Batchpipes for zOS is a separately orderable software product that can dramatically shorten business-critical batch processing. The BatchPipes for zOS solution addresses the **shrinking** of the daily batch window due to the demand for longer online availability.

BatchPipes for zOS offers a way to connect jobs so that data from one job can move through processor storage to another job without going to DASD or tape. It addresses a growing problem that faces installations with batch workloads such as insufficient time to complete that work. Given a batch jobstream with a data flow of certain characteristics, BatchPipes can dramatically shorten the elapsed time of the jobstream. The jobs can run faster and process larger volumes of data in the available batch window; therefore customers' processors are freed to run more work perhaps extending the period of time that interactive applications are available or supporting work in another time zone. In short, they might get more work from their processors.
BatchPipes, running on z/OS:

* Allows two or more jobs that formerly ran serially to run concurrently
* Reduces the number of physical I/O operations by transferring data through processor storage rather than transferring data to and from DASD or tape. This also alleviates workload to channels and SAP engines for other jobs to use.
* May reduce tape mounts and use of DASD real estate not requiring intermediate data sets.

What difference does parallel processing make to a customer's batch workload?

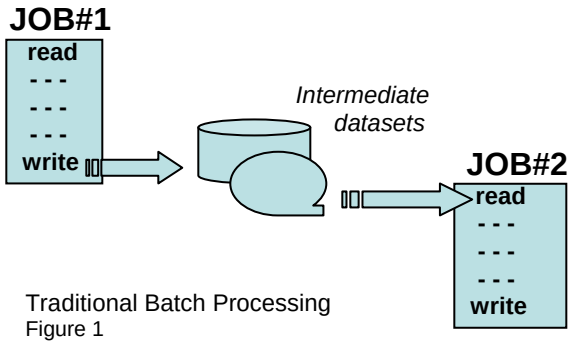
Large tightly-coupled processing systems running zOS can have many jobs in various stages of processing at one time. Batch applications have not traditionally taken advantage of this multiple processing capability. Rather, they often consist of multiple jobs that run serially one after another. With BatchPipes, the processor can run two or more jobs in a jobstream at one time. Jobs that once ran sequentially can now run in parallel because data records or blocks are available to the next job immediately after they are written. That is, the whole sequential file does not have to be written and then closed before the next job can access the file.

What difference does keeping data in the processor make to a customer's batch workload?

When data moves from one job to another through processor storage, the transfers take place in microseconds as compared to the milliseconds required to transfer data to and from tape or DASD. Keeping data in processor storage reduces the number of physical I/O operations, causes less I/O contention, and frees the device that holds the intermediate data set. An additional benefit of transferring data through processor storage is a reduction in the mounting and managing of tapes for applications using BatchPipes.

An Example of How BatchPipes Changes Traditional Batch Processing -

Most installations have batch job streams that use intermediate data sets such that output from one process (a job or step) is written to DASD or tape. This data is then available to be read by a second process immediately after the data set closes or the first process ends. Here shows the traditional job-to-job data flow:

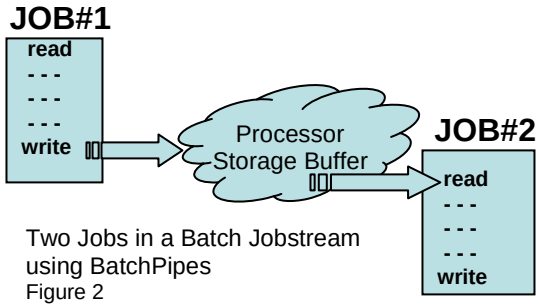


Traditional Batch Processing
Figure 1

Job1 writes data to an I/O device. When all the records have been written and the data set is closed, Job2 starts.

Job2 reads the data from the device. After Job2 finishes processing all the records, the data set is no longer required and can be deleted. In effect the transfer of data from one job to the other is at a data set level.

Compare the processing of the two jobs with figure 2, which shows those same two jobs using BatchPipes. Notice that external storage devices are absent, replaced by a processor storage buffer holding a small portion of the intermediate data set.



Two Jobs in a Batch Jobstream using BatchPipes
Figure 2

Job1, a **writer** to the processor storage buffer, and Job2 , a **reader** from that buffer, run concurrently.

Job2 can obtain data from the processor storage buffer as soon as Job1 writes the first block. Output from Job1 becomes immediately available as input to Job2. You can think of the data as "flowing" from Job1 to Job2. The processor storage buffer is called, in BatchPipes terms, a **pipe**, through which data flows, always in the same direction, from a writer job to a reader job. Writer-->pipe-->reader are known as a **pipeline**.

To understand the elapsed time savings, compare timelines of traditional job-to-job processing and BatchPipes job-to-job processing. The timeline for the traditional processing of Job1 and Job2 might look like this:

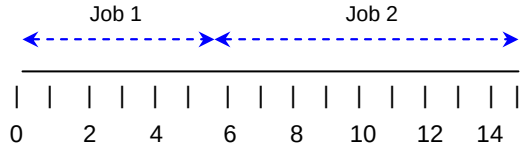


Figure 3 – Timeline showing a traditional two-job jobstream

With BatchPipes, the two jobs can run concurrently in **less** time than Job2 (the longer-running job) formerly needed. Elapsed time savings come from concurrent processing of the two jobs, from elimination of **I/O wait times** (time intervals during which a job waits for I/O transfers to and from DASD or tape), and from reduction of I/O transfer times between jobs.

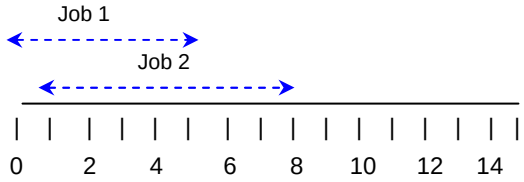


Figure 4 - Timeline Showing Two Jobs Using BatchPipes

Running the two jobs concurrently demonstrates a form of **parallelism**, the simultaneous processing of many tasks (or jobs) within one application. Figure 4 shows two jobs running concurrently in a multiprocessing environment. Without BatchPipes, the jobs would not run concurrently.

BatchPipes runs as a subsystem installed on zOS. If your customers plan to use BatchPipes, the writer and reader run concurrently; therefore, all resources for those jobs must be available for the specific time span.

IMPORTANT - BatchPipes offers large improvements to some applications, but not to all. Your customer will need to identify jobs to use BatchPipes. In some cases an approach to some extent will be how much work their application developers are willing to do to gain the BatchPipes improvements.

With the newer infrastructure, enabling HiperDispatch along with BatchPipes can potentially speed up nightly batch runs even more.

– – –