CSL862 Major Exam
Advanced Topics in Operating Systems
Sem 1 2017-18
21 November 2017

Answer all 5 questions Max. Marks: 52

1. Efficient Network Reachability Analysis Using a Succinct Control Plane Representation

a. Routing protocols may involve several intermediate configurations which may be erroneous (e.g., node A may not be reachable from node B). Does ERA identify erroneous intermediate states? Or is it able to identify only whether the configuration in the steady-state may be erroneous? Briefly explain your answer. [3]

b. In ERA, a route is encoded using a 128-bit vector. The paper discusses ERA's working using a 4-bit encoding of a route x3 x2 x1 x0 (see Section 5.2, an illustrative example). The 4-bit encoding helps simplify the discussion. Later in the same section, the paper says that Vout = <some predicate over x0, x1, x2, x3>. What are Vin and Vout? What does the predicate over x0, x1, … mean, and how does it help ERA's analysis? [4]

2. EbbRT

a. What is a library operating system? How is it different from a general-purpose operating system? What are its advantages over a general-purpose OS? [3]

b. Why is a library OS more relevant today than it was in 1980s-90s? [1]

c. Consider the event handling loop discussed in Section 4.2. The authors explain how the event handling framework can allow a network driver to implement adaptive polling. What is the need for adaptive polling in a network card? [1] What is the point that the authors are trying to make through this discussion? [1]

d. “The fast-path cost of an Ebb invocation is one predictable conditional branch and one unconditional branch more than a normal C++ object dereference.”

Explain this statement. Why is the fast-path cost of an Ebb invocation similar to the cost of a normal C++ object dereference? [1.5] What is the likely reason for an additional conditional branch and an additional unconditional branch? [1.5]

e. “Because event-driven programming splits one logical flow of control across multiple stacks, exceptions must be handled at every event boundary. This puts the burden on the developer to catch exceptions at additional points in the code and either handle them or forward them to an error handling callback.”

Briefly explain this statement [3]. How do monadic futures address this problem [2].

3. Tensorflow

a. How is TensorFlow different from previous work on parameter servers? Briefly describe two major differences. [3]

b. Why are batch dataflow systems (such as MapReduce and DryadLINQ) unsuitable for machine learning applications? In other words, how is TensorFlow more suited to such applications? Explain briefly. [2]

c. How does TensorFlow currently support the execution of the same operation on multiple types of devices, e.g., CPU, GPU, TPU? [1]

d. Explain the following program, by explaining each block of statements (1, 2, and 3) briefly. [4]

```
# 1. Construct a graph representing the model.
x = tf.placeholder(tf.float32, [BATCH_SIZE, 784])  # Placeholder for input.
y = tf.placeholder(tf.float32, [BATCH_SIZE, 10])   # Placeholder for labels.

W_1 = tf.Variable(tf.random_uniform([784, 100]))   # 784x100 weight matrix.
b_1 = tf.Variable(tf.zeros([100]))                 # 100-element bias vector.
layer_1 = tf.nn.relu(tf.matmul(x, W_1) + b_2)      # Output of hidden layer.

W_2 = tf.Variable(tf.random_uniform([100, 10]))    # 100x10 weight matrix.
b_2 = tf.Variable(tf.zeros([10]))                  # 10-element bias vector.
layer_2 = tf.matmul(layer_1, W_2) + b_2            # Output of linear layer.

# 2. Add nodes that represent the optimization algorithm.
loss = tf.nn.softmax_cross_entropy_with_logits(layer_2, y)
train_op = tf.train.AdagradOptimizer(0.01).minimize(loss)

# 3. Execute the graph on batches of input data.
with tf.Session() as sess:                         # Connect to the TF runtime.
  sess.run(tf.initialize_all_variables())          # Randomly initialize weights.
  for step in range(NUM_STEPS):                    # Train iteratively for NUM_STEPS.
    x_data, y_data = ...                           # Load one batch of input data.
    sess.run(train_op, {x: x_data, y: y_data})     # Perform one training step.
```

Figure 1: An image classifier written using TensorFlow's Python API. This program is a simple solution to the MNIST digit classification problem [48], with 784-pixel images and 10 output classes.

4. SCONE: Secure containers with Intel SGX

   a. Discuss a use-case of Intel SGX to explain why it is useful? [3]

b. List one advantage of containers over processes? [1] List one advantage of containers over VMs? [1]

c. Explain how user-level thread in combination with asynchronous system calls helps SCONE design, starting with what problem they solve and then how they solve it. [3]

5. Netbricks

a. What is network function virtualization, and why is it relevant? [2]

b. What is the key idea in the Netbricks paper? [1] What are the primary challenges with this approach? [2]

c. For packet processing, each packet is represented as (i) a stack of headers; (ii) the payload; and (iii) a reference to any per-packet metadata.

Explain each of these. Why stack? [3]

d. Explain the ‘GroupBy’ operator. [1]

e. Why does NetBricks perform so much better (e.g., 7x faster than when containers are connected using SoftNIC) than using one container per network-function? [2]

f. How does NetBricks ensure isolation across two different network functions? [2]