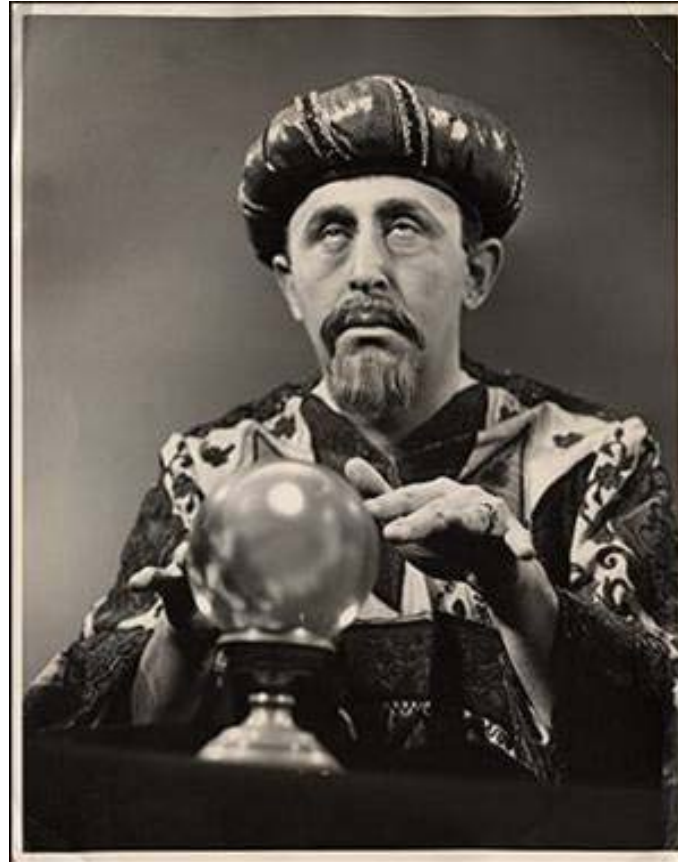


# > BUSINESS MADE **SIMPLE**



**Cost Based  
Performance /Capacity  
analysis**



**Eugene Margulis**

*eugene\_margulis@yahoo.ca*

**April 2009**

**NORTEL**



- What issues/questions are addressed by performance “stuff” ?
- Performance/capacity cost model
- Examples and demo
- Using the cost based model
- Cost Based Model and the development cycle
- Benefits



## **What issues are addressed by Performance / Capacity activities?**

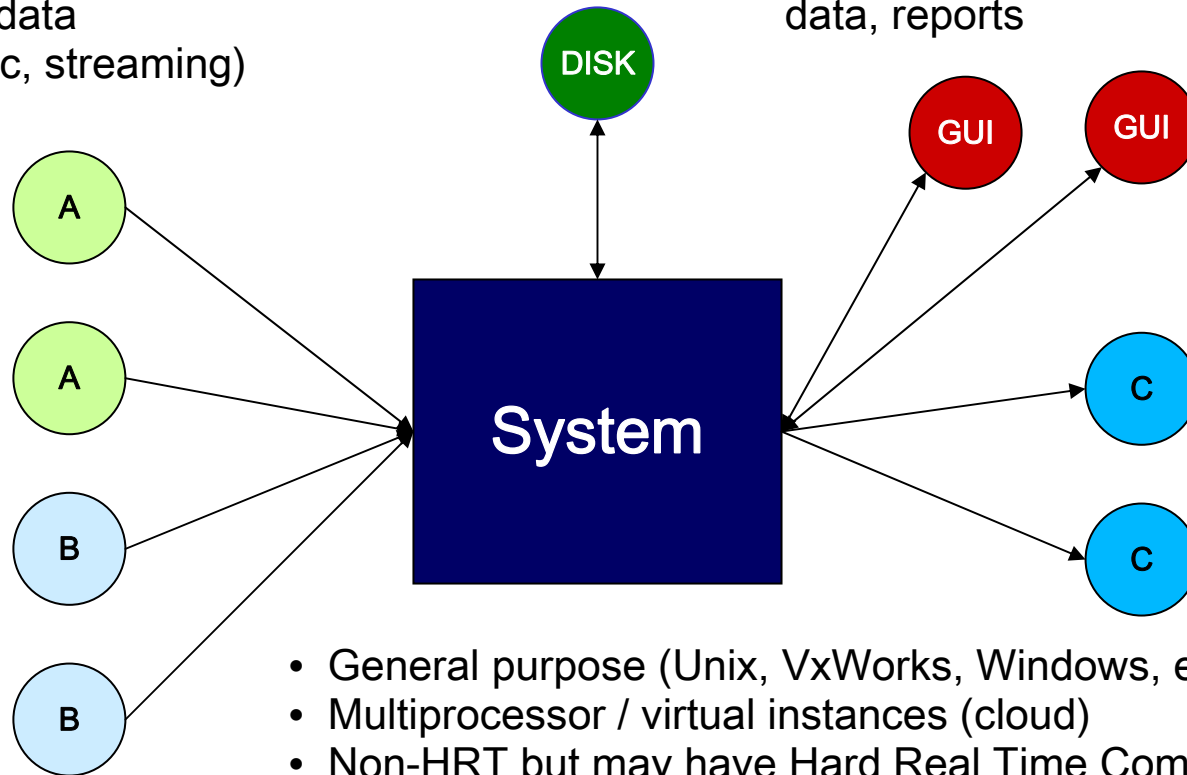
*... lets define the reference system for the sake of discussion*

# Reference System



Inputs/Devices:  
events/data  
(periodic, streaming)

Outputs: GUI, devices,  
data, reports



- General purpose (Unix, VxWorks, Windows, etc)
- Multiprocessor / virtual instances (cloud)
- Non-HRT but may have Hard Real Time Component
- Has 3<sup>rd</sup> party code – binary only, no access
- Heterogeneous s/w – scripts, C, multiple JVMs

## Real world “challenges”:



- **Requirements / Behavior uncertainty:**
  - Performance Requirements are not well defined
  - Load levels (S/M/L) or “KPI”s are speculated, not measured (cannot be measured)
- **Code uncertainty:**
  - No Access to large portions of code / can’t rebuild/recompile
- **H/W uncertainty:**
  - Underlying H/W architecture is not fixed (and can be very different)

*This is not a strictly Testing/Verification activity but rather an **exploratory exercise** where we need to **discover/understand rather than verify***

- **With additional complication:**
  - Designers in India, Testers in Vietnam, Architects in Canada, Customers in Spain (How to exchange information??)

## Some questions addressed by performance “stuff”



- Will **timing requirements** be met? All the time? Under what conditions? Can we **guarantee** it? What is relationship between latency and max rate?
- Will we have **enough disk space** (for how long)?
- **What if** we run the system on **HW** with 32 slow processors? (instead of 4 fast ones?) What would be max supported rate of events then?
- **What if** the amount of **memory** is reduced? What would be max supported rate of events then?
- **What if** some GUIs are in **China?** (increase RTT)
- Do we have **enough** spare capacity for an **additional application X?**
- Is our system performing better/worse compare to the last release (**degradation**)?
- What **customer visible activity** (not process name/id, not an IP port, not a 3<sup>rd</sup> party DB) **uses the most of resources?** (e.g. CPU? Memory? Heap? BW? Disk?)
- **What if** we have **two times as many of type A devices?** What is the max size of network we can support? How does performance map to **Business Model?**

## A few observations on the questions...



- *Yes, we can test it in the lab (at least some) ....  
... but can we have the answers by **tomorrow??***
- Notice that:
  - End Customers do not care about CPU%ge, memory utilization, etc. – but do care about **latency and scalability** (w.r.t customer “payload” activity). However, resource utilization metrics can be used for degradation tracking internally.
  - End Customer / Engineering do not care about resource utilization per a particular process (e.g. using top) – but do care about **resource utilization** w.r.t. **customer payload activity (behaviour)**
  - Very **few performance aspects are pass/fail** (outside of HRT/military/etc.)

## Areas addressed by perf questions:



- Any Useful Performance approach needs to address:

- Business Mapping
- Characterization
- Forecasting
- Optimization
- Tracking



- Lab testing alone does not address this (efficiently)

## What we need...



- A **flexible** mapping between customer **behaviour** and **performance/capacity metrics** of the system
- But there is a problem...
  - There is **HUGE** number of different behaviours – even in the simplest of system...
- Trade-offs:
  - Can we reduce the problem space and still have something useful/practical?
  - Willing to trade-off accuracy for speed (how accurate are the inputs?)

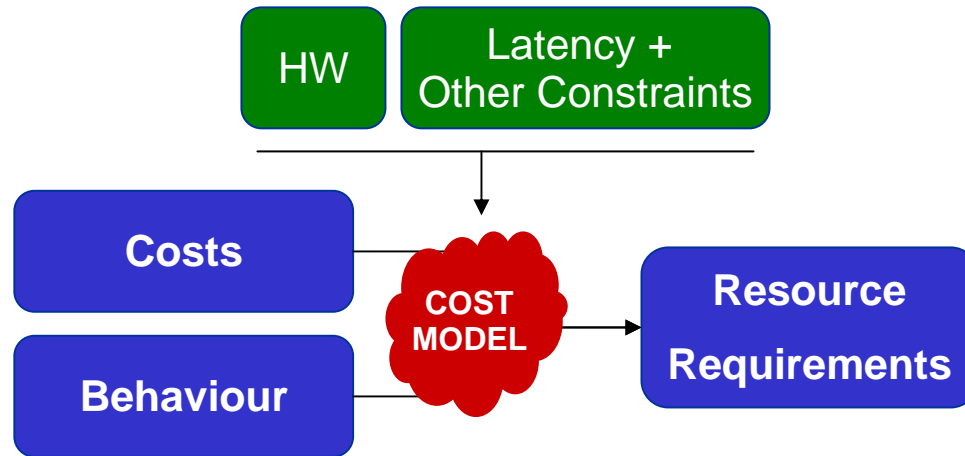


- System processes **TRANSACTIONS**
- 20% of **TRANSACTIONS** responsible for 80% of “performance” during Steady state operations
- Focus on **steady state** (payload) - but other operation states can be defined
  
- **What** does the system do most of the time?
  - Processes events of type X from device B (...*transaction T1*)
  - Produces reports of type Y (... *transaction T2*)
  - Updates GUI (... *transaction T3*)
  - Processes login’s from GUI (... *transaction T4*)
  
- **How often** does it do it?
  - Processes events of type X from device B – on avg, 3 per sec.
  - Produces reports of type Y – once per hour
  - Updates GUI – once every 30 sec
  - Processes login’s from GUI – on demand, on avg 1 per 10 min.



## **Cost Based Model**

# Performance/Capacity – 3+ way view



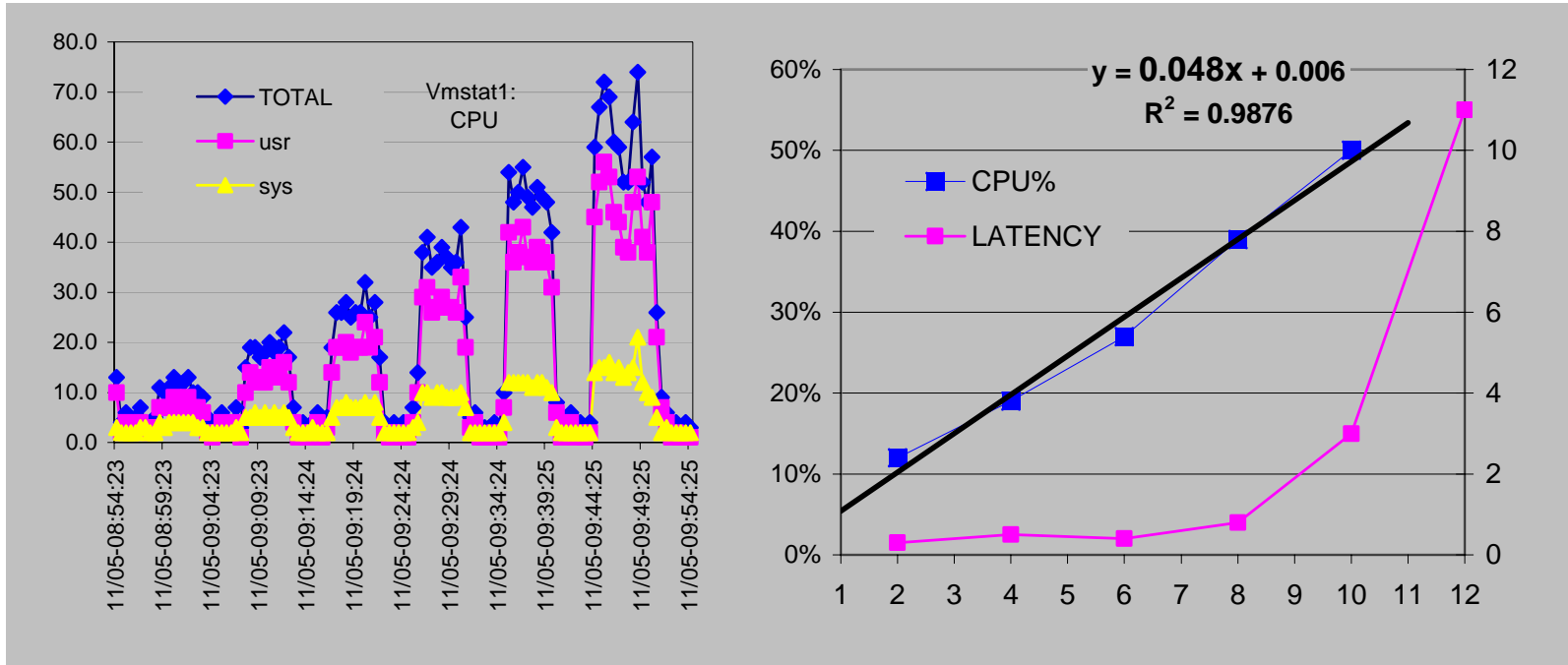
- **Behaviour – transactions and frequencies**
  - E.g. faults, 10 faults/sec
  - authentication, 1 authentication/sec
- **Costs – the price in terms of resources “paid” per transaction**
  - E.g. 2% of CPU for every fault/sec
  - E.g. 8% of CPU for every RAD Authentication per/sec
- **Resource Utilization – the price in terms of resources for the given behaviour:**
  - E.g. (2% of CPU for every fault/sec \* 10 faults/sec) + (8% of CPU for every Authentication per/sec \* 1 authentication/sec) = 28%
- **Costs can be used directly to estimate latency impact (lower bound)**
  - E.g.: 2 AA/sec -> 16% CPU impact
  - 3 sec 10 AA/sec burst with only 10% CPU available -> 24 sec latency (at least!)

# Steps to build the Cost Model



- **Behaviour**
  - Decompose system into **mutually-orthogonal performance** transactions
  - Identify expected frequencies (ranges of frequencies) per transaction
- **Costs**
  - Measure the **incremental** costs per transaction on a given h/w – one TX at a time
  - Identify boundary conditions (Cpu? Threading? Memory? Heap?)
- **Constraints**
  - Identify latency requirements and other constraints
- **Build spreadsheet model**
  - COSTS x BEHAVIOR -> REQUIREMENTS (assume linearity at first)
  - Calibrate based on combined tests

# Costs example



Resources (CPU%/Latency) Measured for 2/4/6/8/10/12 requests/sec

LATENCY = exponential after 10 RPS => **MAX RATE = 10 RPS**

- Process is NOT CPU bound (there is lots of spare CPU% @ 10 RPS)
- (In this case it is limited by the size of a JVM's heap)

Incremental CPU utilization = **4.8% of CPU per request**

- Measured on Sun N440 (4 CPUs, 1.6 GHz each) – 6400 MHz total capacity
- **COST = 4.8% \* 6400 MHz = 307.2 MHz per request**

# Constraints / Resources



- **CPU**

- Overall CPU utilization is additive per transaction (most of the time)
- If not – then transactions are not orthogonal – break down or use worst case

- **MEMORY / Java HEAPs**

- If there is no virtual memory (e.g. vxWorks) then additive; treat like CPU
- If there is virtual memory – then much trickier, no concept of X% utilization need to do direct testing.
- Heap sizes for each JVM – can be additive within each JVM

- **DISK**

- Additive, must take purging policies and retention periods into account.

- **IO**

- Additive, read/write rates are additive, but total capacity would depend on %waiting / svt and depend on manufacturer, etc. Safe limits can be tested separately

- **BW**

- Additive
- “effective” BW depends on RTT

- **Threading**

- Identify threading model for each TX – if TX is single-threaded then scale w.r.t. clock rate of the single HW Thread; if multithreaded then scale w.r.t. entire system e.g:
  - Suppose a transaction X “costs” 1000 MHz and is executed on a 32 CPU system with 500 MHz per CPU
  - If it is **single-threaded** – it will take NO LESS than **2 seconds**
  - If it is **multi-threaded** – it will take NO LESS than  $1000/(32*500) \sim$  **0.03 seconds**

- **Latency**

- For “long” transactions - measure base latency – then scale using threading. Use RTT to compute impact if relevant
- Measure MAX rate on different architectures – to calibrate

# Do we need to address everything???



- There are lots of constraints...
- May be additional constraints based on 3<sup>rd</sup> party processing
  - Addressing ALL of the in a single model may be impractical
- However – not all of them need to be addressed in every case for a useful model.  
For example:
  - vxWorks, 1 CPU, 512MB of memory, no virtual memory, pre-emptive scheduling – focus on MEM
  - Solaris, 8 CPUs, 32 h/w strands, 32G memory, - focus on CPU/Threading



# Model / Example

Workflow	Rate (/sec)
AU	5
AUPE	7
RAD	0
PAMFTP	0
PAMTEL	0
PAMFTPC	0
PAMTELC	0

**Behaviour**

**COST MODEL**

Workflow	s-MHz
AU	111
AUPE	222
GET	333
RAD	777
PAMFTP	555

**Costs**

**Resource Requirements**

Constraint Audit		Total CPU%	64.2%
Security/AM	Total Security rate greater then AM Max		
Security/PAM	OK		
Sustainability	At least one rate is not sustainable		
Alarm Rate	Composite alarm rate (INS+UPD) not sustainable		
NOS Trigger	OK		
CWD Clients	OK		
Overall CPU	Unlikely Sustainable		

Constraint	CPU	Es Disk	Nes Disk	BW
Max Utilization	75%	80%	90%	80%
Max NE Supported	623	800	4482	21836
Constraint	AEPS	RRPS		
Max Utilization	80	5		
Max NE Supported	3154	4485		
<b>Projected Max Nes</b>		<b>623</b>		



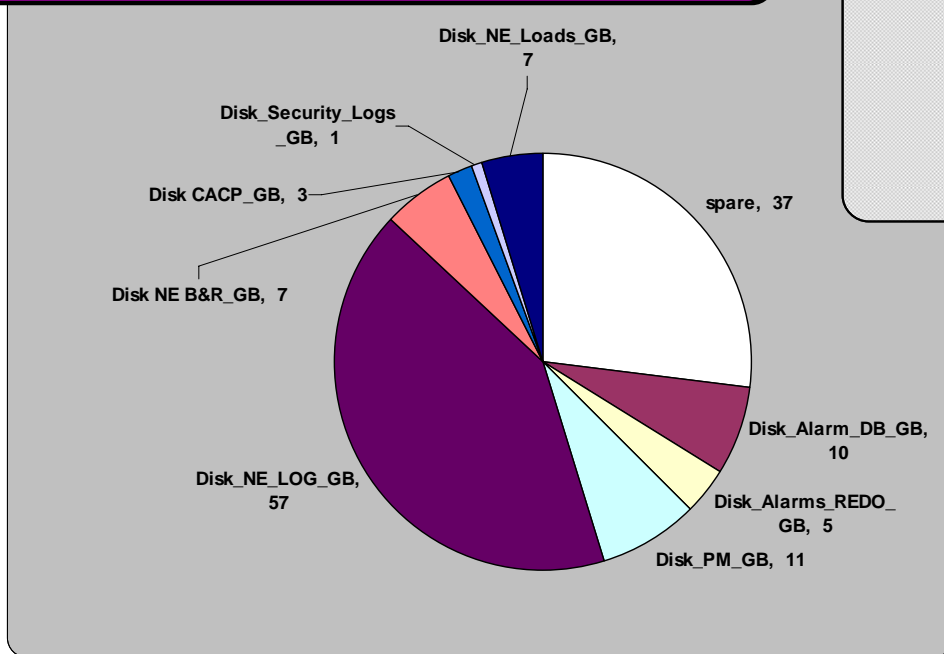
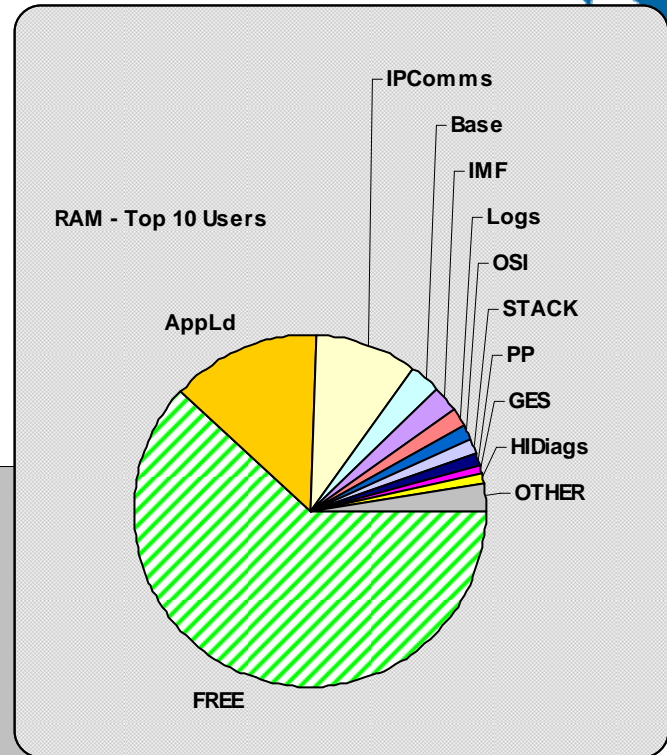
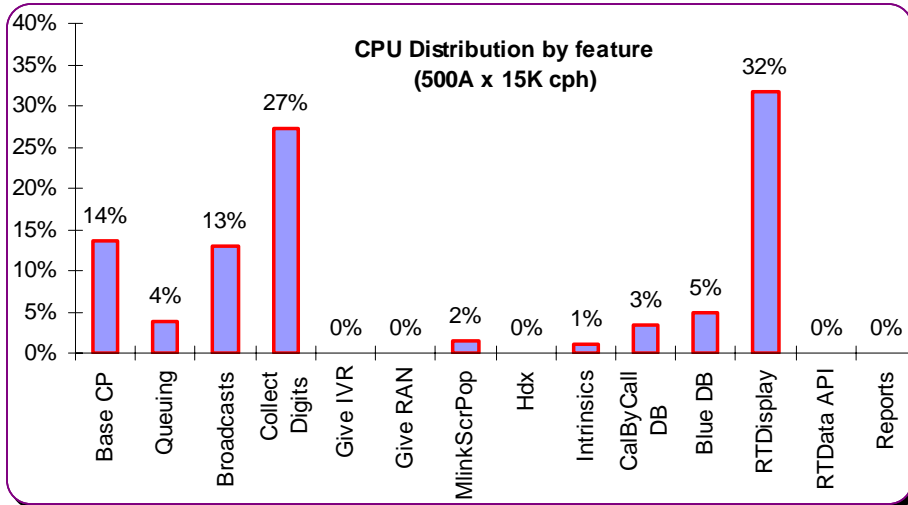
## Using model for scalability and bottleneck analysis

- Mapping between any behavior and capacity requirements
- Mapping the model to different processor architectures
- Can **Quantify** the impact of a Business request
- Can iterate over multiple “behaviors”
  - Extends “What-if” analysis
  - Enables **operating envelope visualization**
  - Enables resource **bottleneck identification**



## Using the Cost Based Model / Demo

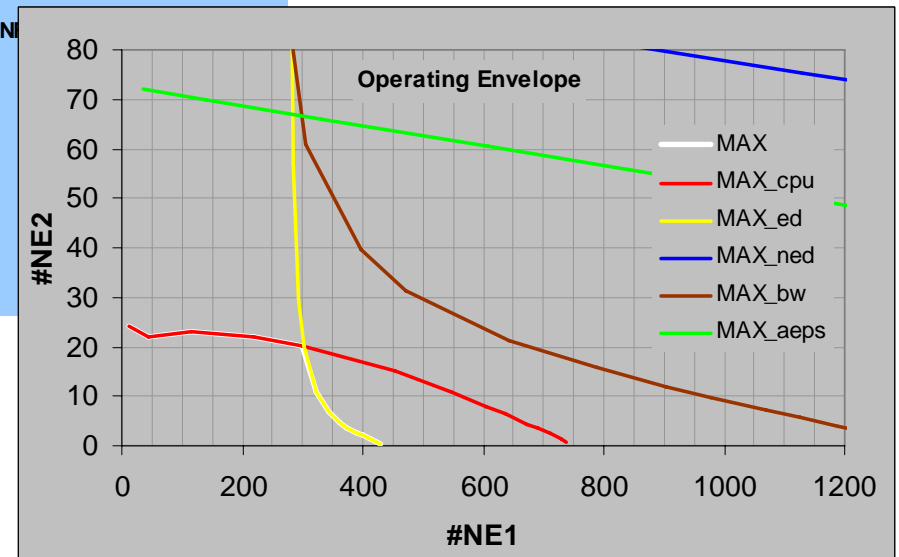
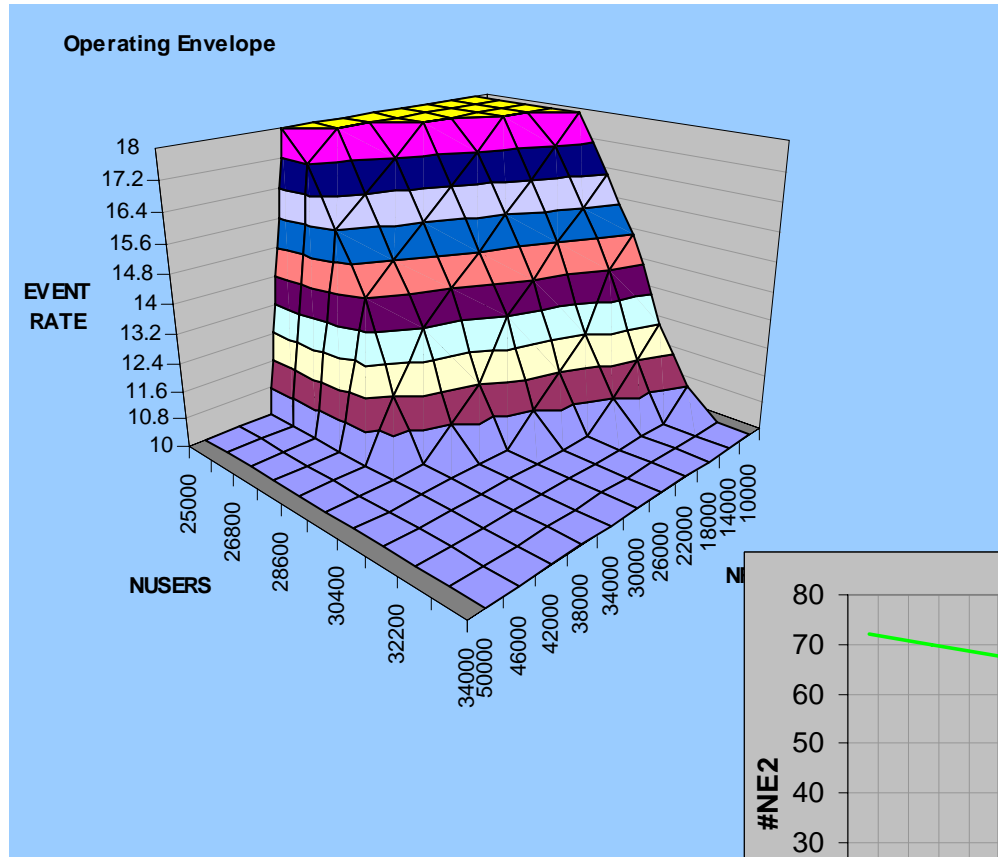
# Identifying resource allocation – by TRANSACTIONS / Applications



# Compute operating envelope



Iterate over multiple behaviours – to compute operating envelope





## Nice charts – but how accurate are they?

*Models are from God.... Data is from the Devil* (<http://www.perfdynamics.com/>)

- Initially WAY more accurate than behavior data
- Within 10% of combined metrics – for an “established” model
- Less accurate as you extrapolate further from measurements
- Model includes guesses as well as measurements
- The value is to establish patterns rather than absolute numbers.



## Projects where this was applied

- Call Centre Server (WinNT platform, C++)
- Optical Switch (VxWorks, C, Java)
- Network Management System (Solaris, Mixed, 3<sup>rd</sup> party, Java)
- Management Platform Application (Solaris, Mixed, 3<sup>rd</sup> party, Java)
- ...



**How does the Cost Model fit in the dev cycle?**

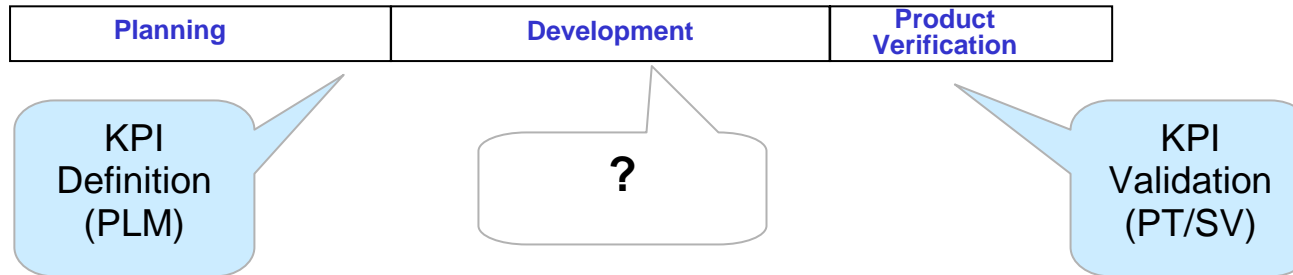


- Performance/Capacity Requirements
- Performance/Capacity Validation
- Forecasting/Estimation
- Tracking/Monitoring
- Communication

Traditionally done with “BST” / “KPI”

- Do not represent “real requirements” – usually worst impossible case
- Need LOTS of h/w and effort
- Hard to scale/communicate
- Cannot do often

# Performance/Capacity Typical Focus at the wrong places



- Limited knowledge/understanding of expected customer scenarios at planning stage (at the time of KPI commitment – specifically for platform)
- Issues discovered late – expensive to fix (=tiger teams) or over-engineering
- No ability to provide early capacity/performance estimates to customers (solutions)
- No sensitivity analysis – what is the smallest/greatest contributor to resources? Under what conditions?
- Validation involves BST type of tests; expensive; small number of scenarios (S/M/L)
- **No results portability: validation results are difficult/impossible to map/generalize to specific customer requirements**

## Performance/Capacity – Model driven

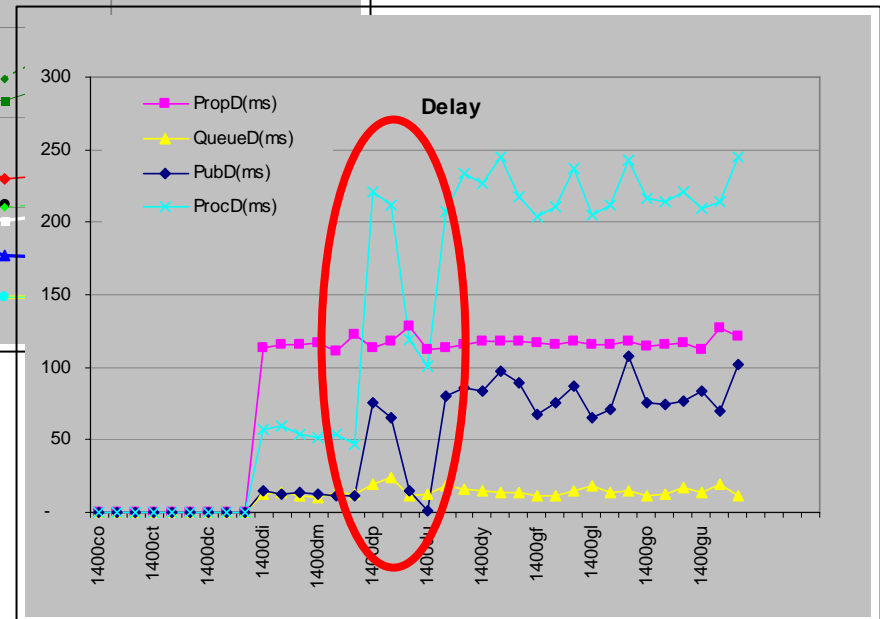
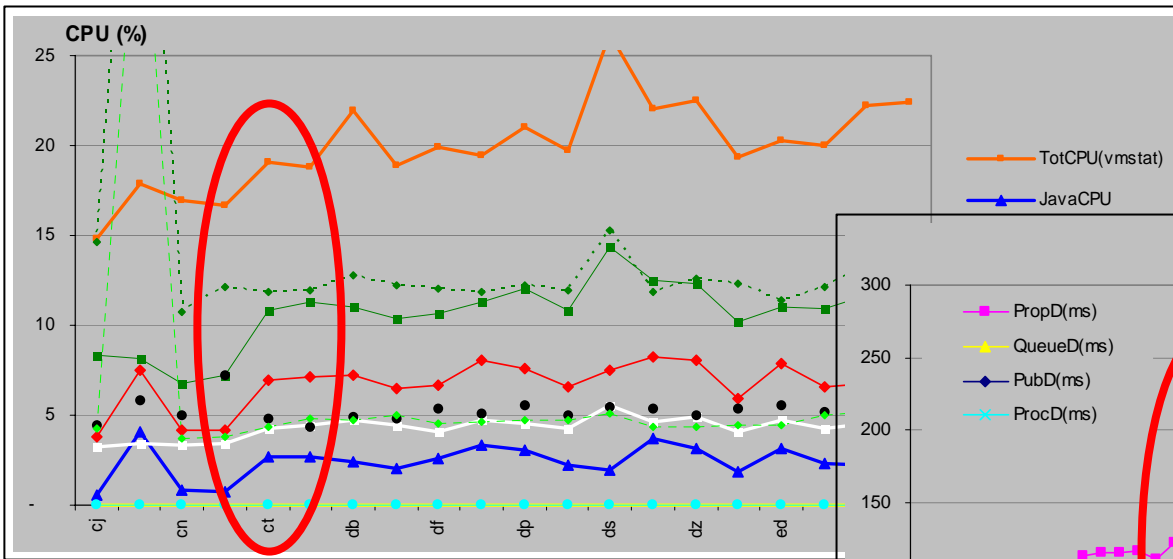


- Identify key transactions throughout the dev cycle
- Quantify behaviour in terms of transactions
- Automate test/measurements per transaction (not all, but most important)
- Automate monitor/measurement/tracking of transaction costs – as part of sanity process (weekly? Daily? – automated)
- Tight cooperation between testers/designers
- Model is developed in small steps and contains latest measurements and guesses
- Product verification
  - runs “official” test suite (automated) per transaction
  - Runs combined “BST” (multiple transactions) – to calibrate the model

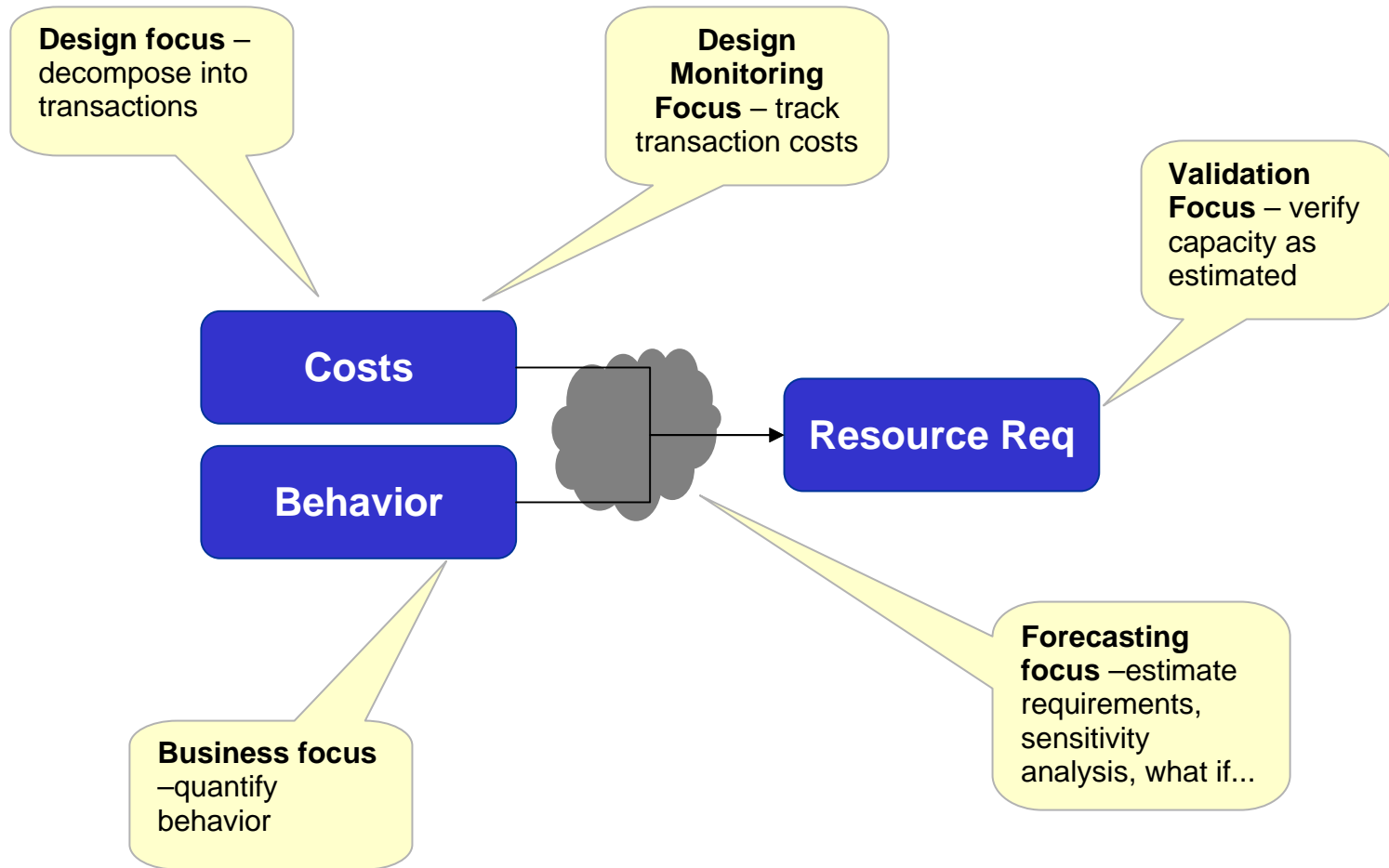
# Automated Transaction Cost Tracking



- Approximately 40 performance/Capacity CRs raised prior system verification stage
- Identification of bottlenecks (and feed-back to design)
- Continuous capacity monitoring – load-to-load view
- Other metrics collected regularly



# Cost Based Approach – Responsibilities and Roles





## **Benefits of using the model-driven performance engineering**

## Benefits – technical and others



- **Communication across groups** – everyone speaks the same language (well defined transactions/costs).
- **“De-politization” of performance eng** – can’t argue/negotiate – the numbers and trade-offs are clear.
- **Better requirements – quantifiable**, PLM/Customer can see value in quantifying behaviour
- **Documentation reduction** – engineering guides are replaced by the model; the perf related documentation can focus on improvements, etc.
- **Early problem detection** - most performance problems are discovered before the official verification cycle
- **Easy resource leak detection** – easily traceable to code changes
- **Reproducible/automated tests** – same tests scripts used by design/PV
- **Cost Reduction** – less need for BST type of tests, less effort to run PV, reduced “over-engineering”



**Things not discussed here...**

# Other issues to consider



- Tools
  - Automation (!!!!)
  - perf tracing/collection tools, transaction stat tools, transaction load, visualization, data archiving
  - native, simple, ascii + excel
- Organization (info flow/responsibilities)
  - good question, would depend on size and maturity of the project
  - Best if driven by design rather than qa/verification
  - Start slowly
- Performance Requirements definition
  - trade-offs, customer traceable, never “locked”
- Performance documentation
  - Is ENG Guide necessary?
- Using LOADS instead of transactions
  - possible if measurable directly
- Linear Regression instead of single TX testing
  - possibly for stable systems



**Questions?**



# Appendix: useful links

<http://technet.microsoft.com/en-us/commerceserver/bb608757.aspx>

– Microsoft's Transaction Cost Analysis

[www.spe-ed.com](http://www.spe-ed.com)

– Software Performance Engineering

[www.perfdynamics.com](http://www.perfdynamics.com)

– Performance Dynamics

[www.cmg.org](http://www.cmg.org)

– CMG: Computer Measurement Group

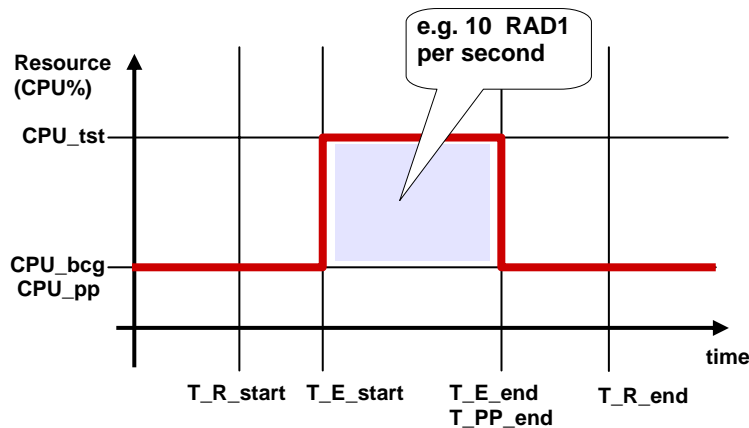


# Appendix: Good Test



- **How to measure workflow cost?**

- For each workflow , run at least 4 test cases, each corresponding to the **different rate of workflow execution**.
  - For example, for RAD1 run 4 test cases for 1, 3, 6 and 10 radius requests per second. The actual rate should result in CPU utilization between 20% and 75% for the duration of the test. If the resulting CPU is outside of these boundaries – modify the rate and rerun the test (the reason is that we want the results to represent sustainable scenarios, short term burst analysis is a separate issue).
- For each test collect and report CPU, memory and latency (as well as failure rate) **before**, **during** and **after** the test (about 5 min before, 5 min for test, 5 min after).
- Preserve all raw data (top/prstat, etc. outputs) for all tests – these may be required for further analysis.



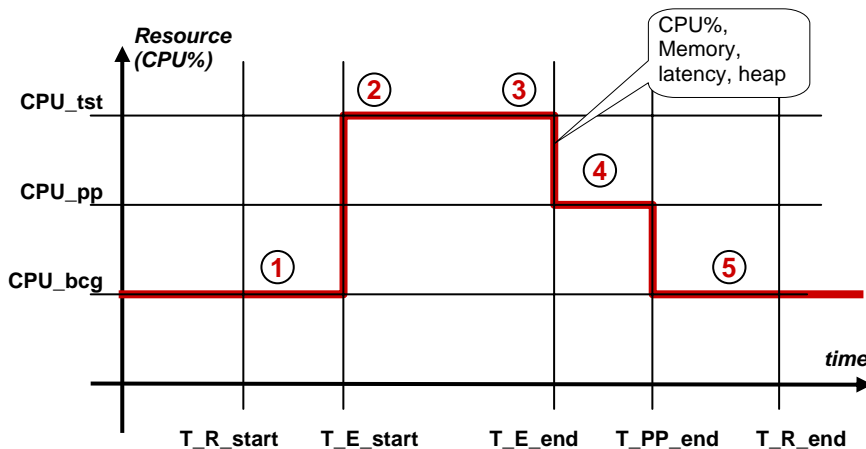
- **Automate** the test-case so that it is possible to run it after each sanity to track changes
- **Data to report/publish**
  - Marginal CPU/resource per workflow rate
  - I can help with details

# Metrics to be recorded/collected during a test



In this chart CPU is used as an example, but the same methodology applies to all resources – memory, heap, disk io, CPU, etc.

Key metrics to <u>collect</u> during a test	
T_R_start	Time data recording started
T_E_start	Time Event injection started. Assuming events are injected at a constant rate for the entire duration of the test
T_E_end	Time Event injection ended
EPS	Rate of event injection during the test (between T_E_start and T_E_end). <b>Rate is constant</b> during the test
T_PP_end	Time Post-Processing ended
T_R_end	Time Recording is ended
CPU_tst	CPU% utilization during test
CPU_pp	CPU% utilization during post-processing
CPU_bcg	Background CPU% utilization
<p>Enough samples must be collected to be able to produce a chart as below for all resources: <b>CPU</b> (total and by process) <b>Memory</b> (total and by process); <b>Heap</b> (for specific JVMs), IO, disk. The chart does not need to be included in the report but it must be available for analysis.</p> <p>Application should also monitor/record its <b>Latency</b> and <b>Failure rate</b> – this is application specific, but it should be collected/recorded in such a way that it can be correlated with the resource chart. <b>Avg latency</b> and <b>Avg Failure</b> rate during the test is <b>NOT sufficient</b> – it does not show the trends.</p>	



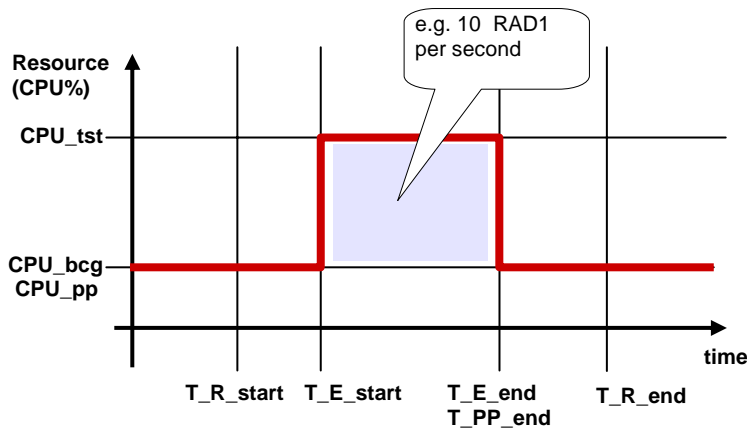
Derived Metrics – to be <u>included</u> in performance report	
mCPU_tst	$CPU\_tst - CPU\_bcg$ (marginal test cost)
mCPU_pp	$CPU\_pp - CPU\_bcg$ (marginal post-processing cost. If post-processing is not 0 then the <b>EPS rate is not sustainable</b> over long time)
mT_tst	$T\_E\_end - T\_E\_start$ (duration of the test/injection)
mT_pp	$T\_PP\_end - T\_E\_end$ (duration of post-processing – ideally this should be 0)
<p>Ideally the resource utilization during the test is “flat” and returns to pre-test levels after the test is completed. To verify this compare the measurements before/after tests (points 1 and 5 on the chart) and at the beginning and at the end of the test (points 2 and 3 on the chart)</p>	
dCPU_bcg	$CPU\_5 - CPU\_1$ (if >0 then resource is not fully released after test)
dCPU_tst	$CPU\_3 - CPU\_2$ (if >0 then there may be a resource “leak” during the test)

TOOLS	
<p>Any tool can be used to collect the metrics – as long as it can collect multiple periodic samples. As a rule of thumb collect about 100 samples for the pre-test idle, 200 samples per test and another 100 after test. If you collect a sample once per 10 sec the overall test duration will be a bit more then 1 hrs. The following are examples:</p>	
prstat -n700	for individual process CPU and memory (-n700 to collect up to 700 processes regardless of their cpu% - to make sure you get a complete memory picture)
TOP / ps	These can be used instead of prstat
vmstat	For global memory/cpu/kernel CPU
lsof	If you suspect io issues
jstat -gc	For individual JVM heap/GC metrics; look at OC and OU parameters.

# Perfect Case - SUSTAINABLE



No Post processing	$mT_{pp} = 0$
Resource utilization is flat during test	$dCPU_{tst} = dMEM_{tst} = 0$
All resources recover completely	$dCPU_{bcg} = dMEM_{bcg} = 0$
CPU% per 1 EPS	$mCPU_{tst}/EPS$
Memory specific	<p>Process /system memory and heap may grow – if logically the events create objects that should stay in memory (e.g. you are doing discovery and are adding new data structures).</p> <p>Memory/heap may also grow initially after <math>T_{E\_start}</math> but should stabilize before <math>T_{E\_end}</math> – this represents build-up of working sets. In this case memory may not be released fully upon completion of the test. In that case run the test again – if the memory keeps on increasing this may indicate a leak.</p>



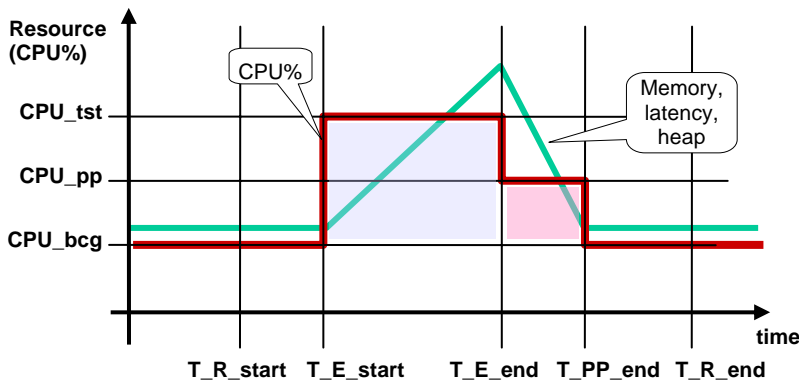
The overall CPU used in this case is the area under the utilization curve – the blue square

**Make sure that latency/success rate during the test is acceptable.** It is possible that the resource profile may look perfect but the events may be rejected due to the lack of resources.

# Post-Processing – NOT SUSTAINABLE / BURST test



<p>Post processing detected</p>	<p><math>mT_{pp} &gt; 0</math> means that the system was not able to process events at the rate they arrive, this can be due to CPU utilization, or due to threading or other resource contention. In this case you may see that</p> <ul style="list-style-type: none"> <li>– the latency is continuously increasing between <math>T_{E\_Start}</math> and <math>T_{E\_end}</math></li> <li>– Memory (or old heap partition of a JVM) is continuously increasing between <math>T_{E\_start}</math> and <math>T_{E\_end}</math> and then starts decreasing during post-processing. This is because the events that cannot be processed in time must be stored somewhere. (see <b>green</b> line on the chart).</li> <li>– The failure rate may increase towards the end of the test</li> </ul>
<p>Load unsustainable</p>	<p>This load is unsustainable over the long period of time - can be hours or days – but the system/process will either run out of memory or will be forced to drop outstanding events</p>
<p>May be acceptable for short burst/peaks</p>	<p>Although this rate is unsustainable for long time it may be acceptable for short bursts / peaks. The duration of post-processing and the rate of growth of the bounding factor (memory/heap/threads) will help determine the max duration of the burst.</p>
<p>CPU% per 1 EPS</p>	<p>The overall CPU used in this case is the area under the utilization curve – the blue square and the pink square. It is possible to predict how much CPU would be used by 1 EPS if the <b>bottleneck is removed</b> (e.g. if threading is a bottleneck and we add more threads):</p> $(mCPU_{tst} + mCPU_{pp} * mT_{pp} / mT_{tst}) / EPS$



In case of post processing it is important to determine what is the boundary condition:

if CPU utilization during test is 90% or more then it is likely that we are bounded by CPU

If memory/heap of component A grows and component A has to pass events to component B, then B may be a bottleneck

If component B uses 1 full CPU (25%) then it is likely single-threaded and threading is the issue

If component B does disk io, or other type of access that requires waiting then this can be the bottleneck