Optimizing Performance at the Speed of Light: Why I/O Avoidance is Even More Important Today

Scott Chapman

Enterprise Performance Strategies, Inc.

Scott.Chapman@EPStrategies.com

Larry Strickland

DataKinetics

lstrickland@dkl.com

Contact, Copyright, and Trademarks

Questions?

Send email to performance.questions@EPStrategies.com, or visit our website at https://www.epstrategies.com or http://www.pivotor.com.

Abstract (why you're here!)

Long gone are the days of the mainframe processors having relatively low processor speeds and relatively small memory sizes. The modern mainframe can have TBs of memory, and the processor clock speeds are among the fastest in the world. Fast processors wait at the same speed as slow processors, but the opportunity cost of waiting is higher. Optimizing processor performance today is all about keeping data as close as possible to the CPU core and reducing the instruction count needed to get to a piece of data. In this webinar, special guest ***Larry Strickland*** from Data Kinetics will join ***Scott Chapman*** for an interesting discussion into why I/O avoidance is more important today and different ways that can be accomplished.

EPS: We do z/OS performance…

- Pivotor - Reporting and analysis software and services
  - Not just reporting, but analysis-based reporting based on our expertise

- Education and instruction
  - We have taught our z/OS performance workshops all over the world

- Consulting
  - Performance war rooms: concentrated, highly productive group discussions and analysis

- Information
  - We present around the world and participate in online forums
    https://www.pivotor.com/content.html

EPS

DATAKINETICS
Z PERFORMANCE & OPTIMIZATION

DKL intro

- Established in 1977
- The global leader in data performance and optimization solutions.
- Solving IT problems and reducing IT costs for the Fortune 500
- Flagship Product tableBASE has been deployed more than 40 years
- IBM Business partner

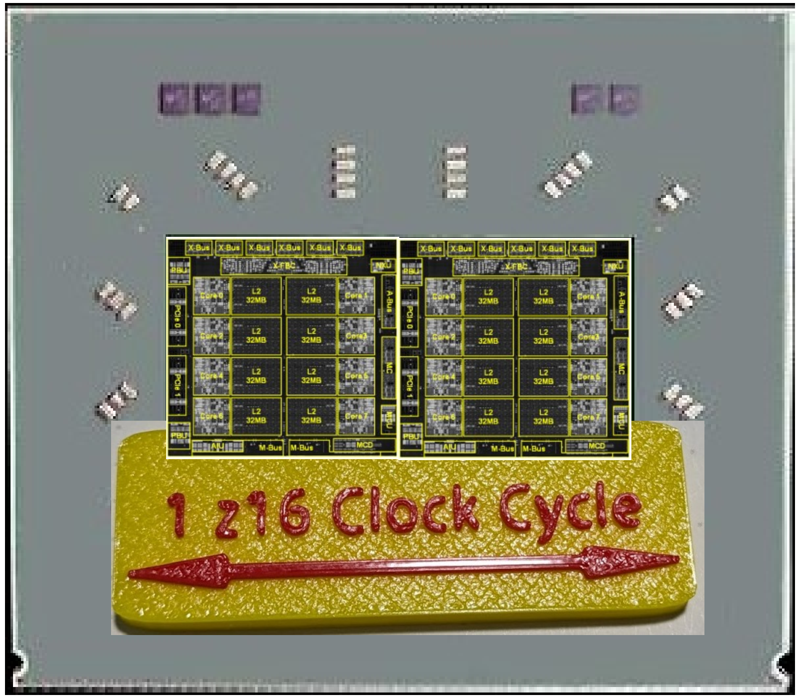
Agenda

- How fast are modern processors and how slow is modern I/O?
- How can we find opportunities for improvement?
- How can we make improvements?

Fast processors, slow I/O

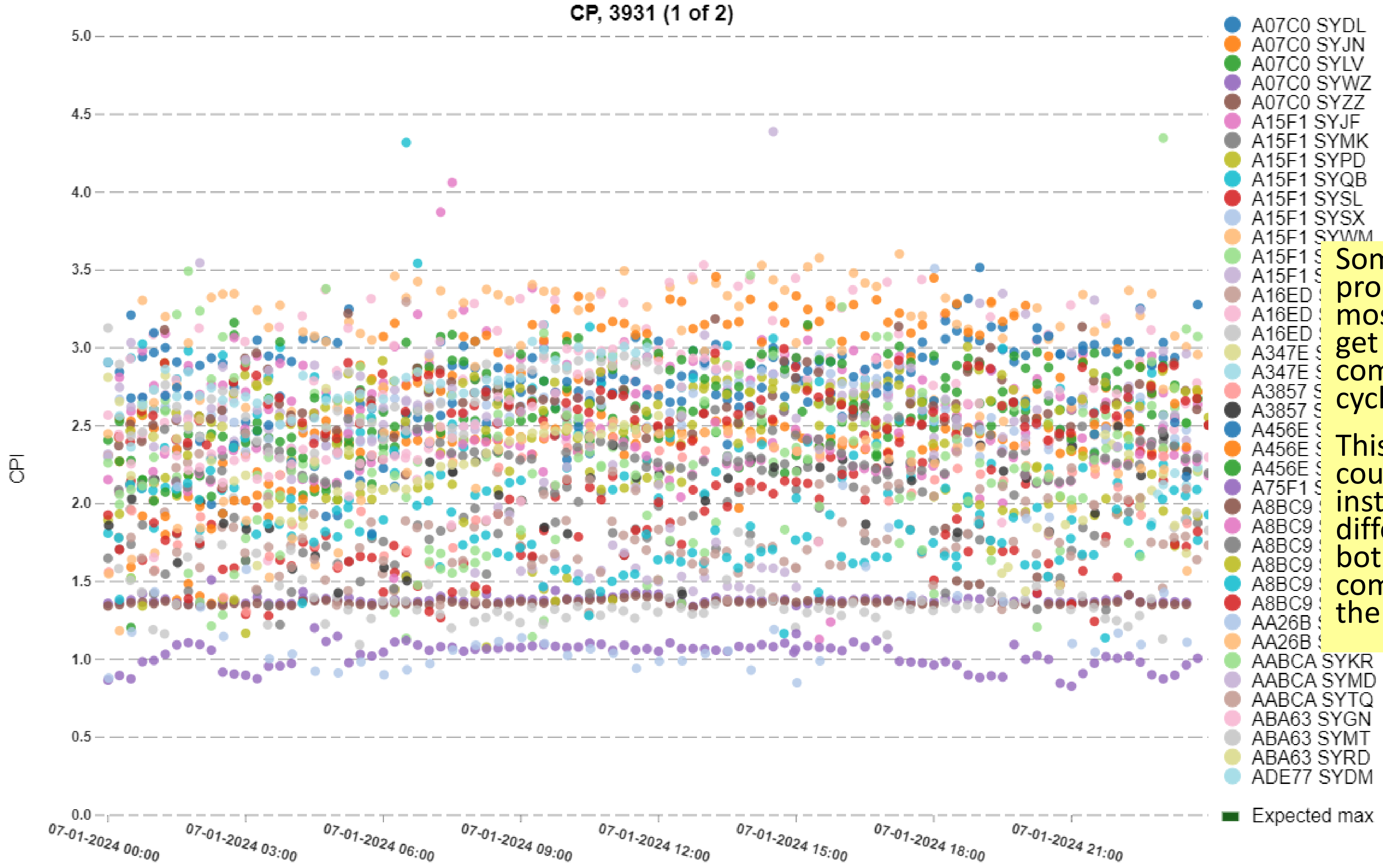
Clock Speed and Cycles

- In one z16 clock cycle, light in a vacuum can only travel just over 2 inches!
  - Electrical signal in a circuit is much slower (40-70% of c)
  - 1 meter in fiber ~ 5 ns (>25 clock cycles!)
- Need to make a round trip
- Signal paths aren't as the mosquito flies
  - IBM's "Miles of wire in the chip" numbers:
    - zEC12 – 7.7 miles
    - z13 – Over 13 miles
    - z14 – 14 miles
    - z15 – 15.6 miles
    - z16 – 19 miles
- Physical distance matters!
  - Basically, to get off the chip, it's going to take multiple clock cycles



Cycles Per Instruction

By z/OS Hardware Model



Somehow (magic of good processor engineering) most systems are able to get an instruction completed every 2-3 clock cycles.

This is a broad average of course—different instructions will take different amounts of time both due to instruction complexity and how close the data is to the core.

System Event Timescale (Seconds)

900.0	Common RMF measurement Interval
10.0	WLM policy interval
2.0	HiperDispatch interval
1.0	1 second (s) (Common RMF sampling interval)
0.250	SRM sampling interval
0.2	Faster-than-average human visual reaction time
0.1	Typical LPAR VH processor time slice
0.0125	Typical LPAR VM/VL processor time slice
0.008	Typical cache miss disk I/O (spinning disk)
0.0032	Typical default zIIPAWMT
0.001	1 millisecond (ms, thousandths)
0.0005	Typical average modern I/O
0.0002	Typical cache hit I/O
0.000030	Possible major z/OS time slice
0.000016	Possible average successful zHyperlink I/O
0.000006	Possible minor z/OS time slice
0.000003	Possible fast successful zHyperlink I/O
0.000001	1 microsecond (µs, millionths)
0.000000162	Main memory access(?)
0.000000001	1 nanosecond (ns, billionths)
0.0000000002	1 z13 machine cycle (5 Ghz)
0.00000000019	1 z14/z15/z16 machine cycle (5.2 Ghz)

milliseconds

microseconds

So we should use zHyperLink?

- Not all I/O eligible to be converted to zHyperLink
- zHyperLink promises response times on order of 10s of µs *for cache hits*
  - Also, no I/O interrupt delay because processor spins while waiting on the I/O
  - Some of I/O overhead of spinning offset by improved CPU L1/L2 cache hits
  - Unsuccessful zHyperLink I/O will result in driving a FICON I/O to complete the I/O
- This is still much slower than main memory
  - Extrapolating from some published numbers, z16 main memory access might be on the order of 162ns (0.162µs)
    - Rough estimate, for memory on the same book
    - L3 would of course be even faster—supposedly on the order of 12ns[1]
- Reading 4K from memory probably 15-100x faster than zHyperlink
  - Means corresponding reduction in the CPU time impact too!

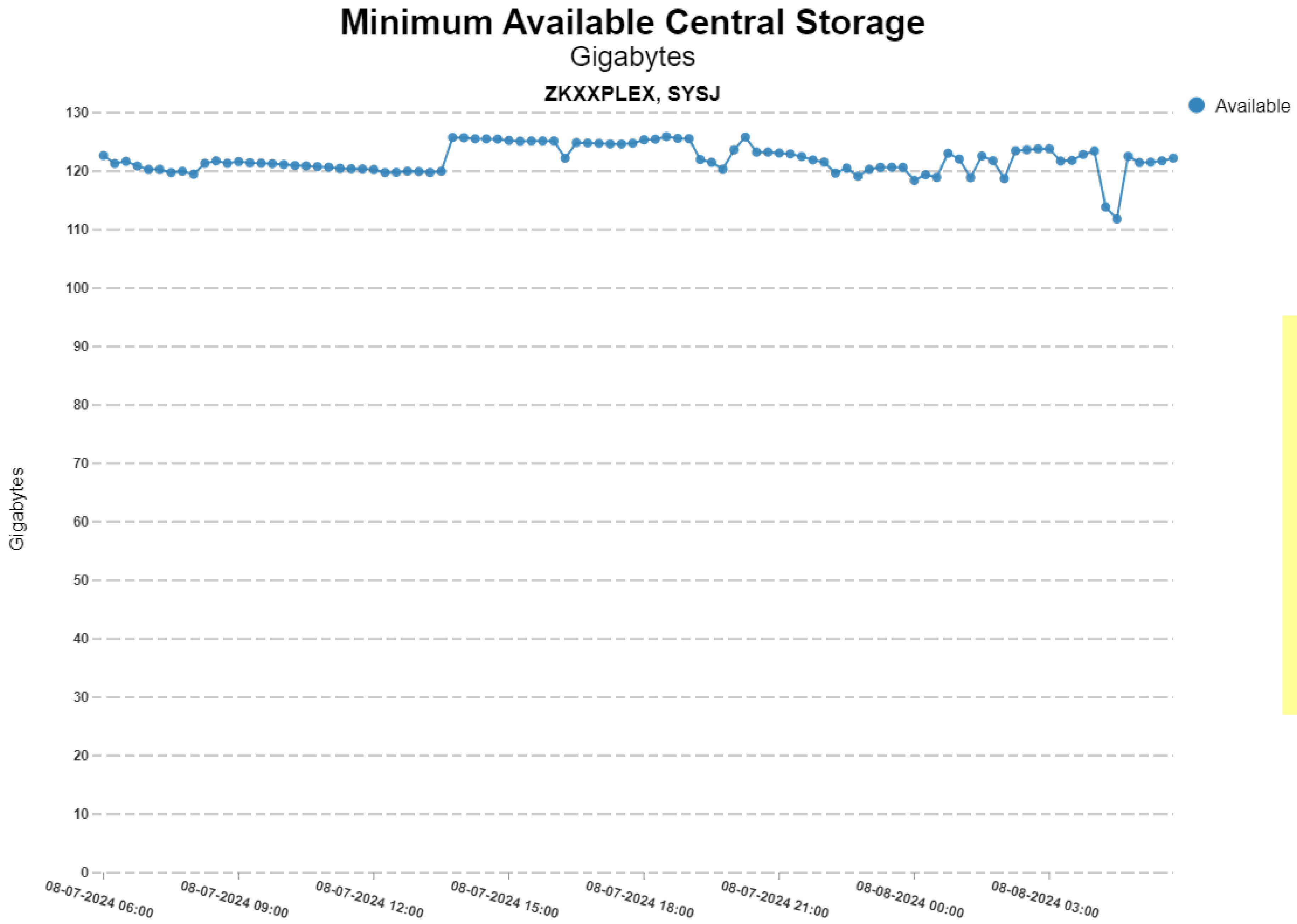
1: https://www.anandtech.com/show/16924/did-ibm-just-preview-the-future-of-caches

DASD cache is limited

- Controller cache is good, but somewhat limited
  - IBM DS8900F – max cache size is 4.3 TB
  - Hitachi DS 5600 – 2 TB/controller block up to 6 TB
  - Dell PowerMax – 15* TB on PowerMax 2500 and 45* TB on PowerMax 8500
    - But those are raw numbers and there's cache mirroring
- Processor can have huge memory in comparison
  - z16 A01 – Max 40 TB
  - z16 A02 – Max 16 TB
  - z/OS LPAR – 16TB (z/OS 3.1)
  - Recent review of our customers' configs showed largest LPAR was 4 TB!
    - LPARs > 1TB certainly becoming more common

The only good I/O is no I/O

- Yes, I/O can be really fast today, but it still takes time
  - But memory is extremely fast
  - I/O: hundreds of microseconds
  - z/Hyperlink: up to tens of microseconds
  - Memory: fraction of a microsecond
- I/O still takes CPU
  - Giving up the CPU while waiting for the I/O to complete means that when redispatched, the work likely won’t have its data and instructions in L1 cache
  - zHyperLink spins on CPU while waiting for completion
- Software cost driven by CPU utilization
  - Usually: Software Cost > Hardware Cost
- Performance gated by bottlenecks
  - I/O not always the bottleneck, but is a common one



LPARs with 10s to even 100s of GBs of free memory are not that unusual today!

Maybe we can use that to keep more data closer to the processor!

So, if you avoid I/O…

- Performance is improved, making the users happier
  - To the degree that users are happy with better performance
  - (And the degree that they notice)
- Possibly reduce CPU consumption, possibly reducing software cost
  - Financial people are only happy with zero cost, but maybe they'll be less unhappy?
- Possibly make better use of unused resources, i.e. memory
  - Management will find something else to critique
- So avoid "unnecessary" I/O
  - Is any read I/O "necessary"? (Yes, but… maybe pretend not!)

Finding opportunities

Finding opportunities in SMF data

- I/O related information is all over in the SMF records
  - Type 14/15 – Old DD-related I/O
  - Type 30 – Summary I/O at job/step
  - Type 42 – Volume and dataset level I/O
  - Type 64 – VSAM Status
  - Type 71 – Paging
  - Type 72 – I/O by service class/report class
  - Type 74 – Volume level I/O details
  - Type 75 – I/O by page dataset
  - Type 100-102 – Various Db2 details, including Db2 I/O
  - Type 110 – CICS details, including CICS I/O
  - Others – Sort, Vendor-specific DASD measurements, etc.

Particularly interesting and "easy"

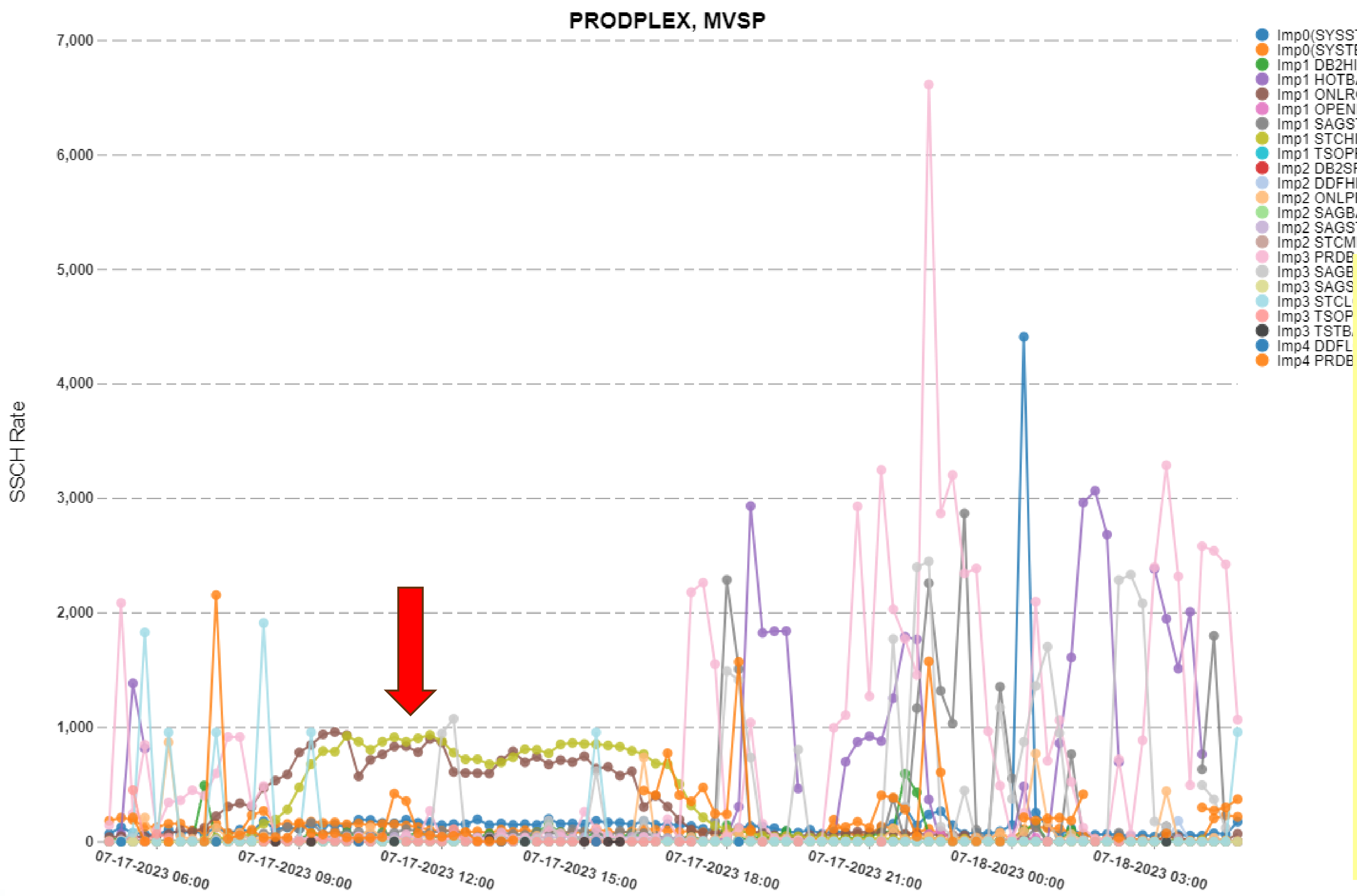
SMF 42: not quite ideal, but good

- Has both interval and “close” statistics at dataset level
  - Interval statistics controlled by SMF interval: ideally 5, 10, or 15 minutes
    (If your interval is >15 minutes please change to 15, and sync those intervals!)
  - But intervals are not written if no I/O, so final close record may cover larger than expected timeframe
- Has some I/O response times at microsecond level precision
  - Some also at 128-microsecond (0.128ms) precision like RMF/CMF
- Has details about cache hit/misses

- Overall, 42.6 excellent source for understanding what is doing I/O!

Example 1

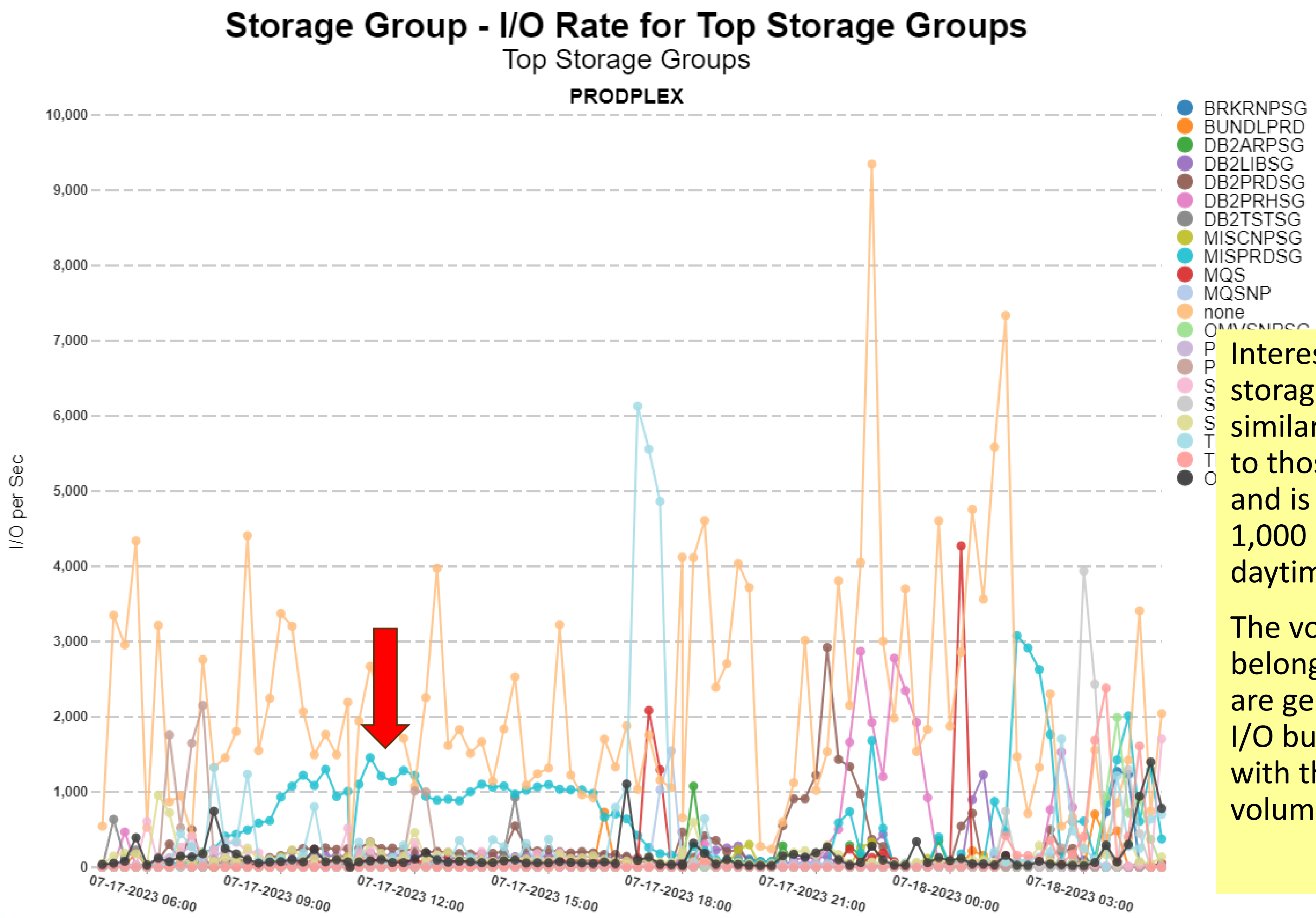
WLM DASD - I/O SSCH Rate by Service Class Period Over Time



Note the two importance 1 service classes each doing a bit under 1000 IOPS during the daytime hours.

This customer's CPU generally was highest during those hours, so despite the higher I/O rates later in the day I was most interested in that daytime activity.

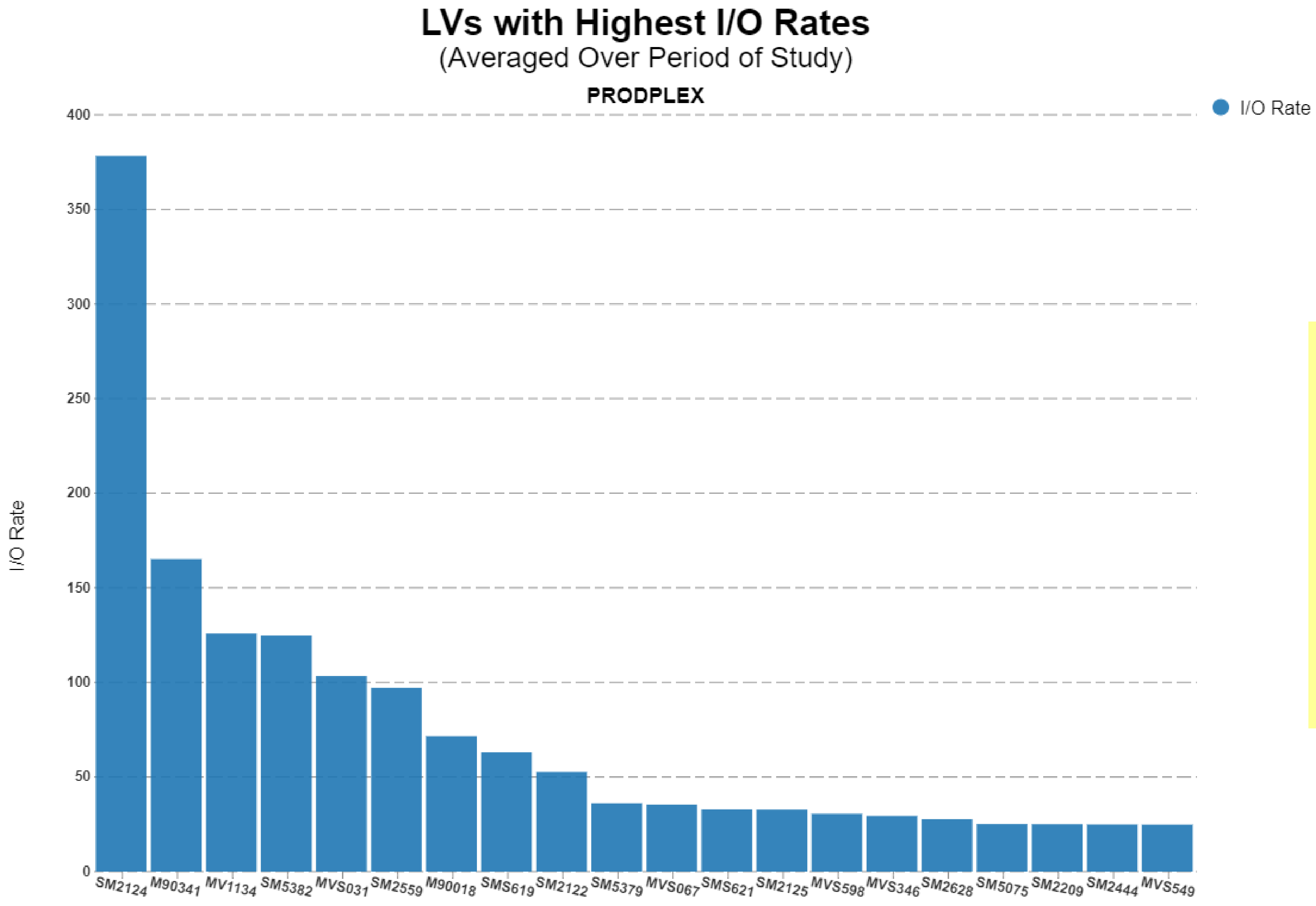
This data comes from the SMF 72 records.



Interestingly, the MISPRDSG storage group shows a similar profile of I/O activity to those two service classes and is actually a bit over 1,000 IOPS during those daytime hours.

The volumes that don't belong to a storage group are generating even more I/O but I thought I'd start with the storage group volumes.

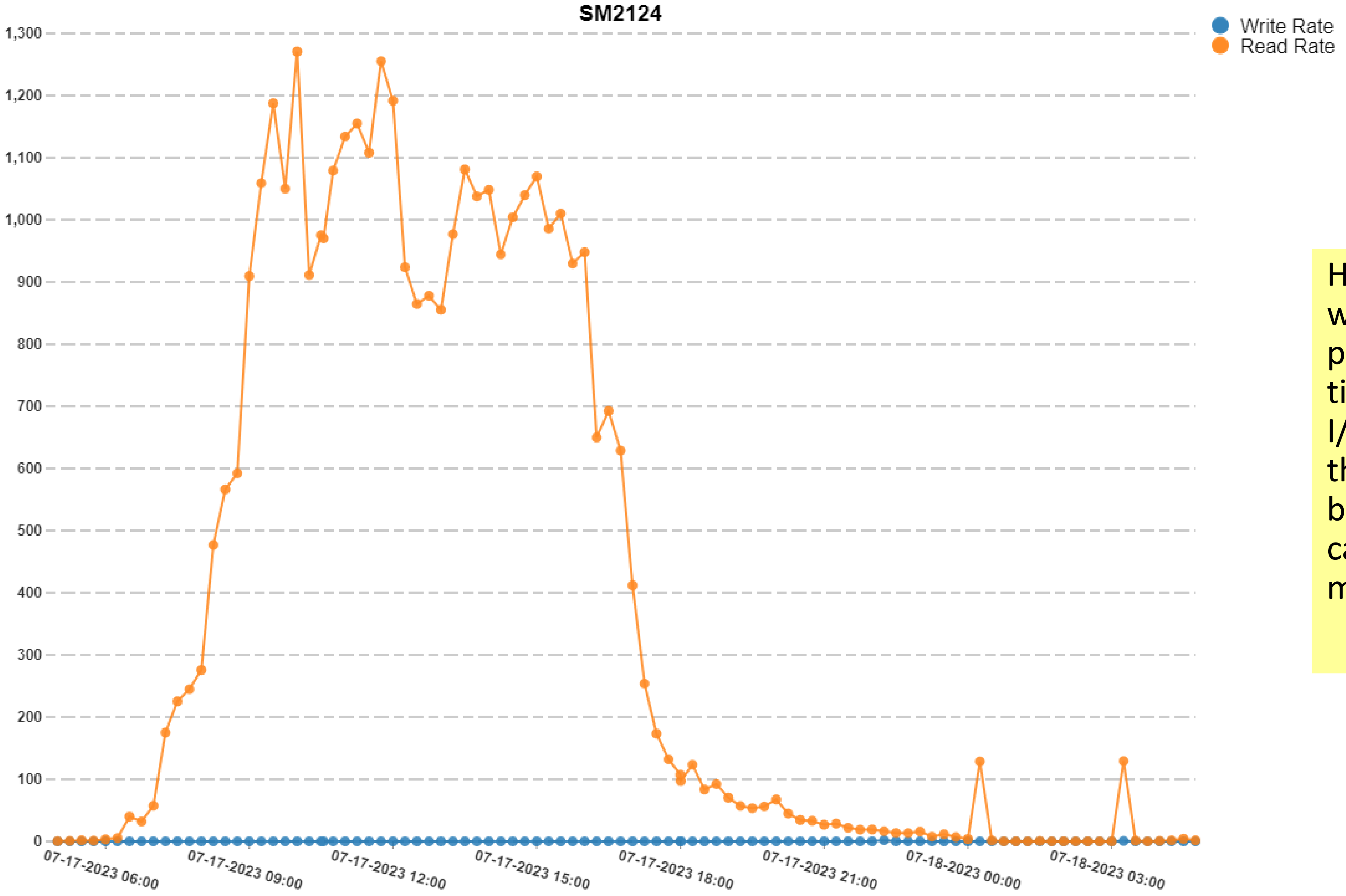
LVs with Highest I/O Rates



So I wondered if any of these top volumes would just happen to be in that storage group of interest. As it turned out, in fact SM2124 was!

Note this is an average I/O rate over 24 hours.

Logical DASD Volume Explorer



Here's the read and write rate for that particular volume over time. Virtually all of the I/O is read I/O, implying that perhaps that could be avoided if we could cache that data in memory.

Volume Details

2023-07-17

smfdate	Storage Gro...	Volser	Device Number	LCU ID	Logical DevType	Capacity GiB	Alloc GiB	Free GiB	Free %	Frag Index	Free DSCBs	Free VIRs
Select Fil	Select Filter	Selec	Select Filter	Sele	Select Filter							
2023-07-18	MISPRDSG	SM2124	961F	96	3390-30	25.932	1.440	24.492	94.000	101.000	90,623.000	1,907.000

In this case we also have at least the volume-level DCOLLECT data from the customer so we can see that surprisingly, there's less than 1.5 GB of data stored on the volume!

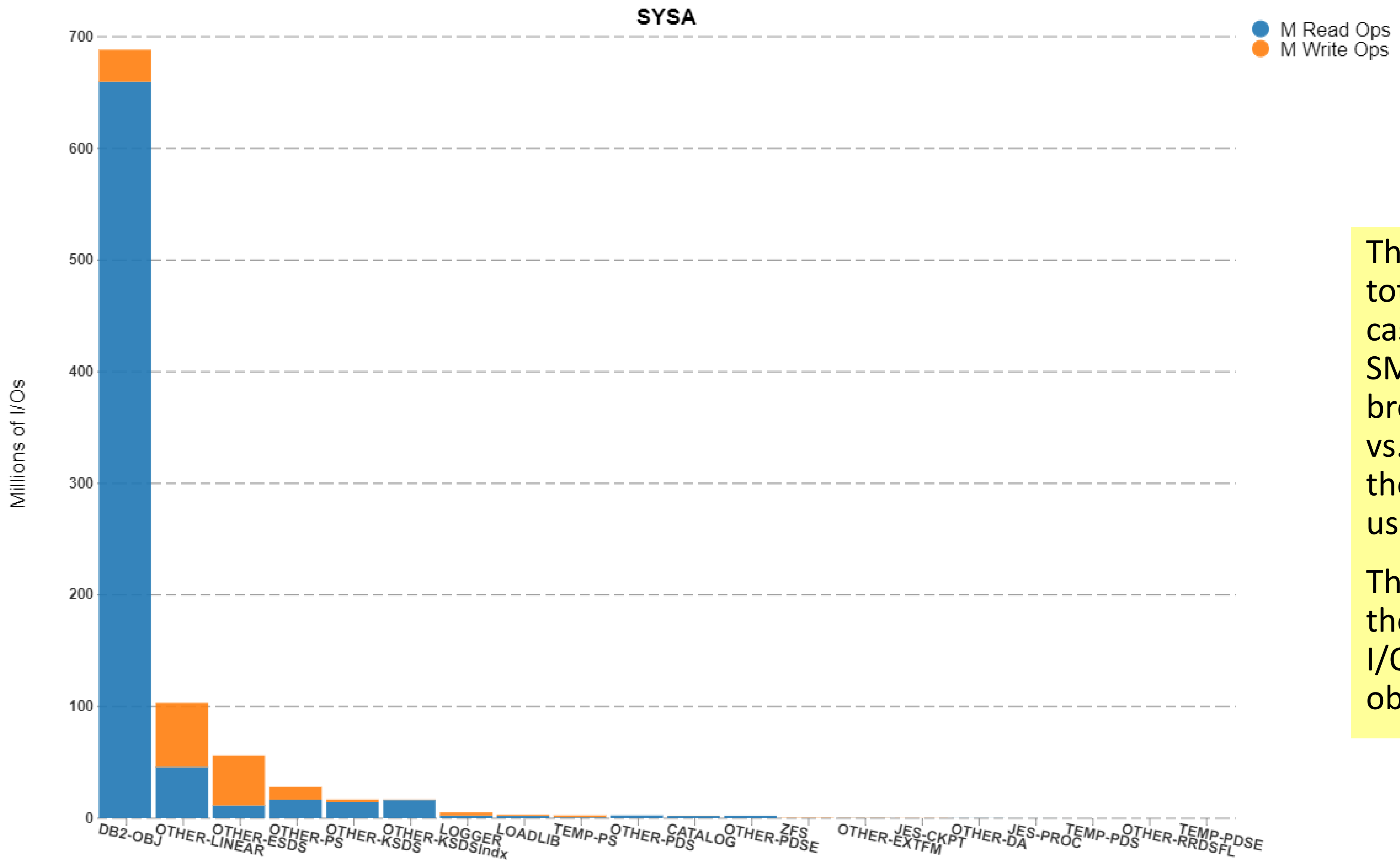
Lacking the SMF 42.6 records, we don't know what datasets are doing the I/O but 1.5 GB is small enough that they could easily store all of that in memory. Doing so could get rid of a significant chunk of their daytime I/O.

Example 2

Coming from the SMF 42 Perspective

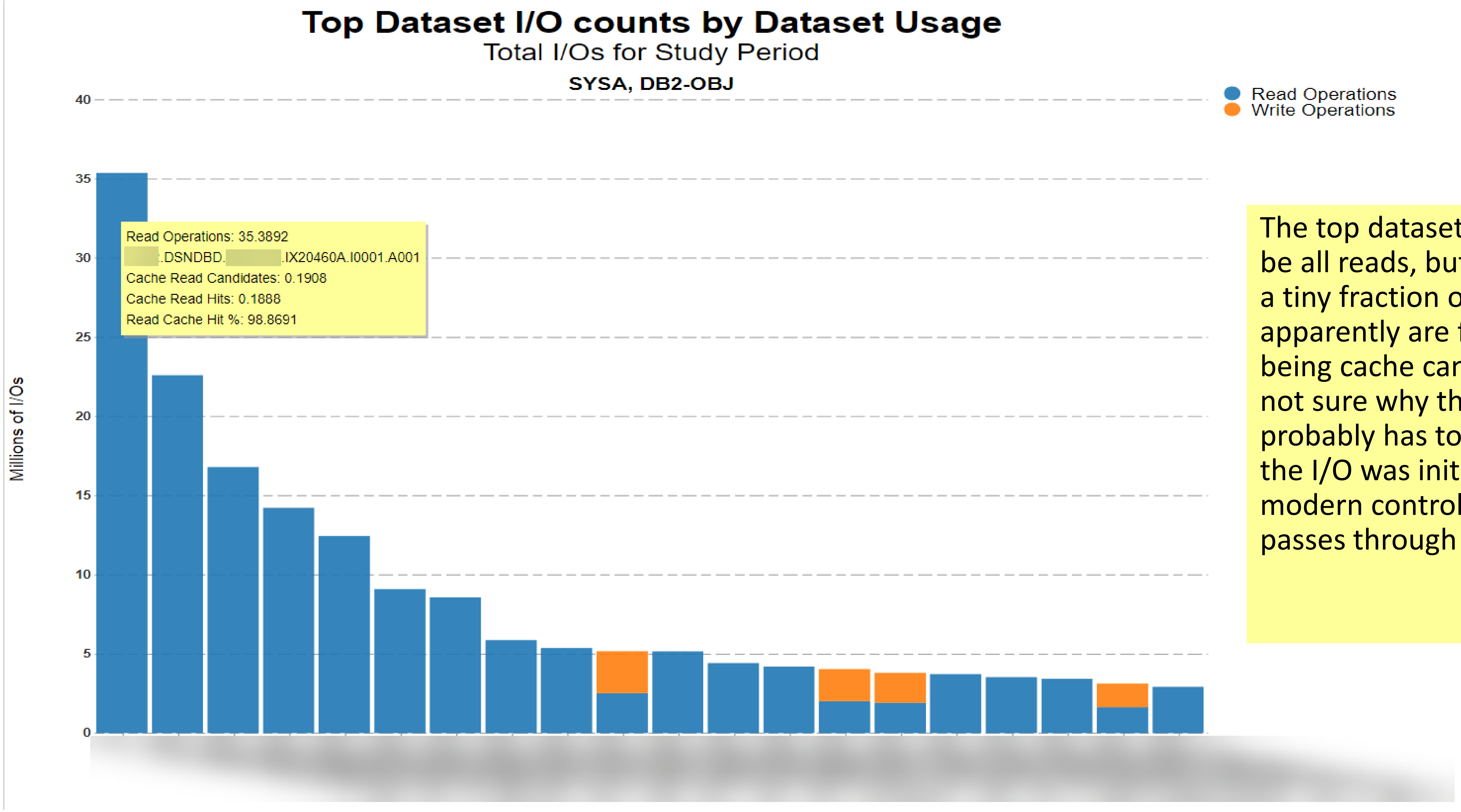
Top Dataset I/O Counts by Dataset Usage

Total I/Os for Study Period



This reports looks at the total I/O over (in this case) a day from the SMF 42 records and breaks it down by reads vs. writes and by what the dataset is (probably) used for.

This site is not unusual: the vast majority of the I/O is reading from DB2 objects.



The top dataset appears to be all reads, but oddly, only a tiny fraction of those apparently are flagged as being cache candidates. I'm not sure why that is, but probably has to do with how the I/O was initiated. In modern control units all I/O passes through cache.

Top Datasets by Cache Read Hits

2023-07-17

Date	Usage	DS Name	M Cache Read Hits	Cache Hit Pct	Read MiB	Allocated MiB	Read-Allocated Ratio	Read O...	Write Ops	42.6 Records	Volume
Select Fil	Select Filte	Select Filter									
2023-07-17	DB2-OBJ	.DSNDBD. .IX20460A.I0001.A001	0.189	98.869	4,201,991.004	569.841	7,373.973	35.389	0.000	54.000	2.0
2023-07-17	DB2-OBJ	.DSNDBD. IX08956B.I0001.A001	20.108	97.188	664,607.695	2,361.233	281.466	22.601	0.001	251.000	8.0
2023-07-17	DB2-OBJ	.DSNDBD. 02.TS06435.J0001.A001	1.450	98.796	1,880,165.453	569.841	3,299.457	16.798	0.000	245.000	5.0
2023-07-17	DB2-OBJ	.DSNDBD. IX00854E.I0001.A001	14.170	99.960	69,569.031	1,339.897	51.921	14.244	0.006	10.000	4.0
2023-07-17	DB2-OBJ	.DSNDBD. IX08956B.I0001.A002	11.436	97.651	288,683.320	1,159.947	248.876	12.444	0.004	241.000	8.0
2023-07-17	DB2-OBJ	.DSNDBD. .TS01452.J0001.A001	8.968	99.152	71,634.121	1,203.719	59.511	9.097			
2023-07-17	DB2-OBJ	.DSNDBD. TS07315.I0001.A001	7.206	98.403	696,717.207	4,175.324	166.865	8.580			
2023-07-17	DB2-OBJ	.DSNDBD. .TS20310.I0001.A001	5.076	96.649	135,041.418	4,720.846	28.605	5.882			
2023-07-17	DB2-OBJ	.DSNDBD. TS08957.J0001.A001	4.991	99.302	228,975.305	4,721.657	48.495	5.390			
2023-07-17	DB2-OBJ	.DSNDBD. TS00854.I0001.A014	3.724	89.910	198,572.117	1,957.563	101.438	5.179			
2023-07-17	DB2-OBJ	.DSNDBD. .TS01451.J0001.A001	1.365	94.870	332,869.598	306.401	1,086.384	4.428			
2023-07-17	DB2-OBJ	.DSNDBD. TS07315.I0001.A001	3.588	99.894	499,237.086	4,377.969	114.034	4.210			
2023-07-17	DB2-OBJ	.DSNDBD. TS07315.J0001.A001	3.121	99.978	417,840.731	4,341.493	96.244	3.551			
2023-07-17	DB2-OBJ	.DSNDBD. .TS07292.I0001.A001	2.965	99.891	77,346.273	4,722.468	16.378	3.444			
2023-07-17	DB2-OBJ	.DSNDBD. .TS17613.I0001.A001	0.156	94.461	345,553.938	284.516	1,214.534	2.930			
2023-07-17	DB2-OBJ	.DSNDBD. .TS07292.J0001.A002	2.255	99.791	71,282.477	4,721.657	15.097	2.738			
2023-07-17	DB2-OBJ	.DSNDBD. TS02809.J0001.A009	2.170	89.721	77,474.367	3,465.251	22.358	2.688			
2023-07-17	DB2-OBJ	.DSNDBD. TS02813.J0001.A004	2.085	95.299	63,996.520	3,465.252	18.468	2.609			
2023-07-17	DB2-OBJ	.DSNDBD. .TS06562.I0001.A001	2.316	96.014	37,477.902	1,738.704	21.555	2.499			
2023-07-17	DB2-OBJ	.DSNDBD. TS17820.I0001.A001	1.968	92.237	73,247.258	2,691.953	27.210	2.451			
2023-07-17	DB2-OBJ	.DSNDBD. .TS17613.J0001.A001	0.062	93.959	295,867.445	284.516	1,039.899	2.446			

This table report joins the SMF 42 data with the DCOLLECT data to get the total allocated size (summed across multiple volumes if necessary) of the datasets.

Note there's little write activity and a number of these datasets are only a few GB.

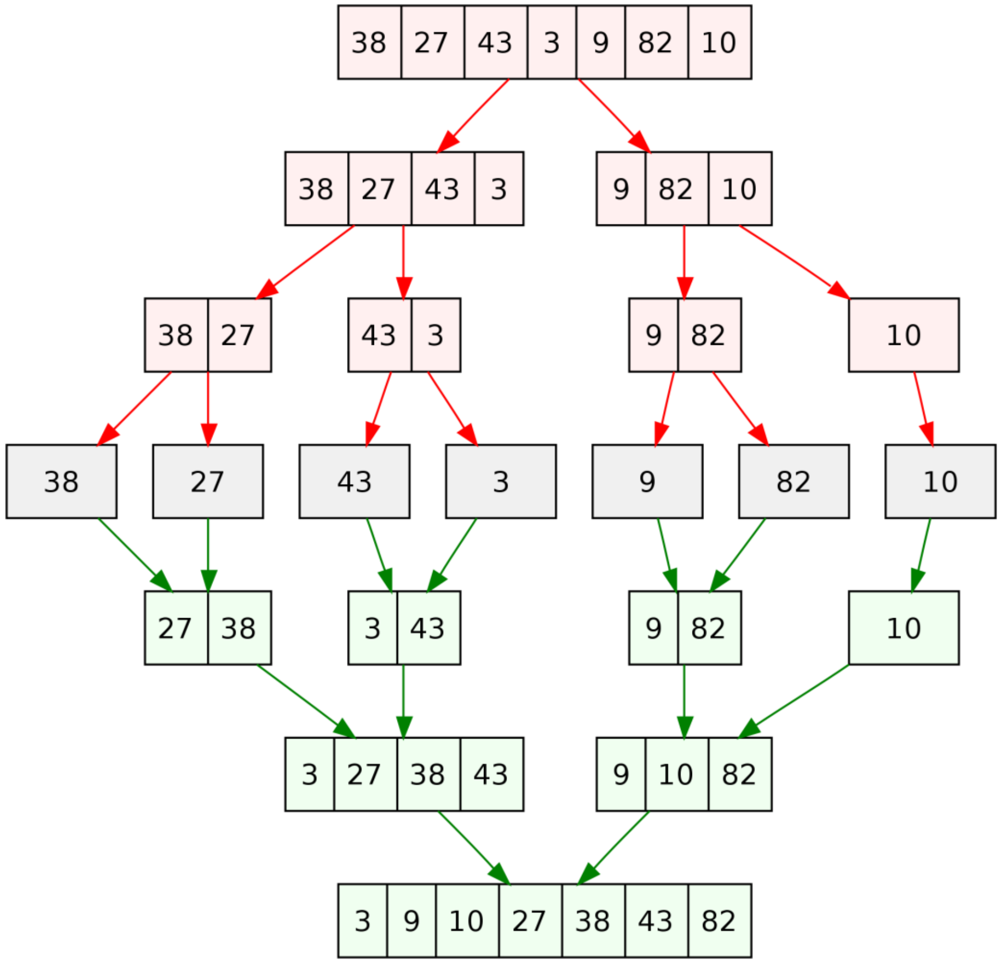
Even if they can't all go into memory, probably some can, saving 10s of millions of I/Os.

How can we do less I/O?

Sorting

- Sort products can make good use of memory for in-memory sorts
- This has been true for decades
  - Make sure you have your parms right: use available memory, don't cause problems
- New SORTL facility on z15 allows even more improvement
- DFSORT exploitation called "IBM Z Sort" https://www.ibm.com/support/pages/ibm-z-sort-and-dfsort-considerations
  - Requires memory >= 70% of dataset size, 200% is recommended planning number
  - IBM test of 44GB sort resulted ~50% reduction in ET and ~40% reduction in CPU vs. in-memory sort without ZSORT
  - But, somewhat oddly, using SORTWK was actually about the same CPU as ZSORT, although it took almost 9x longer (341 seconds vs 39 seconds)
    - So performance savings, but not really CPU savings compared to doing I/O
- I haven't seen a SyncSort benchmark

In-Memory Sort



In-Memory Sort :DB2

- Db2 v12 improved its RDS sort processing using more memory:
- Expanded the maximum number of nodes in a sort tree, from 32,000 to 512,000 for non-parallel sorts or 128,000 for parallel sorts under child tasks.
- These enhancements might require more memory to be allocated to the thread for sort activities, but can result in a significant CPU reduction.
- Requires the use of more memory – but ….

Sort performance measurements (DB2 v12)

- In-memory sorts that previously required work files for sort and merge processing
  - 75% reduction in CPU time

- Increased sort pool size
  - 50% reduction in elapsed time and CPU time

Sort performance measurements (cont'd)

- SAP workloads
  - SAP CDS Fiori: 5% CPU time reduction for several queries (1% CPU time reduction across the entire workload)
  - SAP CDS FINA: 1.8% reduction in CPU time for the entire workload (12% reduction in the total number of GETPAGEs)
- IBM Retail Data Warehouse
  - Two queries: 14% and 6% CPU time reduction

In-memory Table examples

- DB2 – In-memory table
- DB2 – Table fixed in buffer pools
  - Structures still support on disc
- VSAM
  - Fully buffered

Controlled/managed predominately by DBMS

- Pure In-Memory Tables
  - IBM IZTA
  - DKL tableBASE
- Cobol Internal Tables (other languages too!)
  - Limited to a primary index
  - Not Shareable
- Home Grown In-Memory Accelerators
  - Often from when people built their own everything

Controlled/managed predominately by Application (developer)

Db2 Buffer Pools

- Basically: pin the objects in a buffer pool
- Make BP big enough to hold the entire object(s)
  - Db2 systems with 100s of GBs of buffer pools are increasingly common
- Optionally set PGSTEAL(NONE)
  - Indicates to Db2 you believe the BP is big enough to hold all of the object(s) in the BP
    - Doesn't mean that Db2 won't steal pages from it if need be
    - Doesn't mean that the pool is read-only
  - Db2 will use async prefetch to pre-load the objects on first reference
  - Note: Don't use PGSTEAL(NONE) and FRAMESIZE(2G) together.
    - NONE & 2G will be treated as LRU & 2G. Use NONE & 1M instead!
- Remember to page-fix your production BPs (at least, maybe dev/test too)
  - CPU reduction for every I/O to/from the BP

Db2 – Group Buffer Pools

- Rule of thumb for GBP size is sum(local BPs) / 3
  - Goal is to avoid directory entry reclaims
- BPs with very little update activity may not need as much
  - Might also consider GBPCACHE(NO) for such
    - GBP will only be used for cross-invalidation; writes will suffer though
- Other idea: use GBP storage instead of LBP storage
  - Instead of really large LBPs, use really large GBP
  - Saves on the amount of memory you need overall
  - Set with GBPCACHE ALL on the object level
    - Pages will be copied to GBP as they're read regardless of inter-system read/write interest
  - Benefit similar to zHyperLink without actually having to implement zHyperLink
    - Probably actually a little better since DB2 will check the GBP anyways

VSAM Buffering

- There are 4 types of buffer pool management for VSAM:
  - NSR - Nonshared Resource
  - LSR – Local Shared Resource
  - GSR – Global Shared Resource (no longer used)
  - RLS – Record-Level Sharing
- Set by the open, not part of the VSAM dataset definition
- See Chapters 4-6 of VSAM Demystified Redbook
  - https://www.redbooks.ibm.com/abstracts/sg246105.html

Why application managed In-Memory

- *a priori* knowledge of data in-memory – allows for optimized sort/search
- Code path and algorithms optimized to 4k pages, maximizing effectiveness of Cache
  - Remember following table?

....

milliseconds	0.008	Typical cache miss disk I/O (spinning disk)
	0.0032	Typical default zIIPAWMT
	0.001	1 millisecond (ms, thousandths)
	0.0005	Typical average modern I/O
	0.0002	Typical cache hit I/O
microseconds	0.000030	Possible major z/OS time slice
	0.000016	Possible average successful zHyperlink I/O
	0.000006	Possible minor z/OS time slice
	0.000003	Possible fast successful zHyperlink I/O
	0.000001	1 microsecond (µs, millionths)
	0.0000003	Fetch by Key - average
	0.000000162	Main memory access(?)
	0.00000008	Fetch Next - average
	0.000000001	1 nanosecond (ns, billionths)
	0.0000000002	1 z13 machine cycle (5 Ghz)
	0.00000000019	1 z14/z15/z16 machine cycle (5.2 Ghz)

Best Candidates

High row read rate (small tables, frequent reads)

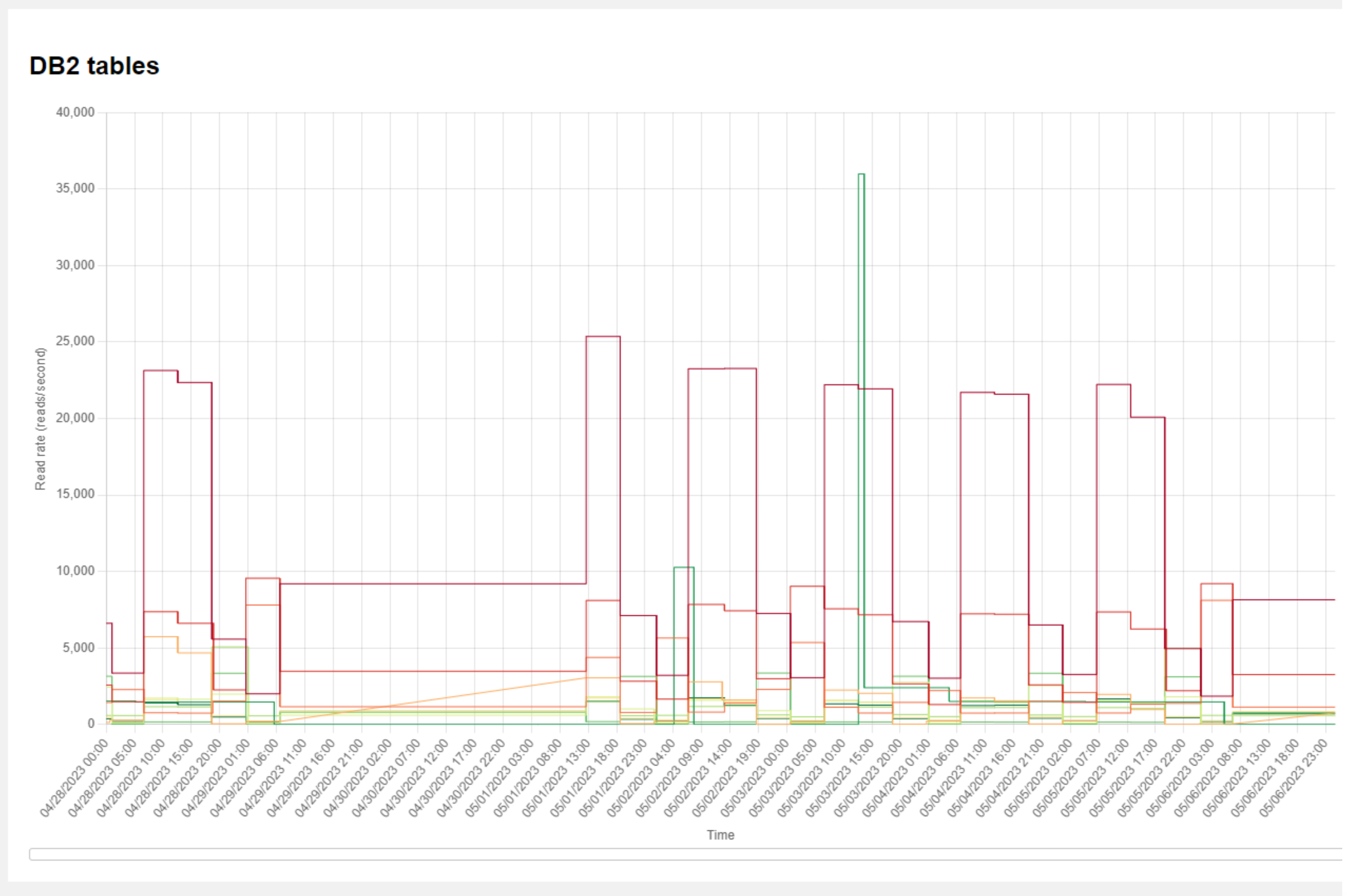
- Reference Data – the many small tables touched during transaction processing
- Rules tables – multiple rules returned to define processing for each transaction

Temporary Tables

- Created, sorted/searched/filtered then abandoned
- Avoid I/O

DB2

Finding small frequently read DB2 tables



NB:data from System Tables, not SMF

VSAM

Derived from SMF64 data

Rea

EXCPS=> I/O => Wall C

Opens => Both

VSAM Files aggregated by DSN

Excel | CSV | Print | Column visibility

Search:

DSN	Reads	EXCPS	Changes	Open and Close Count	Unique Jobs	Largest Size (GB)	From	To	Time Open (h:m:s)	Time Open (secs)
YA1JFXV.OY.UCYF6FJH	804,801,608	67,124,595	0	6,428	843	0.08	3/11/2025, 12:02:21 AM	3/12/2025, 11:59:09 PM	28:8:40	101320.5
YK81EVR.OC.ERJ.F03SKG	792,485,934	331,645	0	4,842	849	0.04	3/11/2025, 12:03:18 AM	3/12/2025, 11:59:44 PM	44:0:5	158405
YV6AWWT.OY.JGBKUYOD.S25KXA	739,836,054	1,391,218	0	45,922	1,105	0.07	3/11/2025, 12:03:18 AM	3/12/2025, 11:59:45 PM	43:16:59	155819.5
K6FF7HY.OY.LEC.EVWNCYY.BV	491,921,269	419,247	0	52	9	3.30	3/11/2025, 7:58:19 AM	3/11/2025, 2:35:38 PM	1:50:56	6656
SEVC15A.OY.J5DBN24J.U6A35WR	414,296,900	1,145,343	10,526,749	98	10	2.79	3/11/2025, 2:39:36 AM	3/12/2025, 9:06:00 PM	0:23:6	1386
BWWE5DQ.OY.J5DBN24J.U6A35WR	351,875,303	533,479	5,724,961	140	11	1.63	3/11/2025, 2:13:37 AM	3/12/2025, 11:11:33 AM	0:18:31	1111
B04OS7S.OY.J8KLX8XQ.LU8H	314,261,564	3,176,916	0	672	118	8.12	3/11/2025, 12:04:28 AM	3/12/2025, 11:38:53 PM	5:47:55	20875
YK81EVR.OY.ERJ1.F03SKG	249,672,636	17,711	0	186	59	0.04	3/11/2025, 12:45:56 AM	3/12/2025, 11:47:36 PM	4:4:57	14697
YK81EVR.OY.ERJ1.BYW0XSI	248,222,058	658	0	114	35	0.00	3/11/2025, 12:37:28 AM	3/12/2025, 3:12:20 PM	1:3:59	3839
YA1JFXV.OY.LW397S	239,551,630	13,140,882	0	6,404	839	0.02	3/11/2025, 12:02:21 AM	3/12/2025, 11:59:09 PM	28:8:53	101333.5

Showing 1 to 10 of 16,278 entries

Previous 1 2 3 4 5 … 1,628 Next

Summary

- Many (but not all) systems are memory-rich today
  - And if you're not, maybe you should be?
- Take advantage of that memory to avoid I/O to
  - Improve performance
  - (Potentially) reduce CPU and thus (potentially) reduce costs
- SMF has a plethora of data to help you find your I/O
  - SMF 42 records has a good level of detail
- Once you've found your I/O, avoid it by keeping the data in memory
- Consider in-memory tables to avoid Db2/VSAM and save even more CPU

Questions??