

Sistemas, Virtualización y Seguridad

Jesús Durán Hernández

Rebeca Bárcena Orero

Práctica 1: Caracterización del rendimiento

- Tarea 1: SPECcpu
 - Entero y flotante
 - 32 y 64 bits
 - Peak
 - Perf
- Tarea 2:
 - SPECweb
 - Apache
 - SPECjbb
 - Oracle JVM
 - OpenJDK

Tarea 1: SPECcpu

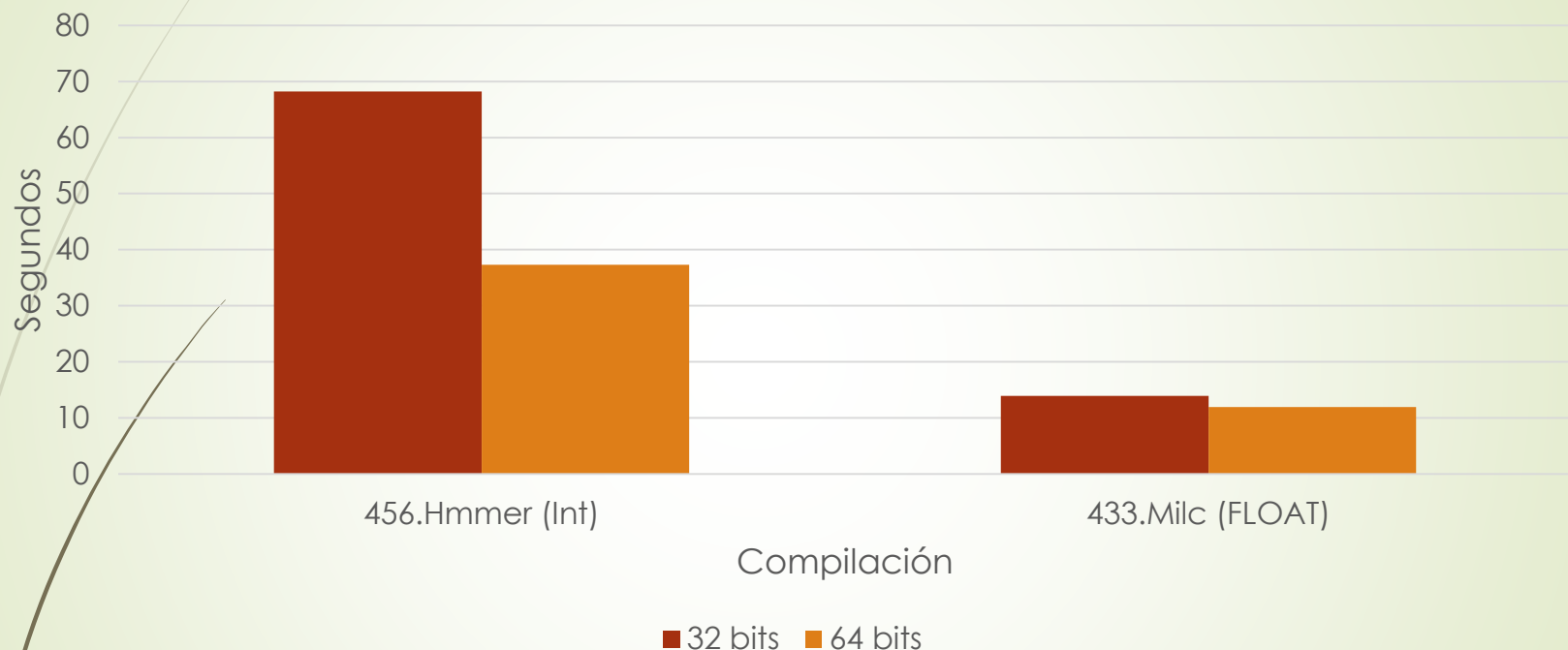
- Configurar el entorno para C, C++ y Fortran
- Crear una configuración propia (tiene que estar en la carpeta config)
- Compilar el benchmark runspec y `-action=build`
- Resolver los errores
 - 483.xalancbmk: `CXXPORTABILITY=-DSPEC_CPU_LINUX -include cstdlib -include cstring` en el archivo de configuración
 - 481.wrf: comentar la línea `wrf_data_header_size = 8`
- Ejecutar las pruebas con el comando runspec y las opciones

Tarea 1: SPECcpu

- Para 64 no hay que modificar el fichero de configuración
- Para 32 se añade el flag “-m32” en el fichero de configuración
- Para Peak o FDO hay que añadir
 - default=peak=default=default:
 - PASS1_CFLAGS = -prof_gen
 - PASS2_CFLAGS = -prof_use
 - PASS1_CXXFLAGS = -prof_gen
 - PASS2_CXXFLAGS = -prof_use
 - PASS1_FFLAGS = -prof_gen
 - PASS2_FFLAGS = -prof_use
 - PASS1_LDFLAGS = -prof_gen
 - PASS2_LDFLAGS = -prof_use
 - notes0005 = C,C++,Fortran peak flags: -fast +FDO

Tarea 1: SPECcpu

Comparación de compilaciones



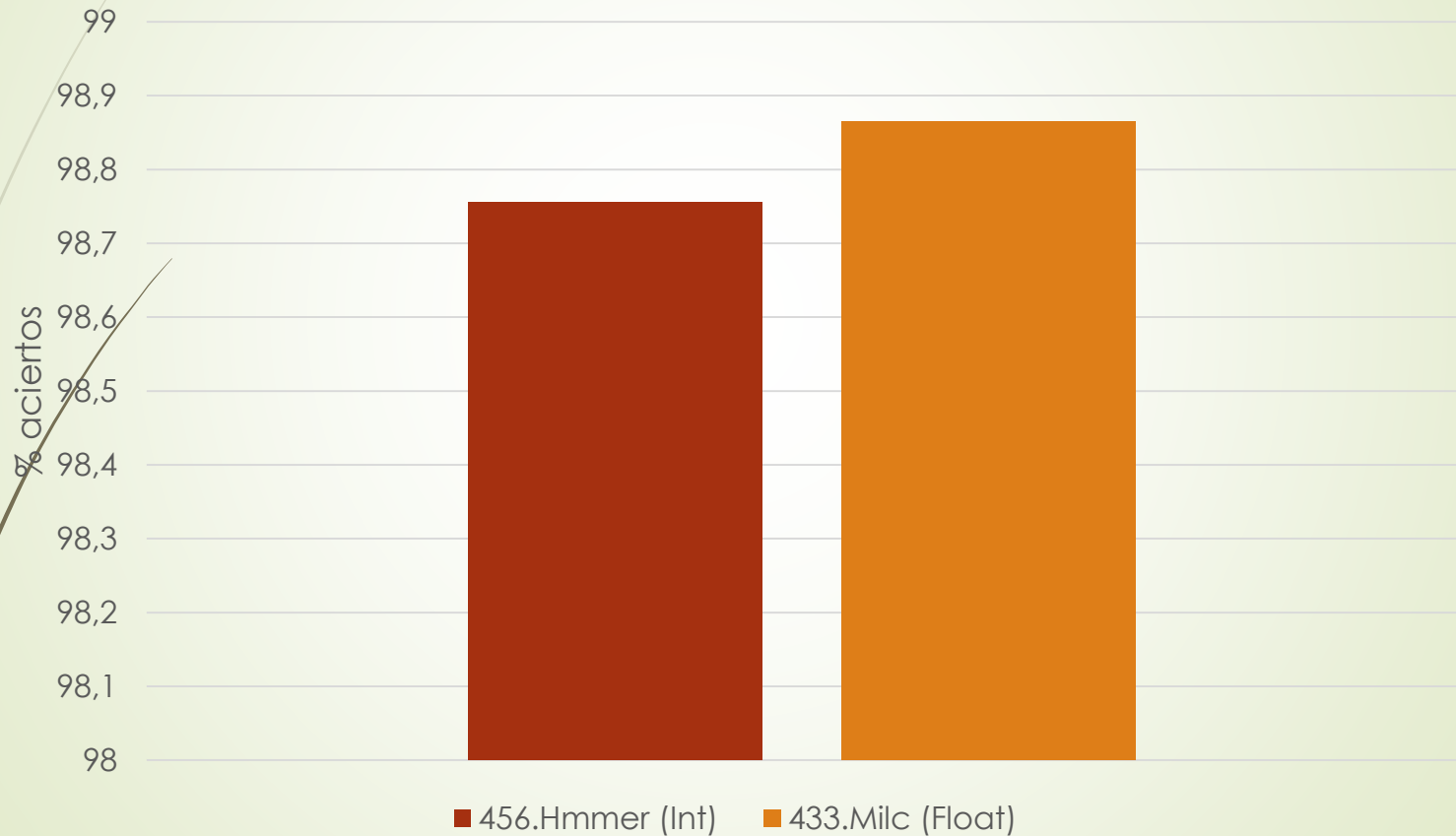
- El milc es memory bound y la mayor parte del tiempo de ejecución lo pasa realizando operaciones de memoria, por lo que no tiene tanta diferencia de ejecución entre sus versiones
- El hmmer es CPU bound, por lo que pasa gran parte de su tiempo de ejecución en la CPU y, es por eso, que los tiempos en sus cambios son tan significativos

Tarea 1: SPECcpu

- Para analizar parámetros del rendimiento se ha usado el comando perf
- Consultar el manual de Desarrolladores de Software para arquitecturas Intel 64 e IA-32
 - Branch Instruction Retired
 - Branch Misses Retired
 - UnHalted Core Cycles
 - Instruction Retired
- A partir de ellos obtenemos
 - Eficiencia predictor de saltos
 - IPC

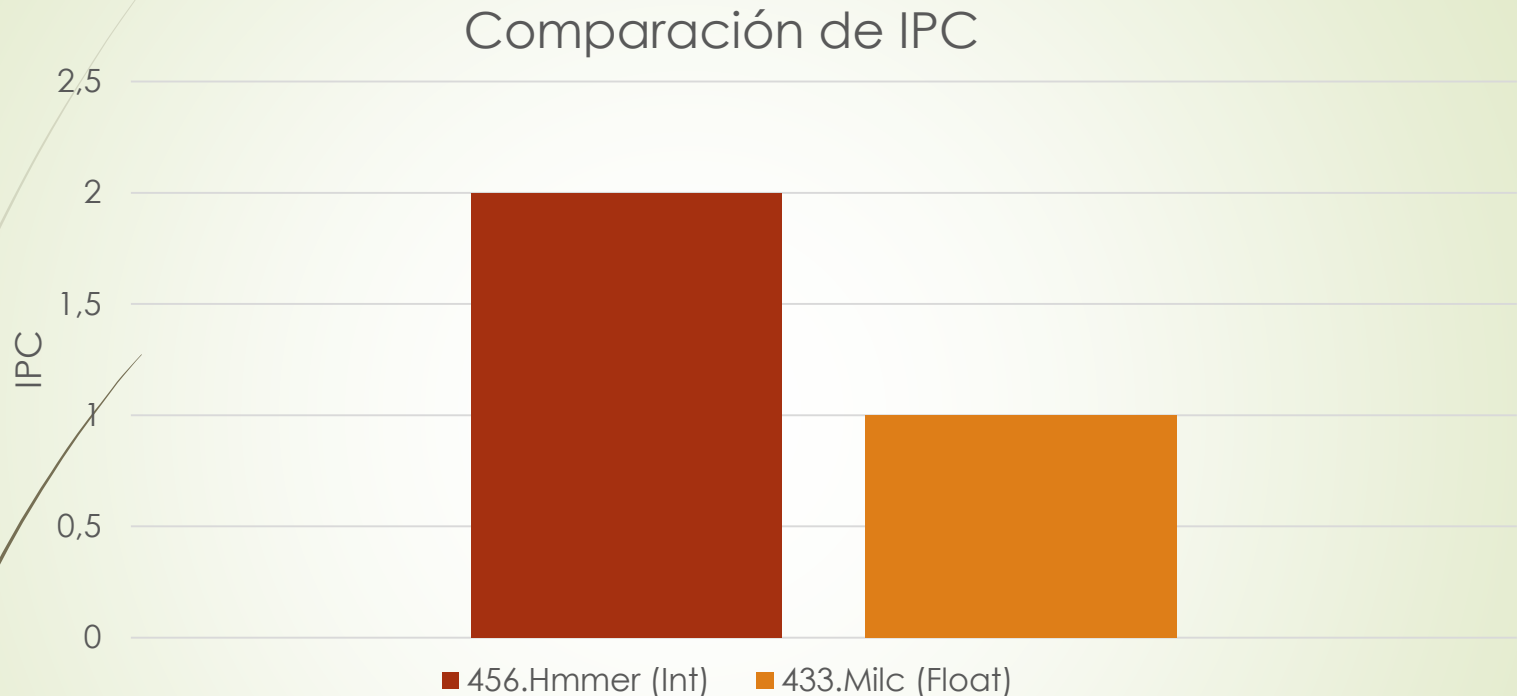
Tarea 1: SPECcpu

Comparación de la eficiencia del predictor de saltos



➡ El predictor de saltos es prácticamente igual

Tarea 1: SPECcpu

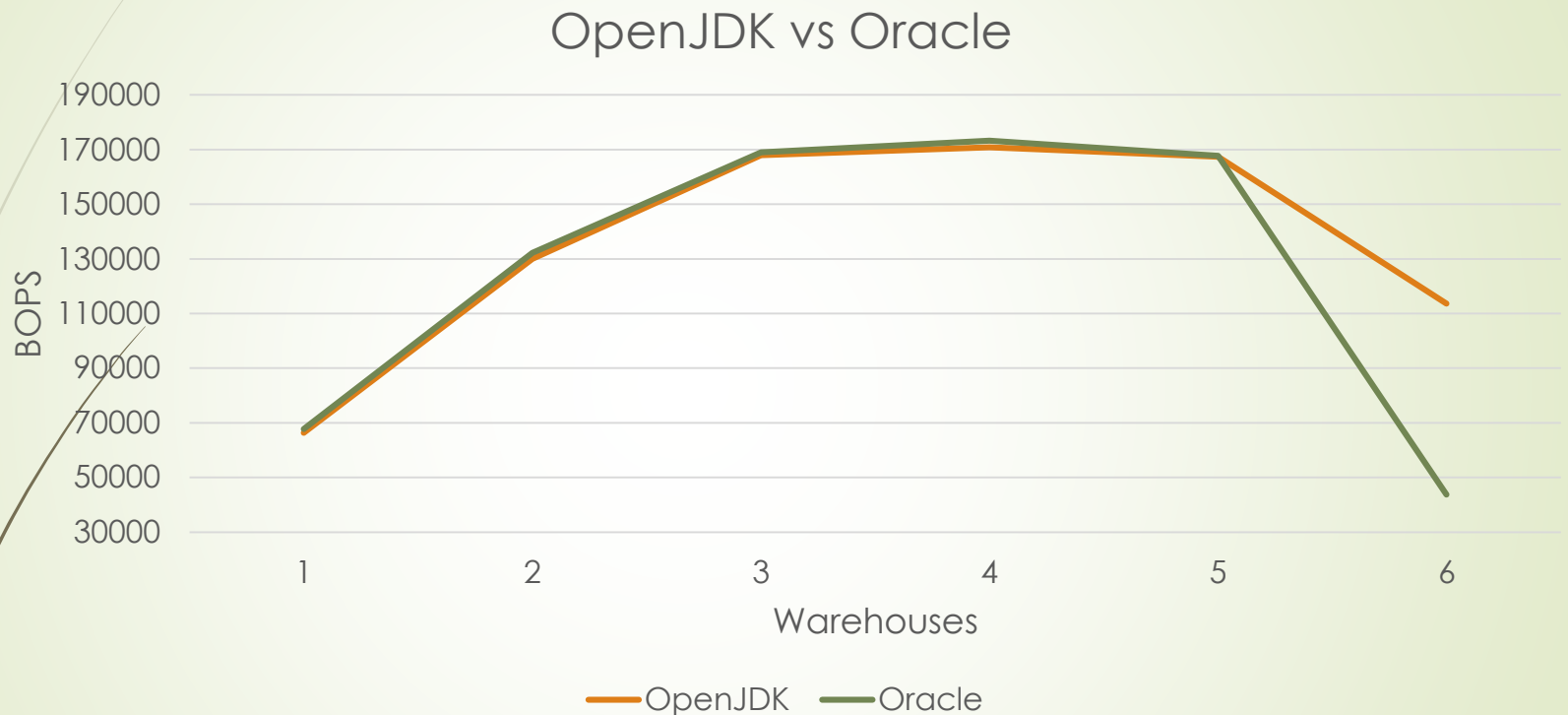


- Como el Milc pasa más tiempo en memoria, sólo hace una instrucción por ciclo
- El Hmmer, en cambio, como pasa la mayor parte del tiempo en la CPU, hace el doble de instrucciones por ciclo

Tarea 2: SPECjbb

- El objetivo es evaluar el rendimiento de Java en el servidor
 - OpenJDK
 - Oracle JVM
- Preparación
 - Descargar Benchmark
 - Descargar los dos Java
 - Establecer las variables de entorno
 - Modificar los ficheros de configuración (SPECjbb.props y SPECjbb_config.props)
- Problemas
 - Si sale un error por falta de memoria, aumentarla en el archivo SPECjbb_config.props

Tarea 2: SPECjbb



- Los valores hasta el quinto warehouse son muy similares
- El pico más alto en ambos es en el cuarto warehouse (nuestra máquina tiene 4 núcleos)
- A partir del cuarto warehouse empiezan a descender
- En Java de Oracle la caída en el sexto warehouse es mucho mayor que en OpenJDK

Tarea 2: SPECweb

- El objetivo es evaluar el rendimiento de los servidores destinados a conexiones de red
- Descargar y descomprimir el benchmark
- Realizar una instalación completa
- Descargar Apache2, PHP5 y fastcgi
- Dentro de SPECweb hay tres benchmarks:
 - Ecommerce
 - Support
 - Banking

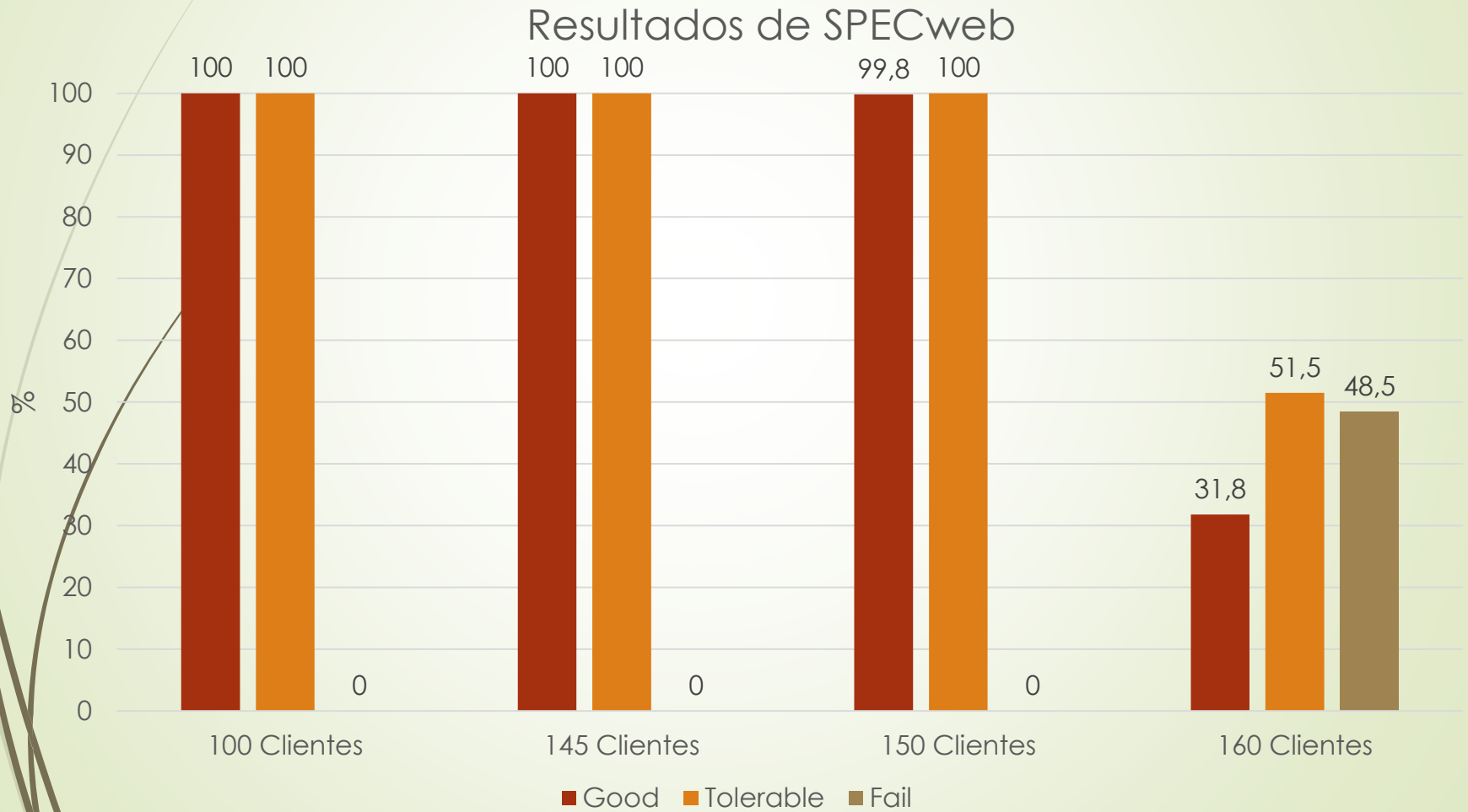
SPECweb Support

- Hay tres módulos
 - Clientes
 - BeSim
 - Web server
- Modificar el archivo de configuración
 - Tiempo de rampup
 - Tiempo de rampdown
 - Tiempo de ejecución
 - Número de iteraciones
 - Número de clientes o sesiones
 - Configurar IPs y puertos
- Se cambia el número de sesiones y se ejecutan los scripts Wafgen para generar el contenido

SPECweb Support

- Para solucionar fallos
 - Copiar los scripts de la carpeta scripts de SPECweb en /var/www
 - Dar permisos de escritura al usuario apache
 - Crear el archivo init_vars.php (el Benchmark escribe valores de variables)
 - Comprobar que el Wafgen se ha generado acorde al número de clientes

SPECweb Support



Práctica 2: Xen

- Tarea 1: Configuración de Xen
 - Pre-instalación
 - Instalación
 - Post-instalación
- Tarea 2: Creación de máquinas virtuales
- Tarea 3: Gestión de domUs
 - Añadir a los Xen store
 - Instalar kernel de backports
 - Eliminar snapshots y clones

Tarea 1: Configuración de Xen

- Pre-instalación
 - Comprobar si VT-x está activado
 - Crear partición de 40 GB
 - Dentro de ella, un volumen lógico
- Instalación
 - Descargar Xen 4.1 desde repositorio
 - Configurar Xen por defecto en el GRUB
 - Modificar el archivo de configuración e indicar cuál es la entrada por defecto
- Post-instalación
 - Configurar la red para Xen (descomentar *network-script network-bridge* en el archivo de configuración)
 - Activar X11 forwarding en el dom0

Tarea 2: Creación de máquinas virtuales

- Crear la instalación de Debian 7 (Wheezy) en un domU
 - Sobre un volumen lógico de 5 GB
- Crear un archivo de configuración
- Crear la VM a partir del archivo y sobre el volumen lógico
- Descargar SPECcpu2006 dentro del domU
- Crear un clon del domU usando snapshot
 - `lvcreate -L 15Gi -s -n hvm2 /dev/vol1/hvm`
 - Se crean dos volúmenes lógicos, porque más adelante se requiere de otro clon

Tarea 3: Gestión de domUs

- Añadir domUs a xend-store
 - Instalar Python-lxml
 - `xm new _.cfg`
- Instalar kernel desde back-ports
 - Añadirlo a `/etc/apt/sources.list`
 - Si no existe `/etc/apt/preferences`, crearlo
 - Descargar back-ports
 - Descargar el kernel
- Eliminar snapshots y clones
 - Eliminar volúmenes lógicos
 - Eliminar archivos de configuración

Práctica 3: Optimización del rendimiento

- Tarea 1
 - Máquina HVM
 - Máquina PV
- Tarea 2: Comparación de SPECcpu en distintas máquinas
- Tarea 3: QoS para SPECjbb

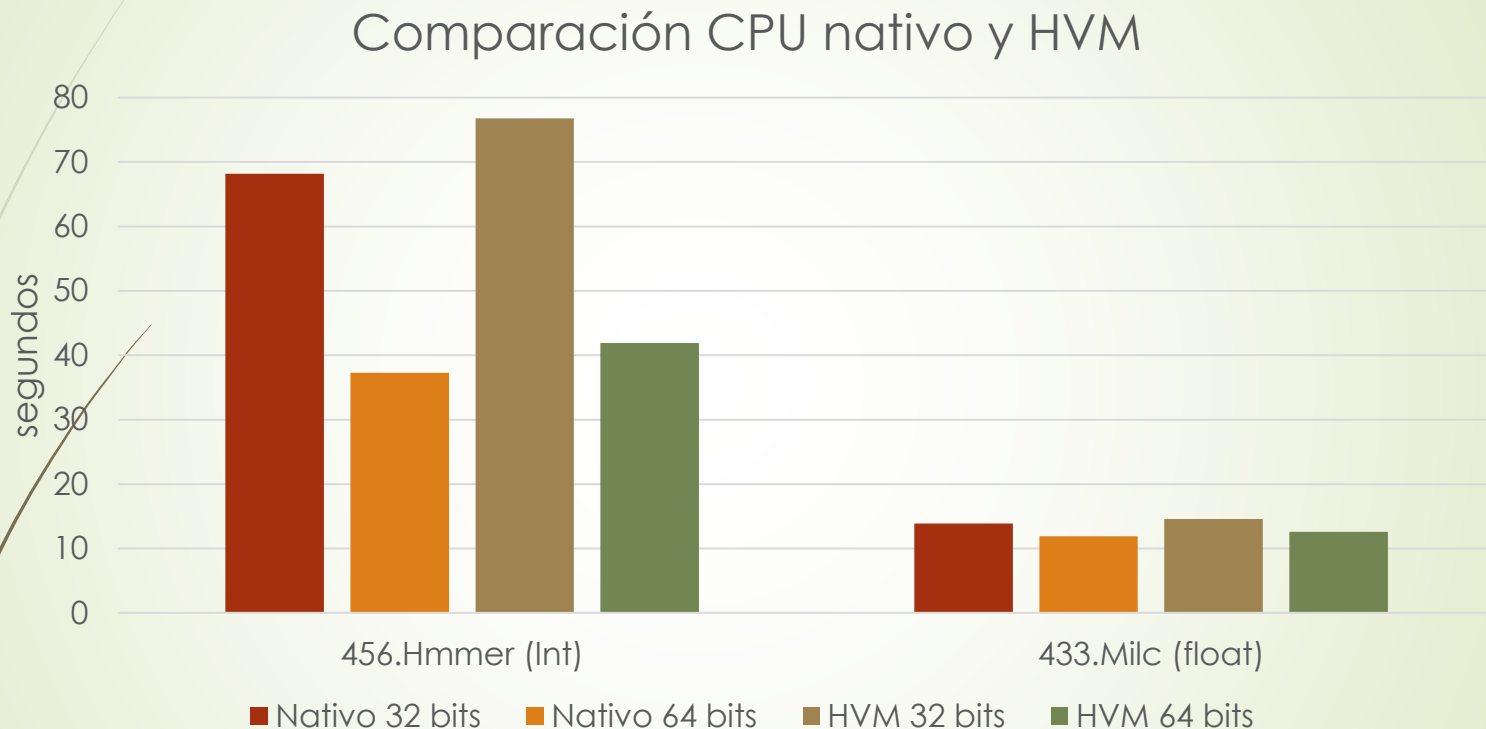
Tarea 1: Máquina HVM

- Crear una configuración para la HVM
- Crear el volumen lógico de 15 GB
- Descargar en el domU SPECcpu2006, SPECjbb2005 y SPECweb2005
- Al principio se usaba el front-end gráfico del dom0 para acceder al domU (*vncviewer*)
- Después se configuró la consola para usar la de Xen
 - Reconfiguración de la distribución del teclado
 - *dpkg-reconfigure keyboard-configuration*
 - *service keyboard-setup restart*
 - Hacerlo permanente modificando el fichero */etc/default/keyboard*

Tarea 1: Máquina PV

- A partir de una imagen de HVM se crea una PV
 - Añadir la consola hvc0 en /etc/inittab
 - Habilitar los drivers de PV
- No se pueden arrancar simultáneamente
- Problema con el inicio de sesión
 - Se harán las pruebas con HVM
 - Se descarga SPECcpu y SPECweb

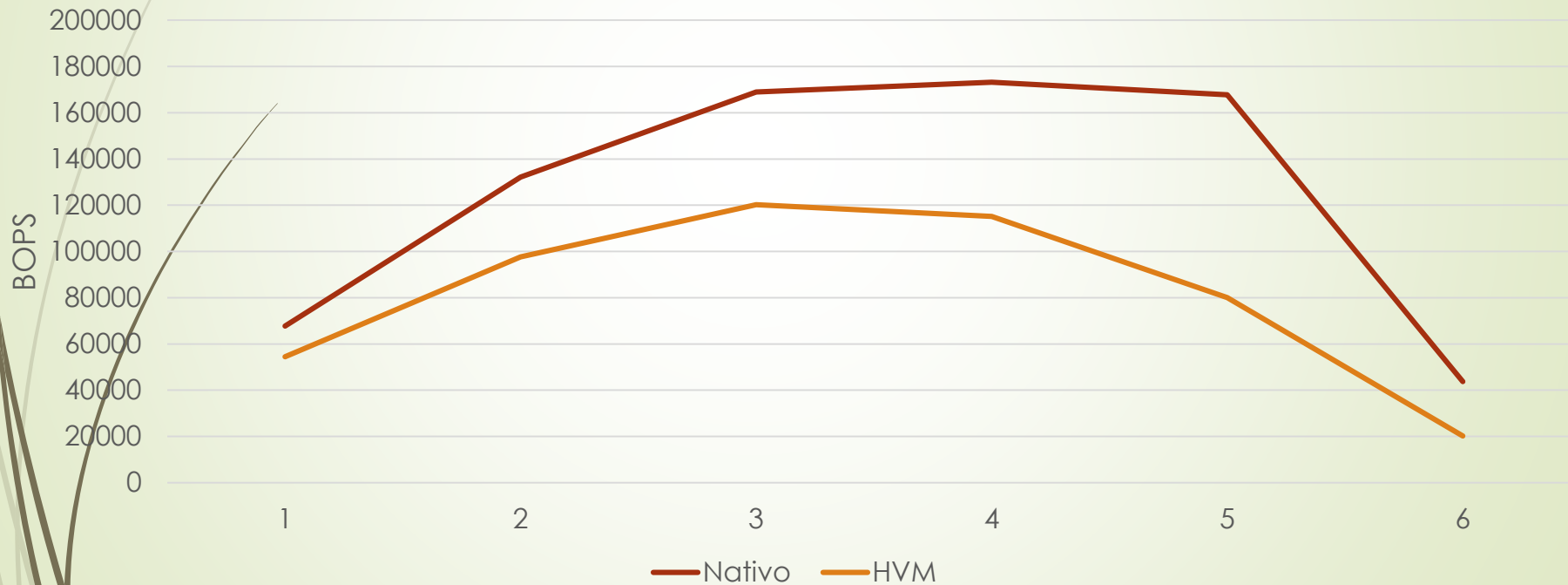
Tarea 2: Comparación de SPECcpu en distintas máquinas



- Se comparan Nativo y HVM, porque PV no funciona
 - Si funcionase, se comprobaría sólo con PV porque como no usa VT-x da igual si está habilitado o no

Tarea 3: QoS para SPECjbb

SPECjbb en nativo y HVM



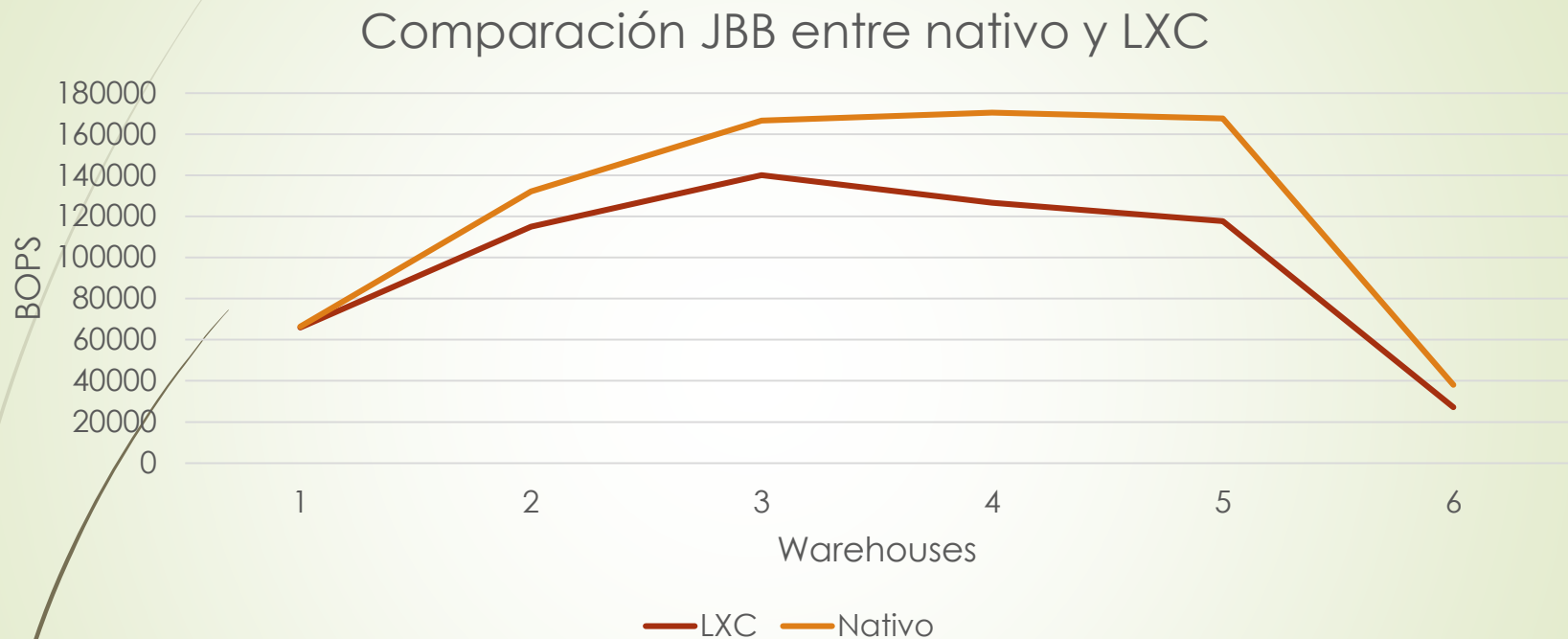
Práctica 4: Contenedores y clouds públicas

- Tarea 1: LXC
 - QoS para SPECjbb
- Tarea 2: Docker
 - SPECweb
 - Comparación con Xen
 - Dockerhub
- Tarea 3: Google Cloud
 - SPECweb

Tarea 1: LXC

- Instalar wheezy-backports
- Añadir backports a los repositorios (*/etc/apt/sources.list*)
- Crear contenedor con la última versión de Debian
- Poner red al contenedor (modificar el fichero */var/lib/lxc/my_jessie*)
- Descargar en el contenedor SPECjbb
- Hacer un clon del contenedor
- Jugar con las prioridades y número de CPUs (cgroups) y probar SPECjbb con Java de Oracle para conseguir un 75% del rendimiento en nativo

Tarea 1: LXC



- 3 CPUs al LXC, otra compartida con el clon
- Más del 75% en los 3 primeros casos
- Desciende al 70-75%
- Los 3 últimos casos tienen un 72,19% de rendimiento global
- Teniendo en cuenta todos los casos, un 79,91% global

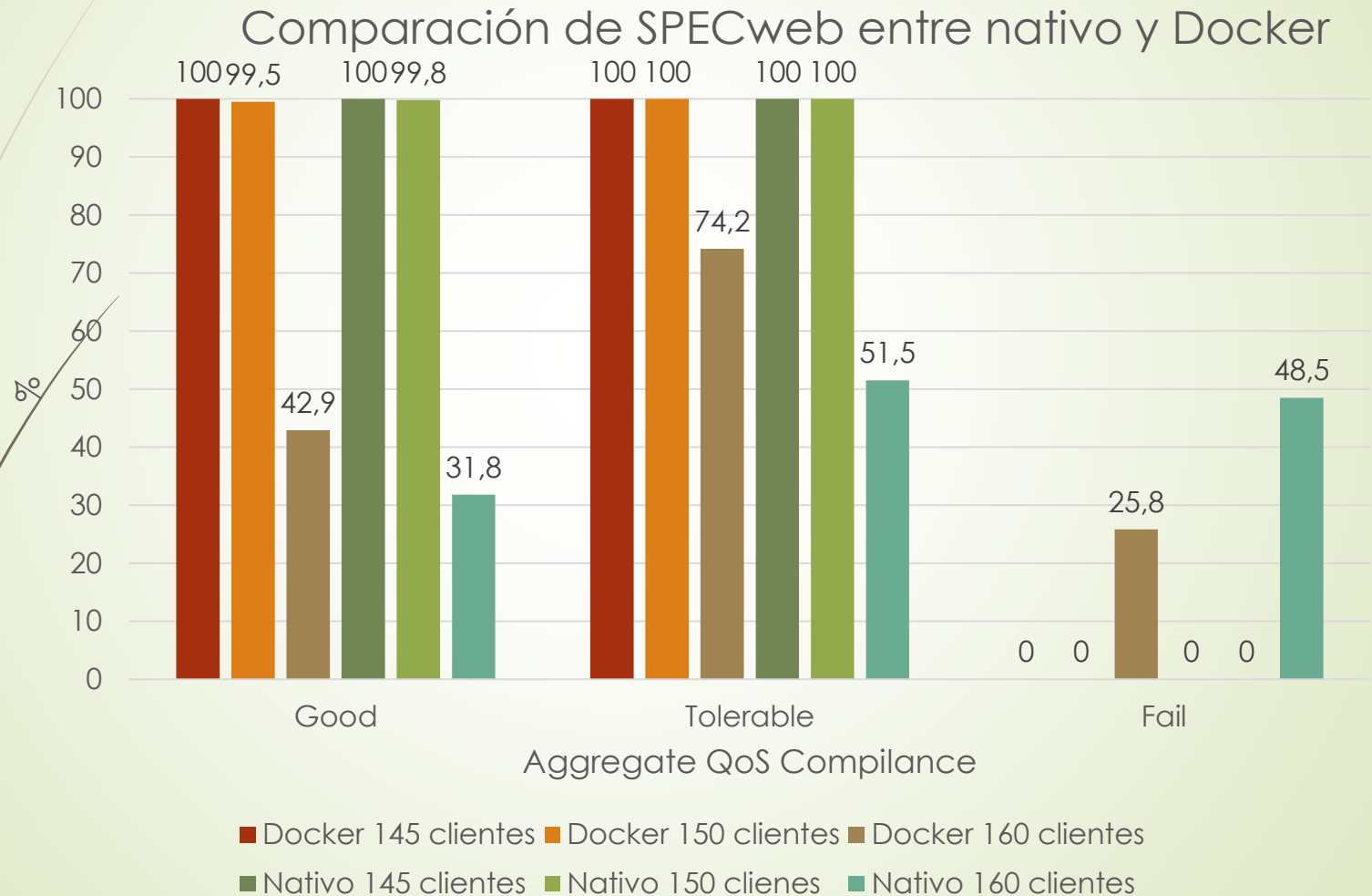
¿Cómo imitar la política de planificación en LXC?

- Habilitar CAP_SYS_NICE en el contenedor
 - Aumentar procesos y prioridades de threads
- Esta capacidad permite la llamada `sched_setscheduler()` que es la necesaria para poner SCHED_RR.
 - Fija la política de planificación y parámetros para un thread
- Añadir en el archivo de configuración la llamada `lxc.cap.keep=sys_nice`.
 - El contenedor conserva esa opción
- Las aplicaciones tienen que tener también CAP_SYS_NICE o ser ejecutadas desde root (en cuyo caso ya lo tienen).

Tarea 2: Docker

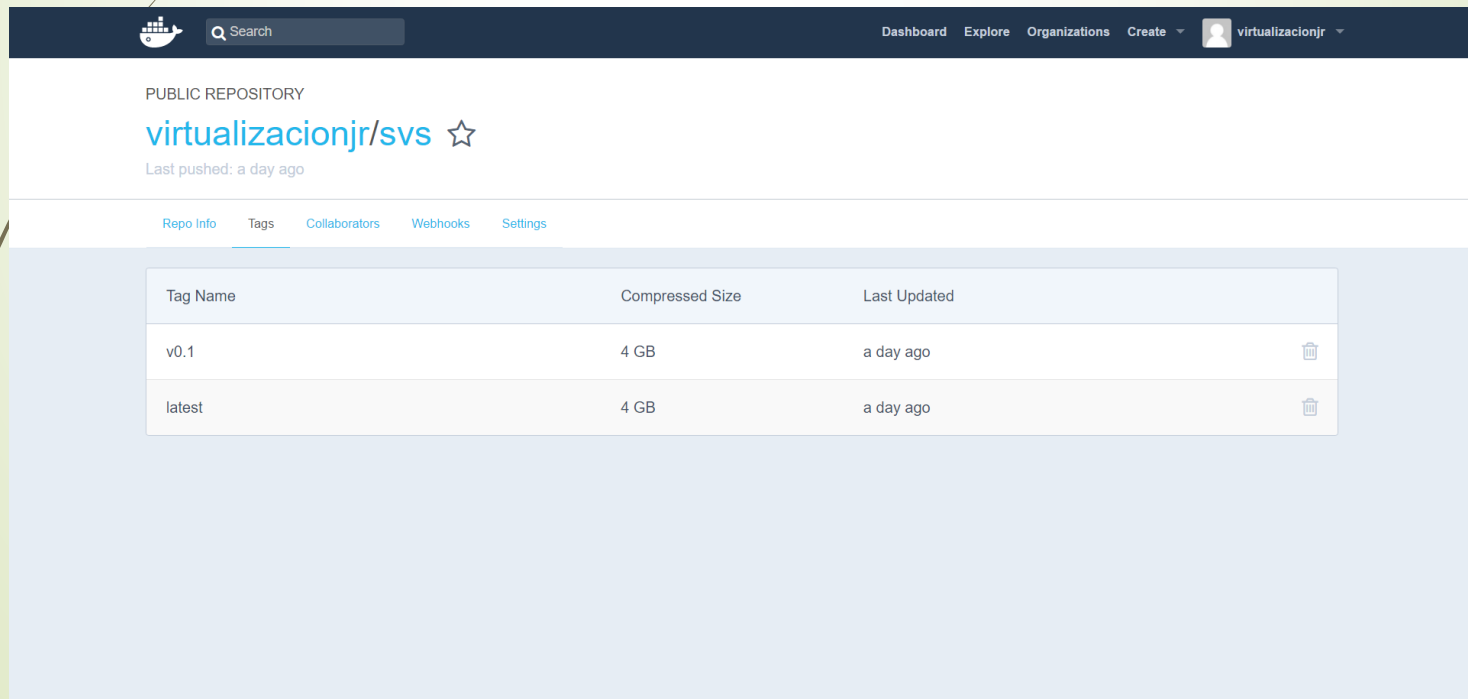
- Uso de Dockerfile para crear el primer contenedor
 - Descarga SPECweb y lo descomprime
 - Descarga e instala la versión de Java 1.5
 - Establece las variables de entorno
- Se crean otros dos contenedores a partir del Dockerfile
- Al crear los contenedores se mapean los puertos 22, 80, 81, 1099
- Probar SPECweb con varios contenedores
 - Uno se configura como Webserver, otro como Besim y otro como Client (modificar Test.config de cada uno con las IP correspondientes)

Tarea 2: Docker



Tarea 2: Docker

- Creación de cuenta en Dockerhub y un repositorio
- Añadir tag al contenedor
- Subir contenedor a Dockerhub



The screenshot displays the Docker Hub interface for a public repository named 'virtualizacionjr/svs'. The repository was last pushed 'a day ago'. The 'Tags' tab is selected, showing a table with two tags: 'v0.1' and 'latest', both with a compressed size of 4 GB and updated 'a day ago'. The interface includes a search bar, navigation links (Dashboard, Explore, Organizations, Create), and a user profile for 'virtualizacionjr'.

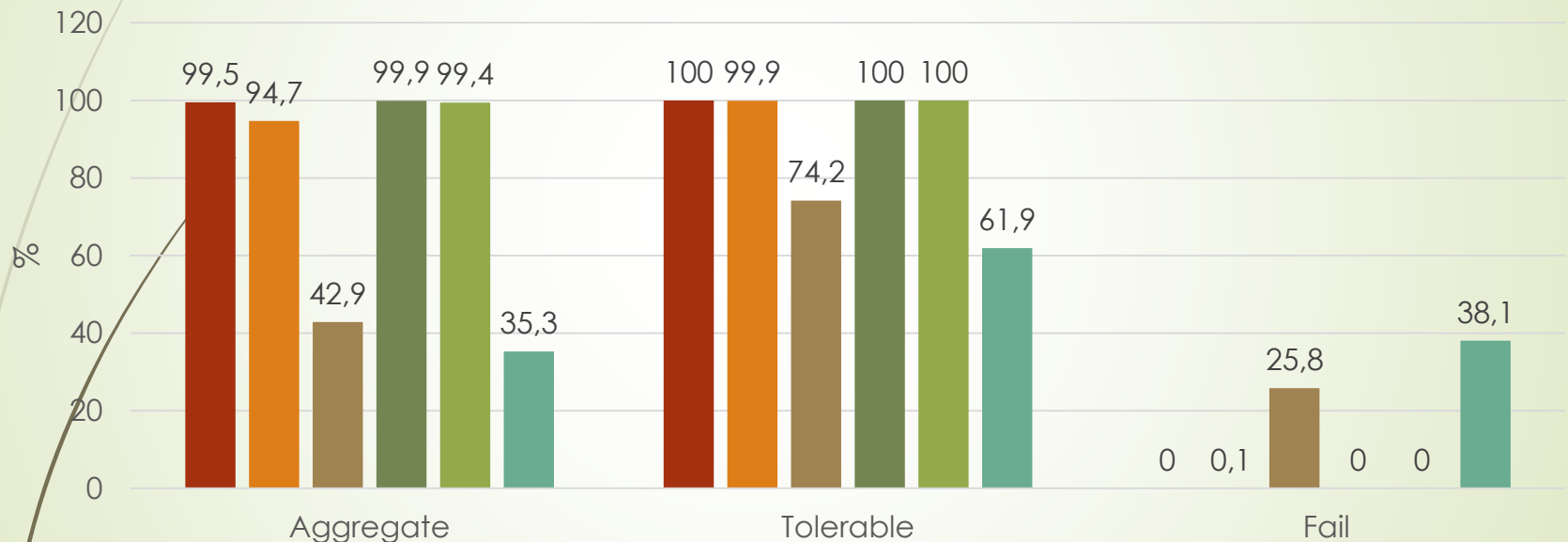
Tag Name	Compressed Size	Last Updated
v0.1	4 GB	a day ago
latest	4 GB	a day ago

Tarea 3: Google Cloud

- Crear cuenta en Google Cloud Platform
- Crear máquina virtual
 - Tipo g1-small
 - Una CPU
 - 1,7 GB de memoria
- Descargar Docker en la VM
- Descargar el contenedor de Dockerhub
- Descargar y configurar el SPECweb

Tarea 3: Google Cloud

Comparación de Docker y Google



Aggregate QoS Compliance

■ Docker 150 clientes ■ Docker 155 clientes ■ Docker 160 clientes
■ Google 150 clientes ■ Google 155 clientes ■ Google 160 clientes

Herramientas Utilizadas

- Drive
 - Para tomar notas en común
- Github
 - Como repositorio para resultados y archivos de configuración
 - <https://github.com/rebecabarcena/SVS>
- Dockerhub
 - Para subir los contenedores
 - <https://hub.docker.com/r/virtualizacionjr/svs/>

Jesús Durán Hernández
Rebeca Bárcena Orero