**Dongyoon Lee, Benjamin Wester, Kaushik Veeraraghavan. University of Michigan**

# Respec: Efficient Online Multiprocessor Replay via Speculation an External Determinism

CARLOS MARTÍN FLORES GONZÁLEZ, Carné: 2015183528

**Instituto Tecnológico de Costa Rica**
**Maestría en Computación**
**Sistemas Operativos Avanzados**
**Profesor: Francisco Torres Rojas, Ph.D**

Sistemas de repetición determinista (*deterministic replay systems*) son usados para grabar y reproducir la ejecución de un sistema de hardware o software. Esta habilidad puede ser usada para mejorar sistemas a lo largo de muchas dimensiones, incluyendo confianza, seguridad y debugabilidad (*debuggability*). La idea general detrás de repetición determinista es la de registrar (*log*) todos los eventos no deterministas durante la grabación y reproducir estos eventos durante la repetición. Repetición determinista para uniprocesadores puede ser proporcionado a un bajo *overhead* porque los eventos no deterministas ocurren en frecuencias relativamente bajas, por esto es que el registrar eventos añade poco *overhead*.

Respec se basa en dos ideas: (1) puede registrar de manera optimista el orden de las operaciones en memoria de forma menos precisa que el nivel necesario para garantizar repetición determinista, mientras ejecuta la ejecución grabada especulativamente para garantizar seguridad. Luego de un número configurable de faltas de especulación (*misspeculations* - esto es cuando la información registrada no es suficiente para asegurar repetición determinista para un intervalo), Respec devuelve la ejecución al inicio del intervalo actual y se re-ejecuta con un *logger* más preciso. (2) Respec puede detectar *misspeculations* para un intervalo al repetir concurrentemente el intervalo grabado en *spare cores* y verificando si la salida del sistema y los estado final del programa coincide con los de la ejecución grabada. Fases de Respec: (1) Registra la mayoría de las operaciones de sincronización ejecutadas por un programa *multithreaded* de memoria compartida. (2) Detecta cuando operaciones de sincronización registradas son insuficientes para reproducir un intervalo de la corrida original. Respec repite concurrentemente un intervalo grabado en *spare cores* y lo compara con la corrida original. (3) Ejecución especulativa para ocultar los efectos de intervalos de repetición fallidos y para retroceder de forma transparente ambas ejecuciones, la grabada y la repetida. (4) Luego de devolverse, Respec reintenta la ejecución de un intervalo fallido de ejecución serializando los *threads* y registrando el orden, lo que garantiza que la repetición tendrá éxito para ese intervalo.

**Replay guarantees:** *Fidelity level:* sistemas de repetición difieren en su fidelidad de repetición y en el costo resultante de proporcionar esta fidelidad. Un ejemplo de fidelidades diferentes está en el nivel de abstracción en que una repetición está definida (nivel de máquina, de proceso, de programa). Repetición multiproceso añade otra dimensión de fidelidad: cómo debería la ejecución repetida reproducir las instrucciones de intercalación de diferentes *threads*. No se ha propuesto ninguna aplicación de repetición que requiera de ordenamiento basado en tiempo exacto de todas las instrucciones que se van a reproducir. La dificultad e ineficiencia de métodos de registro pesimistas han conducido a explorar un nuevo nivel de fidelidad para repetición, llamado repetición externa determinista (*externally deterministic replay*). Garantiza que: (1) la ejecución repetida sea indistinguible

de la ejecución original desde la perspectiva de un observador externo y (2) la ejecución repetida es una ejecución natural del programa de interés. El primer criterio implica que una secuencia de instrucciones ejecutada durante la repetición no puede ser probada que difiera de la secuencia de instrucciones ejecutadas durante la ejecución original porque todas las salidas observables de las dos ejecuciones son las mismas. El segundo criterio implica que cada estado visto durante la repetición estuvo fue capaz de ser producido por el programa de interés. *Online versus offline replay: Offline*: usos forenses o *debugging*, la repetición se realiza luego que la ejecución original haya finalizado. El sistema de repetición se ejecuta más lento que el original. *Online:* verificaciones paralelas, tolerancia a fallos. La ejecución repetida procede en paralelo con la original. La velocidad de la ejecución repetida es importante porque puede limitar el rendimiento del sistema. Respec está diseñado para usarse en escenarios *online*, busca minimizar el *overhead* de registro y repetición para que pueda ser usado en entornos de producción con garantías sincrónicas de tolerancia a fallos o verificación de errores de programa.

**Design:** Respec proporciona repetición determinista para uno o más procesos. Se repite en la abstracción del proceso al registrar los resultados de *system calls* y de operaciones de sincronización de bajo nivel ejecutadas por el proceso de grabación y proporcionando los resultados registrados a los procesos de repetición en lugar de re-ejecutar el correspondiente *system call* y operaciones de sincronización. Respec crea puntos de verificación (*checkpoints*) para el proceso de grabación a intervalos semi-regulares llamados *epochs*. El proceso de repetición inicia y finaliza un epoch en exactamente el mismo punto en la ejecución que el proceso de grabación. Durante un epoch, cada *thread* grabado registra la entrada y salida de sus *system calls*. Cuando un *thread* de repetición encuentra un *system call*, en lugar de ejecutarlo, lo emula leyendo el registro para producir valores de retorno y modificaciones de espacio de direcciones idénticos a aquellos vistos por el *thread* de grabación. *Divergence Checks:* verificar el estado del programa al final de cada epoch no es estrictamente necesario para garantizar repetición externa determinista. Sin embargo, verificar el estado intermedio trae beneficios: (1) permite a Respec hacer *commit* en epochs y entregar la salida del sistema. (2) reduce la cantidad de ejecución que debe ser devuelta cuando una verificación falla. (3) permiten a otras aplicaciones paralelizar verificaciones de confianza.

1. ¿CUÁL ES EL PROBLEMA QUE PLANTEA EL *PAPER*?
2. ¿POR QUÉ EL PROBLEMA ES INTERESANTE O IMPORTANTE?
3. ¿QUÉ OTRAS SOLUCIONES SE HAN INTENTADO PARA RESOLVER ESTE PROBLEMA?
4. ¿CUÁL ES LA SOLUCIÓN PROPUESTA POR LOS AUTORES?
5. ¿QUÉ TAN EXITOSA ES ESTA SOLUCIÓN?