Technical perspective: DFI: The Data Flow Interface for High-Speed Networks

Gustavo Alonso
Systems Group
Computer Science Department
ETH Zurich
alonso@inf.ethz.ch

Optimizing data movement has always been one of the key ways to get a data processing system to perform efficiently. Appearing under different disguises as computers evolved over the years, the issue is today as relevant as ever. With the advent of the cloud, data movement has become *the* bottleneck to address in any data processing system. In the cloud, compute and storage are typically disaggregated, with a network in between. In addition, cloud systems are scale-out, i.e., performance is obtained by parallelizing across machines, which also involves network communication. And while it is possible to use machines with large amounts of memory, the pricing models and the virtualized nature of the cloud tends to favor clusters of smaller computing nodes. Nowadays, the problem of optimizing data movement has become the problem of using the network as efficiently as possible.

Unfortunately, optimizing the network is easier said than done. Network communication is actually computationally quite expensive. As network bandwidth grows and the potential number of concurrent flows (connections) increases, the computational power required to run the network protocol can be quite high. In the cloud, this problem is made more acute by the need to support many concurrent virtual machines on the same physical node (implying many more connections) and the necessity of virtualizing the network, which adds considerable overhead. Such an overhead has led to the proliferation of smart NICs where most, if not all, of the computation related to the network protocol is offloaded, thereby freeing the CPU from the task.

It is in this context that Remote Direct Memory Access (RDMA) plays a big role. As the name implies, RDMA enables the transfer of data directly across the memories of the sender and receiver with no (single-sided operations) or only minimal (two-sided operations) intervention of the CPU since the transfer is managed by the NIC. Doing so removes many of the inefficiencies of conventional network protocols (involvement of the operating system which introduces overhead for system calls and context switches, need to copy the data from the network buffers to the operating system and then to user space, need to reserve CPU capacity to process network packages, etc.), leading to significant lower latencies.

Unfortunately, nothing is perfect. RDMA offers only a low level interface that requires the developer to manage a great deal of complexity. In high performance computing, this has been addressed through the development of libraries such as MPI. However, MPI is surprisingly unsuitable for conventional applications for a wide variety of reasons (the very restrictive computational model, poor support for multithreading, etc.). Thus, it seems to make sense to develop an equivalent to MPI for data processing using RDMA. The paper highlighted in the next section does exactly this by proposing the DFI (*Data Flow Interface*), a library that uses RDMA in the background to provide a declarative way to specify data exchanges across nodes specifically tailored to data processing applications. DFI abstracts away many of the elements of RDMA and replaces them with data structures and interfaces more suitable to data management. For instance, RDMA requires to register memory in advance with the NIC so that the NIC can actually operate on that region of memory without CPU or OS involvement. This adds overhead to every exchange and is difficult to manage. In data processing, the amount of data to transfer depends on the selectivity, which varies greatly from query to query. This would imply that a large part of memory would have to be reserved for RDMA exchanges in case a lot of data is involved even if it is not often used. Given the demand for main memory in today's systems, such overprovisioning would be highly problematic. The DFI addresses this problem by automatically introducing and managing ring buffers where data can be continuously be sent without having to register new regions mid-way through the exchange or having to resort to overprovisioning. Similarly, using RDMA directly, developers need to take care of partitioning the data across nodes and of routing the data as needed. DFI takes care of such tasks by supporting pluggable partition operators and also letting the developer indicate the type of exchange to implement in a declarative manner.

The paper makes a convincing case that DFI is more suitable for data processing than existing solutions by presenting several use cases where DFI takes care of complex tasks that RDMA would force the developer to manage manually. The performance numbers are impressive, showing the DFI does not add any significant overhead and that sometimes leads to better results as some of the complex steps required when using RDMA directly are not longer needed. With the growing number of systems and research relying on RDMA, DFI is a valuable contribution that opens up many possibilities for building systems but also by enabling research on network optimizations for data processing.