Actualización de Servicios

Unidad 4
Gestión de SLA durante actualizaciones

Índice

Bibliografía

[KM90] Jeff Kramer, Jeff Magee: “The Evolving Philosophers Problem: Dynamic Change Management”. IEEE Trans. Software Eng. 16(11): 1293-1306 (1990)

[Li11] Wei Li: “Evaluating the impacts of dynamic reconfiguration on the QoS of running systems”. Journal of Systems and Software 84(12): 2123-2138 (2011)

[Li12] Wei Li: “QoS Assurance for Dynamic Reconfiguration of Component-Based Software Systems”. IEEE Trans. Software Eng. 38(3): 658-676 (2012)

Objetivos

- Revisar y utilizar los sistemas de predicción de rendimiento para tomar decisiones de reconfiguración adecuadas.
- Revisar cómo afecta la actualización de un servicio a sus niveles de calidad y a sus SLO.
- Aplicar técnicas adecuadas de planificación de actualizaciones para reducir el impacto de éstas sobre el SLA.

Índice

1. Introducción

- En la unidad anterior se estudiaron estrategias de reconfiguración / actualización de servicios elásticos...
  - La actualización se necesita en cualquier ciclo de vida.
  - En un entorno de computación en la nube...
    - Esa actualización debe respetar el SLA entre usuario y proveedor.
    - No basta con conocer las estrategias para realizar la actualización.
    - Debemos conocer también cómo planificarla y realizarla para cumplir el SLA.

1. Introducción

- Ninguno de los trabajos citados en las unidades anteriores...
  - Analizó cómo afecta la actualización a las métricas de rendimiento de un sistema o componente.
  - Analizó qué tipo de planificación sería recomendable utilizar durante la actualización.
- Analizaremos esos aspectos en esta unidad.

1. Introducción

- Wei Li estudia y evalúa en [Li11, Li12] diferentes mecanismos y estrategias para realizar una actualización, considerando sus efectos sobre la QoS.
- Organiza su análisis en función de los apartados siguientes:
  - Características de la QoS.
  - Organización y planificación.
  - Evaluación.
- Utilizaremos esos mismos apartados en las secciones siguientes.

Índice

2. Características de la QoS

- Li distingue cinco características de la QoS que deberían garantizarse en un proceso de actualización.
  - Tienen un impacto creciente sobre la QoS.
  - Son las siguientes:
    1. Consistencia global.
    2. Disponibilidad.
    3. Coexistencia y continuidad.
    4. Transferencia de estado.
    5. Minimización de sobrecargas.
- Las revisaremos a continuación, estudiando los mecanismos a utilizar para proporcionarlas.

2. Características de la QoS

- Cuando se consigan todas se podrá afirmar que la calidad de servicio estará garantizada.
- El orden en que se han listado en la hoja anterior...
  - Es el orden recomendable para garantizarlas.
  - No tiene sentido intentar garantizar la característica “i+1” si previamente no se han garantizado todas las anteriores (1..i).

2.1. Consistencia global

- Para definir la consistencia global se necesitan algunos conceptos previos:
  - Transacción: Una "*transacción*" es una operación distribuida que puede afectar a múltiples componentes de un sistema.
  - Transacción correcta: Una transacción se considera "*correcta*" cuando sus resultados esperados no se ven alterados por la actualización.

2.1. Consistencia global

- Transacción completa: Una transacción se considera “completa” si no es abortada durante la actualización.
- Estado crítico: El estado crítico de un sistema en funcionamiento es un conjunto de estados que se consideran esenciales para la corrección funcional del sistema.
- Si el sistema original utiliza un determinado protocolo para gestionar las transacciones y el sistema actualizado necesita un protocolo diferente...
    - Dependencia de componentes: Un conjunto de componentes es dependiente si todos ellos utilizan un mismo protocolo para gestionar transacciones y una transacción necesita utilizar todos esos componentes para ser procesada.

2.1. Consistencia global

- Un sistema de actualización se considera globalmente consistente cuando mantiene...
  - la corrección y compleción de las transacciones activas,
  - su estado crítico y...
  - la dependencia entre sus componentes.

2.2. Disponibilidad

- Un sistema de actualización garantiza la disponibilidad de servicio si el sistema permanece activo y acepta transacciones durante el intervalo de actualización.
- Para garantizar simultáneamente consistencia global y disponibilidad...
  - Se necesita restringir el inicio de una actualización a un estado seguro del sistema.

2.2. Disponibilidad

- Kramer y Magee [KM90] demostraron que la inactividad ("*quiescence*") es una condición suficiente para garantizar un estado seguro para iniciar la actualización.
  - Pero la inactividad implica un bloqueo previo al inicio de la actualización o durante ella (se aceptan transacciones pero no se procesan).
    - ¡Eso debería evitarse!!
    - Véase la próxima característica.

2.3. Coexistencia y continuidad

- Coexistencia: Cuando el nuevo subsistema se activa en una actualización antes de parar y eliminar el antiguo subsistema.
  - La coexistencia es un prerrequisito para la continuidad.
- Continuidad: El servicio de un sistema jamás se bloquea al realizar una actualización. Así, el sistema es siempre capaz de aceptar **y procesar** peticiones.

2.3. Coexistencia y continuidad

- Para proporcionar continuidad...:
  - Se requiere coexistencia.
    - La coexistencia implicará una arquitectura parcialmente compartida entre versiones de los componentes.
  - No podrá utilizarse inactividad.
    - Habrá que emplear una “gestión dinámica de versiones”.

2.3.1. Gestión dinámica de versiones

- Para implantarla se requiere:
  - Tres tipos de portadores de versión: sistema, componente, conector.
  - El sistema utiliza globalmente un número de versión.
    - Similar a un reloj lógico global.
  - Las transacciones reciben su versión cuando son iniciadas (la del sistema).
  - Habrá como máximo dos números de versión en los portadores de un sistema.
    - No habrá nuevas actualizaciones mientras no termine la actual.

2.3.1. Gestión dinámica de versiones

- Para implantarla se requiere (ii):
  - Tres fases en la actualización:
    - Instalación
      - La nueva versión instala sus componentes y conectores. Etiquetados con su número de versión.
    - Transformación
      - Convivirán transacciones con ambos números de versión.
      - Cada transacción escoge conectores y componentes (con su misma versión).
    - Eliminación
      - Cuando ya no queden transacciones con la versión antigua se podrán eliminar los componentes y conectores de esa versión.

2.3.1. Gestión dinámica de versiones

Instalación Transform. Elim.

Version carrier		T_0 the system is in normal running before a reconfiguration	T_1 the new sub-system is ready	T_2 the v_{old} transactions commit
System (v_s)		v_{old}	$v_s = v_{new}$	v_{new}
Connectors (v_c)	Existing and reserved	v_{old}	v_{old}	v_{new}
	Existing but to be removed	v_{old}	v_{old}	
	Newly installed		$v_c = v_{new}$	v_{new}
Threads (v_t)	Created before T_1	$v_t = v_s$	v_{old}	
	Created after T_1		$v_t = v_s$	v_{new}

2.4. Transferencia de estado

- La transferencia de estado es uno de los pasos más delicados en una actualización.
  - La continuidad de servicio dificulta su gestión.
  - Si no se exigiese continuidad y se utilizara inactividad, la transferencia sería trivial.

2.4. Transferencia de estado

- Para resolver la transferencia, Li [Li11] propone la “equivalencia *stateless*”.
  - Equivalencia *stateless*: Un sistema “*stateful*” es equivalente “*stateless*” durante una actualización si la transferencia de estado no causa ninguna suspensión en la actividad del sistema.
    - Es decir, no se bloquea ninguna transacción.

2.4. Transferencia de estado

- Una forma de lograr esta equivalencia es permitir la compartición de estado.
- Esa compartición requiere:
  - Compatibilidad entre el formato del estado de la vieja y la nueva versión.
  - Encapsulación del estado del componente a actualizar en un único objeto “wrapper”.
    - Para que la nueva versión pueda utilizarlo.
  - Acceso en exclusión mutua al estado compartido durante la coexistencia de versiones.
  - La lógica de la aplicación debe admitir esa compartición.
  - El componente a reemplazar y el que lo reemplace deben ubicarse en la misma máquina.

2.5. Minimización de sobrecarga

- Para realizar la actualización se necesitan iniciar algunas transacciones específicas:
  - Instalación de nuevos componentes.
  - Gestión de versiones.
  - Eliminación de componentes obsoletos.
  - …
- Esas tareas introducen carga adicional al sistema.

2.5. Minimización de sobrecarga

- La última característica (y la más exigente, pues incluye a todas las demás) de QoS en un sistema de actualización es la “garantía de QoS”:
  - Garantía de calidad de servicio: Un sistema en ejecución está “QoS garantizado” si su calidad de servicio está físicamente mantenida por encima de un umbral predeterminado durante toda la actualización.

2.5. Minimización de sobrecarga

- Para llegar a ese nivel:
  - La organización y planificación de la actualización deben considerar todas las transacciones introducidas por la propia actualización.
    - Además de la carga normal de transacciones que ya introducen los usuarios del sistema.
  - La planificación debe tener en cuenta la utilización de cada uno de los recursos del sistema, evitando su saturación.

Índice

3. Organización y planificación

- Según [Li11], para implantar un sistema de actualización habrá que considerar dos tipos de control:
  - Control lógico de la calidad de servicio. Gestionará:
    - inactividad,
    - gestión dinámica de versiones,
    - transferencia de estado mediante copia y...
    - transferencia de estado mediante compartición.
  - Control físico de la calidad de servicio.
    - Implica planificación de uso de recursos.
    - Tipos de planificación:
      - Competitiva
      - No competitiva
      - Controlada

3. Organización y planificación

- Un “organizador” de actualización es el componente que realizará el control lógico.
- Tras un análisis del estado del arte, Li distingue tres clases de organizadores:
  - De disponibilidad (“*avl*”). Implanta inactividad y transferencia de estado mediante copia.
  - De continuidad (“*con*”). Implanta gestión dinámica de versiones y transferencia de estado mediante copia.
  - De equivalencia (“*sle*”). Implanta gestión dinámica de versiones y equivalencia “stateless” mediante compartición.

3. Organización y planificación

- Un “planificador” de actualización es el componente que realizará el control físico.
- Li propone tres clases de planificadores:
  - Competitivo (“cpt”): Asigna la misma prioridad a las transacciones de actualización que a las de los usuarios.
  - Expulsivo, o no competitivo (“pre”):
    - Asigna una prioridad mínima a las transacciones de actualización.
    - Las transacciones iniciadas por los usuarios siempre expulsarán a las de actualización.
  - Controlado (“ts”): Divide el tiempo de uso de los recursos en intervalos y reparte estos intervalos entre las dos clases de transacciones.
    - Similar a un “round-robin”.
    - Utilizar cuando la carga normal ya sature los recursos.

Índice

4. Evaluación

- Li propone un benchmark específico para sistemas de actualización.
- La aplicación a utilizar debe:
  - Ser “stateful”.
    - Exige una transferencia no trivial.
  - Admitir una carga variable.
    - El tamaño del estado y el número de transacciones por segundo podrán variarse libremente.
  - Tener componentes dependientes.
    - Deben actualizarse a la vez.
  - Tener componentes independientes.
    - Pueden ser compartidos por versiones diferentes.
  - Admitir concurrencia.

4. Evaluación

- Es una aplicación distribuida.
  - Pero no se exige que utilice replicación.
  - Ni que sea escalable.
  - Es un caso básico para considerar la calidad de servicio.

4. Evaluación

- ¿Qué sistemas de actualización compara?
  - Algunas combinaciones entre organizadores y planificadores.
  - Ver sección 3.
  - Clases evaluadas:
    - Avl-cpt: Garantiza consistencia y disponibilidad.
    - Con-cpt: Garantiza consistencia, disponibilidad y continuidad.
    - Sle-cpt: Garantiza consistencia, disponibilidad, continuidad y equivalencia.
    - Qos-pre: Garantiza calidad (en situaciones de insaturación).
    - Qos-ts: Garantiza calidad (en situaciones de saturación previa).

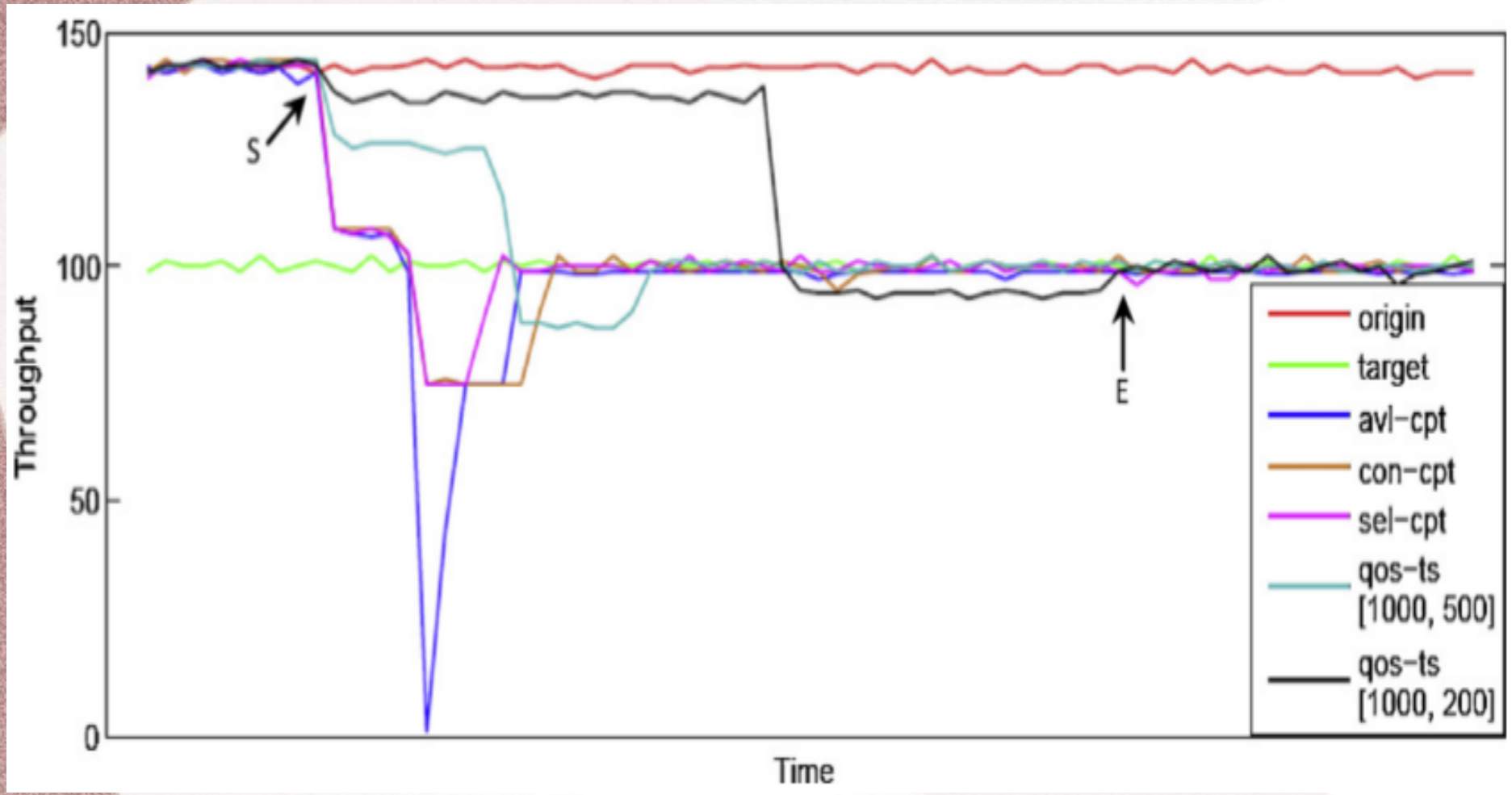
4. Evaluación

- ¿Cuáles son los objetivos?
  - Incrementar la funcionalidad del sistema.
  - La aplicación implanta un mecanismo básico de gestión de conexiones seguras.
  - Tras la actualización, su funcionalidad aumenta:
    - Autenticación de agentes,
    - No repudio,
    - Confidencialidad...
    - No se persigue un mayor rendimiento.
      - De hecho, el rendimiento empeora.

4. Evaluación

- ¿Qué parámetros considera en la QoS?
  - Rendimiento: Transacciones/segundo.
  - Tiempo de respuesta.
- La disponibilidad es un requisito.
  - Se asume que estará garantizada.
- Distingue dos escenarios:
  - Sistema ya saturado.
  - Sistema no saturado previamente.
- Veamos los resultados obtenidos...

4.1. Saturación - Rendimiento



4.1. Saturación - Rendimiento

- Analicemos qué se consigue con cada estrategia:
  - "avl-cpt":
    - Utiliza inactividad y copia de estado; planificación competitiva.
    - Llega a bloquear por completo el sistema.
      - Rendimiento cero.
      - ¡Incumplirá el SLA!!!
    - Finaliza rápidamente la actualización.
      - Aunque la estrategia "sle-cpt" es ligeramente mejor.
    - Los mecanismos utilizados son fácilmente implantables.

4.1. Saturación - Rendimiento

- "con-cpt":
  - Utiliza gestión dinámica de versiones y copia de estado; planificación competitiva.
  - No incurre en bloqueos.
    - Pero el rendimiento mínimo obtenido (70 trans/seg) está por debajo del SLO fijado (100 trans/seg) durante un tiempo apreciable.
  - Necesita más tiempo que la anterior para completar la actualización.

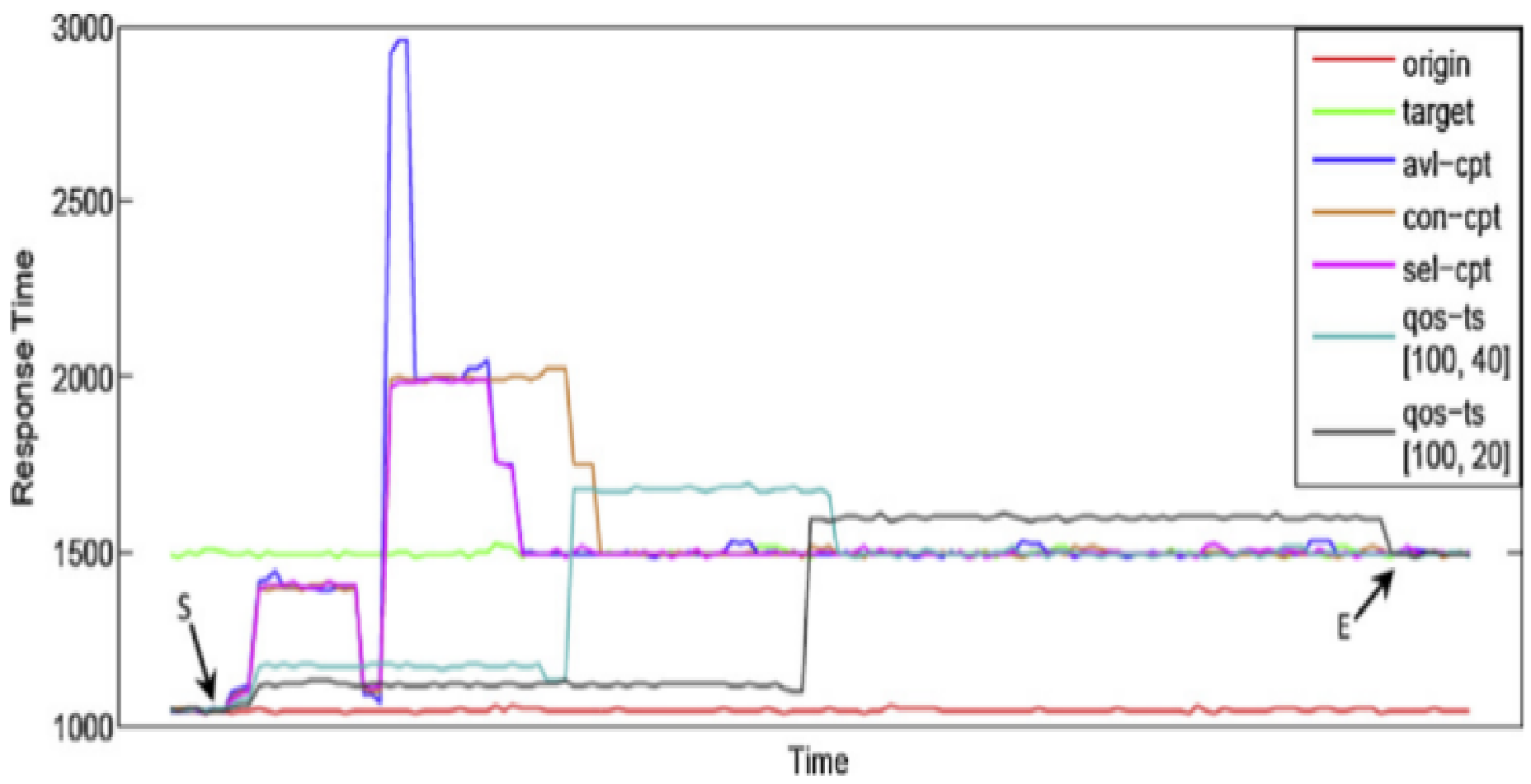
4.1. Saturación - Rendimiento

- “sle-cpt”:
  - Utiliza gestión dinámica de versión y compatibilidad “stateless” mediante compartición; planificación competitiva.
  - Es la mejor estrategia en cuanto a tiempo necesario para actualizar.
  - Incurre en un rendimiento inferior al planteado como SLO.
    - Pero durante menos tiempo que la estrategia anterior.

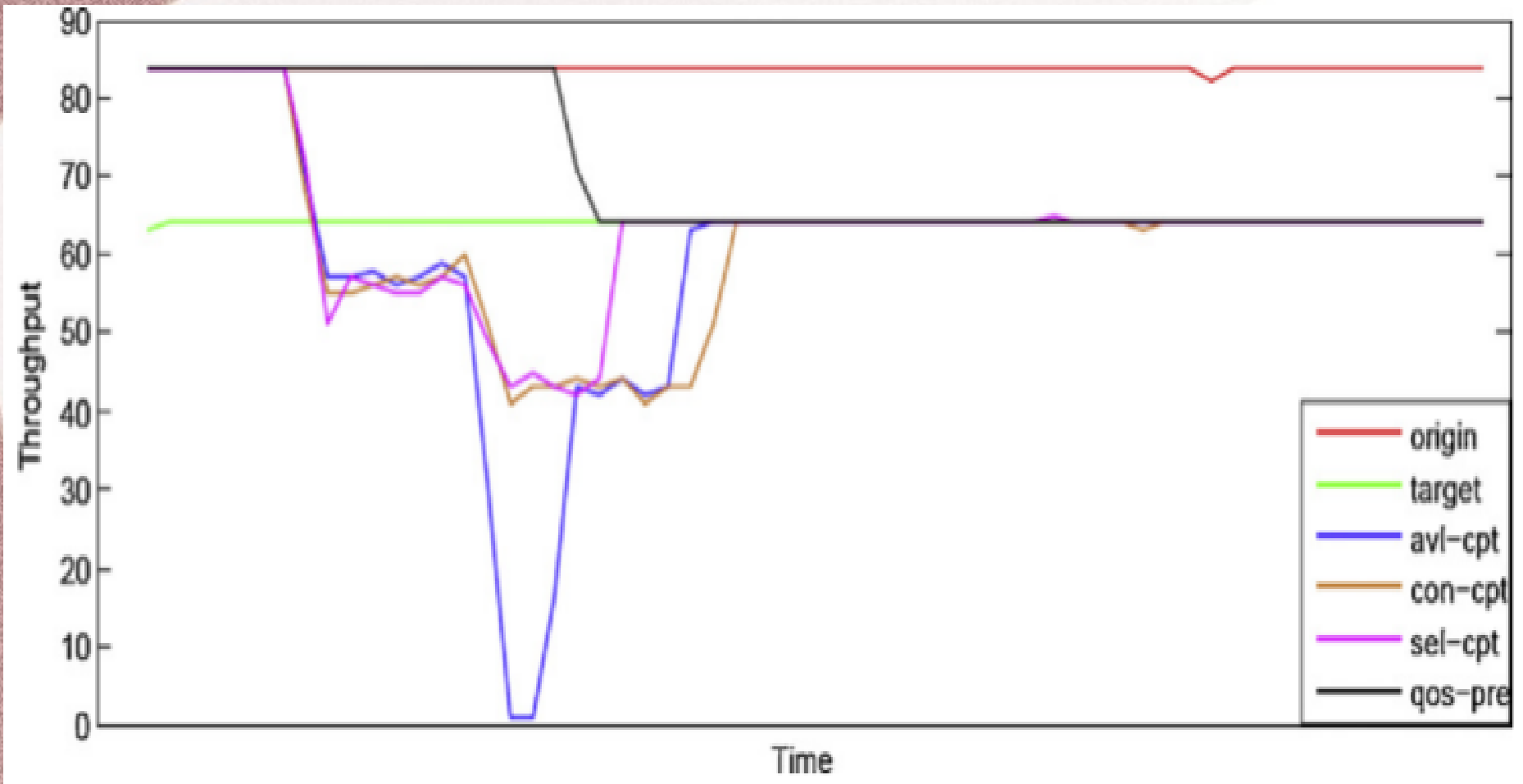
4.1. Saturación - Rendimiento

- "qos-ts":
  - Dos variantes:
    - [1000,500]. Reparto equitativo de intervalos entre las dos clases de transacción.
    - [1000,200]. Sólo el 20% del tiempo para las transacciones de actualización.
  - Ambas prolongan más que cualquier otra estrategia anterior la duración de la actualización.
  - Sin embargo, generan unos niveles de calidad muy cercanos al SLO comprometido.
    - 85 trans/seg en la variante [1000,500]
    - 95 trans/seg en la variante [1000,200]
  - Son las mejores opciones: minimizan las penalizaciones.

4.2. Saturación – Tiempo de respuesta



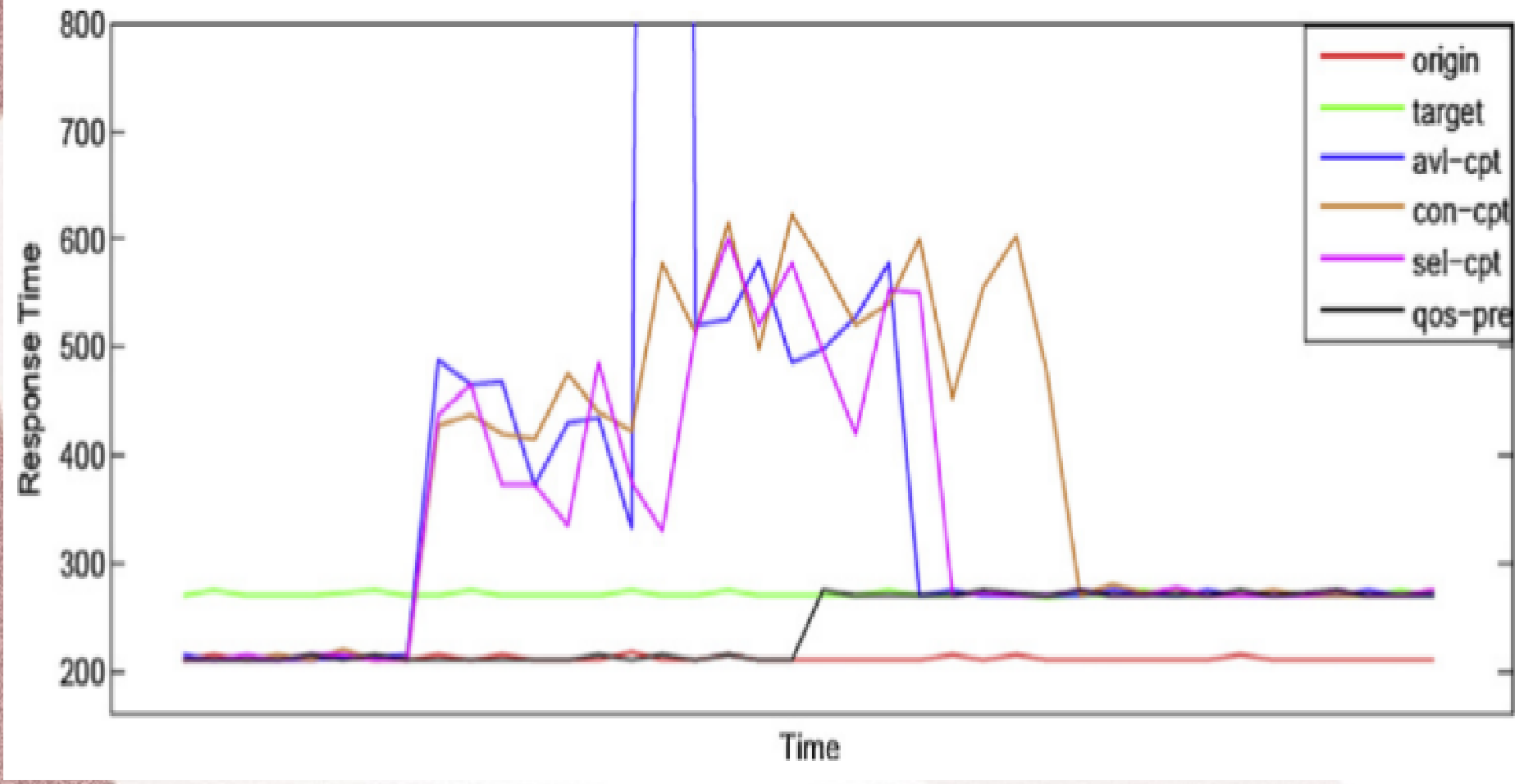
4.3. Insaturación – Rendimiento



4.3. Insaturación - Rendimiento

- La comparativa entre los resultados de “avl-cpt”, “con-cpt” y “sle-cpt” merece la misma valoración que en la sección 4.1.
- Pero ahora “qos-pre” es claramente la mejor opción:
  - Es más rápida que cualquiera de las demás.
  - No incumple jamás el SLO comprometido.
  - ¡Es la estrategia a utilizar!!

4.3. Insaturación – Tiempo de respuesta



Índice

5. Conclusiones

- Los resultados obtenidos permiten afirmar:
  - La inactividad sólo será utilizable en aquellos casos en que el sistema ya esté saturado y sea crítico finalizar la actualización cuanto antes.
    - A costa de perder totalmente la capacidad de servicio durante un intervalo mínimo.

5. Conclusiones

- Esta capacidad de servicio puede garantizarse mediante una gestión dinámica de versiones, pero...
  - Por sí sola no garantiza el mantenimiento de los SLO típicos (tiempo de respuesta, rendimiento...).
  - Debe combinarse con:
    - Compartición de estado para ofrecer equivalencia "*stateless*".
      - En aquellos servicios que lo admitan.
    - Una organización y planificación adecuadas.

5. Conclusiones

- Esta planificación debe ser...
  - Expulsiva, otorgando mínima prioridad a las transacciones de actualización.
    - Cuando la carga no sature los recursos.
      - Garantizará el mantenimiento de los SLO en esos casos.
  - Controlada, en caso de recursos saturados.
    - Minimiza las penalizaciones por incumplimiento de SLO.
    - Pero prolonga la duración de las actualizaciones.

Índice

6. Resultados de aprendizaje

- Al finalizar esta unidad, el alumno debe ser capaz de:
  - Revisar y analizar el impacto que tienen diferentes mecanismos de actualización sobre el cumplimiento de los SLO.
  - Conocer el concepto de estrategia de actualización.
  - Identificar qué estrategias de actualización conviene utilizar en función de los SLO de un servicio.