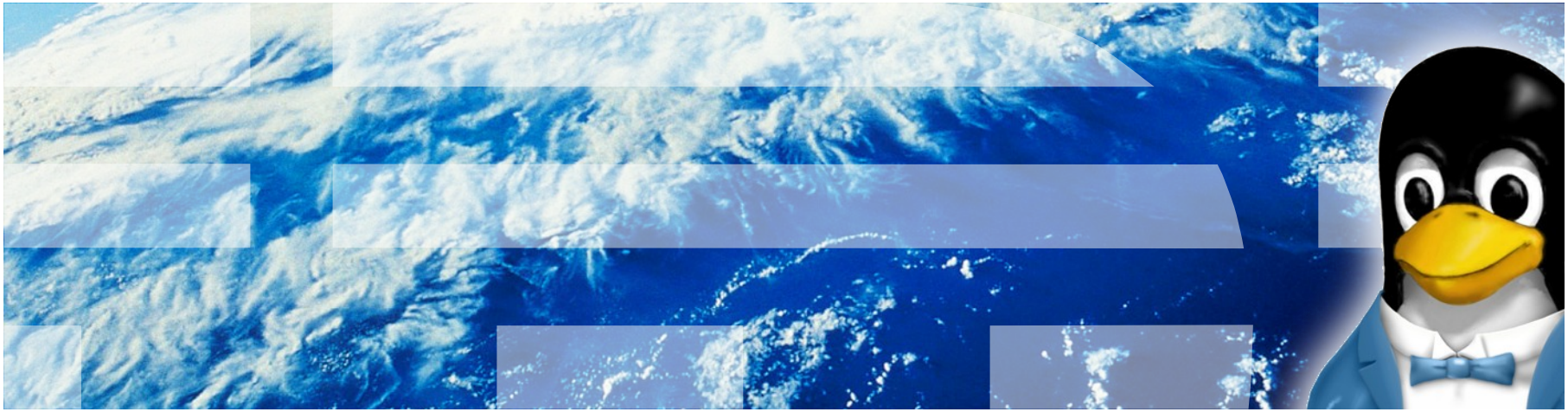
Ciju Rajan K [cijurajan@in.ibm.com]

Linux Technology Center

Analyzing the impact of sysctl scheduler tunables

LinuxCon, Vancouver 2011



Agenda

- Introduction to scheduler tunables
- How to tweak the scheduler tunables
- Introduction to CFS
- A deep dive into scheduler tunables
- Test environment
- A quick overview about the workloads used
- Impact of sched_latency_ns
- Impact of sched_min_granularity_ns
- Impact of sched_compat_yield
- Impact of sched_wakeup_granularity_ns
- References

Introduction to scheduler tunables

- **What are the scheduler tunables?**
  - Scheduler knobs exported to the user
  - Controls the behavior of the scheduler
  - Exported via sysctl: ***/proc/sys/kernel/sched_****
- **Why do we need them?**
  - Scheduler is used from small embedded systems to large HPC clusters
  - Application scheduling behavior might have to be tweaked
  - Workload characteristics are different
  - Default scheduler settings might not be optimal always
- **What will you gain from this presentation?**
  - How to tune the scheduler tunables
  - How to arrive at the optimal set of values
  - Performance improvements obtained by tuning the scheduler knobs

How to tweak the scheduler tunables

- **There are two ways to alter the default values**
  - Change the values directly: ***/proc/sys/kernel/sched_****
  - Using the sysctl command to change the kernel parameters at run time

```
[root@hs22 kernel]# pwd
/proc/sys/kernel
[root@hs22 kernel]# ls sched_*
sched_autogroup_enabled      sched_migration_cost      sched_rt_period_us
sched_time_avg               sched_child_runs_first    sched_min_granularity_ns
sched_rt_runtime_us          sched_tunable_scaling     sched_latency_ns
sched_nr_migrate             sched_shares_window       sched_wakeup_granularity_ns

sched_domain:
cpu0 cpu1 cpu10 cpu11 cpu12 cpu13 cpu14 cpu15 cpu2 cpu3 cpu4 cpu5 cpu6 cpu7
cpu8 cpu9
```

- **Using /etc/sysctl.conf**
  - Eg: **kernel.sched_latency_ns = 24000000**
  - **# sysctl -p**

Introduction to CFS

- Completely Fair Scheduler
- Part of mainline kernel since v2.6.23
- Fairness imbalance is expressed via per task wait_runtime
- Tasks are ordered in a RB Tree sorted by “rq->fair_clock – p->wait_runtime”
- Scheduler picks the left most task
- CFS does not have any notion of 'timeslices'

A deep dive in to the scheduler tunables

- **sched_latency_ns**
  - Targeted preemption latency for CPU-bound tasks
  - A period in which each task runs once
  - Default = 6ms * (ilog(ncpus))
    Unit = ns
  - Not the same as time slice length

- **sched_min_granularity**
  - The minimum time after which a task become eligible to be preempted
  - The minimum possible preemption granularity
  - Default:
    0.75 msec * (ilog(ncpus)
  - sched_latency / nr_tasks

- **sched_compat_yield** *
  - Makes sys_schedule_yield more aggressive
  - Moves the yielding task to the last in the rb tree
  - Retained for compatibility

- **sched_migration_cost**
  - Tunable to determine if a task can be migrated from one cpu to another
  - Larger value means, the chances of the tasks to be migrated to another cpus becomes less
  - Also determines if the current task is cache hot

* Not present in mainline kernel anymore

A deep dive in to the scheduler tunables

- **sched_nr_migrate**
  - Number of tasks to iterate in a single load balance run
  - Limited because this is done with IRQs disabled

- **sched_child_runs_first**
  - If set to 0 (default) then parent will (try to) run first otherwise child.

- **sched_wakeup_granularity_ns**
  - Reduces over-scheduling
  - Gives an hint whether to preempt the current task or not

- **sched_tunable_scaling**
  - The initial- and re-scaling of tunables
  - Default: Scaled logarithmically
  - Scaling now takes place on all kind of cpu add/remove events

* Not present in mainline kernel anymore

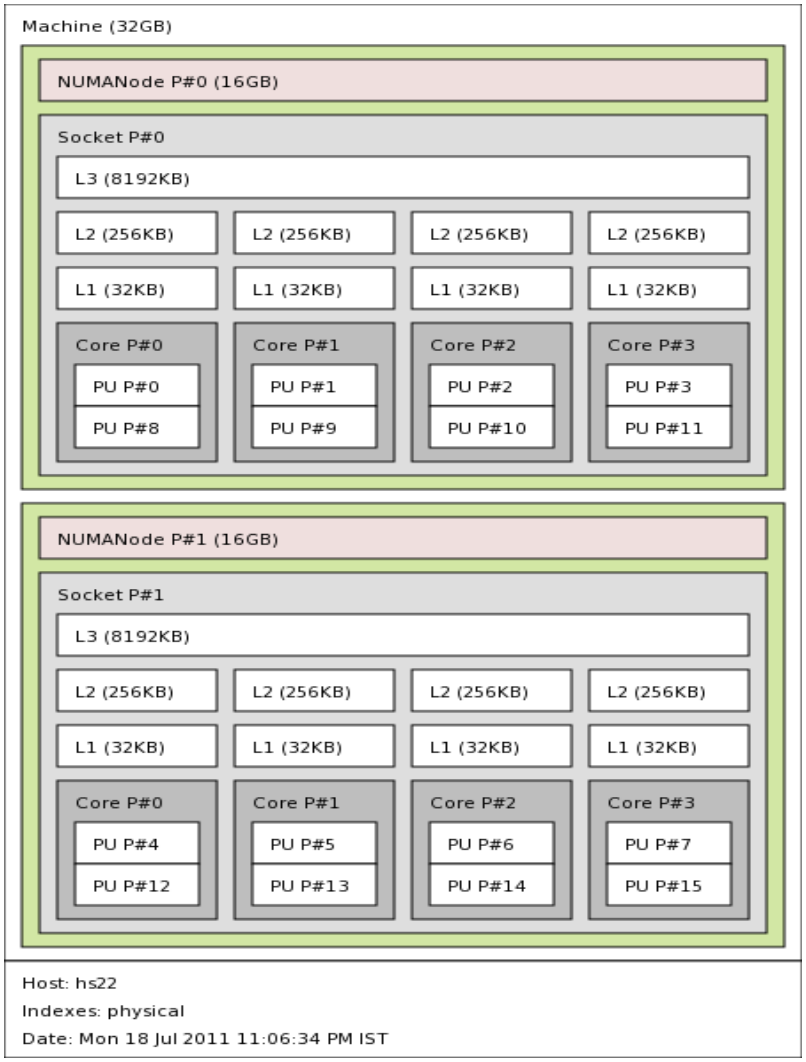
A deep dive in to the scheduler tunables

- **sched_rt_period_us**
  - Period over which we measure -rt task cpu usage in micro seconds
  - The scheduling period that is equivalent to 100% CPU bandwidth
  - Default = 1s

- **sched_time_avg**
  - Period over which we average the RT time consumption
  - Default: 4ms

- **sched_rt_runtime_us**
  - A global limit on how much time realtime scheduling may use
  - Part of the period that we allow rt tasks to run in micro seconds
  - Default: 0.95s
  - A run time of -1 specifies
    runtime == period

Test environment

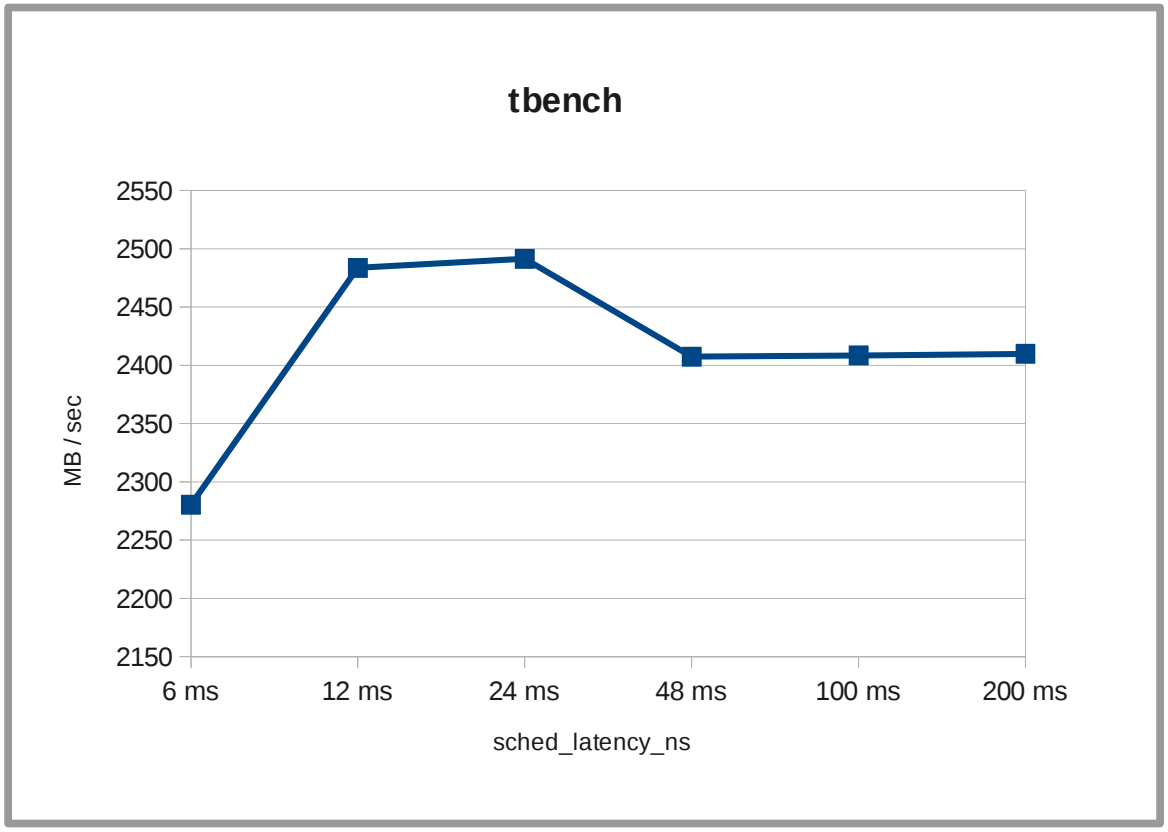


- **Hardware: Dual socket quad core with HT support**
- **Linux Distribution: Fedora 14**
- **Benchmarks**
  - Tbench
  - Dbench
  - SPECJbb
  - Lmbench
  - Kernbench
  - Hackbench

A quick overview about the workloads used

- **Dbench**
  - Generate IO loads
  - Used to stress the filesystems
- **Tbench**
  - Produces TCP and process load
  - Does invoke the socket() calls
- **SPECJbb**
  - Java based benchmark
  - Simulates database transactions
  - Cpu intensive workload
- **Hackbench**
  - Simulates the connections established for a chat room
- **Kernbench**
  - Cpu throughput benchmark
  - Used to compare the different kernels on the same machine
- **Lmbench**
  - Suite of micro benchmarks
  - Bandwidth (cached file read, m/m read / write, pipe)
  - Latency (context switches, system call overhead, m/m read / write latency / remote wakeups)

Impact of *sched_latency_ns*

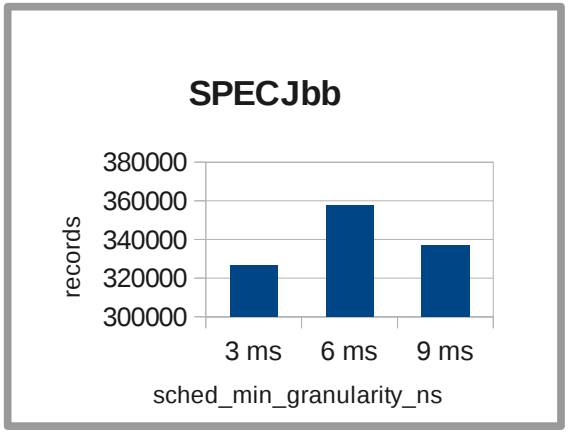


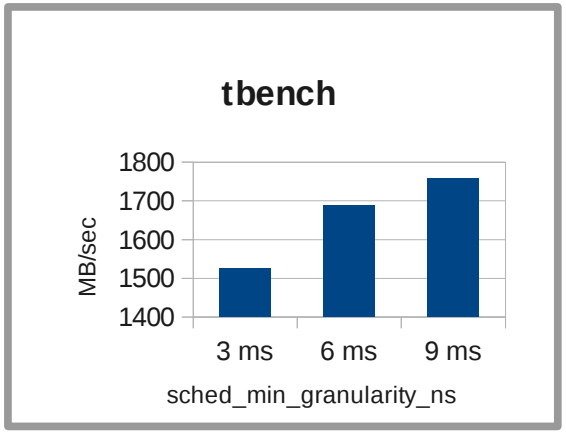
- Server and clients were running in the same machine
- Number of clients: 50
- Gave the best throughput at 24 ms
- Variation upto +/- 10%
- Matches with the equation

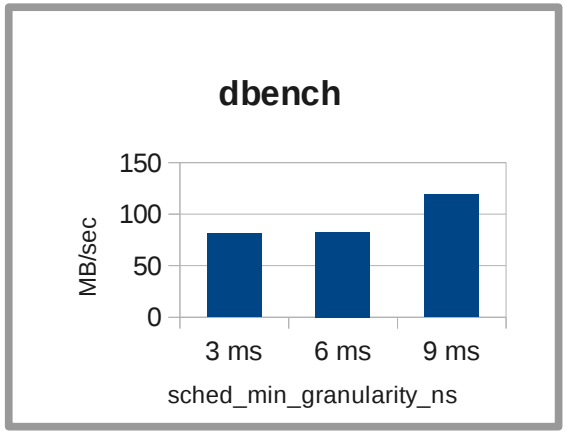
  sched_latency =

  6ms * log(nr_cpus)

Impact of *sched_min_granularity_ns*

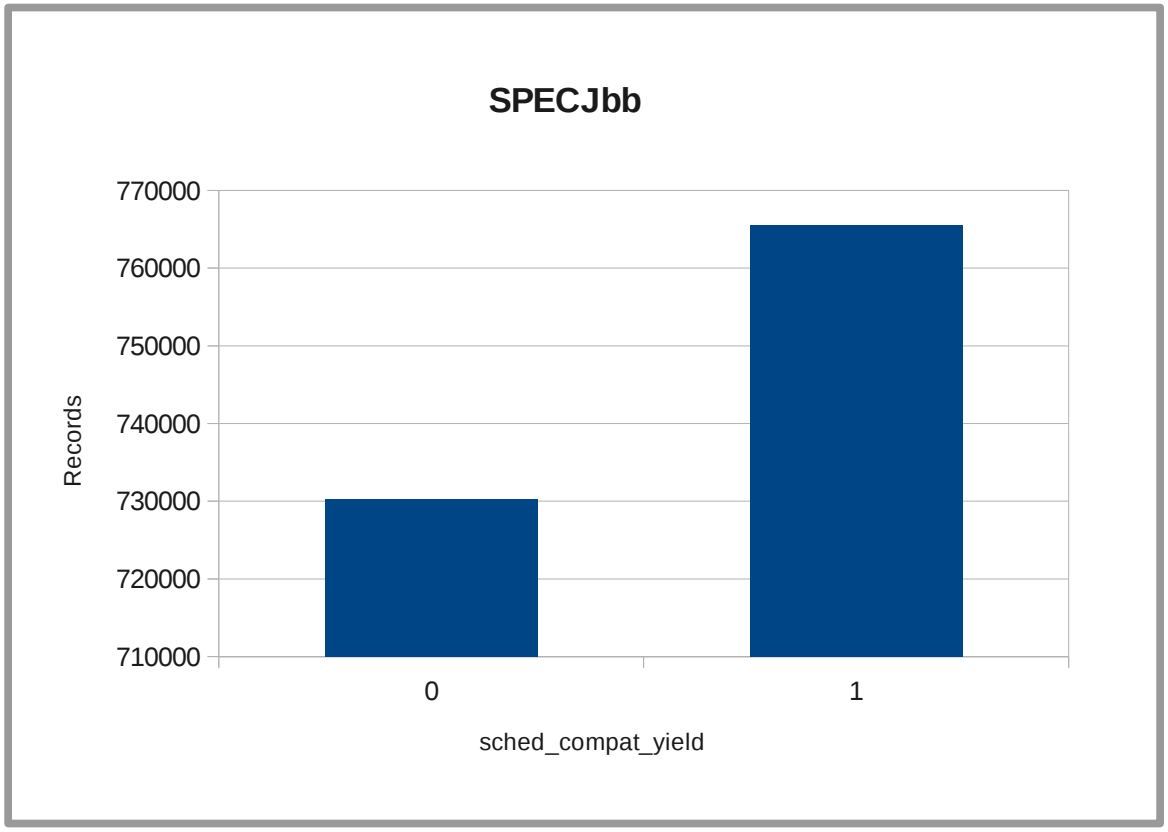






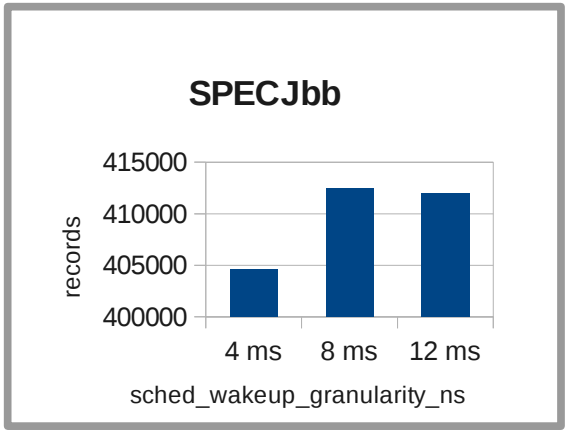
- Mixed workloads were used
- SPECJbb 32 warehouses
- Tbench 25 clients
- Dbench 25 clients
- Significant improvement in the performance
- Longer execution cycles helping the workloads
- Kernel: 2.6.35.fc14

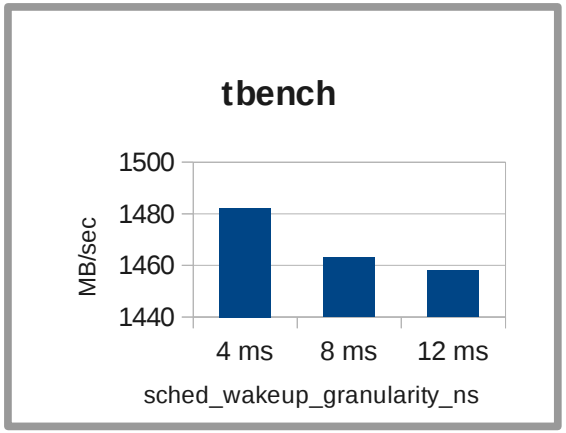
Impact of *sched_compat_yield*

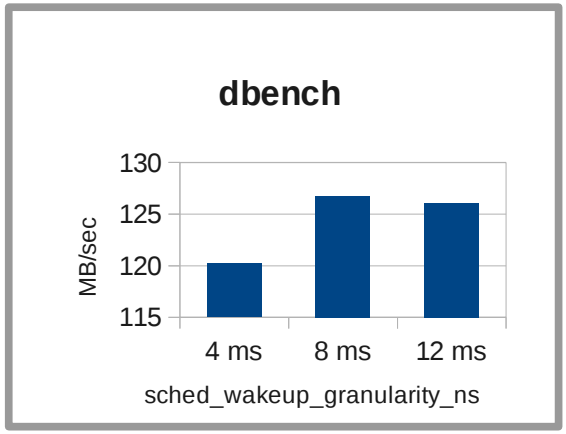


- **Kernel: 2.6.35.6-45.fc14**
- **SPECJbb**
  - 8 Instances with 4 warehouses each
  - Total 32 warehouses
- **+5% improvement**
- **Around +15% improvement was observed in a 2.6.32 based kernel**
- **sched_compat_yield is no longer present in mainline kernels**

Impact of *sched_wakeup_granularity_ns*







- Mixed workloads were used
- SPECJbb 32 warehouses
- Tbench 25 clients
- Dbench 25 clients
- Significant improvement in the performance for SPECJbb & dbench
- Tbench performance goes down as it is a client/server benchmark, which needs faster responses

References

- CFS documentation: Kernel/Documentation/sched-*
- CFS scheduler: kernel/sched_fair.c
- CPU bandwidth control: http://lwn.net/Articles/452584/
- Cgorups: Kernel/Documentation/cgroups/
- My blog: http://krm4linux.blogspot.com/

Acknowledgments

- Prasad Krishnan
- Vaidyanathan Srinivasan
- Bharata B Rao
- Nikunj A Dadhania
- Srivatsa V
- Larry B. Kessler
- Naren A Devaiah

Questions / Discussions

Thank You!

Stay current on Linux and Open Virtualization at IBM

Linux

Follow us on Twitter:
@ Linux_at_IBM

Like us on Facebook:
Linux at IBM

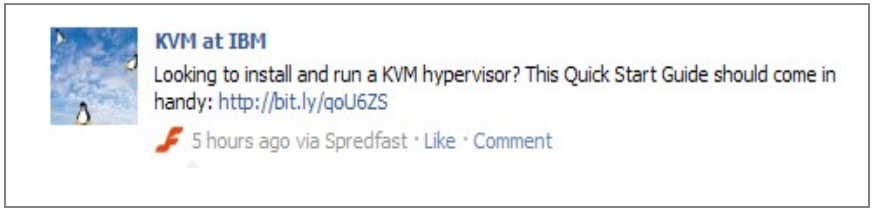
www.ibm.com/linux

Open Virtualization & KVM

Follow us on Twitter:
@ OpenKVM

Like us on Facebook:
KVM at IBM

www.ibm.com/systems/kvm



Legal Statement

- Copyright International Business Machines Corporation 2011
- Permission to redistribute in accordance with Linux Foundation LinuxCon 2011 submission guidelines is granted; all other rights reserved.
- This work represents the view of the authors and does not necessarily represent the view of IBM.
- IBM, IBM logo, ibm.com are trademarks of International Business Machines Corporation in the United States, other countries, or both.
- Intel is a trademark or registered trademark of Intel Corporation or it subsidiaries in the United States and other countries.
- Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
- Other company, product, and service names may be trademarks or service marks of others.
- References in this publication to IBM products or services do not imply that IBM intends to make them available in all countries in which IBM operates.

Legal Statement

- **INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you. This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.**