High-performance communication fabric for a dataflow system

Master Thesis Proposal

Modern dataflow systems employ a push-based (data-driven) communication paradigm where operators are eagerly evaluated instead of waiting for demand from dependent downstream operators. This asynchronous, many-to-many communication is different from typical HPC applications, which are often structured as a sequence of stages separated by communication rounds that also act as a synchronisation barrier.

The design space for an high-performance communication fabric in low-latency data-driven dataflow systems appears to be largely unexplored in the literature. We believe we can leverage modern network hardware, the information on computation structure and progress information from the system to build a networking layer tailored to these systems.

There are multiple engineering and research questions to explore in this area:
(i) what networking platform and API is most suited for dataflow? (Sockets, DPDK, RDMA, ...)
(ii) how should the communication fabric be structured? Do we need multiple layers of (de-) multiplexing of messages headed to different destinations?
(iii) how do we tame the inherent burstiness of dataflow messaging and how do we avoid overloading the hardware with too many commands and messages? Can we perform intelligent message pooling (“boxcar-ing”)?

This Thesis will look at the design for an high-performance communication fabric for Timely Dataflow, a data-parallel programming framework designed to efficiently implement complex algorithms on large volumes of data.

System architecture. In Timely worker threads are logically connected by a network fabric that allows arbitrary point-to-point connection.

All workers are peers and run the same program. Each worker (i) consumes incoming messages (from all senders), (ii) performs some computation, and (iii) sends messages to multiple (often all) destinations. These three operations may be interleaved. A worker has an inbox where incoming messages are deposited (senders should never block waiting for a receiver) and a worker should not be expected to process incoming messages immediately. Sending/receiving can take the form of (i) a write to a shared-memory queue, (ii) a write to a pre-allocated buffer, (iii) a “write”/”send” syscall, etc. In general, a worker requires a way to deposit some bytes into a buffer so they can be delivered to a specific destination.

Progress Tracking in Timely Dataflow is a lightweight coordination mechanism that keeps track of tuples in the system and enables operators to determine whether data dependencies have been fulfilled and it’s safe to proceed. This peer-to-peer protocol conceptually performs an asynchronous all-reduce operation repeatedly to establish a global view of progress across the system. In addition to the messaging fabric, we have the opportunity to explore how to best provide the communication primitives for this critical component.

Related work The Volcano database query processor was parallelised with a dataflow-like, push-based architecture: this paper is an excellent introduction to the issues that arise in implementing a communication fabric across operators that span multiple cores and machines.

1. G. Graefe. “Parallelizing the Volcano database query processor”. In: Digest of Papers Compcon Spring ’90. Thirty-Fifth IEEE Computer Society International Conference on Intellectual Leverage (). DOI:10.1109/cmpcon.1990.63729 URL:

If you are interested in this project please contact Andrea Lattuada (andrea.lattuada@inf.ethz.ch) or Frank McSherry (frank.mcsherry@inf.ethz.ch). The proposed thesis will be supervised by Prof. Timothy Roscoe.

1https://github.com/frankmcsherry/timely-dataflow