A Database Guy's Journey Into RDMA Multicast

OFA Workshop 2016

April 4-8, 2016, Monterey, CA

Christian Tinnefeld
SAP Labs, Inc.
3410 Hillview Avenue
94304 Palo Alto, United States
christian.tinnefeld@sap.com

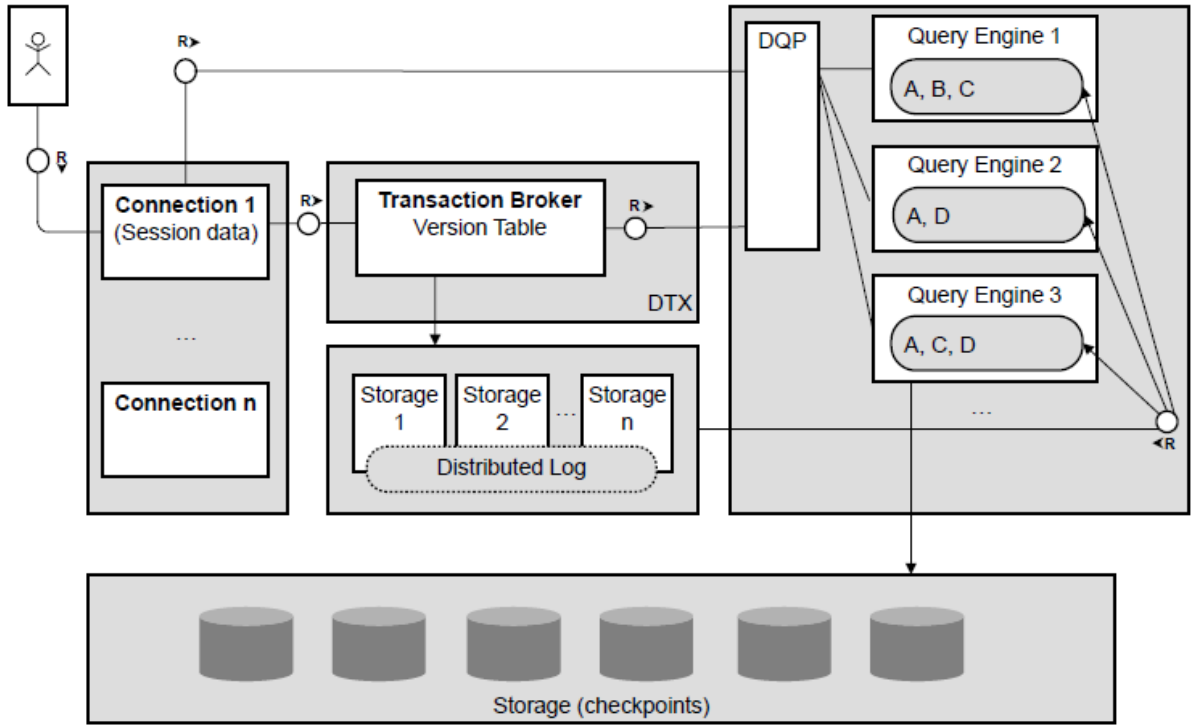
Markus Dreseler
Hasso Plattner Institute
University of Potsdam
August-Bebel-Str. 88
14482 Potsdam, Germany
markus.dreseler@hpi.de

Content

1. Motivation
    1. Project Hana Vora at SAP
    2. Which communication framework to choose?
2. Multicast and group communication operations in a distributed DBMS
    1. Which DBMS scenarios and operators can benefit?
    2. What kind of payload do we have?
    3. What are open / unanswered questions?
3. RDMA multicast
    1. Benefits of hardware support for multicast
    2. Experimental setup
    3. Insights
4. Conclusions

Part 1) Project Hana Vora at SAP

- Vora is a distributed computing platform built on in-memory technology to scale to 1000s of nodes using commodity hardware both on-premise and in cloud deployments.
- Based on data-centric JIT compilation of SQL queries to byte / C / machine code with LLVM
- Uses NUMA-aware data structures and algorithms
- Extensibility: Provide easy to use APIs available for C, C++, go, Python, Java, Scala etc.
- Written from scratch but shares concepts from other in-memory projects
- Tight integration with Apache Hadoop ecosystem and Spark



Part 1) Communication framework requirements

- From an application developer's perspective
  - Dead-simple interface
  - As few instructions as possible
  - One code line, best usage of available networking hardware
  - Additional language bindings (e.g. GO, python)
- Aware of NUMA regions, memory affinity for addressing and dispatching
- TCP stack that works anywhere (no matter which OS, hardware etc.)
- RDMA stack that supports
  - Rendezvous
  - Scatter/Gather for non-consecutive memory regions
  - Atomic operations
- (MPI-like) group communication operations
  - Basic one-to-all
  - Advanced all-to-all, barrier etc.
- Integration with external polling context
- Integration with custom memory allocators

Part 1) IB-verbs / Accelio / libfabric / MPI

- IB-verbs
  - Low-level and verbose
  - Take care of things you take for granted (e.g. flow control)
  - Comparatively low developer productivity / easy to make mistakes
- Accelio / libfabric
  - No support of all popular operating systems
  - No support for any group communication operations (broadcast, multicast)
- MPI
  - Mainly intended as tool for HPC community
  - Although there are ways to ensure fault-tolerance, dynamic scheduling rarely used in distributed data processing projects

Part 2) Which scenarios can benefit from group communication operations?

- Transfer of data partitions
  - Needed for load balancing, scale-out, recovery in failure cases
  - Requirements: High bandwidth, **basic one-to-many operations**, work on pinned memory, address memory on a per-NUMA level
- Accessing (meta) data in the catalog
  - Needed for storing table meta data, cluster information, possibly data dictionaries etc.
  - Requirements: Smart serialization needed, frequent operations on small data items, latency matters
- Distribution of intermediate results during query execution
  - Needed for supporting the exchange operator and execution of complex distributed operations
  - Requirements: High bandwidth, work on pinned memory, address memory on a per-NUMA level, extensive use of group communication operations (**one-to-many, many-to-one, all-to-all**)
- Storing and accessing data in the distributed log
  - Needed to provide persistent data storage
  - Requirements: similar as transfer of data slices plus RDMA atomic operations (e.g. increment operation for the sequencer)
- Transaction processing
  - Needed to provide some sort of isolation level, ACID properties
  - Requirements: Heavily exploit low latency, **basic one-to-many operations,** focus on RDMA-specific operations for acquiring/releasing locks etc.
- Check pointing intermediate results
  - Needed for storing intermediate results during query execution for failover cases
  - Requirements: High bandwidth, **basic one-to-many operations**, smart memory management (e.g. when are intermediate results no longer needed)
- In general: deployment of Vora
  - We want to be able to deploy Velocity anywhere
  - Requirements: Network stack must run on Windows, Linux, Mac, x86, ARM, high-end server etc.

Part 2) Joins + Multicast

- Back-of-the-envelope calculations how much data gets transferred with each group communication operator invocation. The goal is to get feedback from Intel what is feasible/what is not.
- We take join executions as example with a right semi join performed on relations (aka columns) *R* and *S* where the result includes the matching positions in S. We assume that one positional entry is of 8 bytes size
- We assume that *R* holds 100.000.000 records per node and *S* holds 10.000.000 records per node (*R* = 10x *S*). The record size is 4 bytes. We assume the selectivity of the join is 10%. The amount of records grows linearly with the number of nodes.
- The joins can be either executed via Grace Join or Distributed-Block-Nested-Loop Join (DBNL Join). The execution of the algorithms can benefit from the group communication operators in the following ways:
  - Grace Join:
    - initial redistribution of to be joined relations: multicast
    - consolidation of join result: gather
  - DBNL Join:
    - (non streaming) replication of relation *S*: multicast
    - (streaming of S, decentralized orchestration): redistribution of *S*: scatter
    - (streaming of S, centralized orchestration): redistribution of *S*: all-to-all
    - consolidation of join result: gather

Part 2) Joins + Multicast

		10 nodes	100 nodes	1000 nodes	10000 nodes
Grace Join	redistribution of to be joined relations: **scatter** (one invocation per relation per node) Data transfer in GB	20 operator invocations total 4.4 GB total	200 operator invocations in total 44 GB total	2000 operator invocations in total 440 GB total	20000 invocations in total 4400 GB total
	consolidation of join result: **gather** Data transfer in GB per operator invocation	0.08 GB	0.8 GB	8 GB	80 GB
DBNL Join	(non streaming) replication of relation S: **multicast** Data transfer in GB	10 operator invocations total 0.4 GB total	100 operator invocations total 4 GB total	1000 operator invocations total 40 GB total	10000 operator invocations total 400 GB total
	(streaming of S, decentralized orchestration): redistribution of S: **scatter** Data transfer in GB per operator invocation	10^2 operator invocations total 0.4 GB total	100^2 operator invocations total 4 GB total	1000^2 operator invocations total 40 GB total	10000^2 operator invocations total 400 GB total
	(streaming of S, centralized orchestration): redistribution of S: **alltoall** Data transfer in GB per operator invocation	10 operator invocations total 0.4 GB total	100 operator invocations total 4 GB total	1000 operator invocations total 40 GB total	10000 operator invocations total 400 GB total
	consolidation of join result: **gather** Data transfer in GB	0.08 GB	0.8 GB	8 GB	80 GB

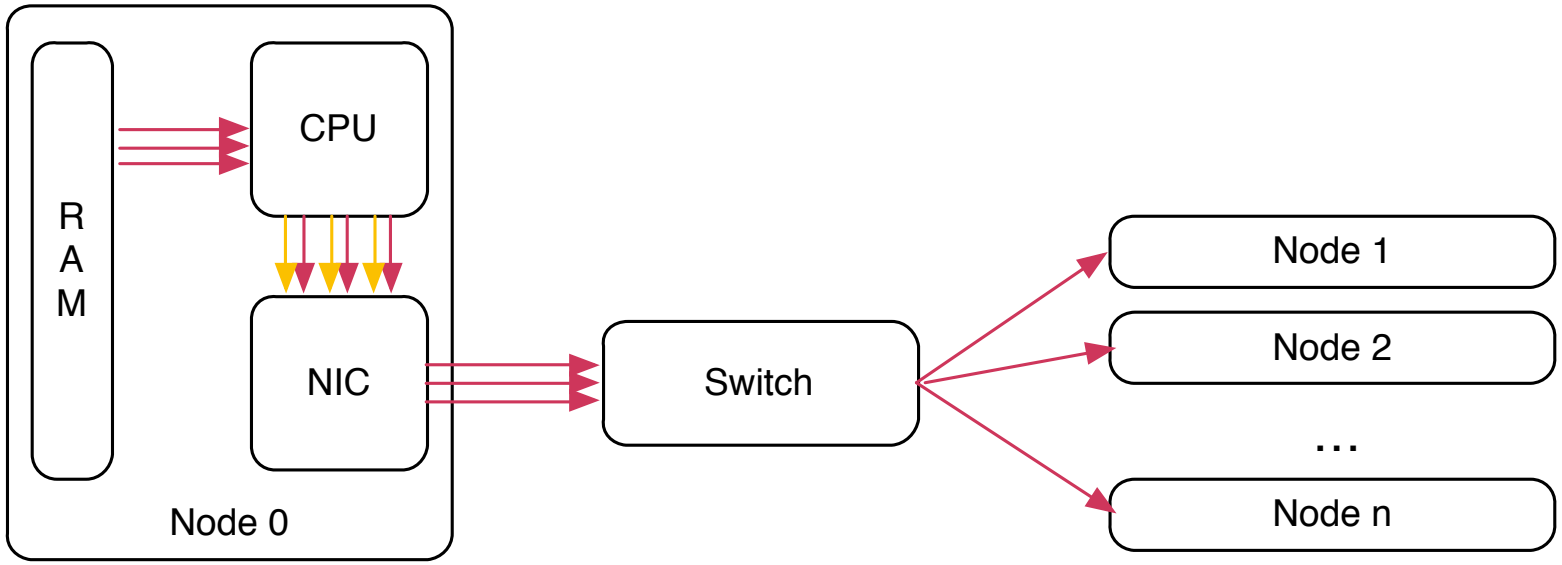
Note: we ignore that in practice the overall data transfer amount and the number of operator invocations would be 1/(number of nodes) smaller

Part 2) Open Questions

- How well do multicast operations scale…
  - …with varying payload sizes?
  - ...with a varying number of participants?
  - ...with different levels of congestions?
- What are the costs for creating/modifying multicast groups?
- What are the performance benefits in comparison to unicasts?

Part 3) Hardware support for multicast

Traditional (Unicast) Messaging

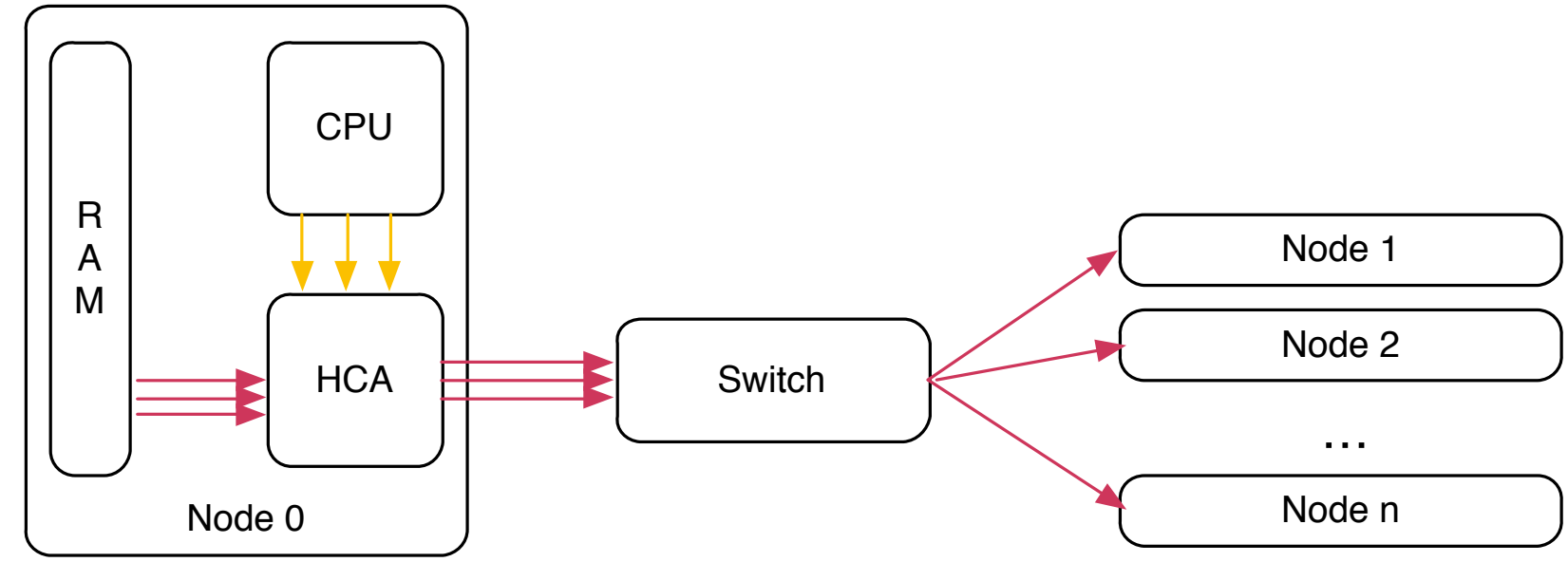


Data Flow

"Control" Flow

Part 3) Hardware support for multicast

Unicast Low-Latency Messaging

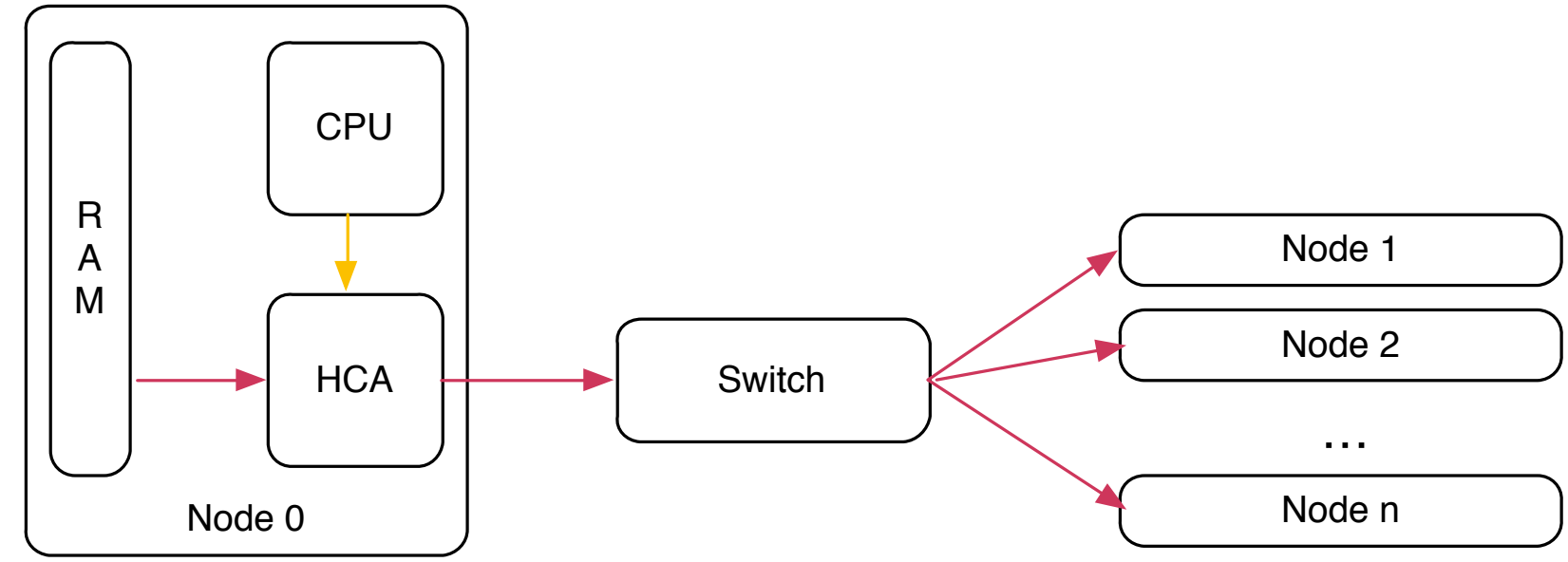


→ Data Flow

→ “Control” Flow

Part 3) Hardware support for multicast

Multicast Low-Latency Messaging

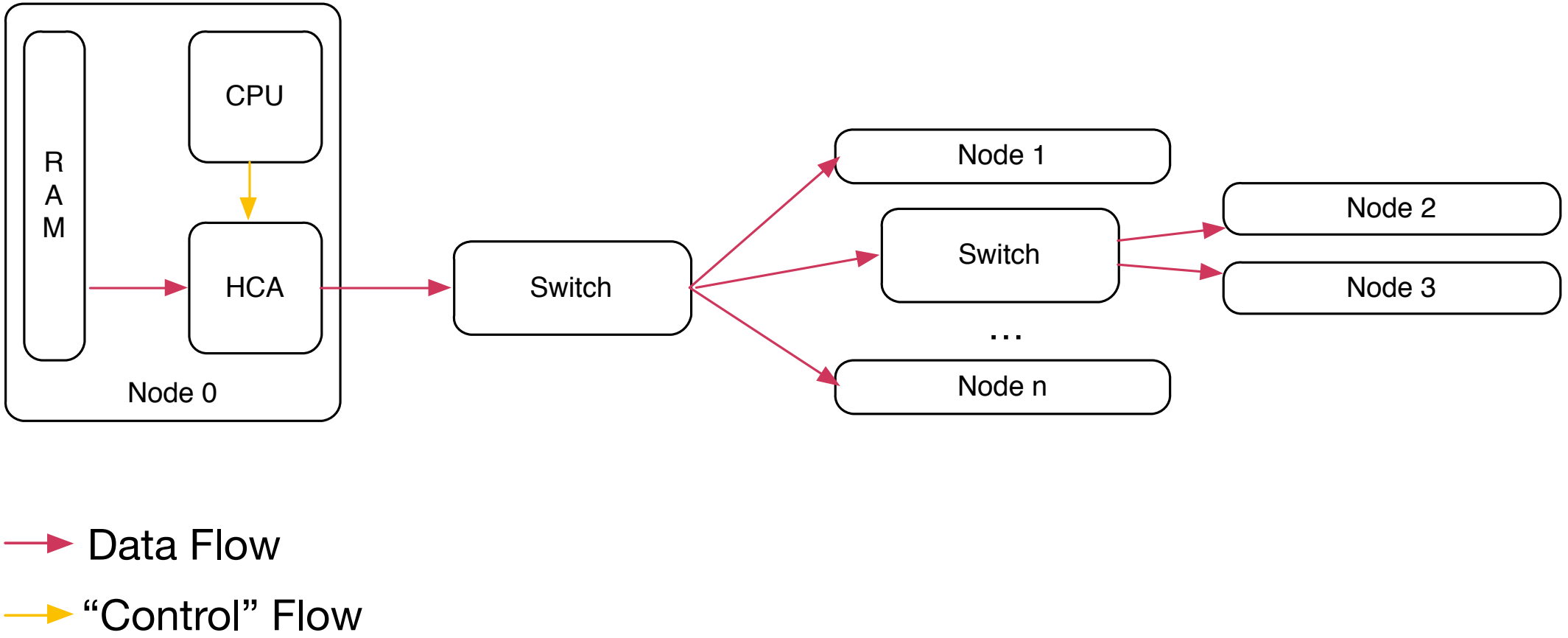


→ Data Flow

→ “Control” Flow

Part 3) Hardware support for multicast

Complex Multicast Low-Latency Messaging



Part 3) How to make use of it?

- Nodes have to join multicast groups in order to receive messages
- Right now, only ibverbs (the lowest-level library) supports this out-of-the-box
- rdma_join_multicast(id, addr, [context])
- id is similar to a network socket, addr is the IB multicast address

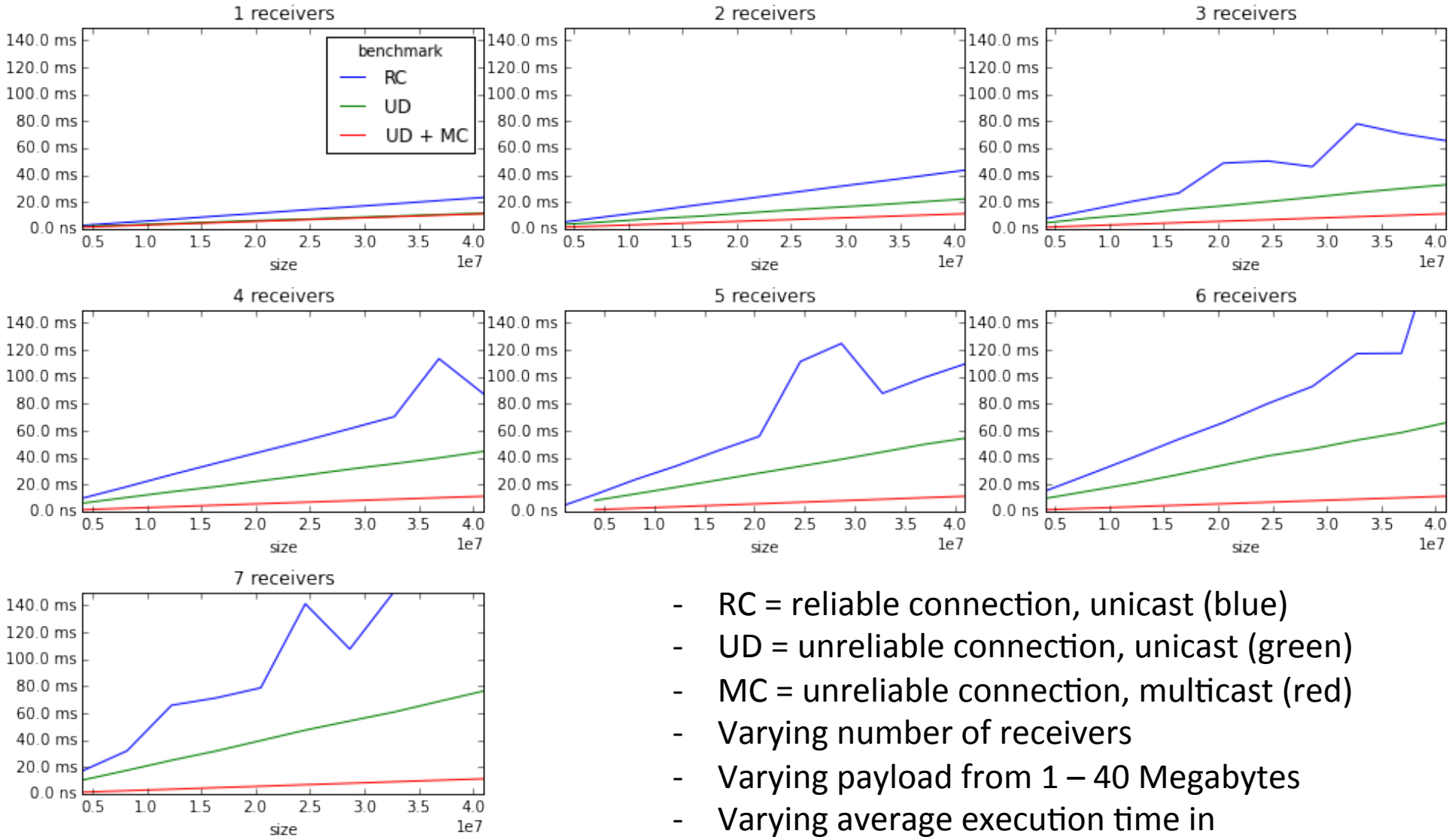
Part 3) Management of multicast groups

- The join method asks the subnet manager to program the switches according to the multicast configuration
- From then on, everyone may send datagrams to the multicast group, even when they are not part of it
- One cycle of joining and leaving MC groups takes approx. 400 us

Part 3) Benchmarks

- Setup
  - 8 phyiscal nodes equipped with Intel Xeon CPUs
  - Mellanox ConnectX-3 VPI adapter; single-port QSFP; QDR IB (40Gb/s) and 10GigE
  - Mellanox SwitchX®-2 InfiniBand Switch
  - Ibverbs + rdmacm, based on mckey
- Big thanks to Intel and Karthik Kumar for sponsoring the cluster and fruitful discussions

Part 3) Benchmarks



- RC = reliable connection, unicast (blue)
- UD = unreliable connection, unicast (green)
- MC = unreliable connection, multicast (red)
- Varying number of receivers
- Varying payload from 1 – 40 Megabytes
- Varying average execution time in milliseconds

Conclusions

- Communication framework
  - No one-size-fits-all framework available yet
  - Accelio / libfabric:
    - lack of supporting standard operating systems
    - no group communication support as of now
- Group communications
  - MPI is the top dog
  - Relevant for modern distributed data processing systems as they constantly grow in size
  - Currently, no/very little use of any group comm. patterns in modern data processing projects
  - Chicken / egg problem:
    - Network vendors make amazing hardware
    - Advanced features are hard to use (please use ibverbs if you want multicast)
    - No clear quantification of benefits of modern hardware

Thank you!

christian.tinnefeld@sap.com