Adding Fault Tolerance to NPB Benchmarks Using ULFM

Zachary W. Parchman (TN Tech)

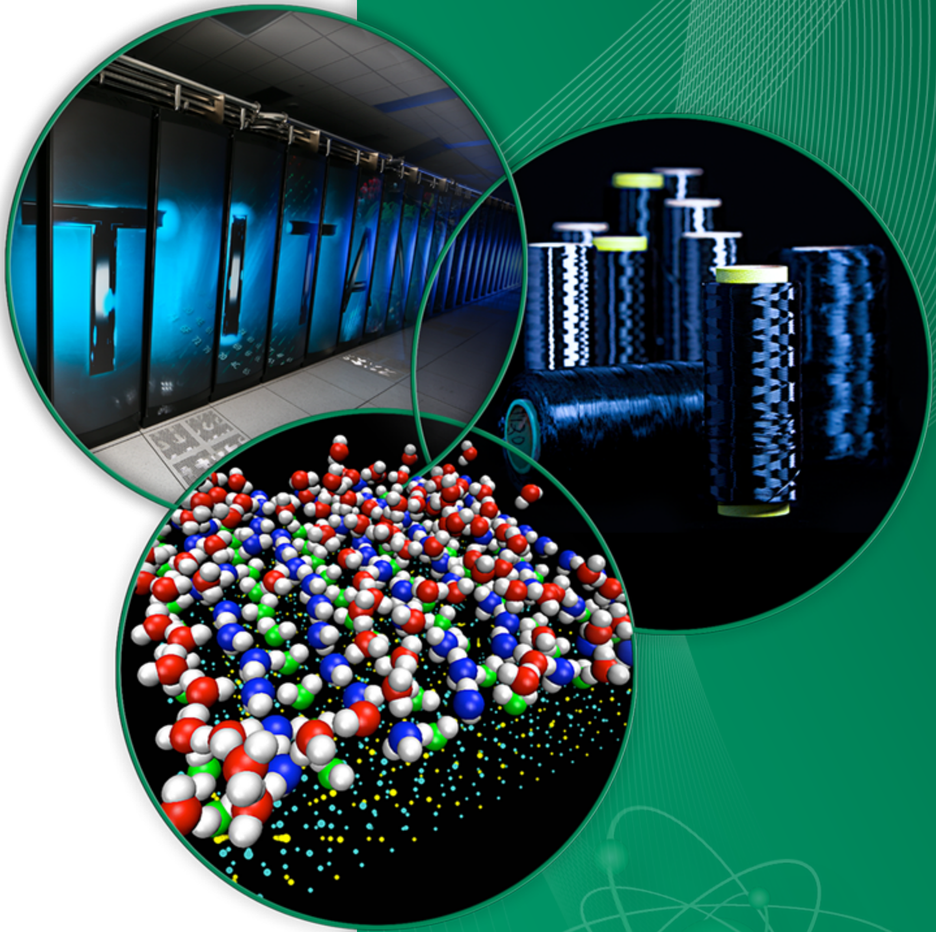
Geoffroy Vallee (ORNL)

Thomas Naughton (ORNL)

Christian Engelmann (ORNL)

David Bernholdt (ORNL)

Stephen L. Scott (TN Tech)



Motivation

- Fault tolerance (FT) and application resilience is becoming a primary concern for HPC
- Research community has been very active over the past decade
  - Checkpoint/restart
  - Redundancy
  - Application Based Fault Tolerance
- Standards start to consider FT extensions

The MPI forum fault tolerance working group is proposing the User-Level Failure Mitigation (ULFM)

Goals & Contributions

- This study does NOT aim at evaluating the performance of an ULFM implementation
- This study aims at
  - Investigating application-level strategies that leverage ULFM
  - Providing benchmarks that can illustrate the use of ULFM
- Other contributions
  - In-memory checkpoint/restart mechanism
    - Direct integration into the benchmarks to minimize dependencies
    - Avoids the cost of going through the file system layers
  - Fault injection tool based on the ULFM model

ULFM Overview

- Target failure model: Process failure
  - Explicitly handle fail-stop failures
  - Transient failures are masked as fail-stop
  - Failures are detected only in the context of MPI operations
  - NO support for silent and byzantine errors
- Target capabilities
  - Failure detection
  - Failure notification
  - Recovery of the MPI layer only

ULFM API - Overview

- Detection
  - Assumes perfect fault detectors
  - Via return codes of MPI communication functions
- Recovery of the MPI layer
  - MPI_Comm_shrink()
  - Creates a new communicator that includes only the surviving ranks
- Agreement
  - MPI_Comm_agree() and MPI_Comm_iagree()
  - All surviving ranks perform a collective consensus to agree on a value

In-memory Application Checkpointer

- Avoid the cost of going through the file system
- Implemented using MPI send/receive primitives
  - Simple
  - Potentially enables more opportunities to detect failures with ULFM
  - Checkpointed data is sent to one or more “neighbors”
- 2 strategies implemented
  - SINGLE_REDUNDANCY: the data is saved in a single neighbor’s memory
  - FULL_REDUNDANCY: the data is saved in all MPI ranks’ memory
  - Tradeoff between cost (extra communications and memory) vs. number of tolerated failures

Failure Injection Tool

- Based on ULFM model
  - Detection only in the context of MPI operations
  - Failures can be injected right before or after an MPI operation, other locations will not change the behavior
- Annotations of the application code
- Rely on a configuration file
  - Help reproducing experiments
  - Many injection points can be "activated"
  - When an activated injection point is reached, the MPI rank is terminated

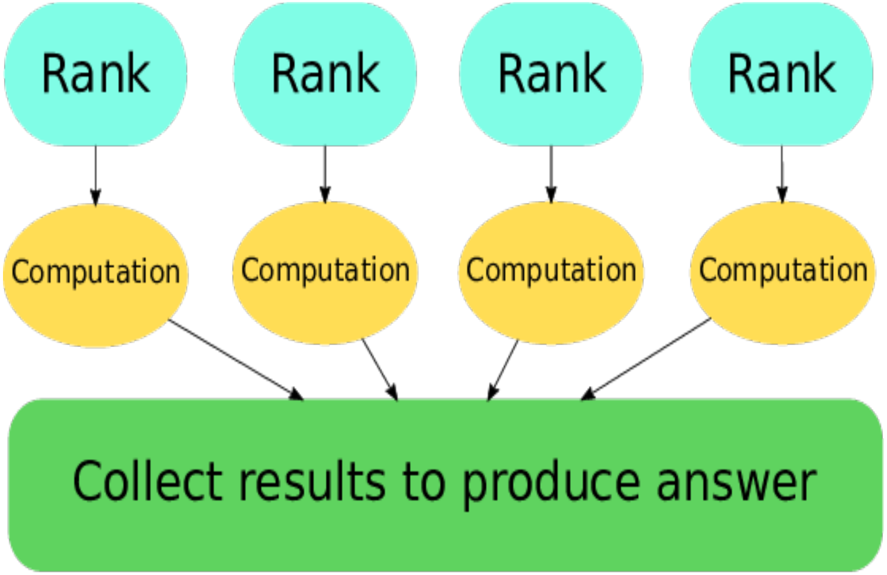
NPB Modified for ULFM Support

- 3 benchmarks have been extended
  - Embarrassingly Parallel – EP
  - Data Transfer – DT
  - Integer Sort – IS
- EP, DT and IS benchmarks have been modified
  - Only DT and EP are presented
  - Based on SINGLE_REDUNDANCY

Experimental Setup

- Up to 32 ranks, 1 rank per node to highlight the cost of in-memory checkpoint/restart via the network
- Nodes have two 12-core AMD Opteron 6164 HE, 64 GB RAM and bonded dual non-blocking 1 Gbps Ethernet
- Comparison of
  - No FT mechanism enabled
  - 0 failures: highlight the checkpoint/restart cost
  - 1 and 2 failures: highlight the overhead and behavior of the FT strategy
- Performed 10 runs per experiment

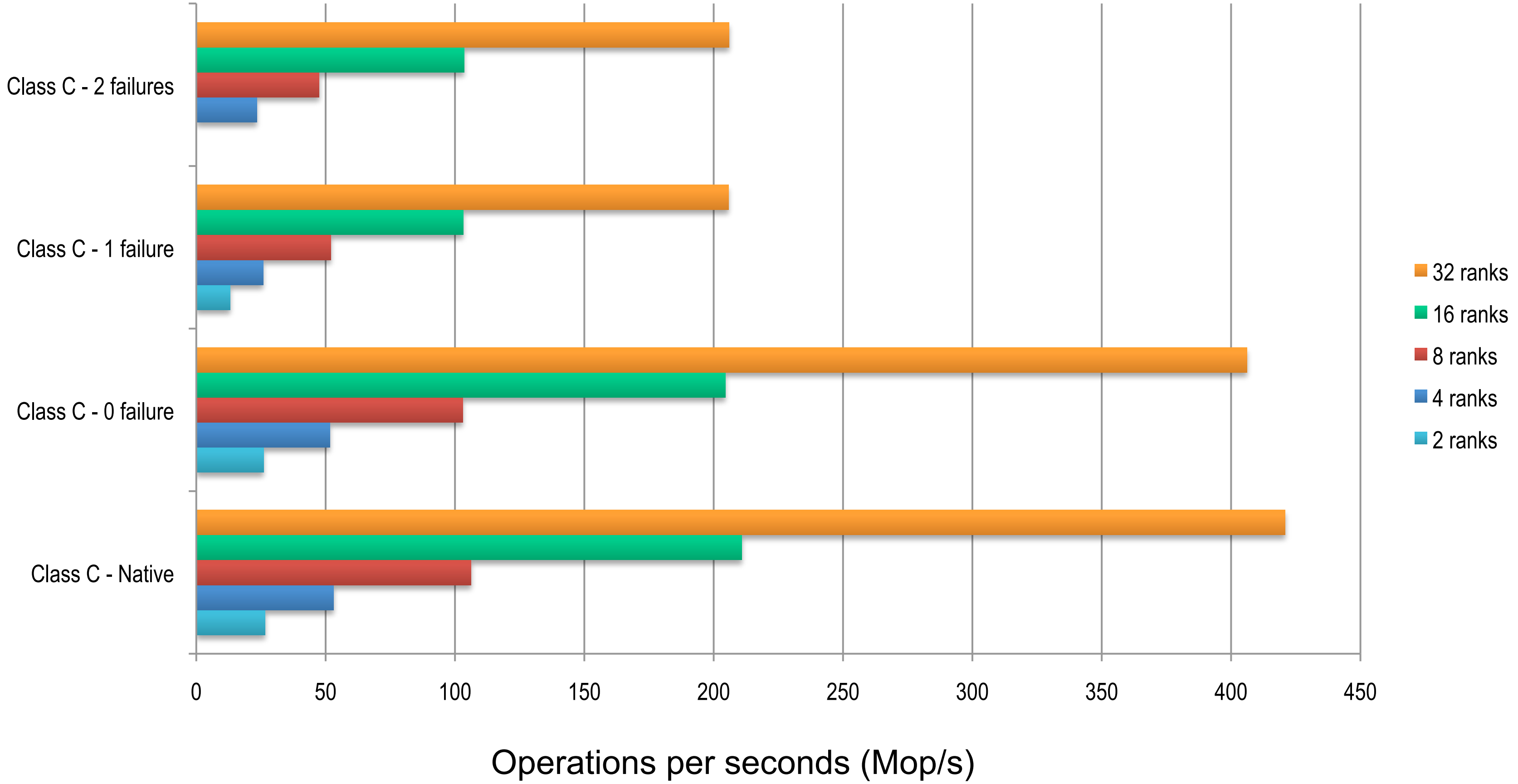
Extension of the EP Benchmark



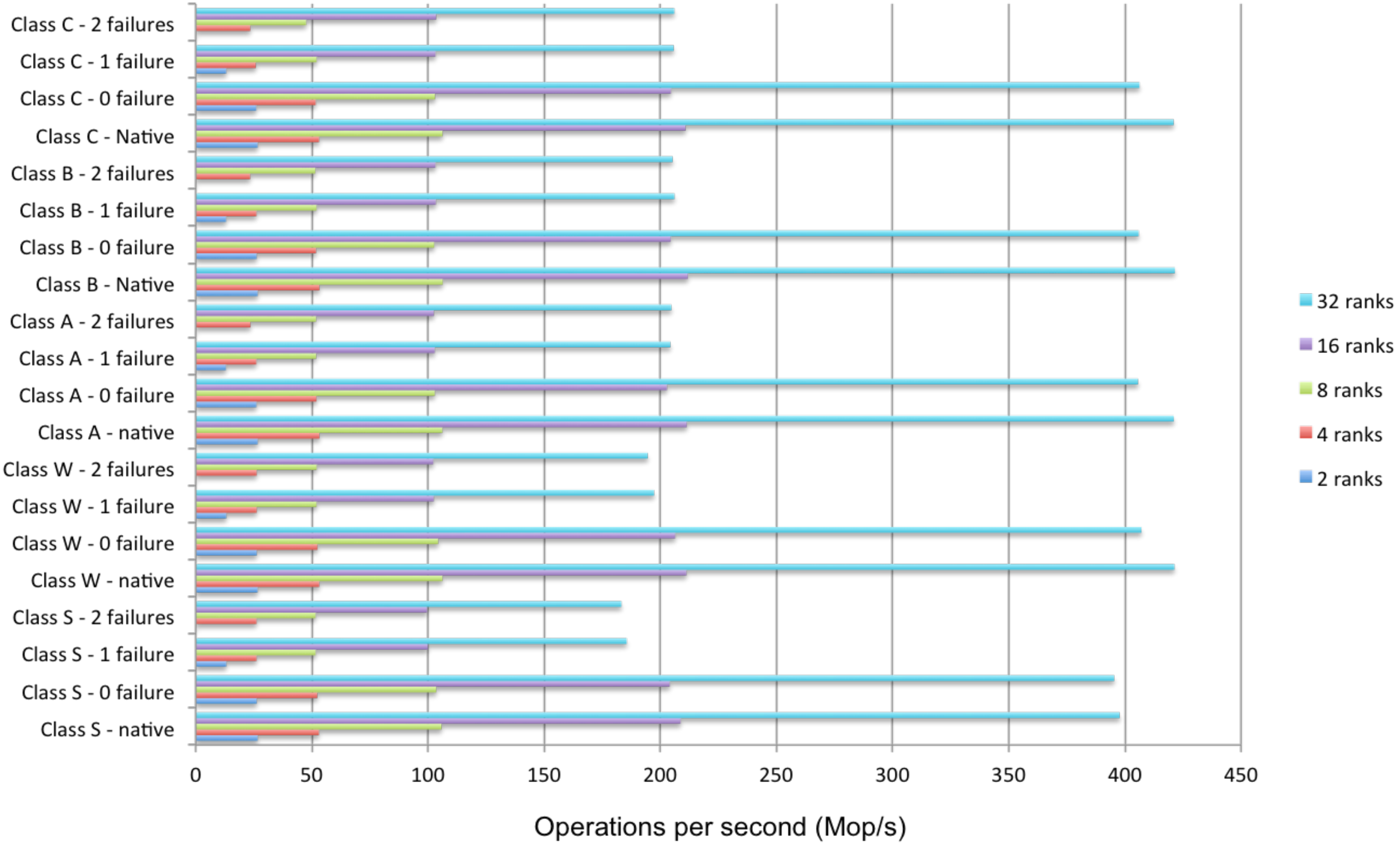
- Communication pattern
  - Barrier during initialization
  - Allreduce to collect the results
- FT Modifications
  - Detect failure during the allreduce communication phase
  - Redistribute the work that was lost because of the failure, after surviving ranks complete their work
- Experiments
  - Inject failures before executing the allreduce operation

EP – Results (1)

- Results for Class C



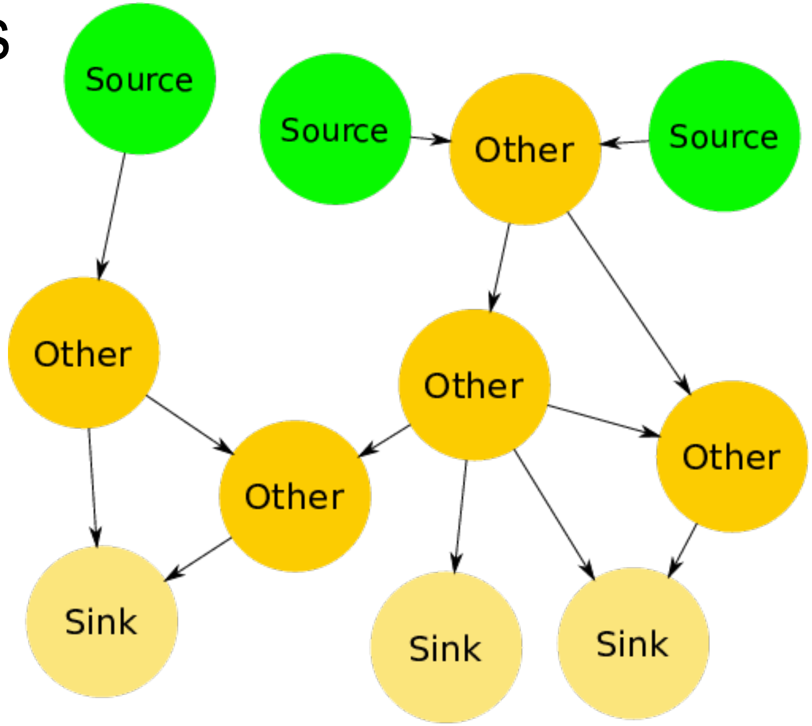
EP – Results (2)



Class C - 2 failures
Class C - 1 failure
Class C - 0 failure
Class C - Native
Class B - 2 failures
Class B - 1 failure
Class B - 0 failure
Class B - Native
Class A - 2 failures
Class A - 1 failure
Class A - 0 failure
Class A - native
Class W - 2 failures
Class W - 1 failure
Class W - 0 failure
Class W - native
Class S - 2 failures
Class S - 1 failure
Class S - 0 failure
Class S - native
0
50
100
150
200
250
300
350
400
450
Operations per second (Mop/s)
32 ranks
16 ranks
8 ranks
4 ranks
2 ranks

Extension of the DT Benchmark

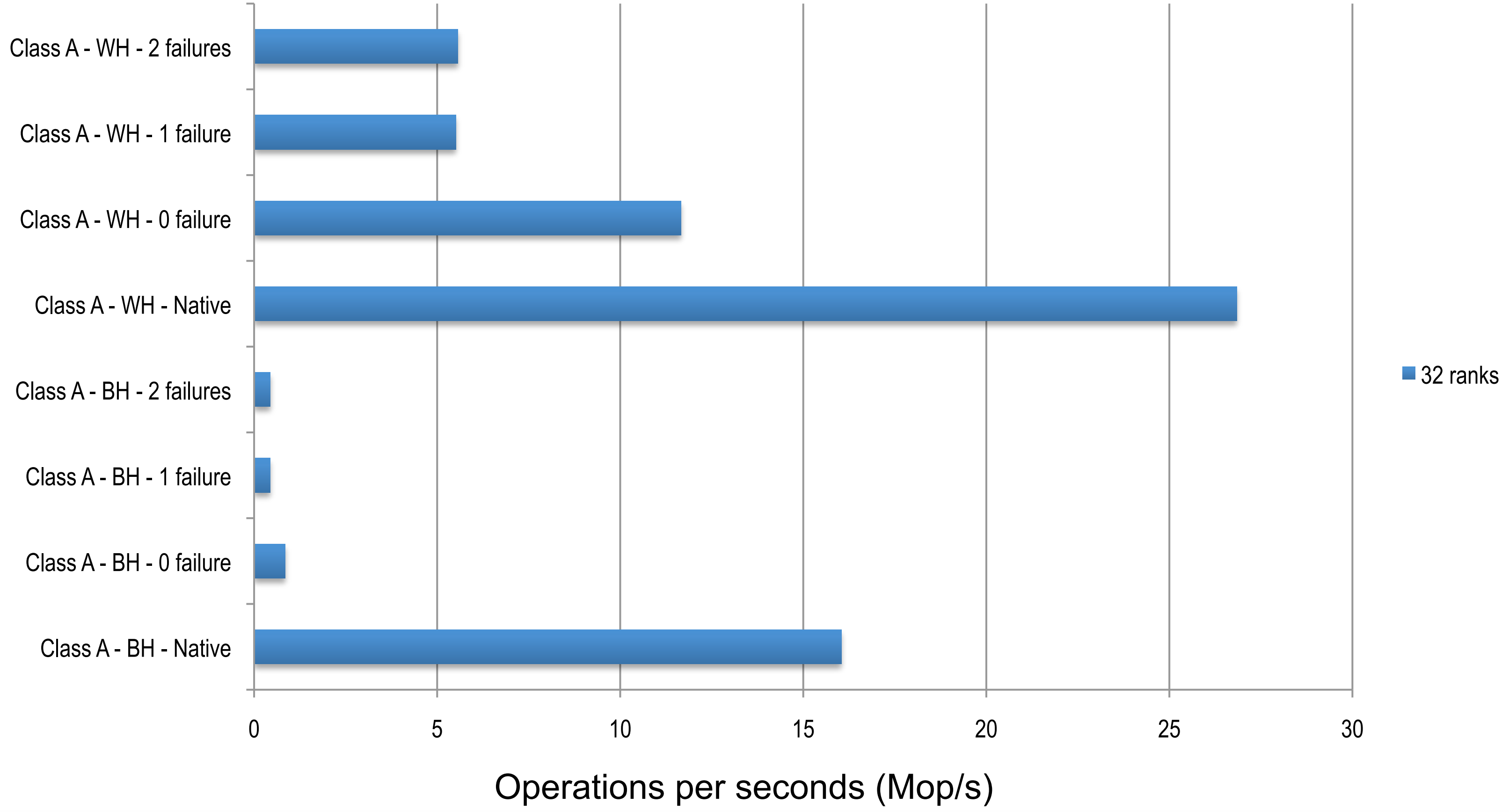
- Communication/computation patterns
  - Irregular communications
  - One-to-one mapping task/rank
  - All ranks have the same vision of the initial graph
  - At the end, rank0 collects all the results



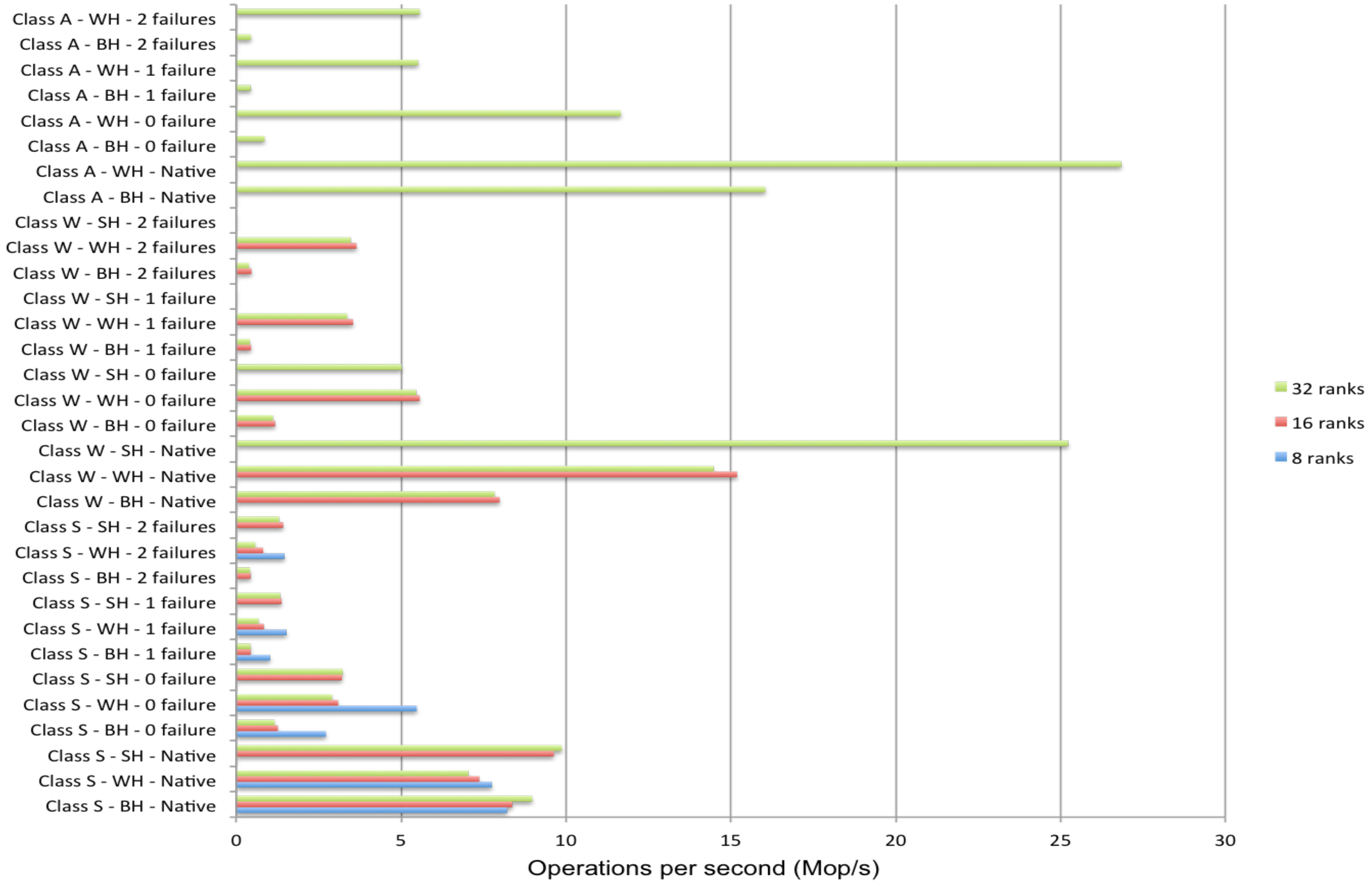
- FT Modifications
  - Serialize the graph for checkpointing
  - If a rank fails, a spare rank is "activated" and executes the task
- Experiments
  - Inject a failure before the result can be retrieved by rank0

DT – Results (1)

- Results for Class A (both BH and WH graphs)



DT – Results (2)



Class A - WH - 2 failures
Class A - BH - 2 failures
Class A - WH - 1 failure
Class A - BH - 1 failure
Class A - WH - 0 failure
Class A - BH - 0 failure
Class A - WH - Native
Class A - BH - Native
Class W - SH - 2 failures
Class W - WH - 2 failures
Class W - BH - 2 failures
Class W - SH - 1 failure
Class W - WH - 1 failure
Class W - BH - 1 failure
Class W - SH - 0 failure
Class W - WH - 0 failure
Class W - BH - 0 failure
Class W - SH - Native
Class W - WH - Native
Class W - BH - Native
Class S - SH - 2 failures
Class S - WH - 2 failures
Class S - BH - 2 failures
Class S - SH - 1 failure
Class S - WH - 1 failure
Class S - BH - 1 failure
Class S - SH - 0 failure
Class S - WH - 0 failure
Class S - BH - 0 failure
Class S - SH - Native
Class S - WH - Native
Class S - BH - Native
0
5
10
15
20
25
30
Operations per second (Mop/s)
32 ranks
16 ranks
8 ranks

Conclusion

- No one-size-fits-all solution
  - ULFM only provides basic primitives and capabilities: detection, notification and MPI runtime recovery
  - Best strategy really depends on the application (communication patterns and checkpoint size)
  - Understanding the various overheads is difficult
  - Witnessed behavior basically matches what we expected (based on work redistribution)
- Good illustration of the ULFM potential
- Work in progress
  - Detailed profiling
  - Provide additional strategies, including re-init
  - Integration of SCR