

Caro framework requirements

1. Introduction

1.1 Overview

The carto framework is responsible for cartography of the host, in the first phase, and the entire network in the second phase. The cartography information from the carto framework is needed by other frameworks to let them connect to the network more efficiently. The frameworks that use this information should decide, eventually, which port to use in order to connect to other processes in the job. The carto framework should supply a weighted graph that contains:

- Processors.
- NUMA memory nodes.
- Ports.
- Weights.

The carto framework should:

1. Provide a full cartography graph.
2. Provide a cartography graph per interconnect. I.e. A graph for Infiniband will not include memory nodes and Ethernet ports and a graph for Shared Memory will not include Ethernet and Infiniband ports.
3. Provide a distance list from a node to all other nodes from a certain type.

1.2 Scope

The carto framework will contain several components. The first component is carto-file. This component reads the cartography information from a file provided by the user. The second component is auto-detect. This component discovers the cartography information automatically by reading the information directly from the system. This component will be implemented differently for each OS. The other components will discover the entire network cartography information. These components are T.B.D.

This document specifies the requirements for the carto-file component and for the auto-discovery component. This document does not specify the requirement from the network cartography components or the requirements from the carto framework customers. (E.g. BTL and collective frameworks).

2. Requirements

2.1 Carto general requirements

2.1.1 Carto file format

Since the cartography information is actually a graph, it will be representing in the carto file as adjacency list in the following format:

$$V_1 \quad V_2 : W_2, V_3 : W_3$$
$$V_2 \quad V_1 : W_1, V_3 : W_3$$
$$V_3 \quad V_1 : W_1, V_2 : W_2$$

Where V's are the vertices and W's are the vertices weight. For example: let the host architecture be like in figure 1.

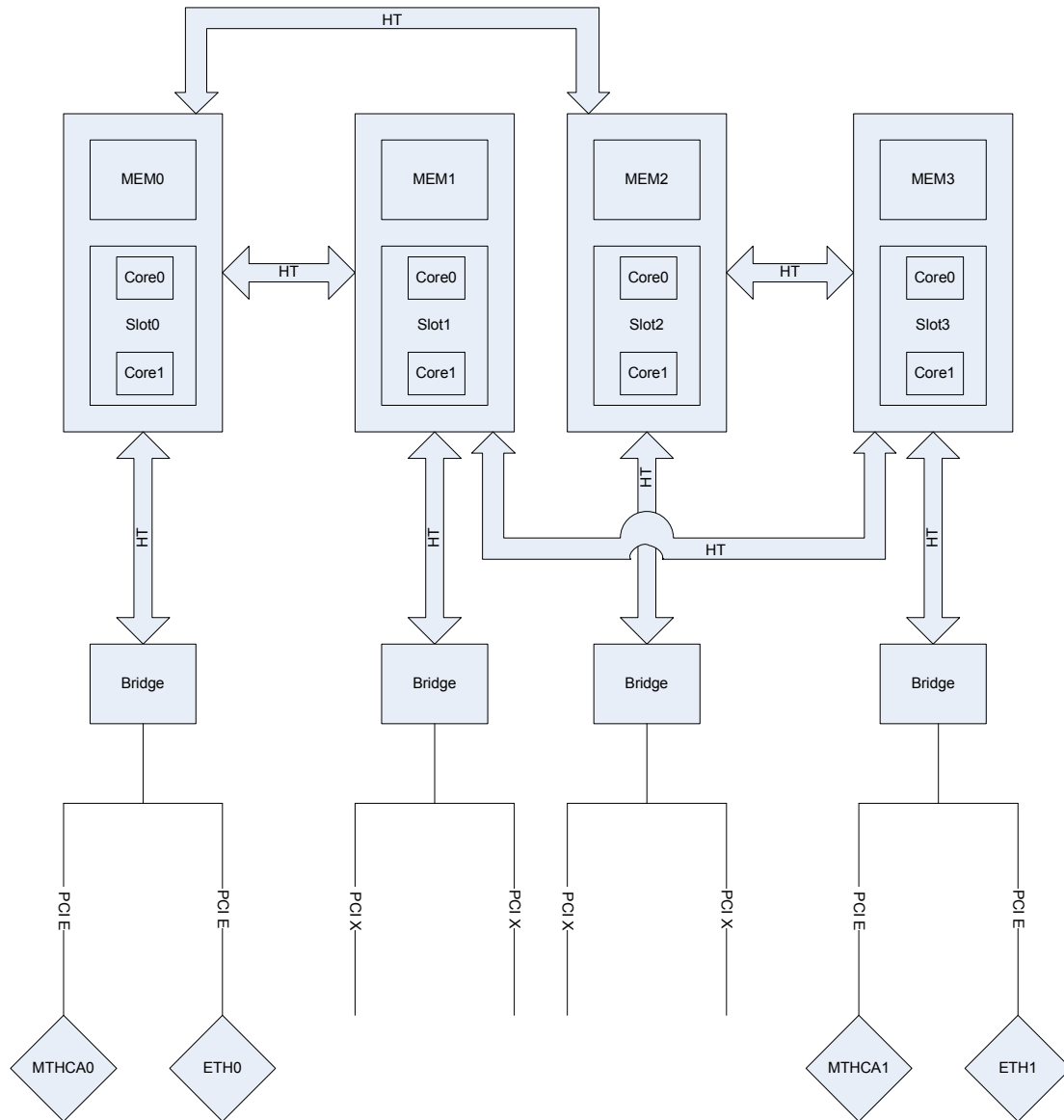


Figure 1

The resulting cartography graph should look like figure 2.

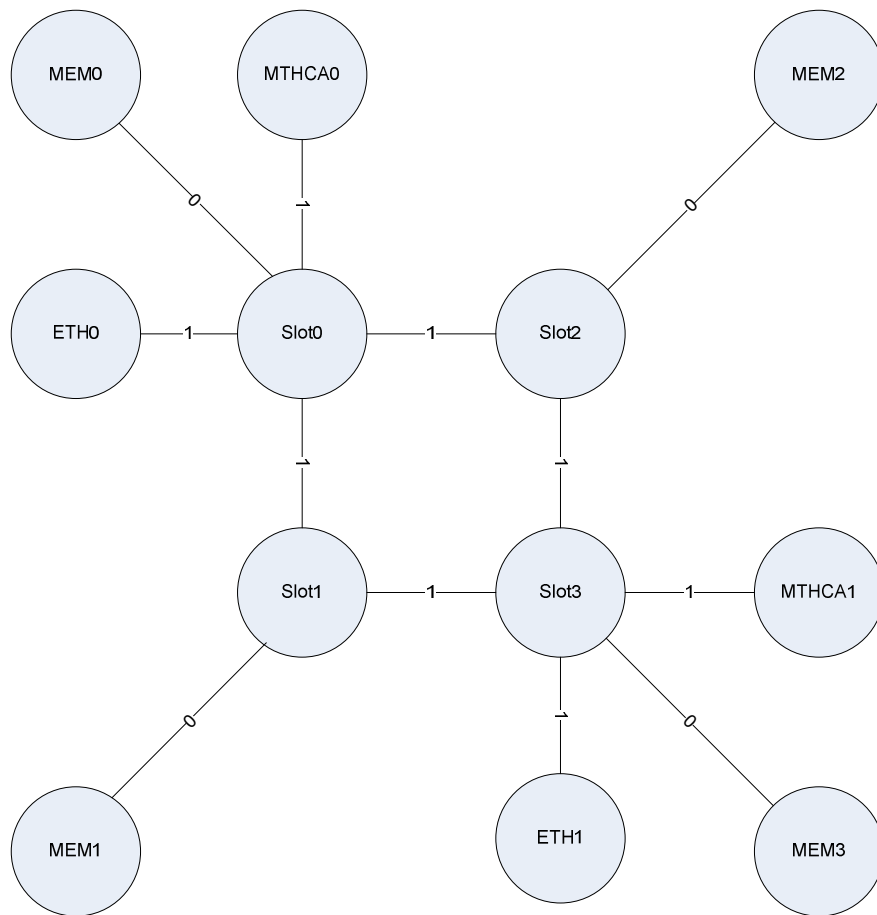


Figure 2

The carto file that represents this graph is shown in figure 3.

#vertex	connected to
MEM0	Slot0:0
Eth0	Slot0:1
MTHCA0	Slot0:1
Slot0	Eth0:1,MTHCA0:1,MEM0:0,Slot2:1,Slot1:1
Slot1	MEM1:0,Slot0:1,Slot3:1
MEM1	Slot1:0
Slot2	MEM2:0,Slot0:1,Slot3:1
MEM2	Slot2:0
Slot3	Eth1:1,MEM3:0,MTHCA1:1,Slot1:1,Slot2:1
Eth1	Slot3:1
MEM3	Slot3:0
MTHCA1	Slot3:1

Figure 3

The vertices names in the carto file should be:

- For memory nodes: MEM<Number>. Where the numbers are an arbitrary number to distinguish between memory nodes.
- For processors: Slot<Number>. Where the numbers are the slot number read from the system (i.e. read from components like PLPA)
- For ports :<Port name>. the port name the BTL will read from the system (i.e. using calls like: `ibv_get_device_list` or `SIOCIFCFG ioctl`)

2.1.2 Database

The carto framework will have an internal global (global in the framework) database that contains the cartography graph.

2.2 *Carto initialization*

The main difference between the carto components (at least in the first phase) is in the initialization of the component. At the end of the initialization the carto framework should have a cartography graph ready to use by all other frameworks.

2.2.1 Carto file component initialization

In initialization, the carto file component reads the carto file and builds the cartography graph in the database.

2.2.2 Auto detect component initialization

In initialization, the auto detect component detects the host cartography by using system calls and components like PLPA (in Linux). The system cartography detection is T.B.D

2.2.3 Network carto component initialization

T.B.D

2.3 *Carto interfaces*

2.3.1 `carto_module_init`

- **Purpose:** Initialize the carto framework.
- **In:** None
- **Out:** Error code (int)
- **Operation:** See 2.2

2.3.2 `carto_module_finalize`

- **Purpose:** finalize the carto framework.
- **In:** None
- **Out:** Error code (int)
- **Operation:** Clears the carto database, frees all the allocated memory and unregisters.

2.3.3 `carto_get_host_graph`

- **Purpose:** Returns the host cartography graph.
- **In:**
 - Graph type. An enumeration that includes:
 - MEM –for SM BTL
 - IB – for openIB BTL
 - ETH – for TCP BTL
 - ALL – for the entire graph
 - An unallocated pointer for the graph

- **Out:** Error code (int)
- **Operation:** The get host graph should allocate memory for the graph, strip un relevant data from the graph (according to graph type) and return the graph.

2.3.4 carto_get_nodes_distance

- **Purpose:** return a sorted list of the nodes (memory nodes, processors and ports) and their “distance” (weight) from a selected node. The list is sorted according the distance. From the closest to the most far.
- **In:**
 - Node type. An enumeration that includes:
 - MEM –for SM BTL
 - IB – for openIB BTL
 - ETH – for TCP BTL
 - SLOT – for processors.
 - ALL – for the entire tree
 - The selected node. All the distances are measured from this node. the structure of a node is:
 - An enumerator to specify the node type. (like the one above)
 - the name of the node.
 - An unallocated pointer to a list of node names and distance (opal_list_t*)
- **Out:** Error code (int)
- **Operation:** The get node distance build a list of nodes and their distance according the node type and the shortest paths on the carto tree. In our example, if we running on slot 1 core1, the list of memory nodes is:
 - MEM1:0
 - MEM0:1
 - MEM3:1
 - MEM2:2

2.4 Carto configuration

The configuration of the carto framework is done by using MCA parameters.

2.4.1 Carto file path

The location of the carto file is specifying by using the **carto_file_path <PATH>** MCA parameter.

2.4.2 Carto component selection

By default the carto framework uses the carto file component. The user can change this by using **carto autodetect** or **carto net** MCA parameter.

2.5 Carto Operation

In the first phase, there is no operation that the carto framework is needed to do.