ENABLING HPC DESIGN SPACE EXPLORATION VIA SIMULATION

Argonne NATIONAL LABORATORY

CODES/TRACER TUTORIAL: SESSION II

MISBAH MUBARAK
Postdoctoral Researcher
Mathematics and Computer Science Division (MCS)
Argonne National Laboratory

3rd Summer of CODES workshop, TCS Conference Center, Jul 12th 2017

GOALS OF THE SESSION

- How to generate background network traffic?
- How to do storage placement on networks?
- Using model-net API
- Continue with hands on exercises

GENERATING BACKGROUND NETWORK TRAFFIC

WHY BACKGROUND TRAFFIC?

- On production HPC systems, a significant fraction of network nodes can be occupied
- How to introduce communication interference if a single application trace is being replayed on the simulation?
- Running multiple traces at a large-scale can be expensive
- One solution is to mix synthetic traffic patterns and HPC application traces

EXAMPLE SYNTHETIC PATTERNS

- Uniform Random: A network node is equally likely to send to any other network node (traffic distributed throughout the network)
- All to All: Each network node communicates with all other network nodes
- Nearest neighbor: A network node communicates with near by network nodes (or the ones that are at minimal number of hops)
- Permutation traffic: Source node sends all traffic to a single destination based on a permutation matrix
- Bisection pairing: Node 0 communicates with Node ‘n’, node 1 with ‘n-1’ and so on.
- …

SYNTHETIC TRAFFIC IN CODES

```
   /* in case of uniform random traffic, send to a random destination. */
   if(traffic == UNIFORM)
   {
    b->c1 = 1;
    local_dest = tw_rand_integer(lp->rng, 0, num_nodes - 1);
   }
   else if(traffic == NEAREST_GROUP)
   {
    local_dest = (local_id + num_nodes_per_grp) % num_nodes;
    //printf("\n LP %ld sending to %ld num nodes %d ", local_id, local_dest, num_nodes);
   }
   else if(traffic == NEAREST_NEIGHBOR)
   {
    local_dest =  (local_id + 1) % num_nodes;
//    printf("\n LP %ld sending to %ld num nodes %d ", rep_id * 2 + offset, local_dest, num_nodes);
   }
   assert(local_dest < num_nodes);
//    codes_mapping_get_lp_id(group_name, lp_type_name, anno, 1, local_dest / num_servers_per_rep, local_dest % num_servers_per_rep, &global_dest);
   global_dest = codes_mapping_get_lpid_from_relative(local_dest, group_name, lp_type_name, NULL, 0);
   ns->msg_sent_count++;
   model_net_event(net_id, "test", global_dest, PAYLOAD_SZ, 0.0, sizeof(svr_msg), (const void*)m_remote, sizeof(svr_msg), (const void*)m_local, lp);
```

- Typical patterns supported are uniform random and nearest neighbor.
- All to all and stencil patterns have been tested (pending integration)
- See `src/network-workloads/model-net-synthetic-custom-dfly.c and related files`

GENERATING BACKGROUND TRAFFIC WITH CODES

- Communication based on uniform random traffic
- Kicks off when the main workload starts
- A notification is sent to the background traffic node to stop generating traffic once the main workload finishes
- How to enable synthetic traffic generation?
- Simply add “synthetic” instead of DUMPI trace path in workloads config file

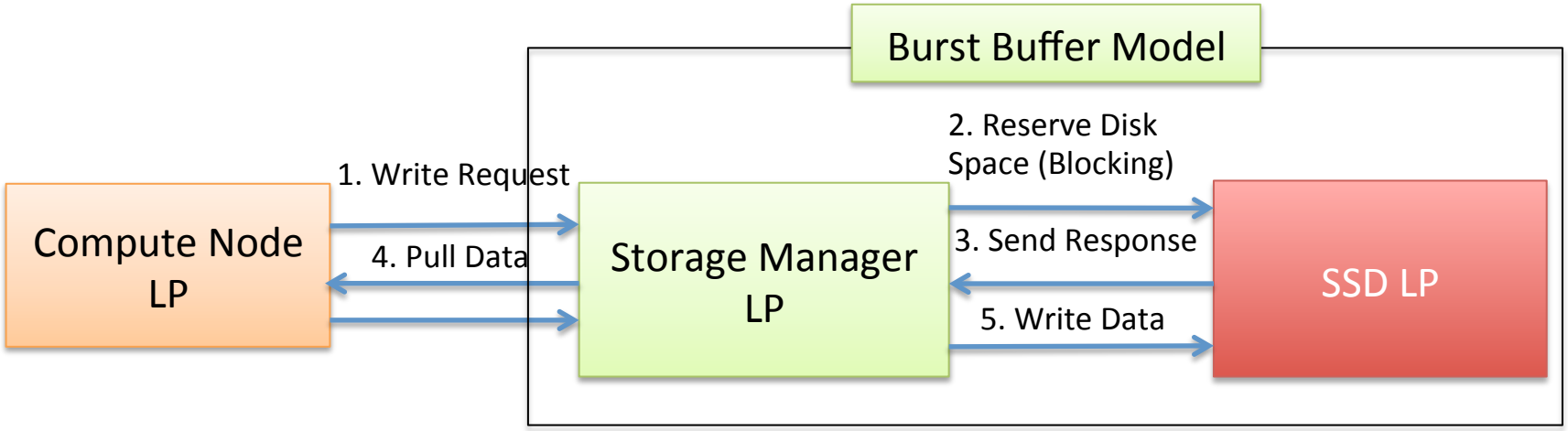
```
216 synthetic
125 /path/to/Multigrid/Multigrid_125/dumpi-2014.03.06.23.48.13-
```

STORAGE PLACEMENT ON INTERCONNECTS

CODES STORAGE MODEL

- General purpose model for read and write operations
- Concurrent, pipelined RDMA requests
- Comprises of the following:
  - a storage manager
  - a disk/local storage model
  - A resource tracker
- Placement of storage over the network can be modified using the network config file

PROTOCOL FOR WRITE OPERATIONS



USING THE STORAGE MODEL

- `codes_store_init_req (is_write, priority, obj_id, xfer_offset, xfer_size, codes_req)` → For initializing the request
- `codes_store_send_req(codes_req, dest_id, sender, network_id, mapping_context, ..)` → For sending the request
- `codes_store_send_req_rc` → For reverse computation

- Repo available at: `https://xgitlab.cels.anl.gov/codes/codes-storage-server`

CONFIGURING STORAGE OVER THE NETWORK

```
# triton server - implements forwarding and acking protocols based on placement
# algorithm
codes-store
{
    # number of threads to multiplex transfers over
    req_threads = "4";
    # buffer size used by each thread
    thread_buf_sz = "1000000";
    # size of ram in bytes
    memory_size = "64000000000";
    # size of storage in bytes
    storage_size = "6400000000000";
    # burst buffer threshold in bytes
    bb_threshold = "1000000000000";
}

# size of LP buffer
resource
{
    available="6464000000000";
}

# params taken from triton-fault-sim configs
lsm
{
    # table of metrics based on requset size
    # here metrics will be used for any request size
    # request size in bytes
    request_sizes   = ("0");

    # write/read rates in MB/s
    write_rates     = ("5700.0");
    read_rates      = ("5700.0");

    # seek latency in microseconds
    write_seeks     = ("2500.0");
    read_seeks      = ("2500.0");

    # latency of completing the smallest I/O request, in microseconds
    write_overheads = ("20.0");
    read_overheads  = ("20.0");
}
```

Number of concurrent requests

Buffer size for each thread

Size of the Memory (RAM)

Storage size (for disk/LSM)

Aggregate memory+storage size

Disk bandwidth/seek configuration

CONFIGURING STORAGE OVER THE NETWORK

```
LPGROUPS
{
    #dragonfly_router can be in one group only!
    DRAGONFLY_GRP
    {
        repetitions = "150";
        codes-store="2";
        lsm="2";
        resource="2";
        test-dummy = "2";
        test-checkpoint-client="60";
        modelnet_simplenet="1";
        modelnet_dragonfly_custom="64";
        modelnet_dragonfly_custom_router="16";
    }
    EXTERNAL_STR_GRP
    {
       repetitions="1";
       codes-external-store="1";
       modelnet_simplenet="1";
    }
}
```

Two storage manager entities per 60 clients/compute nodes

Local storage model entity (disk). One to one correspondence

A total of 64 network nodes

If the data from burst buffer needs to be drained to the external storage entity

USING MODEL-NET API

CONFIGURATION

- Model-net– An abstraction layer on top of network models – topology details are specified through the config files
- A valid network configuration file – examples can be found in the repo
- Network model must be registered – `model_net_register`
- CODES mapping must be setup – `codes_mapping_setup`
- Use model-net function calls – `model_net_event(network id, source, destination, message size,…)`
- Example of using model-net – tests/model-net-test.c

EXERCISES

RUNNING INTERCONNECT SIMULATIONS

- Checkout the exercises at the wiki link:

https://xgitlab.cels.anl.gov/codes/codes/wikis/quick-start-interconnects

THANK-YOU