Virtualización en Sistemas Embebidos Multicore

Francisco Acosta
Departamento de Ingeniería de
Sistemas e Industrial
Universidad Nacional de Colombia
Bogotá Colombia
sfacostale@unal.edu.co

ABSTRACT

Dado el continuo aumento en la cantidad de unidades de procesamiento o "cores", así como la continua evolución en los diferentes bloques funcionales de los procesadores usados en el diseño y construcción de sistemas embebidos, la idea de implementar técnicas de virtualización en el diseño de dichos sistemas se ha vuelto práctica. Las técnicas de virtualización en el ámbito de los servidores han sido usadas exitosamente para ofrecer diferentes entornos aislados de ejecución y para promover el aprovechamiento de los recursos computacionales del hardware subyacente. Aunque de momento el conjunto de técnicas de virtualización y su uso en el contexto de los servidores están bien definidos, con algunos tópicos aún en proceso de consolidación[1], el escenario no es el mismo en el contexto de los sistemas embebidos. Este artículo pretende mostrar un panorama de la adopción de la virtualización en el diseño de sistemas embebidos. Dada la masiva acogida de procesadores basados en arquitectura ARM por parte de la industria, el artículo relaciona en su mayoría información relacionada con esta arquitectura. Más específicamente las arquitecturas ARMv7-A y ARMv8-A, las cuales incluyen soporte hardware en procura de mejorar y/o mantener el rendimiento de los sistemas en algunos casos de virtualización. Cuando el tópico lo amerita otras arquitecturas como MIPS y X86 son mencionadas.

CCS CONCEPTS

• Computer systems organization → Embedded systems

KEYWORDS

Hypervisor, multicore, safety, critical systesm, ARM

1 INTRODUCCION

La virtualización es una tecnología que fue desarrollada por IBM en los años 60, sin embargo, la adopción de ésta en el diseño de sistemas embebidos e recientes. La principal tarea de cualquier software de virtualización, es permitir la creación de grupos lógicos de recursos que parezcan ser los recursos físicos de un entorno de computación dado, pero que en realidad son recursos compartidos o particionados, asociados a una plataforma de hardware única. En el contexto de procesadores "multicore" la virtualización es una alternativa para el aprovechamiento ordenado de estos recursos. A la vez que posibilita la implementación de estrategias orientadas a mantener la seguridad del sistema o algunos de sus componentes, así como la implementación de estrategias de tolerancias a fallos. Este artículo pretende mostrar un panorama de la adopción de la virtualización en el diseño de sistemas embebidos. La sección dos introduce algunas de las motivaciones que han impulsado el desarrollo de la virtualización en los sistemas embebidos. En la sección tres se hace una corta introducción a los sistemas embebidos, su contexto y una clasificación superficial, que aborda principalmente la componente de procesamiento en tiempo real. En la sección cuatro se describen algunas de las estrategias más relevantes de virtualización. En la sección cinco, se describen los diferentes módulos hardware implementados en los procesadores ARM para asistir la virtualización. En la sección seis e hace un acorta descripción de un domino de problemáticas que pueden ser afrontadas mediante la virtualización en sistemas embebidos. La relación de la

literatura existente y una descripción de los trabajos más representativos es presentada en la sección siete. La sección ocho hace un recuento de las diversas aplicaciones de la virtualización en sistemas embebidos. La sección nueve presenta algunas perspectivas de desarrollo y por último en la sección diez se presentan algunas conclusiones.

2 VIRTUALIZACION EN SISTEMAS EMBEBIDOS

La adopción de la virtualización como estrategia en el desarrollo de sistemas embebidos trae consigo la posibilidad de implementar algunas de las siguientes funcionalidades o características[2]:

- Permite la ejecución de varios sistemas operativos (similares o no) en una única maquina
- Al ser posible distribuir y encapsular diferentes módulos de software, tiene el potencial de mejorar la calidad en el diseño de este.
- Al proveer un aislamiento entre máquinas virtuales, mejora la seguridad del sistema.
- Incrementa la flexibilidad del sistema.
- Mejora la administración de las cargas de procesamiento.
- Promueve la independencia del hardware subyacente.

En principio la virtualización agrega una capa de código denominada "hypervisor" o también conocida como monitor de máquina virtual (VMM virtual machine monitor). Este es responsable de crear y manejar las máquinas virtuales y de proveer la abstracción de los recursos disponibles en el hardware subyacente, permitiendo que las instrucciones de cada máquina virtual sean ejecutadas correctamente en el hardware real. Un gran impulso a esta tecnología ha sido el hecho de que los fabricantes de procesadores han agregado dentro de sus arquitecturas (Intel VT-x, AMD-V, ARMv7-A, MIPS VZ) módulos para soportar y asistir el desarrollo de hypervisores, esto a su vez ha impulsado el desarrollo de los mismos. Uno de los objetivos que persiguen estos desarrollos es reducir el tiempo que conlleva la interpretación de las instrucciones por parte del hypervisor, antes de su ejecución en el procesador real, para en últimas lograr que este retardo sea despreciable, y el rendimiento de una aplicación dada sea semejante al que se percibiría en un contexto sin virtualización. Los diferentes módulos orientados a atenuar las latencias introducidas por la pila de software que conforma un hypervisor son descritos en la sección cinco.

Es de anotar que aunque la virtualización en sistemas embebidos tiene otros matices, y existen escenarios con diversos requerimientos, la amplia oferta de plataformas multicore [3], junto con la disponibilidad de herramientas de virtualización de propósito general, pone a disposición de los desarrolladores de sistemas embebidos muchas de las técnicas ya consolidados en el campo de los servidores. Por otra parte ahora es factible construir servidores, un campo usualmente dominado por procesadores Intel, basados en procesadores ARM, estos últimos ampliamente usados en el desarrollo de sistemas embebidos.[4].

En cuanto al segmento de los sistemas embebidos, la virtualización tiene el potencial de reducir costos de diseño y producción. Posibilita el reusó de software antiguo junto con nuevas aplicaciones, posibilita la combinación en un solo chip, de sistemas operativos de propósito general como Linux, junto con sistemas operativos de tiempo real como FreeRTOS. Una característica de las diferentes aproximaciones a la virtualización en sistemas embebidos, es que tratan de resolver un problema de optimización, en el cual de un lado es necesario cumplir con requerimiento de tiempo real, como el despacho y ejecución de tareas dentro de los límites temporales requeridos por el sistema, a la vez que se debe ejecutar un amplio conjunto de aplicaciones y servicios de propósito general, sin requerimientos específicos en cuanto a tiempo de ejecución y en su mayoría orientados a soportar algún tipo de interacción con el usuario. La virtualización en sistemas embebidos también viabiliza la implementación de estrategias orientadas a la seguridad de los sistemas.

3 SISTEMAS EMBEBIDOS

Los sistemas embebidos solían ser dispositivos relativamente simples, con un número fijo de tareas. Esta simplicidad ha venido desvaneciéndose a medida que los procesadores con que son construidos integran varias unidades de procesamiento además de una gran variedad de módulos, incluyendo módulos de comunicaciones (cableadas e inalámbricas), video, audio y otros tipos de aceleradores y periféricos. Estos sistemas están presentes en una gran variedad de aplicaciones, algunos de las cuales se relacionan a continuación:

- Electrónica De Consumo

- Dispositivos Médicos
- Equipos de comunicaciones
- Automóviles
- Automatización
- Hogar/Oficina

Aunque una clasificación más amplia de los diferentes tipos de sistemas embebidos esta fuera del alcance de este documento, aquí se describen algunos aspectos relevantes. Dependiendo en los efectos que conlleve una falla dada, el sistema embebido se puede clasificar en crítico o no crítico. Los sistemas cuya falla se puede traducir en serios daños al ambiente o incluso en la muerte de alguien son clasificados como críticos. Sistemas cuya falla solo causa una degradación en el funcionamiento del sistema del cual hace parte, son clasificados como no críticos [5].

Dependiendo de las restricciones o requerimientos temporales, el sistema se puede clasificar como de tiempo real o no. Si dado un evento de entrada, el sistema debe generar una respuesta correcta antes de que transcurra un intervalo de tiempo plenamente especificado, se dice que el sistema tiene un requerimiento de tiempo real. Si la cantidad de requerimientos de tiempo real es abundante y/o las magnitudes de los intervalos que hay que cumplir, son del orden de los milisegundos (depende de la velocidad de procesamiento disponible), se dice que el sistema es de tiempo real fuerte [6].

Otras características propias de los sistemas embebidos son el bajo consumo, periodos de funcionamiento prolongados o continuos y en algunos casos, la no interacción con humanos. Finalmente el conjunto de restricciones que guían el diseño de un sistema embebido esta dodo en su mayoría por el sistema del que aran parte.

4 TECNICAS DE VIRTUALIZACION

4.1 Hypervisor Basado en Kernel de Particionamiento

4.1.1 Base Conceptual

El hypervisor basado en el esquema del kernel de separación o perticionado, es una aproximación alternativa y complementaria a los esquemas AMP/SMP en el contexto de procesadores multicore. El Kernel de separación es un sistema operativo reducido que se ejecuta sobre un procesador multicore y particiona o separa los recursos (cores, memoria y periféricos) para el uso de las aplicaciones, en principio aplicaciones simples y posiblemente con requerimientos de tiempo real. Cuando el kernel de separación ha sido implementado con base en los mecanismos propios de un hypervisor, las aplicaciones hospedadas también pueden ser sistemas operativos y no solo aplicaciones simples, cada una con acceso a los recursos disponibles dentro de su partición.

El kernel de separación fue por primera vez introducido como una estrategia para implementar sistemas con reglas de seguridad de acceso multinivel [7], en el cual la información es clasificada en niveles: descalcificada, ultra secreto, secreto, entre otros. Los derechos de acceso de los usuarios a la información dentro de estos niveles de seguridad son restringidos en base a sus credenciales.
Los principios básicos que soportan la fiabilidad de un sistema basado en un kernel de separación son:

- El sistema es partido en una sección confiable y reducida (“Trusted Computer Base”), constituida por una cantidad fácilmente auditable de líneas de código [8] y una sección menos confiable pero más grande. Esto en terminod de lineas de codigo.
- Las decisiones y operaciones relevantes para la seguridad del sistema son ejecutadas por la sección confiable, haciendo la sección no confiable irrelevante para la seguridad general del sistema.
- La sección confiable es sometida a un riguroso proceso de verificación, en procura de comprobar que tiene las propiedades requeridas para mantener la seguridad del sistema.

La filosofía detrás del mecanismo propuesto por esta técnica es adicionar complejidad a la sección confiable, en vez de agregar confiabilidad a la sección más compleja [8]. Las implementaciones que siguen estrictamente esta estrategia de separación, suministran más confianza en términos de seguridad, pero son rígidas y no suministran mecanismos de acceso compartido a dispositivos de entrada/salida, ya que estos en principio son adjudicados de manera estática dentro de una u otra partición. La propuesta base tampoco prevé la existencia de mecanismos de comunicación entre particiones.

Como se ha misionado, esta es una técnica de virtualización a fin con la implementación de estrategias que refuercen la seguridad de un sistema, en el cual los dispositivos de entrada/salida no son compartidos y no hay interacción entre estos. Los dispositivos de entrada/salida como los conectados a el bus USB o SATA , pueden ser asignados de manera dedicada a alguno de los sistemas operativos hospedadas, estos dispositivos a su vez solo pueden leer o

escribir en los bloques de memoria designados para al sistema operativo asociado. Dependiendo de la implementación de este modelo de virtualización, esta puede que no sea flexible en términos de acceso compartido a dispositivos físicos y mecanismos de comunicación entre máquinas virtuales. A continuación se hace una breve descripción de Jailhouse[9][10][11][12], un hypervisor basado en la filosofía antes descrita.

4.1.2 Jailhouse

Es un hypervisor de código abierto, para plataformas multicore. Hace uso de la virtualización asistida por hardware para implementar la estrategia de aislamiento. En vez de contar con un abundante número de funcionalidades, Jailhouse tiene un enfoque orientado a la simplicidad y al control del acceso a recursos. La asignación de dispositivos y/o recursos es única por cada sistema operativo o aplicación hospedada. Es posible asignar uno o más cores a una única aplicación hospedad, pero no es posible compartir un único core entre 2 o más aplicaciones o sistemas operativos hospedados. Esta condición hace innecesaria la conmutación de contextos entre sistemas operativos o aplicaciones simples. Lo mismo aplica para los dispositivos del sistema, estos se asignan de manera única a un sistema operativo o aplicación hospedada. Jailhouse se debe compilar como una modulo del kernel. Por último es de notar que el aislamiento entre los sistemas hospedados puede ser reducido mediante el uso de los mecanismos de comunicación entre particiones (cells), implementados en este hypervisor. Este estudio presenta los resultados obtenidos, al comparar el rendimiento de diferentes hypervisores en procesadores ARM, incluido Jailhouse [13] en la plataforma Banana Pi. La figura 1 esquematiza la adjudicación de recursos y separación de máquinas virtuales implementadas mediante un hypervisor basado en kernel de separación.

4.1.3 Mango Hypervisor[14]

Este hypervisor desarrollado por la empresa ilbers GmbH, también se basa en la asignación estática de los recursos del hardware y periféricos, para permitir la ejecución de varios sistemas operativos en procesadores multicore. Este hypervisor ejecuta cada uno de los sistemas operativos hospedados completamente aislados de cada uno, y es muy importante anota que cada huésped se ejecuta dentro de su subconjunto real de hardware con un rendimiento nativo. Al igual que Jailhouse, los sistemas operativos hospedados no tienen que ser modificados, y también permite la comunicación entre estos.

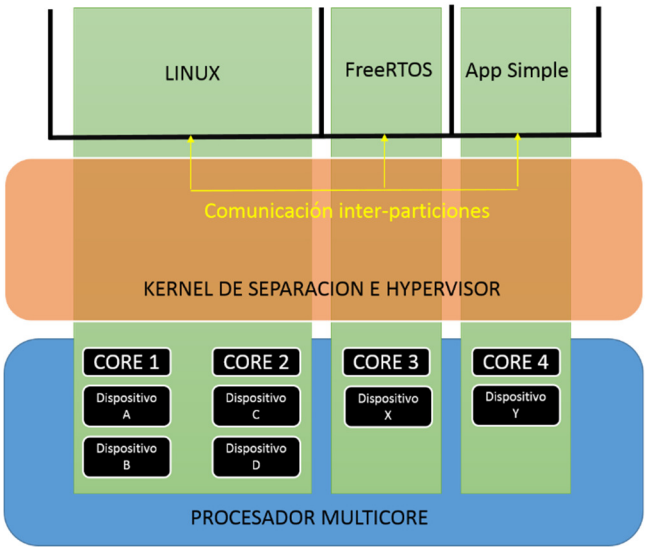

Figura 1 Esquema Hypervisor basado en Kernel de Particionamiento

.

4.1.4 Otros Hypervisores basados en kernel de Particionamiento

Quest-V[15] y PikeOS

4.2 Hypervisores Tipo 1 y 2

4.2.1 Base Conceptual

Antes de introducir los hypervisores Tipo I y II, se inicia con una breve explicación de la implementación de hypervisores en un contexto en el que no hay soporte hardware para la implementación de los mismos. En este contexto, los procesadores tienen 2 modos de operación, kernel y usuario. En el primero, cualquier instrucción del conjunto completo del set de instrucciones puede ser ejecutada, incluso aquellas que manejan las interrupciones, las que permiten manipular los registros internos del procesador y los periféricos, además las instrucciones para el manejo directo de la memoria. En el segundo modo, básicamente solo se pueden ejecutar instrucciones para calcular y procesar datos. En un escenario sin virtualización, es decir donde todos los programas, incluido el sistema operativo se ejecuta directamente sobre el hardware subyacente, el sistema operativo se ejecuta en modo kernel, y los programas de usuario de ejecutan en modo usuario. Al introducir un hypervisor, este se ejecuta en modo kernel, y el sistema operativo junto con los programas de usuario, se ejecutan en modo usuario, con lo

cual desde el código del sistema operativo ya no es posible ejecutar ciertas instrucciones directamente[16][2].

Una primera aproximación para la implementación de hypervisores es la denominada traducción binaria dinámica. En esta técnica, el hypervisor debe capturar y traducir las instrucciones que el sistema operativo huésped (ahora ejecutándose en modo usuario) intenta ejecutar, pero cuya ejecución está restringida al modo kernel del procesador. Esta traducción se hace de manera dinámica, lo cual tiene el potencial de incrementar las latencias en la ejecución, esto a su vez puede no ser adecuado para un sistema con requerimientos de tiempo real. Esta técnica se denomina virtualización pura o "full virtualización"[17] [2]. Una aproximación a la implementación de este tipo de hypervisores en procesadores de arquitectura ARMv7-A, sin soporte hardware para la virtualización, es presentada en [18].

Otra aproximación para el manejo del código del sistema operativo huésped, que se debe ejecutar en modo kernel, es sustituirlo por llamadas a bloques funcionales del hypervisor que lo hospeda, denominadas "hypercalls". Esta técnica se denomina para-virtualización, e implica la modificación del código del sistema operativo que se quiere hospedar. El hypervisor debe entonces definir una interface compuesta por llamadas al sistema para ser usadas por el sistema operativo huésped. Esta técnica conlleva una mejora en el rendimiento respecto a la virtualización pura, antes descrita.

Dado los beneficios que conlleva la virtualización, los fabricantes de procesadores desarrollaron módulos dentro de sus procesadores para asistir el proceso de la virtualización. Una de las primeras modificaciones fue agregar un tercer modo de operación del procesador. Este nuevo modo de operación en el caso de los procesadores ARM se llamó "Hyp"[19][20][21]. La adición de este tercer modo de operación, permitió que en los diferentes escenarios de virtualización, el sistema operativo se ejecutara nuevamente en el modo kernel, y el hypervisor se ejecute en el nuevo modo de operación. Las aplicaciones de usuario, continúan ejecutándose en modo usuario. Esta y otras extensiones dispuestas en los procesadores ARM con arquitecturas ARMv7-A y ARMv8-A son descritas en la sección CINCO, y suponen una reducción en la complejidad del hypervisor, respecto a las primeras implementaciones basadas en virtualización total o "*full virtualization*" y la para-virtualización.

4.2.2 Hypervisor Tipo 1

Los hypervisores tipo 1, como Xen[22], se componen de un módulo de software que se ejecuta directamente sobre el hardware, el cual provee los mecanismos esenciales de virtualización para las máquinas virtuales que se ejecutan sobre el hypervisor. Los hypervisores tipo 1, suelen presentar dificultades relacionadas con la implementación repetida de drivers para todo el hardware soportado. En el caso específico de Xen, esta problemática es resuelta mediante una implementación mínima de drivers (manejadores y soporte para el hardware) como parte del hypervisor. Un segundo módulo de software, que no es más que una máquina virtual con privilegios para acceder al hardware, y que en Xen se llama Dom0, ejecuta un sistema operativo como Linux, este a su vez hace uso de todo los drivers disponibles para acceder a todos los dispositivos de entrada/salida. Xen usa entonces este Dom0 para llevar a cabo cualquier operación de entrada/salida requerida por las otras máquinas virtuales, también conocidas como DomUs. La siguiente figura esquematiza la arquitectura de Xen, como hypervisor tipo 1, y en el contexto de la virtualización asistida por hardware. La figura 2 muestra un esquema de la implementación de Xen en procesadores ARM.

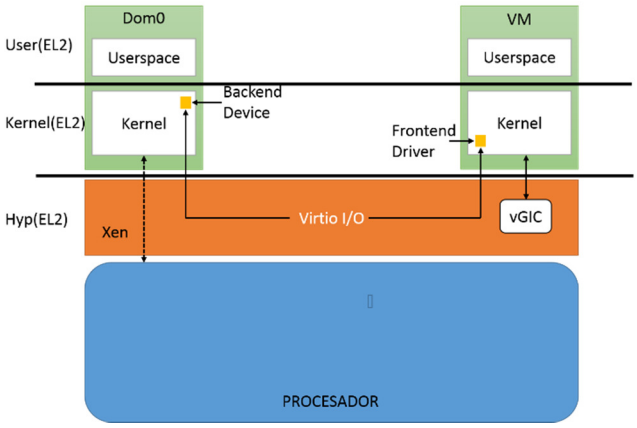

Figura 2. Esquema de implementación de Xen en procesadores ARM

4.2.3 Otros Hypervisores Tiypo 1

Xvisor[23] y K-hypervisor[24]

4.2.4 Hypervisor Tipo 2

El tipo 2 de hypervisores, operan como una aplicación más dentro del sistema operativo. Los hypervisores de este tipo, como por ejemplo KVM[20][19], hacen parte del sistema operativo que se ejecuta directamente sobre el hardware. Tanto las aplicaciones como las máquinas virtuales son

ejecutadas sobre este sistema operativo. Estos hypervisores típicamente se integran al sistema operativo existente para facilitar la ejecución de las máquinas virtuales. Esta integración se puede dar de dos maneras, bien sea mediante la integración del monitor de máquinas virtuales (virtual machine monitor VMM) dentro del código fuente base del sistema operativo anfitrión, o mediante la instalación del monitor de máquinas virtuales como un driver dentro del sistema operativo. En el caso de KVM (Kernel-based Virtual Machine), este se encuentra integrado directamente en el kernel de Linux. Soluciones tales como VMware, usa un driver cargable en el sistema operativo existente, para monitorear las máquinas virtuales. El sistema operativo integrado junto con un hypervisor tipo 2, es comúnmente referenciando como el sistema operativo anfitrión (host OS), en contraste con el sistema operativo huésped (guest OS), el cual se ejecuta dentro de las máquinas virtuales. Una característica de este tipo de hypervisor, es que dada su arquitectura, es que se facilita, el reusó de las funcionalidades dispuestas en el sistema operativo anfitrión, en especial el driver de cada dispositivo y hardware disponible. Una plataforma para la simulación de escenarios de virtualización con KVM en procesadores Cortex-A15 [25]. La figura 3 muestra un esquema de la implementación de KVM en procesadores ARM.

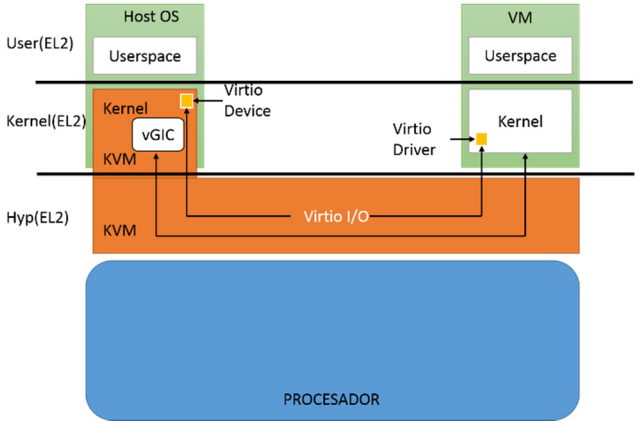

Figura 3: Esquema de la implementación de KVM en procesadores ARM.

4.3 Virtualización a Nivel de Proceso.

Este tipo de virtualización, tiene lugar a nivel del sistema operativo. Una técnica común para este tipo de virtualización consiste en la denominada virtualización basada en contenedores. Esta técnica, a diferencia de la virtualización a nivel de hardware, no requiere de un hypervisor para ejecutar un sistema operativo huésped. Esta técnica es basada en la virtualización de las llamadas al sistema (system calls). De este modo varios contenedores software y varios recursos físicos del hardware subyacente pueden ser compartidos.

5 SOPORTE HARDWARE PARA LA VIRTUALIZACION EN ARQUITECTURAS ARMv7-A Y ARMv8-A

La transición en la que el control del sistema se pasa desde la máquina virtual hospedada hacia el hypervisor ocurre siempre que el hypervisor debe ejercer algún tipo de control sobre el sistema, tal como procesar una interrupción o procesar una operación de entrada/salida. Una vez el hypervisor ha completado su trabajo con el hardware, este transfiere el control del sistema, de vuelta a la máquina virtual hospedada, permitiendo que la carga de procesamiento en esa máquina virtual continúe ejecutándose. Esta transición adiciona un retardo significativo en la comunicación entre el hypervisor y la máquina virtual. El principal objetivo en el diseño tanto de un hypervisor, como de los diferentes módulos hardware para asistir la virtualización, es reducir tanto como sea posible, la frecuencia y el costo temporal de dichas transiciones[19]. A continuación se relacionan los diferentes módulos.

- Virtualización de la CPU
- Virtualización de la Memoria Física
- Virtualización de Interrupciones
- Virtualización de Temporizadores

6 FUSION DE TECNOLOGIAS OPERACIONALES Y TECNOLOGIAS DE LA INFORMACION

La automatización de procesos industriales es estructurada con base en tecnologías operacionales que No implementan mecanismos de seguridad. Con el advenimiento de paradigmas como el Internet de las Cosas y la industria 4.0, se ha venido forjando una fusión entre las tecnologías operacionales y las tecnologías de las información, acoplando las problemáticas de seguridad de estas últimas a ambientes críticos , no preparados para afrontar el amplio espectro de riesgos operacionales que conlleva dicha fusión. El reconocimiento de estos riesgos ha traído consigo un rechazo por parte de algunas industrias a la adopción de estos nuevos paradigmas, que en principio pretenden fortalecer los procesos productivos. Por esto es

procedente el estudio de la virtualización como una alternativa para aislar esto dos contextos operacionales, en sistemas embebidos que integran funcionalidades de ambos tipos, además como un mecanismo que posibilita la implementación de servicios/funcionalidades redundantes, que puedan garantizar el restablecimiento de las funcionalidades en caso de accesos no autorizados. Dada la disponibilidad de procesadores/plataformas multicore, es también procedente estudiar las diferentes alternativas de manejo de los recursos del hardware subyacente, que pueden ser implementadas dado uno u otro hypervisor. La figura 4 esquematiza la situación descrita.

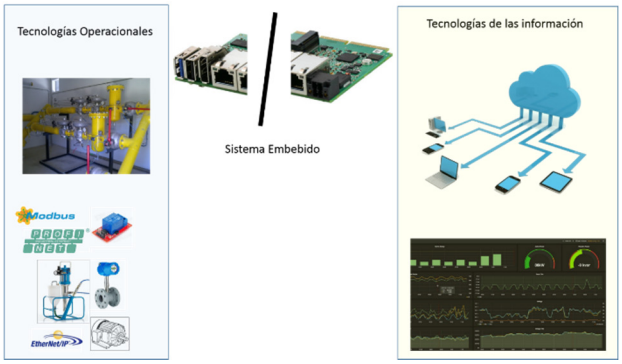

Figura 4: Fusión de tecnologías operacionales y tecnologías de la información.

Dentro del conjunto de tecnología operacional, se destacan las redes de comunicación industriales, mediante las cuales se comunicas actuadores y diferentes tipos de instrumentación. Pero también se incluyen elementos básicos de mando como relevos, salidas de corriente, o cualquier otro mecanismo que le permite al sistema interactuar con la infraestructura de automatización.

7 TRABAJOS RELACIONADOS

7.1 Fundacionales

En Sur américa el tema de la virtualización en sistemas embebidos ha estado liderado por el profesor *Fabiano Heseel*[30] de la universidad Católica Rio Grande do Soul en Brasil, muchos de los artículos y proyectos escritos y dirigidos por él, son referenciados por gran parte de los artículos encontrados durante la revisión. En su mayoría son artículos fundacionales [32][16][2] acerca de los principios de la virtualización y su adopción en el diseño de sistemas embebidos. En especial el artículo escrito junto con *A. Aguiar* en el 2010 [31], en el cual se describen los principios de la virtualización en el contexto de los sistemas embebidos, y en su quinta sección presenta 4 escenarios de uso típico de la virtualización en sistemas embebidos no críticos . Otros trabajos[6]

7.2 Procesadores MIPS

El mismo profesor Fabiano Hesse, ha dirigido varios trabajos relacionados con la virtualización en sistemas embebidos y procesadores MIPS [33][34][35].

7.3 Procesamiento de Tiempo Real y Estudios Comparativos

Dado las latencias adicionales que involucra la virtualización, la evaluación y estudio de cómo estas pueden afectar el funcionamiento de sistemas con requerimientos de tiempo real ha sido el tema de varios artículos. Algunos de estos se han centrado en evaluar y comparar estos criterios en plataformas basadas en hypervisores ya consolidado como Xen[36][37][38][39][40][41][42][43] [44][45]y KVM[46][40].
Paralelo a los trabajo basados en lo hypervisores de más amplio uso, también hay trabajos que relacionaban hypervisores diferentes[47][37][23][48]

En [23] (2015) se comparan los hypervisores Xvisor, Xen y KVM, este trabajo presenta el primero como una solución a las deficiencias en el rendimiento de los dos últimos. Las ventajas de la implementación de Xvisor son presentadas en términos de "guest" simulación IO, manejo de las interrupciones por parte del "host", latencia de sincronización de mecanismos de acceso a recursos compartidos, espacio en memoria y manejo de memoria. En cuanto al mecanismo de simulación de operaciones de entrada/salida, el documento expone como Xvisor incurre en menos conmutaciones de contexto respecto a Xen y KVM, el mismo argumento es expuesto para demostrar la eficiencia de Xvisor para el manejo de las interrupciones. En cuanto a la latencia de sincronización de mecanismos de acceso a recursos compartidos, el documento expone que no hay una diferencia representativa. En cuanto al manejo de memoria el documento expone como que Xvisor adjudica bloques contiguos de memorias al momento de la creación de la máquina virtual, al garantizar una continuidad física de estos bloques de memoria asignados, se ayuda a la efectividad de las estrategias de acceso especulativo de la memoria cache, lo cual su vez mejora el acceso a la memoria por parte de los sistemas operativos hospedados. En cuanto al tamaño en memoria el estudio presenta las siguientes tablas, que muestran el tamaño requerido para instalar uno u otro hypervisor.

Tabla 1: Memoria requerida por Xen en procesadores ARM reportada en [23]

Xen ARM	Installations		
	Xen Kernel	*Dom0 Linux zImage*	*Toolstack*[a]
Size	600+ KB	3-4 MB	20 MB <
Memory on cubieboard2	160+ MB[b]	20+ MB	---

Tabla 2: Memoria requerida por KVM en procesadores ARM reportada en [23]

KVM ARM	Installations	
	Host Linux zImage	*QEMU*
Size	3-4 MB	20 MB <
Memory on cubieboard2	20+ MB	---

Tabla 3: Memoria requerida por Xvisor en procesadores ARM reportada en [23]

Xvisor ARM	Installations
	Xvisor kernel[c]
Size	1-2 MB
Memory on cubieboard2	4+ MB out of 16MB[d]

En [23] también se presentan los resultados de pruebas dinámicas a cada uno de los hypervisores. Los *"tests"* aplicados fueron; *Dhrystone, Cachebench, Stream y Hackbench.* Es importante resaltar que en este trabajo también se relacionan los resultados obtenidos en un escenario sin hypervisor, en el cual las pruebas son ejecutas de manera nativa sobre el hardware subyacente. La graficas presentadas sugieren que los sistemas operativos huéspedes bajo en control de Xvisor tienen latencias menores en comparación a Xen y KVM. Esto implica que sistemas operativos, ya sean de propósito general o de tiempo real ejecutándose como huéspedes de Xvisor y en procesadores ARM tendrán un rendimiento cercano a un escenario sin virtualización.

En [13] (2016), se comparan Xen un hypervisor tipo 1 y Jailhouse, un kernel de particionamiento con algunas funcionalidades de hypervisor. Es ente trabajo se usa un mecanismo de despacho de máquinas virtuales, disponible a partir de la versión 4.5 de Xen, denominado *"Real Time Deferrable Server"*. Este es un mecanismo de despacho, alternativo al mecanismo por defecto de Xen, basado en créditos. Este mecanismo permite implementar estrategias de despacho para procesamiento en tiempo real.

Nuevamente el objetivo del trabajo es comparar los diferentes hypervisores en procura de establecer cual es más idóneo para aplicaciones de tiempo real. Es importante resaltar que en este trabajo también se relacionan los resultados obtenidos en un escenario sin hypervisor, en el cual las pruebas son ejecutas de manera nativa sobre el hardware subyacente. La prueba básicamente consiste en medir el tiempo de ejecución de 100000 iteraciones de la aplicación *"Cpuburn-a8"*. Los resultados son sintetizados en las siguientes tabla y figura.

Tabla 4: Tiempos de ejecución y el porcentaje asociado al tiempo operacional del hypervisor (overhead) reportados en [13].

	Execution time [ns]	Relative Overhead
Native Linux OS	≈ 111.37	-
Xen Hypervisor using Credit Scheduler	≈ 135.42	≈ 21.6 %
Xen Hypervisor using RTDS Scheduler	≈ 119.63	≈ 7.4 %
Jailhouse Hypervisor	≈ 111.42	≈ 0.04 %

Basado en los resultados de la anterior tabla, Xen presenta un rendimiento significativamente menor cuando se una el esquema de despacho basado en créditos. Este mejora al usar el mecanismo denominado *"Real Time Deferrable Server"*. En todo caso Jailhouse tiene el mejor rendimiento. Esto debido principalmente a que Jailhouse ni virtualiza, ni comparte los recursos del hardware subyacente entre los sistemas operativos o aplicaciones hospedadas, y de este modo no existe una latencia asociada a la conmutación.

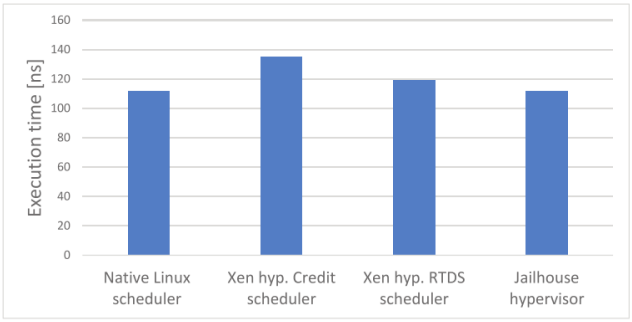

Figura 5: Comparación grafica de los tiempo de ejecución de la aplicación Cpuburn-a8, reportada en [13]

7.4 ARM

De los artículos revisados, hay una cantidad importante asociados a la arquitectura de procesadores ARM. Algunos de estos detallan la implementación de KVM para esta arquitectura[19][20]. Otros artículos presentan estudios sobre el rendimiento de sistemas basados en virtualización sobre procesadores ARM[49][50][51]

7.5 Seguridad y Confiabilidad

La capacidad de asilamiento de subsistemas que brinda la virtualización, se ve reflejada en los artículos que relacionan a la misma como parte fundamental en la implementación de esquemas de seguridad[52][53][54][55]. Esto en el contexto de sistemas "Cyber" fisicos[41][56], la industria automotriz[53], como en equipos móviles[57][58][8]. Aunque las diferentes técnicas de virtualización permiten implementar estrategias de control de accesos y esquemas de control del flujo de datos entre máquinas virtuales, hay trabajos orientados a la evolución de escenarios en los que el encapsulamiento y aislamiento puedan ser violados, por lecturas no autorizada de datos de una máquina virtual (violación de la confidencialidad), mediante la modificación no autorizada de secciones de datos de una máquina virtual (violación de la integridad) o, fuga de información, es decir datos alojados en la memoria o en los componentes del hardware, los cuales pueden ser explotados por software malicioso. *Oliver Schwarz* en [8] estudia algunas estrategias a nivel de código de bajo nivel que pueden garantizar este aislamiento.

7.6 Virtualización Como Mecanismo de Tolerancia a Fallas

Trabajos como [59], describen como la migración de máquinas virtuales puede ser usada como mecanismo de tolerancia a fallos de un sistema embebido. Este trabajo en particular se enmarca en plataformas multicore homogéneas y sistemas con restricciones de tiempo real, y evalúa bajo qué condiciones de tamaño de una máquina virtual y tiempo de migración, es factible implementar la migración de máquinas virtuales como estrategia de restauración del servicio frente a fallas del hardware, con tiempos predecibles para ejecutar esta tarea. La arquitectura se presenta como una alternativa al mapeo estático de máquinas virtuales a procesadores. El tipo de virtualización presentada en el artículo es tipo 1 con paravirtualización.

7.7 Computación en la Niebla

La computación en la niebla es una alternativa a los sistemas que concentran todo el procesamiento en la nube. En el contexto del internet de las cosas, plante la inserción de nodos intermedios o "gateways" ente los sensores y/o actuadores remotos y la nube. Esto con el propósito de tratar problemáticas de procesamiento en tiempo real, y consolidación de la información transmitida a la nube. En este campo se revisaron diferentes trabajos que asocian la virtualización como mecanismo para una implementación ordenada de la pila de software que deberían tener estos dispositivos intermedios[17] [60][61][62].

7.8 Relación de Contenidos

Las figuras 5 y 6 presentan respectivamente la distribución de procesadores relacionados en la bibliografía, así como los tópicos encontrados en la misma.

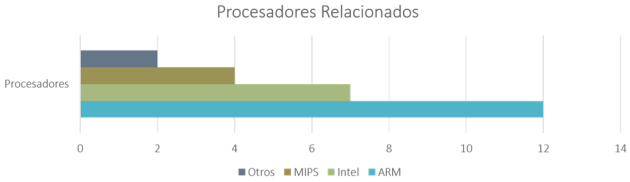

Figura 6: Relación de Procesadores mencionados en la bibliografía.

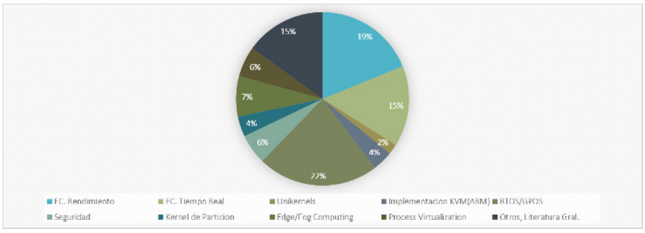

Figura 7: Relación de los diferentes tópicos encontrados en la bibliografía.

Las figuras 8 y 9 muestran en conjunto los diferentes hypervisores relacionados en la bibliografía.

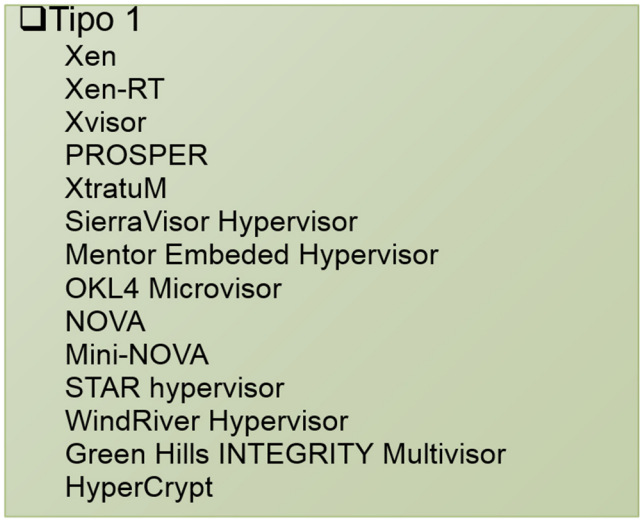

Figura 8. Hypervisores tipo 1 referenciados en la bibliografía

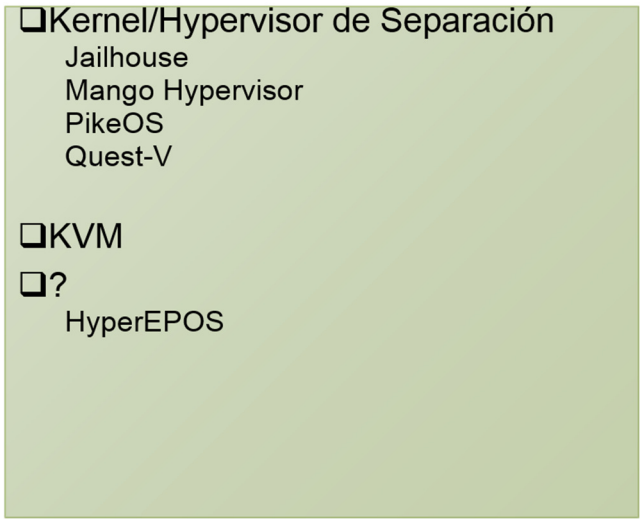

Figura 9

8 APLICACIONES

La virtualización en sistemas embebidos, junto con la disponibilidad de procesadores multicore permita la consolidación de múltiples y diferentes tipos de carga de procesamiento en un solo chip. Por ejemplo un proceso industrial puede estar compuesto de un subsistema encargado de la recolección de datos, un subsistema en procesando señales, un subsistema de monitoreo por video, un subsistema e control automático y otro encarado de procesar la interacción con el usuario, por medio una interface hombre-máquina. Estos diferentes y variados subsistemas requieren de hardware, el cual tiene un costo, requiere espacio, energía, etc. La virtualización junto con la disponibilidad de procesadores multicore provén la capacidad de combinar todas las funcionalidades e un único hardware.

La virtualización en sistemas embebidos, junto con la disponibilidad de procesadores multicore y la conectividad a diferentes servicios dispuestos en la nueve, permita la implantación de estrategias flexibles de aprovisionamiento, en las cuales el dispositivo actualiza de manera segura sus funcionalidades. Es posible por ejemplo instalar una capa de manejo, junto con secciones donde se instalaran máquinas virtuales con las diferentes lógicas operacionales, lógicas asociadas al modelo de negocio, y lógicas de seguridad. La primera capa s la que se instala en fábrica, las restantes son actualizadas dependiendo del escenario de uso del sistema.

En el contexto de los sistemas conectados al internet, la seguridad es de primera importancia. Un sistema dado puede controlar desde el sistema de calefacción de una casa, una planta de generación de energía o algún tipo de medio de transporte. En favor de la protección de algunos recursos o funcionalidades, el aislamiento que se configura en esquemas basados en virtualización es beneficioso. Si un tipo de ataque tiene éxito en tomar el control de parte del sistema, el aislamiento puede ayudar a limitar los daños. Además si se prevé la existencia de máquinas virtuales redundantes, sería factible deshabilitar la sección atacada y restablecer el servicio de dicha sección.

Una aplicación maliciosa que se ejecute con derechos de administrador podría instalar un software que trate de recabar información privada. Al implementar un sistema basado en un kernel de partición, como estrategia de virtualización, la aplicación maliciosa no podría acceder a la maquinal virtual que hospeda la aplicación o datos que se quieren asegurar, directamente a través de una medio como una memoria USB o remotamente a través de una interface Ethernet si estos dispositivos no han sido adjudicados a dicha máquina

8.1 Casos de Uso.

- Seguridad
- Separación de contextos operacionales y/o tecnológicos
- Logísticas de distribución de “firmware” y/o funcionalidades alternativas.
- Mezcla de sistemas operativos de tiempo real y de propósito general
- Redundancia como estrategia de restablecimiento de servicios

9 PERSPECTIVA DE DESARROLLO

En cuanto a preservar la seguridad del sistema se está trabajando en mecanismos de análisis de las arquitecturas del set de instrucciones para aprender acerca posibles flujos de información que puedan ocurrir durante la ejecuciones no privilegiadas[8].

También se está estudiando que tanto se puede preservar el aislamiento, frente a por ejemplo periféricos con acceso directo a memoria, y en general mecanismo que puedan violar dicho aislamiento entre máquinas virtuales.[8]

La secuencia de arranque de los sistemas es crítica, en la medida en que código no autorizado puede ejecutarse antes que el hypervior, y de esta manera violar el aislamiento entre máquinas virtuales. En este sentido se pretende desarrollar mecanismo a nivel de software que impidan esta ejecución[8]

Xvisor es presentado en [23] como un hypervisor con métricas de rendimiento sobresalientes respecto a los hypervisores clásicos Xen y KVM, sin embargo este artículo, baso su estudio en una plataforma dual core, queda por explorar sus características en plataformas con más de 2 núcleos, y en contextos de virtualización de redes y almacenamiento.

10 CONCLUSIONES

Aunque en relación a los procesadores ARM, se han hecho importante avances, por el momento la variedad de hypervisores disponibles y en desarrollo, al igual que la variedad de plataformas disponibles, no permite, a excepción de KVM, la implementación de esquemas estandarizados de virtualización, lo cual dificulta la tarea de elegir una u otra plataforma hardware sin tener un fuerte condicionamiento dada la disponibilidad o no de uno u otro hypervior.

La aparición de los procesadores multicore, además de los diferentes modulo hardware para la asistencia de la virtualización ha alivianado el procesamiento ejecutados por los hypervisores. La aparición de los procesadores multicore junto con los módulos para el soporte de la virtualización ha cambiado las funciones de los hypervisores. En una primera instancia y dada una implementación en la que a cada sistema operativo virtualizado se le asigna uno o varios cores, ya no es necesario que el hypervisor ejecute la conmutación de contexto entre diferentes sistemas operativos virtualizados. Aún persiste como responsabilidad del hypervisor, la gestión del acceso a los periféricos o en general recursos compartidos entre sistemas virtualizados. Este último aspecto deriva en tres alternativas:

1. Agrupar el conjunto de drivers dentro del hypervisor y que este gestione el acceso a los mismos.
2. Agrupar el conjunto de drivers dentro de una implementación independiente del hypervisor.
3. No compartir el acceso a drivers y segmentar la disponibilidad de los mismos por cada sistema operativo virtualizado. Es decir cada sistema mantiene su segmento propios drivers.

Se pueden reconocer dos aproximaciones al diseño de ambientes virtualizados: el particionamiento del hardware y los hypervisores. Con el particionamiento del hardware, las maquinas físicas son divididas en diferentes particiones, y cada una puede ejecutar un sistema operativo diferente. Típicamente, habrá tantas particiones como procesadores. Un ejemplo de esta aproximación es Jailhouse.

Aunque Xvisor aún no cuenta con un “stack” de administración de máquinas virtuales tan amplio como Xen, este supone una alternativa respecto a los hypervisores clásicos Xen y KVM, esto dado el sobresaliente rendimiento que se presenta en [23].

Cuatro implementaciones sobresalen en los trabajos revisados: Xen, KVM, Xvisor y Jailhouse, todos con abundantes documentación y recursos disponible de manera libre en la web.

Junto con el desarrollo de los procesadores, es factible que en los próximos años aparezcan nuevos esquemas de virtualización y en general estrategias de gestión y construcción de sistemas heterogéneos basados en un solo chip, con procesadores múltiples e igualmente heterogéneos. La elección de uno u otro hypervisor dependerá de los objetivos del sistema.

REFERENCIAS

[1] A. Madhavapeddy *et al.*, "Unikernels: Library Operating Systems for the Cloud," *ASPLOS'13*, 2013.

[2] A. Aguiar and F. Hessel, "Current techniques and future trends in embedded system's virtualization," *Pr. Exper*, vol. 42, pp. 917–944, 2012.

[3] Banana-pi.org, "BPI-M3 Octa-core Development Board," 2016. [Online]. Available: http://www.banana-pi.org/m3.html.

[4] Kim, "Prototype of Light-weight Hypervisor for ARM Server Virtualization Young-Woo Jung, Song-Woo Sok, Gains Zulfa Santoso, Jung-Sub Shin, and Hag-Young," *Embed. Syst. Appl. 215*, 2015.

[5] Y. Li, R. West, and E. Missimer, "A Virtualized Separation Kernel for Mixed Criticality Systems," *VEE '14*, 2014.

[6] G. Taccari, L. Taccari, A. Fioravanti, L. Spalazzi, and A. Claudi, "Embedded Real - Time Virtualization : State of the Art and Research Challenges," 2014.

[7] J. M. Rushby and B. Randell, "A DISTRIBUTED SECURE SYSTEM * (Extended Abstract) since thefact that to be a single multilevel secure UNIX system , it is actually a distributed system is completely hidden fromits users andtheir programs . Thisisachieved through theuseof the . " Newcas," *Construction*, pp. 127–135, 1983.

[8] O. Schwarz, "No Hypervisor Is an Island: System-wide Isolation Guarantees for Low Level Code," 2016.

[9] M. Baryshnikov and M. Sojka, "Jailhouse hypervisor," Czech Technical University in Prague, 2016.

[10] Kiszka Jan (Siemens), "Tutorial_ Bootstrapping the Partitioning Hypervisor Jailhouse - YouTube," *Embedded Linux Conference*, 2016. [Online]. Available: https://www.youtube.com/watch?v=7fiJbwmhnRw.

[11] K. J. (Siemens), "Building Mixed Criticality Linux Systems with the Jailhouse Hypervisor - Ralf Ramsauer - YouTube," *Embedded Linux Conference*, 2016. [Online]. Available: https://www.youtube.com/watch?v=pvs0fv-gnvw.

[12] Siemens, "GitHub - siemens_jailhouse_ Linux-based partitioning hypervisor," *Github*. [Online]. Available: https://github.com/siemens/jailhouse.

[13] S. Toumassian, R. Werner, and A. Sikora, "Performance Measurements for Hypervisors on Embedded ARM Processors," *2016 Int. Conf. Adv. Comput. Commun. Informatics*, 2016.

[14] ilbers GmbH, "Mango Hypervisor." [Online]. Available: http://www.ilbers.de/en/mango.html.

[15] Y. Li, R. West, and E. Missimer, "The Quest-V Separation Kernel for Mixed Criticality Systems."

[16] A. U. C. do R. G. do S. Alexandra and), "ON THE VIRTUALIZATION OF MULTIPROCESSED EMBEDDED SYSTEMS," 2014.

[17] B. Bardhi, A. Claudi, L. Spalazzi, G. Taccari, and L. Taccari, "Virtualization on embedded boards as enabling technology for the cloud of things," in *Internet of Things: Principles and Paradigms*, 2016.

[18] N. Penneman, D. Kudinskas, A. Rawsthorne, B. De Sutter, and K. De Bosschere, "Evaluation of dynamic binary translation techniques for full system virtualisation on ARMv7-A," 2015.

[19] C. Dall and J. Nieh, "KVM/ARM: The Design and Implementation of the Linux ARM Hypervisor," 2014.

[20] C. Dall and J. Nieh, "KVM/ARM: Experiences Building the Linux ARM Hypervisor," 2013.

[21] C. Dall, S. W. Li, J. T. Lim, J. Nieh, and G. Koloventzos, "ARM Virtualization: Performance and Architectural Implications," in *Proceedings - 2016 43rd International Symposium on Computer Architecture, ISCA 2016*, 2016.

[22] S. (Aporeto) Stefano, "Xen and the Art of

Embedded Systems Virtualization - Stefano Stabellini, Aporeto - YouTube," *Embedded Linux Conference*, 2017. [Online]. Available: https://www.youtube.com/watch?v=GYb-Qn3KAUM&index=98&list=PLbzoR-pLrL6pSlkQDW7RpnNLuxPq6WVUR.

[23] A. Patel, M. Daftedar, M. Shalan, and M. W. El-Kharashi, "Embedded hypervisor xvisor: A comparative analysis," in *Proceedings - 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2015*, 2015.

[24] W. K. I. Kang, "K-hypervisor," *2016 IEEE 22nd Int. Conf. Embed. Real-Time Comput. Syst. Appl.*, 2016.

[25] V. O. Systems, "experience KVM port to ARM Cortex-A15 and big." [Online]. Available: http://www.virtualopensystems.com/en/solutions/guides/kvm-on-arm/.

[26] ARM Ltd, "ARM Architecture Reference Manual ARMv8, for ARMv8-A architecture profile." 2013.

[27] ARM Ltd, "ARM Architecture Reference Manual ARMv7 and ARMv7-R." pp. 2004–2012, 2012.

[28] ARM Ltd, "ARM Cortex -A15 Technical Reference Manual." 2011.

[29] ARM Ltd, "ARM Generic Interrupt Controller Architecture Specification." 2013.

[30] Fabiano Heseel, "Fabiano Hessel @ IoT Evolution 2016, Las vegas - YouTube," 2016. [Online]. Available: https://www.youtube.com/watch?v=SS9Ik5gpaQI.

[31] A. Aguiar and F. Hessel, "Embedded Systems' Virtualization: The Next Challenge?," 2010.

[32] A. Aguiar and F. Hessel, "Current techniques and future trends in embedded system's virtualization," *Pr. Exper*, vol. 42, pp. 917–944, 2012.

[33] C. Moratelli, S. Filho, and F. Hessel, "Hardware-assisted interrupt delivery optimization for virtualized embedded platforms," *2015 IEEE Int. Conf. Electron. Circuits, Syst.*, 2015.

[34] C. Moratelli, S. Zampiva, and F. Hessel, "Full-Virtualization on MIPS-based MPSOCs embedded platforms with real-time support," 2014.

[35] S. Zampiva, C. Moratelli, and F. Hessel, "A hypervisor approach with real-time support to the MIPS M5150 processor," in *Proceedings - International Symposium on Quality Electronic Design, ISQED*, 2015.

[36] S. Xi *et al.*, "Real-Time Multi-Core Virtual Machine Scheduling in Xen," *ESWEEK'14 Oct.*, 2014.

[37] T. Shimada, T. Yashiro, N. Koshizuka, and K. Sakamura, "A real-time hypervisor for embedded systems with hardware virtualization support," in *Proceedings of 2015 TRON Symposium, TRONSHOW 2015*, 2016.

[38] S. Xi, J. Wilson, C. Lu, and C. Gill, "RT-Xen: Towards Real-time Hypervisor Scheduling in Xen," 2011.

[39] R. Kashyap and D. P. Vidyarthi, "Security-aware Real-time Scheduling for Hypervisors," 2014.

[40] G. Phi, C. Tran, Y.-A. Chen, D.-I. Kang, J. P. Walters, and S. P. Crago, "Hypervisor Performance Analysis for Real-Time Workloads," *2016 IEEE High Perform. Extrem. Comput. Conf.*, 2016.

[41] B. Jablkowski and O. Spinczyk, "CPS-Xen: A Virtual Execution Environment for Cyber-Physical Applications," 2015.

[42] J. G. Lee, K. W. Hur, and Y. W. Ko, "Minimizing scheduling delay for multimedia in Xen hypervisor," in *Communications in Computer and Information Science*, 2011.

[43] M. Ferroni, J. A. Colmenares, S. Hofmeyr, J. D. Kubiatowicz, and M. D. Santambrogio, "Enabling power-awareness for the Xen Hypervisor."

[44] G. Taccari, L. Taccari, A. Fioravanti, L. Spalazzi, and A. Claudi, "Embedded Real - Time Virtualization : State of the Art and Research Challenges," 2014.

[45] Y. De Bock, J. Broeckhove, and P. Hellinckx, "Hierarchical Real-Time Multi-core Scheduling through Virtualization: A Survey," in *Proceedings - 2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC 2015*, 2016.

[46] J. Zhang, K. Chen, B. Zuo, R. Ma, Y. Dong, and H. Guan, "Performance analysis towards a KVM-

based embedded real-time virtualization architecture," in *Proceeding - 5th International Conference on Computer Sciences and Convergence Information Technology, ICCIT 2010*, 2010.

[47] T. Xia, J. C. Prevotet, and F. Nouvel, "Mini-NOVA: A Lightweight ARM-based Virtualization Microkernel Supporting Dynamic Partial Reconfiguration," in *Proceedings - 2015 IEEE 29th International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2015*, 2015.

[48] W. K. I. Kang, "K-hypervisor," *2016 IEEE 22nd Int. Conf. Embed. Real-Time Comput. Syst. Appl.*, 2016.

[49] C. Dall, S. W. Li, J. T. Lim, J. Nieh, and G. Koloventzos, "ARM Virtualization: Performance and Architectural Implications," in *Proceedings - 2016 43rd International Symposium on Computer Architecture, ISCA 2016*, 2016.

[50] S. Toumassian, R. Werner, and A. Sikora, "Performance Measurements for Hypervisors on Embedded ARM Processors," *2016 Int. Conf. Adv. Comput. Commun. Informatics*, 2016.

[51] P. Varanasi and G. Heiser, "Hardware - Supported Virtualization on ARM," 2011.

[52] C. Dévigne, J.-B. Bréjon, Q. L. Meunier, and F. Wajsbürt, "Executing secured virtual machines within a manycore architecture," *Microprocess. Microsyst.*, 2017.

[53] S. S. Thiebaut, A. D. De Rosa, and R. Sasse, "Secure Embedded Hypervisor Based Systems for Automotive," in *Proceedings - 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN-W 2016*, 2016.

[54] J. Gotzfried, N. Dorr, R. Palutke, and T. Muller, "HyperCrypt: Hypervisor-Based Encryption of Kernel and User Space," in *2016 11th International Conference on Availability, Reliability and Security (ARES)*, 2016.

[55] D. U. Viktor, "Security Services on an Optimized Thin Hypervisor for Embedded Systems," 2011.

[56] R. Schmitt, M. Mateus, K. Ludwich, and A. A. Fröhlich, "Virtualizing Mixed-Criticality Operating Systems."

[57] L. Xu, Z. Wang, and W. Chen, "The Study and Evaluation of ARM-Based Mobile Virtualization," 2014.

[58] J. Shuja, A. Gani, and S. A. Madani, "A Qualitative Comparison of MPSoC Mobile and Embedded Virtualization Techniques," 2016.

[59] S. Groesbrink, "Virtual machine migration as a fault tolerance technique for embedded real-time systems," in *Proceedings - 8th International Conference on Software Security and Reliability - Companion, SERE-C 2014*, 2014.

[60] W. Steiner and S. Poledna, "Fog computing as enabler for the Industrial Internet of Things," vol. 1337, pp. 310–314, 2016.

[61] F. Mehdipour, B. Javadi, and A. Mahanti, "FOG-engine: Towards Big Data Analytics in the Fog."

[62] H. R. Arkian, A. Diyanat, and A. Pourkhalili, "MIST: Fog-based data analytics scheme with cost-efficient resource provisioning for IoT crowdsensing applications," 2017.