

# Supercomputer within supercomputer: discovery of interconnect topologies by complete supercomputing system emulation

Lukasz P. Orłowski

lukasz.orłowski@stonybrook.edu<sup>1, 2, 3</sup>

<sup>1</sup>A\*STAR Computational Resource Centre

<sup>2</sup>Dept. of Applied Mathematics and Statistics  
Stony Brook University

<sup>3</sup>Institute of Advanced Computational Science  
Stony Brook University

January 23, 2017

## **Abstract**

Computers are entering post-Moore era, where the number of transistors on a processor chip no longer doubles every 18 months. Nevertheless, the need for more powerful supercomputing systems is ever increasing. Performance of supercomputers may be significantly improved by choosing appropriate topologies of interconnect networks. Since one can no longer solely rely on improvement in the processor performance, I turn my attention to topologies of interconnects.

Supercomputing network topologies are designed to serve a wide variety of computational problems. This is a compromise between performance and general purpose application of supercomputers. Some

computational problems perform better on systems with a particular interconnect topology e.g. stencil computations with periodic boundary conditions excel on torus networks. This shows that for a computational problem, there may exist an optimal interconnect topology. Exploration and discovery of interconnect topologies best suited for common computational challenges is the key factor driving my research.

I propose emulation of entire supercomputing system covering network, compute and storage. In such approach parameters characterising CPU, memory, network and storage, which in a physical system are either invariant or difficult to modify, become variables. With an emulator, not only may one observe how altering one parameter affects the others, but also one may modify those parameters while the system is running. Variety of network or node failure and malfunction scenarios may be simulated with such tool. Furthermore an emulated environment permits to observe how different versions of operating system, kernel, software stack or drivers affect the performance of a system, which is impossible to model mathematically.

To attain abstraction over network, CPU, memory and storage I use software-defined networking and virtual machines. The target platforms for the initial implementation of supercomputer emulator are symmetric-multiprocessing systems.

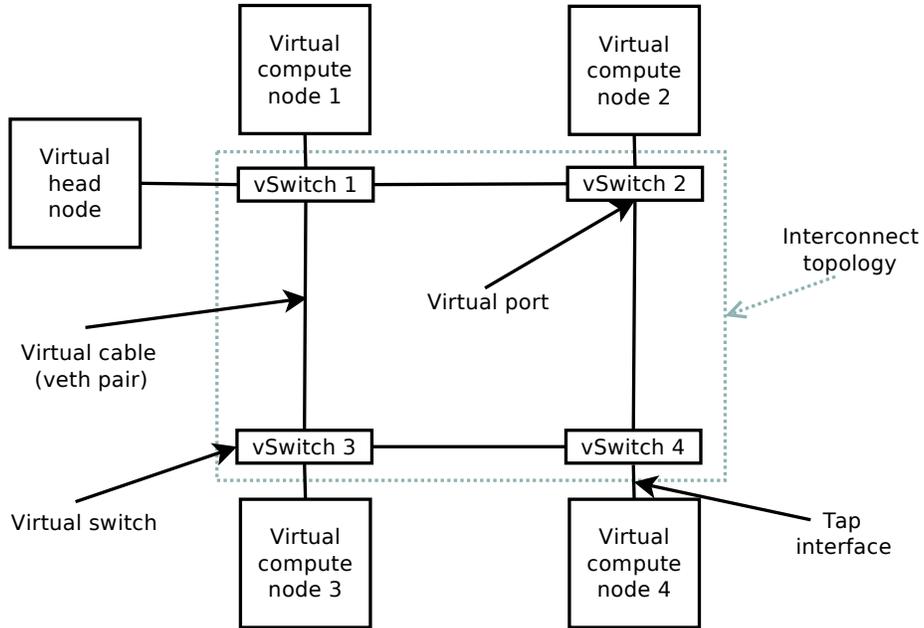


Figure 1: Architecture diagram for sample  $2 \times 2$  torus topology

The virtual interconnect network is spanned between virtual switches, which are governed by a topology-aware software-defined networking (SDN) controller. Every virtual switch directly connects to exactly one virtual compute node over a TAP interface. Additionally, one virtual switch is connected to a virtual head node and an SDN controller. The switches are interconnected with virtual Ethernet pairs, or veth pairs in short, which enables traffic shaping capabilities (see fig. 1). The virtual interconnect is used *only* for interprocess communication during computation.

Connecting virtual compute nodes to corresponding switches makes the interconnect topology transparent for the compute nodes, thus once the network addresses are assigned, change in interconnect topol-

ogy becomes a concern of the SDN controller and its topology management engine. This feature combined with in-memory compute node operating system images, makes virtual compute nodes stateless. Statelessness helps to avoid artefacts from previous runs.

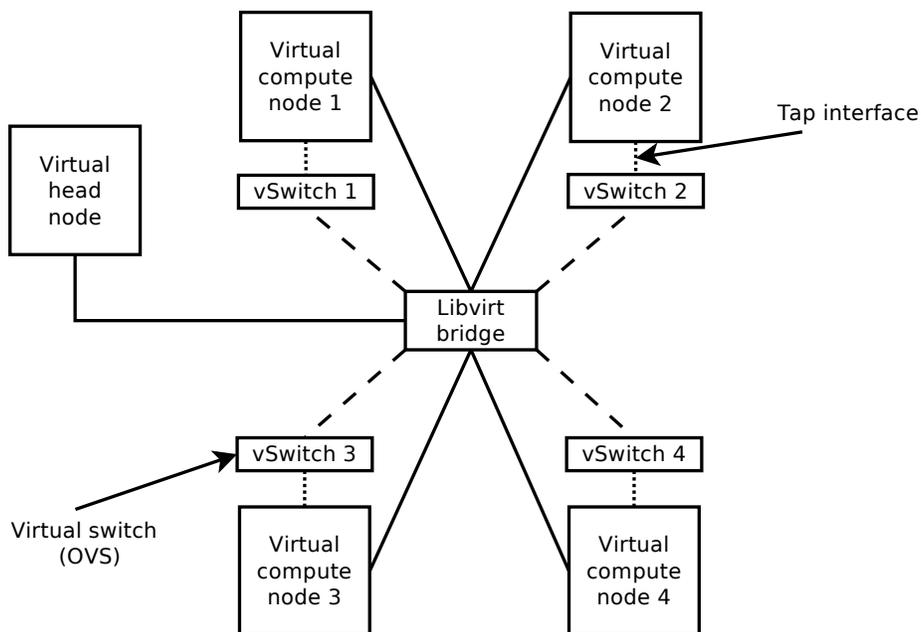


Figure 2: Management network diagram for 4-node deployment

Additionally to the virtual interconnect, the emulator implements a management network for OpenFlow traffic, network storage, virtual machine management, etc. (see fig. 2). This approach keeps the virtual cluster operational even when the virtual interconnect is disconnected or in the process of being rewired.

Current implementation has been tested on 96 single-core virtual compute nodes and generates reproducible results and I am working on compiling a matrix of problem-to-topology mappings.