Using Multiple Step Processes to Improve Performance for Connect:Direct UNIX

Users of Connect:Direct UNIX in a heavy work load environment can. build Processes so they will execute in the least amount of time to improve performance. This paper describes the most efficient way to create a Process to improve throughput.

Many users of Connect:Direct transfer one or two hundred files per day while other users of Connect:Direct transfer thousands of large files per day. To increase throughput and improve performance, use multiple steps in a Connect:Direct Process script when Connect:Direct is transferring files to the same node
A sample process script called **d_dir/ndm/bin/sample.cd is provided.**

One Step Process – Least Efficient

Most Process open and build the network connection, transfer **one** file, and tear down the network connection. This is a one step Process.

Following is a sample one step Process script:

```
Sample process snode=remote_node

step01
  copy from (file =/cdunix/ndm/bin/Direct
             pnode
            )
       ckpt = 2M
       compress = extended
       to   (file = /export/home/jsmith/cddelete.me
             snode
             disp = new
             sysopts=":datatype=binary:"
            )

pend;
```

sample_process.cd

The majority of session time in this sample is spent managing the network connection. The time spent building the session connection~~,~~ and then tearing it down after the transfer may take longer than the time it takes to transfer a file.

Most users build simple scripts to submit a Process like the sample shown above. Then, during a company's batch cycles, they release multiple Processes to their job queue. Even though Connect:Direct can handle this load, it still must wait while network connectivity is constructed and deconstructed for each Process. This method is not the most efficient and can affect performance in a heavy workload environment.

Multiple Step Process – Most Efficient

Most of these delays can be removed by building Process scripts that contain multiple steps – instead of just one step.

Here is an example of a Process script with multiple steps.

```
Sample process snode=remote_node

step01
copy from (file =/cdunix/ndm/bin/file_01 pnode)
     ckpt = 2M compress = extended
     to   (file = /export/home/jsmith/file_01 snode
           disp = new sysopts=”:datatype=binary:”)

step02
copy from (file =/cdunix/ndm/bin/file_02 pnode)
     ckpt = 2M compress = extended
     to   (file = /export/home/jsmith/file_02 snode
           disp = new sysopts=”:datatype=binary:”)
step03
copy from (file =/cdunix/ndm/bin/file_03 pnode)
     ckpt = 2M compress = extended
     to   (file = /export/home/jsmith/file_03 snode
           disp = new sysopts=”:datatype=binary:”)

pend;
```

sample2_multi_step.cd

In this example, three files are sent within one Process script, by using three Process steps. This example illustrates the manually entry of the file names. However, this script can be built using variables and to include as many steps as needed to transfer files. Variables allow you to provide file names at runtime. The Connect:Direct Process script text is limited to 64K. The three-step Process example above is only 623 bytes.

Another way to achieve multiple step processes is to use the wild card feature included with Connect:Direct version 3.4.00 or later. The wild card feature builds the steps for the Process script at runtime. Below is a sample script that uses wild cards:

```
Sample process snode=remote_node

step01
copy from (file =/cdunix/ndm/stage_01/* pnode)
     ckpt = 2M compress = extended
     to   (file = /export/home/batch/ snode
           disp = new sysopts=”:datatype=binary:”)


pend;
```

sample_wildcard.cd

When this Process runs, Connect:Direct obtains a list of all files in the stage_01 Directory and then sends them to the remote node, placing them in the batch Directory. After all files have been sent, the network connection is closed.

These examples illustrate a few ways of creating multiple Process scripts. Using multiple steps in your Process scripts improves throughput by eliminating the need to establish a network connection between each file transfer. Other ways to create multiple step Processes include shell scripts or hard coded Process scripts.

The backlog caused by single-step Processes is not clear during initial testing since only a handful of jobs are submitted. The backlog is evident when thousands of jobs are started in a production level environment.

Using multiple steps in Process scripts eliminates most backlog situations by eliminating duplicated network management tasks.