A Concurrent Crash-Resilient Graph Data Structure for Non-Volatile Memory

Problem Statement
- Real-time analysis on large dynamically changing graph datasets is a hard problem
- Current practice is to prepare read-only snapshots for analysis.
- Problems with current practice:
  - Taking snapshots can take considerable time, which slows down analysts
  - Global lock on the graph store conflicts with updates, that need to be throttled
  - Recording and replaying large input streams infeasible
  - No support for crash resilience

Atlas
- Programming model for durability guarantees on persistent memory
- Persistency API
  - nvm_{alloc,free}()
- NVM_{Create,Delete,Find}Region()
- Consistency API
  - NVM_{BEGIN,END}_DURABLE()
- Runtime automatically identifies critical sections as Failure-Atomic SEctions (FASEs), provides all or nothing guarantee
- Transforms lock-based multi-threaded code to crash resilient code
- Open Source: https://github.com/HewlettPackard/Atlas

Our Approach
- Taking read-only snapshots dumps is a sensible approach but can be made faster
- Keep primary graph in persistent memory:
  - Allows efficient updates
  - Provide crash resilience with Atlas runtime for precious data, ie, recover to consistent state in case of failure
- Take periodic time-and-space efficient snapshot dumps in transient memory for analysis – Compressed Sparse Row (CSR) format
- Challenge: Obtaining efficient and consistent snapshot dumps concurrently without throttling updates to the persistent graph store

Preliminary Results
- Time for V=1M, E~7.6M inserts
- Same with periodic snapshots (1 per second)
- Comparison of wall clock time for just inputs vs inputs with periodic snapshots

Hardware and Software Configuration:
- Intel(R) Xeon(R) CPU X5650 @ 2.67GHz
- Ubuntu 16.04.1 Linux kernel 4.4