Data Set Buffering

Introduction

In IBM® InfoSphere™ DataStage® job data flow, the data is moved between stages (or operators) through a data link, in the form of virtual data sets. An upstream operator will process the records and deliver them as output. The downstream operator receives the output records from the upstream operator as input and processes the records. When the upstream operator's record producing rate matches with the downstream operator's record consuming rate, there will be smooth transition of data between the operators without any uncertainty.

In a few cases the producing and consuming rate of the operators differ. For example—if the upstream operator's record producing rate is higher than the downstream operator's record consuming rate, then the upstream operator has to hold the output records till the consuming operator reads the records.

What is a Deadlock?

When two or more upstream operators are processing data at the same time, and delivering output records to the same downstream operator, deadlock may result, forcing one or both upstream operators to stop its operations. See Figure 1.

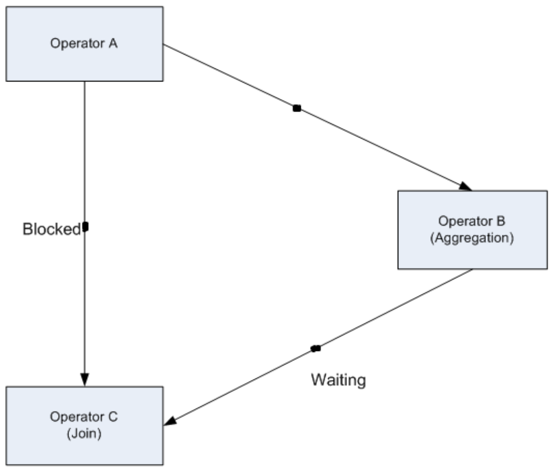


Figure 1: Deadlock Scenario

The above figure shows the fork join situation in the data flow. Operator A produces two output data sets, one data set goes to Operator B and the other data set goes to Operator C. Again, Operator B produces the output dataset after processing, and delivers to Operator C.

Operator C has to join the two input data sets from Operator A and Operator B; Operator C will not read records from Operator A until the output records from Operator B are produced. Operator C tends to block the data from Operator A, and will wait for the output records from Operator B. Eventually, these circumstances cause deadlock where one operator is not able to write its output and another operator is not able to read its input.

Buffer Operator

To avoid deadlock, a hidden buffer operator is automatically inserted into the DataStage job data flow by the parallel framework itself. See Figure 2 below.

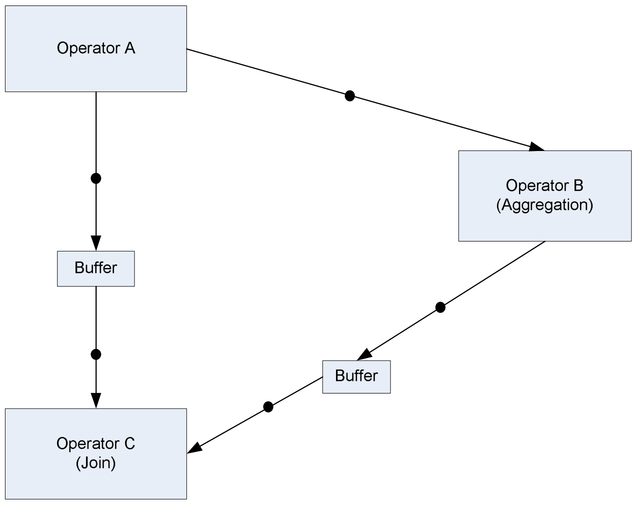


Figure 2: Buffering Mechanism to Avoid Deadlock

The buffer will temporarily hold the data— meaning, the producing operator writes the output to the buffer, then the consuming operator reads these output records from the buffer. Thus operators do not need to wait to write or read records from the other operators.

The buffer will hold data up to the default memory space allocated. If more capacity is needed, the data will be written to the scratch disk resource or the buffer disk pool defined in the system configuration file.

Notice the "buffer" operator (**parallel buffer (0)**) inserted in the DataStage log below.

Example:

```
It has 6 operators:
op0[1p] {(sequential Row_Generator_0)
on nodes (
ecc3671[op0,p0]
)}
op1[1p] {(sequential Row_Generator_1)
on nodes (
ecc3672[op1,p0]
)}
op2[1p] {(parallel APT_LUTCreateImpl in Lookup_3)
on nodes (
ecc3671[op2,p0]
)}
op3[4p] {(parallel buffer(0))
on nodes (
ecc3671[op3,p0]
ecc3672[op3,p1]
ecc3673[op3,p2]
ecc3674[op3,p3]
)}
op4[4p] {(parallel APT_CombinedOperatorController:
(APT_LUTProcessImpl in Lookup_3)
(APT_TransformOperatorImplV0S7_cpLookupTest1_Transformer_7 in Transformer_7)
(PeekNull)
) on nodes (
ecc3671[op4,p0]
ecc3672[op4,p1]
ecc3673[op4,p2]
ecc3674[op4,p3]
)}
op5[1p] {(sequential APT_RealFileExportOperator in Sequential_File_12)
on nodes (
ecc3672[op5,p0]
)}
It runs 12 processes on 4 nodes
```

Buffering Design Assumptions

The primary task of buffering is to prevent potential deadlocks in the data flow. Thus whenever there is a DataStage job with a fork join structure in the flow, the buffer operator will be inserted in the input data links, as shown in Figure 3.

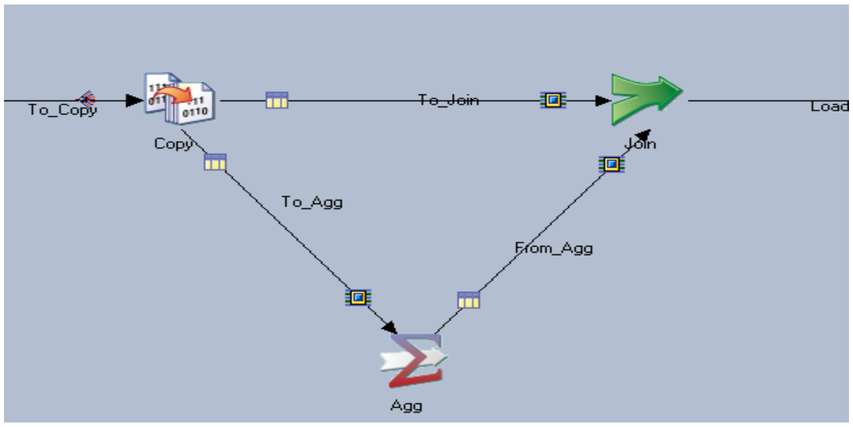


Figure 3: DataStage Job with Fork Join

The buffering mechanism tends to control the flow of data with little memory or disk usage. The goal being, not to write data to the disk as much as possible. In general operators will pass data between each other whenever possible. Buffering comes to play a role, only when backlogs exist in the data link between the operators. If the downstream operator does not read data from the buffer, then the buffer operator will throttle back the output data from the upstream operator, to restrict an increase of the buffer size and to not write data to the disk. The buffering always assumes upstream record producing operators will wait for the downstream record consuming operators.

Buffering Specifications

Parallel framework operators will use buffering by themselves, as necessary. If needed, the default behavior can be changed by altering the buffering specification. In this section, we will discuss environmental variables used to override the default buffering behavior and to control buffering. But changing these environmental variable default values is not recommended as it will degrade performance, cause deadlocks, and in some cases lead to DataStage job locks.

The environmental variables used to override the default buffering behavior are:

APT_BUFFER_MAXIMUM_MEMORY – Indicates the maximum amount of virtual memory used per buffer. The default size is 3MB. This default value could be increased (in case of wide rows) so that the buffer consumes more data.

APT_BUFFER_DISK_WRITE_INCREMENT – Indicates the size of blocks of data moved to / from the disk by the buffer operator. The default size is 1MB. Increasing the default size might reduce disk I/O but decreases performance when data is in smaller units. Decreasing the size might cause frequent disk I/O but increases the throughput.

APT_BUFFER_FREE_RUN – Indicates an available in-memory buffer for consuming records from the upstream operator before the buffer operator offers resistance to any new data coming to the buffer. The default size is 0.5MB – 50% of MaximumMemoryBufferSize specified in APT_BUFFER_MAXIMUM_MEMORY. If the size is set to more than 100%, the buffer operator will stop offering resistance to the upstream operator.

In Figure 4, buffer capacity exceeded the APT_BUFFER_FREE_RUN limit; the arrow mark depicts the buffering threshold.

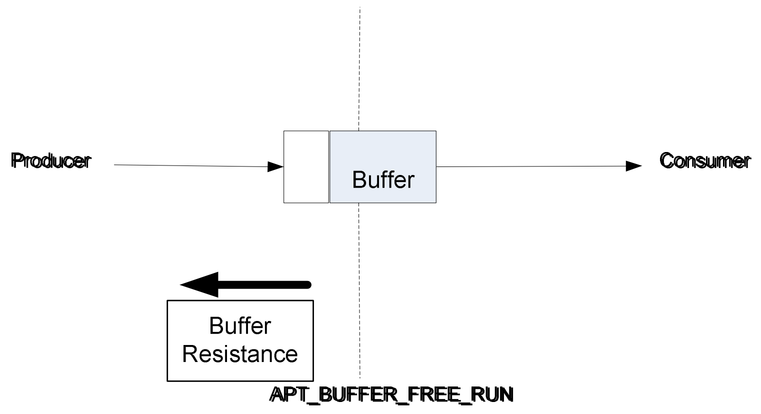


Figure 4: APT_BUFFER_FREEN_RUN Threshold

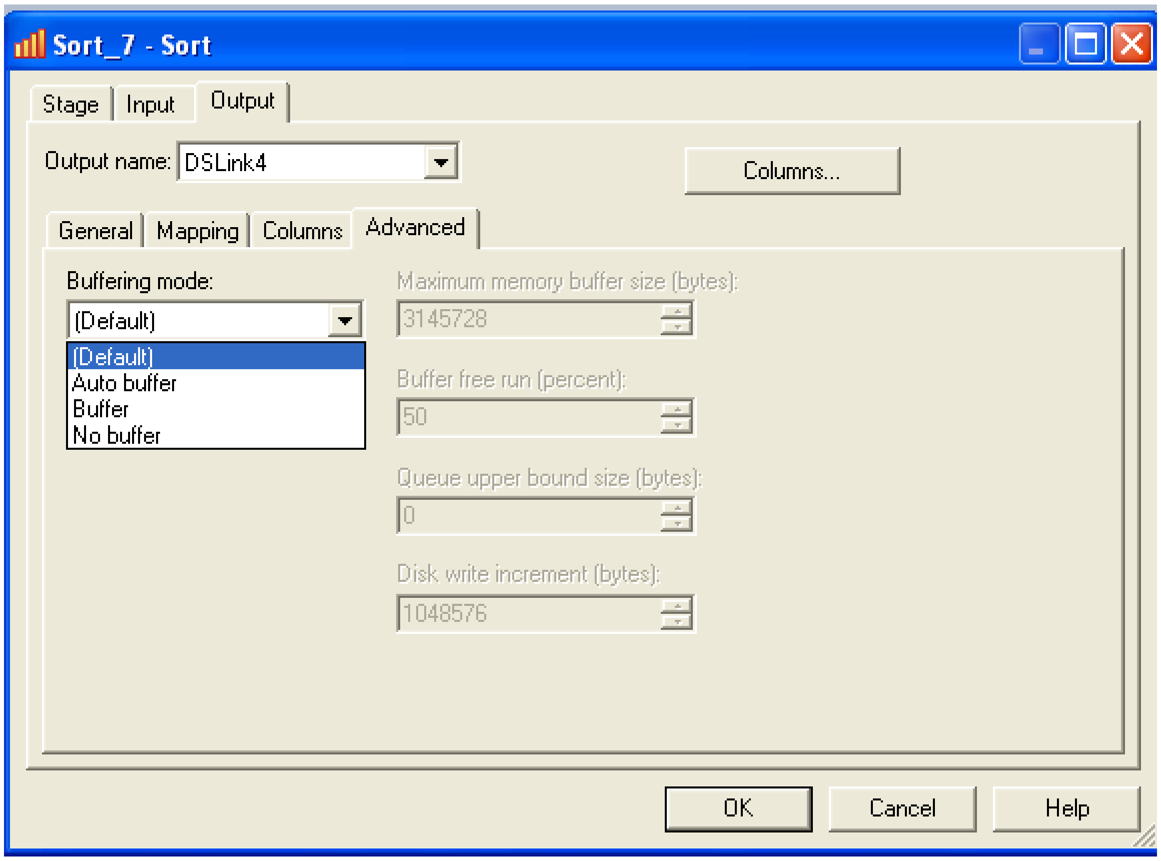
APT_BUFFERING_POLICY – Indicates the buffer control policy for all virtual data sets in all DataStage jobs. There are three options for this environment variable setting:

1) AUTOMATIC_BUFFERING – This is the default option. This option will insert a buffer operator to hold data sets only if necessary, in the DataStage job data flow to avoid deadlock.

2) FORCE_BUFFERING – This option will force all data sets to buffer. It might degrade performance.

3) NO_BUFFERING – Never buffer data sets. Inappropriate usage of this option leads to deadlock.**Buffering Within DataStage Stages**

Buffering can also be controlled through the "Buffering m**ode" option in the DataStage stage** properties. See Figure 5.


Sort_7 - Sort

Output name: DSLink4

Buffering mode: (Default)

(Default)
Auto buffer
Buffer
No buffer

Maximum memory buffer size (bytes): 3145728

Buffer free run (percent): 50

Queue upper bound size (bytes): 0

Disk write increment (bytes): 1048576

Figure 5: DataStage Stage Properties

Few DataStage stages need the entire data set to process. When they do, these stages internally buffer the entire dataset (or groups of data sets) before they output a row to the next stage. For example, stages like Sort, Join, Merge, Remove Duplicate, etc. need the entire data, or groups of data, to process its operation, and will not produce output until the logical end of the data group, or end of data.

For example, for Join stage operations - The first link (Left), i.e. primary link, will read the rows one at a time. The second link (Right), i.e. reference link, will read the rows as groups that belong to the same key value and are loaded to the buffer.

Based on the job design requirement, one has to use the stages appropriately; otherwise the buffering will affect performance.

System Configuration for Buffering

Below is a two node configuration file—when the maximum buffering size is exceeded, the data will be spilled to disk. Parallel framework automatically uses the defined buffer disk pool on a node for data storage. If no buffer pool is specified in the configuration file, then the data will be stored in a default disk pool or default directory specified in $TMPDIR (i.e., the UNIX /tmp directory).

```
{
node node1 {
fastname "node1_css"
pool "" "node1" "node1_css"
resource disk "/orch/s0" {}
resource scratchdisk "/scratch0" {pool "buffer"}
resource scratchdisk "/scratch1" {}
}
node node2 {
fastname "node2_css"
pool "" "node2" "node2_css"
resource disk "/orch/s0" {}
resource scratchdisk "/scratch0" {pool "buffer"}
resource scratchdisk "/scratch1" {}
}
}
```

Summary

In DataStage job designs, try to avoid buffering contention, by maximizing the possibility of data sets not loading into a buffer. Also avoid fork join buffer contention in the data flow by splitting a job into two jobs, and storing the intermediate data sets in a temporary resource.

As a whole, use the buffering mechanism to prevent deadlock and to optimize performance. Parallel framework takes care of buffering automatically whenever it is necessary in data flow. However, changing the default behavior of buffering, through changing default values in the environmental variables, may reduce performance.

® IBM and DataStage are registered trademarks of International Business Machines (IBM Corporation).

™ InfoSphere is a trademark of International Business Machines (IBM Corporation).