The Skinny on Coupling Thin Interrupts

Session 16644

Frank Kyne
Editor and Technical Consultant
Watson and Walker

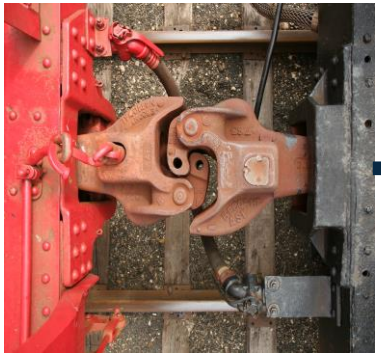
#SHAREorg

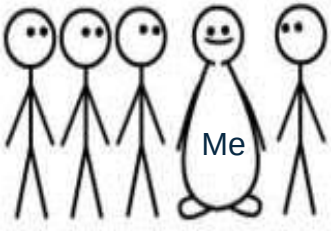
SHARE is an independent volunteer-run information technology association that provides **education**, professional **networking** and industry **influence**.

SHARE in Seattle 2015

© Copyright Watson & Walker 2014,2015 Copyright (c) 2014 by SHARE Inc. Except where otherwise noted, this work is licensed under http://creativecommons.org/licenses/by-nc-sa/3.0/

Session objectives





© Copyright Watson & Walker 2014,2015

Welcome

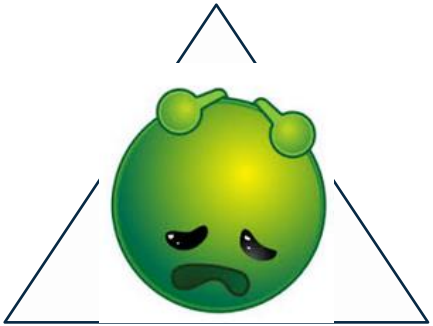
- Hi, thanks for coming
- Who I am and what I do
- What we are going to talk about:
  - WHY good Coupling Facility response times are important to your business.
  - WHAT Coupling Thin Interrupts are and HOW they contribute to better CF response times
  - WHEN to fine tune your overall CF response time profile by manipulating XES thresholds
- PLEASE ask questions as I go along

© Copyright Watson & Walker 2014,2015

Why care about CF response times?

- CF response times are so short (measured in microseconds), why would you care about them?

- A typical data sharing customer does 2-3 times as many CF requests as DASD I/O.
  - Yet most companies spend far more time, money, and resource on DASD performance management than they do on CF performance.



© Copyright Watson & Walker 2014,2015

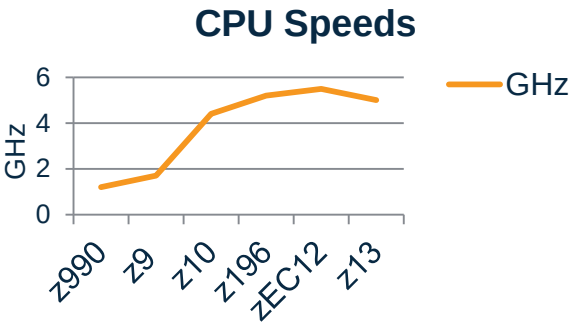
Why care about CF response times?

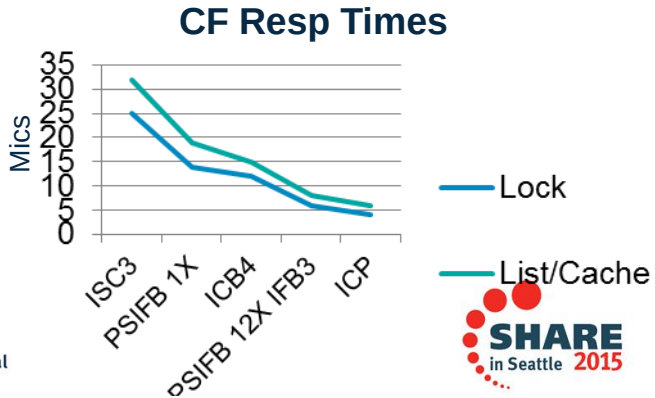
- AND, for DB2 data sharing at least, most DASD I/Os happen *before* the txn starts, or *after* it completes:
  - Prefetch data
  - Some DB2 log writes are asynchronous to transaction execution
  - Updated data is hardened to database after transaction ends
- But most CF requests happen during the life of the txn.
  - Get locks
  - Register interest in data in the GBP
  - Retrieve copy of data from GBP
  - Write updated data to GBP
  - Release locks

© Copyright Watson & Walker 2014,2015

Why care about CF response times?

- IBM have stated that z CPU speeds are going to stabilize as Moore's Law comes to an end. If CPUs are not getting faster, you need to find your performance improvements elsewhere.

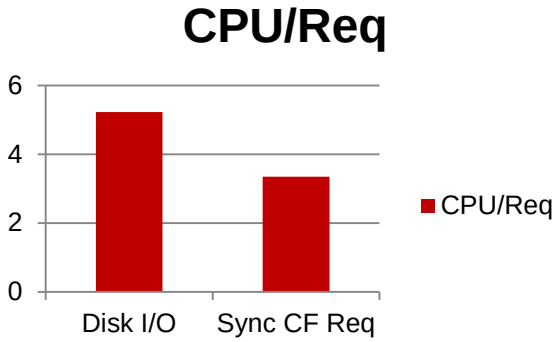


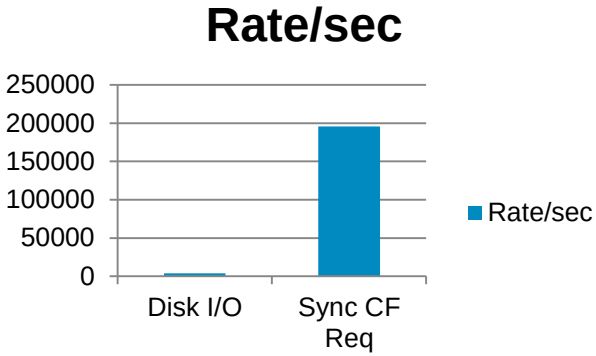


© Copyright Watson & Walker 2014,2015

Comparative performance

- Just to get *some feel* for the comparative cost and performance of disk vs CF, we ran a job to read 4KB blocks from a sequential data set, then ran same job reading 4KB blocks from CF list structure.





- Actual measurements, but *not* in a controlled lab environment
- Provided purely for illustration of the relative performance and CPU cost of DASD I/O vs. CF requests

© Copyright Watson & Walker 2014,2015

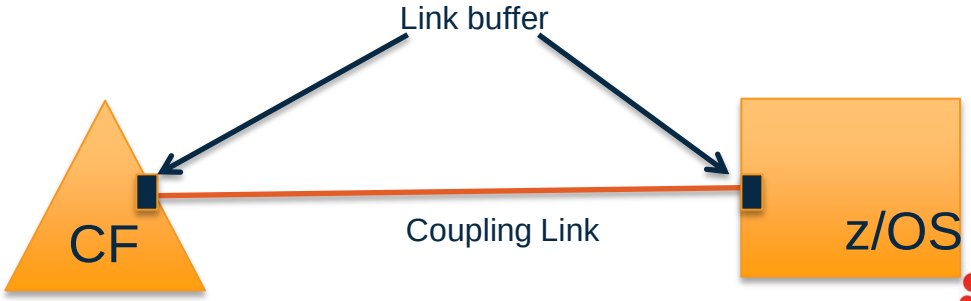
Why care about CF response times?

- Short-running CF requests consume less resource (that means less $) than longer requests:
  - Subchannels are busy for the entire response time, so the shorter the response time, the lower the utilization:
    - You get more subchannels (to reduce subschannel utilization) by adding more CF Link CHPIDs or maybe buying more CF links, so by minimizing CF response times you minimize CF link requirements.
  - Long synchronous CF requests consume more z/OS CPU time than short synchronous requests (because z/OS CPU consumption of a synchronous request = service time of that request).
  - Asynchronous CF requests consume more z/OS CPU time than short synchronous requests. They also lower the z/OS capture ratio, making chargeback more challenging.
  - Long-running requests elongate transaction and batch job elapsed time and cause more lock contention (which also costs more CPU time).

© Copyright Watson & Walker 2014,2015

Coupling Thin Interrupts

- What exactly IS a Coupling Thin Interrupt?
  - Prior to Driver 15 (zBC12 and zEC12 GA2) the arrival of an unsolicited signal on a Coupling Facility link did not generate an interrupt.
    - This meant that the users of Coupling Links (Coupling Facilities and z/OS systems) needed some other mechanism to detect the presence of something in the link buffer that needed to be attended to.



© Copyright Watson & Walker 2014,2015

Coupling Thin Interrupts

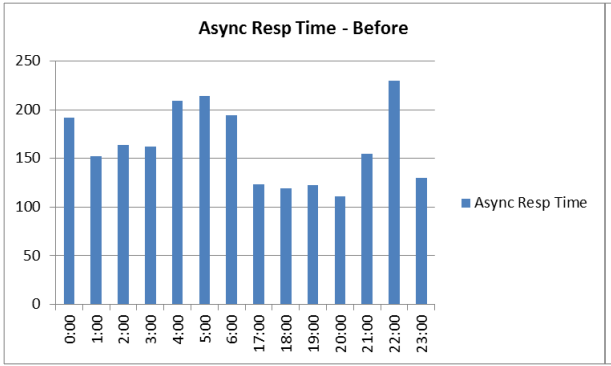
- Driver 15 introduced the ability for the link hardware to generate an interrupt when something arrives in the link buffer.
  - This ability can be enabled and disabled by the “operating system” that owns the link buffer.
- Because the CF signals are simpler than other forms of I/O (DASD I/O, for example) the interrupts associated with them are less complex.
- Hence:
  - Coupling Because they are for Coupling Links
  - Thin Because they are “light weight” – they do not carry as much information as I/O interrupts
  - Interrupts Because they generate an interrupt.
- Note that there are also other types of “thin” interrupts – for QDIO, for example.

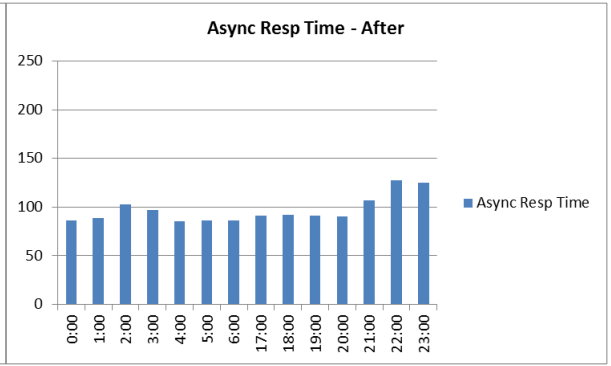
© Copyright Watson & Walker 2014,2015

Coupling Thin Interrupts

This is what Coupling Thin Interrupts can do for YOU

Production environment, Customer 1



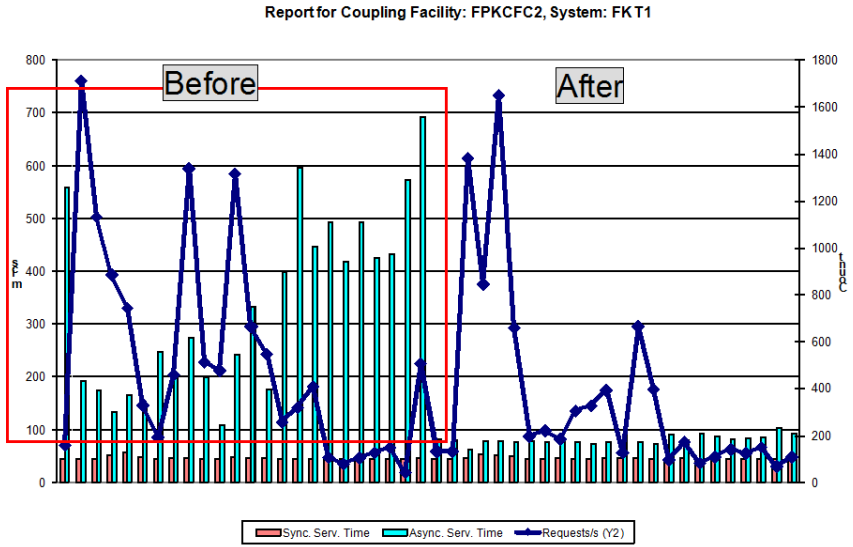


© Copyright Watson & Walker 2014,2015

Coupling Thin Interrupts

Or maybe this….

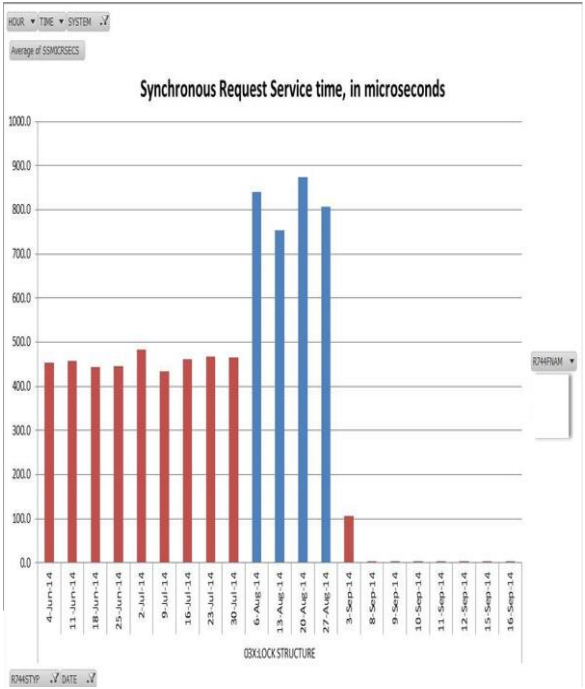
Test environment, Customer 2

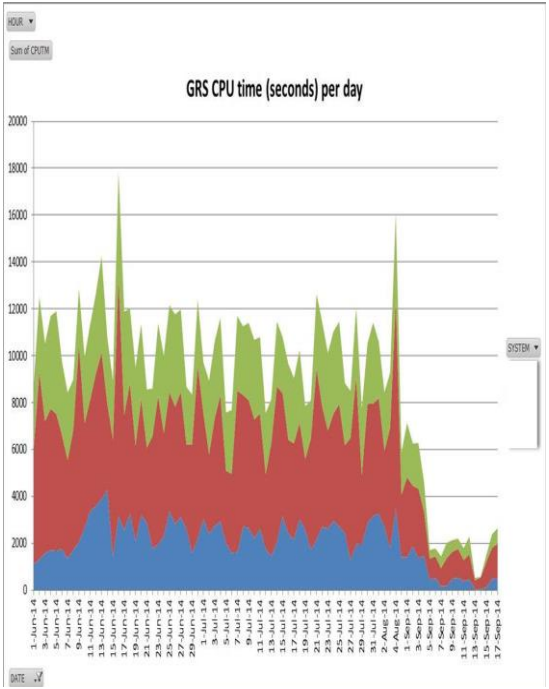


© Copyright Watson & Walker 2014,2015

Coupling Thin Interrupts

Or even this (service time for ISGLOCK and CPU time for GRS)





© Copyright Watson & Walker 2014,2015

Coupling Thin Interrupts

- The Coupling Thin Interrupt capability can be exploited by both Coupling Facility and by z/OS, but in different ways.

- Let’s look at z/OS first.

© Copyright Watson & Walker 2014,2015

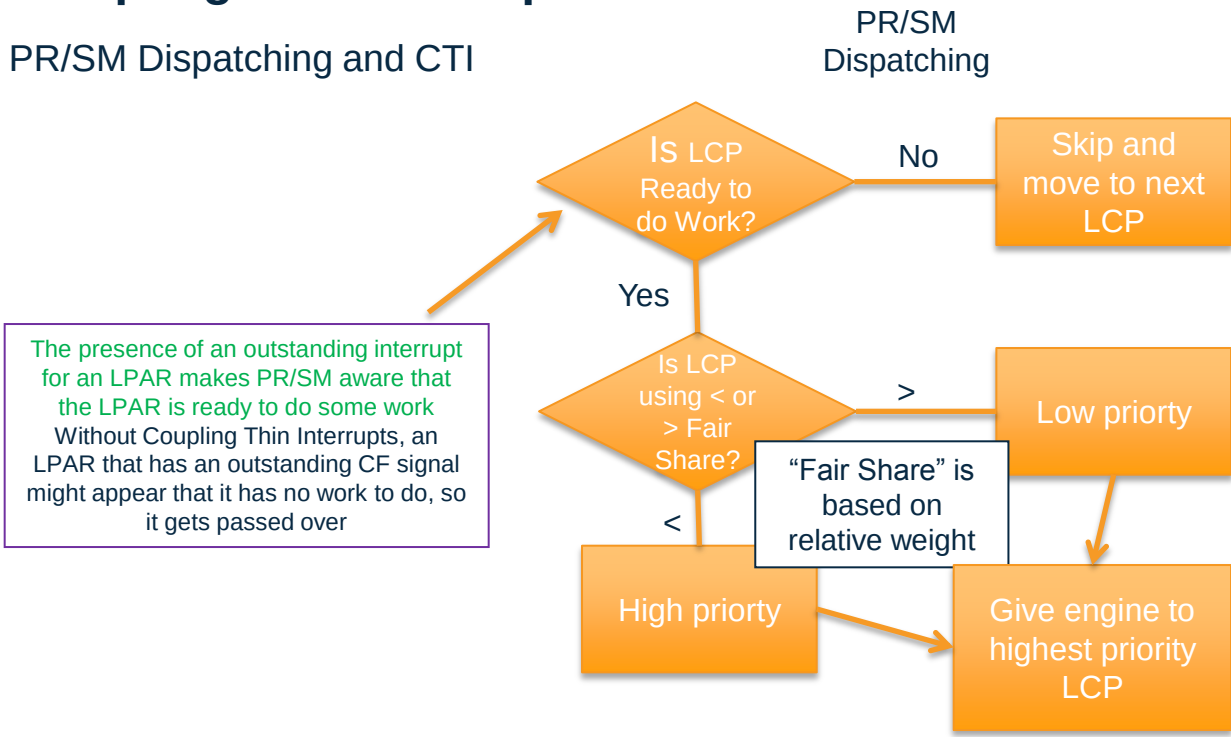
Coupling Thin Interrupts

- We need a little background information first….
- On z/OS, Coupling Thin Interrupts can change:
  - How PR/SM dispatching works for a z/OS LPAR:
    - On the z/OS end, Coupling Thin Interrupts can be used regardless of whether z/OS is using shared or dedicated engines.
  - How XES becomes aware that he has some work to do.

© Copyright Watson & Walker 2014,2015

Coupling Thin Interrupts

PR/SM Dispatching and CTI

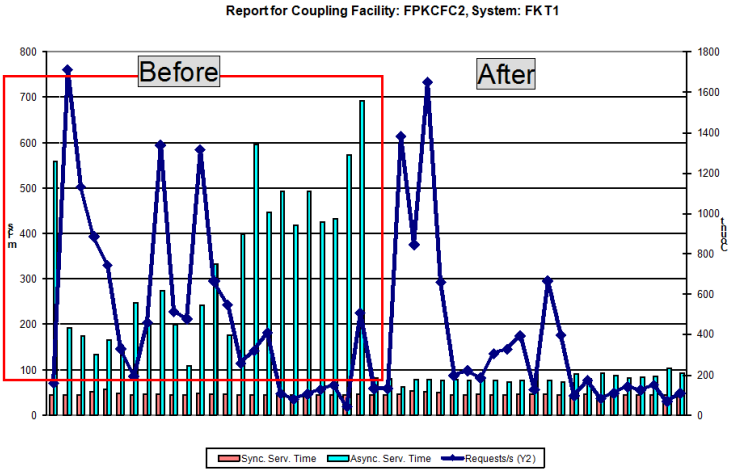


© Copyright Watson & Walker 2014,2015

Coupling Thin Interrupts

Benefit #1 (PR/SM / LPAR Level):

- If z/OS is using a shared engine and either has a low weight or is not very busy (this example is a test z/OS in prod plex), the existence of an outstanding interrupt can decrease the time that the Logical CP has to wait to get dispatched again.

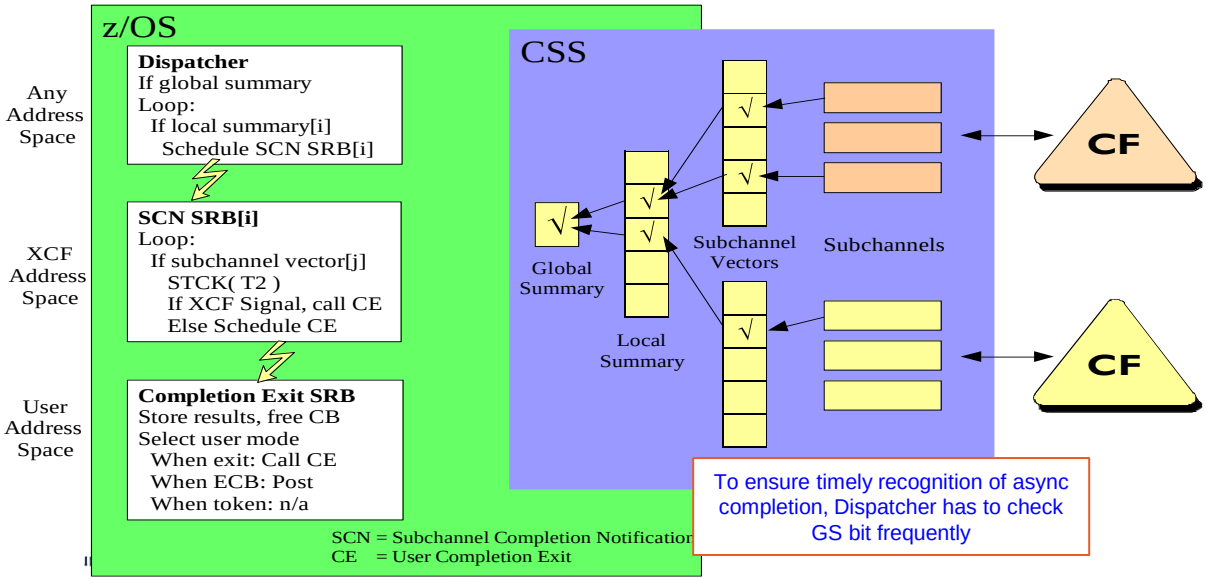


© Copyright Watson & Walker 2014,2015

Coupling Thin Interrupts

Now that we are dispatched, need to realize that there is a signal waiting to get processed.

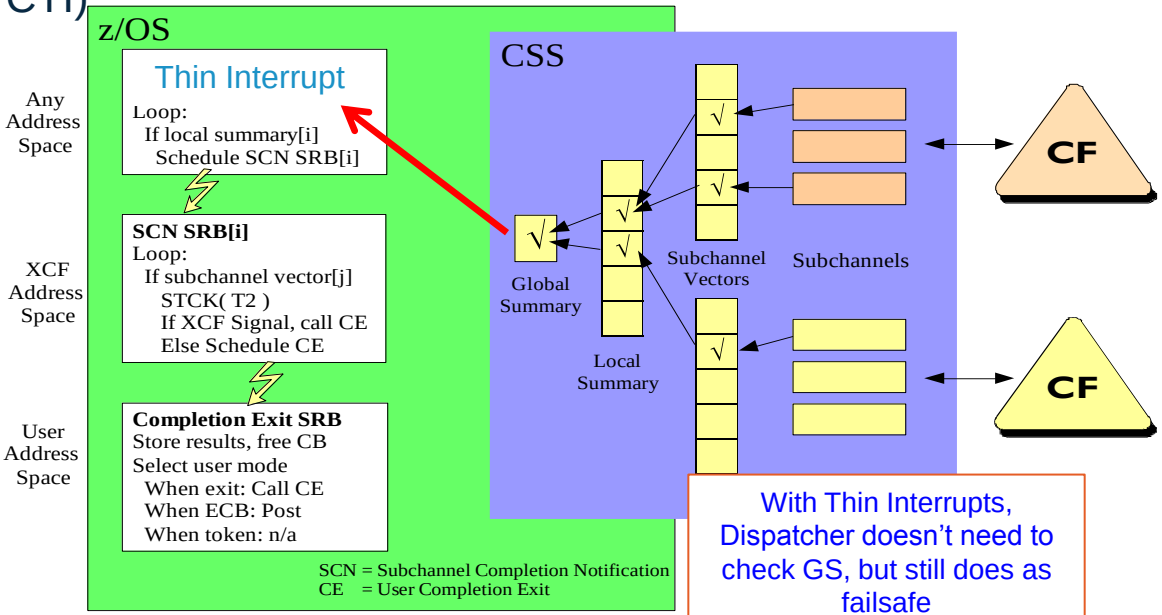
Processing for asynch response or CF notification (**Before** CTI)



© Copyright Watson & Walker 2014,2015

Coupling Thin Interrupts

Processing for asynch response or CF notification (**After** CTI)

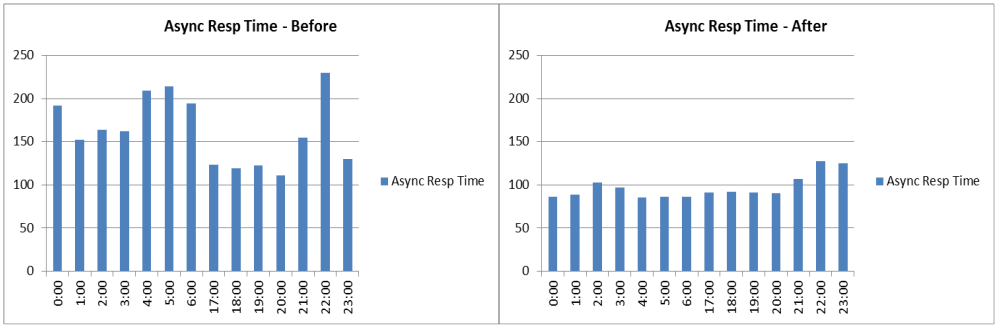


© Copyright Watson & Walker 2014,2015

Coupling Thin Interrupts

Benefit #2 (z/OS / XES Level):

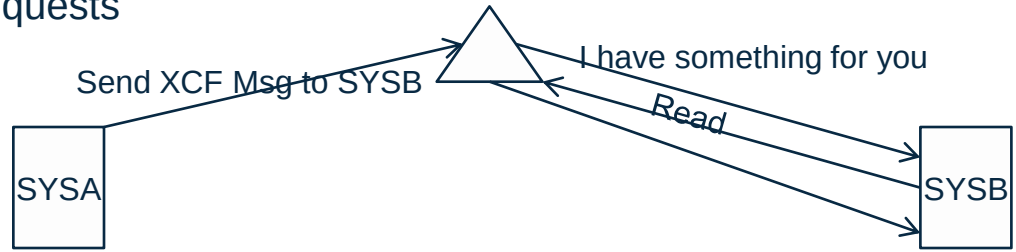
- Because the arrival of a CF signal generates an interrupt, you no longer need to wait for the dispatcher to check the Global Summary bit.
  - Reduces interval between when signal arrives from CF and when XES gets called to process it.
  - Delivers more consistent response times because interrupt will be processed more or less immediately rather than having to wait a variable amount of time for the dispatcher to get control and check Global Summary bit



Coupling Thin Interrupts

Benefit #2 (z/OS / XCF Level):

- This doesn't only apply to the response to asynchronous requests



- XCF system-to-system signaling observes shorter end-to-end times
- Shared Message Queue exploiters (IMS and MQ) are more responsive

© Copyright Watson & Walker 2014,2015

Coupling Thin Interrupts

- z/OS exploitation of Coupling Thin Interrupts is automatically enabled as long as z/OS is running on the required hardware and software level.
- You can *display* status using D XCF,C command:

```
D XCF,C
IXC357I  22.30.24  DISPLAY XCF 549
...
   OPTIONAL FUNCTION STATUS:
     FUNCTION NAME                  STATUS      DEFAULT
     DUPLEXCF16                     DISABLED    DISABLED
     SYSSTATDETECT                  ENABLED     ENABLED
     USERINTERVAL                   DISABLED    DISABLED
     CRITICALPAGING                 ENABLED     DISABLED
     DUPLEXCFDIAG                   DISABLED    DISABLED
     CFLCRMGMT                      ENABLED     DISABLED
     COUPLINGTHININT                ENABLED     ENABLED
```

- You can *control* it using FUNCTIONS parm in COUPLExx
- You can *change* it dynamically using SETXCF FUNCTIONS command

© Copyright Watson & Walker 2014,2015

Coupling Thin Interrupts

- Prerequisites – for **z/OS exploitation** of Coupling Thin Interrupts:
  - z/OS must be running on zEC12 GA2 or zBC12 GA1
    - z/OS LPAR can be using shared *or* dedicated engines
    - All CF link types are supported
  - z/OS V2.1
  - z/OS V1.13 with APARs OA38734, OA37186, OA38781, OA42682
  - z/OS V1.12 with APARs OA38734, OA37186, OA38781, OA42682
  - There are **NO** service or CF Level requirements for connected CFs – can be *any* supported CF Level, running on *any* supported CEC, connected by *any* supported link type including ICP (microcode) links.
  - Scope is single system – you can enable it on *none*, *some*, or *all* systems in the sysplex.

© Copyright Watson & Walker 2014,2015

Coupling Thin Interrupts

- Summary for the z/OS end of CTI:
  - As long as you are on a CPC with Driver 15 or later, and running z/OS 1.12 or later with the required fixes, z/OS will AUTOMATICALLY use Coupling Thin Interrupts
  - The profile of systems that are most likely to benefit are:
    - Systems where asynch response times are a LOT higher than synchronous response times.
    - Systems where asynch response times are significantly greater than other systems in the same sysplex.
    - LPARs with shared engines and low weights.
    - LPARs with large variances in asynch response time over different times of day.

© Copyright Watson & Walker 2014,2015

Coupling Thin Interrupts

- We said that both z/OS and CFs can exploit Coupling Thin Interrupts, but also that the considerations are different for the two.

- Why is CF different?

© Copyright Watson & Walker 2014,2015

Coupling Thin Interrupts

- 1) z/OS systems have millions of things to do – processing CF signals is just one of them – hence the old model where the MVS Dispatcher would only check the Global Summary bit every so often.
- CFs, on the other hand, ONLY process CF requests. To do that as quickly as possible, Coupling Facility Control Code basically spends its time either processing a request, or looking in the link buffers for the next request.
- IF the CF LPAR is always dispatched (that is, it has a dedicated engine), it will very quickly detect the pending request in the link buffers. Therefore, generating an interrupt would not provide any response time benefit in that situation.
- As a result, Coupling Thin Interrupts only make sense and can *only* be enabled on CFs with shared engines.

© Copyright Watson & Walker 2014,2015

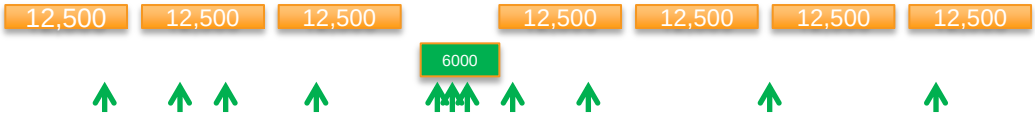
Coupling Thin Interrupts

- 2) CFs role in life is to deliver the best response time it can – let's say 5 mics.
- For a CF using a shared engine, it knows that when it loses the engine, it will probably be waiting THOUSANDS of mics before it is dispatched again – meaning that requests that arrive during that time will have to wait a LONG time until the CF LPAR is dispatched again. In an attempt to avoid this delay, CFCC used to try to hold on to the engine as long as possible:
  - For a CF LPAR running with DYNDISP OFF, CFCC will never release the engine until PR/SM takes it away.
  - For a CF LPAR running with DYNDISP ON, CFCC will finish all its work, then hang on for a while longer (hoping that some more work shows up) then finally go to sleep and release the engine back to PR/SM.
- This aversion to releasing the engine was based on the fact that there was no interrupt mechanism for CF signals.

© Copyright Watson & Walker 2014,2015

Coupling Thin Interrupts

- So how does Coupling Thin Interrupts change things?
- 1) Because Coupling Thin Interrupts provide an interrupt mechanism, requests to a CF that is not currently dispatched will now be observed and processed much sooner.



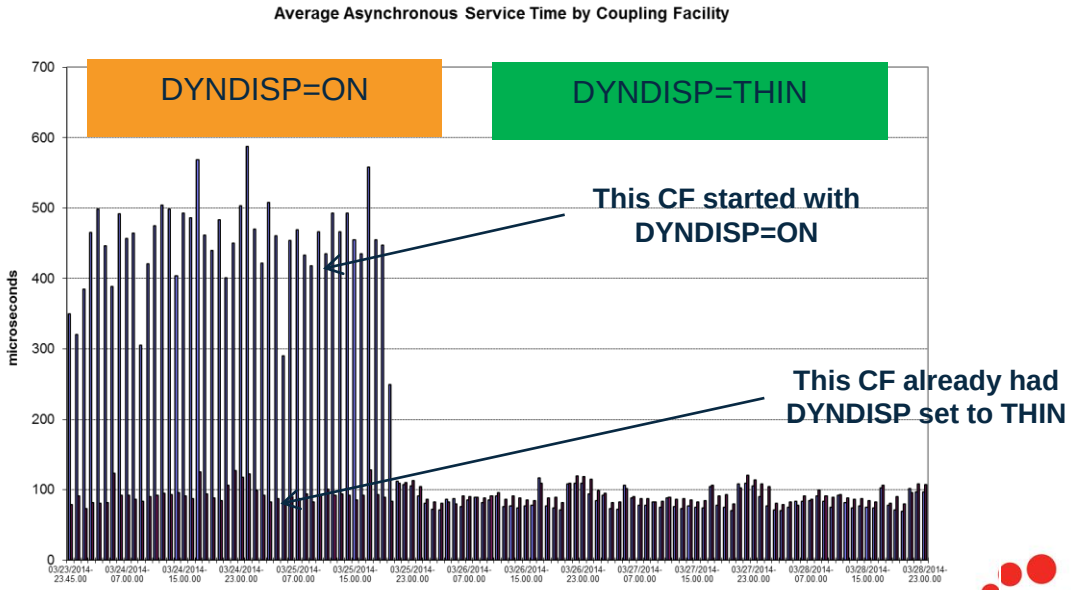
Prod(OFF)
Tst(ON)

Prod(OFF)
Tst(THIN)

© Copyright Watson & Walker 2014,2015

Coupling Thin Interrupts

- How enabling Thin Interrupts impacts response times



© Copyright Watson & Walker 2014,2015

Coupling Thin Interrupts

What else? Does it make the coffee?

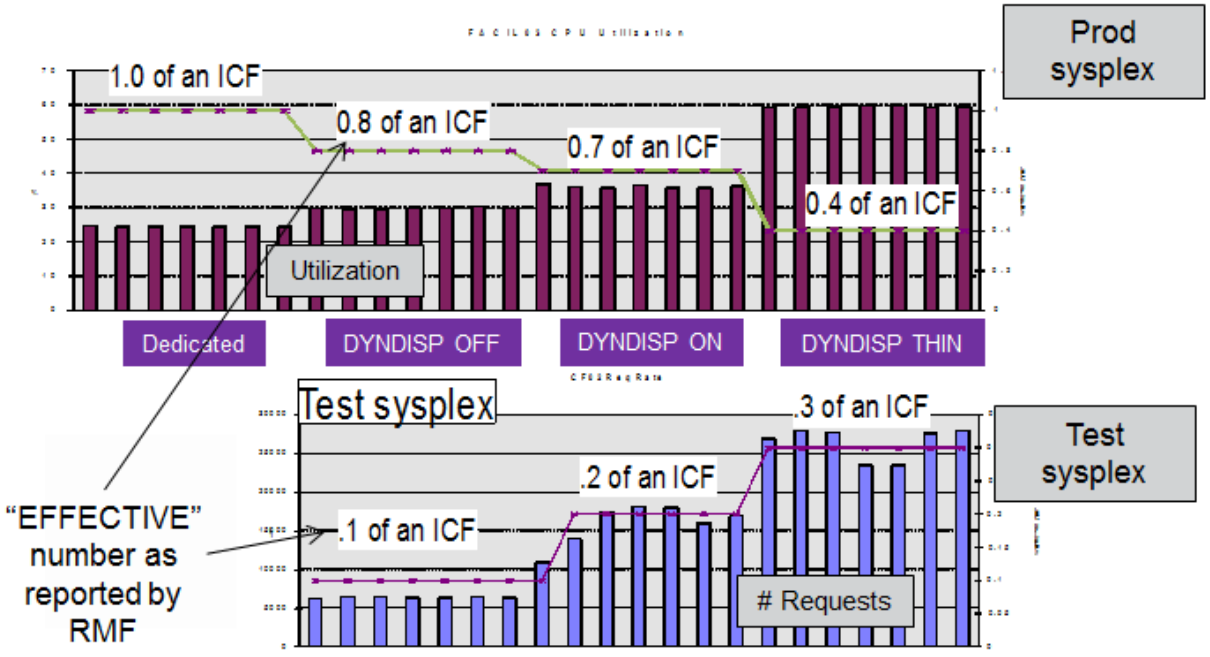
2) Because CFCC knows that it will be able to react to newly arriving requests much sooner when DYNDISP is set to THIN, CFCC places the logical ICF PU in wait state much sooner when there are no requests to process. As a result, the physical ICF is given back to PR/SM and may then be used by another logical ICF PU.. This converts CFCC from being a CPU hog into being a good neighbor (for anyone sharing an engine with it).

(coffee maker is an optional feature – please submit an RPQ and loosen up your checkbook…)

© Copyright Watson & Walker 2014,2015

Coupling Thin Interrupts

This shows the impact of different configuration options on the CF that is being changed (Prod) **and** on the CF it is sharing the engine with (Test)



© Copyright Watson & Walker 2014,2015

Coupling Thin Interrupts

This illustrates a very important point about the impact of Coupling Thin Interrupts on CFs – they don't only impact the CF LPAR where they are enabled, they ALSO impact any other CF LPARs that they are sharing engines with.

It is natural when you make a change to monitor the thing you changed – when investigating the impact of Coupling Thin Interrupts, you also have to monitor OTHER sysplexes (which means a separate set of RMF reports).

Implementation is easy, monitoring the impact is the hard part!

© Copyright Watson & Walker 2014,2015

Coupling Thin Interrupts

- Whatever type of CEC is at the other end of the CF link is irrelevant.
  - Could be CEC that supports Coupling Thin Interrupts or one that doesn't – makes no difference.
- It is irrelevant whether Coupling Thin Interrupts are *turned on* on the z/OS LPAR.
  - The system that sends the signals has *no role* to play in whether an interrupt is generated when that signal reaches the target LPAR.
  - Whether the hardware generates an interrupt is COMPLETELY under the control of the "operating system" running in that LPAR.

© Copyright Watson & Walker 2014,2015

Coupling Thin Interrupts

- Prerequisites – for CF exploitation of Coupling Thin Interrupts:
  - CF must be running on CPC with Driver 15 or later.
  - Coupling Thin Interrupts work with *any* type of CF link – ISC, ICP, PSIFB (1X/12X/IFB Mode/IFB3 Mode), ICA.
  - CF LPAR *must* be using shared engines – DYNDISP command is not accepted in CF LPARs with dedicated engines.
  - Coupling Thin Interrupts must be explicitly turned on for that LPAR using the DYNDISP THIN command.
    - Unlike z/OS LPARs, Coupling Thin Interrupts are NOT automatically enabled in CF LPARs.
  - A Swiss bank account to hold all the awards and bonuses you will get after implementing this.

© Copyright Watson & Walker 2014,2015

Coupling Thin Interrupts

- Recommendations:
  - For CFs that are using DYNDISP ON today, we recommend that you switch to DYNDISP THIN.
  - If your CFs are using DYNDISP OFF today, we recommend that you at least try DYNDISP THIN (you can switch back and forth non-disruptively).
  - If your production CF has a dedicated engine AND runs at extremely low utilizations (peak <10%), you MIGHT consider testing it with a shared engine and DYNDISP THIN before the next time you upgrade the CF CPC.
    - You might find that you can get acceptable response times without needing a dedicated engine.
    - But you can only do this if at least one of your CFs is already in a zEC12/zBC12.
    - And changing your CF engine from dedicated to shared and back requires the CF LPAR to be deactivated/reactivated, so is somewhat disruptive

© Copyright Watson & Walker 2014,2015

Coupling Thin Interrupts

- References:
  - Setting Up a Sysplex
  - PR/SM Planning Guide (for EC12 or later)
  - Excellent IBM White Paper 102400 – '*Coupling Thin Interrupts and Coupling Facility Performance in Shared Processor Environments*' by **Barbara Weiler**

© Copyright Watson & Walker 2014,2015

Modifyable Sync/Async Thresholds

- Coupling Thin Interrupts can deliver reduced CF response times in certain situations.
  - In many cases, they might be sufficient to turn unacceptable response times into acceptable ones.
- However, there may still be situations (most likely in sysplexes with a large number of CF requests) where some fine tuning may deliver better overall average response times, or possibly even some z/OS CPU savings.
  - The SYNCASYNC function provides a mechanism to apply this fine tuning…

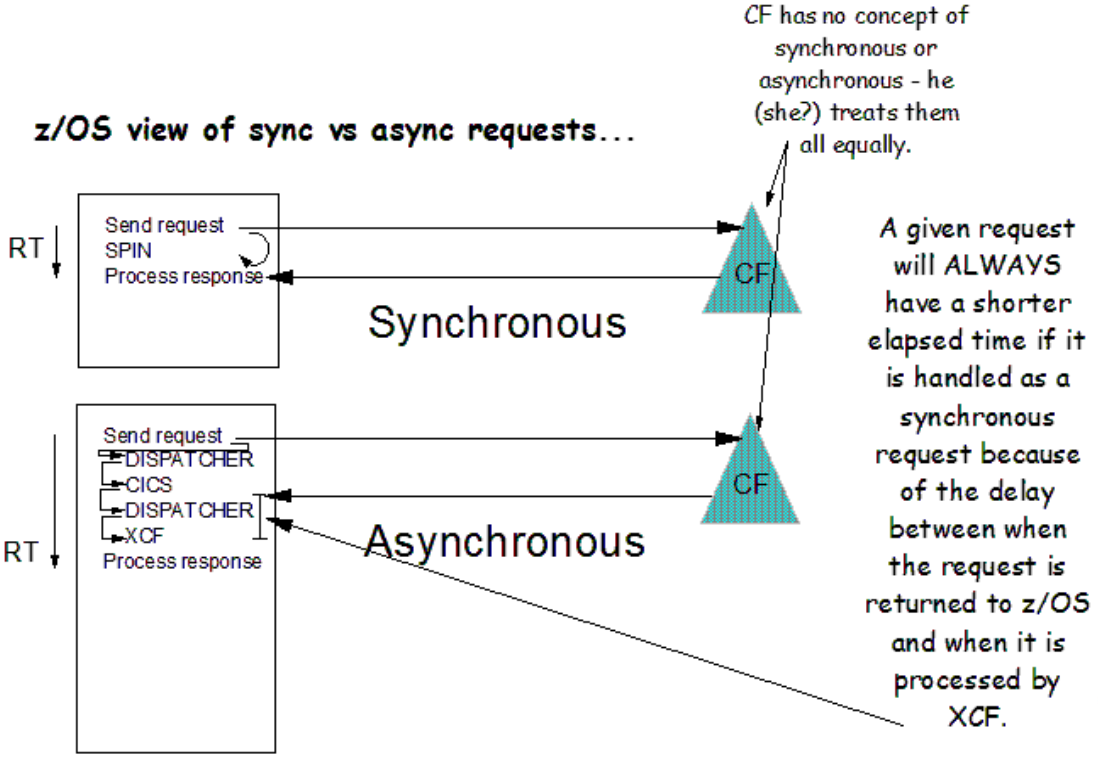
© Copyright Watson & Walker 2014,2015

Modifyable Sync/Async Thresholds

- z/OS 2.1 (and rolled back to z/OS 1.12 and 1.13) lets you control the thresholds used by the XES heuristic algorithm. This ability is called SYNCASYNC.
- To understand what this means and why you should be interested, we will cover:
  - Some background - What are sync and async requests? What IS the "heuristic algorithm"?
  - How to determine if this capability would be valuable to you.
  - How to implement this new capability.

© Copyright Watson & Walker 2014,2015

Modifyable Sync/Async Thresholds

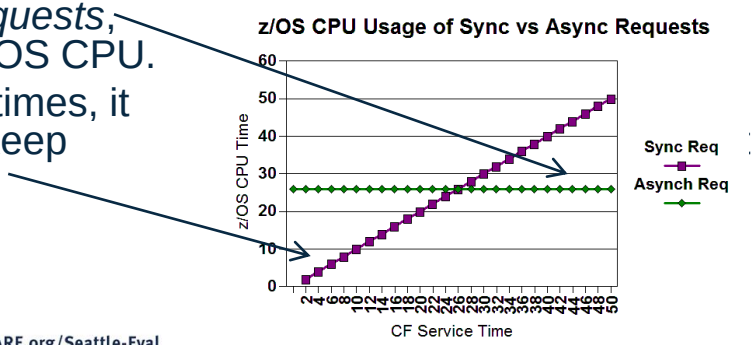


© Copyright Watson & Walker 2014,2015

Modifyable Sync/Async Thresholds

- Because XES consumes z/OS CPU until the response to a sync request is received back from the CF, a very-long-running synchronous CF request can consume a lot of z/OS CPU.
- On the other hand, because XES doesn't spin while an *asynchronous* request is running, the z/OS cost of an asynchronous request (in terms of the number of instructions executed) is more or less fixed.

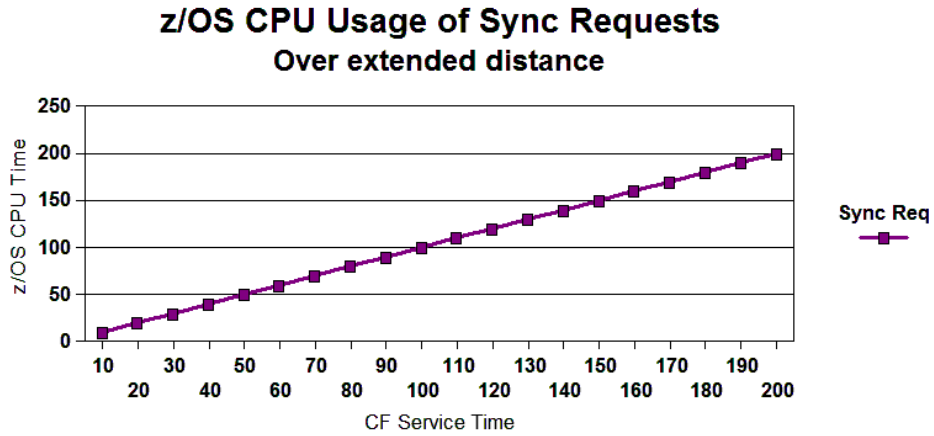
- For *long-running requests*, asynch uses less z/OS CPU.
- For short response times, it is more efficient to keep as synch.



© Copyright Watson & Walker 2014,2015

Modifyable Sync/Async Thresholds

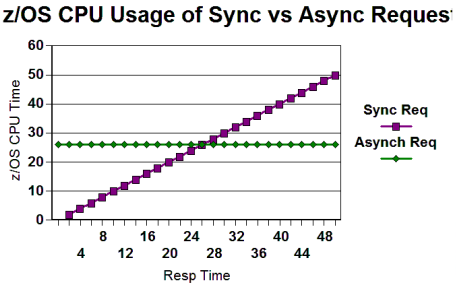
- When customers started implementing multi-site sysplexes, the CF service time, and therefore the z/OS CPU consumption, of synchronous requests started increasing dramatically (roughly 10 mics per km.)....



© Copyright Watson & Walker 2014,2015

Modifyable Sync/Async Thresholds

- To protect systems from excessive CPU consumption by long-running synchronous requests, z/OS 1.2 introduced a heuristic algorithm. The objective of the algorithm was to handle CF requests as efficiently as possible from a z/OS CPU consumption perspective.
- If the *expected* CF service time (based on a rolling average of sync response times) for a request is less than the z/OS CPU time required to handle an async request (the green line), the request would be sent synchronously.
  If the expected response time was higher, the request would be sent asynchronously.
- This was all automagical and you had no control over it.

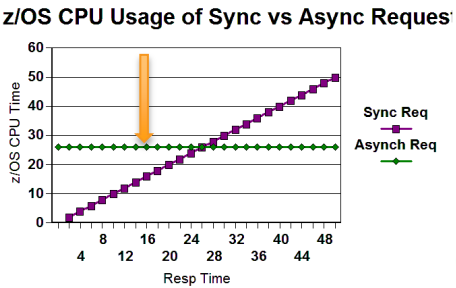


Complete your session evaluations online at www.SHARE.org/Seattle-Eval

© Copyright Watson & Walker 2014,2015

Modifyable Sync/Async Thresholds

- The elapsed time to process the 4 task switches associated with an asynchronous request is related to the CPU speed - a slower CPU will take more time to process x instructions than a faster one.
- So, as CPUs got faster, the amount of time required to complete the "fixed" number of instructions associated with asynchronous requests got shorter and shorter and shorter.....
- The threshold used by the heuristic algorithm is determined by the elapsed time to perform the task switches, so as the CPU speed increased, the threshold kept getting lower



© Copyright Watson & Walker 2014,2015

Modifyable Sync/Async Thresholds

- If this were to continue, soon most requests would be processed asynchronously. This is not good for overall performance, and some CF exploiters don't like having their synchronous requests converted to asynchronous ones.
- To protect from this situation, APAR OA21635 adjusted the algorithm to set a minimum threshold of 26 mics - any request expected to take longer would be converted to async, any expected to take less would be left as synchronous.
- The threshold might still be higher on older or slower CECs. Use the D XCF,C command to display the threshold for *your* system:

```
D XCF,C
   INTERVAL    OPNOTIFY      MAXMSG     CLEANUP       RETRY    CLASSLEN
        165         168        2000          15          10         956
...
   SYNC/ASYNC CONVERSION    THRESHOLD    -SOURCE-    DEFAULT
              SIMPLEX              26     SYSTEM      IN USE
              DUPLEX               26     SYSTEM      IN USE
         LOCK SIMPLEX              26     SETXCF      IN USE
         LOCK DUPLEX               26     SYSTEM      IN USE
```


© Copyright Watson & Walker 2014,2015

Modifyable Sync/Async Thresholds

- Customers have various reasons for wanting control over the synchronous/asynchronous threshold:
  - If the threshold is just below your median synchronous service time, you could increase the percent of requests being processed synchronously by increasing the threshold by a small amount.
  - The accounting of used z/OS CPU time is handled differently for a synchronous CF request than for an asynchronous one. Having requests switching back and forth between synchronous and asynchronous can make chargeback more complex.
    - High percentages of asynchronous requests impact capture ratio. In a measurement where the only workload was driving CF requests, capture ratio was 98% for pure sync workload vs. 63% for pure asynchronous.
  - Changing CPU speed, either because of a technology change or a change in the number of engines in the LPAR, can cause the threshold to change, meaning that the balance between sync and async requests, and the overall average response time, can change when you change the CPU or LPAR config.
  - Especially for lock requests, a change from short synchronous response times to longer, asynchronous, ones can impact certain workloads.

© Copyright Watson & Walker 2014,2015

Modifyable Sync/Async Thresholds

- What to look for in an RMF report

Is this a structure that you care about?

Are some systems getting a higher % of Sync?

```
 STRUCTURE NAME = DB2P_LOCK1          TYPE = LOCK   STATUS =
          # REQ  -------------- REQUESTS --------------
SYSTEM    TOTAL                #      % OF  -SERV TIME(MIC)-
NAME      AVG/SEC              REQ     ALL     AVG   STD_DEV

FPKA      33966    SYNC         2K     0.1    26.3     928.1
          566.1    ASYNC       32K    20.3    70.9     988.0
                   CHNGD        0     0.0  INCLUDED IN ASYNC
                   SUPPR        0     0.0

FPKB       123K    SYNC       117K    74.7    21.9     247.2
          2050     ASYNC     5680     3.6    64.6    1474.6
                   CHNGD        0     0.0  INCLUDED IN ASYNC
                   SUPPR        0     0.0

-------------------------------------------------------------

TOTAL      157K    SYNC       119K   74.8     21.9     391.5
          2616     ASYNC       38K   23.9     69.9    1119.9
                   CHNGD        0     0.0
                   SUPPR        0     0.0
```

Increasing the threshold on system FPKA *might* result in a higher % of sync requests

© Copyright Watson & Walker 2014,2015

Modifyable Sync/Async Thresholds

- z/OS 2.1 (and rolled back to z/OS 1.12 and 1.13) provides new COUPLExx keywords to let you override the thresholds if you wish.
- In the COUPLExx Parmlib member, there is a new statement:
  - SYNCASYNC keyword(value) keyword(value) .....

```
COUPLE SYSPLEX(&SYSPLEX.)
       PCOUPLE(SYS1.&SYSPLEX..SYSPLEX.CDS01)
       ACOUPLE(SYS1.&SYSPLEX..SYSPLEX.CDS02)
       SYNCASYNC SIMPLEX(DEFAULT) DUPLEX(40)

        D XCF,C
           INTERVAL    OPNOTIFY      MAXMSG     CLEANUP       RETRY    CLASSLEN
                165         168        2000          15          10         956
           SSUM ACTION  SSUM INTERVAL  SSUM LIMIT     WEIGHT  MEMSTALLTIME
               ISOLATE              0         900          50           300
            CFSTRHANGTIME
                      900
                                    ...

            SYNC/ASYNC CONVERSION    THRESHOLD   -SOURCE-   DEFAULT
                       SIMPLEX            26     PARMLIB    IN USE
                        DUPLEX            40     PARMLIB        26
                                          26     SYSTEM     IN USE
                                          26     SYSTEM     IN USE
```

© Copyright Watson & Walker 2014,2015

Modifyable Sync/Async Thresholds

- You can also change the thresholds dynamically using the SETXCF MODIFY,SYNCASYNC,keyword=value command:
  - Where “keyword” is one of the following:
    - SIMPLEX - for simplex list and cache requests
    - DUPLEX - for duplexed list and cache requests
    - LOCKSIMPLEX - for simplex lock requests
    - LOCKDUPLEX - for duplexed lock requests
  - And “value” can be:
    - Numeric value in range 1 to 10000 (microseconds)
    - DEFAULT – to use the system determined threshold value

- Note that each system in the sysplex could potentially have different thresholds, and the scope of the SETXCF MODIFY,SYNCASYNC command is also a single system.

© Copyright Watson & Walker 2014,2015

Modifyable Sync/Async Thresholds

- Before you change any thresholds, remember that the default thresholds were designed to optimize z/OS CPU usage by CF requests.
  - Reducing the thresholds will result in more requests being processed asynchronously (and therefore getting longer response times). Impact on z/OS CPU is difficult to predict (although such a change will definitely move reporting of some of the z/OS CPU time from requestor address space to XCFAS).
    - However, CPU speeds have improved considerably since the current minimum threshold of 26 mics was established – on full speed z196 and zEC12, that threshold might be significantly higher than the actual z/OS CPU cost of an async request…..
  - Increasing the thresholds may result in some more z/OS CPU usage in return for better overall average CF response times.
  - Before you change the thresholds, suggest that you enable Coupling Thin Interrupts in z/OS to see if that can reduce async response times to an acceptable level.
- Remember that some requests (XCF, DB2 Secondary GBPs, for example) are designed to be asynchronous requests and will never be synchronous, regardless of the value of the thresholds.

© Copyright Watson & Walker 2014,2015

Modifyable Sync/Async Thresholds

- Prerequisites:
  - Hardware:
    - Any System z CEC supported by z/OS 1.12 or later
    - No dependencies on any particular CF Level or CF CEC type
  - Software:
    - z/OS 2.1 (in the base code)
    - z/OS 1.13 + PTF UA69637
    - z/OS 1.12 + PTF UA71120

© Copyright Watson & Walker 2014,2015

Modifyable Sync/Async Thresholds

- Powerful capability to fine tune performance of your CF workloads
  - Rolled back to z/OS 1.12
  - Works with *any* CF Level supported by z/OS 1.12 or later
- Recommend to monitor SMF Type 113 records and see if changing the threshold has any impact on Relative Nest Intensity
  - EVERYONE should have HIS turned on and be collecting Type 113 records all the time, irrespective of this new function. If you don't have any other tool to process the Type 113 records, you can use CP3KEXTR and zPCR.
- For most customers, probably not necessary to adjust the thresholds
- Use with care and monitor RMF reports to understand impact

© Copyright Watson & Walker 2014,2015

Coupling Thin Interrupts

If you liked this session, here are some others you might like:

- 16813 - *Coupling Technology Overview and Planning - What's the Right Stuff for Me?* - Gary King
- 16831 - *RMF and Coupling Facility Health* – Brad Snyder

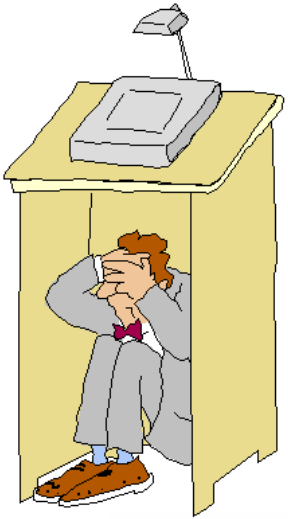
- 17154 - *SMFPRMxx Parameters - Which can Help; Which can Hurt* – Cheryl Watson, Frank Kyne
- 16461 - *The Cheryl and Frank zRoadshow* – Cheryl Watson, Frank Kyne

- Also, if you like SMF data (and who doesn't??!!), please see our new AND IMPROVED(!) *SMF Reference Summary* at www.watsonwalker.com/references.html

© Copyright Watson & Walker 2014,2015

Frank Kyne
Editor and Technical Consultant
Watson and Walker

Any questions?



© Copyright Watson & Walker 2014,2015

Frank Kyne
Editor and Technical Consultant
Watson and Walker

Thank you for coming

Please remember to complete an evaluation
Session number is 16644

Complete your session evaluations online at www.SHARE.org/Seattle-Eval

© Copyright Watson & Walker 2014,2015