TensorFI: A Configurable Fault Injector for TensorFlow Applications

Guanpeng (Justin) Li, UBC
Karthik Pattabiraman, UBC
Nathan DeBardeleben, LANL
Motivation

• Machine learning taking computing by storm
  – Many frameworks developed for ML algorithms
  – Lots of open data sets and standard architectures

• ML applications used in safety-critical systems
Error Consequences

Example: Self Driving Cars

Single bit-flip fault $\rightarrow$ Misclassification of image (by DNNs)

Our Focus: TensorFlow (TF)

• Open-source ML framework from Google
  – Extensive support for many ML algorithms
  – Optimized for execution on CPUs, GPUs, etc.
  – Many other frameworks target TF
  – Significant user-base (> 1500 Github repos)
What is TF?

- **TensorFlow (TF)** - framework for executing dataflow graphs
  - ML algorithms expressed as dataflow graphs
  - Can be executed on different platforms
  - Nodes can implement different algorithms
Goals

• Build a fault injector for injecting both hardware and software faults into the TF graph
  – High-level representation of the faults
  – Fault modeled as operator output perturbation

• Design goals
  – Portability – no dependence on TF internals
  – Minimal impact on execution speed of TF
  – Ease of use, compatibility with other frameworks
Challenges

• TF is basically a Python wrapper on C++ code
  – C++ code is highly system and platform specific
  – Wrapped under many layers – hard to understand

• Python interface offers limited control
  – Cannot modify operators “in place” in the graph
  – Cannot modify graph inputs and outputs at runtime
  – No easy way to intercept a graph once it starts executing (a lot of the “magic” happens in C++ code)
Approach: TensorFI

• Fault injector for TensorFlow applications

• Operates in 2 phases:
  – **Instrumentation phase**: Modifies TF graph to insert fault injection nodes into it
  – **Execution phase**: Calls the fault injection graph at runtime to emulate TF operators and inject faults
TensorFI: Instrumentation Phase

- **Idea:** Makes a copy of the TF graph and inserts nodes for performing the fault injection.
TensorFl: Execution Phase

- **Idea**: Emulate the operation of the original TF operators in the fault injection nodes
  - Inject faults into the output of operators

![Diagram showing the execution phase with fault injection nodes and operations involving placeholders and constants.](image)
TensorFI: Post-Processing

• Inject faults one at a time during each run
  – Log files to record the specifics of each injection

• Gather statistics about the following:
  – Injections: Total number of injections
  – Incorrect: How many resulted in wrong values
  – Difference: Diff between correct and wrong value

• Need to specify application specific checks for determining difference with FI outcome
TensorFI: Usage Model

Instrument code

```python
# Add the fault injection code here to instrument the graph
fi = ti.TensorFI(sess, name = "Perceptron", logLevel = 50, disableInjections = True)

correctResult = sess.run(accuracy, feed_dict={X: mnist.test.images,
Y: mnist.test.labels})

print("Testing Accuracy:", correctResult)

diffFunc = lambda x: math.fabs(x - correctResult)

# Make the log files in TensorBoard
logs_path = "./logs"
logWriter = tf.summary.FileWriter( logs_path, sess.graph )

# Initialize the number of threads
numThreads = 5

# Now start performing fault injections, and collect statistics
myStats = []
for i in range(numThreads):
    myStats.append( ti.FIStat("Perceptron") )

# Launch the fault injections in parallel
fi1.launch( numberOfInjections = 100, numberOfProcesses = numThreads, computeDiff = diffFunc, collectStatsList = myStats)

# Collate the statistics and print them
print( ti.collateStats(myStats).getStats() )
```
TensorFI: Config File

```yaml
# This is a sample YAML file for fault injection configuration
# The fields here should correspond to the Fields in fiConfig.py

# Deterministic fault seed for the injections
# Seed: 1000

# Type of fault to be injected for Scalars and Tensors
# Allowed values are {None, Rand, Zero}

ScalarFaultType: Rand
TensorFaultType: Rand

# Add the list of Operations and their probabilities here
# Each entry must be in a separate line ad start with a '-'
# each line must represent an OP and it's probability value
# See fiConfig.py for a full list of allowed OP values
# NOTE: These should not be any tabs anywhere below

Ops:
# - ALL = 1.0 # Chooses all operations
  - ADD = 1.0
# - DIV = 0.0 # This does not exist - and should be ignored (Test)
# - SUB = -0.5 # This should raise an exception

# How many times the set of above operations should be skipped before injection
# SkipCount: 1
```
Example Output: AutoEncoder

Original image, no faults
Fault injection prob. = 0.1
Fault injection prob. = 0.5
Fault injection prob. = 0.7
Fault injection prob. = 1.0
Reconstructed image (no faults)
TensorFI: Open Source (MIT license)

https://github.com/DependableSystemsLab/TensorFI
Benchmarks

• 6 open source datasets
  – UCI open source ML dataset repository
  – Can be modeled as classification problems

• 3 ML algorithms
  – k nearest neighbor (kNN)
  – Neural network (2-layer ANN)
  – Linear regression
Experimental Setup

• **Fault injection configurations**
  – Repeat 100 FI campaigns per benchmark (One fault per run)
  – FI rates (prob. of injection): 5%, 10%, 15% and 20%

• **Metric: Average accuracy drop**
  – Original accuracy without fault injection (OA)
  – Accuracy after fault injection (FA)
  – Average accuracy drop = average of (OA-FA) among all FI runs
Results

- SDC rate increases are different as fault injection rates increase
- SDC rates are different for different models
- kNN has lower SDC rates and lower rate of increase
Future Work

• Investigate the error resilience of different ML algorithms under faults
  – Understand reasons for difference in resilience
  – Build a mathematical model of resilience
  – Choose algorithms for optimal resilience

• Understand how different hyper-parameters affect resilience and choose for optimality
TensorFI: Summary

• Built a configurable fault injector for injecting both h/w and s/w faults into the TF graph
  – High-level representation of the faults

• Design goals
  – Portability – no dependence on TF internals
  – Speed of execution not affected under no faults
  – Ease of use, compatibility with other frameworks

Available at: https://github.com/DependableSystemsLab/TensorFI

Questions ? karthikp@ece.ubc.ca