VELOCITY
SOFTWARE

Linux on z/VM Performance

Large Linux Guests

Session 13486

Rob van der Heij

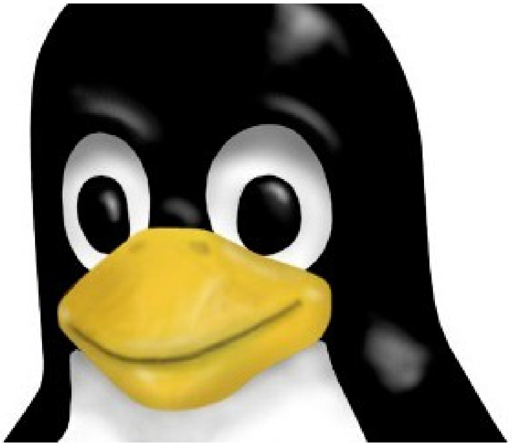
Velocity Software

http://www.velocitysoftware.com/

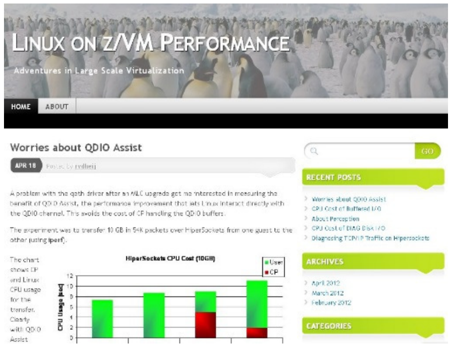
rvdheij@velocitysoftware.com

Agenda

What do you consider large?
Why use large Linux guests?
Managing performance data



Encounters with large guests
- Linux Large Pages
- Virtual CPUs
- Single guest or multiple guests
- Taming the Page Cache
- Java applications



http://zvmperf.wordpress.com/

VELOCITY SOFTWARE

What do you consider large?

Experiment in 2006

z/VM on P/390

- 3-4 MIPS
- 128 MB Main Memory
- 100 Linux Guests

This was small,
even was in 2006…

Penguins on a Pin Head

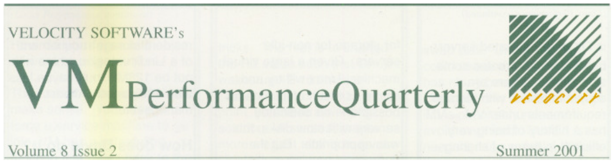
Experiences with tuning Linux on a P/390

Rob van der Heij
Velocity Software, Inc

rvdheij@velocitysoftware.com

http://velocitysoftware.com/

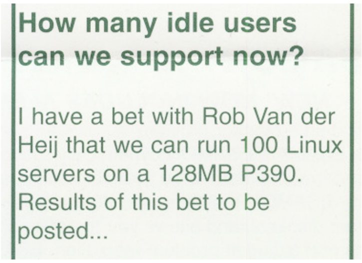
A complete System/390 processor on a single PCI card.



VELOCITY SOFTWARE's

VMPerformanceQuarterly

Volume 8 Issue 2 — Summer 2001



How many idle users can we support now?

I have a bet with Rob Van der Heij that we can run 100 Linux servers on a 128MB P390. Results of this bet to be posted...

VELOCITY SOFTWARE

What do you consider large?

Penguins on a Pin Head

- 3-4 MIPS
- 128 MB Main Memory
- 100 Linux Guests
  - Virtual machines 30 MB
  - Resident 0.5 – 4 MB
  - Overcommit 3-4

Customer in 2013

- 50,000 MIPS
- 1500 GB Main Memory
- 100 Linux Guests
  - Virtual machines 20-80 GB
  - Resident 20-50 GB
  - Overcommit 2-3

This is bigger

- CPU 10,000
- Memory 10,000
- Guest size 10,000

Number of guests is about the same

What do you consider large?

Hypervisor

- z/VM image today maximum 1 TB
- z/VM supports up to 32 logical CPUs

Linux Guest

- Wide range of possible configurations
- Depends on the number of virtual machines sharing
- Often around 1-10% of the hypervisor resources

How big should the guest be so that we do not have any performance problems?

VELOCITY SOFTWARE

Why use large Linux guests?

Average guest is larger than a few years ago

- Less focus on resource efficiency
  - Different style of applications and application design
- Enterprise Application Ecosystems
  - Manage their own resource pool
- Increased workload
  - More data and higher transaction rates

VELOCITY SOFTWARE

Less Focus on Resource Efficiency

Content-rich user interface

- Dynamic Content Management
- Customized and personalized application interface
- Integration of other data sources in user interface
  - Correlation with social network or shopping history

Different style of application design

- Building-block application development
  - Often takes more memory and CPU cycles
  - Not always perfect fit
  - May encourage adding additional eye candy
- Java-based application frameworks
  - Table-driven application design
  - Platform independent

Enterprise Application Ecosystems

Multi-threaded application middleware

- Acquires resources from Linux operating system
- Uses internal strategy to run and optimize the workload
- Assumes sole ownership of resources (no shared resources)
- Memory resources are retained until service is stopped

Many popular enterprise applications

- JVM with Java Application (WebSphere AS, JBoss)
- Databases (DB2, Oracle)
- ERP / CRM Applications (Siebel, SAP)

Performance Challenges

- Resource usage may not correlate with workload patterns
- Configuration of guest and application must match

Increased Workload

More data and higher transaction rates

- It is all just much more and bigger than before
  - It helps to look at other metrics too
  - At best it scales linear, sometimes much worse
- Linux on z/VM is part of many enterprise solutions
  - Applications deal with much larger workload than before
  - Aspect of being a mainstream platform
- Platform serves a very wide range of workloads
  - Scalability is normally taken for granted
  - Do not expect it to work without additional resources
  - Expectation sometimes scales less well

"I know this is inefficient, but if it works for 100,000 records, why would it be a problem with 107,000,000 records ?"

Managing performance data

All performance data is needed to understand performance

- It does not work with just part of the data
- Production and Development share resources
- Systems are often used 24 hours per day
- Chargeback data is needed
  - Even if only to encourage resource efficiency

Managing performance data is critical

- Especially with that much more resources

Performance management must scale for large systems

- Group data in different ways with full capture
- Apply thresholds to keep only interesting data
- Summarize complete data for chargeback and planning
- Condense older data to allow long term archival

Needle in a haystack

Data from many processes

- Can be a challenge to manage
- Thresholds to keep interesting data
- Condense the data in larger intervals
  - Still 10,000 lines of process data per day
- Grouping by application or user

Last week we used 60% even at night
Now we are down to 50% Why?

node/	<-Process Ident->			Nice	PRTY	<------CPU Percents---->				
Name	ID	PPID	GRP	Valu	Valu	Tot	sys	user	syst	usrt
00:30:00										
SPOOKY16	0	0	0	0	0	0.59	0.20	0.39	0.00	0.00
SPOOKY18	0	0	0	0	0	1.14	0.35	0.78	0.00	0.00
SPOOKY13	0	0	0	0	0	1.10	0.29	0.48	0.14	0.19
SPOOKY3	0	0	0	0	0	0.70	0.31	0.26	0.02	0.12
snmpd	1294	1	1293	-10	6	0.55	0.30	0.23	0.01	0.01
SPOOKY33	0	0	0	0	0	2.73	0.89	1.49	0.06	0.30
java	4151	1	4151	0	20	1.46	0.50	0.96	0	0
SPOOKY34	0	0	0	0	0	1.48	0.48	0.99	0.00	0.00
java	5237	1	5237	0	20	0.63	0.16	0.47	0	0
SPOOKY30	0	0	0	0	0	1.98	0.87	1.10	0.00	0.00
db2sysc	4621	4619	4621	0	20	1.11	0.44	0.67	0	0
SPOOKY20	0	0	0	0	0	0.64	0.28	0.35	0.00	0.00
SPOOKY25	0	0	0	0	0	2.32	0.47	1.06	0.37	0.43
db2fmcd	3008	1	3008	0	20	0.81	0.01	0.00	0.37	0.43
db2sysc	3620	3618	3620	0	20	0.60	0.09	0.51	0	0

Needle in a haystack

Grouping data from different servers

- Grouping in user class or node groups
- Aggregated usage from related servers
  - Tiers that make up an application
  - Servers that share the load
- Helps to manage performance data

Node/ Date Time	Process/ Application name	ID	<---Processor Percent---> Total	<Process> sys	user	<Children> syst	usrt
Node Groups							
*Spooky	*Totals*	0	24.1	7.0	12.5	1.4	3.2
	cogboots	0	2.9	0.8	2.1	0	0
	db2fmcd	0	2.0	0.0	0.0	0.9	1.1
	db2syscr	0	2.4	0.4	2.0	0	0
	init	0	2.1	0.0	0.0	0.3	1.7
	java	0	5.9	1.8	4.1	0	0
	kr4agent	0	1.4	0.1	1.3	0	0
	kynagent	0	0.5	0.1	0.4	0	0
	snmpd	0	4.9	3.2	1.6	0.0	0.0

Node/ Date Time	Process/ Application name	ID	<---Processor Percent---> Total	<Process> sys	user	<Children> syst	usrt
Node Groups							
*Spooky	*Totals*	0	30.3	7.5	18.8	1.5	2.5
	cogboots	0	1.5	0.8	0.7	0	0
	db2fmcd	0	2.2	0.0	0.0	1.0	1.2
	db2syscr	0	1.8	0.3	1.5	0	0
	httpd2-p	0	6.6	0.1	6.5	0	0
	init	0	1.4	0.0	0.0	0.4	1.0
	java	0	6.0	1.6	4.4	0	0
	kr4agent	0	1.5	0.1	1.4	0	0
	mysqld	0	1.5	0.3	1.2	0	0
	snmpd	0	5.4	3.6	1.7	0.0	0.0

Encounters with large guests

Inspired by real customer scenarios

- Sometimes reproduced in lab environment
- Often simulated with artificial workload

Relevant for both small and large systems

- Ignorance and personal taste may not scale
- Bad ideas show best in extreme cases

Data presented here has been collected with zVPS on real customer systems, sometimes reproduced in a lab environment to avoid distraction.

Linux Large Pages

With large memory size, 4K page granularity is overkill

- Enterprise application will manage the memory itself

Virtual Memory hardware supports larger pages

- Efficient use of hardware address cache
- Enhanced DAT (z10) provides both 4K and 1M page size

z/VM does not support large pages

- z/VM guest will see hardware without the EDAT feature

Linux can emulate large pages using 4K pages

- Does not exploit the hardware advantages
- Still requires manipulation of 4K pages in Linux
- … but it can save memory resources for Oracle database

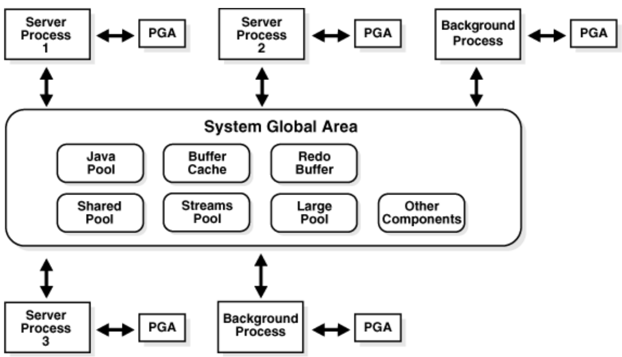
Oracle process uses SGA and PGA

- SGA is shared among all database processes
- Mapped into each process virtual memory
- Page tables duplicated for each process
- Adds up to 2 MB of tables per GB of memory, per process

```
Example:

SGA                 32 GB
Page Tables         64 MB
x 512 processes
= Total Tables      32 GB
```

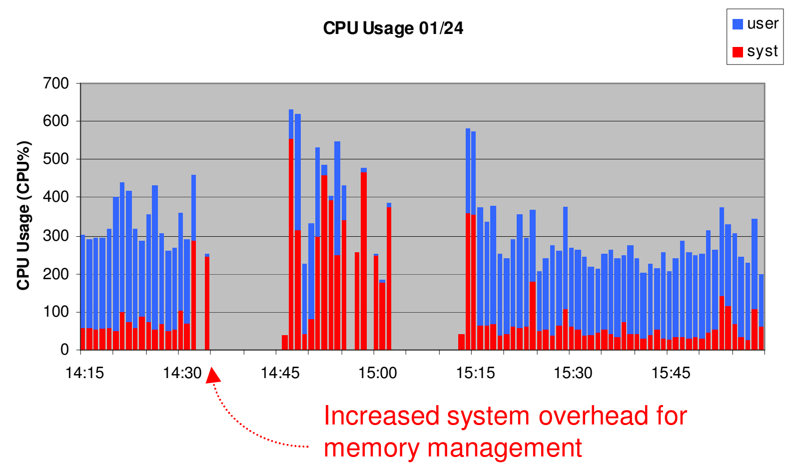
Rule of Thumb: With 500 Oracle connections, tables for 4K pages double your memory requirement

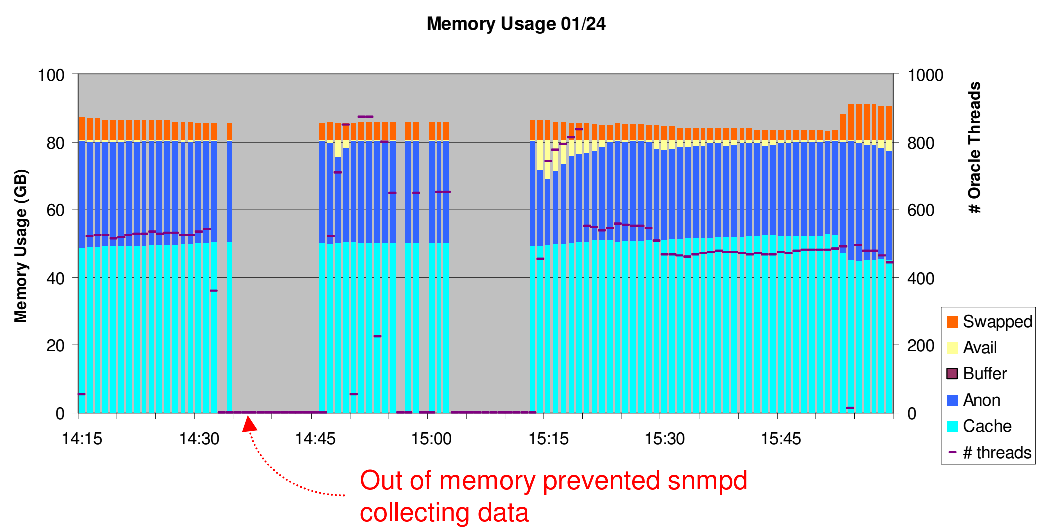


Example: Oracle Database

- SGA ~50G
- Connections ~500
- Linux Guest 80G

- 50G + 50G > 80G
- Only part of SGA actually used
  - Per process less than 50G mapped





Urgent Recommendation:
- Limit Number of Connections
- Use Large Pages

VELOCITY SOFTWARE

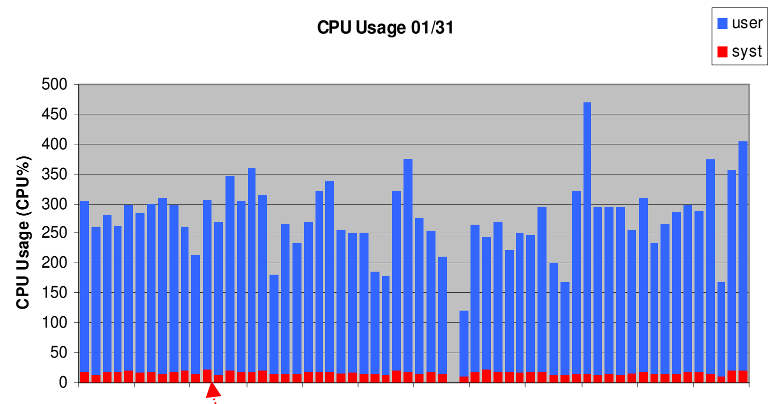
Linux Large Pages

Example: Oracle Database

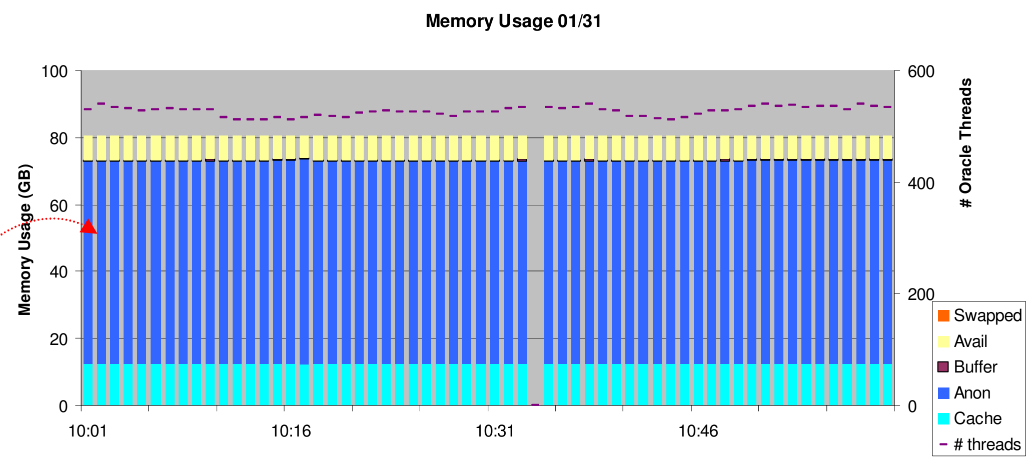
- SGA ~50G
- Connections ~500
- Linux Guest 80G

Using Large Pages for SGA

- Reserved 50G of Linux memory
- System overhead is gone
- All productive Oracle work



Almost no system overhead



SGA now outside cache

VELOCITY SOFTWARE

Linux Large Pages

Oracle SGA using Linux Large Pages

- Savings can be substantial
  - Especially with large number of database connections
- Part of guest memory set aside as “huge pages”
  - Through kernel parameter at boot or dynamic
  - When dynamic, do it early to avoid fragmentation
  - Must be large enough to hold the SGA, anything more is wasted
    - Check the page size (1M versus 2M)
- Disable Oracle Automated Memory Management (AMM)
  - Use SGA_TARGET and PGA_TARGET
- Even with large pages: do not make SGA bigger than necessary
- Do not use Linux “transparent large pages” support

Does not apply to DB2 LUW or JVM Heap

VELOCITY SOFTWARE

z/VM Page Reorder

Part of z/VM memory management

- Identify unreferenced user pages for page-out
- Less useful for most Linux on z/VM systems
- Freezes the entire virtual machine for some time
  - Approximately 1 second for every 8 GB of memory
  - Happens more often on busy Linux guests

Fixed in z/VM 6.3

CP command to disable page reorder

- For all virtual machines or specific ones

Ensure enough expanded memory for paging

- Reduces impact when wrong pages are selected

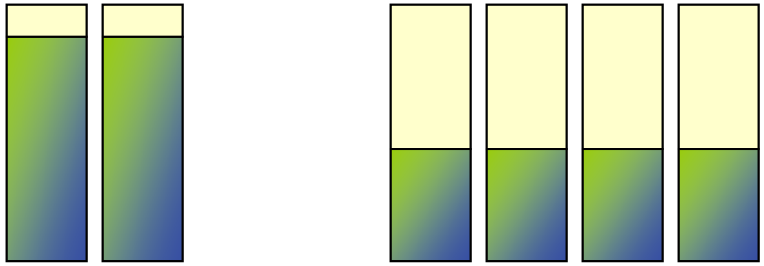
Virtual CPUs

Large workload takes more CPU resources

- Add virtual CPUs to provide peak capacity
- Not more virtual CPUs than expected available
  - Often less than number of logical CPUs
- Extra virtual CPUs don't provide more capacity
  - Scheduler share options determine capacity
- Linux assumes exclusive usage of resources
  - Not guaranteed in shared resource environment
  - When there is a virtual CPU, Linux assumes it will run
  - With more CPUs than capacity, z/VM will spread capacity

Example

- Linux runs 2 important tasks and 2 less important
- With 2 virtual CPUs
  - First run important tasks, other work when time permits
- With 4 virtual CPUs
  - Run all 4 tasks at the same time
  - z/VM will spread CPU capacity equal over virtual CPUs
  - Important work takes longer to complete



180% in 2 CPUs
90% each

180% in 4 CPUs
45% each

Virtual CPUs

Important Configuration Trade-Off

- More virtual CPUs
  - Deliver peak capacity when resources are available
- Less virtual CPUs
  - Improve single-thread throughput
  - Ensure predictable response times when little resources available
- As few as possible to deliver peak capacity

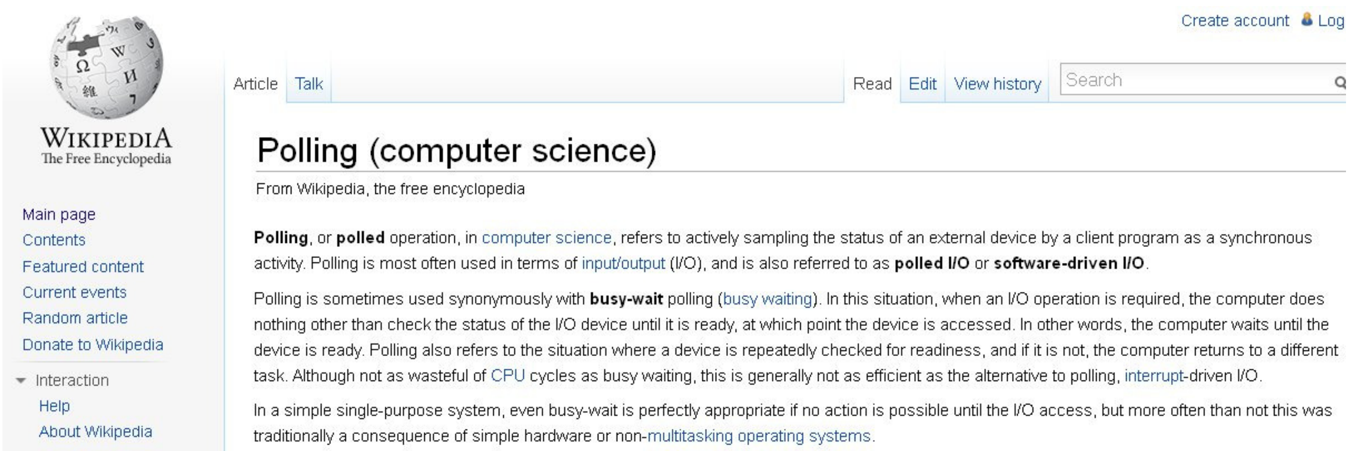
Understand CPU requirement

- CPU usage for peak and average in recent history
  - Shows what he got, not what he wanted
- Virtual CPU wait state analysis shows CPU queue
  - Virtual CPU in queue waiting to run

Virtual CPUs

Application Polling

- Frequent checking the status, busy-wait for service
- Poor design for shared resource environment
  - Mitigated by only installing the actual application
- Virtual CPUs get in queue for no reason
  - Do not consume much CPU and do not need more
  - It does not help much to wait faster



Create account Log

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia

Interaction
Help
About Wikipedia

Article Talk Read Edit View history Search

Polling (computer science)

From Wikipedia, the free encyclopedia

Polling, or **polled** operation, in computer science, refers to actively sampling the status of an external device by a client program as a synchronous activity. Polling is most often used in terms of input/output (I/O), and is also referred to as **polled I/O** or **software-driven I/O**.

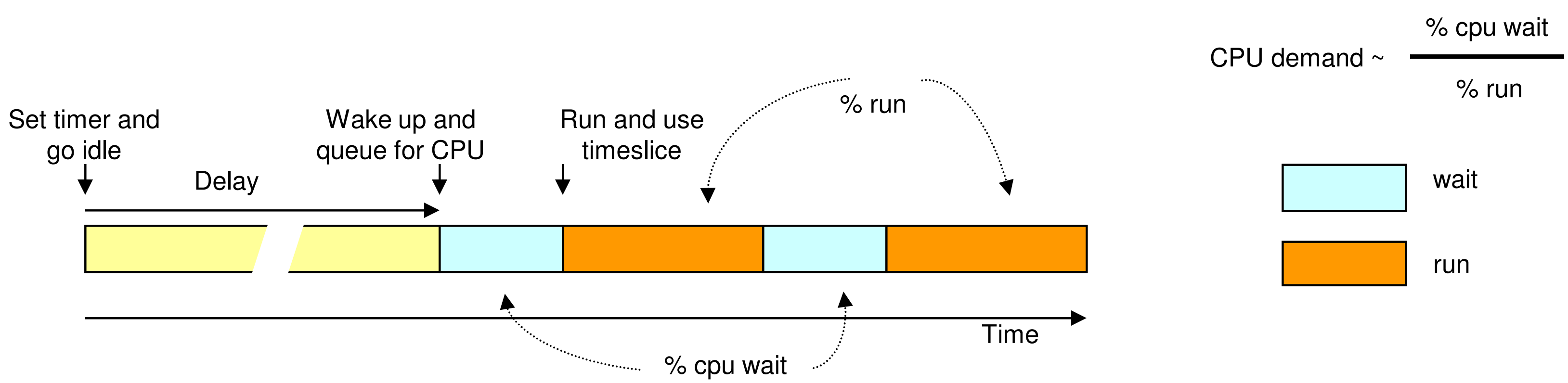
Polling is sometimes used synonymously with **busy-wait** polling (busy waiting). In this situation, when an I/O operation is required, the computer does nothing other than check the status of the I/O device until it is ready, at which point the device is accessed. In other words, the computer waits until the device is ready. Polling also refers to the situation where a device is repeatedly checked for readiness, and if it is not, the computer returns to a different task. Although not as wasteful of CPU cycles as busy waiting, this is generally not as efficient as the alternative to polling, interrupt-driven I/O.

In a simple single-purpose system, even busy-wait is perfectly appropriate if no action is possible until the I/O access, but more often than not this was traditionally a consequence of simple hardware or non-multitasking operating systems.

VELOCITY SOFTWARE

Virtual CPU State Sampling

- Done by z/VM monitor sampling, typically once per second
  - Counts how often running, waiting for CPU, idle, etc
  - CPU wait ratio indicates CPU contention



Polling process:
- Minimal CPU usage
- Short delay
- Mostly waiting for CPU

http://zvmperf.wordpress.com/2012/07/30/when-cpu-wait-is-misleading/

Polling and CPU State Sampling

- Polling inflates the CPU-wait numbers
  - As long as there is polling, Linux still has idle time
- Additional CPU capacity will only make it wait faster
  - CPU wait does not go away

```
1 of 1  Virtual CPU Wait State                    USER ROB01           2097 40F32
                  <--------- Virtual CPU State Percentage --------->  Poll
Time     User       Run CPUwt  CPwt Limit  IOwt PAGwt Othr Idle Dorm  Rate CPU%
-------- -------- ----- ----- ----- ----- ----- ----- ---- ---- ---- ----- ----
15:38:00 ROB01     20.0  26.7     0     0     0     0    0  253    0 648.0 27.1
15:54:00 ROB01     28.3   3.3     0     0     0     0    0 68.3  200 428.5 22.5
```

Virtual 3-way, 250% idle
Goes asleep 650 times/sec
Average 1.5 ms cycle
Using 0.3 ms per cycle

2 CPUs dormant, 70% idle
Less polling
CPUwt numbers are lower

Taming the Page Cache

Linux tries to find use for any excess memory

- Will cache data just-in-case
- Strategy is unproductive in shared environment
- Reference patterns interfere with z/VM paging

Just small enough, avoid excess memory

- Commonly suggested approach
- Even smaller with swap in VDISK to satisfy peaks

Hard to do with varying memory requirements

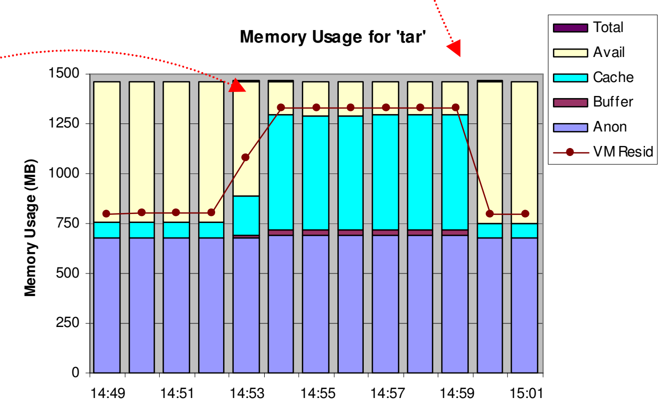
- Re-use of page cache may cause z/VM paging delays
- Large virtual machines require a lot of paging
- Tuning with cpuplugd is too slow to be effective

cmmflush - Flush out unused cached data at useful moments

- Removes all cached data and returns memory to z/VM
  - Use CMM driver to temporarily take away memory from Linux
- Challenge is to find good moment
  - After completion of unusual workload – avoids page-out of data
  - Before starting unusual workload – avoids page-in of data
- Disadvantages
  - Removes also useful data from cache
  - During flush process, the system may run out of memory
  - CPU overhead for returning pages to z/VM

cmmflush

Reading from disk



http://zvmperf.wordpress.com/2012/07/06/using-cmm-to-flush-a-linux-guests-memory/

nocache – Discourage Linux to Cache Data

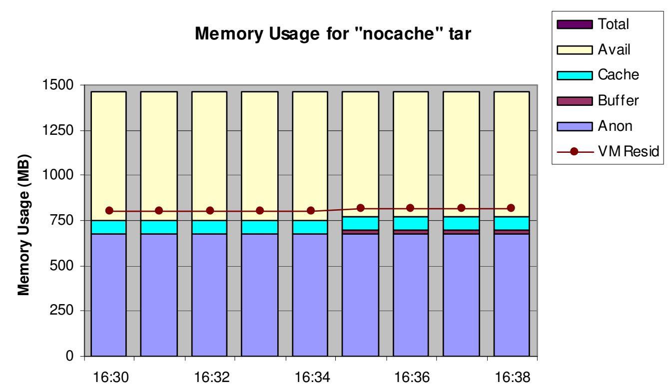
- Wrapper around application that wipes data from cache
  - Applies only to data touched by the application
  - Additional tools to selectively drop files from cache
- Useful for non-core applications
  - Backups, log file archival, security scanning, database load
- Experimental – Unsure yet how to package the function
  - Interested in feedback from users who want to try

total

cached

```
rvdheij@roblnx1:~> md5sum jvm-trc*
dbdeffb03e8e7c4659d869a52a99c202  jvm-trc5.txt
36e1b490a40dc7b01cdb0ea29d7867d2  jvm-trc6.txt
rvdheij@roblnx1:~> minc jvm-trc*
     450       450 jvm-trc5.txt
     450       450 jvm-trc6.txt
rvdheij@roblnx1:~> drop jvm-trc6.txt
rvdheij@roblnx1:~> minc jvm-trc*
     450       450 jvm-trc5.txt
     450         0 jvm-trc6.txt
```

dropped



http://zvmperf.wordpress.com/2013/01/31/the-nocache-approach/

VELOCITY SOFTWARE

Single Guest or Multiple Guests

Single Guest

- No duplication of Linux infrastructure
- Less things to manage
- Obvious approach without virtualized servers
- No communication overhead, less latency
- Less components to break, simple availability

Multiple Guests

- Separation of applications
- Tune each guest separately
- Software levels specifically for application
- Easier to identify performance problems
- Simple charge back and accounting

Single Guest or Multiple Guests

Be prepared to efficiently manage multiple guests

- Invest in processes to create additional guests
  - Often most complexity is beyond actual creating the servers
  - Be aware of manual tasks that need repeated for each server
- Use something that matches skills and tools
  - Shared R/O disks versus "minimal install"
- Look at simplified reporting

Keep unrelated applications in separate guests

- Take advantage of server idle periods
  - Avoid a big guest with "always something going on"
- Simplify software upgrades and availability requirements

Keep related applications apart as long as it makes sense

- Many exceptions (small MySQL or DB2 configuration database)
- Be aware of the level of interaction between tiers

Single Guest or Multiple Guests

Example: Rehost z/OS application on Linux

- z/OS with DB2 and COBOL jobs
- Linux on z/VM with Micro Focus COBOL and DB2 LUW

Initial Configuration

- Linux guest running MF COBOL
- Linux guest with DB2 LUW
- Resulted in excessive run times and high CPU usage

High CPU Usage and Latency

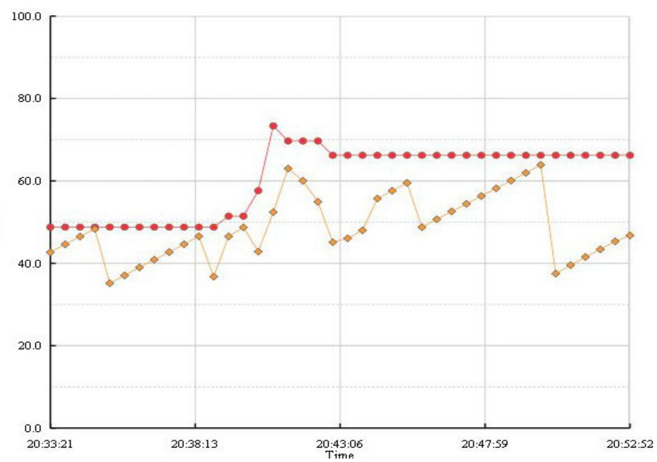
- Introduction of DRDA layer and TCP/IP comminication
  - More expensive than shared memory access under z/OS
- Less efficient cursor-based database access
- Run application and database in a single guest
  - Avoids overhead of DRDA and TCP/IP layer

Java heap size is one of most prominent parameters

- Java applications use the heap to store data
- Both temporary and persistent data
- Managed by regular Garbage Collection scans

Heap size is specified at JVM startup

- Usually kept in properties managed by application
- Defined by min and max heap size
- Heap grows until above configured minimum
  - Garbage collect tries to reclaim space
  - Extends heap until maximum
  - Returns excess beyond minimum



Heap size determines application footprint

- Requirement is determined by the application
  - Number of classes, active users, context size
  - Heap analyzers can reveal requirements
- Dedicated server approach sets min and max the same
  - Retains the full heap during JVM lifetime
  - Reduces GC overhead
  - Less attractive with shared resources
  - Hides heap requirements from Linux tools
- Alternative approach
  - Start with low minimum to see base requirement
  - Later adjust minimum to just above base requirement
  - Set maximum to absorb peaks

Java Applications

Garbage Collector Threads

- Option to spread GC over multiple CPUs
  - Only helps when they really will run
  - Consider to override the default of N slaves

Some applications require multiple JVM's

- Each will need its heap to be sized right
  - Total must fit in Linux memory
- Lower minimum heap size may be effective
  - One JVM can use what the other released
- Ignore single-shot Java programs

Keep production systems clean

- Do not install sample programs there
  - Security exposure
  - More than just disk space

Conclusion

Sizing does matter

- Linux on z/VM scales for large range of workloads
- Configuration options need to be coordinated
- Collect and study performance data
  - Compute normalized resource usage
  - Investigate exceptional usage
  - Your Linux admin may not have seen it that big either

Take advantage of virtualization

- Keep different workloads apart
- Tune the guest for that particular workload

Linux on z/VM Performance

Large Linux Guests

Rob van der Heij

Velocity Software

http://www.velocitysoftware.com/

rvdheij@velocitysoftware.com

Session 13486