FusionIPC Script

Phil Terry <pterry@vmetro.com>

Introduction

This application note describes a simple bash script which together with the Vmetro framework application, “frmwrk,” and some simple assumptions about the networking of the nodes in the chassis demonstrates a very simple program which embodies most if not all of the functionality required in a real application.

The script, called “ghd” in honor of Larry the Cable Guy, generates a number of framework scripts which it then distributes to the nodes in the chassis using “scp” and then invokes the framework application to execute each of those scripts again using the same framework application.

The framework application is using a simple “copy” function as its “signal processing” function.

The ghd script, which is a shell script, is configured by editing a trio of array variables. Each array variable has respectively a node's IP address, IP Fusion Name and a userid for ssh login to the node.

You can define these arrays from 1 to n nodes.

Each node has an in buffer and an out buffer and uses a simple copy function to copy the inbuffer to the outbuffer. It has a “go” event to trigger the DMA'ing of new data into its in buffer from somewhere and the DMA'ing of its output buffer to somewhere.

For each node, somewhere, is the master node which has a corresponding buffer for each of the other nodes as well as a pair for itself. The master node is simply the first node listed in the arrays.

Each node, including the master, uses its DMA engine to copy its out buffer to the somewhere in buffer.

The master node uses its DMA engine to copy all of the out buffers to each of the input buffers on the other nodes, including itself.

The result of this script being executed is therefore a simple indefinite circulation of the buffers:

- from master's <node>_out buffer to node's <node>_in buffer by master's DMA
- from node's <node>_in to node's <node>_out by node's processor copying
- from node's <node>_out to master's <node>_in by node's DMA
- from master's <node>_in to master's <node>_out by master's processor copying

Networking Assumptions

It is assumed that each processor powers up into a standalone Linux compute node and that the FusionIPC driver is installed on each node.

It is assumed that one of the processors assumes, presumably because of being sys_con in the chassis, a DHCP server mode and that all other nodes are DHCP clients. It is assumed that the addresses allocated include those listed in the ghd script.

We do not assume that the chassis networking between nodes is the rionet of the RapidIO backplane

though this will usually be the case. It could also be an Ethernet backplane or front or rear panel connected Ethernet network.

It is assumed that all nodes listed in the ghd script are connected on a RapidIO backplane with support for DMA.

Features Demonstrated

Although this is a very simple example all of the major components required by a signal processing application are used in this example, albeit in a simple manner.

SMB's are allocated, bound to each other with multiple DMA engines, controlled by a set of distributed events, SMBs are mapped to application space and the pipeline features of the framework application are shown.

The master invocation of the copy function demonstrates the flexibility of polymorphic functions and how simply they are setup and used by the framework application.

To coordinate the setup by the master node the use of distributed synchronization objects is shown.

The distribution of configuration data, the framework scripts, via the ssh scp program and invocation of the remote application instances on those scripts via ssh is shown.

In a more realistic program the only features which will be expanded upon is simply more of the same. There would be:

- multiple binds per SMB to more than one destination, the distributed corner turn,
- finer grain control by more events named amongst those binds,
- more scientific processing functions registered by the framework application instances, rather than the example copy function,
- more maps per SMB for functional differentiation,
- etc.

The ghd script

Here is the ghd script

```
#!/bin/bash
while getopts :pude OPT; do
    case $OPT in
       p|+p)
           dopipe=yes
           ;;
       u|+u)
           dounixpipe=yes
           ;;
       d|+d)
           dodist=yes
           ;;
       e|+e)
           doexec=yes
           ;;
       *)
```

```
            echo "usage: ${0##*/} [+-p] [+-d] [+-e] [--] ARGS..."
            exit 2
    esac
done
shift $[ OPTIND - 1 ]

# Customize your script by editing the following lines
#=====================================================
# Which backend drivers to use for fabric and DMA.
fabric_name=vfi_fabric_rionet
dma_name=vfi_dma_rio
# Create a one to one corresponding list of IP addresses and host names
# The host names don't need to be DNS names but are used to name locations
# within FusionIPC
# This is a simple 2 host example.
hosts=(192.168.0.86 192.168.0.108)
hostnames=(eightsix oneoeight)
# List the one to one corresponding ssh logins for each host.
user=(phil phil)
#=====================================================
# You should not need to edit anything else in this file

# The first host listed is assumed to be the "master" which orchestrates
# everyone else.
echo "I am $hostnames ($hosts)"

# Build initial section of masters script file.
echo "# Build the host name tables" > $hostnames.fil
echo
"location_create://fabric#1:1?default_ops(private),fabric_name($fabric_name),dma_name($dma_name)"
>> $hostnames.fil
echo "location_create://${hostnames}.fabric#1:1?default_ops(private)" >> $hostnames.fil

# Now create the list of hosts in master file and the initital scripts for the hosts
host=1
while [ $host -lt ${#hosts[*]} ]; do
    echo ${hostnames[$host]} is ${hosts[$host]}
    echo "location_create://${hostnames[$host]}.fabric#$[ host + 1]?default_ops(public)" >>
$hostnames.fil

    echo "# Register ourselves on the fabric" > ${hostnames[$host]}.fil
    echo
"location_find://fabric?default_ops(public),fabric_name($fabric_name),dma_name($dma_name),wait" >>
${hostnames[$host]}.fil
    echo "location_find://${hostnames[$host]}.fabric?wait" >> ${hostnames[$host]}.fil
    echo "" >> ${hostnames[$host]}.fil
    echo "# Tell $hostnames we are ready" >> ${hostnames[$host]}.fil
    echo "sync_wait://ready.${hostnames}.fabric?wait" >> ${hostnames[$host]}.fil
    echo "" >> ${hostnames[$host]}.fil
    echo "# Wait till ${hostnames} has created everything for us" >> ${hostnames[$host]}.fil
    echo "sync_wait://go.${hostnames}.fabric?wait" >> ${hostnames[$host]}.fil

    host=$[ host + 1 ]
done

# Finish off the master file.
echo "" >> $hostnames.fil
echo "# Create the sync points we will need" >> $hostnames.fil
echo "sync_create://go.${hostnames}.fabric#${#hosts[*]}" >> $hostnames.fil
echo "sync_create://ready.${hostnames}.fabric#${#hosts[*]}" >> $hostnames.fil
echo "" >> $hostnames.fil
echo "# Wait for everyone to register on the fabric" >> $hostnames.fil
echo "sync_wait://ready.${hostnames}.fabric" >> $hostnames.fil
echo "" >> $hostnames.fil
echo "# Now create everything needed by each host" >> $hostnames.fil

# At this point in the scripts everyone is on the FusionIPC fabric with their names
# So now continue master script to create all SMBs, binds and transfers
```

```
host=0
while [ $host -lt ${#hosts[*]} ]; do
       echo "" >> ${hostnames}.fil
       echo "# Create xfer engine, smbs and binds for ${hostnames[$host]}" >> ${hostnames}.fil
       echo "xfer_create://xfer.${hostnames[$host]}.fabric" >> ${hostnames}.fil
       echo "smb_create://${hostnames[$host]}_in.${hostnames[$host]}.fabric:4000" >>
${hostnames}.fil
       echo "smb_create://${hostnames[$host]}_out.${hostnames[$host]}.fabric:4000" >>
${hostnames}.fil
       if [ $host -gt 0 ] ; then
           echo "smb_create://${hostnames[$host]}_in.${hostnames}.fabric:4000" >> ${hostnames}.fil
           echo "smb_create://${hostnames[$host]}_out.${hostnames}.fabric:4000" >>
${hostnames}.fil
       fi
       echo
"bind_create://xfer.${hostnames}.fabric:4000/${hostnames[$host]}_in.${hostnames[$host]}.fabric?eve
nt_name(go)=${hostnames[$host]}_out.${hostnames}.fabric?event_name(go)" >> ${hostnames}.fil
       if [ $host -gt 0 ] ; then
           echo
"bind_create://xfer.${hostnames[$host]}.fabric:4000/${hostnames[$host]}_in.${hostnames}.fabric?eve
nt_name(go)=${hostnames[$host]}_out.${hostnames[$host]}.fabric?event_name(go)" >> ${hostnames}.fil
       fi
       host=$[ host + 1 ]
done

# Now everything is in place for the hosts to continue their scripts to use
# the events and smb's to create their maps and pipes. So have the master
# release the sync.
echo "" >> $hostnames.fil
echo "# Now tell everyone we are done and they can continue" >> $hostnames.fil
echo "sync_wait://go.${hostnames}.fabric" >> $hostnames.fil

# Now continue hosts scripts to map their SMBs, find their events and run their
# pipes.

host=1
while [ $host -lt ${#hosts[*]} ]; do
       echo "" >> $hostnames.fil
       echo "# Mmap ${hostnames[$host]}'s buffers" >> $hostnames.fil
       echo
"mmap_create://${hostnames[$host]}_in.$hostnames.fabric?map_name(${hostnames[$host]}_in)" >>
$hostnames.fil
       echo
"mmap_create://${hostnames[$host]}_out.$hostnames.fabric?map_name(${hostnames[$host]}_out)" >>
$hostnames.fil

       echo "" >> ${hostnames[$host]}.fil
       echo "# Mmap our buffers" >> ${hostnames[$host]}.fil
       echo
"mmap_create://${hostnames[$host]}_in.${hostnames[$host]}.fabric?map_name(${hostnames[$host]}_in)"
>> ${hostnames[$host]}.fil
       echo
"mmap_create://${hostnames[$host]}_out.${hostnames[$host]}.fabric?map_name(${hostnames[$host]}_out
)" >> ${hostnames[$host]}.fil
       echo "" >> ${hostnames[$host]}.fil
       echo "# Find our events" >> ${hostnames[$host]}.fil
       echo "event_find://go.${hostnames[$host]}.fabric" >> ${hostnames[$host]}.fil
       if [ "x$dopipe" = "xyes" ] ; then
           echo "" >> ${hostnames[$host]}.fil
           echo "# Do our pipe" >> ${hostnames[$host]}.fil
           echo
"pipe://${hostnames[$host]}_in<copy(go.${hostnames[$host]}.fabric)>${hostnames[$host]}_out" >>
${hostnames[$host]}.fil
       fi
       if [ "x$dounixpipe" = "xyes" ] ; then
           echo "" >> ${hostnames[$host]}.fil
           echo "# Do our pipe" >> ${hostnames[$host]}.fil
```

```
            echo "unix_pipe://copy go.${hostnames[$host]}.fabric <${hostnames[$host]}_in
>${hostnames[$host]}_out" >> ${hostnames[$host]}.fil
        fi
        host=$[ host + 1 ]
done

echo "" >> ${hostnames}.fil
echo "# Mmap our buffers" >> ${hostnames}.fil
echo "mmap_create://${hostnames}_in.${hostnames}.fabric?map_name(${hostnames}_in)" >>
${hostnames}.fil
echo "mmap_create://${hostnames}_out.${hostnames}.fabric?map_name(${hostnames}_out)" >>
${hostnames}.fil
echo "" >> ${hostnames}.fil
echo "# Find our event" >> ${hostnames}.fil
echo "event_find://go.${hostnames}.fabric" >> ${hostnames}.fil
# Now the piece de la resistance...
if [ "x$dopipe" = "xyes" ] ; then
    outnamea=(${hostnames[*]/%/_out})
    outnames=${outnamea[*]/#/>}
    innamea=(${hostnames[*]/%/_in})
    innames=${innamea[*]/%/<}
    echo "" >> ${hostnames}.fil
    echo "# Do our pipe" >> ${hostnames}.fil
    echo "pipe://${innames// /}copy(go.$hostnames.fabric)${outnames// /}" >> $hostnames.fil
fi
if [ "x$dounixpipe" = "xyes" ] ; then
    outnamea=(${hostnames[*]/%/_out})
    innamea=(${hostnames[*]/%/_in})
    echo "" >> ${hostnames}.fil
    echo "# Do our pipe" >> ${hostnames}.fil
    echo "unix_pipe://copy go.$hostnames.fabric ${innamea[*]/#/<} ${outnamea[*]/#/>}" >>
$hostnames.fil
fi

# Now distribte the host script files to each host in turn and run it there.
host=0
while [ $host -lt ${#hosts[*]} ]; do
        if [ "x$dodist" = "xyes" ] ; then
            scp "${hostnames[$host]}.fil" "${user[$host]}@${hosts[$host]}:${hostnames[$host]}.fil"
        fi
        # This will need to run as background cos it sync_waits
        if [ "x$doexec" = "xyes" ] ; then
            xterm -e ssh ${user[$host]}@${hosts[$host]} "frmwrk -f ${hostnames[$host]}.fil ; bash
-i" &
        fi
        host=$[ host + 1 ]
done
```

The Master Framework Script

This is the master framework script produced by the above ghd script. This script is called eightsix.fil as the IP address ends in 86. Note that the ghd script very generously comments the scripts it produces.

```
# Build the host name tables
location_create://fabric#1:1?default_ops(private),fabric_name(vfi_fabric_rionet),dma_name(vfi_dma_
rio)
location_create://eightsix.fabric#1:1?default_ops(private)
location_create://oneoeight.fabric#2?default_ops(public)

# Create the sync points we will need
sync_create://go.eightsix.fabric#2
sync_create://ready.eightsix.fabric#2

# Wait for everyone to register on the fabric
sync_wait://ready.eightsix.fabric
```

```
# Now create everything needed by each host

# Create xfer engine, smbs and binds for eightsix
xfer_create://xfer.eightsix.fabric
smb_create://eightsix_in.eightsix.fabric:4000
smb_create://eightsix_out.eightsix.fabric:4000
bind_create://xfer.eightsix.fabric:4000/eightsix_in.eightsix.fabric?event_name(go)=eightsix_out.ei
ghtsix.fabric?event_name(go)

# Create xfer engine, smbs and binds for oneoeight
xfer_create://xfer.oneoeight.fabric
smb_create://oneoeight_in.oneoeight.fabric:4000
smb_create://oneoeight_out.oneoeight.fabric:4000
smb_create://oneoeight_in.eightsix.fabric:4000
smb_create://oneoeight_out.eightsix.fabric:4000
bind_create://xfer.eightsix.fabric:4000/oneoeight_in.oneoeight.fabric?event_name(go)=oneoeight_out
.eightsix.fabric?event_name(go)
bind_create://xfer.oneoeight.fabric:4000/oneoeight_in.eightsix.fabric?event_name(go)=oneoeight_out
.oneoeight.fabric?event_name(go)

# Now tell everyone we are done and they can continue
sync_wait://go.eightsix.fabric

# Mmap oneoeight's buffers
mmap_create://oneoeight_in.eightsix.fabric?map_name(oneoeight_in)
mmap_create://oneoeight_out.eightsix.fabric?map_name(oneoeight_out)

# Mmap our buffers
mmap_create://eightsix_in.eightsix.fabric?map_name(eightsix_in)
mmap_create://eightsix_out.eightsix.fabric?map_name(eightsix_out)

# Find our event
event_find://go.eightsix.fabric

# Do our pipe
unix_pipe://copy go.eightsix.fabric <eightsix_in <oneoeight_in >eightsix_out >oneoeight_out
```

This being the master script is the most complex.

A sample node script

We present here just one of the other scripts produced for the rest of the listed nodes which are essentially identical in form with only the names changing. This is the oneoeight.fil (one oh eight because the IP address ends in 108).

```
# Register ourselves on the fabric
location_find://fabric?default_ops(public),fabric_name(vfi_fabric_rionet),dma_name(vfi_dma_rio),wa
it
location_find://oneoeight.fabric?wait

# Tell eightsix we are ready
sync_wait://ready.eightsix.fabric?wait

# Wait till eightsix has created everything for us
sync_wait://go.eightsix.fabric?wait

# Mmap our buffers
mmap_create://oneoeight_in.oneoeight.fabric?map_name(oneoeight_in)
mmap_create://oneoeight_out.oneoeight.fabric?map_name(oneoeight_out)

# Find our events
event_find://go.oneoeight.fabric
```

```
# Do our pipe
unix_pipe://copy go.oneoeight.fabric <oneoeight_in >oneoeight_out
```