z/OS Heuristic Conversion of CF Operations from Synchronous to Asynchronous Execution (for z/OS 1.2 and higher) V2

z/OS 1.2 introduced a new heuristic for determining whether it is more efficient in terms of the z/OS host ("sender") system's CPU utilization, to issue a synchronous command to the coupling facility synchronously or asynchronously. With this support, referred to simply as "the heuristic" throughout the remainder of this section, z/OS will automatically and dynamically control the issuing of CF requests either synchronously or asynchronously, as needed, so as to optimize CPU consumption related to CF requests on the sender z/OS system.

As background, z/OS has the ability to issue requests to the coupling facility either synchronously (i.e. the processor issuing the request waits or spins until the operation is complete at the coupling facility) or asynchronously (i.e. the processor issuing the request is freed up to run another task rather than synchronously waiting or spinning, while the command is transferred to and executed at the coupling facility).

In general, asynchronous CF operations take somewhat longer to complete than synchronous operations (i.e. their elapsed time or "service time" is longer than a similar request issued synchronously), and they require more software processing to complete the request once the CF has completed the asynchronous execution of the command. Furthermore, asynchronous requests tend to impact the efficiency of the underlying host processor hardware due to context switching between tasks, scheduling of units of work to process the back-end completion of the asynchronous CF request, suspending and resuming the unit of work initiating the CF request, and other "overhead" associated with the asynchronous processing.

However, despite all the additional costs associated with asynchronous request processing, from a total CPU capacity impact standpoint, a sufficiently long-running *synchronous* CF operation will eventually cost *more* than the same operation processed in an asynchronous fashion, simply due to the "opportunity cost" of the processor having to wait or spin for a long time while the CF operation is occurring. The faster the z/OS processor issuing the CF request is, the greater this "opportunity cost" appears to be, for a given CF service time, because a faster processor might have done more work than a slower processor during the time spent waiting for the CF request to complete. And of course, the longer the CF operation is likely to take, the greater this "opportunity cost" in terms of work the processor might have been able to execute during the execution of the CF request, becomes.

Therefore, as the processor executing the CF request gets faster, or as the CF operations themselves take longer (for any reason), it becomes less attractive to perform CF operations synchronously, and more attractive to perform those operations asynchronously. At some threshold, a "crossover point" is reached where issuing the

operation asynchronously costs less in terms of sender CPU overhead than issuing the same operation synchronously would have cost.

In releases prior to z/OS 1.2, z/OS used a relatively simple and static algorithm to try to manage this tradeoff between the efficiency of synchronous and asynchronous CF operations. In those releases, lock structure requests were *never* converted to asynchronous execution, and list or cache requests were converted from synchronous to asynchronous execution based on some simple rules of thumb, including:

- Certain intrinsically long-running types of CF commands, for example, commands to scan through large sets of entries within the structure, were always converted to asynchronous.
- CF commands involving large (e.g. >4K bytes) data transfers to or from the CF were always converted to asynchronous.
- Particular combinations of processor types for the sending z/OS system and the receiving CF were recognized and singled out for additional opportunities for conversion to asynchronous.

While these simple rules of thumb somewhat accomplished their intended purpose, they did not handle some very important conditions which were specific to certain configurations or workloads, for example:

- Geographically dispersed parallel sysplex (GDPS), in which a CF might be located a considerable distance from the sending z/OS processor. A particular CF request might complete in a short time if executed on a nearby CF, but would take much longer if sent to a distant CF at the other GDPS site, due to speed-of-light latencies. In this case, it could be more efficient to convert the requests to the distant CF to asynchronous execution, but still drive similar CF requests to the local CF synchronously.

- Highly variable workloads, in which fluctuations in the CF workload (and hence, the CF processor utilization) cause fluctuations in the synchronous CF service time, independent of the inherent processor speed of the CF.

- Low-weighted CF partition with shared CPUs, which tends to greatly elongate the synchronous CF service times, independent of the inherent processor speed of the CF. A CF with shared processors will tend to yield good service times during those intervals when the CF is actually dispatched on a shared processor, but will tend to yield *very* poor service times during intervals where the CF is NOT dispatched on a shared processor, and thus it cannot possibly service the request.

In z/OS 1.2 and higher, a much more sophisticated heuristic was provided to explicitly recognize the “crossover point” between the cost of synchronous and asynchronous CF request processing, in a general manner for all combinations of processor configurations, CF link technologies, types of workloads, ranges of CF utilizations and other variations in CF responsiveness, distance-related latencies, etc.

The heuristic drives requests to the CF in the appropriate synchronous or asynchronous execution mode, based on the actual observed service times for similar requests. At the same time, CF lock structure requests were also enabled for asynchronous execution based on the heuristic, similar to list and cache structure requests.

The heuristic dynamically monitors actual observed synchronous CF request service times, at a fine granularity, for each type of CF command, on a per-CF basis (and on a per-pair-of-CFs basis for those CFs participating in System-Managed CF Structure Duplexing). This monitoring also takes into account the amount of data transfer requested on the operation, and several other request-specific operands which can significantly influence the service time for the request. These observations are then recorded in a table, organized by request type and the other factors described above, in which z/OS maintains a moving weighted average synchronous service time for each specific category of request. This moving, weighted average is biased towards recent history, so z/OS can react responsively to factors suddenly affecting a CF's performance, such as a sudden workload spike.

The heuristic then compares these actual observed synchronous service times against dynamically calculated thresholds, in order to determine which kinds of operations would be more efficient if they were to be executed asynchronously. z/OS calculates several different thresholds for conversion – reflecting the fact the relative costs of asynchronous processing for list, lock, and cache requests are not the same, nor are the relative costs of asynchronous processing for simplex and duplexed requests the same. All of the calculated thresholds are then normalized based on the effective processor speed of the sending processor (which in turn incorporates information about both the intrinsic processor speed of the machine, and multiprocessing effects based on the number of CPs online for the z/OS image at the time), to accurately reflect the "opportunity cost" of synchronous execution for the processor on which z/OS is executing. The heuristic and the calculation of these conversion thresholds are not externally adjustable in any way.

As CF requests are submitted for processing, the characteristics of each request are examined to determine the recently-observed performance of similar requests. If z/OS determines similar requests have been experiencing good performance (i.e. the moving weighted average service time of those requests is below the calculated threshold for conversion), then z/OS will direct the current request to be executed synchronously. If z/OS determines similar requests have been experiencing poor performance (i.e. the moving weighted average service time of those requests is above the calculated threshold for conversion), then z/OS will convert the current request to asynchronous execution if it is possible to do so.

Note z/OS will occasionally "sample" synchronous performance for CF requests even when the heuristic has determined it would be more efficient to perform the request asynchronously, by performing every N^{th} request of a given type synchronously. This mechanism ensures if some factor should change which results in the CF's performance improving, the improvement will be observed and acted on. Thus, the heuristic does not

convert 100% of the requests to asynchronous execution; a small percentage of requests are always issued synchronously.

The following are some other considerations to be aware of, relative to the z/OS heuristic:

- CF exploiters may explicitly request asynchronous execution of their requests. Such requests are not subject to heuristic conversion at all, z/OS simply drives them asynchronously as requested. These requests are NOT reported as CHANGED on RMF reports, and are reported as ASYNC requests.

- CF requests may be converted to asynchronous execution due to the lack of subchannel resources needed to process them. Such requests are converted to asynchronous and queued for later execution when a subchannel becomes available. The heuristic is not involved in this kind of conversion. These requests are reported as CHANGED on RMF reports, and are reported as ASYNC requests.

- CF requests may be converted to asynchronous execution due to serialized list or lock contention. The heuristic is not involved in such conversion. These requests are NOT reported as CHANGED on RMF reports. They undergo various forms of software processing related to contention resolution, and may in fact perform several individual CF accesses as contention management processing is performed, and those CF accesses may be performed in a CPU-synchronous fashion even though the request itself has become asynchronous with respect to the requestor.

- CF requests converted to asynchronous execution due to the heuristic are NOT reported as CHANGED on RMF reports, rather they are reported as ASYNC requests.

- As a result of the way CHANGED requests are counted and reported (as described above), a high percentage of CHANGED requests is indicative of a lack of subchannel resources and a corresponding lack of sufficient CF link connectivity to the CF in question. The CHANGED counter does not reflect anything related to conversion for other reasons, including the heuristic.

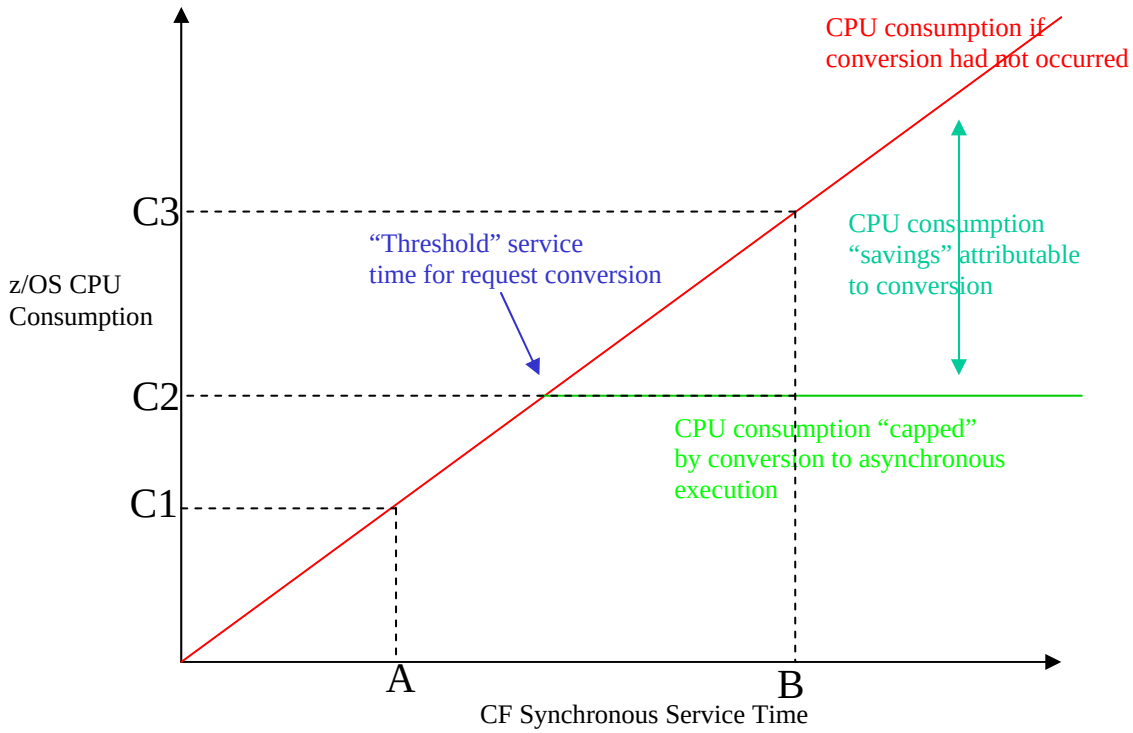
- When averaged over time and for a mixture of request types, heuristic conversion is seldom all-or-nothing. One generally sees a mix of synchronous and asynchronous requests over an interval of time; when the synchronous service time is high you will tend to see more asynchronous requests, and when the synchronous service time is low you will tend to see more synchronous requests. Generally, a small percentage of requests always remain synchronous, due to service time sampling.

- In a configuration where some CFs are local and some CFs are remote with respect to a given z/OS system, there will tend to be more conversion to asynchronous execution for requests going to the remote CF (because of the higher service time caused by distance latency). The farther away the CF is located, the more likely conversion becomes.

- When something in the configuration changes and more heuristic conversion takes place, it is normal to observe an increase in total CPU consumption for the workload. As synchronous service times get longer, CPU consumption inevitably increases, because each synchronous CF access takes longer and therefore causes the processor to wait/spin for a longer time. When service times get long enough, however, the heuristic will tend to "cap" this growth in CPU consumption by converting to asynchronous execution. Once the threshold is reached, further increases in synchronous service time will NOT result in further increases in CPU consumption, since the CPU cost of performing a CF operation asynchronously is fixed, and largely independent of the service time.

  So, when a configuration change (e.g. introduction of distance latencies into a sysplex previously located in one physical site) suddenly results in significant amounts of heuristic conversion to asynchronous execution, in general one sees an increase in CPU consumption on the sending z/OS systems – but the increase will be *less* than it otherwise would have been, had the conversion to asynchronous execution not taken place.

  Shown pictorially in the figure below, an increase in CF service time from A to B (crossing the service time threshold for conversion) will definitely result in an increase in CPU consumption (from C1 to C2) even with the heuristic; however, the increase is less than it would have been (C1 to C3) had the heuristic not "capped" the CPU utilization increase.



- Upgrades to faster z/OS processors may suddenly result in significant increases in the amount of heuristic conversion, since the calculated thresholds for conversion will be lower, the faster the sender processor is. The purpose of the heuristic is to *automatically and dynamically* react to such configuration changes, and adjust the flow of requests accordingly to so as to optimize z/OS CPU consumption.

- More generally, any configuration or workload changes affecting the observed CF request service times, or result in a change to the calculation of the conversion threshold, may result in substantial changes in the amount of heuristic request conversion observed. Examples of such changes include:
  - Changing to a faster or slower type of CF link technology.
  - Upgrading or downgrading a z/OS processor to a faster or slower technology, or changing the number of online processors in a z/OS image (which changes the effective processor speed somewhat).
  - Upgrading or downgrading a CF processor to a faster or slower technology.
  - Adding or removing CF processors from a CF image.
  - Changing the shared/dedicated nature of the CF processors, or, for a CF with shared processors, changing the LPAR weight of the CF partition, or enabling/disabling the use of Dynamic CF Dispatching.
  - Changes in the amount of distance latency occurring between the sending z/OS and the target CF.
  - Changes in the CF Level of the target CF.

    - Changes in CF utilization, caused by workload changes or other factors (such as placement of additional structures in a particular CF).

- z/OS CPU time accounting is affected by the heuristic conversion of CF requests. When a CF request is processed synchronously, generally all of the CPU time spent processing the request is attributed to the address space initiating the CF request. However, when some or all of these requests are processed asynchronously, the CPU time will get split among the address space initiating the CF request, and the XCFAS address space. This is because a significant portion of the back-end processing for an asynchronous CF request occurs in the XCF address space. Minor components of the CPU consumption may occur in other address spaces as well.

To summarize, at z/OS 1.2 and above, the heuristic for synchronous to asynchronous conversion of CF requests is capable of dynamically recognizing, in a very general and responsive way, situations in which z/OS host processor capacity/efficiency is being impacted by poor CF request synchronous service times, and taking the appropriate action by converting those requests to asynchronous execution, thereby "capping" the impact. The resulting higher elapsed time for the requests converted to asynchronous execution should, in general, have no noticeable effect on end-user response time for a given transaction processing workload.