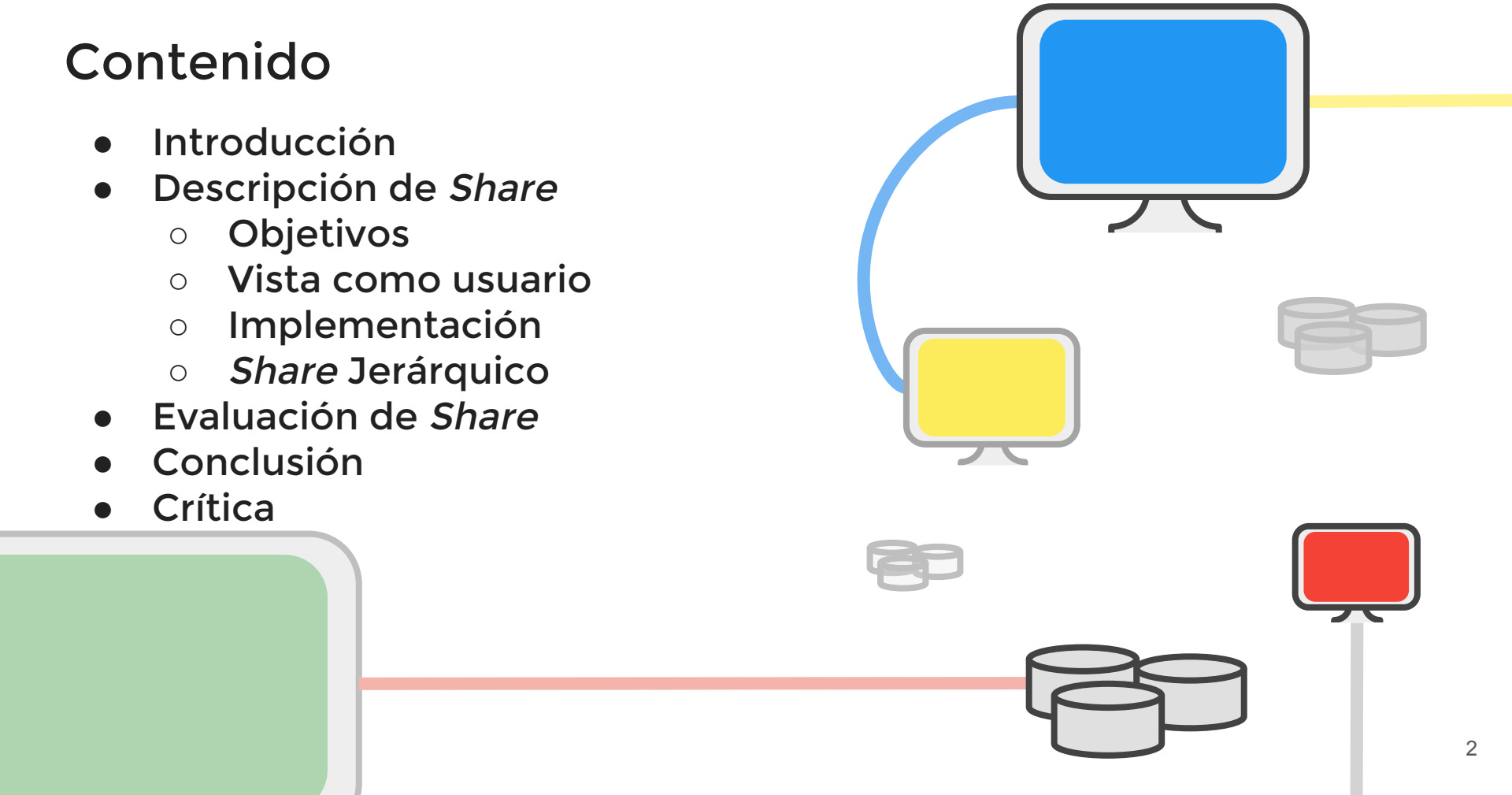
A fair share scheduler

Kay, Judy, and Piers Lauder. "A fair share scheduler." *Communications of the ACM* 31.1 (1988): 44-55.

María Fernanda Mora Alba
22 septiembre, 2016

Contenido

- Introducción
- Descripción de *Share*
  - Objetivos
  - Vista como usuario
  - Implementación
  - *Share* Jerárquico
- Evaluación de *Share*
- Conclusión
- Crítica



Introducción

Contexto

- Ilusión de muchos CPUs: Virtualización
- ¿Cómo?
  - Mecanismos de bajo nivel (cambio de contexto)
  - Políticas de alto nivel (programación de procesos)
- Programación de procesos:

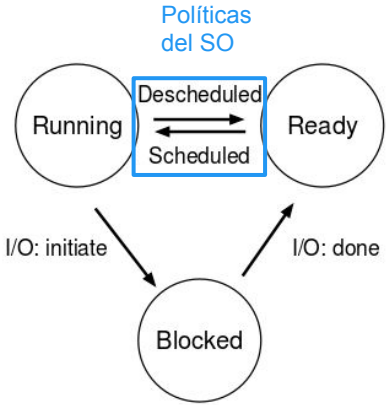


Figure 4.2: **Process: State Transitions**

Políticas del SO

Time	$Process_0$	$Process_1$	Notes
1	Running	Ready	
2	Running	Ready	
3	Running	Ready	$Process_0$ initiates I/O
4	Blocked	Running	$Process_0$ is blocked,
5	Blocked	Running	so $Process_1$ runs
6	Blocked	Running	
7	Ready	Running	I/O done
8	Ready	Running	$Process_1$ now done
9	Running	–	
10	Running	–	$Process_0$ now done

Figure 4.4: **Tracing Process State: CPU and I/O**

Fuente: Operating Systems: Three Easy Pieces, Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau, Arpaci-Dusseau Books. March, 2015

¿Cómo son esos programadores?

- 2 enfoques básicos:
  - Trabajos más cortos primero
  - Alternancia entre trabajos
- Programador de tiempo compartido: cada proceso obtiene los mismos recursos
- Programador “justo”: más recursos a los procesos más importantes
- 1975: programadores justos con los usuarios.
- Fair share scheduler (1988)

Desarrollo de *Share*

- Escenario:
  - 1000 estudiantes
  - Muchas clases
  - Picos de trabajo frecuentes
- Programador típico de Unix:
  - Usuario con más procesos, más recursos
  - Sin registro de uso histórico
  - Pobre respuesta para todos ante picos
  - Todas las tareas son igual de importantes

Injusto para el usuario

Share

- Solución parcial: mecanismo de presupuesto fijo por usuario.
- Problema: sólo sirve para recursos como espacio en disco, impresiones, # conexiones por período - > *Share*
- El artículo se enfoca en *Share* para asignar CPU
- *Share* puede servir para asignar otro tipo de recursos
- *Share* puede usarse en otros escenarios
- *Share* puede extenderse a grupos de usuarios

Descripción de Share

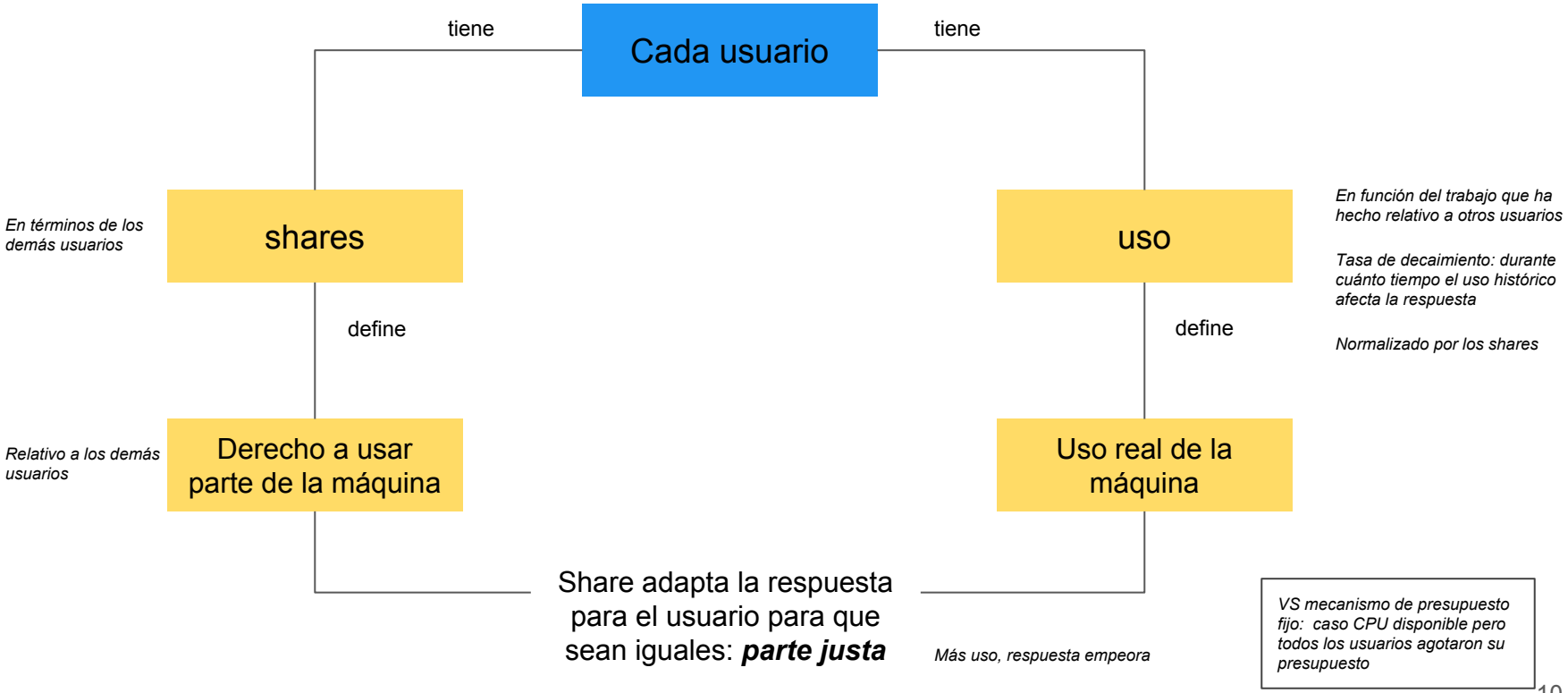
Objetivos de *Share*

- Idea: Cada usuario tiene derecho a *su parte justa* (relativo a los otros usuarios) de la máquina
  - Nadie obtiene más de su parte justa en el largo plazo
  - La máquina puede ser bien usada
- Requerimientos:

Parezca justo
Entendible
Predecible
Sea flexible a necesidades especiales
Responda bien a picos

Dé respuesta razonable
Evite inanición
Rápido
Poco overhead

Vista como usuario



Consideraciones (“administrativas”)

- Número de shares por usuario (crítico para ser justo)
- Los usuarios pueden cambiar la prioridad de sus procesos usando *nice* (nice>0 permite peores respuestas)
- Problema: la gente NO es buena por naturaleza -> procesos con nice grandes son más baratos
- Cobro diferenciado:
  - Por tipo de recurso (memoria, llamados a sistema, CPU)
  - Por hora del día: más caro en horas pico
- Cobros inestables son inaceptables (Kleijnen, 1968) -> el cobro cambia, pero la respuesta cambia poco a poco

Implementación (conceptual)

- Programador a nivel usuario:
  a. Actualizar el uso con todos los recursos consumidos por sus procesos (+tasa, +normalización, +cobros diferenciados).
  b. Actualizar el consumo de recursos por planeación, monitoreo.
- Su uso vs los usos de otros es un buen indicador de la respuesta que pueden esperar
- Esta info es visible al usuario
- Este programador es el más costoso

Implementación (conceptual)

- Programador a nivel proceso:
  a. Activación de nuevo proceso: (renuncia al CPU, interrupción, cesión de control al de mayor prioridad)

     Actualizar costos incurridos por el proceso actual

     Correr el proceso con la mayor prioridad

  b. Ajuste de prioridad del proceso actual:

     Disminuir la prioridad del proceso actual (uso, shares y procesos activos)

  c. Disminución de la prioridad de todos los procesos:

     Disminuir la prioridad de todos los procesos (más suave para procesos con prioridad baja)

Implementación (detalle)

Figure 2. Share implementation

Every t1 seconds: *user level scheduling*

for each user

Decay usage and update with costs incurred in last t1 seconds

$$usage_{user} = usage_{user} \times K1 + charges_{user}$$

Reset cost tally

$$charges_{user} = 0$$

t1 grande=4 secs (240X ajuste prioridad)

Every t2 seconds: *decay of process priorities*

cambian mucho, t2<t1
t2=1 sec

for each process

$$priority_{process} = priority_{process} \times K2 \times (nice_{process} + K3)$$

K2, K3 permten que nice=0 decaiga más rápido
evita muchos procesos con nice=0

Every t3 seconds: *priority adjustment*

$$priority_{current\_process} = priority_{current\_process} + \frac{usage_{current\_user} \times active\_processes_{current\_user}}{shares^2_{current\_user}}$$

t3<t2<t1

At each scheduling event: *current process selection*

actualizar costos
correr proceso más prioritario

$$charges_{current\_user} = charges_{current\_user} + cost_{event}$$

run process with lowest priority

Evita que si un usuario entra con usage=0 no consuma todos los recursos

Evita que un proceso consuma mucho y luego no pueda consumir más

Share jerárquico

Share simple:

- Asignación de shares por org= Share de la máquina
- Usuarios dentro de org son igualmente activos
- K1 sirve a nivel organización y es constante para todos los usuarios
- Mismos costos y parámetros (K2, t1, t2, t3)

Ajustes:

- Asignación de shares nivel usuario: como en el *Share* normal
- Recalcula shares a nivel organización
- Permite distintos niveles de actividad: el uso se recalcula al entrar y salir
- Tasas de decaimiento diferenciadas: complicado
- Otros parámetros: no se permite

Evaluación de Share

Evaluación (métricas)

Indicadores de monitoreo de operación:

- Uso real de recursos entre grupos
- Uso real de recursos entre usuarios
- Distribución efectiva de la parte justa
  - Gráfica de usuarios vs usos normalizados
- Frecuencia de consumo de recursos
- Share de los recursos a lo largo del tiempo

Indicadores no visibles:

- Tests de verificación para garantizar que *Share* preserva las relaciones correctas de shares, uso y número de procesos entre usuarios
- Simulación es difícil -> ¡experiencia real!

Evaluación (requerimientos)

- ✓ **Parezca justo:** asignación de shares corresponde a su parte justa + uso arriba de su parte justa se penaliza + percepción de justo
- ✓ **Entendible y predecible:** respuesta pobre indica que se excedió
- ? **Sea flexible a necesidades especiales:** es posible dar más shares por un periodo pequeño
- ✓ **Responda bien a picos:** penalización incremental por uso en hora pico

Evaluación (requerimientos)

? **Reparto de carga:** baja respuesta promueve estabilización de la carga

✓ **Respuesta razonable:** sólo permitir usar los recursos si es posible alcanzar buena respuesta

✓ **Evite inanición:** todos los procesos tienen recursos

✓ **Rápido:** cálculos costosos se calculan poco

✓ **Poco overhead:** menor overhead que otros programadores

Conclusión

Conclusión

- Usuarios perciben a *Share* como justo y atribuyen baja respuesta a su uso histórico
- *Share* es justo para usuarios y grupos: no se puede engañar al sistema
- *Share* da una predicción adecuada de la respuesta esperada
- *Share* da una idea al usuario del costo de los recursos
- Incentiva a balancear la carga
- Share ha probado ser útil en la práctica (académico), otros usos son posibles (acceso a servidor)

Crítica

Crítica

- Mal escrito
- No discute las decisiones “administrativas”, especialmente la asignación de shares que es crucial
- Esas decisiones pueden ser muy subjetivas y arruinar el propósito esencial de *Share*: ser justo
- ¿Trabajo futuro?
- *Share* debería incluir en su implementación el recálculo de las “decisiones administrativas”

Crítica

- No consideran exhaustivamente el *overhead*
- Usuarios con niveles de uso muy inestables pueden tener una respuesta indeseada
- ¿Cuál es el overhead real para que Share pueda funcionar? ¿Vale la pena?
- ¿No hay esquemas más sencillos?
- No define a detalle en qué otros escenarios puede ser usado *Share*
- No es escalable: complicado agregar nuevos usuarios

Gracias