Notes on Coflow Experiments

Trung Luu

Avignon, March 31, 2021

1 Coflow Generator: inputs

Parameter	*Definition*
N	Number of machines
min_C, max_C	Minimum and maximum number of coflows
min_M, max_M	Minimum and maximum number of mappers
min_R, max_R	Minimum and maximum number of reducers
$Mean\_volume$	Mean volume of flows
$[a, b, c]$	Three possible prefixed capacities for links in the fabric (a, b, and c can be the same to have all links with same capacity)

2 Coflow Generator : generation process

- Build a $N \times N$ fabric based on the Big Switch model (N ingress links, and N egress links)
- Each link has a capacity randomly chosen between a, b, and c
- Randomly choose a number K of coflows between min_C and max_C
- Each coflow is assigned an ID k
- For each coflow k:
  - Randomly choose a number $M_k \in [min\_M, max\_M]$ of mappers
  - Randomly choose a number $R_k \in [min\_R, max\_R]$ of reducers
  - Randomly assign an ingress link to each mapper
  - Randomly assign an egress link to each reducer
  - For each mapper:
    - Create a flow from its ingress link to each reducer's corresponding egress link (if not in the same machine) → at most $M_k \times R_k$ flows
    - Each flow has an ID, its corresponding coflow ID k, and a volume calculated with an exponential law with mean $Mean\_volume$

3 Repository coflow_material_09_02_2021

Note: Need to add of folder and all subfolders to path

Table 1: Folders

Repository	*Definition*
CoflowOrderAlgos	Algorithms to generate the optimal coflow orders • OneRPARIS • WeightBasedAlgos
generated_instances	Archive of results: csvFiles, instances, order
network_elements	Class declaration of `coflow`, `flow`, and `fabric` objects
ResourceAllocationAlgos	greedyAllocation, OnePARIS_Allocation
simulation_config	Configuration of simulations

utils	Necessary tools to generate the instance of `fabric`, `coflows`, *etc.*
	architecture_config = utils.fabric_generation(architecture_config);
	architecture_config = utils.coflows_generation(architecture_config);

Algorithms:

1. One-phase algorithms:

   ResourceAllocationAlgos.varys.varys_offline_basic

   ResourceAllocationAlgos.OnePARIS_Allocation.ONE_PARIS_v3

2. Two-phase algorithms:

   (a) Phase 1: Finding the optimal order (repo: `CoflowOrderAlgos`)
   CoflowOrderAlgos.OneRPARIS.oneParis_v4_sorting(fabric, coflows)
   CoflowOrderAlgos.WeightBasedAlgos.sincronia_BSSI(fabric, coflows)

   (b) Phase 2: Allocation (repo: `ResourceAllocationAlgos`)
   ResourceAllocationAlgos.greedyAllocation.greedyFlowScheduling
   ResourceAllocationAlgos.OnePARIS_Allocation.prioritized_coflows_processing(fabric, coflows, prio_order)

Example:

1. Generate `fabric` and `coflows` objects

   (a) fromCSVToCoflows("test_flow.csv", 3)
   → Output: `generatedCoflowsFabric.mat` (varargin: H configuration of Huawei, `0` by default)

   (b) fromCSVToCoflows_simplified("test_flow.csv", 3, 0, “output_named_by_user")
   → Output: `user_output_name.mat`

2. Load `*.mat` file to get `fabric` and `coflows`

3. Run algorithm

   (a) One-phase algorithm:
   my_out = ResourceAllocationAlgos.varys.varys_offline_basic_v4(fabric, coflows)

   (b) Two-phase algorithm:

   - Phase 1 (obtain order):
     my order = CoflowOrderAlgos.OneRPARIS.oneParis_v4_sorting(fabric, coflows)
   - Phase 2 (final results):
     my out = ResourceAllocationAlgos.OnePARIS_Allocation.prioritized coflows processing(fabric, coflows, my order)