Towards fine-tuning of multi-level checkpointing using machine learning: The case of VeloC

ST M2.1 (6) Numerical Methods and Resilience
11th JLESC Workshop, September 9th, 2020

Tonmoy Dey†, Kento Sato†2, Jian Guo†2, Bogdan Nicolae†3,
Jens Domke†2, Weikuan Yu†, Franck Cappello†3, Kathryn Mohror†4

† Florida State University, USA
†2 RIKEN Center for Computational Science (R-CCS), Japan
†3 Argonne National Laboratory, USA
†4 Lawrence Livermore National Laboratory, USA
Challenges of Checkpointing at Exascale

Two major challenges:

- Performance and scalability
  - More failures at Exascale implies need to checkpoint more frequently
  - Less I/O bandwidth available per processing element
- Heterogeneity and complexity of storage hierarchy
  - Many options in addition to PFS: burst buffers, object stores, caching layers, etc.
  - Many vendors, each with its own API


Multi-Level Checkpointing

- Leverages heterogeneous storage hierarchy to write a hierarchy of checkpoints
- Key idea: use slow storage only when recovery from fast storage is not possible
- Consequence: Less I/O bottlenecks because PFS will be used infrequently
VeloC: Very Low Overhead Checkpointing

- High Performance and Scalability
- Hides complexity of interaction with deep storage stacks
- Configurable MLC (multi-level checkpointing)
  - L1: Local write
  - L2: Partner replication, XOR encoding, RS encoding
  - L3: Optimized transfer to external storage
- Configurable mode of operation:
  - Synchronous mode: resilience engine runs in application process
  - Asynchronous mode: resilience engine in separate backend process (VeloC does not die if app dies due to software failures)
- Easily extensible:
  - Custom modules can be added for additional post-processing in the engine (e.g. compression)

Web: https://veloc.readthedocs.io
Many configurations in C/R libraries

- Checkpoint location
  - Capacity v.s. Performance v.s. Reliability

- Checkpoint interval
  - Each level of checkpoint interval in multi-level checkpointing

- Erasure encoding
  - What erasure encoding should be used?
  - Group size (or Failure group size)

- Many others ...

Given an execution environment, finding “good” configurations is challenging as C/R scheme becomes more complicated
Finding good interval for efficient checkpointing

- **Tradeoff**
  - Frequent checkpoint: Unnecessarily spend more I/O time for checkpointing
  - Infrequent checkpoint: You may lose much more useful computation on a failure

- Even if you use state-of-the-art C/R techniques, poorly determined checkpoint intervals make system resilience worse than simple C/R

⇒ Finding optimal checkpoint interval is important for efficient C/R
Modeling for optimal checkpointing

<table>
<thead>
<tr>
<th>Checkpointing model</th>
<th>Formulation (Efficiency)</th>
<th>Analytical solution (Optimal interval)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single level checkpointing</td>
<td>$T = \frac{1}{\lambda} e^{\lambda (\lambda - C + R)} (e^{\lambda (T + C)} - 1)$</td>
<td>$\sqrt{2 \times C / \lambda}$</td>
</tr>
<tr>
<td>Mutil-level Checkpointing (SCR)</td>
<td></td>
<td>Complicated C/R models have finding analytical solution harder</td>
</tr>
<tr>
<td>Mutil-level Checkpointing (FTI)</td>
<td></td>
<td>Numerical solution</td>
</tr>
</tbody>
</table>

We tried to model to evaluate resiliency of more complicated erasure encodings. We found that it is significantly difficult to formulate C/R models unless we simply the model and/or make strong assumption.
We revisit simulation approaches instead of modeling approaches

Pros
— Simulation can be applied to more complicated
— Simulation can estimate expected execution time much more accurately than modeling approach

Cons
— Simulation takes time to explore different C/R parameters and find optimal checkpoint interval

While simulation is useful when evaluating efficient of C/R
If one wants to know the optimal checkpoint interval when submitting a job,
Simulation is not practical approach
AI for C/R

- Combine simulation with AI techniques
- Training data from simulation
  - Generate training data consisting input/output data by running simulator in many different scenarios
    - Checkpoint and recovery time, failure rates, type of erasure encodings (partner, XOR, RS etc.), node allocation, network topology (fat tree, torus etc.)
- Training
  - Train and Build a C/R NN model to find good configurations (e.g., checkpoint location, checkpoint intervals and a type of erasure coding)
- Why AI for C/R?
  - We do not need the best configuration as long as the configurations are relatively good. (We do not need 100% accuracy to find the best configuration)
  - Many other “good configurations” giving comparable efficiency to “the best configuration”
Evaluation

- Neural network
  - We used a simple 3-layer FCNN (fully connected neural network)
  - Hidden layer has 25 nodes

- Evaluations
  - FCNN give relatively same interval that simulator gives
  - FCNN give more accurate optimal interval compared to random forest and LightGBM

**Optimized Neural Network**

- Mean Absolute Error: 40.7 seconds

**Random Forest**

- Mean Absolute Error: 49.05 seconds

**LightGBM**

- Mean Absolute Error: 49.5 seconds
Conclusion, Future work and collaboration

- Conclusion
  - Finding “good” configurations is challenging as C/R scheme becomes more complicated
  - Even a simple FCNN give as good intervals as the simulator does
  - FCNN gives more close intervals to the simulator than random forest and lightGBM

- Future work & collaboration
  - We only targeted checkpoint intervals
  - We would like to incorporate other configurations to the simulator and training deeper NN for the simulator