Table of Contents

Tests with a simple topology

After the tests with a super-simple model we created a small 6-node topology to focus on the definition of the base elements of a Network.

Executive summary

We added new Network concepts to PowerDEVS: Routes, Flows, Links, Servers and Links.

Conclusions from the comparison tests are similar as with the super-simple model:

- Using exp distributions PowerDEVS, JMT, and MVA yield simular results
- Using other distributions there is a moderate diference
- Regarding performance, we made some improvements to PowerDEVS but it is still 2 times slower than JMT.

Simple Topology

Below is the 6-node topology to be used: 2 source servers, 2 destinations, 2 routers, 5 links.
The topology was taken from the On integrating fluid models with packet simulation paper, although in the paper the scenario is completly different targeted to TCP flows, so resuls are not comparable.

In PowerDEVS

NOTE: the NICQueueSampler inside the Links are not part of the model anymore. See performance improvements.

In JMT

New Network Elements

We created atomic models and c++ clases to represent the fundamental elements in a network:
Models:

- **Server:** it generates new packets based on the flows assigned to it
- **Link:** it is a coupled model (maybe in the future would be good to make it a single atomic model to improve performance).
  It applies a delay to the packets based on their size. While a packet is being transmited, other arriving packets are queued. This queue represents the NIC buffer where the link is connected to. It might be strage to think that the queue is inside the Link model...
- **Router:** it receives packets from an inPort and forwards them to an outPort base on the packet's route
- **Sink:** receives packets and logs some measurements (ej: residenceTime, received_bits, etc).

With this models simple topologies can be defined using the GUI. For model complex topologies, the first step is to be able easily instanciate these models in c++ code (some ideas here). In a later future a high-level language to define a topology can generate that c++ code.

Classes:

- **Route:** represents a path that packets take in order to go from a source node (eJ: a Server) to a destination node (ej: aPacketSink), traversing other network nodes (Ej: Routers).
- **Flow:** Flows (or classes in the the MVA parlance) define the communication between different nodes. Flows define the rate at which packets are generated, the size of the packets, and the route these packets will take.
- **FlowDefinitions:** a static class where all the flows of the system are defined.

The Twiki here explains these classes, how to create instances and use them.

Now flows can be defined in c++ code. Easily if there are not too many flows. In a later future a high-level language to define a topology can generate that c++ code.

Model Verification

To verify the model we performed a small experiment setting up a bottleneck link. We visualized the link utilization and buffering.
The same experiment was perfomed in JMT which yield similar results.

Configuration

For this topology we defined 2 flows as follows: servers generating traffic at 35Mbps and 55Mbps . Felix1 talking to Dst1, and Felix2 talkig to Dst2. All links configured at 200Mbps, except the central link at 100Mbps (bottleneck).

```
void FlowDefinitions::defineFlows(){
FlowDefinitions::addFlow(0, std::make_shared<ExponentialDistributionParameter>((double)1/4375), //
4375Hz
std::make_shared<ExponentialDistributionParameter>(1000*8), // 1KB / 4375Hz = 35Mb/s
{ // Route
{0, "FelixServer1"},
{0, "Router1"},
{0, "Router2"},
});


std::shared_ptr<Flow> flow (new Flow(0,
std::make_shared<ExponentialDistributionParameter>((double)1/6875), // 6875Hz
std::make_shared<ExponentialDistributionParameter>(1000*8), // 1KB / 6875Hz = 55Mb/s
{ // Route
{0, "FelixServer2"},
{0, "Router1"},
{1, "Router2"},
}));
FlowDefinitions::addFlow(flow);
}

// Links
FelixNICQueue1.maxBuffer = -1; // buffer in the link's in NIC
```

```
Link2.capacity = 100 * M; // (bits/s)
FelixLink.capacity = 200 * M; //
DstLink.capacity = 200 * M; //
link.delay = 0;
```

Results

The 5 **links utilization** shows as expected: 35Mbps for the Felix1 and Dst1, 45Mbps for Felix2 and Dst2, and 90Mbps for the central link.

The **buffer in the NICs** show to be all almost empty, except for the bottle neck link which has maximum buffer usage of ~11Kb (also not much as we are in 90% link usage).
It can be seen that the Router2 buffers are less used than the Felix ones, because they are behind the bottle neck buffer. Felix2 buffer is more that Felix1 buffer because it sends more data.

The **latency** for each flow is also as expected: the flow from Felix1 has little less latency as it generates less

traffic, nevertheless the main cause of the latency is the central link which is shared by both flows.
It can be observed that the peaks in the latency match the peaks in the central buffer (comparing this plot with the previuos at for example 8, 18, 70 and 87 seconds).

Comparison (, JMT, JMT-MVA)

Comparison of predicted values

The interesting part of the topology is Link2 (the bottle neck). We study its buffer size (mean and max values), its usage , and the overall system responseTime (latency) which is an effect of the system's buffers.

We simulated 400s sweeping the Flow2 arrival rate from 1Hz to 8125HZ (the saturation point). In this case always using exponential distributions as it is the only one supported by MVA. Results with other distributions in the next section.

Below is the comparison of the predicted values for the 3 tools: PowerDEVS (discrete), JMT(discrete), JMT-MVA (MVA)

The 4 plots show expected results. As the rate increases the network gets used more (linear increase in plot 3) and shows the effects of saturation: buffer increases exponentially (plot 1 and 2) affecting the latency (plot 4).

Buffer max value

As it can be observed the 3 tools yield almost the same predicted values for all arrival rates. The main discrepancy is for the max value (plot 2), which is rather important as it will be the main focus of our studies in ATLAS. MVA does not provide an expected max value for the queues, so in the plot the mean value is used for MVA (same as in plot 1). It is strange to see that JMT predicts almost the same values for mean and max buffer sizes. On the other hand, PowerDEVS predicts bigger max values as expected.
An unverified theory is that the max value published by JMT is not really the maximum observed value, but instead that maxium of the sampled values (maxium of the average values). JMT samples several times during the experiment and saves the mean value for each sampled period. Maybe the max value is the maxium of this averages and not the overall maximum value as published by PowerDEVS.
For ATLAS we need the overall maximum.

Another observation is that the maximum value in PowerDEVS follows the same "shape" as the mean value (but scaled). This could be useful if we plan to use MVA as an hybrid option (because we need to study the maximum but MVA does not provide it).

Comparison of execution performances

We compare execution time performance of the different simulators.

JMT performs twice better than PowerDEVS.

Performance improvements in PD

Performance was improved compared to the measurements performed in Comparison of PowerDEVS, JMT and JMT-MVA (with a super simple model) as follows:

1. The QueueSamplers were removed and now using the SamplerLogger. This improves performances, as each link has one less atomic model.
   Farther improvement was achieved by changing the NetworkQueue as not to produce "notification" events, as they are not needed any more. Much less DEVS events are produced.
2. The C++ optimization flag was set to -O3 (before we were using -O1).
3. The simulation creates and destroys lot of small objects (Packets) very fast. An attempt of using Memory Pools for creating packets was performed, but did not improved performance.

Configuration:

We simulated 400s sweeping the Flow2 arrival rate from 1Hz to 8125HZ (the saturation point).
At the highest rate (8125HZ from flow2 + 4375Hz from flow1), 12.5K packets are genereted per simulated second. So for 400s the simulated generated and processed 5M packets.
JMT samples as explained here for 15 metrics PowerDEVS samples every 1s (so 400s samples in total) for 238 variables. For PowerDEVS the time includes initialization (which is taking 8s.. a LOT LOT) and scilab save (~2s).

Results

For PowerDEVS performance 4 different times are shown: **1)** Before the optiomizations ("Sampling rate 1s") **2)** with queue optimizations, removing the notification to samplers ("optimized&no sampler") **3)** with the changes from 2 plus the attempt of using memory pools ("optimized&withMemPool") **4)** same as 2, but without taking initialization and scilab times ("optimized, noInitNoScilab")

MVS time is obviously constant and very low.

PowerDEVS improved its performance a little bit thaks to removing the Sampler models and using the sampling directly at the logger. The attempt of using memory pools did not performed better.
PowerDEVS execution time can be approximated linearly by T(rate) = 0.00653 * rate + 9.4 (**PowerDEVS slope is 0.00652**).
or T(totalPackets)=0.000016*totalPackets +3.47 (totalPackets=400s*(flow1rate+flow2rate) (**PowerDEVS process each packet in 0.00016s**).

JMT performs better than any execution of PowerDEVS.
JMT execution can be approximated linearly by T(rate) = 0.00345 * rate +0.5. (**JMT slope is 0.00345**).

JMT performs twice better than PowerDEVS. Before the performance improvements it was 5 times better (see here).

Performance outlook for a real PhaseI simulation

Felix servers will have at least 40Gpbs links. Assuming an event size of 2M distributed in 64 felix servers, it is 31K in each felix server per event (this is what we simulate as a packet).
To have a 100% utilization of the link, we would need a rate of 161KHZ events in each felixServer. For 80% utilization the rate should be 129MHZ

Assuming a performance approximation as before (this is optimistic as the network will be more complex), to simulate a single felixserver for 1s would take:
T(100% utilization) = T(161K packets) = PowerDEVS -> 6s
= JMT ->

To simualte the 64 servers for 1 minute at 100% utilization:

PowerDEVS -> T(161K packets * 60s) * 64 servers = 158.03s*64 = 10113s = **2.8 hours**

Nevertheless, this approximation has only 5 links. The phaseI network would have O(100) links. The difference between the 1 link network and the 5-link network is quite bit, so other approximation should be more appropiate.

Test with other distributions

We tested changing the distributions used in order to evaluate if MVA could be used even if distributions are not exponential.

We compare the base scanrio (exponential distribution for both arrivalRate and packetSize) and evaluated 3 extra scenarios **1)** with normal distribution for packet size **2)** with constant distribution for packet size **3)** with normal distribution for arrival rate.

Conclusion

We observe a moderate difference using different distributions, always predicting less buffer utilization than using an exponential distirbutions. The higher the rate (closer to saturation), the bigger the difference.
Using normal distribution for arrivalRate does not differ much from using a normal distribution for packetSize.

Future Steps

TODOS:

- (DONE) Perform some tests to verify the model, and compare with JMT.
- (Semi-DONE) Think of a mechanism to read flow parameters from the configuration.
  We are reading each distribution parameters as usual from the config (nothing special for flows)
- Reading logging and debug setting from configuration is making initialization slow (8s). This is because it tries to read ~200 variables from Scilab. Need to think of a way to avoid this time consuming reading of variables (¿Is there a way to know all defined variables in Scilab from PowerDEVS?).

Possible modeling improvements:

- Evaluate to make the link (queue+server+sampler) a single atomic model to improve performance.
- Add a delay to the routers.
- It might be strage to think that the queue is inside the Link model...
- Add queueing policies to routers. This will require thinking the best way for the Router model to get the information of all queues (now inside the Link model) connected to it.
  This might be a problem because of putting the queue inside the link... maybe rethink..
- Now the sampling and logging can be set in configuration preatty easily. But with more models might become tricky. Develop "vectorial" configuration or something of the sort.
-

-- MatiasAlejandroBonaventura - 2016-06-15

This topic: Main > PhaseISimpleTopology
Topic revision: r9 - 2016-07-05 - MatiasAlejandroBonaventura