OpenKasugai Demo

v1.0.0

Contents

1. Demonstration procedure of basic functions

This document shows the procedure from DataFlow (DF) deployment to data communication confirmation as a demonstration procedure of the OpenKasugai controller's basic functions.

- Deploy DF using FPGA F/R (Filter & Resize). Deploy two DFs with the same configuration.
  - df-test-1-1: CPU Decode - (PCIe connection) → FPGA F/R - (PCIe connection) → GPU Advanced Inference
  - df-test-1-2: CPU Decode - (PCIe connection) → FPGA F/R - (PCIe connection) → GPU Advanced Inference

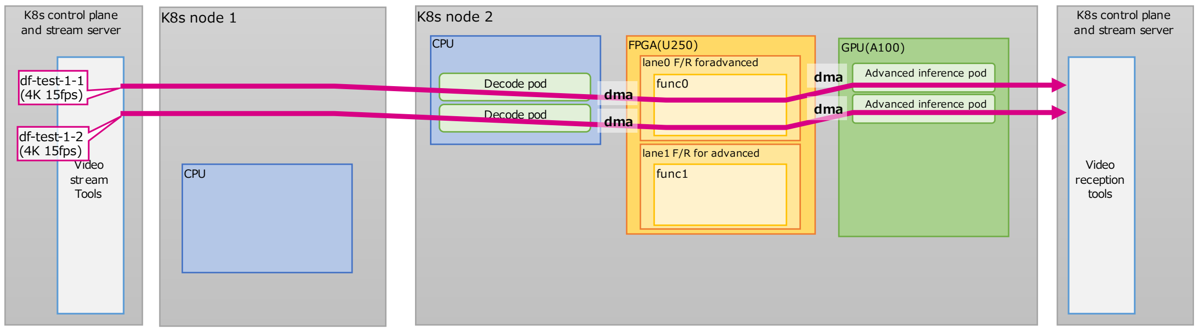

Figure 1: Deployment image of two DFs with FPGA F/R

1.1 Advance preparation

1.1.1 Deployment of test scripts

Target：K8s control plane, all K8s nodes

Deploy the scripts used in this document to simplify the evaluation procedures (various shell scripts and YAML files for deploying video streaming and receiving tools).

1. Copy the test script directory (script) in the acquired material "CRC Source Code & Sample Data" to the home directory.

```
$ cd ~/
$ cp -rf ~/controller/test/script .
```

2. If necessary, modify the following variables in each script appropriately for the environment.

Table 1 List of Scripts and Variables

File name	Target	Variable name	Setting value
run_controllers.sh	control plane	YAML_DIR K8S_SOFT_DIR	・absolute path of yaml/ ・absolute path of controller/
delete_all.sh		YAML_DIR K8S_SOFT_DIR	・absolute path of yaml/ ・absolute path of controller/
reset_ph3_fpga.sh	node with FPGA card	BIT_DIR	・Absolute path of the directory containing BIT files

*In the above reset_ph3_fpga.sh, Bitstream is written using the mcap tool, but the -x (PCI device ID specification) and -s (BDF specification) options in the mcap command are environment-dependent values, so it is necessary to change them according to the environment to be built. The values of -x and -s for each environment (K8s node) can be checked with the lspci command. In the execution example of lspci below, the information of the number of FPGA cards is displayed, and xx: xx.x (e.g., 1f: 0.0) is the BDF, and 903f is the PCI device ID. The blue letters are the -s value, and the red letters are the -x value..

```
$ lspci |grep Xilinx
xx:xx.x Processing accelerators: Xilinx Corporation Device 903f
・・・

$ vi ~/script/reset_ph3_fpga.sh
・・・
BIT_DIR=$HOME/hardware-design/example-design/bitstream
cd $BIT_DIR
$ sudo mcap -x 903f -s xx:xx.x -p OpenKasugai-fpga-example-design-1.0.0-2.bit
・・・
```

1.1.2 Editing DataFlow yaml

Target：K8s control plane

For each DataFlow yaml deployed in Section 1.3.1, change the network information (IP address and port number) used by Pod of each processing module in DataFlow as necessary.
If the IP address set for each server's 100G NIC is different from the IP address shown in Figure 1 in Section 1.1 of the separate "OpenKasugai-Controller-Install Manual," change the IP address.
For instructions on how to make the changes, please refer to the sheet "2 (Supplement). DataFlow yaml settings" in the separate document "OpenKasugai-Controller-InstallManual_Attachment1".

Table 2 List of DataFlow yaml deployed in Section 1.3.1

DataFlow Types	yaml to be edited
1st FPGA F/R DF (df-test-1-1)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-1/df-test-1-1.yaml
2nd FPGA F/R DF (df-test-1-2)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-1/df-test-1-2.yaml

* When changing DataFlow name (metadata.name value) in the above sample data, the name must be 14 characters or less. Refer to the attached "OpenKasugai-Controller_Attachment1 (CR/CM Specification)" for details on the character limit.

1.2 Environment initialization

*When redeploying DataFlow and streaming video, be sure to perform "1.3.4 Video stream and reception stop" and "1.4 Stop Environment" before starting from the following steps.

1.2.1 Write FPGA Bitstream and auto collect & rerun CM creation tool

Target：K8s node with FPGA card

(1) Writing FPGA Bitstream

Execute the following script to write the Bitstream to the FPGA and reset it.

```
$ cd ~/script
$ ./reset_ph3_fpga.sh
```

* Writing the Bitstream takes several tens of seconds. If the writing is successful, "FPGA Configuration Done!!" will be displayed in the log.

(2) Running the automatic collection & CM creation tool

Run the automatic collection & CM creation tool on the target K8s node to restore the ConfigMap and CustomResource associated with the FPGA to their initial state.

```
$ export
  PKG_CONFIG_PATH=${PKG_CONFIG_PATH}:$HOME/controller/src/submodules/fpga-
  software/lib/build/pkgconfig
$ cd ~/controller/src/tools/InfoCollector/
$ ln -s ../../fpgadb/test/bitstream_id-config-table.json bitstream_id-config-
  table.json
$ make all
```

*In case of K8s Node without NVIDIA GPU, since DCGM is not installed, the following error message is output in the InfoCollector log, but it can be ignored.

“INFO infocollect/infocollect.go:405 dcgm.Init() Error but Maybe there are NOT any GPU. {"error": "libdcgm.so not Found"}

1.2.2 CRC start up

Target : K8s control plane

(1) Running each CRC

Execute the following command to start each CRC and apply the function catalog yaml.

```
$ cd ~/script
$ ./run_controllers.sh
```

(2) Checking CRC startup and CRD installation.

Enter the following command to check that each CRC pod is running, all CRDs (19 types) are installed, and the FunctionChain of cpu-decode-filter-resize-high-infer-chain is Ready.

If the check result is different from the output shown below, identify the cause by checking the log of each Pod or the result of kubectl describe, and then execute "1.4 Environmental shutdown". After taking the corrective action, start again from "1.2 Environment initialization"

```
$ kubectl get pod
NAME                                 READY   STATUS     RESTARTS    AGE
crc-cpufunction-daemon-gh4gb         1/1     Running    0           15s
crc-cpufunction-daemon-xzf9s         1/1     Running    0           15s
crc-deviceinfo-daemon-qgv5w          1/1     Running    0           18s
crc-deviceinfo-daemon-rwmqh          1/1     Running    0           18s
crc-ethernetconnection-daemon-h67pj  1/1     Running    0           15s
crc-ethernetconnection-daemon-t2hnm  1/1     Running    0           15s
crc-fpgafunction-daemon-ts8lt        1/1     Running    0           15s
crc-fpgafunction-daemon-zxxrs        1/1     Running    0           15s
crc-gpufunction-daemon-mzkwk         1/1     Running    0           15s
crc-gpufunction-daemon-r596t         1/1     Running    0           15s
crc-pcieconnection-daemon-hvrg6      1/1     Running    0           15s
crc-pcieconnection-daemon-tg79x      1/1     Running    0           15s

$ kubectl  get pod -n whitebox-k8s-flowctrl-system
NAME                                                    READY   STATUS    RESTARTS  AGE
whitebox-k8s-flowctrl-controller-manager-74b469d866-c8q7f    2/2     Running   0         5m47s
$ kubectl get pod -n wbfunction-system
NAME                                                    READY   STATUS    RESTARTS  AGE
wbfunction-controller-manager-6df5bf76d6-wvj65          2/2     Running   0         5m40s
$ kubectl get pod -n wbconnection-system
NAME                                                    READY   STATUS    RESTARTS  AGE
wbconnection-controller-manager-78b44dd58b-lbpnq        2/2     Running   0         6h44m
$ kubectl get crd |grep example.com
childbs.example.com                               2024-11-11T00:18:56Z
computeresources.example.com                       2024-11-11T07:29:35Z
connectiontargets.example.com                      2024-11-11T07:29:35Z
connectiontypes.example.com                        2024-11-11T07:29:35Z
cpufunctions.example.com                           2024-11-08T08:29:56Z
dataflows.example.com                              2024-11-11T07:29:36Z
deviceinfoes.example.com                           2024-11-10T22:56:32Z
ethernetconnections.example.com                    2024-11-11T07:29:36Z
fpgafunctions.example.com                          2024-11-11T00:18:56Z
fpgas.example.com                                  2024-11-11T00:18:56Z
functionchains.example.com                         2024-11-11T07:29:36Z
functiontargets.example.com                        2024-11-11T07:29:36Z
functiontypes.example.com                          2024-11-11T07:29:36Z
gpufunctions.example.com                           2024-11-11T00:25:38Z
pcieconnections.example.com                        2024-11-11T07:29:36Z
schedulingdata.example.com                         2024-11-11T07:29:36Z
topologyinfos.example.com                          2024-11-11T07:29:36Z
wbconnections.example.com                          2024-11-11T07:29:36Z
wbfunctions.example.com                            2024-11-11T07:29:36Z
$ kubectl get functionchains.example.com -A
NAMESPACE      NAME                                        STATUS
chain-imgproc   cpu-decode-filter-resize-high-infer-chain   Ready
・・・
```

1.3 Video stream

1.3.1 DataFlow deployment

Target : K8s control plane

(1) Deploying DataFlow

The following describes how to deploy the DataFlow (DF) of the pattern A shown at the beginning of the chapter.

Apply the DF yaml of the sample data df-test-1-1 and df-test-1-2 one by one.

```
$ cd ~/controller/test/sample-data/sample-data-demo
$ kubectl apply -f yaml/dataflows/test-1/df-test-1-1.yaml
$ kubectl get dataflow -A
(Wait about 30 seconds for STATUS to be Deployed)

$ kubectl apply -f yaml/dataflows/test-1/df-test-1-2.yaml
```

(2) Checking deployment results

Check the results (STATUS) of the kubectl get command to ensure that each custom resource is generated normally and each Pod is deployed normally.

Check the following from the command execution result.

・Status of each DataFlow is Deployed

・The STATUS of each CPUFunction, GPUFunction, FPGAFunction, EthernetConnection, and PCIeConnection is Running.

・ FROMFUNC_STATUS and TOFUNC_STATUS of each EthernetConnection and PCIeConnection are OK.

・ No resources have been created for each EthernetConnection ("No resources found" is displayed)

・The STATUS for each Pod is Running

```
$ kubectl get dataflows.example.com -A
$ kubectl get cpufunctions.example.com -A
$ kubectl get gpufunctions.example.com -A
$ kubectl get fpgafunctions.example.com -A
$ kubectl get pcieconnections.example.com -A
$ kubectl get ethernetconnections.example.com -A
$ kubectl get pod -n test01 -o wide
```

1.3.2 Start video reception

Target : K8s control plane

(1) Method for starting video reception

i. Deployment of video reception tool and creation of reception script (advanced inference result reception ×2).

(Only perform the first time, Step i is unnecessary from the second time onward.)

```
$ cd ~/controller/sample-functions/utils/rcv_video_tool/
$ kubectl create ns test
$ kubectl apply -f rcv_video_tool.yaml
$ kubectl get pod -n test    *Checking the pod name of video stream tool
$ kubectl exec -n test -it rcv-video-tool-xxx -- bash  *The xxx part is
changed according to the pod name confirmed above.
# vi start_test1.sh  *Paste the following and save it.
```

```
#!/bin/bash -x

for i in `seq -w 01 02`
do
    gst-launch-1.0 -e udpsrc buffer-size=21299100 mtu=8900 port=20${i} !
'application/x-rtp, media=(string)video, clock-rate=(int)90000, encoding-
name=(string)RAW, sampling=(string)BGR, depth=(string)8, width=(string)1280,
height=(string)1280, payload=(int)96' ! rtpvrawdepay ! queue ! videoconvert !
'video/x-raw, format=(string)I420' ! openh264enc ! 'video/x-h264, stream-
format=byte-stream, profile=(string)high' ! perf name=stream${i} !
h264parse ! qtmux ! filesink location=/tmp/output_st${i}.mp4 sync=false >
/tmp/rcv_video_tool_st${i}.log &
done
```

```
# chmod +x start_test1.sh
# exit
```

ii. In a new window, open a terminal for the receiving tool, climb into the Pod and start receiving video.

```
$ kubectl exec -n test -it rcv-video-tool-xxx -- bash  *Change pod name to
suit your environment
# ./start_test1.sh
```

(CRITICAL/WARNING logs are output only on the first time, but you can return to the prompt by pressing Enter. If the process startup can be confirmed in the next step, there is no problem.)

iii. Check the startup of the video reception tool inside the container.

* Verify that there are two processes starting with COMMAND as ¥_ gst-launch-1.0 running using ps auxwwf.

```
# ps auxwwf
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         971  5.0  0.0   6048  2836 pts/49   Rs+  10:03   0:00 ps auxwwf
...
root         954  0.0  0.0 317188 13380 pts/48   Sl+  09:57   0:00  ¥_ gst-
launch-1.0 -e udpsrc buffer-size=21299100 mtu=8900 port=2008 !
application/x-rtp, media=(string)video, clock-rate=(int)90000, encoding-
name=(string)RAW, sampling=(string)BGR, depth=(string)8, width=(string)1280,
height=(string)1280, payload=(int)96 ! rtpvrawdepay ! queue ! videoconvert !
video/x-raw, format=(string)I420 ! openh264enc ! video/x-h264, stream-
format=byte-stream, profile=(string)high ! queue ! perf name=stream08 !
h264parse ! qtmux ! filesink location=/tmp/output_st08.mp4 sync=1
root         943  0.0  0.0 317188 13392 pts/47   Sl+  09:57   0:00  ¥_ gst-
launch-1.0 -e udpsrc buffer-size=21299100 mtu=8900 port=2007 !
application/x-rtp, media=(string)video, clock-rate=(int)90000, encoding-
name=(string)RAW, sampling=(string)BGR, depth=(string)8, width=(string)1280,
height=(string)1280, payload=(int)96 ! rtpvrawdepay ! queue ! videoconvert !
video/x-raw, format=(string)I420 ! openh264enc ! video/x-h264, stream-
format=byte-stream, profile=(string)high ! queue ! perf name=stream07 !
h264parse ! qtmux ! filesink location=/tmp/output_st07.mp4 sync=1
...
```

1.3.3 Start of video stream

Target：K8s control plane

(1) Method for starting video distribution

i. Deployment of video streaming tools and creation of streaming scripts (two 4K 15fps streams for advanced inference)
(Execute only the first time, Step i is unnecessary from the second time onward)

```
$ cd ~/controller/sample-functions/utils/send_video_tool/
$ kubectl apply -f send_video_tool.yaml
$ kubectl get pod -n test    *Checking the pod name of video stream tool
$ kubectl exec -n test -it send-video-tool-xxx -- bash  *Change pod name to
suit your environment
# vi start_test1.sh    *Paste and save the following
```

```
#!/bin/bash

sleep_time=$1

./start_gst_sender.sh /opt/video/input_4K_15fps.mp4 192.174.90.101 5004 1
${sleep_time:-3}

./start_gst_sender.sh /opt/video/input_4K_15fps.mp4 192.174.90.102 5004 1
${sleep_time:-3}
```

```
# chmod +x start_test1.sh
# exit
```

Table 3: Explanation of arguments for start_gst_sender.sh in the above script

nth	Description	About Values
1	Video file path	Specifies the video file to be delivered. Change the settings according to the movie file to be used. (The above example video was created in section 7.8 of the "OpenKasugai-Controller-InstallManual".) Use 4 K 15fps video files. The length of the video is preferably about 30 to 60 seconds.
2	Destination IP	Specify the destination IP address. Change it to match the IP address assigned to the CPU decoding described in the DataFlow yaml.
3	Destination Port	Specify the destination port number. Change the port number to match the reception port number set for the CPU decoding described in the DataFlow yaml.
4	Number of startup processes	Specify the number of delivery processes to generate (If you specify 2 or more, start the specified number of processes that distribute the same video file to the same destination IP. Destination port number is incremented each time the delivery process is started)

5	Streaming Delay Interval	Video streaming start delay [seconds] (interval for starting streaming processes when 2 or more is specified for the fourth argument)

ii. In a new window, you open a terminal for the distribution tool, climb into the Pod, and start streaming.

```
$ kubectl exec -n test -it send-video-tool-xxx -- bash  *Change pod name to
suit your environment
# ./start_test1.sh
```

(A CRITICAL log may be output only the first time, but you can return to the prompt by pressing Enter.There is no problem if the process startup can be confirmed in the next step.)

iii. Confirm the startup of the video streaming tool inside the container.
*Verify that two gst-launch runs on ps aux.

```
# ps aux | grep gst-launch
root       2335  0.0  0.0 309636 12272 pts/3    Sl   11:17   0:00 gst-launch-
1.0  filesrc  location=/mnt/input_4K_15fps.mp4  !  qtdemux  !  video/x-h264  !
h264parse   !   rtph264pay   config-interval=-1   seqnum-offset=1   !   udpsink
host=192.174.91.81 port=5004 buffer-size=2048
root       2336  0.0  0.0 309636 12272 pts/3    Sl   11:17   0:00 gst-launch-
1.0  filesrc  location=/mnt/input_4K_15fps.mp4  !  qtdemux  !  video/x-h264  !
h264parse   !   rtph264pay   config-interval=-1   seqnum-offset=1   !   udpsink
host=10.38.119.67 port=5004 buffer-size=2048

. . .
```

(2) Video input/output check of DataFlow

By following this procedure, video input from the video distribution tool is processed by the deployed DataFlow, and the resulting video is delivered to the video receiving tool. The video file of the distributed video will stop streaming once the video time elapses, but you can stop the distribution at any time by following the steps in the next section.

You can check if the video is being received by monitoring the mp4 files under/tmp in the video receiving tool container.

```
$ kubectl exec -n test -it rcv-video-tool-xxx -- bash  *Change pod name to suit
your environment
# cd /tmp/
# ls -l
(Check the increase in size of each mp4 file.)
```

1.3.4 Video stream and reception stop

Target：K8s control plane

1. Stop within the container of the video distribution tool started in Section 1.3.3 and confirm that it has stopped.

```
$ kubectl exec -n test -it send-video-tool-xxx -- bash  *Change pod name to suit
your environment
# ./stop_gst_sender.sh
# ps aux
    (Make sure that gst-launch is not running)
```

2. Stop within the container of the video reception tool as follows and confirm that it has stopped.

```
$ kubectl exec -n test -it rcv-video-tool-xxx -- bash  *Change pod name to suit
your environment
# pgrep gst-launch-1.0 | xargs kill -2 > /dev/null
# ps aux
    (Make sure that gst-launch is not running)
```

*The following error message may be output, but it can be ignored.
[OpenH264] this = 0x0x7fccc40058c0, Error:CWelsH264SVCEncoder::EncodeFrame(), cmInitParaError.

3. Copy each output video file to the host.

```
$ kubectl -n test cp  rcv-video-tool-xxx:/tmp/output_stXX.mp4 ./output_stXX.mp4
    --retries 10
 (XX:01~02)
```

*Execute the command with the --retries 10 option because the copy operation may fail.
*Even if the following message is output, it can be ignored if the video file is copied.
"tar: Removing leading `/' from member names"

4. The video output is a video in which the Bounding Box(BB) for the detected person is drawn.
*In the case of an DF that includes the configuration of "FPGA F/R (PCIe connection) to GPU advanced inference" deployed in Section 1.3.1, a blank image is shown for the first approximately one minute of the output video, and then a BB image is shown, which is a normal result. The blank image is the dummy input data when no data has yet come to GPU advanced inference.

1.4 Environmental shutdown

1.4.1 Delete DataFlow, Delete various CRs, Stop each CRC

Target: K8s control plane

Execute the following script to delete DataFlow, delete various CRs, and stop each CRC.

```
$ cd ~/script
$ ./delete_all.sh
```

After executing the above script, it is normal that some CRD and ConfigMap registered in the separate "OpenKasugai-Controller-InstallManual" procedure remain as shown below, so there is no need to manually delete them.

```
$ kubectl get crd |grep example.com
childbs.example.com                                2024-11-11T00:18:56Z
cpufunctions.example.com                           2024-11-08T08:29:56Z
deviceinfoes.example.com                           2024-11-10T22:56:32Z
ethernetconnections.example.com                    2024-11-11T23:30:33Z
fpgafunctions.example.com                          2024-11-11T00:18:56Z
fpgas.example.com                                  2024-11-11T00:18:56Z
gpufunctions.example.com                           2024-11-11T00:25:38Z
pcieconnections.example.com                        2024-11-11T23:29:56Z

$ kubectl get cm
NAME                                       DATA   AGE
connectionkindmap                          1      6d11h
cpufunc-config-copy-branch                 1      6d11h
cpufunc-config-decode                      1      6d11h
cpufunc-config-filter-resize-high-infer    1      6d11h
cpufunc-config-filter-resize-low-infer     1      6d11h
cpufunc-config-glue-fdma-to-tcp            1      6d11h
deployinfo                                 1      22d
filter-resize-ch                           1      22d
fpgafunc-config-filter-resize-high-infer   1      6d11h
fpgafunc-config-filter-resize-low-infer    1      6d11h
function-unique-info                       1      6d11h
functionkindmap                            1      6d11h
gpufunc-config-high-infer                  1      6d11h
gpufunc-config-low-infer                   1      6d11h
infrastructureinfo                         1      22d
kube-root-ca.crt                           1      83d
region-unique-info                         1      6d11h
```

Target: All K8s node with an FPGA card

Use the following command to remove related files for use with PCIe connections.

```
$ sudo rm -rf /var/run/dpdk/*
$ sudo rm -rf /dev/hugepages/*
```

*When deploying DataFlow and distributing video again, start from "1.2 Initializing the environment."

2. [Appendix] Extension demonstration procedure

In this chapter, as a demonstration procedure of the OpenKasugai controller extension function, the procedure from the deployment of copy branch DataFlow and Glue DataFlow to the confirmation of data communication is shown.

1. Deploy one DF that branches into multiple functions in the middle using copy branch
   - df-test-ext-1-1: CPU Decode —(Ethernet connection)→ CPU F/R —(Ethernet connection)→ CPU Copy Branch —(Ethernet connection)→ GPU Advanced Inference ×2

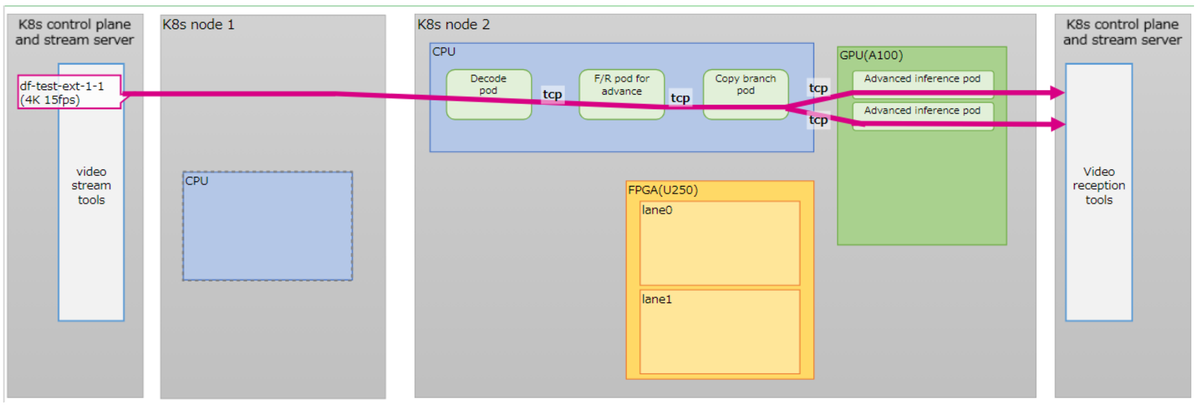

Figure 2　DF deployment image branching into multiple functions

2. Use connection type conversion processing (Glue) to deploy a single DF that connects functions with different connection types.
   - df-test-ext-2-1: CPU Decode - (PCIe connection) → FPGA F/R - (PCIe connection) → CPU Glue -(Ethernet connection) → GPU advanced inference

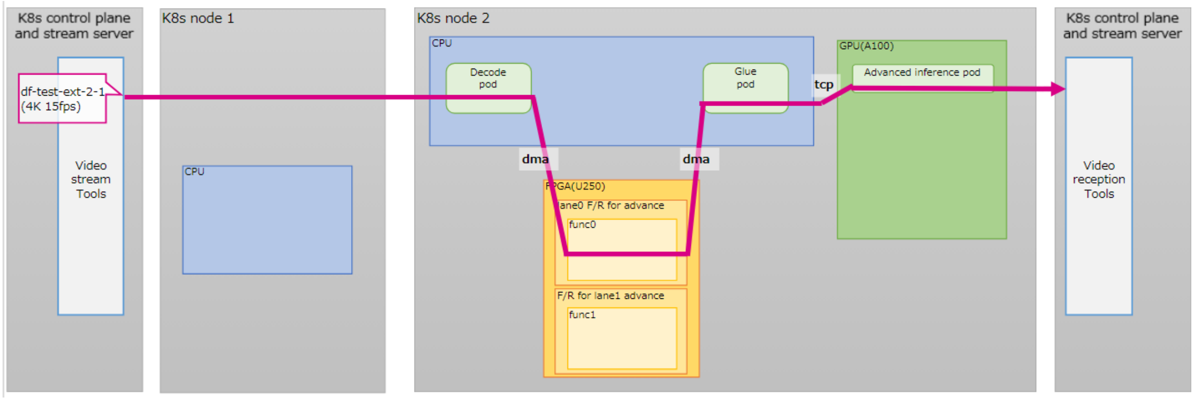

Figure 3　DF deployment image connecting functions of different connection types

2.1 Advance preparation

2.1.1 Deploying test scripts

Targe: K8s control plane, all K8s nodes

Perform step 1.1.1. If already performed, skip this step.

2.1.2 Editing DataFlow yaml

Target：K8s control plane

For each DataFlow yaml deployed in Section 2.3.1, change the network information (IP address and port number) used by Pod of each processing module in DataFlow as necessary.
If the IP address set for each server's 100G NIC is different from the IP address shown in Figure 1 in Section 1.1 of the separate "OpenKasugai-Controller-Install Manual," change the IP address. For the method of change, refer to the separate sheet of "OpenKasugai-Controller-InstallManual_Attachment1" "2 (Supplement) DataFlow yaml settings"

Table 4 List of DataFlow yaml to deploy in Section 2.3.1

DataFlow Types	yaml to edit
1. Copy Branch DF (df-test-ext-1-1)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-ext-1/df-test-ext-1-1.yaml
2. Glue DF (df-test-ext-2-1)	~/controller/test/sample-data/sample-data-demo/yaml/dataflows/test-ext-2/df-test-ext-2-1.yaml

2.2 Environment initialization

Execute section 1.2.

2.3 Video stream

2.3.1 DataFlow deployment

Target：K8s control plane

(1) Deploying DataFlow
The following describes how to deploy DataFlow as shown at the beginning of Chapter 2.

i. For copy branch DF
Apply the DataFlow yaml in the sample data df-test-ext-1-1.

```
$ cd ~/controller/test/sample-data/sample-data-demo
$ kubectl apply -f yaml/dataflows/test-ext-1/df-test-ext-1-1.yaml
$ kubectl get dataflow -A
(Wait about 20 seconds until it becomes Deployed)
```

ii. For DF with Glue conversion
Apply the DataFlow yaml in the sample data df-test-ext-2-1.

```
$ cd ~/controller/test/sample-data/sample-data-demo
$ kubectl apply -f yaml/dataflows/test-ext-2/df-test-ext-2-1.yaml
$ kubectl get dataflow -A
(Wait about 20 seconds until it becomes Deployed)
```

(2) Checking deployment results

Check the results (STATUS) of the kubectl get command to ensure that each custom resource is generated normally and each Pod is deployed normally.

i. In the case of copy branch DF:

Check the following from the command execution result:

・Each DataFlow has a STATUS of Deployed

・Status is Running for each CPUFunction, GPUFunction, and EthernetConnection

・FROMFUNC_STATUS and TOFUNC_STATUS are OK for each EthernetConnection

・The STATUS for each Pod is Running

```
$ kubectl get dataflows.example.com -A
$ kubectl get cpufunctions.example.com -A
$ kubectl get gpufunctions.example.com -A
$ kubectl get ethernetconnections.example.com -A
$ kubectl get pod -n test01 -o wide
```

ii. In the case of Glue conversion DF

Check the following from the command execution result:

・Each DataFlow has a STATUS of Deployed

・Status is Running for each CPUFunction, GPUFunction, FPGAFunction, PCIeConnection, and EthernetConnection

・FROMFUNC_STATUS and TOFUNC_STATUS for each PCIeConnection and each EthernetConnection must be OK.

・The STATUS for each Pod is Running

```
$ kubectl get dataflows.example.com -A
$ kubectl get cpufunctions.example.com -A
$ kubectl get gpufunctions.example.com -A
$ kubectl get ethernetconnections.example.com -A
$ kubectl get fpgafunctions.example.com -A
$ kubectl get pcieconnections.example.com -A
$ kubectl get pod -n test01 -o wide
```

2.3.2 Start video reception

Target：K8s control plane

(1) Method for starting video reception

i. For copy branch DF

Execute the same procedure as in section 1.3.2.

ii. For DF with Glue conversion

Deployment of the video distribution tool and creation of a receiving script (1 for receiving advanced inference results).

(Execute only the first time, step i is unnecessary from the second time onward)

```
$ cd ~/controller/sample-functions/utils/rcv_vide_tool/
$ kubectl create ns test
$ kubectl apply -f rcv_video_tool.yaml
$ kubectl get pod -n test    *Checking the pod name of video stream tool
$ kubectl exec -n test -it rcv-video-tool-xxx -- bash  *The xxx part is changed according to the pod name confirmed above.
# vi start_test2.sh    *Paste and save the following
```

```
#!/bin/bash -x

for i in `seq -w 01 01`
do
    gst-launch-1.0 -e udpsrc buffer-size=21299100 mtu=8900 port=20${i} ! 'application/x-rtp, media=(string)video, clock-rate=(int)90000, encoding-name=(string)RAW, sampling=(string)BGR, depth=(string)8, width=(string)1280, height=(string)1280, payload=(int)96' ! rtpvrawdepay ! queue ! videoconvert ! 'video/x-raw, format=(string)I420' ! openh264enc ! 'video/x-h264, stream-format=byte-stream, profile=(string)high' ! perf name=stream${i} ! h264parse ! qtmux ! filesink location=/tmp/output_st${i}.mp4 sync=false > /tmp/rcv_video_tool_st${i}.log &
done
```

```
# chmod +x start_test2.sh
# exit
```

In a new window, open a terminal for the receiving tool, climb into the Pod and start receiving video.

```
$ kubectl exec -n test -it rcv-video-tool-XXX -- bash  *Change pod name to suit your environment
# ./start_test2.sh
```

Check the start of the video reception tool.
*ps auxwwf to see that 1 process with COMMAND starting at ¥_ gst-launch-1.0 is running.

```
$ ps auxwwf
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         971  5.0  0.0   6048  2836 pts/49   Rs+  10:03   0:00 ps auxwwf
...
root         954  0.0  0.0 317188 13380 pts/48   Sl+  09:57   0:00  ¥_ gst-
launch-1.0 -e udpsrc buffer-size=21299100 mtu=8900 port=2008 !
application/x-rtp, media=(string)video, clock-rate=(int)90000, encoding-
name=(string)RAW, sampling=(string)BGR, depth=(string)8,
width=(string)1280, height=(string)1280, payload=(int)96 ! rtpvrawdepay !
queue ! videoconvert ! video/x-raw, format=(string)I420 ! openh264enc !
video/x-h264, stream-format=byte-stream, profile=(string)high ! queue !
perf name=stream08 ! h264parse ! qtmux ! filesink
location=/tmp/output_st08.mp4 sync=1
...
```

2.3.3 Start of video stream

Target : K8s control plane

(1) Method for starting video stream

*Copy Branch DF and Glue Transform DF Common

i. Deployment of video stream tool and creation of stream script
(4K 15fps (-for advanced inference) × 1 stream)

(Perform this procedure only for the first time. This procedure (1) is not necessary for the second and subsequent times.)

```
$ cd ~/controller/sample-functions/utils/send_video_tool/
$ kubectl apply -f send_video_tool.yaml
$ kubectl get pod -n test    *Checking the pod name of video stream tool
$ kubectl exec -n test -it send-video-tool-xxx -- bash  *The xxx part is
changed according to the pod name confirmed above.
# vi start_test2.sh    *Paste and save the following
```

```
#!/bin/bash

sleep_time=$1

./start_gst_sender.sh /opt/video/input_4K_15fps.mp4  192.174.90.101  5004  1
${sleep_time:-3}
```

```
# chmod +x start_test2.sh
# exit
```

✧ Explanation of arguments for start_gst_sender.sh written in the above script.

- *See paragraph 1.3.3 (1).

ii. In a new window, you open a terminal for the distribution tool, climb into the Pod, and start streaming.

```
$ kubectl exec -n test -it send-video-tool-XXX -- bash  *Change pod name to
suit your environment
# ./start_test2.sh
```

(A CRITICAL log may be output only on the first time, but you can return to the prompt by pressing Enter. If the process startup can be confirmed in the next step, there is no problem.)

iii. Confirm start within the video stream tool's container.

*Verify that one gst-launch is running on ps aux.

```
# ps aux | grep gst-launch
root       2335  0.0  0.0 309636 12272 pts/3    Sl   11:17   0:00 gst-launch-
1.0  filesrc  location=/mnt/input_4K_15fps.mp4  !  qtdemux  !  video/x-h264  !
h264parse   !   rtph264pay   config-interval=-1   seqnum-offset=1   !   udpsink
host=192.174.90.101 port=5004 buffer-size=2048
```

(2) Video input/output check of DataFlow

Confirm using the same procedure as in section 1.3.3 (2).

2.3.4 Video stream/reception stop

Check section 1.3.4.

2.4 Environmental shutdown

Check section 1.4.