Completion Queue Handler Group for User Verbs Applications

Guangyu Shi, Pallab Bhattacharya
Oracle
Table of Content

• Introduction
  – Polling on Completion Queue (CQ) vs. Event mode
  – Problem with random interrupt assignment

• CQ Handler (CQH) Group for Uverbs
  – Concept
  – Implementation

• Experiment
  – OFUV micro-benchmark
  – OLTP benchmark

• Conclusion + Future work
Introduction

• InfiniBand (IB) Unified I/O Architecture
  – IB support running a large number of protocols concurrently on the same physical device

• OS must act as a nice arbitrator
  – It needs to allocate resources properly among protocols to ensure fairness and Quality of Service (QoS)

• However, OS cannot arbitrate completion event (device interrupt) based on IO protocol itself
Introduction: Polling on CQ

Polling on CQ wastes CPU cycles.
Introduction: Event mode

Event mode is CPU-efficient
Introduction: Problem with random interrupt assignment

- Evil app #1: Massive QP creator
Introduction: Problem with random interrupt assignment

- Evil app #2: High frequency event generator
CQ Handler (CQH) Group: Concept

- Group a set of vectors for a given type of IO.
- When there is a completion event, IB device will use only the interrupt vectors in its group.
  - Implemented in Oracle Solaris for kernel ULPs
CQ Handler (CQH) Group: Concept
CQ Handler (CQH) Group: Concept

Interrupt vectors

HCA

Red cqh group

Default cqh group
CQ Handler (CQH) Group: Concept

Interrupt vectors

HCA

Red cqh group  Blue cqh group  Default cqh group
CQH: Implementation

- **user**
  - `ibv_open_device`
  - `ibv_alloc_pd`
  - `ibv_create_cq`

- **kernel**
  - `Open IB device`
  - `Allocate PD`
  - `Create CQ`

...
CQH: Implementation

ibv_open_device

Open IB device

ibv_alloc_pd

Allocate PD

ibv_create_cq

Create CQ with CQH

cqh_name

Creates CQH and pass it to requestor

...+CQH

Create CQ with CQH
Experiment: Setup

Diagram showing the setup with Storage, SUT 1, SUT 2, and Clients connected by arrows indicating data flow.
Experiment: ib_send_lat

Baseline (ib_send_lat latency on idle systems).
Experiment: `ib_send_lat`

Latency numbers are normalized to the baseline
Experiment: ib_send_lat result

- No CPU Binding
Experiment: ib_send_lat result

- Assign exclusive CPUs to ib_send_lat

![Normalized Latency vs Message Size](chart)

- Bar chart showing normalized latency for different message sizes (16, 128, 1K, 8K, 64K) with and without CQH.
Experiment: OLTP result

Throughput Response Time

Without CQH With CQH

Throughput: 104.1%
Response Time: 95.1%
Conclusion

- CQ handler group is an efficient approach to guarantee QoS and fairness among different types of IO protocols running concurrently on the same IB device.
- Normalized latency improved by ~50% in OFUV u-benchmark when running with 100 QP (connection) SDP
- 4%~5% improvement seen in OLTP benchmark.
Thank You