SWEN90010: High Integrity Systems Engineering
Department of Computing and Information Systems
The University of Melbourne

Workshop 11

Introduction

In lectures we discussed different voting algorithms for dealing with faults in high-integrity systems. In this workshop, we'll explore measurement of systems, some fundamental statistical concepts for measurement, and measure the effectiveness of median voting as a technique to fault tolerance.

Some fundamental statistics

When measuring the reliability of something; e.g. a hardware sensor or a software component; it is typical to run multiple trials of that thing and to record what we see over those trials. For example, to measure the reliability of a type of hard drive before 6 months, we can run 1000 of those hard drives for 6 months, and record how many failed before that time.

In this workshop, we look at measurement of components that return a value within a range. The reliability of that component is how consistent it is in returning the correct value. For example, to measure the reliability of a temperature sensor, we place the sensor in an area with a set temperature, such as 60C, and take a series of readings. We then look at some statistics to measure how reliable the sensor is.

This process is taking a *sample* of readings, which are then used to estimate the reliability all sensors, or the reliability of the *real population*.

Three fundamental measures are:

1. the *mean* (or *average*) of a sample;
2. the *standard deviation*, which is a measurement of variability or diversity of a sample; and
3. the *confidence interval* of the mean, which is a range that we are *confident*, to a certain level, the real mean of the population falls within.

More formally, if we have N samples $x_1, x_2, ..., x_N$, then the mean, μ, and standard deviation, σ, are:

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i, \qquad \sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

A confidence interval must be stated from a particular confidence value; e.g. 95%. A 95% confidence interval therefore says that, given the samples we have taken, we are 95% confident that the mean of the real population falls within this interval.

Formally, the 95% confidence interval is calculated as:

$$[\mu - 1.96\frac{\sigma}{\sqrt{N}},\ \ \mu + 1.96\frac{\sigma}{\sqrt{N}}],$$

in which 1.96 is derived from the fact that 95% of the area under a normal distribution curve falls with 1.96 standard deviations. If we wanted to computer the 99% confidence interval, we would use 2.58 instead of 1.96.

Your tasks

1. In the source code accompanying the workshop, there are several Ada packages/files of note:

   `sensor.adb/s`: contains a simulation of a sensor that reads values around 60.0, with a standard deviation of 1.0 (uses `randomfloat.adb/s` to generate random values). However, it can also fail, at which point it starts return values with a uniform distribution between 40.0 and 80.0.

   `precisestats.adb/s`: calculates mean, standard deviation, and confidence intervals for a sample of data.

   `testsensor.adb`: a procedure that runs a series of samples of the sensor and uses `precisestats.adb/s` to calculate the mean, standard deviation, and confidence intervals from this sample.

2. Compile `testsensor.adb` and run this. The output shows the mean, standard deviation, and confidence interval of a series of 10 measures of the sensor simulation (where 10 is set by the variable `Max`).

   Record the mean, standard deviation, and confidence interval for this (used for comparison later).

3. In `testsensor.adb`, the variable `Max` is set at 10. Increase this to 100, 1000, and 10,000, re-compiling and re-running the program each time. What happens to the output?

   Record the mean, standard deviation, and confidence interval for these.

4. The next task asks you to create an *array* of sensors, and to use median voting to get the output from this array.

   For, declare an array of sensors using:

   ```
   type SensorArray is array(1..NUM_SENSORS) of Sensor.Sensor;
   Sensors :  SensorArray;
   ```

   Take the readings from this array and store them into an array of type `Sensor.ReadingArray` (defined in `sensor.ads`). This is declared as:

   ```
   Readings :  Sensor.ReadingArray(1..NUM_SENSORS);
   ```

   where `NUM_SENSORS` is a constant ≥ 1.

   Next, implement a median voter on the array of sensors, to get a single sensor reading, and assign this value to the variable `SensorReading`.

   There is a procedure `Sort` in the `Sensor` package, which sorts a `Sensor.ReadingArray` into non-decreasing order. You can use this to implement the median voter.

5. Set `Max` back to 10, and with 3 sensors, print the statistics for `Max` readings of the voted output (that is, the median value). Print the statistics for the voted output of the three sensors. What difference to you see with these values compared to just using a single sensor?

6. Increase the number of sensors to 10 and 1000, re-compiling and re-running the program each time. What happens to the output? What do you think this means with respect to the median voting algorithm?