

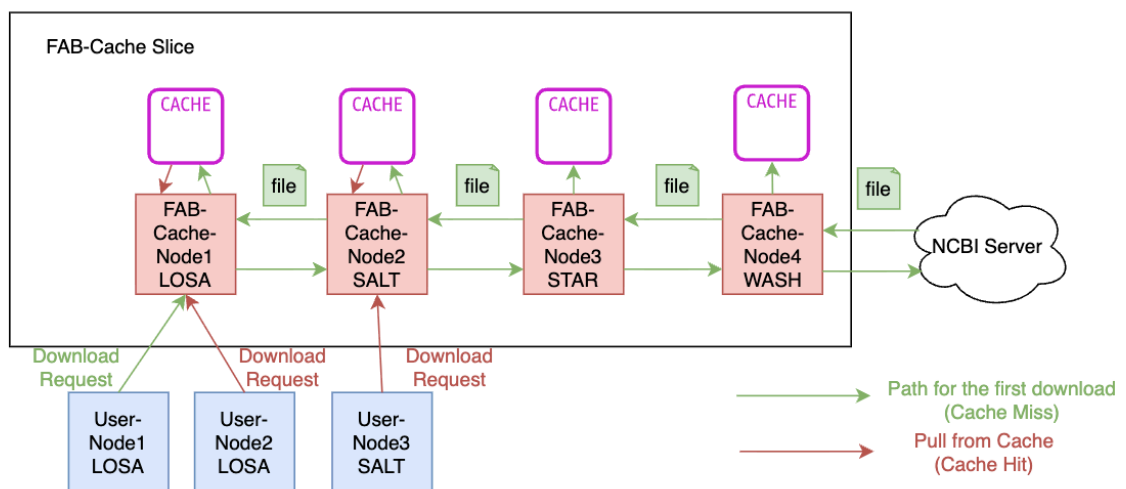
KNIT11 Tutorial: FABRIC In-Network Data Caching Service(FAB-Cache)

The FABRIC In-Network Data Caching Service (FAB-Cache) is designed to take advantage of the Fabric testbed to optimize the downloading and caching of large datasets, such as those from the NCBI website. Instead of each user directly downloading the dataset from the Internet, the system leverages the storage and network resources within the Fabric testbed, deploying a network of strategically distributed caching nodes to significantly enhance both the speed and efficiency of the user's data download experience.

Tutorial

In this tutorial, we will first create a slice with three nodes. Node1 and Node2 will be created on UCSD and connected to FAB-Cache-Node1. Node3 will be created on LOSA and connected to FAB-Cache-Node2.

We will explore the throughput downloading a file from the NCBI website using FAB-Cache and see the difference.



Step1: Create the slice

```
In [ ]: # Pick an url from https://ftp.ncbi.nlm.nih.gov to download
# Example url: https://ftp.ncbi.nlm.nih.gov/ncbi-asn1/

import random

base_url = "https://ftp.ncbi.nlm.nih.gov/ncbi-asn1/"

# Generate list of URLs from gbbct100 to gbbct199
urls = [f"{base_url}gbbct{i}.asn.gz" for i in range(100, 200)]

# Randomly select one
url = random.choice(urls)
```

```
In [ ]: ###Testing: if you want a specific one
#url = "https://ftp.ncbi.nlm.nih.gov/ReferenceSamples/giab/data/AshkenazimTrio/HG002_NA24385
#print(f"Selected URL: {url}")
```

```
In [ ]: node1_site= "LOSA"
node2_site="LOSA"
node3_site="SALT"
fab_cache_node1_ip = "2602:fcfb:12:2::2"
fab_cache_node2_ip = "2602:fcfb:c:2::2"
```

```
In [ ]: ### backup Config1(path: DALL-ATLA-WASH)

#node1_site= "DALL"
#node2_site="DALL"
#node3_site="ATLA"
#fab_cache_node1_ip = "2602:fcfb:b:2::2"
#fab_cache_node2_ip = "2602:fcfb:15:2::2"
```

```
In [ ]: ### backup config2 (path: INDI-STAR-WASH)

#node1_site= "INDI"
#node2_site="INDI"
#node3_site="STAR"
#fab_cache_node1_ip = "2602:fcfb:18:2::2"
#fab_cache_node2_ip = "2602:fcfb:3:2::2"
```

```
In [ ]: ### backup config3(path: UTAH-SALT-STAR-WASH)

#node1_site= "UTAH"
#node2_site="UTAH"
#node3_site="SALT"
#fab_cache_node1_ip = "2602:fcfb:8:2::2"
#fab_cache_node2_ip = "2602:fcfb:c:2::2"
```

```
In [ ]: ### back Config4(path: UCSD-LOSA-SALT-STAR-WASH)
#node1_site= "UCSD"
#node2_site="UCSD"
#node3_site="LOSA"
#fab_cache_node1_ip = "2602:fcfb:d:2::2"
#fab_cache_node2_ip = "2602:fcfb:12:2::2"
```

```
In [ ]: from fabrictestbed_extensions.fablib.fablib import FablibManager as fablib_manager
fablib = fablib_manager()
fablib.show_config();
```

```
In [ ]: slice_name = 'MySlice'

node1_name = 'Node1'
node2_name = 'Node2'
node3_name = 'Node3'

network1_name='net1'
network2_name='net2'
network3_name='net3'

node1_nic_name = 'nic1'
node2_nic_name = 'nic2'
node3_nic_name = 'nic3'

#Create Slice
slice = fablib.new_slice(name=slice_name)

# Node1
node1 = slice.add_node(name=node1_name, site=node1_site)
iface1 = node1.add_component(model='NIC_Basic', name=node1_nic_name).get_interfaces()[0]

# Node2
node2 = slice.add_node(name=node2_name, site=node2_site)
```

```

iface2 = node2.add_component(model='NIC_Basic', name=node2_nic_name).get_interfaces()[0]

# Node3
node3 = slice.add_node(name=node3_name, site=node3_site)
iface3 = node3.add_component(model='NIC_Basic', name=node3_nic_name).get_interfaces()[0]

# Networks
net1 = slice.add_l3network(name=network1_name, interfaces=[iface1], type='IPv6')
net2 = slice.add_l3network(name=network2_name, interfaces=[iface2], type='IPv6')
net3 = slice.add_l3network(name=network3_name, interfaces=[iface3], type='IPv6')

#Submit Slice Request
slice.submit();

```

Step2: Config the nodes

```

In [ ]: network1 = slice.get_network(name=network1_name)
network1_available_ips = network1.get_available_ips()
network1.show()

network2 = slice.get_network(name=network2_name)
network2_available_ips = network2.get_available_ips()
network2.show();

network3 = slice.get_network(name=network3_name)
network3_available_ips = network3.get_available_ips()
network3.show();

```

```

In [ ]: #node1
node1 = slice.get_node(name=node1_name)
node1_iface = node1.get_interface(network_name=network1_name)
node1_addr = network1_available_ips.pop(0)
node1_iface.ip_addr_add(addr=node1_addr, subnet=network1.get_subnet())
node1.ip_route_add(subnet=fablib.FABNETV6_SUBNET, gateway=network1.get_gateway())
stdout, stderr = node1.execute(f'ip addr show {node1_iface.get_device_name()}')
stdout, stderr = node1.execute(f'ip route list')

#node2
node2 = slice.get_node(name=node2_name)
node2_iface = node2.get_interface(network_name=network2_name)
node2_addr = network2_available_ips.pop(0)
node2_iface.ip_addr_add(addr=node2_addr, subnet=network2.get_subnet())
node2.ip_route_add(subnet=fablib.FABNETV6_SUBNET, gateway=network2.get_gateway())
stdout, stderr = node2.execute(f'ip addr show {node2_iface.get_device_name()}')
stdout, stderr = node2.execute(f'ip route list')

#node3
node3 = slice.get_node(name=node3_name)
node3_iface = node3.get_interface(network_name=network3_name)
node3_addr = network3_available_ips.pop(0)
node3_iface.ip_addr_add(addr=node3_addr, subnet=network3.get_subnet())
node3.ip_route_add(subnet=fablib.FABNETV6_SUBNET, gateway=network3.get_gateway())
stdout, stderr = node3.execute(f'ip addr show {node3_iface.get_device_name()}')
stdout, stderr = node3.execute(f'ip route list')

```

Step3: Run the Tests

Test0: Directly download the file over the management network(No FAB-Cache)

```
In [ ]: direct_download_cmd = f"curl -o /dev/null {url}"
```

```
In [ ]: %%time
stdout, stderr= node1.execute(direct_download_cmd)
clean_stdout = "\n".join(line.strip() for line in stdout.split("\n") if line.strip())

print(clean_stdout)
```

Test1: Download the file via FAB-Cache to node1(pull from NCBI)

The first download will follow the path node1->Cache-Node1->Cache-Node2->Cache-Node3->Cache-Node4->->Cache-Node5->NCBI

```
In [ ]: node1_cmd = f"""curl -X POST "https://[{fab_cache_node1_ip}]:8001/download_data" \
-H "Content-Type: application/json" \
-d '{{"url": "{url}", "metric": "throughput"}}' \
--insecure > /dev/null"""
```

```
In [ ]: %%time
stdout, stderr= node1.execute(node1_cmd)
clean_stdout = "\n".join(line.strip() for line in stdout.split("\n") if line.strip())

print(clean_stdout)
```

Test2: Download the file via FAB-Cache to node2(pull from cache on FAB-Cache-Node1)

Once the download on node1 finishes, the data is also cached on FAB-Cache-Node1 so node2 can pull from cache there.

```
In [ ]: %%time
node2_cmd=node1_cmd
stdout, stderr = node2.execute(node2_cmd)
clean_stdout = "\n".join(line.strip() for line in stdout.split("\n") if line.strip())

print(clean_stdout)
```

Test3: Download the file via FAB-Cache to node3(pull from cache on FAB-Cache-Node2)

Since the data is cached along the path, node3 connected to FAB-Cache-Node2 can pull the cache from there.

```
In [ ]: node3_cmd = f"""curl -X POST "https://[{fab_cache_node2_ip}]:8001/download_data" \
-H "Content-Type: application/json" \
-d '{{"url": "{url}", "metric": "throughput"}}' \
--insecure > /dev/null"""
```

```
In [ ]: %%time
stdout, stderr = node3.execute(node3_cmd)
clean_stdout = "\n".join(line.strip() for line in stdout.split("\n") if line.strip())
print(clean_stdout)
```

Test4: Check cached urls and disk space

```
In [ ]: # Check cached urls on fab-cache-node1
import json
```

```

node1_check_cached_url_cmd = f"""curl -s -X GET "https://{fab_cache_node1_ip}:8001/cached_
stdout, stderr = node1.execute(node1_check_cached_url_cmd, quiet=True)

try:
    cached_urls = json.loads(stdout)
    print(json.dumps(cached_urls, indent=4))
except json.JSONDecodeError:
    print("Error: Response is not a valid JSON")
    print(stdout)

```

```

In [ ]: # check cache disk space usage on fab-cache-node1
node1_check_cache_disk_space_cmd = f"""curl -s -X GET "https://{fab_cache_node1_ip}:8001/c
stdout, stderr = node1.execute(node1_check_cache_disk_space_cmd, quiet=True)

try:
    cache_disk_usage = json.loads(stdout)
    print(json.dumps(cache_disk_usage, indent=4))
except json.JSONDecodeError:
    print("Error: Response is not a valid JSON")
    print(stdout)

```

Delete the slice

```

In [ ]: slice = fablib.get_slice(name=slice_name)
slice.delete()

```