Sensitivity Analysis with LQNX/LQX

Murray Woodside, Carleton University, Mar 10, 2010

This note describes how to compute a sensitivity matrix, using the features recently added to the Layered Queueing Network Solver LQNS to do experiments.

The augmented LQNS is called LQNX, and has an experiment-driving program added optionally to each model using a language called LQX. LQNS models in XML are compatible with LQNX.

Sensitivity

A sensitivity matrix has entries which approximate the derivatives

Hij = d(PerfMeasure i)/d(ModelParameter j)

The features of LQNX that are used for sensitivity calculation are

- declaring a numerical parameter by a string name as $name (similar to models for the earlier experiment system SPEX)
- collecting the variable parameters in an array params[ ], such that each can be incremented in turn
- modifying the parameters and calling the solver, once for a baseline run and once for each incremented parameter
- calculating the derivative approximation for each of several output measures as

Hij = ((PerfMeasure i from model with ModelParameter j incremented)
- baseline value)/increment

- reading the baseline parameter values from a pipe or a file, and outputting H similarly.

Illustrative Example

The attached model illustrates the program required to do this for a small model of a web application. This model could be generalized in various ways, when LQX is finished, but what is here does the job and works at the date of writing.

Figure 7 shows an example LQN representing a web application. The bold rectangles represent concurrent processes (called *tasks* in LQN terminology) with the attached rectangles representing their externally invoked operations (called *entries*), and associated to oval symbols representing their host processors. Entries are labeled with their CPU demand, in suitable time units, and in the figure the entry labels show both a symbolic name prefixed by $, and a value. The name identifies a parameter which was estimated, and the value is the value used in the simulation to generate the data. We can later judge the estimation process by how close it comes to the original value. Entries call

or request service from other entries, indicated by arrows; the solid arrowheads here indicate that the caller waits for the response (blocking calls). An entry may have a second phase or delayed operation, after it sends a response to its requester, so here the entry appOp shows host demands and database requests for two phases.

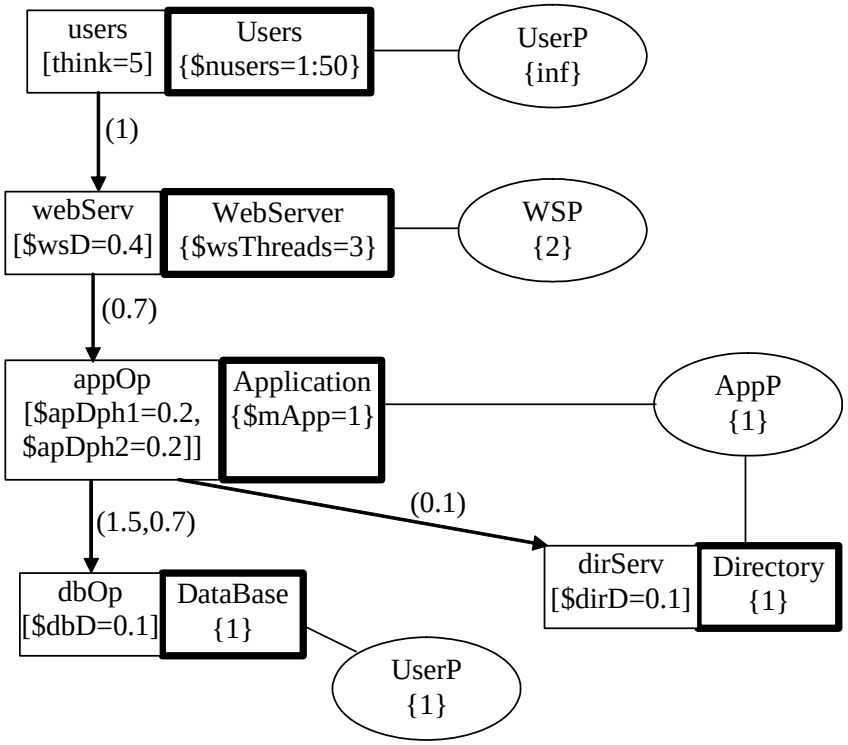

Fig 7. Layered Queueing model used for the illustration. The parameters with $name were estimated; the numbers show the values used in the simulation

This model was simulated to generate data for 10 trials, using numbers of users from 5 to 50 in steps of 10. For estimation of the seven parameters, the configuration vector **u** again consisted of the number of users. Estimation was by Gauss-Newton iteration of the nonlinear regression problem, as described in theAppendix. The derivatives needed for the H matrix were computed by finite differences, after re-running the analytic solver for a slightly (1%) perturbed value of the parameter, for parameters with real values. Parameters with integer values ($wsThreads and $mApp) were perturbed by 1 unit.

The model file webApp-lqx-1.xml follows, including the model in xml format, with the variable parameters embedded in it as parameter values, and a program in LQX.

<?xml version="1.0"?>

<!-- Invoked as: lqn2xml webApp.lqnx -->

<!-- Wed Mar 3 12:59:44 2010 -->

<lqn-model name="webApp" description="Generated by: lqn2xml, version 4.4" xmlns:xsi="htt

p://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="/usr/local/share/

lqns/lqn.xsd">

<solver-params comment="..." conv_val="1e-05" it_limit="100" print_int="1" underrelax

_coeff="0.9"/>

<processor name="InfP" scheduling="inf">

<task name="Users" scheduling="ref" multiplicity="10">

<entry name="users" type="PH1PH2">

<entry-phase-activities>

<activity name="users_ph2" phase="2" host-demand-mean="5">

<synch-call dest="respTime" calls-mean="1"/>

</activity>

</entry-phase-activities>

</entry>

</task>

<task name="Response" scheduling="inf">

<entry name="respTime" type="PH1PH2">

<entry-phase-activities>

<activity name="respTime_ph1" phase="1" host-demand-mean="0">

```
            <synch-call dest="dirServ" calls-mean="0.1"/>
            <synch-call dest="webServ" calls-mean="1"/>
          </activity>
        </entry-phase-activities>
      </entry>
    </task>
  </processor>
  <processor name="WSP" scheduling="ps" multiplicity="2">
    <task name="WS" scheduling="fcfs" multiplicity="3">
      <entry name="webServ" type="PH1PH2">
        <entry-phase-activities>
          <activity name="webServ_ph1" phase="1" host-demand-mean="$wsDemand">
            <synch-call dest="appOp" calls-mean="0.7"/>
          </activity>
</entry-phase-activities>
      </entry>
    </task>
  </processor>
  <processor name="AppP" scheduling="ps">
    <task name="App" scheduling="fcfs" multiplicity="1">
      <entry name="appOp" type="PH1PH2">
        <entry-phase-activities>
          <activity name="appOp_ph1" phase="1" host-demand-mean="$appDph1">
```

```
            <synch-call dest="dbOp" calls-mean="1.5"/>
          </activity>
          <activity name="appOp_ph2" phase="2" host-demand-mean="$appDph2">
            <synch-call dest="dbOp" calls-mean="0.7"/>
          </activity>
        </entry-phase-activities>
      </entry>
    </task>
    <task name="Dir" scheduling="fcfs">
      <entry name="dirServ" type="PH1PH2">
        <entry-phase-activities>
          <activity name="dirServ_ph1" phase="1" host-demand-mean="$DirDemand"/>
        </entry-phase-activities>
      </entry>
    </task>
  </processor>
  <processor name="DBP" scheduling="ps">
    <task name="DB" scheduling="fcfs">
      <entry name="dbOp" type="PH1PH2">
        <entry-phase-activities>
          <activity name="dbOp_ph1" phase="1" host-demand-mean="$DBopD"/>
        </entry-phase-activities>
      </entry>
```

```
    </task>

  </processor>

<lqx><![CDATA[

//array of parameters with intial values, keyed to names without $//

params = [10,0.1,0.4,0.2,0.2,0.1,1,3];

parnames =
["nusers","DirDemand","wsDemand","appDph1","appDph2","DBopD","mApp","wsThre
ad

s"];

//params to perturb for sensitivity//

sensPar = [false,true,true,true,true,true,false,false];



sens = [[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
;

//sens[paramkey][j] is derivative of output j w.r.to parameter parnames[paramkey]//



//data file to control the run... this could come in a pipe//

file_open(datafile, "datafile1", read);



//read   params (key,value) pairs to set values different from defaults//


value = 0.0;

key = 10;

println("enter integer key 0 - 7, and value, terminate -1 0");

while (key != -1)
```

```
{println("ready to read");  //***A*** for no apparent reason it needs this statment//
read_data (datafile,key,value);
if(key>-1)
      {if(key<8)
            {params[key]=value;
            //println("params ",parnames[key],"  ",params[key]);//
            }
      }
}
//println("done inputs of parameters");//


integerPar = [true,false,false,false,false,false,true,true];
deltaFraction = 0.01;


//baseline case//
//unpack all real parameters from param array//
//$nusers=params[0];//
$DirDemand=params[1];
$wsDemand=params[2];
$appDph1=params[3];
$appDph2=params[4];
$DBopD=params[5];
```

```
//$mApp=params[6];//
//$wsThreads=params[7];//

solve();
WS = task("WS");
DB = task("DB");
respTime = entry("respTime");
respTimePh1 = phase(respTime,1);
appOp = entry("appOp");
appOpPh1 = phase(appOp,1);
baseOutputs = [WS.utilization, DB.utilization,respTimePh1.service_time,
appOpPh1.service_time];
println("base outputs", baseOutputs);

//perturb the parameters with sensPar true and record derivative//
foreach(key,value in params)
{
if(sensPar[key])
        {
        increment = params[key]*deltaFraction;
        params[key] = params[key] + increment;
        //perturbed case//
```

```
        //unpack all parameters from param array//
        //$nusers=params[0];//
        $DirDemand=params[1];
        $wsDemand=params[2];
        $appDph1=params[3];
        $appDph2=params[4];
        $DBopD=params[5];
        //$mApp=params[6];//
        //$wsThreads=params[7];//
        solve();
        params[key] = params[key] - increment;
        //compute sensitivity even if a solution failed//
        outputs = [WS.utilization, DB.utilization, respTimePh1.service_time, appOpPh1.se
rvice_time];
        foreach (key2,value in sens[key])
            {sens[key][key2] = (outputs[key2] - baseOutputs[key2])/increment;
            }
        } //end if sensPar//
}//end foreach//
//print sens array, one line per parameter//
println("base outputs and sensitivity of WSutn, DButn, resp-time, appOp phase 1 service"
);
println("base outputs",baseOutputs);
```

println("Sensitivity matrix, one row per parameter");

foreach (key,value in params) {println(parnames[key]," ",sens[key]);}

]]></lqx>

</lqn-model>

This can be run with the statement lqnx webApp-lqx-1.xml. The program lqnx must be in your path; at the time of writing experimental versions are deployed by Greg Franks in /usr/loca/bin. It requires the datafile called datafile1: suitable test data is

1 0.2

2 3.3

-1 10

Requirements to use LQNX

- if you begin with an LQNS model, it must be converted to xml form using lqn2xml. For this purpose, the parameters must all be numeric.
- The string-names for parameters to be varied are then added.
- The lqx program is added at the end. Notice the format in the example.

<lqx><![CDATA[.....]]></lqx>

LQX documentation is in the LQX User Guide, found on the LQNS Documentation page.