Your software can run faster

Revving up

Johan Peeters
with thanks to Guy Thijs

Learning objectives

- Demystify performance engineering
- Conceptual framework for understanding performance issues
- Design cost-effective performance tests
- Find and resolve hot spots

Questions to keep in mind

- How does performance engineering fit into development process?
- When is it better to buy more hardware than to optimize the software?
- When do performance improvements cease to be cost-effective?
- When should performance findings be generalized into architectural guidelines?
- When do performance insights warrant architectural refactoring?

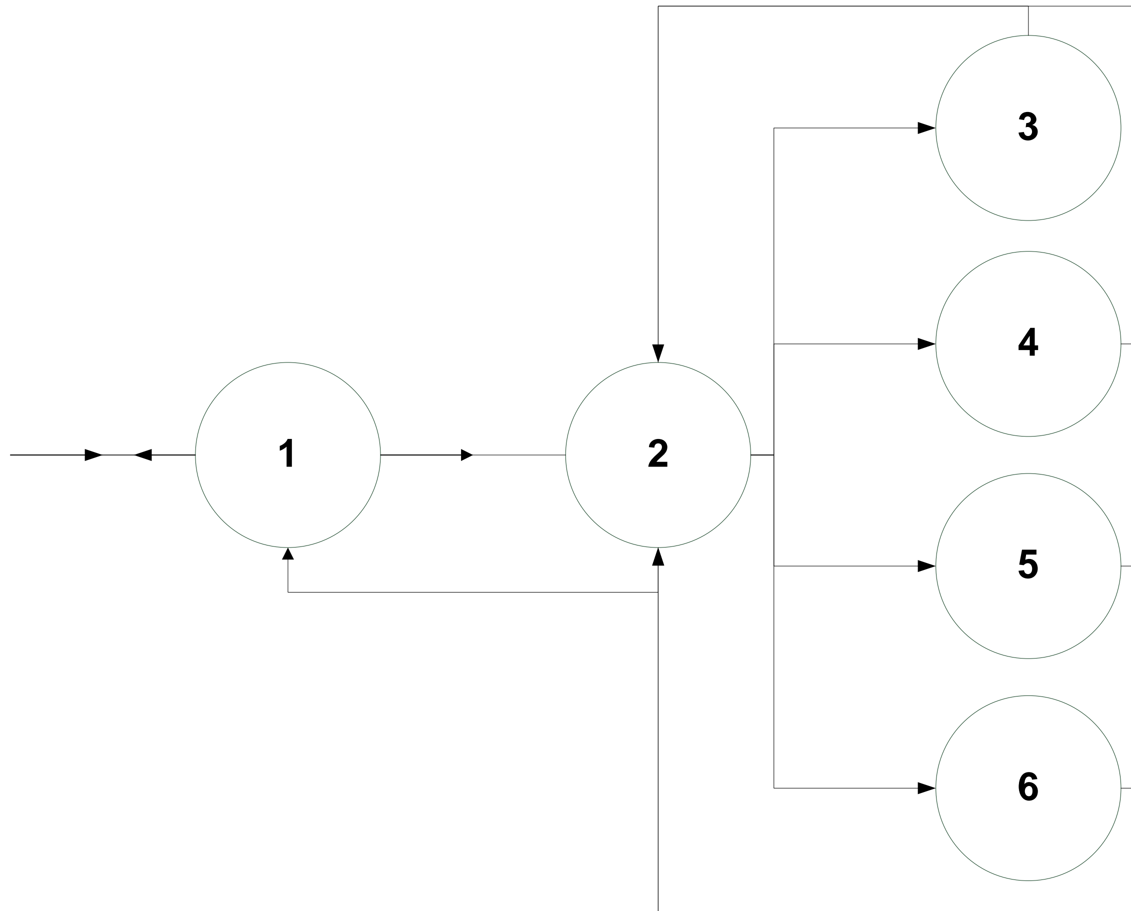
Why is throughput so important?

- Higher throughput means
  - Less hardware and fewer license fees for constant business, or
  - More business with constant hardware
- Correlation between throughput and response time growth with load
  - Details coming up

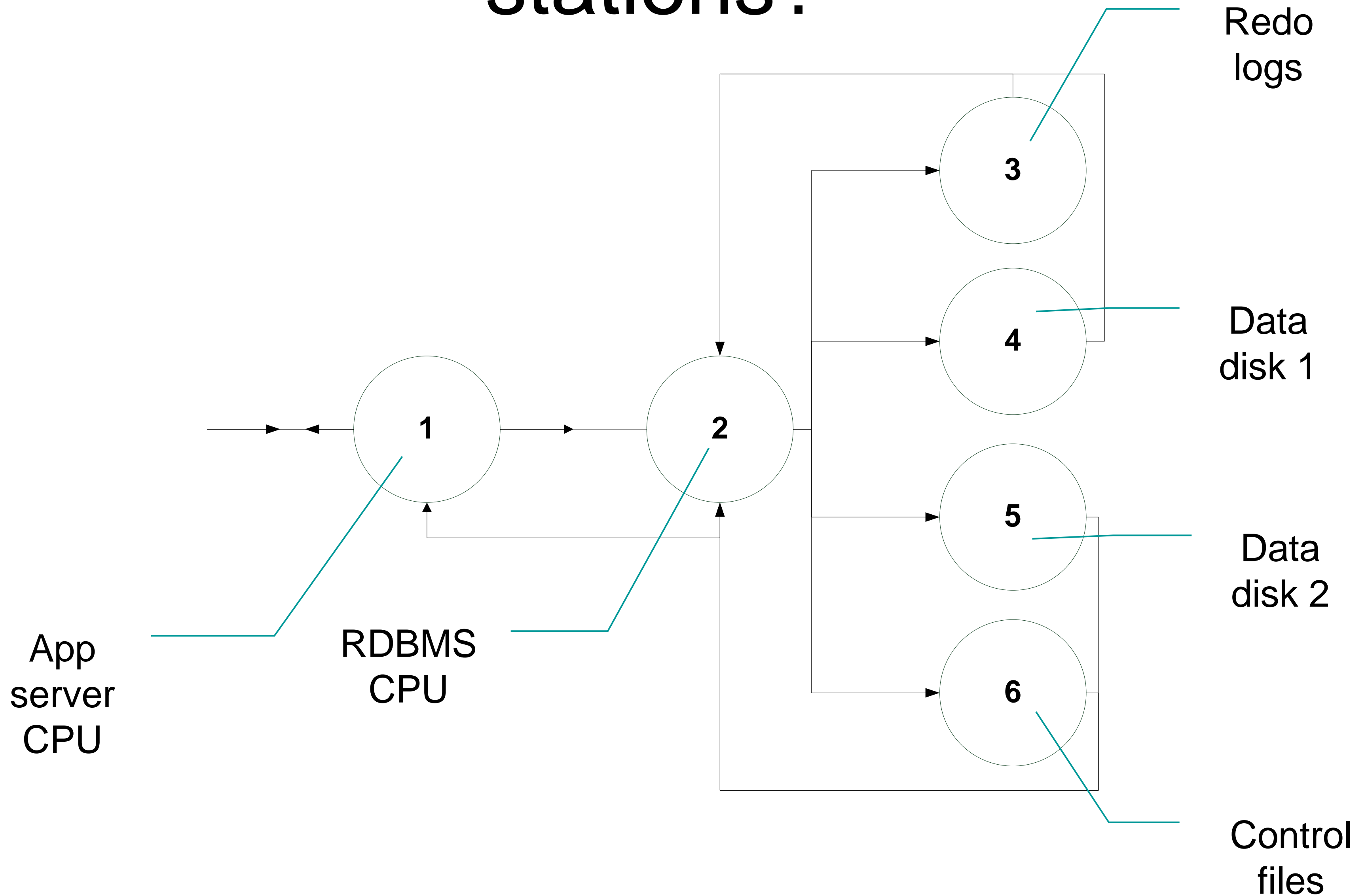
Throughput is not always the right metric, e.g.

- constraints on batch processing:
  - Duration of night window
  - Utilization
- APIs
  - Good API throughput ≠ good caller throughput
- web application usability
  - Typically only 10-20% of time spent requesting HTML
  - See *High Performance Web Sites* by Steve Souders

System model I



What are the relevant service stations?

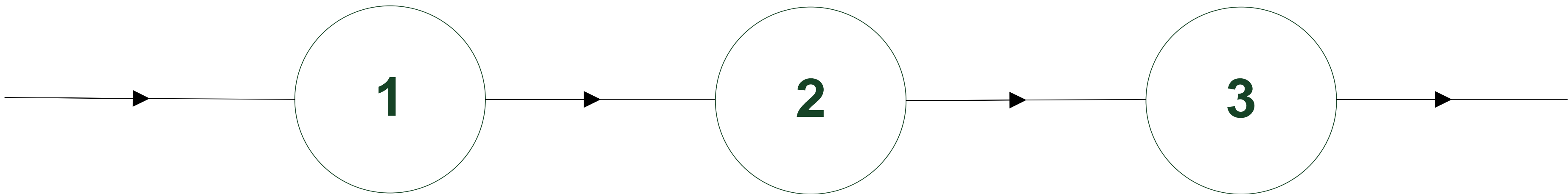


Service demand

$$D_i = \sum_{j=1}^{V_i} S_{ij}$$

$$D_i = V_i \times S_i$$

System model I’



Precisely 1 request in the system

- Good lower bound for response time?

$$\sum_i D_i$$

- Good upper bound for throughput?

$$\frac{1}{\sum_i D_i}$$

N beautifully choreographed concurrent requests

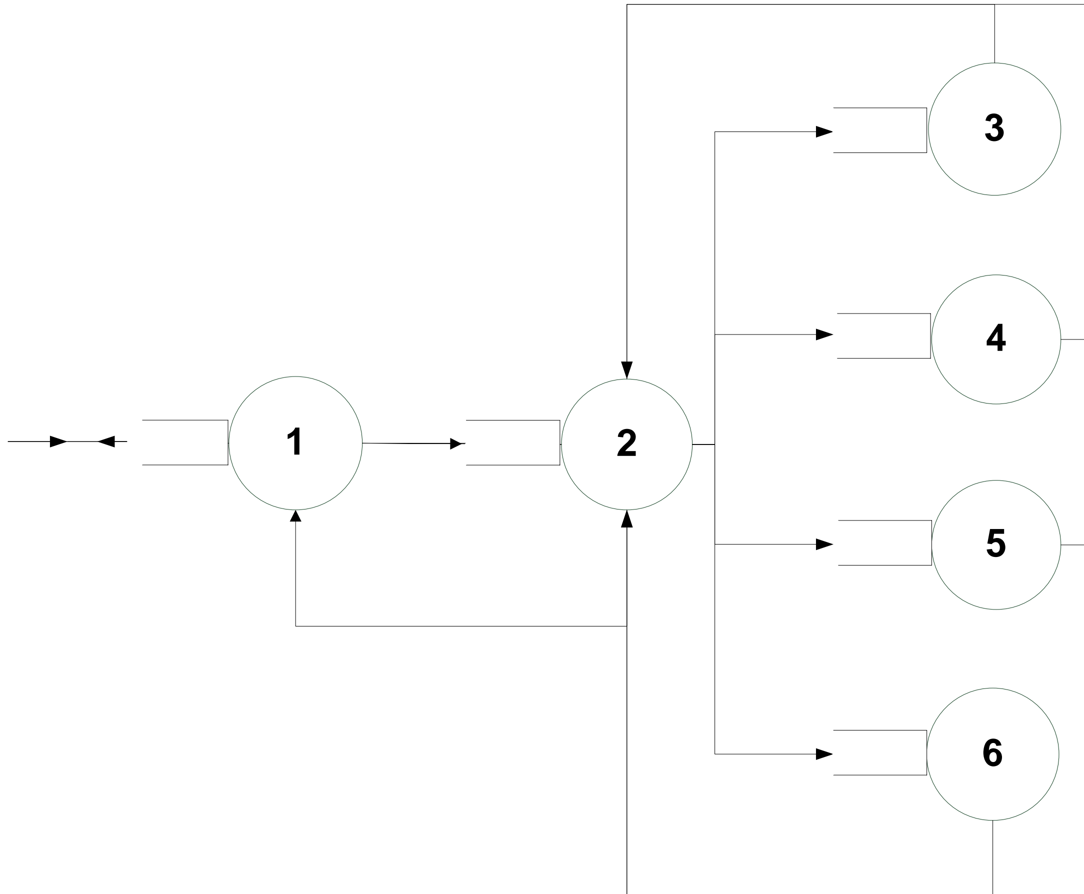
- Good lower bound for response time?

$$\sum_i D_i$$

- Good upper bound for throughput?

$$\frac{N}{\sum_i D_i}$$

System model II

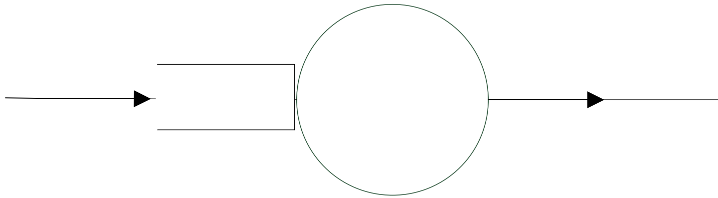


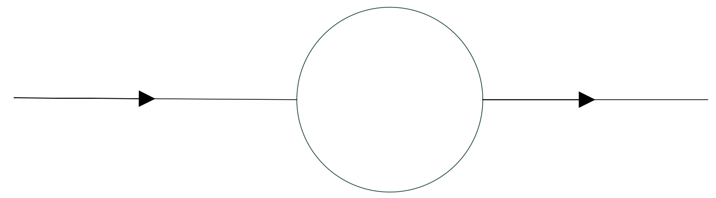
Competition for resources

Good upper bound for throughput?

$$\frac{1}{\max(D_i)}$$

Little's law





if system in steady state

then

$$L = \lambda W$$

Applications of Little's law

$$N = X \times R$$

where

- N = number of requests being serviced
- X = throughput
- R = response time

Competition for resources

Good lower bound for response time?

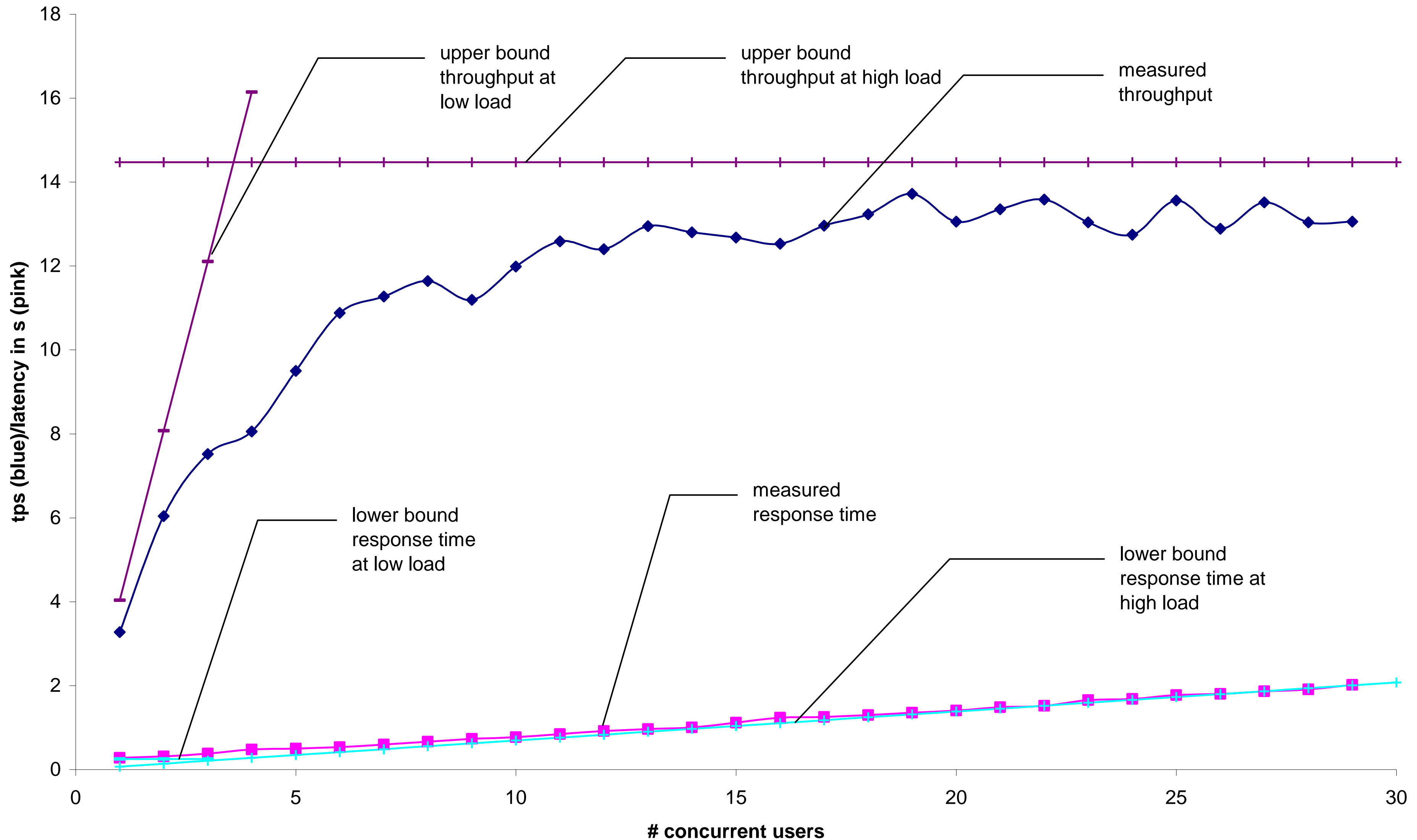
by Little's law:

$$N \times \max(D_i)$$

What does it mean?

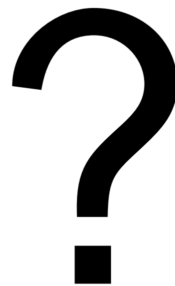
- Optimizing resource utilization other than the bottleneck’s neither improves throughput nor response time
- Response time always rises with load
- Maximizing throughput also limits response time growth

Does the theory fit the practice?



But how can we know...

- response time
- throughput
- service demand



Test cycle

initialize

generate load

measure

monitor

analyze

Load

- Needed for meaningful measurements
  - Response time varies with load
  - Capacity throughput is only reached with sufficient load
- Definition of/metric for load?

N

Alternative load models

- Natural model from production system execution traces
- Artificial model
  - Representative
  - Ease of load generation
  - Ease of analysis

Our choice – a hybrid

1 representative scenario: cash deposits

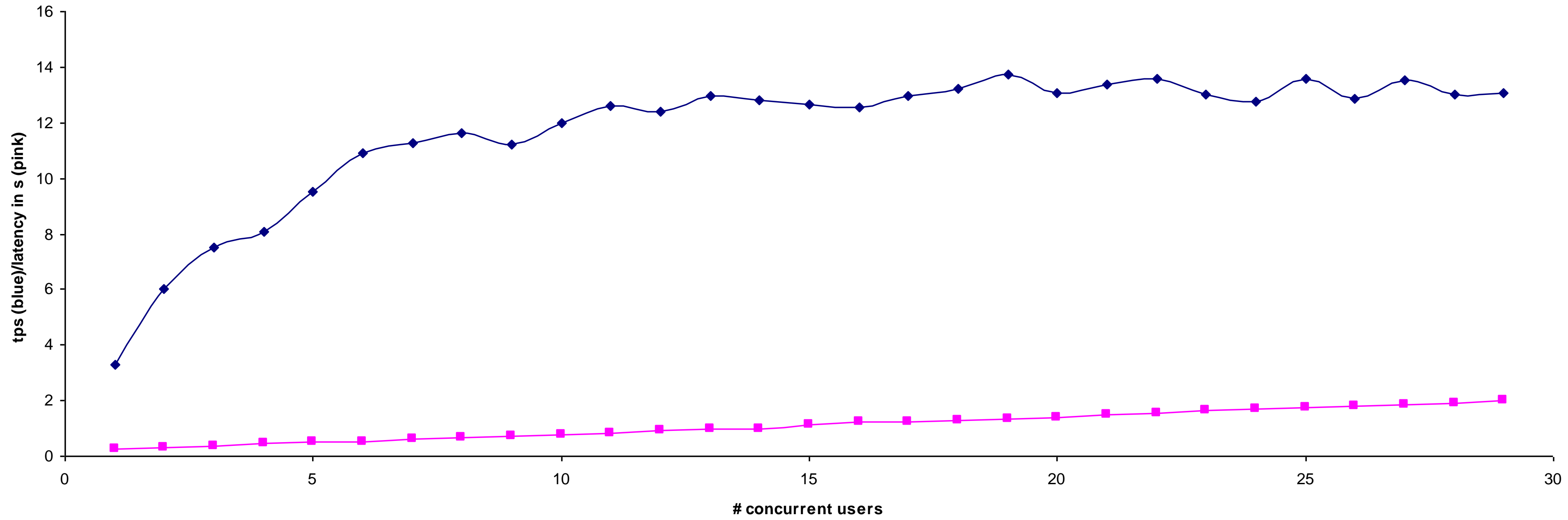
- Over ½ of BCD transactions are cash deposits
- Significant improvements for cash deposit scenario should make a significant overall improvement
- Apply what has been learned with the cash deposit scenario to other scenario's as well

Load generation technology

- Tools
  - LoadRunner
  - JMeter
  - OpenSTA
- ‘Virtual users’ = threads
- Vary requests
- Rampup

Our measurement strategy

- Aim: understand how response time and throughput relate to load
- Load is number of concurrent requests
- Therefore:
  - Each thread ensures that the system is always servicing precisely 1 request on its behalf
  - By ramping up number of concurrent requests, measurements are obtained for a range of load conditions in single run



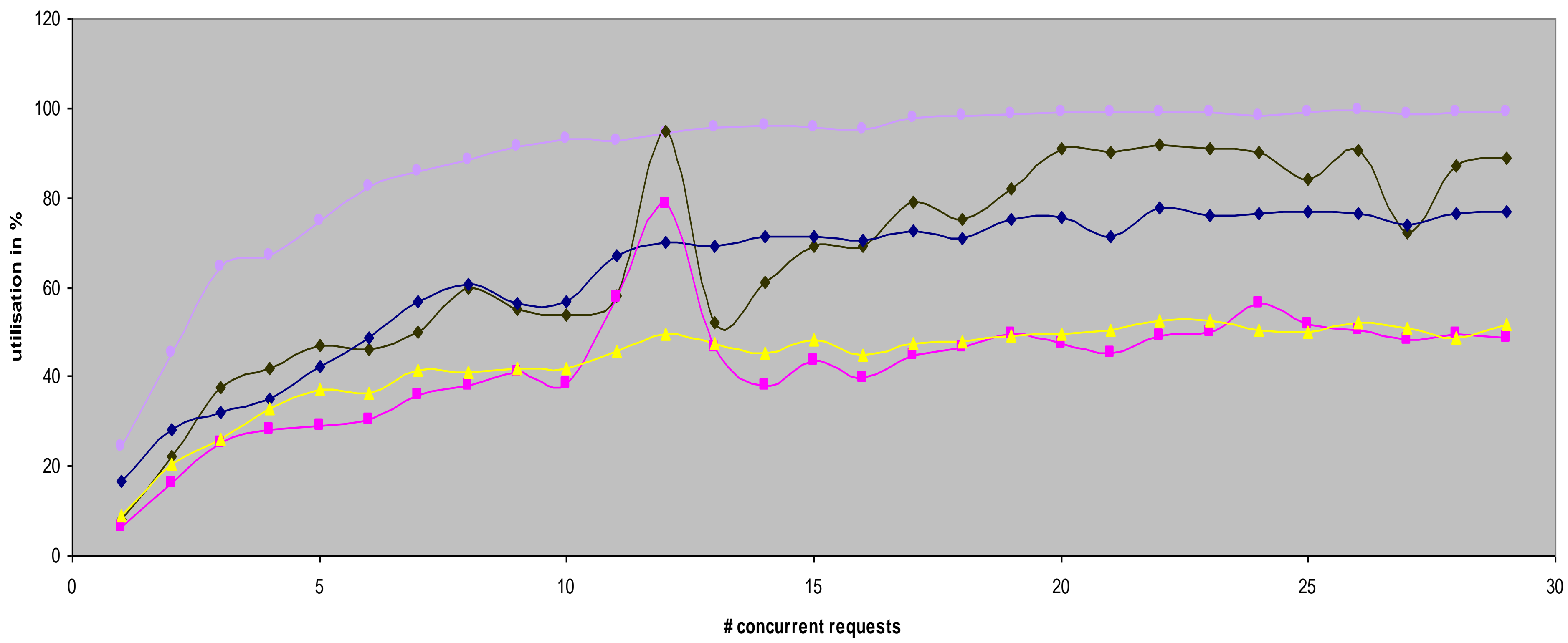
tps (blue)/latency in s (pink)
0
2
4
6
8
10
12
14
16
0
5
10
15
20
25
30
concurrent users

Monitoring

- Based on Unix utilities such as `ps`, `prstat` and `iostat`
- Capture timestamp as well as resource utilization
- Start monitoring scripts on all servers in the system before generating load
- Write monitoring data to file
- Retrieve monitoring data after test run is complete for analysis

Analysis monitoring data

- Correlate timestamps of load test tool logs with monitoring log timestamps
- Obtain utilization statistics for load levels


120
100
80
60
40
20
0
utilisation in %
0
5
10
15
20
25
30
concurrent requests
WebSphere CPU service demand
second data file metadevice on oracle
Oracle CPU utilisation
first data file metadevice on Oracle server
redo log disk on oracle

Utilization

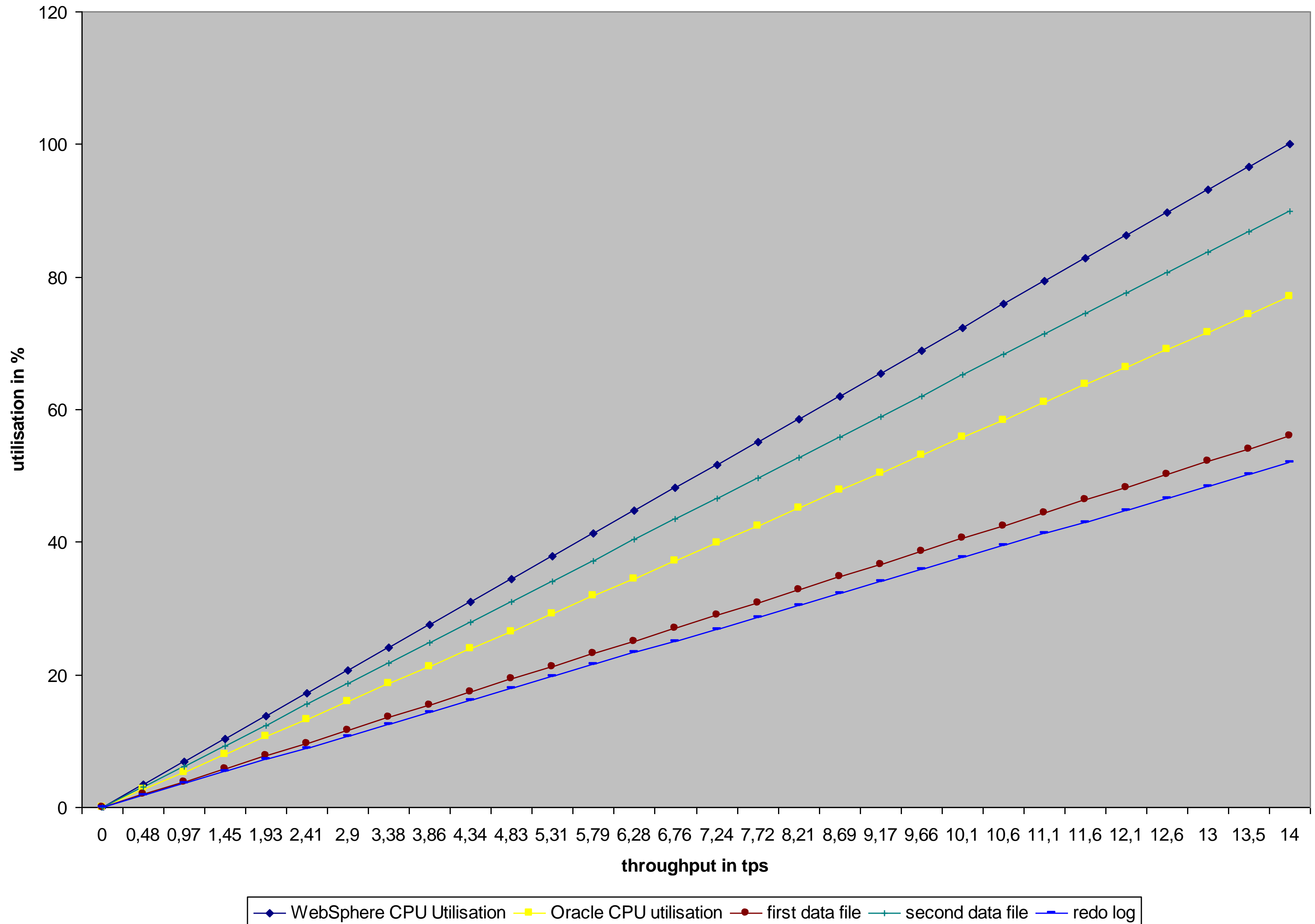
$$D_i = \frac{B_i}{C}$$

$$U_i = \frac{B_i}{T}$$

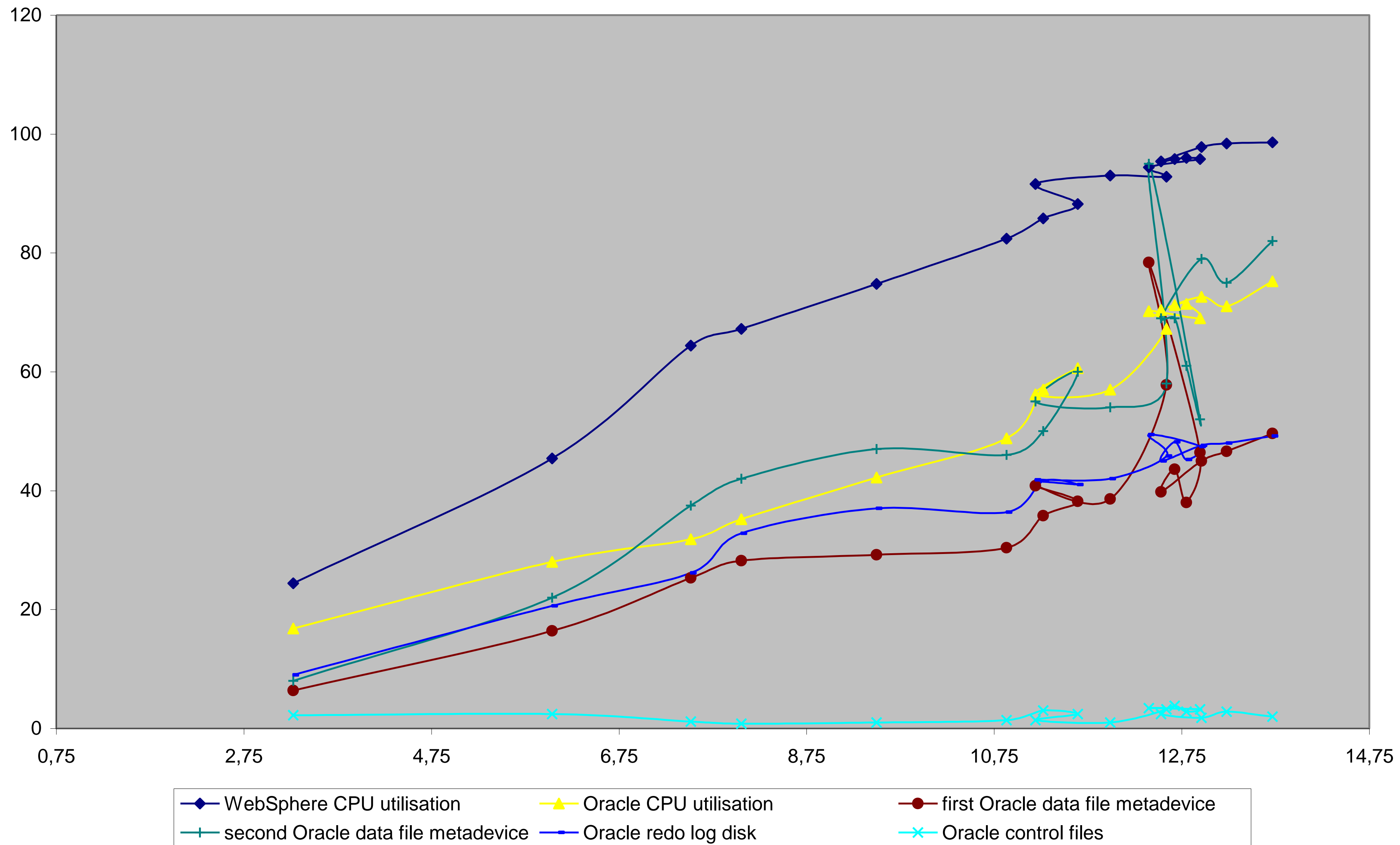
$$U_i = \frac{B_i}{C} \times \frac{C}{T}$$

$$U_i = D_i \times X$$

In theory

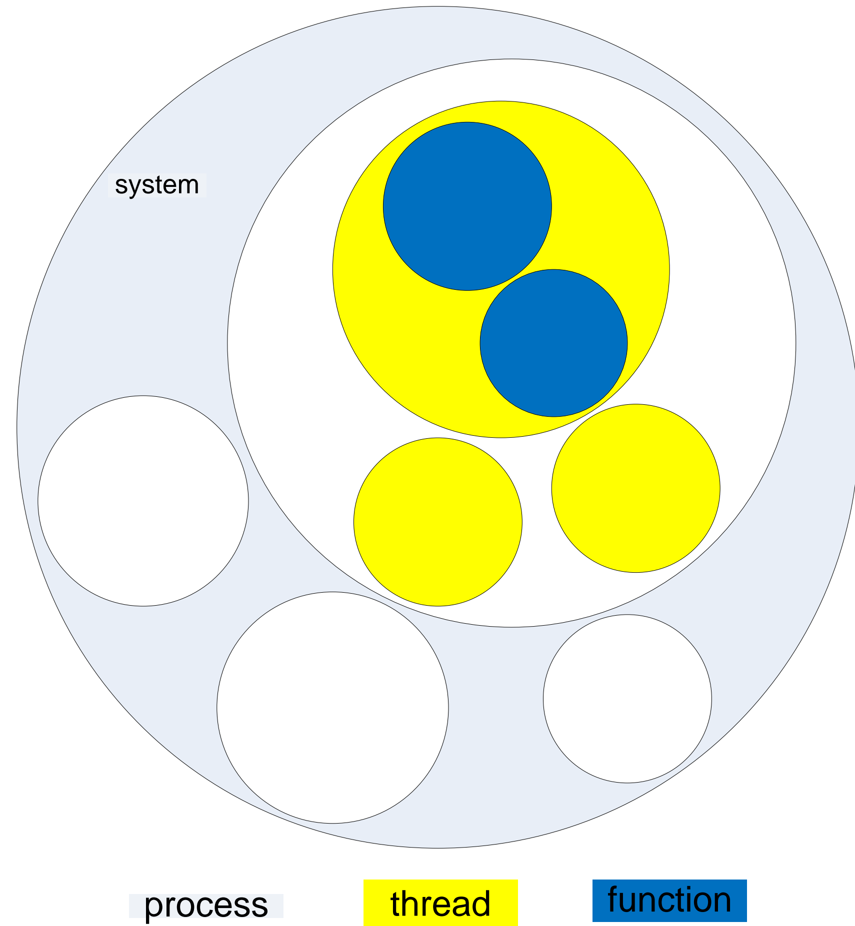


In practice



Example 1: application server CPU is limiting factor

Drilling down



Profiling

n0fe6 called as a library 6.jps - JProfiler 5.2.1

Session View Profiling Go To Window Help

Start Center | Stop | Freeze | Reload | Save Snapshot | Export | Run GC | Add Bookmark | Record Memory | Record CPU | Session Settings | View Settings | Help | Show Graph

Memory Views
Heap Walker
CPU Views
Thread Views
VM Telemetry Views

Thread selection: WebContainer : 0 [main]
Thread status: Runnable
Aggregation level: Methods
Hot spot type: Method calls (add filtered classes to calling class)

Hot spot	Inherent time	Average Time	Invocations
egl.core.StrLib_Lib.clip	46.952 ms (14 %)	939 µs	49.996
com.kbc.n01.cmn.pvt.n01_005fCst_005f0394_005fUQ.<init>	10.077 ms (3 %)	2.255 µs	4.467
com.kbc.n01.itf.svc.in0s03_005fR_005f48.<init>	9.800 ms (3 %)	17.563 µs	558
com.kbc.n01.itf.pvt.in0fe6a.<init>	9.607 ms (2 %)	17.156 µs	560
com.ibm.ws.util.ThreadPool$Worker.run	7.348 ms (2 %)	7.348 ms	1
com.kbc.n01.cmn.pvt.n01_005fPmtr.<init>	7.168 ms (2 %)	3.666 µs	1.955
com.kbc.n01.itf.pvt.in0fe6b.<init>	6.496 ms (1 %)	23.201 µs	280
egl.ui.text.ConverseVar_Lib.initSavedSysVars	6.154 ms (1 %)	328 µs	18.710
com.kbc.n01.cmn.pvt.n01_005fCmn_005fCnst.<init>	4.862 ms (1 %)	322 µs	15.080
com.kbc.n01.ftn.pvt.n0f68.$func_z040_005fGet_005fCty_005fId	4.506 ms (1 %)	8.076 µs	558
com.kbc.logging.log4j.FileAppender.subAppend	3.660 ms (1 %)	1.639 µs	2.233
com.kbc.n01.itf.pvt.in0f73.<init>	3.595 ms (1 %)	2.147 µs	1.674
com.kbc.n01.ftn.pvt.n0f47.$func_z050_005fPcs_005fTrx	3.491 ms (1 %)	12.468 µs	280
egl.core.SqlData.<init>	3.264 ms (1 %)	169 µs	19.268
com.kbc.n01.itf.pvt.in0fe6a.$appendTo_ipt_005fZn__data__smt_005fLst__ohr	3.241 ms (0 %)	5.787 µs	560
com.kbc.n01.itf.pvt.in0fe6a.$appendTo_ipt_005fZn__data__smt_005fLst__am	3.065 ms (0 %)	5.474 µs	560
com.kbc.y41.cmn.pub.y41_005fUcd_005fWlog.<init>	2.711 ms (0 %)	154 µs	17.594
com.kbc.n01.itf.svc.in0s03.<init>	2.289 ms (0 %)	1.367 µs	1.674
com.kbc.n01.itf.pvt.in0f48.<init>	2.229 ms (0 %)	1.997 µs	1.116
com.kbc.n01.itf.pvt.in0f37.$appendTo_ipt_005fZn__data__trx_005fLst__dn...	2.168 ms (0 %)	129 µs	16.770
com.kbc.n01.ftn.pvt.n0f48.$func_z080_005fGet_005fFee	2.166 ms (0 %)	3.882 µs	558
com.kbc.n01.itf.pvt.in0f37.$appendTo_ipt_005fZn__data__trx_005fLst__dn...	2.156 ms (0 %)	128 µs	16.770
com.kbc.y41.rt.blue.EGLCalledProgram._createRunUnit	2.131 ms (0 %)	3.818 µs	558
com.kbc.n01.itf.pvt.in0f37.$appendTo_ipt_005fZn__data__trx_005fLst__dn...	2.124 ms (0 %)	126 µs	16.770
com.kbc.n01.ftn.pvt.n0fe6a.$func_apd_005fTx	2.110 ms (0 %)	73 µs	28.560
com.kbc.n01.itf.pvt.in0f65.<init>	2.091 ms (0 %)	1.874 µs	1.116
com.kbc.n01.itf.pvt.in0f72.<init>	1.946 ms (0 %)	1.393 µs	1.397
egl.core.SysVar_Lib.<init>	1.846 ms (0 %)	98 µs	18.710
com.kbc.y41.cmn.pub.y41_005fUcd_005fWgen.<init>	1.816 ms (0 %)	103 µs	17.594
com.kbc.n01.svc.n0p03c.$func_z010_005fInit_005fPgm	1.805 ms (0 %)	3.235 µs	558
com.kbc.n01.ftn.pvt.n0f48.$func_z140_005fGet_005fBnk_005fCd_005fCz	1.779 ms (0 %)	3.189 µs	558
com.kbc.n01.ftn.pvt.n0f48.$func_z120_005fChk_005fRst	1.732 ms (0 %)	3.104 µs	558
com.kbc.n01.ftn.pvt.n0f37.$func_z010_005fInit_005fPgm	1.723 ms (0 %)	6.154 µs	280
com.kbc.n01.itf.pvt.in0f65.$appendTo_opt_005fZn__hdr__msg_005fAtb_00...	1.706 ms (0 %)	1.529 µs	1.116
com.kbc.n01.cmn.pvt.n01_005fCst_005f0565_005fUQ.<init>	1.701 ms (0 %)	1.219 µs	1.395

View Filters: Reset View Filters

Call Tree | Hot Spots | Call Graph | Call Tracer

15:36 Snapshot

Profiling

n0fe6 called as a library 6.jps - JProfiler 5.2.1

Session | View | Profiling | Go To | Window | Help

Start Center | Stop | Freeze | Reload | Save Snapshot | Export | Run GC | Add Bookmark | Record Memory | Record CPU | Session Settings | View Settings | Help | Show Graph

Memory Views | Heap Walker | CPU Views | Thread Views | VM Telemetry Views

Thread selection: WebContainer : 0 [main] | Thread status: Runnable

Aggregation level: Methods

Hot spot type: Method calls (add filtered classes to calling class)

Hot spot	Inherent time	Average Time	Invocations
egl.core.StrLib_Lib.clip	46.952 ms (14 %)	939 µs	49.996
13,2% - 42.789 ms - 31.274 hot spot inv. com.ibm.javart.operations.Assign.run			
0,7% - 2.195 ms - 6.138 hot spot inv. com.kbc.n01.ftn.pvt.n0f38.$func_c010_005fGet_005fAll_005fLmt_005fFor_005fCty			
0,1% - 440 ms - 1.674 hot spot inv. com.kbc.n01.ftn.pvt.n0f74.$func_z050_005fChg_005fCnt			
0,1% - 398 ms - 1.400 hot spot inv. com.kbc.n01.ftn.pvt.n0fc6.$func_is_005fUcd_005fIpt_005fPrm_005fEmpt			
0,1% - 238 ms - 3.080 hot spot inv. com.ibm.javart.sql.SqlHostVars.setUnicodeClipped			
0,1% - 176 ms - 558 hot spot inv. com.kbc.n01.ftn.pvt.n0f38.$func_is_005fUcd_005fIpt_005fPrm_005fEmpt			
0,1% - 171 ms - 558 hot spot inv. com.kbc.n01.ftn.pvt.n0f65a.$func_is_005fUcd_005fIpt_005fPrm_005fEmpt			
0,1% - 167 ms - 558 hot spot inv. com.kbc.n01.ftn.pvt.n0f53.$func_is_005fUcd_005fIpt_005fPrm_005fEmpt			
0,1% - 166 ms - 558 hot spot inv. com.kbc.n01.ftn.pvt.n0f65.$func_is_005fUcd_005fIpt_005fPrm_005fEmpt			
0,0% - 69.832 µs - 280 hot spot inv. com.kbc.n01.ftn.pvt.n0f47.$func_is_005fUcd_005fIpt_005fPrm_005fEmpt			
0,0% - 68.883 µs - 2.520 hot spot inv. com.kbc.n01.ftn.pvt.n0fc9.$func_z040_005fGet_005fStr			
0,0% - 56.342 µs - 558 hot spot inv. com.kbc.n01.ftn.pvt.n0f30.$func_z060_005fChk_005fTrx_005fType_005fVs_005fAcc_005fType_005fPfl			
0,0% - 10.132 µs - 560 hot spot inv. com.kbc.n01.ftn.pvt.n0fc6.$func_b030_005fVl_005fFrom_005fPcs_005fTrx			
0,0% - 2.902 µs - 280 hot spot inv. com.kbc.n01.ftn.pvt.n0fe3.$func_z050_005fStr_005fMain_005fTrx			
com.kbc.n01.cmn.pvt.n01_005fCst_005f0394_005fUQ.<init>	10.077 ms (3 %)	2.255 µs	4.467
com.kbc.n01.itf.svc.in0s03_005fR_005f48.<init>	9.800 ms (3 %)	17.563 µs	558
com.kbc.n01.itf.pvt.in0fe6a.<init>	9.607 ms (2 %)	17.156 µs	560
com.ibm.ws.util.ThreadPool$Worker.run	7.348 ms (2 %)	7.348 ms	1
com.kbc.n01.cmn.pvt.n01_005fPmtr.<init>	7.168 ms (2 %)	3.666 µs	1.955
com.kbc.n01.itf.pvt.in0fe6b.<init>	6.496 ms (1 %)	23.201 µs	280
egl.ui.text.ConverseVar_Lib.initSavedSysVars	6.154 ms (1 %)	328 µs	18.710
com.kbc.n01.cmn.pvt.n01_005fCmn_005fCnst.<init>	4.862 ms (1 %)	322 µs	15.080
com.kbc.n01.ftn.pvt.n0f68.$func_z040_005fGet_005fCty_005fId	4.506 ms (1 %)	8.076 µs	558
com.kbc.logging.log4j.FileAppender.subAppend	3.660 ms (1 %)	1.639 µs	2.233
com.kbc.n01.itf.pvt.in0f73.<init>	3.595 ms (1 %)	2.147 µs	1.674
com.kbc.n01.ftn.pvt.n0f47.$func_z050_005fPcs_005fTrx	3.491 ms (1 %)	12.468 µs	280
egl.core.SqlData.<init>	3.264 ms (1 %)	169 µs	19.268
com.kbc.n01.itf.pvt.in0fe6a.$appendTo_ipt_005fZn__data__smt_005fLst__ohr	3.241 ms (0 %)	5.787 µs	560
com.kbc.n01.itf.pvt.in0fe6a.$appendTo_ipt_005fZn__data__smt_005fLst__am	3.065 ms (0 %)	5.474 µs	560
com.kbc.y41.cmn.pub.y41_005fUcd_005fWlog.<init>	2.711 ms (0 %)	154 µs	17.594
com.kbc.n01.itf.svc.in0s03.<init>	2.289 ms (0 %)	1.367 µs	1.674
com.kbc.n01.itf.pvt.in0f48.<init>	2.229 ms (0 %)	1.997 µs	1.116
com.kbc.n01.itf.pvt.in0f37.$appendTo_ipt_005fZn__data__trx_005fLst__dn...	2.168 ms (0 %)	129 µs	16.770
com.kbc.n01.ftn.pvt.n0f48.$func_z080_005fGet_005fFee	2.166 ms (0 %)	3.882 µs	558

View Filters: | Reset View Filters

Call Tree | Hot Spots | Call Graph | Call Tracer

15:36 | Snapshot

JProfiler

Profiling

IBM Websphere 6.1 Application Server on localhost - JProfiler 5.2.1

Session View Profiling Go To Window Help

Start Center | Stop | Freeze | Reload | Save Snapshot | Export | Run GC | Add Bookmark | Record Memory | Stop CPU | Session Settings | View Settings | Help | Generate Graph | Add Methods

Memory Views
Heap Walker
CPU Views
Thread Views
VM Telemetry Views

Thread selection: All thread groups
Thread status: Runnable
Aggregation level: Methods

s.util.ThreadPool$Worker.run
l: 665 s, Inherent: 0 µs

com.ibm.io.async.ResultHandler$2.run
Total: 422 s, Inherent: 403 s

com.ibm.ws.management.event.NotificationDispatcher$DispatchANotificationToAListener.run
Total: 0 µs, Inherent: 0 µs

com.ibm.ws.util.ThreadPool.getTask
Total: 4.263 ms, Inherent: 4.263 ms

com.ibm.ejs.util.am._Alarm.run
Total: 238 s, Inherent: 238 s

com.ibm.ws.wlm.threadmanager.SleeperThreadPool.run
Total: 0 µs, Inherent: 0 µs

com.ibm.ejs.oa.pool.PooledThread.run
Total: 0 µs, Inherent: 0 µs

com.kbc.y41.rt.blue.AbstractSer
Total: 1.519 ms, Ir

com.kbc.y41.rt.blue.service.Servic
Total: 17.781 ms, I

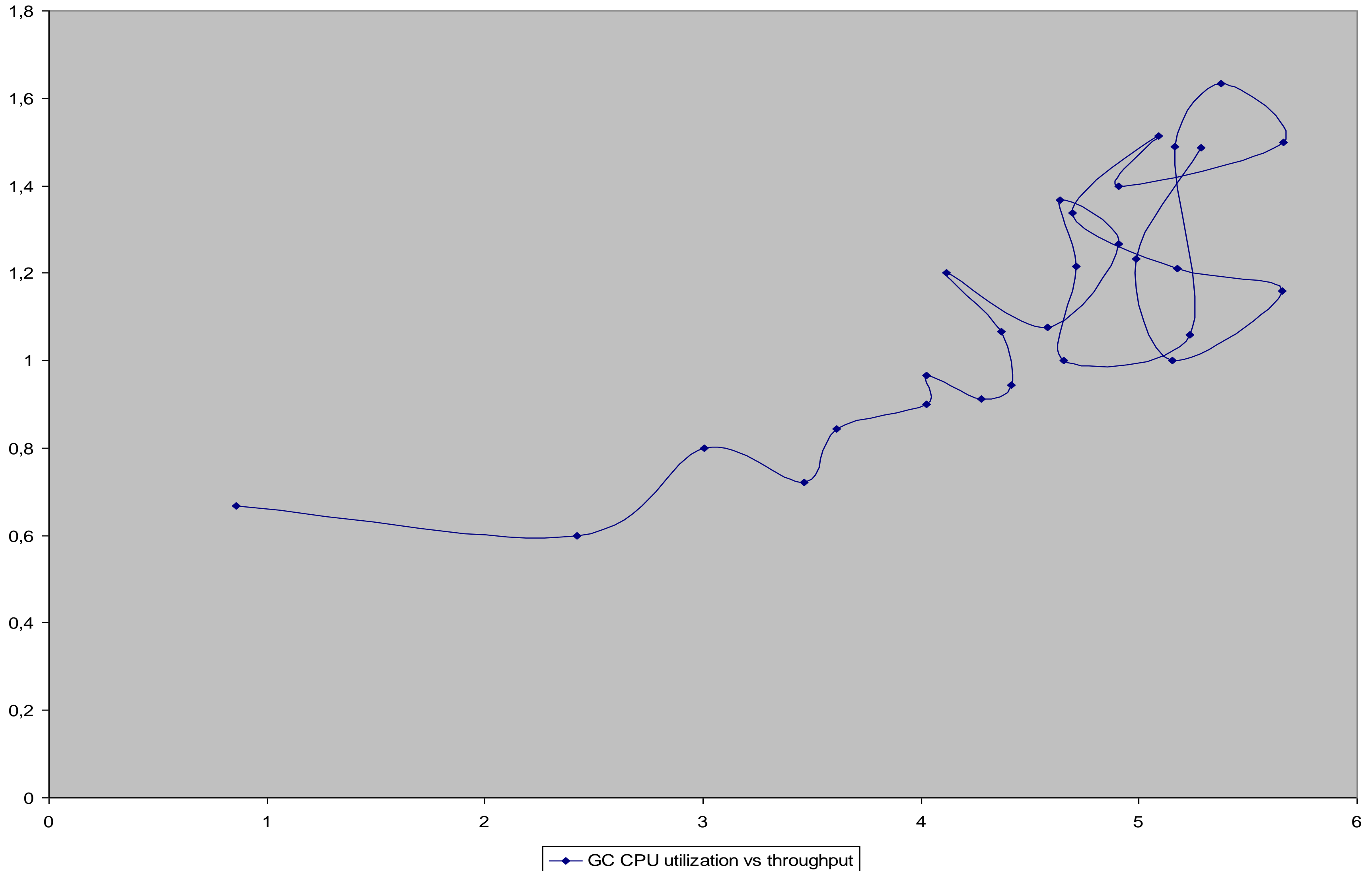
Call Tree | Hot Spots | Call Graph | Call Tracer

204:58 Profiling

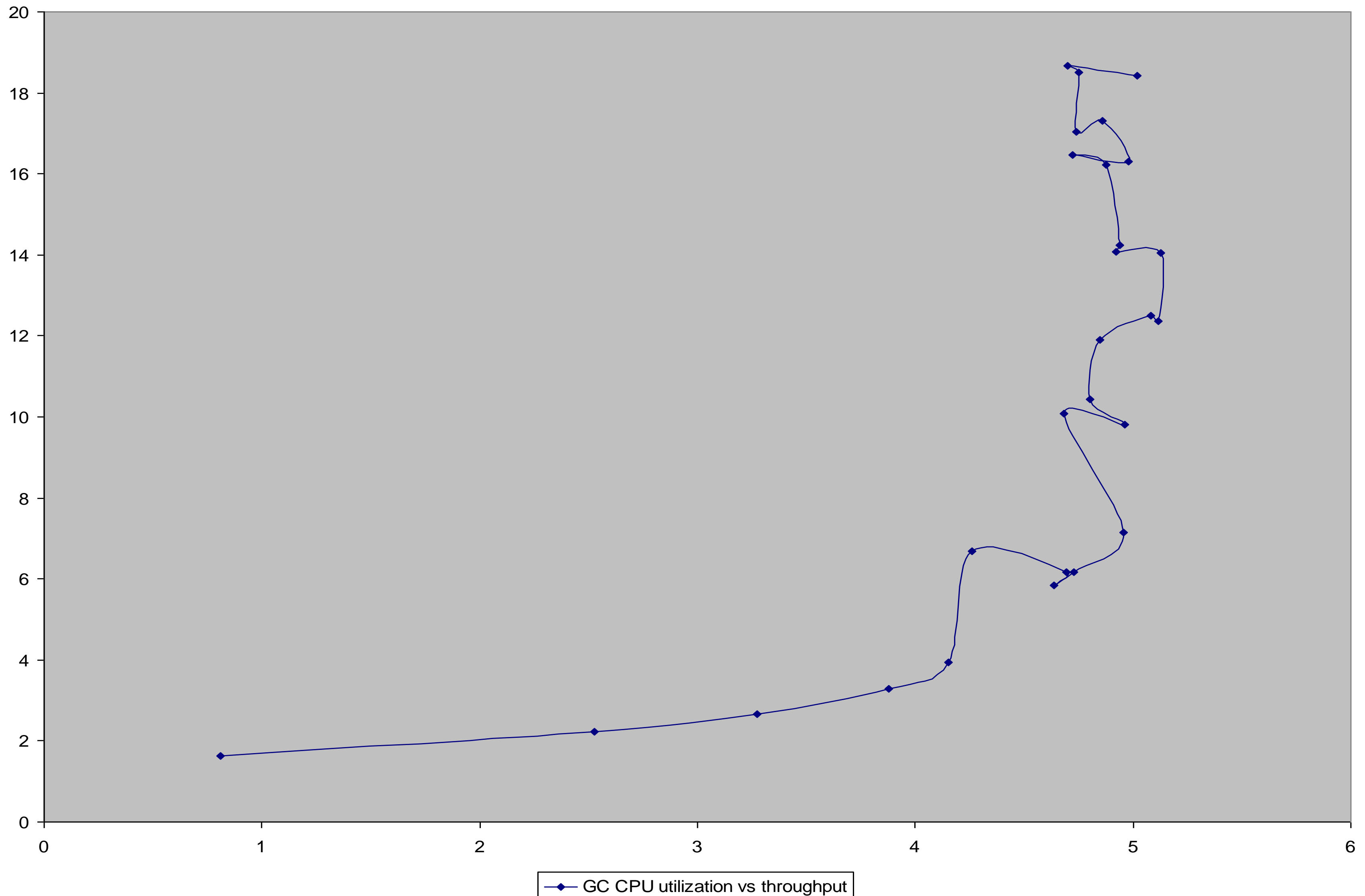
Memory-bound application in garbage-collected language

- Memory-intensive applications have to
  - Allocate memory
  - Initialize datastructures
  - Garbage-collect
- All of the above cost CPU – monitor this
- Look for non-linear behavior

‘Well-behaved’ garbage collection



Problematic garbage collection



Profiling

no logging, no error handling.jps - JProfiler 5.2.1

Session View Profiling Go To Window Help

Start Center | Stop | Freeze | Reload | Save Snapshot | Export | Run GC | Add Bookmark | Record Memory | Record CPU | Session Settings | View Settings | Help | Calculate | Show in Heap Walker | Reset GC history | Mark Current

Calculate new allocation hot spots

Memory Views
Heap Walker
CPU Views
Thread Views
VM Telemetry Views

JProfiler

Recorded allocations of: **All classes**

Liveness mode: **Garbage collected objects**

Aggregation level: Methods

Filtered classes: show separately

Hot spot	Allocated memory	Allocations
com.ibm.javart.OverlayContainer.<init>	1.007 MB (11 %)	1.673.152
com.ibm.javart.UnicodeItem.<init>	991 MB (11 %)	2.725.569
com.ibm.javart.operations.Assign.run(com.ibm.javart.resources.Program, com.ibm.j...	962 MB (11 %)	2.086.486
com.ibm.javart.OverlayUnicodeItem.<init>	765 MB (8 %)	12.541.400
java.sql.Statement.executeQuery	498 MB (5 %)	7.897.345
com.ibm.javart.OverlayCharItem.<init>	390 MB (4 %)	5.682.871
com.ibm.javart.operations.Assign.run(com.ibm.javart.resources.Program, com.ibm.j...	378 MB (4 %)	2.438.014
org.apache.log4j.FileAppender.subAppend	363 MB (4 %)	51.397
java.sql.Connection.createStatement	295 MB (3 %)	924.651
com.ibm.javart.operations.SetEmpty.run(com.ibm.javart.resources.Program, com.ib...	227 MB (2 %)	4.591.155
com.ibm.javart.calls.Caller.call	180 MB (2 %)	605.252
com.ibm.javart.OverlayNumericDecItem.<init>	151 MB (1 %)	1.801.932
com.ibm.ws.util.ThreadPool$Worker.run	147 MB (1 %)	2.745.132
com.ibm.javart.file.FileRecord.<init>	120 MB (1 %)	27.370
com.ibm.javart.sql.Sql.begin	100 MB (1 %)	1.786.474
java.lang.String.<init>	91.742 kB (1 %)	5.834
com.ibm.javart.resources.RunUnit.<init>	81.976 kB (0 %)	1.855.943
com.ibm.javart.UnicodeValue.getValue	81.760 kB (0 %)	887.436
java.lang.StringBuffer.<init>	80.250 kB (0 %)	658.757
com.ibm.javart.operations.Compare.run(com.ibm.javart.resources.Program, com.ib...	79.184 kB (0 %)	2.616.400
java.lang.StringBuffer.append	71.374 kB (0 %)	99.358
com.ibm.javart.OverlaySmallNumericItem.<init>	58.474 kB (0 %)	831.633
com.ibm.javart.resources.JavartProperties.getResourceAssociations	56.487 kB (0 %)	1.372.860
com.ibm.javart.arrays.UnicodeArray.<init>	55.423 kB (0 %)	1.090.190
com.ibm.javart.CharItem.<init>	51.913 kB (0 %)	1.004.353
java.sql.Statement.execute	50.056 kB (0 %)	390.569
com.ibm.javart.operations.SetEmpty.run(com.ibm.javart.resources.Program, com.ib...	41.817 kB (0 %)	394.126
com.ibm.javart.arrays.CharArray.<init>	37.849 kB (0 %)	700.990
com.ibm.javart.operations.Assign.run(com.ibm.javart.resources.Program, com.ibm.j...	36.275 kB (0 %)	63.446
com.ibm.javart.CharItem.setValue	33.605 kB (0 %)	640.658
com.ibm.javart.calls.PowerServer.call	31.349 kB (0 %)	22.283
com.ibm.javart.operations.SetEmpty.run(com.ibm.javart.resources.Program, com.ib...	30.644 kB (0 %)	27.596
com.ibm.javart.ByteStorage.<init>	30.380 kB (0 %)	1.940
com.ibm.javart.ByteStorage.getBytesCopy	30.327 kB (0 %)	970
com.ibm.javart.operations.ConcatValue.run	29.465 kB (0 %)	778.804
com.ibm.javart.IntItem.<init>	28.612 kB (0 %)	732.479

View Filters: Reset View Filters

All Objects | Recorded Objects | Allocation Call Tree | Allocation Hot Spots | Class Tracker

135:54 Snapshot

Example 2: Oracle server CPU is limiting factor

- A CPU-bound database is a well-tuned database
- What is CPU doing?
  - Databases usually gather statistics on their operation
  - They usually offer a consolidated report
  - e.g. Oracle’s Automatic Workload Repository

Oracle AWR report overview

Main Report

- Report Summary
- Wait Events Statistics
- SQL Statistics
- Instance Activity Statistics
- IO Stats
- Buffer Pool Statistics
- Advisory Statistics
- Wait Statistics
- Undo Statistics
- Latch Statistics
- Segment Statistics
- Dictionary Cache Statistics
- Library Cache Statistics
- Memory Statistics
- Streams Statistics
- Resource Limit Statistics
- init.ora Parameters

Wait Events Statistics

- Time Model Statistics
- Wait Class
- Wait Events
- Background Wait Events
- Operating System Statistics
- Service Statistics
- Service Wait Class Stats

SQL Statistics

- SQL ordered by Elapsed Time
- SQL ordered by CPU Time
- SQL ordered by Gets
- SQL ordered by Reads
- SQL ordered by Executions
- SQL ordered by Parse Calls
- SQL ordered by Sharable Memory
- SQL ordered by Version Count
- Complete List of SQL Text

Instance Activity Statistics

- Instance Activity Stats
- Instance Activity Stats - Absolute Values
- Instance Activity Stats - Thread Activity

IO Stats

- Tablespace IO Stats
- File IO Stats

Advisory Statistics

- Instance Recovery Stats
- Buffer Pool Advisory
- PGA Aggr Summary
- PGA Aggr Target Stats
- PGA Aggr Target Histogram
- PGA Memory Advisory
- Shared Pool Advisory
- SGA Target Advisory
- Streams Pool Advisory
- Java Pool Advisory

Wait Statistics

- Buffer Wait Statistics
- Enqueue Activity

Undo Statistics

- Undo Segment Summary
- Undo Segment Stats

Latch Statistics

- Latch Activity
- Latch Sleep Breakdown
- Latch Miss Sources
- Parent Latch Statistics
- Child Latch Statistics

Segment Statistics

- Segments by Logical Reads
- Segments by Physical Reads
- Segments by Row Lock Waits
- Segments by ITL Waits
- Segments by Buffer Busy Waits

Memory Statistics

- Process Memory Summary
- SGA Memory Summary
- SGA breakdown difference

Streams Statistics

- Streams CPU/IO Usage
- Streams Capture
- Streams Apply
- Buffered Queues
- Buffered Subscribers
- Rule Set

Oracle AWR report section

SQL ordered by CPU Time

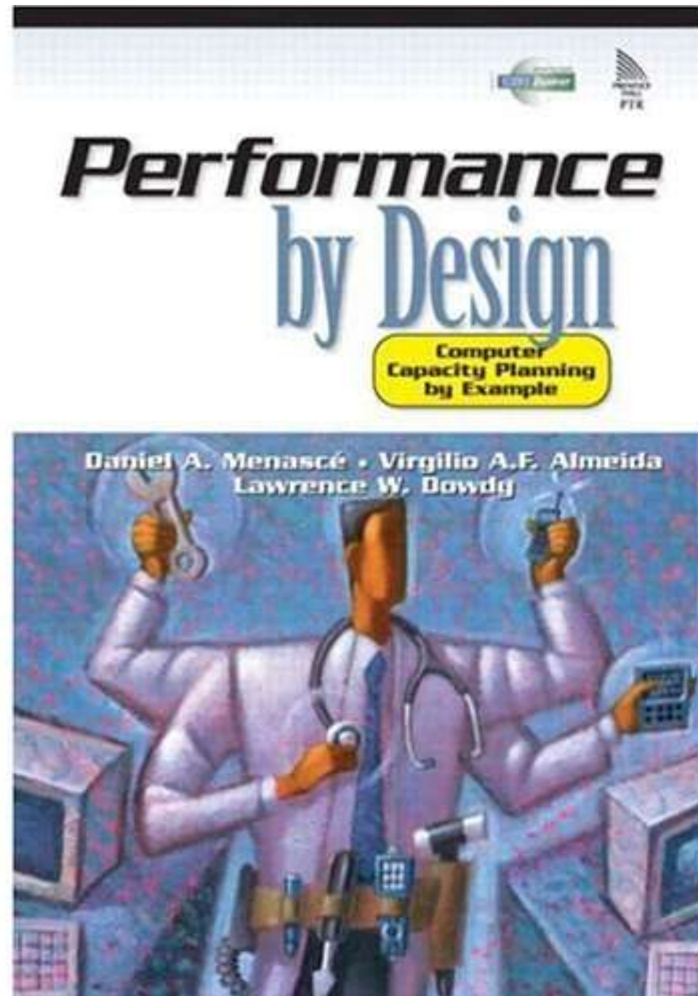
- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- % Total DB Time is the Elapsed Time of the SQL statement divided into the Total Database Time multiplied by 100

CPU Time (s)	Elapsed Time (s)	Executions	CPU per Exec (s)	% Total DB Time	SQL Id	SQL Module	SQL Text
348	644	314,431	0.00	2.05	bvjt04wzw81pa		insert into BCDB.TB_BCDB_DNT (...
309	17,403	16,554	0.02	55.45	08uyj0bcxam2x		insert into BCDB.TB_BCDB_MSO (...
249	513	33,094	0.01	1.63	59j9n0gz34uhm		BEGIN bcdb.pck_payment.prc_con...
147	260	33,095	0.00	0.83	6826t1t07dd44		select count(*) from cst_pym.c...
127	549	16,549	0.01	1.75	7yvunw9csnru7		BEGIN pb3.m24_bcd_interface.st...
111	192	33,094	0.00	0.61	6j5a5f2m04vxa		SELECT CASE WHEN :B6 = :B11 TH...
110	181	33,094	0.00	0.58	cq21hvjy45qsf		select to_char(sysdate, :"SYS...
96	164	33,093	0.00	0.52	g6mngwxav05bh		select cst552.type from cst_py...
83	135	33,094	0.00	0.43	cbvyhs8yhy2g5		select count(:"SYS_B_00") from...
73	125	49,647	0.00	0.40	9ugbj4bygrhcj		select pfs.ID_BCDB_PFFC, pfa....
58	368	49,647	0.00	1.17	1t607p04zsw7z		insert into BCDB.TB_BCDB_TRXB ...
41	1,306	1	40.96	4.16	24b3xmp4wd3tu		delete from sys.wri$_optstat_h...
34	937	16,549	0.00	2.98	ffwx4rwajnavr		insert into BCDB.TB_BCDB_TRX (...
16	399	16,549	0.00	1.27	fdr85gxzf5n0t		insert into BCDB.TB_BCDB_PSI (...

Discussion

- How does performance engineering fit into development process?
- When is it better to buy more hardware than to optimize the software?
- When do performance improvements cease to be cost-effective?
- When should performance findings be generalized into architectural guidelines?
- When do performance insights warrant architectural refactoring?

Reference



Thank you

http://johanpeeters.com
yo@johanpeeters.com