 twice with the cursor on the Job from the Job Performance Summary report. The File Analysis Summary will show that file ROBP001 was opened several times, and adding the statistics for this file together (use F8 again) it is clear this file had the vast majority of reads and updates.

The heavy use (64 %) of get sequential unblocked could indicate that sorting this one file into the sequence in which it is accessed might open up for some performance improvement – the job is active 3 hours with an average CPU percentage of only 16, indicating that most of the time is spent awaiting I/Os. Sorting less than 30 million records does not take very long time, and since no new records seems to be written to the file, one sort should be sufficient.

Sorting a file before such a larger batch run is a good example of the fact that performance sometimes can be improved without touching the application program code.

File number 53 is opened for I/O, but shows only writes. The writes must therefore be unblocked (confirmed with use of F7). Since the records are written to an empty file, blocked writings would save most of the time used here. (The very same file is used similarly, but with fewer I/Os, both earlier and later in the job.)

How do you assess the possibilities for optimizing job GPFSDX030?

Looking at the call stack does not tell very much here, but using F8 twice to reach the File Analysis Summary report shows very strange statistics for the first many files, where the RRN Span column shows the value 1, indicating that the same record was accessed all the time.

Not having seen the source code we have no explanation for this, but also here we would like to have a look at the source code. But it seems we could save 120 million reads by loading file BNE50404 to a user index and access it from there.

Whilst converting a very frequently read file to a user index often might be rather rewarding, there are quite a few cases where a much simpler change can obtain the majority of the optimization obtainable. If a few records often are re-read by key over and over again, the solution could be just to add the code lines shown in red below around the repeatedly used read statement:

```
IF REBATE-CODE NOT = OLD-REBAT-CODE
     READ  PARAMETER-FILE KEY IS REBATE-CODE
     MOVE  REBATE-CODE TO OLD-REBATE-CODE
ENDIF
```

Do you see any optimization possibilities for job E044502?

Using F9 to look at the call stack of the job tells us that 80 % of the time is used on I/Os – that is pretty normal.

F6 = ODP Overview report tells us that the job seems to start more or less from scratch every 5 minutes. The number of files opened is 12 when the job is active, but goes down to 0 in between each active phase. Something else is looking strange: Only reads are shown here – shouldn't there be a result written somewhere?

F10 = Details for Job report confirms that there are mainly reads – but paging down we do see a very few writes from time to time.

F8 = File Analysis shows that we are reading the same files repeatedly – not unexpected by now.

F7 = File statistics of course confirms the picture, and we can see that all the files apparently are read randomly, i.e. not in arrival sequence. We saw never the less QDBGETSQ being used – it could spell positioning the file and then reading a number of records in arriving sequence.

The bottom line is that here we cannot promise any substantial optimization, but then again – maybe anyway. The sort program is very fast. Sorting at least some of these files in a sequence so they could be read sequentially might be very worthwhile – maybe several of them are not updated in between each 5 minutes re-start, so we only need to sort them once.

Obviously there must be some new data – some new records in a file which then serves as the main transaction file driving the run. Maybe that file should be sorted into the sequence of (some of) the other files.

The total number of I/Os is so low within each "round" of this job that a pure sequential blocked access would optimize the job very much – we need to consult the responsible programmer to check what is possible.

Here we have so many files that it is difficult to get an overview over what is going on. But if we hit F8 a third time, we get the summary by file name for files responsible for more than 2 % of the I/Os. We see that three of these files were read around 70 million times each, and one file showed a total of 200 million reads. If we were running on the server where this data was collected, F4 on the file name from one of the other reports would show us not only the text for the file, but also the actual number of records in the file.

With this knowledge and a chat with the programmer we might very well be able to find ways of improving the performance. The name of the game is to cut the number of I/Os, most important the physical I/Os which incur waits for completion of disk access. The logical I/Os consume "only" CPU time, but if we here could make a block of records being moved to the internal program buffer instead of receiving the records one by one, substantial CPU savings can be obtained.

Estimate possible savings in % by improving I/O for most read file in job HCEB400.

To find the most read file: Use option 15 from the GiAPA Menu to reach the Job Performance Summary report, and then hit F8 twice after positioning the cursor on the job. This will display the File Analysis Summary report listing the files in descending order by total number of I/Os. The highest number of reads is found in the 3rd line showing data for file number 30.

Position the cursor on that file and use F7 to get the File Statistics report for that file only. Her you can see that

- the file is opened for I/O (which means that data management will not allow blocked access),
- there are never the less only reads – no writes or updates
- the relative record number is increasing all the time, meaning that the file is read in "arrival sequence", and
- the file is read in the seven minutes between 00:43:45 and 00:50:30.

Use F3 to return to the Job Summary Report, and then use F11 to get the second line for the job showing also the job end time. The job was active 47 minutes, which means that even if the above mentioned file number 30 was changed to input only to speed up the reading, there could only be a rather marginal improvement – maybe 5-7 % of the total job run time.

Using F9 to look at the call stack we can see that there were longer periods where call stack was not collected, e.g. between 00:16 and 00:33. We can now use F21 to obtain a window allowing us to limit the call stack analysis to only show the time interval 00:43:45 and 00:50:30. Doing so will show that sequential read indeed was active in 100 % of the samples – and by entering an * (asterisk) in the input field in the second title line and hitting Enter we see that all samples came from the alike call stacks → the same file is being read. Positioning the cursor on that line and using F11 we can see the program name (WBOR241A) and statement number (2277) doing the reads.

If we hit F10 to get the details per interval we can see the overall behavior of the job. Note that for the columns showing I/Os we have the reads in the first line in green color, the writes are shown in the second line. In the time period from which we had no call stacks the job was only reading all the time, but it was using so little CPU (between 2 and 5 %) that no GiAPA HotSpots were triggered – therefore we did not get any call stacks. The low CPU % is caused by the job having to await completion of the rather many physical reads = disk accesses.

But the high numbers of reads during this period probably mean that we cannot correctly answer the question – odds are that the file(s) being read between 00:16 and 00:32 have more I/Os than any file we could see. So if we wanted to improve performance of this job, we would probably have to use GiAPA Menu option 12 (Job Watch) to request HotSpots for the job continuously, so we could get the complete picture.

What is using 23 % of the elapsed time in job EGRFRHK46B, and do you see any possibilities for speeding up the job?

The first question is easily answered:

From the Job Performance Summary report, position the cursor on the job and use F9 to see the call stack. The answer can be seen in the pink line. To see WHAT this program is doing, set the cursor on the program name and use F4 = Prompt for name.

Possibilities for speeding up are not easily detected here, but hitting F8 twice to reach the File Analysis Summary may give some hints:

13 million reads are used to access 3243 records – here we might be able to save something by keeping this file in a table in memory.

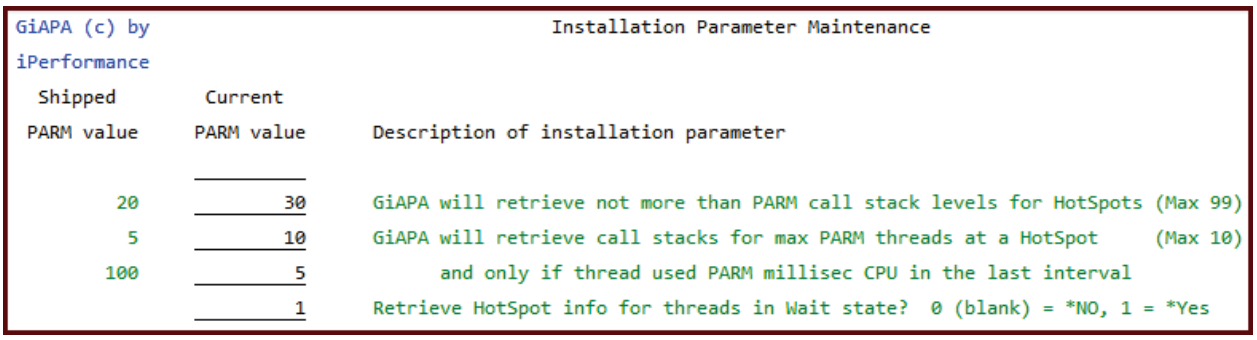
More interesting is maybe all the files that show very few reads and writes, but around 6,3 million "Other I/Os". Other I/Os are normally updates, but since the records are not read, they cannot be updated either, so it must be one of the other possibilities. What also is a surprise is the fact that the column for "RRN Span (High-Low)" shows only 1 for these files (file numbers 3, 7, 26, 25, 38 and 37).

Position the cursor on one of these files and hit F7, and you can see how the count for other I/Os (Update, delete, unlock, etc.) is increasing, while the relative record number stays the same.

We at iPerformance do not have the explanation here – we have not had the opportunity to analyze the job. But it certainly looks as if it would be interesting to have a look at the source code and talk to the programmer. It looks as if something is not right here, and we just might save a lot of time by correcting whatever it is.

To avoid GiAPA HotSpot data collection using too much CPU, installation parameters limit the number of threads for which call stack and open file information should be collected, and also the number of levels of call stacks to be kept. A user wants for a special analysis to focus on one JAVA job, where call stack data in 30 levels for the 10 threads using the most CPU should be collected every 5 seconds for half an hour. File usage data is not wanted, and during this special data collection the user accepts not to get HotSpot data for any other jobs. Explain how that special data collection can be specified (do not start such a collection).

Use GiAPA Menu option 88 to temporary modify the installation parameters:



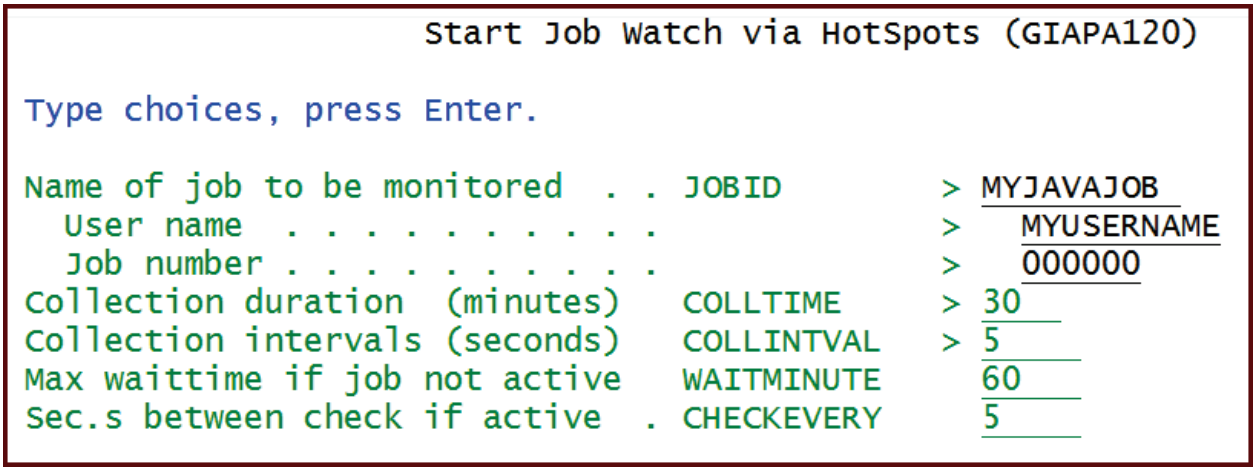
You may or may not want to reduce the standard value of the second last parameter from the default (= shipped) value of 100 – here we decided to try with only 5 milliseconds.

Then start a data collection specifying the maximum possible CPU-limits for triggering HotSpots, and specify that only call stacks should be collected:

GIAPA110 CPULIMIT(29.9) HOTSPOTS(*CALLSTACKS)

CPU-limit cannot be set to a higher value that 29,9, so call stacks may be collected in a few cases for other jobs than the job targeted here, but that shouldn't become a major problem.

Finally use Menu option 12 (= command GiAPA120) to specify HotSpot collection for the wanted JAVA job:



GiAPA's "File Check" data collection (Menu option 51) has been used on library MYTESTLIB.

Using this collection you should

a. **Find how many GB of disk space that could be saved by deleting files which have not been used the last two years.**
b. **Find the names of the two files where a reorganize (removing deleted records) can save a total of one gigabyte of disk storage.**
c. **Find how many files in MYTESTLIB that have the old index type seizing the entire index during an update of a single entry**
d. **Explain how you make GIAPA generate the CL statements needed to**
    - **run the reorganize, and**
    - **change the old indexes to the newer type.**

Use GiAPA Menu option 52 to get the panel where reports on data base object descriptions can be generated when option 51 has been used for the wanted library or libraries.

a. Enter a 1=Select in front of "Files not used recently", and change the number of days to 730. Then type 1 in front of member name MYTESTLIB, and hit enter. Go to the bottom of the report then produced. (It is very common to see much larger saving potentials the first time new GiAPA customers run these reports!)

b. Enter a 1=Select in front of "Reorganize candidates", type 1 in front of member name MYTESTLIB, and hit enter. (Hint: all three reports can of course be selected at the same time, and one or more members may be selected.)

c. Enter a 1=Select in front of "Files having the old *MAX4GB index type", type 1 in front of member name MYTESTLIB, and hit enter.

d. This is the easiest part – GiAPA did this automatically when you selected the reports! Use the GiAPA Manual to see where you can find the results, stored in a source file ready to be compiled.

(Observe that the CL source statements will be overwritten the next time the report runs.)

Data base indexes generated by query functions are not kept, and if the same index is generated repeatedly, resources could be saved by creating a permanent index. Find the file name for which most time was used on index generation, and find the program and statement number calling this repeated incident.

GiAPA Menu option 53, selection 2, lists index generation statistics per file name, taken from the HotSpot data for call stacks. During index generation, the “Function” column of e.g. WRKACTJOB will contain “IDX-*FILENAME*”, so whenever index generations lasts sufficiently long time to trigger HotSpots, GiAPA can see for which physical file the index was created.

The file name here was F1VCPFDL, which that day had an index generated three times, each time lasting around 3½ minutes. It is also shown when the index generation took place, and what the job name, user name, and job number were.

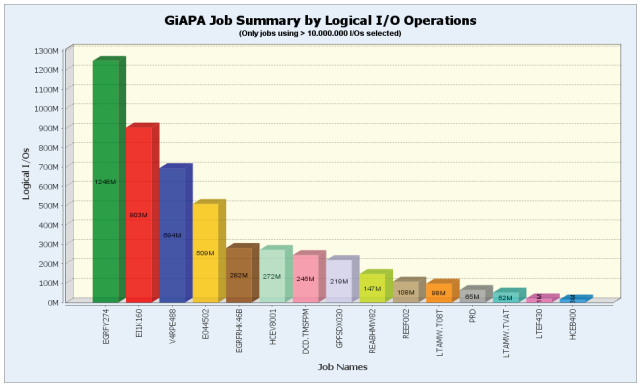
Even the names of the active program and of the last user program in the call stack are shown. NNFV055 called QQQIMPLE, the query implementor.

Turning to the Job Summary report, and selecting F9=Call stack with the cursor on one of these jobs, we can find the call stacks in question – or more precisely, we can see that no call stacks were available for retrieval, quite as normal during index generations.

But when an index is generated in order to be used by a Query, by an OPNQRYF command (Open Query File), or by an SQL function then we normally also get a call stack retrieved immediately after the index generation, and this shows us what we were looking for. Statement 235 of program NNFV055 called QCMDEXEC, which probably in turn was used to run an OPNQRYF – but that can be checked in the source code.

If it had been a Query/400 running then we cannot see the name of the query from the call stack. But also here GiAPA can assist. Menu option 64 can be used to start tracking of use of queries, and with that active we can at any time see which users are using which queries – valuable for cleaning away old queries not being used any more, and for seeing how often each query is called, this allowing us to know when to generate indexes used by frequently running queries.

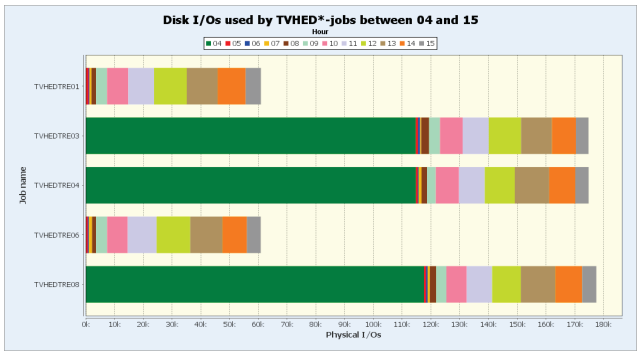
Produce the following three charts:



Request a “Job Performance Summary” report sorted on logical I/Os, and specify selection by *USRPGM “GIAPANOGRP” to avoid the group totals for *BATCH, etc. On the report position the cursor on the last job exceeding 10.000.000 logical I/Os and hit F14.

Use GiAPA Menu option 27 to start GiAPA Graphics in your browser. Select the generated graphics input file, and modify the specifications to use chart type “Column 3D”, to not show decimals, and add the text for the secondary title line. Then click on “Show graph”.

- - - - - - - - - -



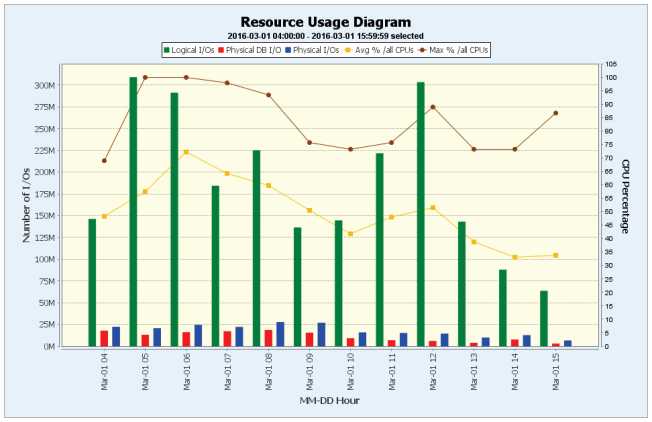
Use Menu option 26, select member TESTDATA, specify a chart name, enter the chart title, use JOB as key field 1 and HOUR as key field 2, select jobs with generic job name TVHED* and time in the range from 04 to 15 hours.

Then specify an “S” for stacked chart for “Total Disk I/Os” – the S will cause the contents of the last key field (HOUR) to be used as identifying text for the values of the stacked bar chart.

To better understand this, try exporting the data to Excel when S is used, and then also Export the result when specifying 1 instead of the S.

Add the texts “Hour”, “Job Name” and “Physical I/Os” manually before displaying the chart.

- - - - - - - - - -



From Menu option 21 select From and To as 160301 040000 and 160301 155959.

Specify 3=Graphics for member TESTDATA.

Hit F11 to request this popular “standard” combined column-and-bar chart. Note that the line data (here CPU) automatically is selected last, i.e. with a higher sequence number.

The two Y-axes texts were modified to include “of I/Os” and “CPU” in addition to the default texts.

(Please note that command GIAPA052 can produce the same diagram for the entire day.)

ADVANCED / Performance Expert Level Exercise: Make sure GiAPA data collection is active. From your interactive session run application "Display orders" (= prompt for command GIAPATEST/CUSTORDERS – the Jobq used must not cause waits). Display the orders for different customer numbers between 00001 and 99999. The response time will mostly be OK, but significant delays may occur – your job seems to be held back by something, and while it is in wait state and not using resources, no GiAPA HotSpots will be triggered to tell what the problem is.

Therefore you should (while testing CUSTORDERS again) try finding the cause of the delays by using the following GiAPA Menu options for special problem investigations:

a. **12 GiAPA's HotSpot watch of your own job once per second**
b. **31–32 IBMs PEX "Statistics" (running under GiAPA) for your own job**
c. **36–37 IBMs PEX "Profile" (running under GiAPA) for program CUSTORDERR**
d. **41–43 IBMs TRCJOB (running under GiAPA) for your own job**

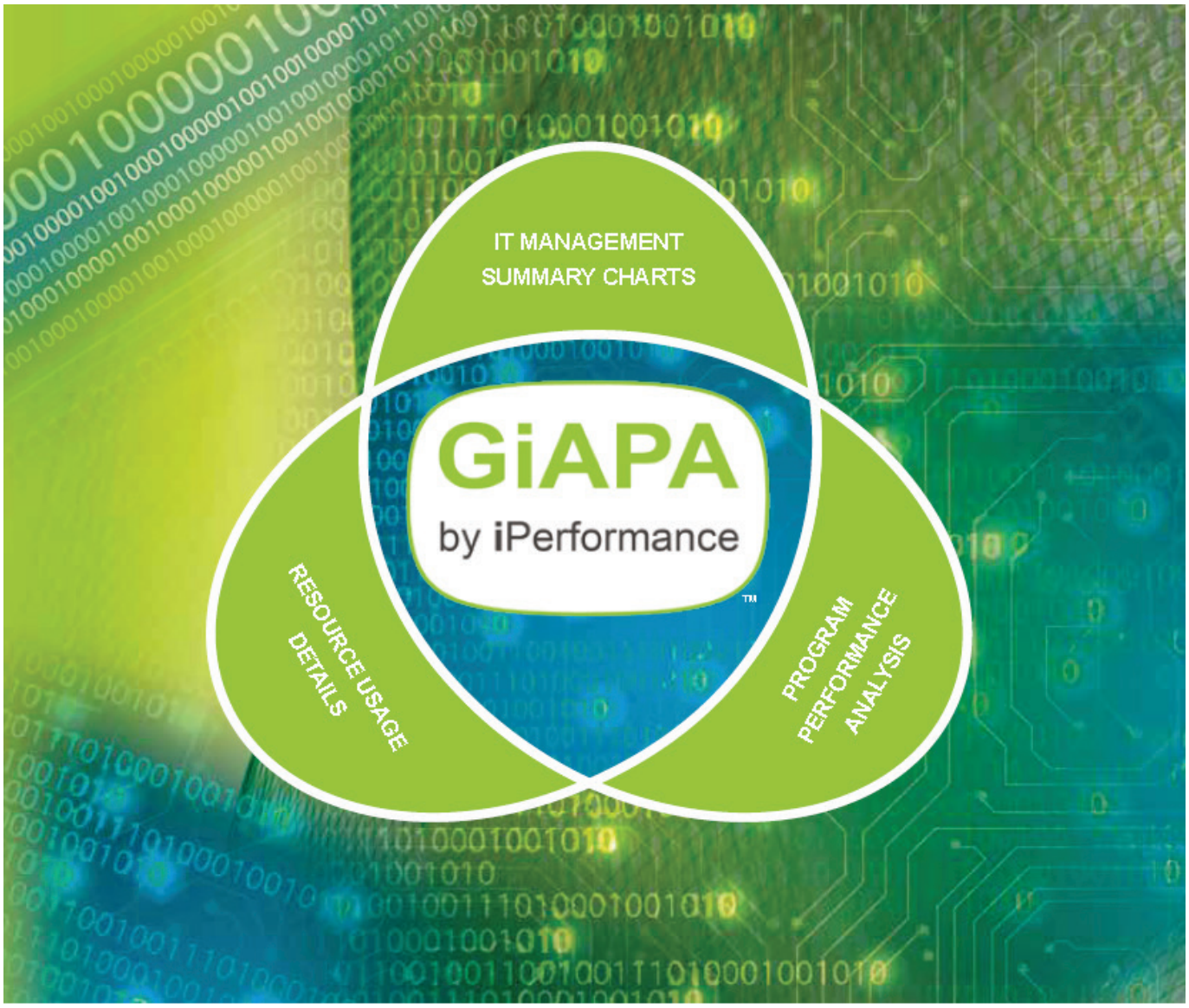
Observe: Watching tutorials 15, 16, and 17 are a prerequisite for solving this exercise!

a. With GiAPA data collection active use GIAPA Menu option 12 to request a HotSpot for your job e.g. 5 times per second:
`GIAPA120 JOBID(123123/MYUSERPRF/MYJOB) COLLTIME(10) COLLINTVAL(0.2)`
Then use command GIAPA130 with parameters Y N (to stop collection, but allow immediate restart), and use Menu option 14 to run data expansion and analysis. You should now be able to see the statement where the job was waiting – probably easiest when you analyze the call stack on all levels, and request to see the job status and statement number. Then you only need to check with the source code to know exactly what happened.

b. and c. When you have seen the tutorial, it should be no problem to run the data collections and generate the reports. We want you to do this so that you know what you can get from PEX. In this case you probably cannot find the wanted answer from any of the two analyses, because it does not show where elapsed time (= waits) is used – but the PEX reports can be useful in special cases. "Statistics" gives very detailed level information on which program or operating system "complex MI function" uses the resources, and if a user program is at the top of the list, the "Profile" run can tell the statement(s) using the resources. But in most cases this information can be found more easily using GIAPAs standard reports.

d. You should try running all the different reports available. Here you also can find what caused the delays – the GiAPA "Trace Job" reports give very detailed information, but high-level program call stack statement numbers are not included here. The "Trace Jobs" it is a good tool to know for the very special cases you only meet once a year or so.

So what was the answer here: Our job is sometimes waiting for a record lock on the ORDERS file. Job status is LCKW (LockWait), we can see the statement number in the CUSTORDERS program, and through this you are able to see which file it is in the program source code.



With IBM Power Systems running IBM i you have a lot of resources available ...

... are they used optimally?