Internet de las Cosas (IoT)

Daniel Bolaños Martínez.

Arquitecturas del IoT:

Imagina ser el jefe de ingeniería de un fabricante de motores automotrices. De repente hay una explosión de oportunidades tecnológicas y desafíos. Necesita una nube de dispositivos para diferentes configuraciones y sus capacidades de actualización, integración y prueba para manejar la explosión del estado de sus sistemas, habilidades de seguridad para mantener las máquinas en un modo definido y, por lo tanto, seguras para el público, nuevas aplicaciones de tableta para diagnóstico y reparación, conectividad a sistemas de planificación de recursos empresariales y análisis de Big Data para respaldar la ingeniería y el mantenimiento durante todo el ciclo de vida.

Lo bueno es obvio. Conocer las necesidades típicas de los usuarios y organizarlas en los productos y flujos de trabajo en tiempo real crea una riqueza de valor:

- Flexibilidad: Los productos se pueden adaptar de forma autónoma a los escenarios de uso, como viviendas asistidas, edificios inteligentes, transporte inteligente, energía, atención médica, transporte o cadenas de suministro completas.
- Usabilidad: Incluso los productos complejos pueden operarse más fácilmente, mejorando así la experiencia del usuario y mitigando los peligros.
- Productividad: El servicio se extiende hacia el mantenimiento predictivo y las mejoras proactivas, mejorando el tiempo de actividad y, por lo tanto, la productividad.

El Internet de las cosas (IoT) impulsará una enorme cantidad de innovación, eficiencia y calidad. Conectar los sistemas de producción, médicos, automotrices o de transporte con los sistemas de TI y la información crítica para el negocio proporcionará un gran valor a las organizaciones. Las principales empresas de TI, como Cisco y SAP, han pronosticado miles de millones de dispositivos en red y un universo de servicios de negocios basados en TI, con las expectativas de un negocio de billones de dólares.

Los líderes empresariales conocen la cadena de valor pero no se preocupan por la tecnología. La fabricación se aleja cuando se enfrenta a la tecnología de software. Los departamentos de TI tienden a pasar por alto en su perspectiva grande y distante que hay productos y clientes reales.

La ingeniería de software para el IoT plantea desafíos a la luz de las nuevas aplicaciones, dispositivos y servicios. La movilidad, el desarrollo centrado en el usuario, los dispositivos inteligentes, los servicios electrónicos, los entornos ambientales, la salud electrónica y los dispositivos portátiles o implantables plantean desafíos específicos para especificar los requisitos de software y desarrollar software confiable y seguro. Las interfaces de software específicas, la organización ágil y la confiabilidad del software requieren enfoques particulares para la seguridad, el mantenimiento y la

sostenibilidad. Las arquitecturas de referencia para IoT tienen como objetivo ayudar a los desarrolladores a enfrentar estos desafíos.

Historia:

El término "Internet de las cosas" fue utilizado por primera vez en 1999 por Kevin Ashton, quien trabajó en un estándar para etiquetar objetos utilizando RFID para aplicaciones de logística. Sin embargo, la idea de la computación ubicua se remonta a finales de los años ochenta. Desde entonces, los investigadores han trabajado en muchos sistemas centrados en etiquetas y redes de sensores, middleware y tecnologías en la nube y redes de comunicación.

Se alcanzó un hito muy visible cuando el Grupo de trabajo de ingeniería de Internet lanzó IPv6, el protocolo que habilita la IoT. Recientemente, el IoT ha recibido un impulso del compromiso comercial y el trabajo en arquitecturas de referencia impulsadas por las principales industrias:

- Google anuncia Brillo como un sistema operativo para dispositivos IoT en hogares inteligentes.
- Los dispositivos están disponibles comercialmente para estándares de comunicación de máquina a máquina (M2M) como Bluetooth, ZigBee, estándares de IPC Global y Wi-Fi de baja potencia.
- Samsung y otras compañías han anunciado una nueva generación de chips para dispositivos inteligentes.
- Muchos informes de implementación han descrito microcontroladores en red que sirven como centros para sensores, actuadores y etiquetado.

Arquitecturas de Referencia:

Identificar y estructurar una arquitectura o modelo es un proceso largo y tedioso con mucha negociación para abstraerse de necesidades y tecnologías específicas. Dicha referencia puede servir como una guía general y genérica; no todas las aplicaciones de dominio requerirán cada detalle para la implementación en la vida real.

Sin embargo, los requisitos son fáciles de comprender:

- La conectividad y las comunicaciones son importantes. Esto puede implicar conectividad uno a uno (unicast) o recolección de datos y difusión de información a múltiples socios (multicast y anycast).
- La administración de dispositivos debe proporcionar soluciones una vez que se agrega un dispositivo o cambia la configuración de un dispositivo y se debe propagar a otros dispositivos.
- La recopilación de datos, el análisis y la actuación son relevantes para extraer información y conocimientos para ofrecer servicios.
- La escalabilidad es importante para manejar los mayores volúmenes de procesamiento para diferentes tamaños de instalación.

- Las funciones de seguridad son necesarias para brindar confianza y privacidad, y se requieren para todos los aspectos de la IoT.

Una arquitectura de referencia IoT maneja esos requisitos y forma un superconjunto de funcionalidades, estructuras de información y mecanismos. La consideración adicional de las entidades y su interacción conduce a un modelo de referencia. Dicho modelo integra aspectos de las entidades relacionadas, tales como usuarios humanos, implementaciones de dispositivos y estructuras de servidor, y proporciona una vista más completa o plantillas para la configuración general y su implementación de dominio. Tanto la arquitectura como el modelo ayudan a describir y asignar tecnologías a casos de negocios.

Arquitecturas Disponibles:

Hay dos arquitecturas principales disponibles: la IoT-A y el IIRA. Ambas propuestas de arquitectura se han preparado a fondo, pero la IoT-A se ha descrito en detalle y se ha ampliado. Desde su lanzamiento en 2012, se ha sincronizado con la comunidad IoT e incorpora varias vistas. En contraste, el IIRA todavía tiene como objetivo la retroalimentación y detalles adicionales.

Ahora comparamos estas arquitecturas con respecto a sus capacidades y capas según tres perspectivas.

- La primera perspectiva es la orientación **semántica**: la interpretación de datos e información para crear conocimiento para casos de negocios. IoT-A se concentra en los aspectos genéricos de la informática en lugar de las facetas de aplicación de la semántica. En contraste, el IIRA se centra en la funcionalidad del dominio de la industria, como negocios, operaciones (pronósticos, monitoreo, optimización, etc.), información (análisis y datos) y aplicaciones (IU, API, lógica y reglas). RAMI 4.0, que es específico del dominio, extiende la visión del IIRA hacia el ciclo de vida y los flujos de valor de las aplicaciones de fabricación. En particular, mejora la estructura de la capa funcional en dos dimensiones: el ciclo de vida y el flujo de valores, así como por niveles jerárquicos.
- La segunda perspectiva es la **orientación a internet** y tiene dos aspectos. El primero es el middleware para el soporte de servicios y la gestión de datos en la nube y los servidores. IoT-A cubre en gran medida el modelado y la estructuración de la gestión de procesos de negocios de IoT, las entidades virtuales, los servicios de IoT y la organización de servicios cruzados desde los puntos de vista funcional, de información y de dominio, de manera abstracta. Los aspectos de la nube, en el sentido de la arquitectura del lado del servidor y su gestión, están definidos por la implementación. Lo mismo se aplica a los agentes y el código en dispositivos específicos de dominio. El IIRA también se centra en estos aspectos, pero permanece más cerca de los casos de negocios y uso.

El segundo aspecto es la red, el transporte y los enlaces de datos. Ambas arquitecturas consideran brevemente estas cosas, pero se refieren principalmente a la comunicación M2M para cubrir las capas inferiores de la pila OSI. Por ejemplo, la capa de red podría implementarse mediante IPv6, mientras que la red y el transporte podrían basarse en UDP (User Datagram Prototcol) y CoAP (Constrained Application Protocol). Realizaciones alternativas podrían emplear el protocolo ligero MQTT, que podría utilizarse sobre TCP / IP en lugar de HTTP.

- La tercera perspectiva es la orientación de las cosas, que se centra en activos como sensores, actuadores y etiquetas, que son cruciales tanto en IoT-A como en IIRA. Este es el enfoque clásico para la industria de la automatización, que trata de definir una referencia de abajo hacia arriba sobre objetos tangibles y sus fuentes de datos individuales y sus necesidades de información.

Ambas arquitecturas tienen mecanismos de gestión y seguridad en todas las capas. Las propuestas de arquitectura ayudan a definir y explicar la estructura general de IoT. Proporcionan modelos descriptivos de cómo los dispositivos de IoT y los seres humanos interactúan y procesan datos, incorporando patrones de estándares de comunicación M2M. Estos modelos tienen diferentes perspectivas y granularidades en la descripción de IoT.

Problemas del Despliegue:

El IoT es rico en enfoques, conceptos y estructuras. Varias iniciativas ya han entregado modelos, arquitecturas y herramientas de IoT. Sin embargo, se requiere una mayor convergencia de enfoques y estándares industriales para la simplificación. Esto plantea cuestiones sobre los habilitadores clave de IoT.

Un habilitador clave es la comunicación eficiente y segura. Existen muchas tecnologías para implementar la pila de comunicación. Se requieren redes inalámbricas de bajo consumo que requieren poco esfuerzo de implementación, para lo cual la seguridad es un factor importante.

El hardware del dispositivo adecuado para implementar el IoT está fácilmente disponible. Sin embargo, es cuestionable si los sistemas operativos en red en tiempo real y orientados a dispositivos existirán para los diferentes dominios de aplicación o si la diversidad de los sistemas operativos en tiempo real continuará.

Finalmente, el análisis de Big Data y la interfaz hombre-máquina forman la interfaz para los usuarios y están cerca del caso de negocios. Debido a las características individuales de las aplicaciones, todavía no hay mucha estandarización disponible. Sin embargo, un primer paso hacia la estandarización podría ser la descripción semántica. Los idiomas estandarizados, como Vorto o Weave, son importantes para describir dispositivos, parámetros, interfaces de usuario, etc.

Hoja de Ruta al Mundo Programable:

La aparición de millones de dispositivos programables de forma remota en nuestro entorno planteará desafíos importantes para los desarrolladores de software. Una hoja de ruta que va desde los sistemas de Internet de las Cosas centrados en la nube de hoy hasta el Mundo Programable destaca los desafíos que no han recibido suficiente atención.

La mayoría de las investigaciones de IoT actualmente giran en torno a la adquisición de datos, el análisis en tiempo real y fuera de línea, el aprendizaje automático, la visualización de datos y otros temas de Big Data. Este enfoque no es sorprendente, dado el enorme potencial comercial que se deriva de la capacidad de recopilar datos de millones de sensores. Los dispositivos digieren y combinan esos datos para proporcionar nuevos conocimientos sobre el comportamiento de las personas y otros fenómenos del mundo real.

Los avances de hardware y la disponibilidad general de chips integrados potentes pero económicos harán posible integrar conectividad y máquinas virtuales de pleno derecho y tiempos de ejecución dinámicos de lenguaje en todas partes. Por lo tanto, nuestras cosas cotidianas (bombillas, pomos de puertas, sistemas de aire acondicionado, rociadores de césped, aspiradoras, cepillos de dientes y el fregadero de la cocina) se conectarán y se podrán programar dinámicamente.

Aquí, presentamos una hoja de ruta desde los sistemas de IoT centrados en la nube y los datos de hoy a un mundo donde los objetos cotidianos están conectados y la ventaja de la red es realmente programable. El mundo programable, programable en un sentido literal, planteará nuevos desafíos para los desarrolladores de software. Los métodos de desarrollo, los lenguajes y las herramientas actuales, o al menos aquellos que se usan ampliamente, no son adecuados para el surgimiento de millones de cosas programables en nuestro entorno. Destacamos problemas y desafíos técnicos que merecen un estudio más profundo más allá de los temas de IoT que reciben la mayor atención en la actualidad.

Arquitectura emergente End-to-End para IoT:

El término "Internet de las cosas" no es nuevo. Hace más de 20 años, los profesores del MIT describieron un mundo en el que las cosas (dispositivos o sensores) están conectadas y pueden compartir datos. Un estudio reciente enumera 115 plataformas en la nube de IoT. Actualmente estamos presenciando un período clásico de "cruce del abismo" en el que los ganadores finales aún no han sido determinados. Lo más probable es que, para 2025, solo un puñado de plataformas y plataformas dominantes de IoT Los vendedores permanecerán.

Lo interesante de estas ofertas de IoT es su similitud conceptual. A pesar de la aparente diversidad y la gran cantidad de proveedores que se dirigen al mercado de IoT, está

surgiendo una arquitectura común de extremo a extremo para las soluciones de IoT, con una serie de elementos que son prácticamente iguales en todos los sistemas.

Dispositivos:

Los dispositivos (o periféricos) son los elementos de hardware que recopilan datos del sensor o realizan una actuación, con capacidades de comunicación integradas para enviar los datos recopilados al ecosistema de IoT más amplio. Funcionalmente, los dispositivos son sensores o actuadores.

Los sensores proporcionan información sobre la entidad física que monitorean. Esta información puede abarcar desde la identidad de la entidad física hasta cualidades mensurables como temperatura, humedad, presión, luminosidad, nivel de sonido, flujo de fluido, vibración y abrasión. Los sensores cuyo único propósito es facilitar un proceso de identificación se denominan etiquetas.

Los actuadores toman energía, generalmente transportada por aire, electricidad o líquido, y la convierten en un cambio de estado, modificando así una o más entidades físicas.

Puertas de acceso:

Los gateways (o hubs) recopilan, preprocesan y transfieren datos de los dispositivos IoT y sus sensores, empleando diferentes protocolos de comunicación (generalmente inalámbricos) como Wi-Fi o Bluetooth Smart. Las puertas de enlace proporcionan una traducción segura del protocolo entre los dispositivos y la nube, y pueden admitir tareas como el almacenamiento y preprocesamiento de datos de sensores intermedios, descubrimiento de servicios, geolocalización, verificación y facturación. Los gateways también entregan las solicitudes de actuación desde la nube a los dispositivos.

En algunos sistemas, los propios dispositivos pueden cargar datos de detección directamente a la nube y recibir solicitudes de activación directamente desde la nube, por ejemplo, a través de redes Wi-Fi, 3G o 4G, o (pronto) NB-IoT (NarrowBand IoT) y 5G. Tales soluciones no requieren puertas de enlace dedicadas. Además, los dispositivos en sí mismos suelen actuar como puertas de acceso a otros dispositivos, posiblemente formando topologías de punto a punto o de malla a través de redes locales.

La Nube:

La computación en la nube y las soluciones de almacenamiento y análisis basadas en la nube son fundamentales para la mayoría de las plataformas de IoT. En la arquitectura genérica de IoT de extremo a extremo, la nube desempeña tres roles principales.

El primero es la adquisición de datos, el almacenamiento y el acceso. Una funcionalidad fundamental en los sistemas de IoT es la recopilación y almacenamiento de datos de sensores. Los dispositivos de IoT con capacidades de detección generalmente recopilan una gran cantidad de datos que deben ser recolectados y almacenados en la nube para su

posterior procesamiento y análisis. Las soluciones en esta área van desde bases de datos locales simples hasta clusters de almacenamiento escalables, tolerantes a fallos y masivos. También se proporcionan las API de consulta y notificación para acceder a los datos recopilados.

El segundo rol es el análisis de datos. Esto se refiere a examinar, conectar y transformar los datos adquiridos del sensor para descubrir y presentar información útil, por ejemplo, para permitir el intercambio remoto de información y la toma de decisiones. La analítica en tiempo real se refiere a las funciones de análisis que se ejecutan inmediatamente después de haber recibido los datos. La analítica fuera de línea se refiere a las operaciones de estilo de lote que se producen después de que ya se hayan acumulado grandes conjuntos de datos. Las tecnologías y algoritmos de aprendizaje automático y de extracción de datos son importantes en esta área.

El tercer papel es el soporte de actuación. Los flujos de datos del sistema de IoT no son solo unidireccionales. Además de la recopilación de datos de sensores desde dispositivos a la nube, es importante la activación segura de los dispositivos desde la nube a los dispositivos. Las capacidades de actuación son un habilitador fundamental para la capacidad de programación remota de dispositivos.

Además, una solución en la nube que admite el IoT suele incluir funciones administrativas como la administración de dispositivos, la administración de cuentas de usuarios, el registro de uso, la supervisión del estado del servidor y las capacidades de informes.

La Hoja de Ruta:

Creemos que el futuro de IoT se basa en la capacidad de organizar y programar topologías grandes y complejas de dispositivos IoT de forma remota. Las capacidades de actuación que ofrecen los dispositivos de IoT forman la base de todo esto. Estas capacidades nos permiten controlar y controlar los objetos cotidianos en nuestro entorno (y potencialmente en todo el planeta) desde la comodidad de un entorno de programación o una aplicación frente a nosotros.

Como mencionamos anteriormente, los avances de hardware conducirán la transición hacia el Mundo Programable. Las capacidades de hardware y los puntos de precio de IoT están alcanzando rápidamente un punto de inflexión en el que podremos ejecutar sistemas operativos completos, como Linux o pilas de software virtualizado o tiempos de ejecución de lenguaje dinámico en casi cualquier tipo de dispositivo. Los chips de bajo costo que han estado disponibles recientemente ya coinciden o superan la memoria y la capacidad de procesamiento de los teléfonos móviles a fines de la década de 1990.

Otra tendencia importante que conduce a la industria hacia el mundo programable es la computación de borde (Edge Computing). Los sistemas de computación en la nube clásicos están altamente centralizados. Si bien la computación centralizada tiene beneficios significativos, puede ser costosa en términos de comunicación y consumo de

energía. Considere un entorno de IoT que consiste en una gran colección de dispositivos que están cerca uno del otro. Puede ser ineficiente transmitir los datos de esos dispositivos a un centro de datos remoto para procesarlos y luego transmitir las solicitudes de actuación desde el centro de datos a los dispositivos. En una implementación de IoT con estrictos requisitos de latencia, la sobrecarga de latencia por sí sola puede hacer que tales soluciones sean poco prácticas.

El término "computación de borde" se acuñó alrededor de 2002 y se asoció originalmente con el despliegue de aplicaciones en redes de entrega de contenido (CDN). El objetivo principal fue beneficiarse de la proximidad de los servidores de borde de CDN para mejorar la escalabilidad y una menor latencia. Las soluciones IoT de Edge Computing aprovechan el borde de la red (generalmente dispositivos de puerta de enlace como enrutadores o estaciones base) para el cálculo, por ejemplo, para preprocesar datos de sensores y activar alertas y solicitudes de actuación localmente sobre la base de criterios predefinidos. Dichos sistemas podrían aprovechar las tecnologías de conectividad local (por ejemplo, Bluetooth Smart o Wi-Fi Direct) para permitir una comunicación directa, más eficiente y descentralizada entre los dispositivos sensores. Las tecnologías de virtualización también desempeñan un papel central en permitir la migración de cómputo entre entidades.

Básicamente, esperamos que las capacidades de programación de IoT evolucionen desde funciones de actuación simples, aplicaciones específicas del proveedor, API de dispositivos y adquisición y procesamiento de datos centrados en la nube, hasta sistemas que aprovechan ampliamente la informática, la virtualización y los contenedores. Dichos sistemas admitirán el desarrollo de aplicaciones portátiles, de fabricantes cruzados y de diferentes industrias. También permitirán, cuando sea necesario, una migración flexible de computación y datos entre la nube y dispositivos de borde heterogéneos. Es seguro predecir que en 5 a 10 años, los dispositivos IoT y sus API habrán convergido significativamente.

Qué hace diferente al desarrollo de las IoT?

El desarrollo de IoT se diferencia del desarrollo de aplicaciones móviles y de aplicaciones web del lado del cliente en seis formas. Estas diferencias son razones clave de las implicaciones y desafíos técnicos que presentamos más adelante.

Primero, los dispositivos de IoT casi siempre son parte de un sistema más grande de dispositivos, por ejemplo, una instalación de dispositivos interoperativos en una casa, una oficina, una fábrica, un centro comercial o un avión. Además, los dispositivos de IoT generalmente son solo una pequeña parte de la arquitectura de extremo a extremo que analizamos anteriormente. Por supuesto, las PC y los teléfonos inteligentes también tienen grandes dependencias en los servicios basados en la nube. Sin embargo, desde la perspectiva del desarrollador de la aplicación, todavía son principalmente dispositivos independientes, con el desarrollador dirigido a una sola computadora o teléfono inteligente.

En segundo lugar, los sistemas de IoT nunca duermen. Las PC, los teléfonos inteligentes y otros dispositivos informáticos independientes se pueden ver como "rebootables": sistemas que pueden y se reiniciarán cuando las cosas vayan mal. En contraste, los sistemas de IoT generalmente no deben o no pueden cerrarse en su totalidad. Aunque los dispositivos individuales pueden apagarse, todo el sistema debe ser resistente a las interrupciones del dispositivo y de la red.

En tercer lugar, la cantidad de unidades informáticas (dispositivos o CPU) en los sistemas de IoT suele ser mucho mayor que en los entornos informáticos tradicionales, y puede llegar a cientos, miles o incluso millones. A diferencia de las PC y los teléfonos inteligentes.

En cuarto lugar, los dispositivos de IoT a menudo están integrados en nuestro entorno, por lo que son físicamente invisibles e inalcanzables. Los dispositivos pueden estar enterrados permanentemente en lugares subterráneos o físicamente incrustados en varios materiales (por ejemplo, sensores de vibración en equipos de minería). Podría ser imposible conectar cables físicos a esos dispositivos o reemplazar hardware o software integrado cuando surjan problemas.

En quinto lugar, los sistemas de IoT son altamente heterogéneos. Sus unidades de computación pueden tener una capacidad de computación, capacidades de almacenamiento, ancho de banda de red y requisitos de energía que varían dramáticamente. Los mecanismos de E/S, las capacidades sensoriales y las modalidades de entrada admitidas también pueden variar considerablemente. Algunos dispositivos tienen botones físicos y pantallas, mientras que muchos dispositivos no tienen una interfaz de usuario visible.

Sexto, los sistemas de IoT tienden a tener una conectividad débil, con conexiones de red intermitentes y, a menudo, poco confiable. Desde el punto de vista del desarrollador de software, tal entorno impone el requisito de prepararse constantemente para el fracaso. Es probable que las aplicaciones con un manejo insuficiente de errores se detengan durante un apagón del dispositivo o de la red, esperando infinitamente un paquete de respuesta o consumiendo excesivamente memoria, energía u otros recursos.

Además de esas diferencias básicas, muchos problemas adicionales surgen de la relativa inmadurez del campo IoT. Aquí hay solo dos de ellos.

Implicaciones y desafíos para el desarrollo de software:

Para resumir las diferencias que acabamos de presentar, los desarrolladores de IoT deben considerar varias dimensiones que no son familiares para la mayoría de los desarrolladores de aplicaciones web y aplicaciones web del lado del cliente, incluyendo

- programación multidispositivo;
- la naturaleza reactiva y siempre activa del sistema;
- heterogeneidad y diversidad;

- la naturaleza distribuida, altamente dinámica y potencialmente migratoria del software; y la necesidad general de escribir software de forma defensiva y tolerante a fallos.

Una aplicación típica de IoT es continua y reactiva. Sobre la base de las lecturas observadas de los sensores, los cálculos se activan (o se vuelven a activar) y eventualmente resultan en varios eventos procesables. Los programas son esencialmente asíncronos, paralelos y distribuidos.

Estrictamente hablando, estas características no son de ninguna manera nuevas en el desarrollo de software. Cualquier desarrollador que haya creado software para sistemas distribuidos o de misión crítica está al menos algo familiarizado con los desafíos que surgen de estas características. Los desarrolladores del software de clúster de servidores de back-end en la nube generalmente también enfrentan estos desafíos.

Lenguajes y herramientas inadecuados:

Los lenguajes de programación y las herramientas que las personas utilizan para el desarrollo de IoT son en gran medida los mismos que se utilizan para el desarrollo de aplicaciones web y de aplicaciones para el cliente. Por ejemplo, los kits de herramientas de desarrollo de software para los tableros de desarrollo de IoT populares de hoy en día, como Arduino, Espruino, Edison y Galileo de Intel y Tessel, ofrecen la opción de desarrollo de C, C #, Java, JavaScript o Python.

Contra todo pronóstico, JavaScript y Node.js (la versión del lado del servidor de JavaScript) se están convirtiendo en herramientas centrales para el desarrollo de IoT debido a su popularidad en el desarrollo web. Esto es desafortunado porque JavaScript no fue diseñado para escribir aplicaciones distribuidas asíncronas o para la programación en general. Recientemente, la expresión "infierno de devolución de llamada" se ha hecho popular en el contexto de JavaScript para caracterizar situaciones en las que la lógica de la aplicación se vuelve imposible de seguir debido a las llamadas de función asíncronas y las funciones de manejo de eventos separadas utilizadas para escribir rutinas de manejo de errores y errores. El mecanismo de promesa agregado en el estándar ECMAScript 6 alivia este problema. Sin embargo, es justo decir que JavaScript no es óptimo para desarrollar ningún sistema de software distribuido.

Los lenguajes de desarrollo de IoT actualmente populares tampoco abordan los aspectos de programación en general. Es decir, no proporcionan instalaciones para sistemas de orquestación que consistan en miles de dispositivos ni mecanismos que permitan que el código migre de manera flexible entre la nube, las puertas de enlace y los dispositivos.

Mirando hacia el futuro, mientras aprendemos del pasado:

Las observaciones y los desafíos anteriores revelan que la aparición de Programmable World requerirá mucho más que el desarrollo de un nuevo hardware, nuevos protocolos y tecnologías de comunicación o nuevas técnicas para digerir y analizar conjuntos de datos masivos. Para aprovechar todo el poder del mundo programable, necesitaremos

nuevas tecnologías, procesos, metodologías, abstracciones y herramientas de ingeniería y desarrollo de software. La experiencia pasada y las tecnologías para desarrollar sistemas distribuidos y software de misión crítica desempeñan un papel importante para evitar trabajos duplicados y reinventar la rueda. En gran parte, los desafíos deben abordarse educando a los desarrolladores de software para que se den cuenta de que el desarrollo de IoT realmente difiere del desarrollo de aplicaciones móviles y aplicaciones web del lado del cliente.

Seguridad:

Un gran desafío para la realización del mundo programable es la seguridad. La gestión remota de instalaciones complejas de dispositivos de IoT en entornos como fábricas, centrales eléctricas o plataformas petroleras obviamente requiere la máxima atención a la seguridad. Las capacidades de activación y programación remotas pueden presentar altos riesgos de seguridad. Los protocolos criptográficos para la seguridad de la capa de transporte, los certificados de seguridad, el aislamiento físico y otras prácticas establecidas de la industria desempeñan un papel fundamental en esta área, pero persisten varios desafíos técnicos interesantes.

Depuración en IoT:

Un principio básico de la IoT es que los sistemas informáticos no tienen fallas intrínsecas. El cumplimiento de este principio requiere técnicas y herramientas de desarrollo de software impecables. No es una tarea fácil, considerando que incluso los sistemas informáticos en red más tradicionales, que tienen un menor grado de distribución, recursos más amplios y menos comportamiento dinámico, no están a la altura del desafío.

La situación es especialmente grave en el contexto de las redes de sensores inalámbricas (WSN): pequeños dispositivos de detección alimentados por batería que interactúan mediante la comunicación inalámbrica. Las WSN son la base de la IoT en el sentido de que son sus antenas, lo que permite la percepción sobre la cual otros dispositivos reaccionan.

Los desafíos de la programación para WSN nos han llevado a reconsiderar las mejores prácticas para desarrollar sistemas de software confiables. Específicamente, debido a la dificultad inherente de predecir todas las condiciones de ejecución, hemos investigado técnicas de depuración en tiempo de ejecución para WSN. Desafortunadamente, las restricciones de hardware y recursos hacen que las técnicas existentes sean inaplicables para esto.

Los desafíos de la programación WSNS confiable:

Podría sentirse tentado a pensar que el pequeño factor de forma de tales dispositivos de detección y los actuadores hace que sean más fáciles de programar que, por ejemplo, las computadoras de escritorio y, por lo tanto, más manejables en lo que respecta a la

seguridad. Sin embargo, cuatro características a la inversa dificultan la fiabilidad de estos dispositivos.

Primero, muchos dispositivos pequeños emplean hardware especializado. Esto requiere nuevos paradigmas de software y herramientas que serán menos maduras que las correspondientes a los sistemas tradicionales.

Segundo, el hardware pequeño y especializado a menudo tiene limitaciones de recursos. Además de obstaculizar la aplicación de las cadenas de herramientas existentes, esto hace que la aplicación de las técnicas existentes de tolerancia a fallos sea difícil o incluso imposible, dado que la mayoría de estas técnicas emplean redundancia (por ejemplo, en el espacio y la memoria o en los ciclos de procesamiento y tiempo).

En tercer lugar, muchos escenarios de IoT requieren una interacción dinámica debido a la naturaleza abierta de los contextos físicos de la vida real a los que se dirigen estas aplicaciones. Esto hace que sea difícil prever todas las restricciones y validar completamente los sistemas antes de desplegarlos.

Finalmente, los entornos dinámicos exacerban aún más la dificultad de predecir de manera exhaustiva los contextos de ejecución. Muchos escenarios de IoT incluyen entornos precisamente austeros que dependen de la tecnología para proteger a los humanos.

Depuración de nodos restringidos de recursos:

Los nodos de sensores inalámbricos (comúnmente llamados motes) están extremadamente limitados por los recursos. Por ejemplo, la familia de motes mica incluye típicamente procesadores de 8 bits con 4 Kbytes de datos o memoria de programa y 128 Kbytes de memoria flash. Esto hace que la programación de tales dispositivos sea un reto; Estos desafíos se trasladan al diseño de soluciones para depurar el software correspondiente. En particular, la depuración completa que permite la reproducción precisa de ejecuciones completas es inviable, y requiere soluciones que rastreen (mantengan un registro de) solo ciertos aspectos de la ejecución en motes.

Flujo de control de rastreo:

TinyTracer es una herramienta que concebimos para rastrear el flujo de control en motes que ejecutan TinyOS, sobre la base de las siguientes ideas.

Eficacia:

La información de control-flujo es efectiva en el diagnóstico de fallas porque muchos defectos se manifiestan a través de un flujo de control anormal. El conocimiento del flujo de control puede ayudar considerablemente en el diagnóstico de una gran clase de errores que cambian el flujo de control. Tales errores incluyen errores lógicos, condiciones de carrera de alto nivel, reinicios de nodo, fallas de red (fallas de nodo o enlace) y defectos de memoria.

Las trazas de flujo de control también capturan adecuadamente algunos efectos de los valores de entrada, como los valores detectados y los mensajes de red. Por ejemplo, si un mensaje de red representa un comando al sensor para enviar los valores detectados actuales a la estación base, el flujo de control refleja el contenido del mensaje. Del mismo modo, si el valor detectado supera un umbral, el flujo de control captura la acción que tomó el sensor.

Para defectos como la corrupción de la tabla de enrutamiento por entidades malintencionadas, registrar valores de entrada podría ser útil.

Sin embargo, tal enfoque no es práctico para las WSN debido a los grandes tamaños de rastreo y las restricciones estrictas en la memoria. Las trazas de flujo de control producen un buen compromiso que permite una reproducción parcial de la ejecución.

Eficiencia:

Las aplicaciones WSN ejecutan repetidamente las mismas secuencias de acciones, con eventos inusuales ocasionales. Por lo tanto, si la ruta del flujo de control se codifica adecuadamente, las secuencias repetitivas se pueden comprimir altamente, lo que lleva a un registro eficiente de los recursos de la información del flujo de control.

Explotando TinyOS:

El modelo de ejecución impulsado por eventos de TinyOS plantea desafíos de programación únicos. Sin embargo, permite soluciones eficientes para entornos severamente limitados de recursos, en comparación con el modelo de ejecución de entornos de programas de propósito general, que permiten intrusiones arbitrarias de hilos.

Una aplicación TinyOS a menudo comprende un conjunto de componentes reutilizables que están "conectados" a través de configuraciones. Los componentes se comunican a través de interfaces que consisten en comandos y eventos. Un componente proporciona servicios a otros componentes a través de comandos. Si un componente solicita un servicio, un evento señala la finalización de la solicitud al componente. Los eventos son también los canales a través de los cuales el hardware se comunica con los componentes.

No hay una abstracción explícita de hilos porque el mantenimiento de múltiples hilos necesita una memoria RAM valiosa y porque los entrelazamientos de hilos fácilmente introducen errores de carrera de datos sutiles. No obstante, las aplicaciones TinyOS necesitan un mecanismo para operaciones paralelas para asegurar la capacidad de respuesta y respetar las restricciones en tiempo real. TinyOS tiene dos fuentes de concurrencia: tareas e interruptores de manejo.

Las tareas son un mecanismo de cálculo diferido. En ausencia de eventos, se ejecutan hasta el final y no se adelantan entre sí. Se publican por componentes. La solicitud posterior se devuelve inmediatamente, aplazando el cálculo hasta que el programador

ejecute la tarea más adelante. Para garantizar una latencia de ejecución de tareas baja, las tareas individuales deben ser cortas. Las operaciones prolongadas se deben repartir entre múltiples tareas.

Los manejadores de interrupciones corresponden a eventos y, por lo tanto, a menudo se denominan eventos. Los eventos también se ejecutan hasta su finalización, pero, a diferencia de las tareas, pueden anticiparse a la ejecución de una tarea u otro evento. Un evento significa la finalización de una operación prolongada (y, por lo tanto, dividida) o un evento del entorno (por ejemplo, la recepción de mensajes o el paso del tiempo). En última instancia, la ejecución de TinyOS está dirigida por eventos que representan interrupciones de hardware.

Las ejecuciones de tareas y eventos son secuenciales o anidadas. Esto proporciona la base para nuestro rastreo eficiente en TinyTracer. TinyTracer aprovecha el conocimiento de las posibles ejecuciones para codificar mínimamente las ejecuciones reales, incluido el flujo de control intraprocedimiento e interprocedimiento basado en una variante específicamente diseñada de codificación de trayectoria de Ball-Larus.5 Para obtener más detalles de nuestro enfoque, consulte la barra lateral "Trazado a nivel de nodos con Tiny Tracer.”

Depuración a través de los nodos:

Muchos defectos de WSN implican la interacción entre varios nodos. La reproducción de WSNs, por lo tanto, requiere que las interacciones se realicen con fidelidad y en el orden correcto. Una reproducción correcta mantiene el ordenamiento causal de los mensajes observados en la ejecución original. Este ordenamiento se define generalmente de la siguiente manera:

- Pedido FIFO. Un envío de mensaje precede causalmente a su recepción y envíos posteriores por el mismo proceso.
- Orden local. Un mensaje m1 recibido por un nodo antes de enviar un mensaje m2 precede causalmente a m2. • La transitividad. El ordenamiento causal es transitivo; si m1 causalmente precede m2 y m2 causalmente precede m3, entonces m1 causalmente precede m3.

Compresión de trazas

En WSN con recursos limitados, cada bit cuenta. Los algoritmos de compresión existentes son inaplicables o requieren adaptación para satisfacer los límites de memoria y recursos de la CPU. Las adaptaciones de estos algoritmos siguen siendo deficientes para los rastreos de WSN debido a los buffers de entrada inherentemente pequeños. Estos buffers son solo unos pocos cientos de bytes en WSN, lo que da lugar a pocas oportunidades para aprender y reemplazar los patrones de repetición.

También tienen un rendimiento relativamente bajo porque no pueden explotar las ricas repeticiones en las trazas.

Nuestro enfoque, denominado Prius por el automóvil híbrido de Toyota; se basa en dos observaciones clave:

Los cálculos de WSN exhiben mucha repetición en poco tiempo.

Los patrones repetitivos en los cálculos de WSN evolucionan solo un poco con el tiempo.

Por lo tanto, Prius emplea capacitación fuera de línea para capturar los patrones repetitivos de WSN que ocurren en las trazas. Incluye aquellos patrones en la memoria del programa utilizando estructuras de datos especialmente adaptadas. El algoritmo de compresión utiliza estos patrones para realizar la compresión en línea. La memoria del programa puede acomodar más patrones que la memoria de datos y, por lo tanto, mejorar potencialmente la relación de compresión.

Las estructuras de datos especializadas pueden contrarrestar este aumento al simplificar considerablemente las búsquedas, pero esto no admite la adición de patrones en tiempo de ejecución. Tales patrones faltantes podrían reducir el rendimiento de compresión. El ahorro de energía obtenido a través de tasas de compresión más altas supera los costos adicionales de CPU. Además, los patrones faltantes son raros porque las ejecuciones de WSN son repetitivas y no evolucionan mucho. Podemos manejar cambios más sustanciales en los patrones de ejecución que surgen de la reprogramación de una WSN cargando un nuevo conjunto de patrones para la última versión del software.

La necesidad de tecnologías eficientes en el uso de recursos persistirá. La historia de la computación ha demostrado que tan pronto como podemos empaquetar más recursos en menos espacio, ocurre una de estas dos cosas. O intentamos explotar esto para hacer que los dispositivos existentes sean más pequeños y más ubicuos, o intentamos implementar una funcionalidad y software más complejos en los dispositivos ahora más potentes.

Referencias:

- Michael Weyrich, Christof Ebert: Reference Architectures for the Internet of Things. IEEE Software 33(1):112-116, January/February 2016. DOI http://dx.doi.org/ 10.1109/MS.2016.20
- Antero Taivalsaari, Tommi Mikkonen: A Roadmap to the Programmable World: Software Challenges in the IoT Era. IEEE Software 34(1):72-80, January/February 2017. DOI http://dx.doi.org/10.1109/MS.2017.26
- Patrick Eugster, Vinaitheerthan Sundaram, Xiangyu Zhang: Debugging the Internet of Things: The Case of Wireless Sensor Networks. IEEE Software 32(1):38-49, January/February 2015. DOI http://dx.doi.org/10.1109/MS.2014.132