Fenix: Online Failure Recovery on top of ULFM

Marc Gamell¹, Keita Teranishi², Manish Parashar¹

¹ Rutgers Discovery Informatics Institute (RDI²), Rutgers University
² Sandia National Laboratories
Introduction

- Exascale next frontier for HPC tremendous opportunities new challenges
- Abstraction of a failure-free machine not sustainable at extreme scales
- Increasing number of components implies more failures
  - Petascale MTBF hours
  - Exascale MTBF minutes!
- Handling process failures and derivatives (node, blade, cabinet…) is critical
- Current approach Coordinated PFS-based Checkpoints On failure, stop application and Restart
  - Infeasible at exascale!

Online failure recovery necessary
Application-aware resilience required
Key contributions

Approach

• On-line, local, semi-transparent recovery from process, node, blade and cabinet failures
• Targets MPI-based parallel applications
• Using application-specific double in-memory, implicitly coordinated, local checkpoints

Fenix

• Design and implementation of the approach
• Deployed on Titan Cray XK7 at ORNL

Implementation details

• Built on top of MPI-ULFM
• Tested up to
  – 8192 cores w/ failures
  – 250k cores w/o failures
• Provides C, C++ and Fortran interfaces

Experimental Evaluation

• S3D combustion numerical simulation
• Sustained performance with MTBF = 47 seconds
• Experiments inject real failures
Table of Contents

• Introduction
• Key Contributions
• **Motivating Use Case**
• Online Failure Recovery Methodology
• API
• Experimental Evaluation
• Conclusion
Motivating use case – S3D production runs

- 24-hour tests using Titan (125k cores)
- 9 process/node failures over 24 hours
- Failures are promoted to **job failures**, causing all 125k processes to exit
- Checkpoint (5.2 MB/core) has to be done to the PFS

**Checkpoint (per timestep)**
- Total cost: 55 s, 1.72%

**Restarting processes**
- Total cost: 470 s, 5.67%

**Loading checkpoint**
- Total cost: 44 s, 1.38%

**Rollback overhead**
- Total cost: 1654 s, 22.63%

**Total overhead**
- Total cost: 31.40%
Motivating use case – Summary of problems

- At exascale, $O(1)$ process/node failure per minute
- Therefore, we need to increase checkpoint frequency
- Current checkpoint cost, $O(1)$ minute
- Total recovery cost, $O(3000)$ s

Unfeasible
Motivating use case – Possible solution

- **Checkpoint frequency** has to be dramatically **increased**
  
  In-memory, application-specific, fine-grained, local, high-frequency checkpointing

- **Recovery** cost must be reduced
- Process failures cannot be promoted to job failures

Online recovery
Table of Contents

• Introduction
• Key Contributions
• Motivating Use Case

• **Online Failure Recovery Methodology**
• API
• Experimental Evaluation
• Conclusion
Online Failure Recovery

• Currently deployed MPI implementations do not address hard failures
  – Runtimes kill all processes

• Online Failure Recovery:
  – Repair environment on failure

• User Level Failure Mitigation (ULFM):
  MPI extension to incorporate Fault Tolerance

• Fenix: Provides semi-transparent fault tolerance to applications
Fenix – Recovery Stages

1. Failure detection

Based on
- ULFM return codes
- MPI profiling interface
- Comm revoke

2. Environment recovery

- Re-spawn or Spare process pool
- Repair world communicator
- Delay recreation of user comms
Fenix – Recovery Stages

1. Failure detection
Based on
- ULFM return codes
- MPI profiling interface
- Comm revoke

2. Environment recovery
- Re-spawn or Spare process
- Repair world communicator
- Delay recreation of user comms

3. Data recovery
- Detect consistent state
- Recover data for new ranks
- Rollback all data to consistent state
- Double in-memory checkpointing reduces data motion

4. Resume processes
Where?
- At the beginning
- After last successful checkpoint
Fenix – Recovery Stages

1. Failure detection
   Based on
   - ULFM return codes
   - MPI profiling interface
   - Comm revoke

2. Environment recovery
   - Re-spawn or Spare process
   - Repair world communicator
   - Delay recreation of user comms

3. Data recovery
   - Detect consistent state
   - Recover data for new ranks
   - Rollback all data to consistent state
   - Double in-memory checkpointing reduces data motion

4. Resume processes
   Where?
   - At the beginning
   - After last successful checkpoint
Table of Contents

• Introduction
• Key Contributions
• Motivating Use Case
• Online Failure Recovery Methodology
• API
• Experimental Evaluation
• Conclusion
Fenix – Basic interface

```c
void Fenix_Init ( Fenix_Status status,
                 MPI_Comm *user_world,
                 Fenix_Resume_mode rp,
                 Fenix_Checkpoint_mode dm);

void Fenix_Finalize ( ) ;

void Fenix_Checkpoint_register ( void *array_pointer,
                                  size_t size_element,
                                  size_t size_array,
                                  Fenix_Array fenix_array );

void Fenix_Checkpoint ( Fenix_Array fenix_array );

void Fenix_Comm_add ( MPI_Comm comm );
```
Fenix – Basic interface

```c
void Fenix_Init ( Fenix_Status status,
                  MPI_Comm *user_world,
                  Fenix_Resume_mode rp,
                  Fenix_Checkpoint_mode dm);

void Fenix_Finalize ( );

void Fenix_Checkpoint_register ( void *array_pointer,
                                 size_t size_element,
                                 size_t size_array,
                                 Fenix_Array fenix_array );

void Fenix_Checkpoint ( Fenix_Array fenix_array );

void Fenix_Comm_add ( MPI_Comm comm );
```
Table of Contents

- Introduction
- Motivating Use Case
- Online Failure Recovery Methodology
- API
- **Experimental Evaluation**
- Conclusion
Goal and Methodology

• Experiment:

  S3D execution mimicking a future extreme-scale scenario
  • Injecting failures every (MTBF):
    • 47 seconds
    • 94 seconds
    • 189 seconds
  • Checkpoint 8.58 MB/core

• Procedure:
  1. Evaluate checkpoint scalability
  2. Calculate and validate optimal checkpoint interval
  3. Evaluate recovery scalability
  4. Run experiment

• Evaluation on ORNL Titan - Cray XK7
1. Failure-free **checkpoint cost (data size)**

**Conclusions:**
- Scale linearly with data size increase
- Huge communication cost

![Graph showing neighbor-based checkpointing](image-url)
1. Failure-free **checkpoint cost (core count)**

Conclusions:
- **Good scalability**
- **Really small footprint: O(0.1s)**
2. Optimal **checkpoint rate**

- Calculated by Young formula:
  - $T_S = \text{checkpoint time} = 0.0748 \text{ s}$ (with 2197 cores)
  - $T_F = \text{system’ MTBF} = \{47, 94, 189\} \text{ s}$ (3 tests)
  - $T_C = \sqrt{2 \ T_S \ T_F} = \{2, 3, 4\} \text{ S3D iterations}$

- Empirically validating the 94-second MTBF optimal rate:
  - Inject only one failure at a specific wall time within the first 94 seconds of a test
  - Repeat using:
    - Different checkpoint rates
    - Different failure injection timestamp
  - Optimal rate must offer smaller overall rollback cost!
Average of all failure injection timestamps

2 to 5 iterations is the optimal checkpoint period, validating Young formula
3. Recovery overhead

- 2197 cores
- 256-core failure (i.e. 16 nodes)
- 16-core failure (i.e. 1 nodes)
4. Recovering from high-frequency failures

Conclusions:
• Online recovery allows the usage of in-memory checkpointing, $O(0.1s)$
• Efficient recovery from high frequency node failures, as exascale compels
4. Recovering from **high-frequency failures**
Conclusion

• Application-awareness can help resilience at extreme scale

• Fenix provides
  – Local, on-line recovery (reducing failure overhead)
  – In-memory, application-specific, **high frequency checkpointing**
  – **Simple API:** only 35 new, changed or rearranged lines in S3D code

• Deployed and empirically tested on Titan Cray XK7
  – **S3D+Fenix tolerate failure rates <1 min** with same overhead as coordinated CR with failure rates of ~2.5 hours
Thank you