TECHNICAL NOTE

MODELLING A COMPLEX BATCH SCHEDULE IN PEOPLESOFT

Prepared By David Kurtz, Go-Faster Consultancy Ltd.
Technical Note
Version 1.01
Wednesday 16 May 2012
(E-mail: david.kurtz@go-faster.co.uk, telephone +44-7771-760660)
File: BatchModel.doc, 16 May 2012

Contents

Introduction

This paper discusses how to model the batch schedule from a PeopleSoft system so that it is possible to determine the effects of changes:

- maximum number of concurrent processes,
- process priorities and process category priorities,
- process execution durations, possibly caused by changes to hardware
- increase in number of batch processes

I will walk through a test that demonstrates how the times that users spent waiting for different process to run changed when the priority of certain processes were changed. Users running the high priority processes spent much less time waiting on high priority process, though this was at the expense of users waiting for lower priority processes. However, this aligned with what the business users defined as being business critical.

In a previous article (More Process Priority levels for the Process Scheduler - http://blog.psftdba.com/2011/03/more-process-priority-levels-for.html) I discussed how to create additional priorities for the Process Scheduler, and how to apply them to Process Categories. I also demonstrated a way to test the behaviour of the Process Scheduler with an Application Engine process that is scheduled to run at a time and then sleeps for a specified period. Other process types can be also be simulated with these Application Engine processes, but additional process types need to be created.

I have extended that idea so that I can run processes that correspond to processes that actually ran in production and for the same time so that I can simulate the behaviour of the Process Scheduler in production and then test the effect of making changes to the priorities of processes.

How does it work?

The model works by replacing processes that ran in the original workload with an Application Engine program that sleeps for the amount of time that the original process ran. These processes are scheduled according to the time that they were originally requested.

The process that sleeps merely invokes the Java sleep method.

```
Local JavaObject &Obj;
Local ProcessRequest &RQST;
…
&Obj = CreateJavaObject("java.lang.Thread");
&Obj.start();
…
/* Directly use the Java sleep() Method for number of seconds in run control */
&Obj.sleep(GFC_SLEEP_AET.DURATION.Value * 1000);
MessageBox(0, "", 0, 0, "Sleep END");
```

The model is run by another application engine AE_SLEEP_RUN that schedules each process at the time it was originally run (relative to the start of the test). It identifies each program to run, and calculates the time at which it should run.

```
%Select(prcstype, prcsname, RQSTDTTM, DURATION, RUN_CNTL_ID )
 SELECT prcstype
 , prcsname
 , %DateTimeOut(%datetimein(%Bind(BEGINDTTM)) + (OFFSET_AMOUNT / 86400))
 , duration
 , run_cntl_id
  FROM PS_GFC_SLEEP_RC
 WHERE last_run_cntl_id = ' '
  ORDER BY rundttm , offset_amount , run_cntl_id
```

Then it schedules the process at the appropriate time with the *CreateProcessRequest* function.

```
Local ProcessRequest &RQST;

/* Create the ProcessRequest Object */
&RQST = CreateProcessRequest();

/* Set all the Required Properties */
&RQST.RunControlID = GFC_SLEEP_AET.RUN_CNTL_ID.Value;
&RQST.ProcessType = GFC_SLEEP_AET.PRCSTYPE.Value;
&RQST.ProcessName = GFC_SLEEP_AET.PRCSNAME.Value;

/* Set any Optional Properties for this Process */
&RQST.RunDateTime = GFC_SLEEP_AET.RQSTDTTM.Value;

/* Schedule the Process */
&RQST.Schedule();
```

Installing the Modeller

I have provided an Application Designer project GFC_SLEEP which contains most of what is necessary to run the test. Some of the data needs to be prepared with SQL scripts described in this document, and the Process Scheduler configuration may need to be changed manually. This project needs to be imported into the database where the batch model is to be run.

- 3 new Process Type Definitions so that the AE_SLEEP program can also be used to model other process types.
  - Pseudo Cobol SQL
  - Pseudo SQR Report
  - Pseudo XML Publisher
- Application Engine Programs
  - *AE_SLEEP* sleeps for a specified period of time and then terminates. If a next process has been specified it will also schedule that processes.
  - *AE_SLEEP1* to *AE_SLEEP11* simply call AE_SLEEP. Different processes are required for each combination of process type, category, priority, maximum concurrency and timeout. You may need to create more copies depending upon thr complexity of the load you are modelling.
  - *AE_SLEEP_RUN* is the process that runs the model. It schedules the *AE_SLEEPn* processes according to the data in *GFC_SLEEP_RC*.
- Process Definitions for all the above Application Engine programs.

Capturing Batch Workload Data

The first thing is to capture information about the load that I want to model into a database table. Below is a query that takes the timings from the process scheduler request table (PSPRCSRQST) for processes that ran in the period of interest.

I also capture:

- The process end timing from the batch timings table (PS_BAT_TIMINGS_LOG.ENDDTTM) if available. This is because the process end time on the process request (PSPRCSRQST.ENDDTTM) is updated again by the distribution server (PSDSTSRV) when posting output files to the report repository.
- The priority and maximum concurrency of the category of the process (from PS_SERVERCATEGORY).
- The priority and maximum concurrency of the server class (from PS_SERVERCLASS).

I excluded requests that relate to Process Scheduler jobs. Instead, I will model the individual processes within the jobs.

In this example, I have also limited the requests to those that run on certain Process Schedulers. When the model is run either the other process schedulers should not be available, or they should be configured not to run the server categories requests for the model.

```
REM batch_model.txt
REM on source database - in this case production - collect data to model.  midnight to 1500
on monday

DROP TABLE sysdevpg.batch_model;
CREATE TABLE sysdevpg.batch_model AS
SELECT r.prcsinstance, r.oprid, r.prcstype, r.prcsname
, GREATEST(r.rqstdttm,r.rundttm) rundttm
, NVL(l.begindttm,r.begindttm) begindttm
, NVL(l.enddttm,r.enddttm) enddttm
, r.enddttm postdttm
, (NVL(l.enddttm,r.enddttm)-NVL(l.begindttm,r.begindttm))*86400 /*-NVL(s.sleeptime,15)/2*/
secs
, p.prcscategory, p.prcspriority, p.maxconcurrent, p.timeoutmaxmins
, NVL(t.prcspriority,5) catprcspriority
, NVL(t.maxconcurrent,0) catmaxconcurrent
, NVL(c.prcspriority,5) classprcspriority
, NVL(c.maxconcurrent,0) classmaxconcurrent
FROM sysadm.psprcsrqst r
       LEFT OUTER JOIN sysadm.ps_bat_timings_log l
       ON l.process_instance = r.prcsinstance
       LEFT OUTER JOIN sysadm.ps_serverdefn s
       ON s.servername = r.servernamerun
--
       LEFT OUTER JOIN sysadm.ps_servercategory t
       ON t.servername = r.servernamerun
       AND t.prcscategory = r.prcscategory
--
       LEFT OUTER JOIN sysadm.ps_serverclass c
       ON c.servername = r.servernamerun
       AND c.opsys = r.opsys
       AND c.prcstype = r.prcstype
--
       LEFT OUTER JOIN sysadm.ps_prcsdefn p
       on p.prcstype = r.prcstype
       AND p.prcsname = r.prcsname
WHERE  r.rundttm BETWEEN TO_DATE('201102140700','yyyymmddhh24mi')
                     AND TO_DATE('201102141500','yyyymmddhh24mi')
AND    r.prcstype != 'PSJob' --omit jobs FROM simulation
AND    r.servernamerun like 'PSUNX%' -- we are only going to simulate the PSUNX process
schedulers
ORDER BY r.prcsinstance
/
```

Preparing Workload Data

Now, I need to copy the table that I have just built on the database when the original load ran, to the database where I am going to run the model. I could use export/import, or create a database link, but I think the easiest way on Oracle is to use the SQL*Plus copy command.

If I am going to model mutual exclusive processing, then I will also need to copy that configuration from the original database.

```
REM now copy table from prod to test
COPY FROM <username>/<password>@HRPRD to sysadm/<password>@HRPERF REPLACE batch_model USING
SELECT * FROM atch_model;
COPY FROM <username>/<password>@HRPRD to sysadm/<password>@HRPERF REPLACE
model_prcsmutualexcl USING SELECT * FROM sysadm.ps_prcsmutualexcl;
```

It is worth checking how many combinations of process/category priority/concurrency exist in the original workload.

```
set lines 120
COLUMN prcstype FORMAT a20
SELECT  prcstype, prcscategory, prcspriority, maxconcurrent
, timeoutmaxmins, catprcspriority, catmaxconcurrent, COUNT(*)
FROM    batch_model
GROUP BY prcstype, prcscategory, prcspriority, maxconcurrent, timeoutmaxmins,
catprcspriority, catmaxconcurrent
ORDER BY 1,2,3
/
```

The model will need as many processes as this query returns combinations. It may be necessary to clone copies of AE_SLEEP_1 and create corresponding process definitions.

PRCSTYPE	PRCSCATEGORY	P	MAXCONCURRENT	TIMEOUTMAXMINS	C	CATMAXCONCURRENT	COUNT(*)
Application Engine	Default	5	0	0	5	4	1
Application Engine	Default	5	0	0	5	8	725
Application Engine	Default	5	8	0	5	8	784
Application Engine	Default	5	12	0	5	8	1005
Application Engine	Default	5	16	0	5	8	542
Application Engine	Default	5	20	0	5	8	1195
Application Engine	Default	9	1	0	5	4	1
Application Engine	T&L Restricted Concurrency	5	16	0	5	4	227
Application Engine	T&L	9	0	0	5	1	55
Application Engine	T&L	9	0	0	5	4	4
COBOL SQL	Default	9	0	0	5	8	6
SQR Report	Default	5	0	0	5	8	173
XML Publisher	Default	5	0	0	5	8	746

I will need some extra columns on my table to do some more preparation work.

- NEW_PRCSNAME. When I run the model I am not going to request exactly the same processes. Instead I will call an Application Engine process that sleeps for the amount of time the process ran. I will have a number of copies of the same process AE_SLEEP1, AE_SLEEP2 etc which all call AE_SLEEP. This is necessary because

the original processes have different priorities, categories, and maximum concurrencies.

- NEXT_PRCSINSTANCE, LAST_PRCSINSTANCE. These columns are used to make one process scheduled another process. I call this linked processing and I will explain it in detail later on.

```
--need some extra columns
ALTER TABLE batch_model ADD
(new_prcsname VARCHAR2(12) --new process name during simulation
,next_prcsinstance NUMBER --link to next process
,last_prcsinstance NUMBER --link batch to last process
,offset_amount NUMBER --offset to next process instance
)
/
```

Where the model is simulating processes other than Application Engine processes, I have created new process type definitions which actually run Application Engine processes. This is necessary because process types have maximum concurrencies in the process scheduler which need to be modelled.

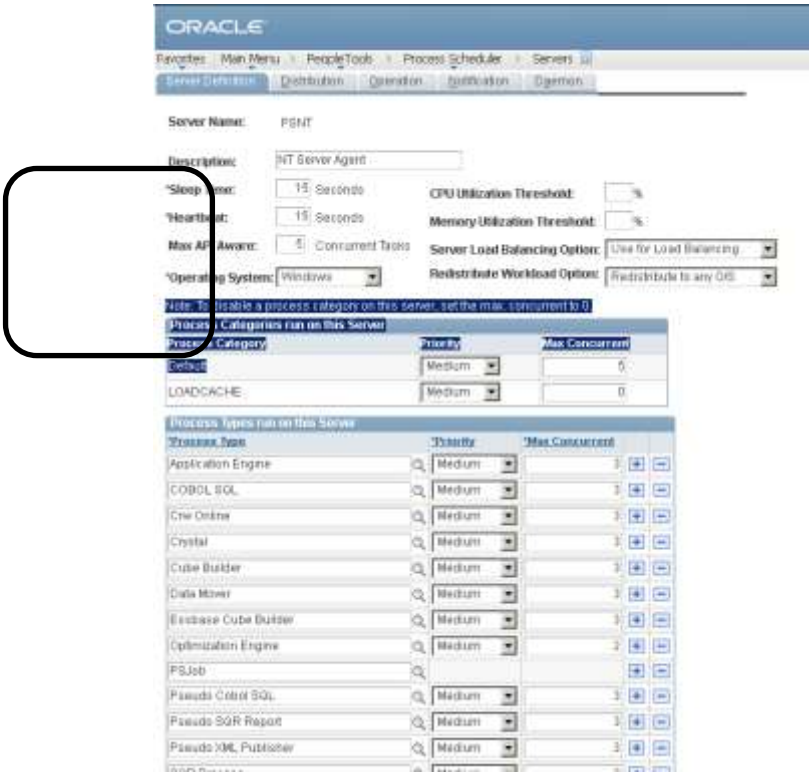
```
--switch process types to new simultaion
UPDATE batch_model
SET prcstype = 'Pseudo Cobol SQL'
WHERE prcstype = 'COBOL SQL'
/
UPDATE batch_model
SET prcstype = 'Pseudo SQR Report'
WHERE prcstype = 'SQR Report'
/
UPDATE batch_model
SET prcstype = 'Pseudo XML Publisher'
WHERE prcstype = 'XML Publisher'
/
```

Similar updates should be made to the copy of the mutual exclusive processing definition.

```
UPDATE model_prcsmutualexcl
SET prcstype = 'Pseudo Cobol SQL'
WHERE prcstype = 'COBOL SQL'
/
UPDATE model_prcsmutualexcl
SET prcstype = 'Pseudo SQR Report'
WHERE prcstype = 'SQR Report'
/
UPDATE model_prcsmutualexcl
SET prcstype = 'Pseudo XML Publisher'
WHERE prcstype = 'XML Publisher'
/
UPDATE model_prcsmutualexcl
SET prcstype_1 = 'Pseudo Cobol SQL'
WHERE prcstype_1 = 'COBOL SQL'
/
UPDATE model_prcsmutualexcl
SET prcstype_1 = 'Pseudo SQR Report'
```

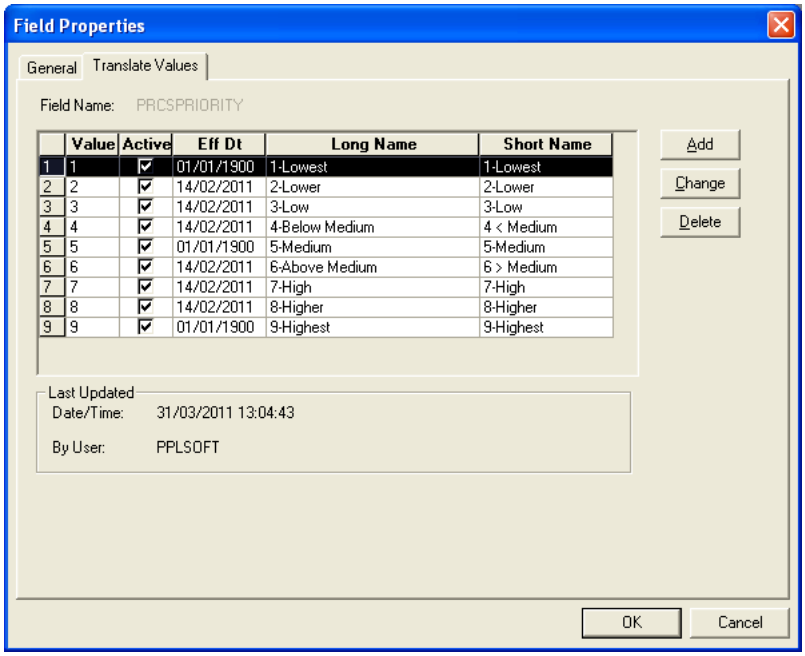
```
WHERE prcstype_1 = 'SQR Report'
/
UPDATE model_prcsmutualexcl
SET prcstype_1 = 'Pseudo XML Publisher'
WHERE prcstype_1 = 'XML Publisher'
/
```

Remember to add the new process types to the server definition.

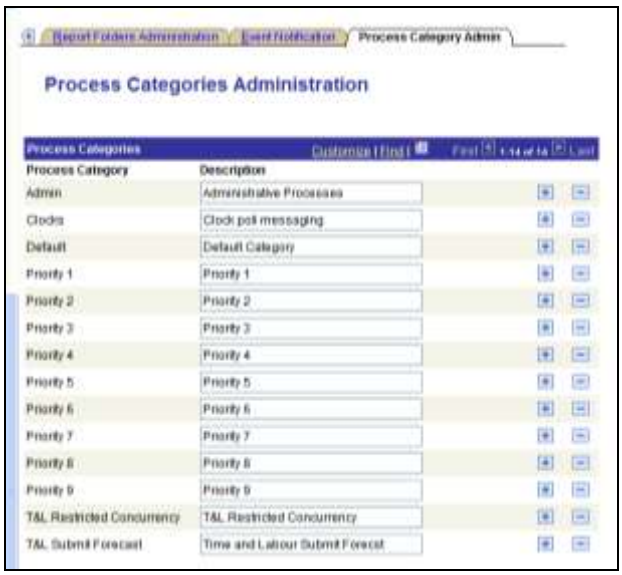


The whole purpose of running the model is to demonstrate what happens when you change something. However, I think it is advisable to run the model without making any changes to prove that it behaves like the original system.

In this example, I want to see what will happen if I adjust some process priorities. I have created additional priorities by specifying new translate values on the field PRCSPRIORITY.



I have created new Process Categories[1].



Then I have assigned those priorities to categories and assigned the categories to Process Definitions.

```
/*update server definition for new categories*/
UPDATE  ps_servercategory SET prcspriority = substr(prcscategory,10)
WHERE prcscategory like 'Priority _'
/
UPDATE  ps_servercategory SET prcspriority = '3'
WHERE not prcscategory LIKE 'Priority _'
/
UPDATE  ps_servercategory SET maxconcurrent = 0
WHERE prcscategory LIKE 'Priority _' AND NOT servername LIKE 'PSUNX%'
/
```

[1] It is necessary to restart the Process Schedulers after creating new Process Categories.

```
UPDATE  ps_servercategory d SET maxconcurrent = (
       SELECT maxapiaware FROM ps_serverdefn s WHERE s.servername = d.servername)
WHERE prcscategory LIKE 'Priority _' AND servername LIKE 'PSUNX%'
/
```

I made the Global Payroll Calculation, GPPDPRUN the highest priority process by putting it into a category whose priority is 9, and the Time & Labor calculation, TL_TMADMIN is in a category of priority 8. Two other custom processes have priority 7, and the rest are unchanged. These priorities reflect the assessment of the business.

```
--All cat 5 to cat 3
UPDATE batch_model SET prcscategory = 'Priority 9' WHERE prcsname = 'GPPDPRUN';
UPDATE batch_model SET prcscategory = 'Priority 8' WHERE prcsname = 'TL_TIMEADMIN';
UPDATE batch_model SET prcscategory = 'Priority 7' WHERE prcsname
IN('GFC_RUN_TADM','GFC_APPAY_01');
```

Now, I need to map the processes that ran on the system to be modelled with one of my processes that will sleep. I need to assign a different program for each combination of process type, process category and process priority. This PL/SQL block will write the name of the process to run during the model in *new_prcsname*, and will update the category and priority on the process definition, updating the version numbers as it goes so that the process Scheduler cache updates automatically.

The list of processes being simulated will be written to the long description of the process definition.

```
set serveroutput on
column new_prcsname heading 'New|Process|Name' format a11
--calculate new process names for simulation
BEGIN
 UPDATE pslock    SET version = version + 1 WHERE objecttypename IN('SYS','PPC');
 UPDATE psversion SET version = version + 1 WHERE objecttypename IN('SYS','PPC');

FOR i in (
  SELECT 'AE_SLEEP'||row_number() over (partition by prcstype order by prcscategory,
prcspriority) as new_prcsname
  ,     x.*
  FROM (
   SELECT DISTINCT m.prcstype, m.prcscategory, m.prcspriority, m.maxconcurrent,
m.timeoutmaxmins, m.catprcspriority, m.catmaxconcurrent
   , e.prcstype mex_prcstype, e.prcsname mex_prcsname
   FROM batch_model m
        LEFT OUTER JOIN model_prcsmutualexcl e[2]
        ON e.prcstype = m.prcstype
        AND e.prcsname = m.prcsname
```

[2] The outer join to model_prcsmutualexcl is so that a separate process definition is used to model each process which has a mutual exclusive processing definition that could affect the batch model.

```
  ) x
 ) LOOP
  dbms_output.put_line(i.prcstype||'.'||i.new_prcsname
   ||', prcscat:'||i.prcscategory
   ||', prcspriority:'||i.prcspriority
   ||', maxconc:'||i.maxconcurrent
   ||', timeout:'||i.timeoutmaxmins);

  UPDATE ps_prcsdefn
  SET    version = (SELECT version FROM psversion WHERE objecttypename = 'PPC')
  ,      descrlong = 'Simulating: '
  ,      prcscategory = i.prcscategory
  ,      prcspriority = i.prcspriority
  ,      maxconcurrent = i.maxconcurrent
  ,      timeoutmaxmins = i.timeoutmaxmins
  WHERE  prcsname = i.new_prcsname
  AND    prcstype = i.prcstype;

--dbms_output.put_line('PS_PRCSDEFN:'||sql%rowcount||' rows');

  UPDATE batch_model
  SET    new_prcsname = i.new_prcsname
  WHERE  prcstype = i.prcstype
  AND    prcscategory = i.prcscategory
  AND    prcspriority = i.prcspriority
  AND    maxconcurrent = i.maxconcurrent
  AND    timeoutmaxmins = i.timeoutmaxmins
  ;

--UPDATE ps_servercategory
--SET    prcspriority = i.catprcspriority
--,      maxconcurrent = i.catmaxconcurrent
--WHERE  servername LIKE 'PSUNX0_'
--AND    prcscategory = i.prcscategory
--;

--dbms_output.put_line('BATCH_MODEL:'||sql%rowcount||' rows');
 END LOOP;
 commit;
END;
/
```

The PL/SQL also reports of what each AE_SLEEP process is modelling.

```
Application Engine.AE_SLEEP1, prcscat:Default, prcspriority:5, maxconc:20, timeout:0
Application Engine.AE_SLEEP2, prcscat:Default, prcspriority:5, maxconc:12, timeout:0
Application Engine.AE_SLEEP3, prcscat:Default, prcspriority:5, maxconc:0, timeout:0
Application Engine.AE_SLEEP4, prcscat:Default, prcspriority:5, maxconc:16, timeout:0
Application Engine.AE_SLEEP5, prcscat:Default, prcspriority:5, maxconc:0, timeout:0
Application Engine.AE_SLEEP6, prcscat:Default, prcspriority:5, maxconc:8, timeout:0
Application Engine.AE_SLEEP7, prcscat:Default, prcspriority:9, maxconc:1, timeout:0
Application Engine.AE_SLEEP8, prcscat:Priority 8, prcspriority:9, maxconc:0, timeout:0
Application Engine.AE_SLEEP9, prcscat:Priority 8, prcspriority:9, maxconc:0, timeout:0
Application Engine.AE_SLEEP10, prcscat:T&L Restricted Concurrency, prcspriority:5, maxconc:16, timeout 0
Pseudo Cobol SQL.AE_SLEEP1, prcscat:Priority 9, prcspriority:9, maxconc:0, timeout:0
Pseudo SQR Report.AE_SLEEP1, prcscat:Default, prcspriority:5, maxconc:0, timeout:0
Pseudo XML Publisher.AE_SLEEP1, prcscat:Default, prcspriority:5, maxconc:0, timeout:0
```

Now I can delete and reinsert the mutual exclusive processing configuration that applies to the model.

```
DELETE FROM ps_prcsmutualexcl
WHERE prcsname   LIKE 'AE_SLEEP%'
OR    prcsname_1 LIKE 'AE_SLEEP%'
/

INSERT INTO ps_prcsmutualexcl
SELECT   DISTINCT m.prcstype, x.new_prcsname, m.seqno, m.prcstype_1, y.new_prcsname
from     model_prcsmutualexcl m
,        (SELECT DISTINCT prcstype, prcsname, new_prcsname from batch_model) x
,        (SELECT DISTINCT prcstype, prcsname, new_prcsname from batch_model) y
WHERE    m.prcstype = x.prcstype
and      m.prcsname = x.prcsname
and      m.prcstype_1 = y.prcstype
and      m.prcsname_1 = y.prcsname
MINUS
SELECT   prcstype, prcsname, seqno, prcstype_1, prcsname_1
FROM     ps_prcsmutualexcl
/
```

This optional piece of PL/SQL updates the long description of the process definition of the processes that will be run in the model to list the processes that it is modelling.

```
--put a meaningful desription on the processes
DECLARE
 l_prcstype VARCHAR2(30) := ' ';
 l_prcsname VARCHAR2(12) := ' ';
BEGIN
 UPDATE pslock    SET version = version + 1 WHERE objecttypename IN('SYS','PPC');
 UPDATE psversion SET version = version + 1 WHERE objecttypename IN('SYS','PPC');

 UPDATE ps_prcsdefn
 SET    version = (SELECT version FROM psversion WHERE objecttypename = 'PPC')
 ,      descrlong = 'Not currently used in simultation'
 WHERE  prcsname LIKE 'AE_SLEEP%'
 AND    prcsname != 'AE_SLEEP_RUN';

 FOR i IN(
  SELECT DISTINCT prcstype, prcsname, new_prcsname
  FROM     batch_model
  ORDER BY prcstype, prcsname, new_prcsname
 ) LOOP
  IF i.prcstype = l_prcstype AND i.new_prcsname = l_prcsname THEN
   UPDATE ps_prcsdefn
   SET    descrlong = descrlong||', '||i.prcsname
   WHERE  prcstype = i.prcstype
   AND    prcsname = i.new_prcsname;
  ELSE /*first time*/
   l_prcstype := i.prcstype;
   l_prcsname := i.new_prcsname;
   UPDATE ps_prcsdefn
   SET    version = (SELECT version FROM psversion WHERE objecttypename = 'PPC')
   ,      descrlong = 'Simulating: '||i.prcsname
   WHERE  prcstype = i.prcstype
   AND    prcsname = i.new_prcsname;
  END IF;
 END LOOP;
 COMMIT;
END;
/
```

Linked Processing

During the model, I could just schedule all the processes to run at the same time relative to the start of the test period at which they were originally scheduled. However, in this business from which this example was taken, an operator runs one process, waits for it to run, and on the basis of the output of that process does some more work and then runs another process. In reality, the request time of the second process depends on when the first process finishes. If the first process finishes earlier, the second process may be requested earlier, and so on. I want to model what effect that has.

This query checks that combination of operator ID and request date/time is unique. It is not guaranteed, but it should be. Otherwise, it is not possible to say which is the 'next process' for an operator.

```
--check uniqueness
SELECT oprid, rundttm, count(*)
FROM   batch_model
WHERE  oprid != 'batchuser'
GROUP BY oprid, rundttm
HAVING COUNT(*)> 1
/
```

Thus, the processes run by each operator become a linked list. When a process finishes, I need to know the next process to run. That is stored in NEXT_PRCSINSTANCE. I also need to know that a process will be scheduled by the previous process, so I put that in LAST_PRCSINSTANCE. I excluded overlapping processes from this process. Each process only schedules one following process, so both of these columns are also unique, but may also be null.

I have deliberately omitted jobs, run on this system by '*batchuser*', from the linked processing because these are scheduled to run at particular times, and so I want this to be the same during the model.

This linking can also be used to model PeopleSoft jobs and job sets.

```
--current, next and last process instance numbers must each be unique
CREATE UNIQUE INDEX batch_model_idx1 on batch_model (prcsinstance);
CREATE UNIQUE INDEX batch_model_idx2 ON batch_model (last_prcsinstance);
CREATE UNIQUE INDEX batch_model_idx3 ON batch_model (next_prcsinstance);

--calculate links back and forth between processes
BEGIN
 FOR i IN (
  select n.oprid, n.prcsinstance, n.prcsname, n.rundttm, n.enddttm
  ,      l.prcsinstance last_prcsinstance
  ,      l.enddttm last_enddttm
  FROM   batch_model l
  ,      batch_model n
  where  l.oprid != 'batchuser'
  and    n.oprid = l.oprid
  and    n.rundttm > l.enddttm
  and    n.rundttm = (
       SELECT MIN(n1.rundttm)
```

```
       FROM    batch_model n1
       WHERE   n1.oprid = l.oprid
       AND     n1.rundttm > l.rundttm)
 ) LOOP
  --set backward from next
  UPDATE batch_model
  SET    last_prcsinstance = i.last_prcsinstance
  ,      offset_amount = (i.rundttm-i.last_enddttm)*86400 --offset from last process
  WHERE  prcsinstance = i.prcsinstance
  ;
  --set forward from last
  UPDATE batch_Model
  SET    next_prcsinstance = i.prcsinstance
  WHERE  prcsinstance = i.last_prcsinstance
  AND    next_prcsinstance IS NULL
  ;
 END LOOP;
 commit;
END;
/
```

This query checks that there is no double linking in either direction. It should be impossible.

```
--check double linking - there should not be any
column prcsinstance format 99999999
column next_prcsinstance format 99999999
column last_prcsinstance format 99999999
select l.prcsinstance, l.next_prcsinstance, n.last_prcsinstance, n.prcsinstance
FROM   batch_model l
,      batch_model n
where  ((l.next_prcsinstance = n.prcsinstance) or (l.prcsinstance = n.last_prcsinstance))
AND    (l.next_prcsinstance IS NULL OR n.last_prcsinstance IS NULL)
order by 1
/
```

However, if you don't want to use this linked processing logic, simply leave or reset the linking fields as null.

```
--suppress linked processing
UPDATE batch_model
SET last_prcsinstance = NULL
,   next_prcsinstance = NULL
/
```

Setting up the Model

Having prepared the data in my working storage table, I need to copy it to the run control table GFC_SLEEP_RC which will be used during the model to schedule the requests.

During my testing, I found I need to apply some corrections to the run times.

- There is an overhead to getting the Application Engine process started, and writing log files at the end. So I have deducted 2 seconds from the time that the process sleeps, so that the elapsed execution time of the processes during the model were the same as in the system being simulated.
- I found that the Process Scheduler was quicker to schedule requests during the simulation because there is very little database activity when the model runs compared to the real system. The distribution server also had less work to do because there were fewer and smaller files to post. So, I had to add 5 seconds to the sleep time to compensate for this.

Note that these corrections may vary depending on hardware, operating system and PeopleTools version. It may not be appropriate to apply the same corrections across the whole of the model. Getting this adjustments right is a matter of train and error. Therefore, it is important to demonstrate that the model can first reproduce the current system behaviour before making any changes.

You might also want to create additional processes to run during the test, or to deliberately increase run times across the board. It depends what change you want to model.

```
TRUNCATE TABLE ps_gfc_sleep_rc
/
REM fudge factors
REM -2 App Engine Overhead
REM +5 Adjustment for posting overhead
insert into PS_GFC_SLEEP_RC
(run_cntl_id, prcstype, prcsname, offset_amount, duration, rundttm, last_run_cntl_id,
next_run_cntl_id)
select a.prcsinstance, a.prcstype, a.new_prcsname
, (a.rundttm-TO_DATE('201102140700','yyyymmddhh24mi'))*86400 offset
, GREATEST(0,a.secs-2+5) duration
, rundttm
, NVL(TO_CHAR(a.last_prcsinstance),' ')
, NVL(TO_CHAR(a.next_prcsinstance),' ')
FROM batch_model a
where a.rundttm BETWEEN TO_DATE('201102140700','yyyymmddhh24mi')
                    AND TO_DATE('201102142359','yyyymmddhh24mi') --due to start in period of
interest
AND a.last_prcsinstance IS NULL --not linked to previous process
union
select a.prcsinstance, a.prcstype, a.new_prcsname
, (a.rundttm-l.enddttm)*86400 offset
, GREATEST(0,a.secs-2+5) duration
, a.rundttm
, NVL(TO_CHAR(a.last_prcsinstance),' ')
, NVL(TO_CHAR(a.next_prcsinstance),' ')
FROM batch_model a, batch_model l
```

```
where a.rundttm BETWEEN TO_DATE('201102140700','yyyymmddhh24mi')
                    AND TO_DATE('201102142359','yyyymmddhh24mi') --due to start in period of
interest
AND a.last_prcsinstance = l.prcsinstance --linked to previous process
AND a.last_prcsinstance IS NOT NULL
union
select a.prcsinstance, a.prcstype, a.new_prcsname
, 0 offset
, GREATEST(0,a.secs-2+5) duration
, a.rundttm
, NVL(TO_CHAR(a.last_prcsinstance),' ')
, NVL(TO_CHAR(a.next_prcsinstance),' ')
FROM batch_model a
where a.rundttm   <= TO_DATE('201102140700','yyyymmddhh24mi') --due to start prior to period
of interest
AND   a.begindttm >= TO_DATE('201102140700','yyyymmddhh24mi') --but actually started after
beginning of period
union
select a.prcsinstance, a.prcstype, a.new_prcsname
, 0 offset
, GREATEST(0,(a.enddttm-TO_DATE('201102140700','yyyymmddhh24mi'))*86400-2+5) duration
, a.rundttm
, NVL(TO_CHAR(a.last_prcsinstance),' ')
, NVL(TO_CHAR(a.next_prcsinstance),' ')
FROM  batch_model a
where a.begindttm <= TO_DATE('201102140700','yyyymmddhh24mi') --running at start of period
AND   a.enddttm   >= TO_DATE('201102140700','yyyymmddhh24mi')
/

REM clear out broken links for linked processing
UPDATE ps_gfc_sleep_rc a
SET a.last_run_cntl_id = ' '
,   a.offset_amount = (a.rundttm-TO_DATE('201102140700','yyyymmddhh24mi'))*86400
WHERE a.last_run_cntl_id != ' '
AND NOT EXISTS(
        SELECT 'x' FROM ps_gfc_sleep_rc b
        WHERE b.run_cntl_id = a.last_run_cntl_id)
/
UPDATE ps_gfc_sleep_rc a
SET a.next_run_cntl_id = ' '
WHERE a.next_run_cntl_id != ' '
AND NOT EXISTS(
        SELECT 'x' FROM ps_gfc_sleep_rc b
        WHERE b.run_cntl_id = a.next_run_cntl_id)
/
UPDATE ps_gfc_sleep_rc
SET    last_run_cntl_id = ' '
WHERE  offset_amount <= 0
AND    last_run_cntl_id != ' '
AND 1=2
/
```

I also check the run control table for any processes linked to processes that have not also been copied into it.

```
--check for broken links in the model table
COLUMN run_cntl_id FORMAT a10
COLUMN next_run_cntl_id FORMAT a10
COLUMN last_run_cntl_id FORMAT a10
SELECT  l.run_cntl_id, l.next_run_cntl_id, n.last_run_cntl_id, n.run_cntl_id
FROM    ps_gfc_sleep_rc l
,       ps_gfc_sleep_rc n
WHERE   ((l.next_run_cntl_id = n.run_cntl_id) or (l.run_cntl_id = n.last_run_cntl_id))
AND     (l.next_run_cntl_id = ' ' OR n.last_run_cntl_id = ' ')
ORDER BY 1
/
COMMIT
/
```

This query reports the priority and concurrency for processes in original workload and for the corresponding processes in the model. They may be the same, or they may deliberately be different.

```
COLUMN orig_prcscategory HEADING 'Original|Process|Category' FORMAT a13
COLUMN prcstype HEADING 'Test|Process|Type' FORMAT a12
COLUMN prcsname HEADING 'Model|Process|Name'
COLUMN prcscategory HEADING 'Model|Process|Category' FORMAT a13
COLUMN orig_prcspriority HEADING 'Org|Prc|Pri' FORMAT a3
COLUMN prcspriority HEADING 'Mod|Prc|Pri' FORMAT a3
COLUMN orig_maxconcurrent HEADING 'Orig|Proc|Max|Conc' FORMAT 999
COLUMN      maxconcurrent HEADING 'Mod|Proc|Max|Conc' FORMAT 999
COLUMN orig_catprcspriority HEADING 'Org|Cat|Pri' FORMAT a3
COLUMN      catprcspriority HEADING 'Mod|Cat|Pri' FORMAT a3
COLUMN orig_catmaxconcurrent HEADING 'Orig|Cat|Max|Conc' FORMAT 999
COLUMN      catmaxconcurrent HEADING 'Mod|Cat|Max|Conc' FORMAT 999
COLUMN orig_classprcspriority HEADING 'Org|Cls|Pri' FORMAT a3
COLUMN      classprcspriority HEADING 'Mod|Cls|Pri' FORMAT a3
COLUMN orig_classmaxconcurrent HEADING 'Org|Cls|Max|Conc' FORMAT 999
COLUMN      classmaxconcurrent HEADING 'Mod|Cls|Max|Conc' FORMAT 999
SELECT  DISTINCT
        r.prcstype, r.prcsname
--      m.prcsname orig_prcsname,
,       m.prcscategory orig_prcscategory
,       p.prcscategory
,       m.prcspriority orig_prcspriority
,       p.prcspriority
,       m.maxconcurrent orig_maxconcurrent
,       p.maxconcurrent
,       m.catprcspriority orig_catprcspriority
,       c.prcspriority
,       m.catmaxconcurrent orig_catmaxconcurrent
,       c.maxconcurrent catmaxconcurrent
,       m.classprcspriority orig_classprcspriority
,       l.prcspriority classprcspriority
,       m.classmaxconcurrent orig_classmaxconcurrent
,       l.maxconcurrent classmaxconcurrent
```

```
--,     c.servername
FROM    ps_gfc_sleep_rc r
,       batch_model m
,       ps_prcsdefn p
,       ps_serverdefn s
,       ps_servercategory c
,       ps_serverclass l
WHERE   TO_NUMBER(r.run_cntl_id) = m.prcsinstance
AND     p.prcstype = r.prcstype
AND     p.prcsname = r.prcsname
AND     c.prcscategory = p.prcscategory
AND     s.servername LIKE 'PSUNX%'
AND     c.servername = s.servername
AND     l.servername = s.servername
AND     l.opsys = s.opsys
AND     l.prcstype = p.prcstype
ORDER BY r.prcstype, r.prcsname
/
```

So, I can see that there are some Processes Schedulers where the maximum concurrency for the category and the server class is different for some schedulers. This is something that I need to correct before I run the test.

```
                                                        Orig  Mod         Orig  Mod          Org  Mod
Test         Model       Original     Model        Org Mod Proc Proc Org Mod  Cat  Cat Org Mod  Cls  Cls
Process      Process     Process      Process      Prc Prc  Max  Max Cat Prc  Max  Max Cls Cls  Max  Max
Type         Name        Category     Category     Pri Pri Conc Conc Pri Pri Conc Conc Pri Pri Conc Conc
------------ ----------- ------------ ------------ --- --- ---- ---- --- --- ---- ---- --- --- --
Application  AE_SLEEP10  T&L Restricte T&L Restricte 5   5     16   16 5   5      4    0 5   5      6    3
Engine                   d Concurrency d Concurrency

Application  AE_SLEEP10  T&L Restricte T&L Restricte 5   5     16   16 5   5      4    4 5   5      6    6
Engine                   d Concurrency d Concurrency
…
```

The model is then run by simply running the AE_SLEEP_RUN application engine.

After the model has run, I use this query to see how it compares with the original. It is also used to produce the charts in the Example Results section below.

```
set lines 130
COLUMN orig_prcsinstance HEADING 'Original|P.I.' FORMAT 9999999
COLUMN prcsinstance      HEADING 'Model|P.I.'    FORMAT 9999999
COLUMN prcsname          HEADING 'Process|Name'
COLUMN orig_prcsname     HEADING 'Original|Process|Name'
COLUMN orig_start_offset HEADING 'Original|Start|Offset'  FORMAT 9999999
COLUMN orig_start_delay  HEADING 'Original|Start|Delay'   FORMAT 9999999
COLUMN orig_duration     HEADING 'Original|Duration'      FORMAT 9999999
COLUMN orig_post_delay   HEADING 'Original|Posting|Delay' FORMAT 9999999
COLUMN test_start_offset HEADING 'Test|Start|Offset'  FORMAT 99999
COLUMN test_start_delay  HEADING 'Test|Start|Delay'   FORMAT 999999
COLUMN test_duration     HEADING 'Test|Duration'      FORMAT 9999999
COLUMN test_post_delay   HEADING 'Test|Posting|Delay' FORMAT 999999
COLUMN next_run_cntl_id  HEADING 'Next Run|Control|ID'format a8
COMPUTE SUM OF orig_duration on report
```

```
COMPUTE SUM OF test_duration on report
COMPUTE SUM OF orig_start_delay on report
COMPUTE SUM OF test_start_delay on report
COMPUTE SUM OF orig_post_delay on report
COMPUTE SUM OF test_post_delay on report
break on report
SELECT x.* from (
SELECT  m.prcsinstance orig_prcsinstance
,       m.prcsname orig_prcsname
--,     m.rundttm orig_sched_time
,       (m.rundttm-TO_DATE('201102140700','yyyymmddhh24mi'))*86400 orig_start_offset
,       (m.begindttm-m.rundttm)*86400 orig_start_delay
,       (m.enddttm-m.begindttm)*86400 orig_duration
,       (m.postdttm-m.enddttm)*86400 orig_post_delay
,       y.prcsinstance
,       y.prcsname
,       (y.rundttm-x.begindttm)*86400 test_start_offset
,       (y.begindttm-y.rundttm)*86400 test_start_delay
,       (NVL(l.enddttm,y.enddttm)-NVL(l.begindttm,y.begindttm))*86400 test_duration
,       (y.enddttm-l.enddttm)*86400 test_post_delay
,       r.last_run_cntl_id
,       r.next_run_cntl_id
,       y.servernamerun
,       y.prcscategory
,       p.prcspriority
FROM    ps_gfc_sleep_rc r
,       batch_model m
,       psprcsrqst y
        LEFT OUTER JOIN ps_bat_timings_log l
        ON l.process_instance = y.prcsinstance
,       psprcsrqst x
,       ps_prcsdefn p
WHERE   TO_NUMBER(r.run_cntl_id) = m.prcsinstance
AND     y.runcntlid = r.run_cntl_id
AND     y.prcsinstance > x.prcsinstance
AND     m.begindttm >= TO_DATE('201102140700','yyyymmddhh24mi') --don't bother with processes
whose runtimes have been shrunk
AND     m.rundttm   >= TO_DATE('201102140700','yyyymmddhh24mi') --don't bother with processes
due to be scheduled before the model
AND     x.prcsinstance = (
                SELECT  MAX(x1.prcsinstance)
                FROM    psprcsrqst x1
                WHERE   x1.prcsname = 'AE_SLEEP_RUN'
                AND     x1.runstatus IN('7','9'))
AND     p.prcstype = r.prcstype
AND     p.prcsname = r.prcsname
) x ORDER BY orig_start_offset, prcsinstance
/
```

This can be run after the test to check how a linked processing in the batch model has worked.

```
rem test linked processing setup
SELECT  m.prcsinstance, m.next_prcsinstance
,       (m1.rundttm-m.enddttm)*86400 model_offset
,       l.enddttm
,       y1.rqstdttm
```

```
,       y1.rundttm
FROM    ps_gfc_sleep_rc r
,       batch_model m
,       batch_model m1
,       psprcsrqst x
,       psprcsrqst y
,       psprcsrqst y1
,       ps_bat_timings_log l
WHERE   TO_NUMBER(r.run_cntl_id) = m.prcsinstance
AND     y.runcntlid = r.run_cntl_id
AND     y1.runcntlid = r.next_run_cntl_id
AND     y.prcsinstance > x.prcsinstance
AND     y1.prcsinstance > x.prcsinstance
AND     l.process_instance = y.prcsinstance
AND     m.begindttm >= TO_DATE('201102140700','yyyymmddhh24mi') --don't bother with processes
whose runtimes have been shrunk because they started before period of interest
AND     m.rundttm   >= TO_DATE('201102140700','yyyymmddhh24mi') --don't bother with processes
due to be scheduled before the model
AND     x.prcsinstance = (
                SELECT  MAX(x1.prcsinstance)
                FROM    psprcsrqst x1
                WHERE   x1.prcsname = 'AE_SLEEP_RUN'
                AND     x1.runstatus IN('7','9'))
AND     m1.prcsinstance = m.next_prcsinstance
AND     m.prcsinstance = 5213435 --arbitary process to check
/
```

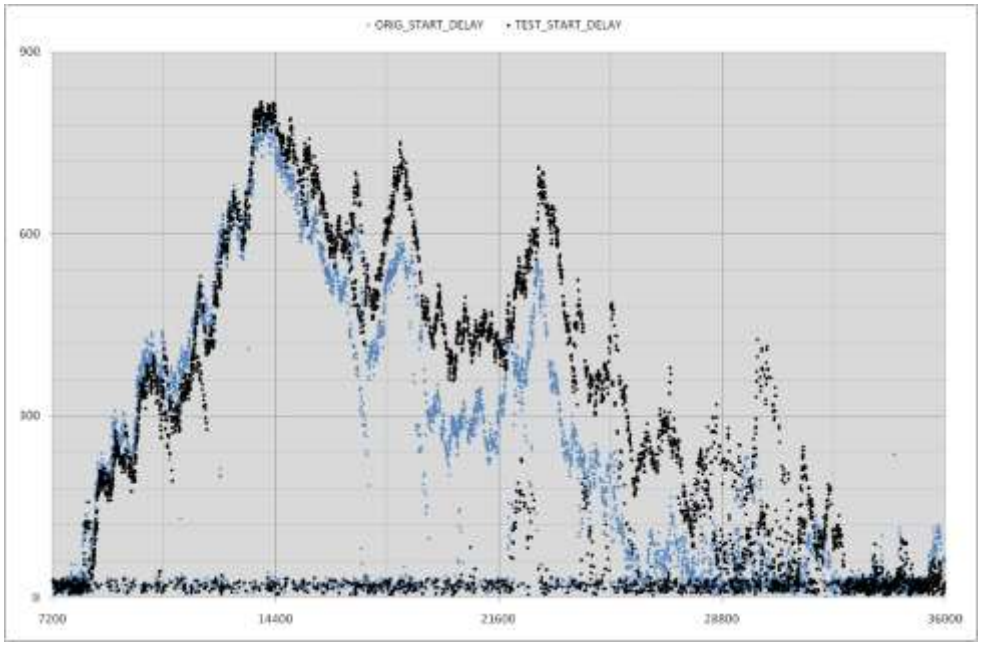
Example Results

I think the easiest way to see the results of the model is graphically. Below, I have simply graphed the amount of time that each request queued (the time between the actual start time and whichever is the greater of the time the request was scheduled, and the requested start time) against time since the start of the period being modelled. The following query was used to generate the data in the charts.

```
set lines 180
COLUMN orig_prcsinstance HEADING 'Original|P.I.' FORMAT 9999999
COLUMN prcsinstance HEADING 'Model|P.I.' FORMAT 9999999
COLUMN prcstype HEADING 'Process Type' Format a20
COLUMN prcsname HEADING 'Process|Name'
COLUMN orig_prcsname HEADING 'Original|Process|Name'
COLUMN orig_start_offset HEADING 'Org.|Start|Offset' FORMAT 99999999
COLUMN orig_start_delay HEADING 'Org.|Start|Delay' FORMAT 9999999
COLUMN orig_duration HEADING 'Org.|Duration' FORMAT 9999999
COLUMN orig_post_delay HEADING 'Org.|Posting|Delay' FORMAT 9999999
COLUMN test_start_offset HEADING 'Test|Start|Offset' FORMAT 99999
COLUMN test_start_delay HEADING 'Test|Start|Delay' FORMAT 999999
COLUMN test_duration HEADING 'Test|Duration' FORMAT 9999999
COLUMN test_post_delay HEADING 'Test|Posting|Delay' FORMAT 999999
COLUMN next_run_cntl_id HEADING 'Next Run|Control|ID'format a8
COMPUTE SUM OF orig_duration on report
```

```
COMPUTE SUM OF test_duration on report
COMPUTE SUM OF orig_start_delay on report
COMPUTE SUM OF test_start_delay on report
COMPUTE SUM OF orig_post_delay on report
COMPUTE SUM OF test_post_delay on report
break on report
SELECT x.* from (
SELECT m.prcsinstance orig_prcsinstance
, m.prcsname orig_prcsname
--, m.rundttm orig_sched_time
, (m.rundttm-z.origin)*86400 orig_start_offset
, (m.begindttm-m.rundttm)*86400 orig_start_delay
, (m.enddttm-m.begindttm)*86400 orig_duration
, (m.postdttm-m.enddttm)*86400 orig_post_delay
, y.prcsinstance
, y.prcsname
, (y.rundttm-x.begindttm)*86400 test_start_offset
, (y.begindttm-y.rundttm)*86400 test_start_delay
, (NVL(l.enddttm,y.enddttm)-NVL(l.begindttm,y.begindttm))*86400 test_duration
, (y.enddttm-l.enddttm)*86400 test_post_delay
, r.last_run_cntl_id
, r.next_run_cntl_id
, y.servernamerun
, y.prcscategory
, p.prcspriority
, m.oprid
FROM (
  SELECT min(rundttm-offset_amount/86400) origin
  FROM   sysadm.ps_gfc_sleep_rc
  WHERE  offset_amount > 0) z
, sysadm.ps_gfc_sleep_rc r
, batch_model m
, sysadm.psprcsrqst y
LEFT OUTER JOIN ps_bat_timings_log l
ON l.process_instance = y.prcsinstance
, sysadm.psprcsrqst x
, sysadm.ps_prcsdefn p
WHERE TO_NUMBER(r.run_cntl_id) = m.prcsinstance
AND y.runcntlid = r.run_cntl_id
AND y.prcsinstance > x.prcsinstance
AND m.begindttm >= z.origin --dont bother with processes whose runtimes have been shrunk
AND m.rundttm >= z.origin --dont bother with processes due to be scheduled before the model
AND x.prcsinstance = (
  SELECT MAX(x1.prcsinstance)
  FROM sysadm.psprcsrqst x1
  WHERE x1.prcsname = 'AE_SLEEP_RUN'
  AND x1.runstatus IN('7','9'))
AND p.prcstype = r.prcstype
AND p.prcsname = r.prcsname
) x ORDER BY orig_start_offset, prcsinstance
/
```

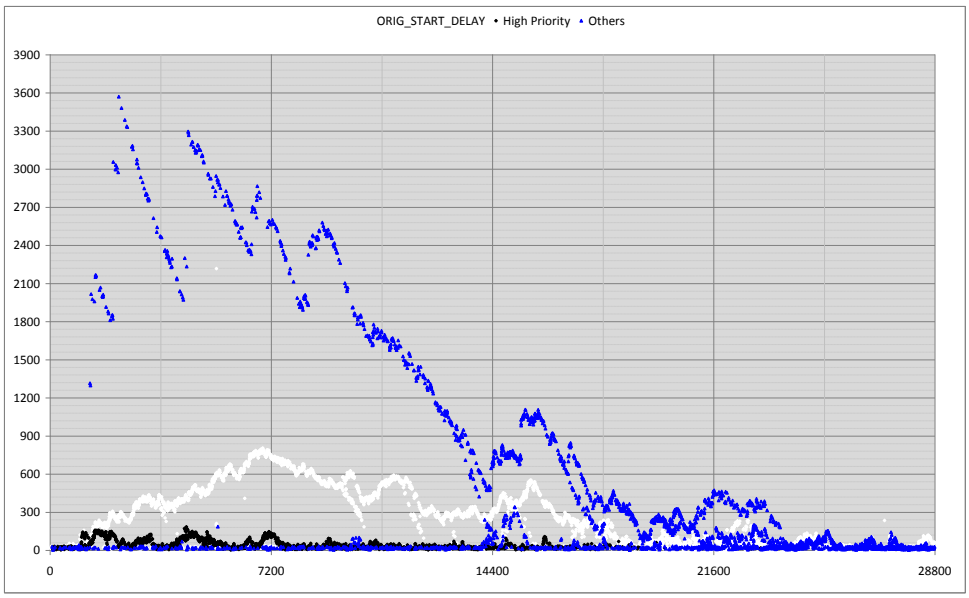
The first graph shows the result of modelling the original load without any changes to the configuration. The data in this chart is based on a real live system. It shows that model is not perfect, but the peaks and troughs do follow each other, though the sets of points do gradually diverge. It is possible that the 5 second fudge factor should be slightly different at different times during the model.



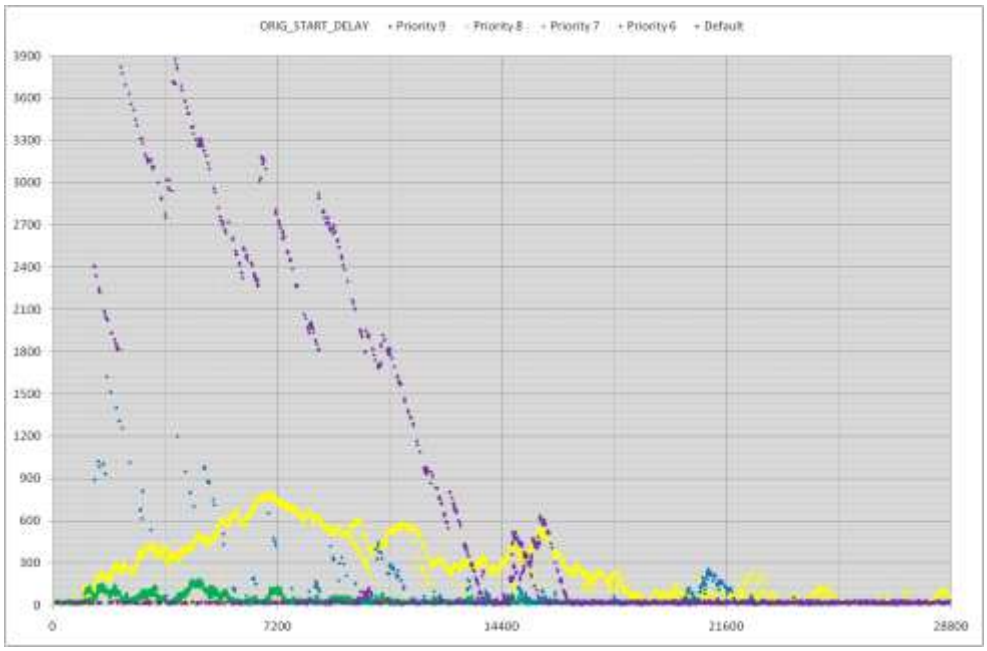
However, I think it is close enough that the model is behaving like production, and it would be reasonable to use this to predict the effect of production changes.

When I ran the model with the new priorities the effect is dramatic.

- The high priority processes only queue for up to 3 minutes instead of 15.
- There is a price to pay, the lower priority processes queue for up to an hour as the Process Scheduler runs the high priority processes.



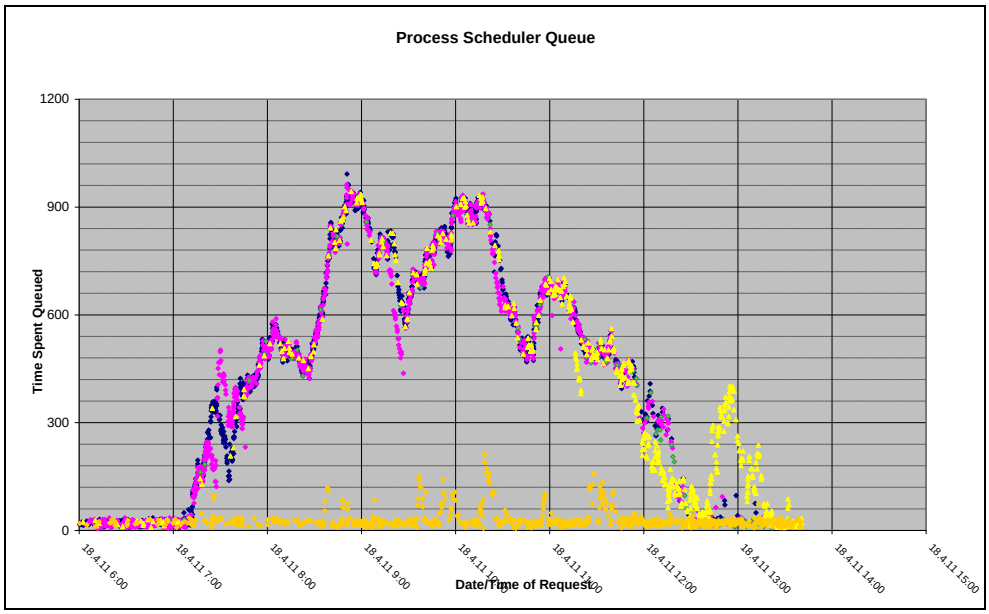
Then I put some other proesses into a category of priority 6, giving them a higher priority than the rest, but not as high as the critical processes that I had already identified. These proesses queued for a while during the early part of the test, but not as long as some of the others. But the peak queue time for the default priority processes when up from 3600 to 3900 seconds.



This test makes the effect and magnitude of the change is very clear. The people responsible for the business that this system supports can now take a decision as to whether this change to Process Scheduler configuration is desirable.

In Production

Rather gratifyingly, when this change was put into production, the production system did indeed behave in the manner predicted by the model. This is how the system looked before the change with different processes in different colours.



And this is how the same system looked 2 weeks later when the Process Scheduler had been reconfigured. This is very similar to the prediction.