La programación Reactiva y de Actor en entornos de Internet de las Cosas

Nelson Rodríguez[1], María Murazzo[1], Susana Chávez[1], Tatiana Runco[3], Diego Medel[1], Javier Rosenstein[2]

[1] Departamento e Instituto de Informática - F.C.E.F. y N. - U.N.S.J.
[2] Instituto de Investigaciones, Facultad de Informática y Diseño, Universidad Champagnat
[3] Alumno Avanzado Licenciatura en Cs. de la Computación - F.C.E.F. y N. - U.N.S.J.

Complejo Islas Malvinas. Ignacio de la Roza y Meglioli. C.P. 5402. Rivadavia. San Juan, 0264 4234129

nelson@iinfo.unsj.edu.ar, marite@unsj-cuim.edu.ar, schavez@iinfo.unsj.edu.ar, tatuj@gmail.com, dmedel@iinfo.unsj.edu.ar, rosensteinjavier@uch.edu.ar

Resumen

El surgimiento de Internet de las Cosas, la Computación Móvil y la rápida adopción de Cloud Computing han cambiado el panorama que presenta la computación distribuida. Estos sistemas distribuidos masivos generarán una cantidad excesiva de datos, que saturarán a las redes. Llevar servicios de red, cómputo y otras funcionalidades al extremo medio (fog) o final (mist) permitirá el uso eficiente de estos recursos, para mantener la continuidad fluida desde el Edge al Cloud. Además, IoT presenta características singulares y los desarrollos actualmente se realizan con herramientas y lenguajes de programación convencionales no adecuados para este tipo de sistemas. Estos lenguajes en su mayoría son inadecuados para desarrollar aplicaciones altamente distribuidas, debido a que no soportan mensajes asincrónicos y los errores se pueden propagar, entre otras limitaciones. Por otro lado la programación reactiva y el modelo de programación de actor proveen recursos adecuados para el desarrollo de soluciones para IoT.

La presente propuesta de investigación analizará y evaluará las ventajas de la programación de actor y la programación reactiva, aplicándolas a una variedad de entornos de IoT que favorezcan la continuidad del Edge al Cloud.

Palabras clave: *IoT, Actor Model Programming, Reactive Programming, Edge Computing,*

Contexto

El presente trabajo se encuadra dentro del área de I/D Innovación en Sistemas de Software y se enmarca dentro del proyecto de investigación: Orquestación de servicios para la Continuidad de Edge al Cloud, que ha sido aprobado y está en desarrollo para el período 2018-2019. Asimismo el grupo de investigación viene trabajando en proyectos relacionados con la computación móvil, distribuida y de alta performance desde hace más de 18 años. En esta oportunidad se han incorporado investigadores de otras universidades de la región lo cual impactará en todas las actividades planificadas. Las unidades ejecutoras para dicho proyecto son el Departamento e Instituto de Informática de la FCEFyN de la UNSJ.

Introducción

Internet de las cosas (IoT) se puede definir como un conjunto de diferentes sistemas (por ejemplo, carreteras inteligentes, edificios inteligentes, redes inteligentes) compuesto de componentes heterogéneos pero interactivos (por ejemplo, humanos, automóviles, teléfonos inteligentes, pasarelas, medidores inteligentes), tanto individual como

colectivamente, que proporcionan servicios ciberfísicos innovadores. En resumen, puede describirse como un ecosistema denso, a gran escala, abierto y dinámico de entidades sociotécnicas y aplicaciones [1].

A pesar de comunicarse con redes inalámbricas, presentan características distintivas: no existen estándares consolidados por lo que la variedad de tecnologías es importante, existen varias arquitecturas de software, las velocidades de transferencia en general son bajas y la longitud de las estructuras de datos son menores que las redes IP en general.

En 1999, Kevin Ashton del MIT acuñó el término IoT durante una conferencia en Procter & Gamble [2].

Permite que una gran cantidad de dispositivos cooperen para lograr una tarea común. Cada dispositivo individual es pequeño y ejecuta un núcleo de software limitado. La inteligencia colectiva se obtiene de la colaboración distribuida y la comunicación por Internet. Las soluciones correspondientes de IoT forman grandes sistemas de software distribuidos que presentan requisitos profesionales: escalabilidad, confiabilidad, seguridad, portabilidad y mantenibilidad

En la actualidad, IoT puede incluir productos industriales, comerciales, cotidianos (lavavajillas y termostatos) y redes locales de sensores para vigilar granjas y ciudades [3].

Estas soluciones que ofrece IoT promueve la incorporación de los mismos a la red y se pronostica que entre 20 y 50 millones de dispositivos se añadirán a Internet para 2020, creando una economía de más de 3 billones de dólares [4]; en consecuencia, 43 billones de gigabytes de datos serán generados y necesitarán ser procesados en los centros de datos de Cloud. Las aplicaciones que generan datos en dispositivos de usuario, como teléfonos inteligentes y tablets, usan actualmente al Cloud como un servidor centralizado, pero pronto se convertirá en un modelo informático insostenible [5]. Se ha comenzado a comprobar que en la actualidad Cloud Computing está encontrando serias dificultades para satisfacer los requerimientos de IoT.

Por ello, surgió como propuesta de solución llevar el almacenamiento, las funciones de red, gran parte del procesamiento hacia el borde de la red, lo que resultó en un nuevo modelo llamado edge computing [6]. En particular si se realiza en los dispositivos, sensores y actuadores se denomina mist computing, mientras que cuando se desarrolla en el extremo medio (fundamentalmente routers o switchers de red) se denomina fog computing.

La programación para este tipo de sistemas se realiza utilizando lenguajes convencionales, los cuales presentan varias limitaciones para desarrollar aplicaciones para IoT. Los mismos no son de naturaleza asincrónica, no soportan aislación de errores y los mismos se pueden propagar (es común tener errores debido a descargas de baterías o falla de conexión por alcance, entre otras causas). Por lo tanto es conveniente utilizar otro tipo de lenguajes, herramientas o frameworks que permitan desarrollar aplicaciones eficientes para estos entornos.

La programación reactiva y el modelo de programación de actor presentan características que dan respuesta a esta problemática.

La Programación Reactiva (RP) es un paradigma de programación donde un sistema se describe en términos de cambio de valores en el tiempo y la propagación de los cambios [7]. En años recientes, RP ha ganado popularidad en varios campos: programación web (React, Vue, y muchos frameworks web similares), desarrollo de aplicaciones móviles (React Native, RxJava, etc.), redes IoT móviles [8], entre otros.

El modelo de programación de actor, si bien fue definido como un modelo matemático en 1973, no ha llegó a difundirse y pocos lenguajes de programación lo han soportado. Recientemente algunos lenguajes de programación como Erlang (Ericsson), Calvin (Ericsson) o Elixir (José Valim - que se ejecuta sobre la máquina virtual de Erlang) o bibliotecas como Akka (Jonas Boner – está escrito en Scala), hacen hincapié en el modelo de programación de actor.

El mismo es inherentemente distribuido, cada actor no comparte estado con el resto de los actores y puede ser ubicado en diferentes máquinas virtuales en diferentes nodos y tienen además re despliegue en tiempo de ejecución.

Programación Reactiva

El término reactivo se ha utilizado para describir las propiedades de ambos lenguajes de programación. y sistemas. Un lenguaje de programación es reactivo si proporciona las construcciones para programadores para implementar de forma declarativa aplicaciones controladas por eventos. Por otra parte, un sistema es reactivo si puede responder a las entradas de manera oportuna [9].

Su concepción y evolución ha ido ligada a la publicación del Reactive Manifiesto [10], que establecía las bases de los sistemas reactivos. Dicho manifiesto indica las condiciones que debe cumplir para lograr el uso eficiente de los recursos y una oportunidad para que las aplicaciones escalen automáticamente.

Un sistema reactivo cumple lo siguiente:

1. Responsive (Sensible): Reacciona a los usuarios. Una aplicación receptiva satisfará las expectativas del usuario en términos de disponibilidad y comportamiento en tiempo real. La latencia es una de las medidas clave a la hora de evaluar qué tan bien funciona un sistema.

2. Elastic (Scalable): Reacciona a la carga. Para poder interactuar continuamente con su entorno, las aplicaciones reactivas deben poder adaptarse a la carga a la que se enfrentan. Utilizando una mayor capacidad computacional cuando sea necesario y funcionando a través de varios nodos de cómputo dependiendo de la carga.

3. Message Driven (Event-Driven): Reacciona a un evento. Las aplicaciones basadas en la comunicación asíncrona reacciona ante eventos discretos como las solicitudes HTTP sin monopolizar los recursos computacionales mientras espera que ocurra un evento. Este nivel natural de concurrencia produce mejor latencia que las llamadas a métodos síncronos tradicionales. Otra consecuencia de la escritura de programas dirigidos por eventos es que los componentes se acoplan de forma flexible, lo que hace que el software sea mucho más fácil de mantener a largo plazo.

4. Resilient (Resistente): Reacciona a las fallas. Debido a que incluso los sistemas de software más sencillos son propensos a fallar (ya sea relacionado con el software o relacionado con el hardware), las aplicaciones reactivas deben ser resistentes a las fallas para satisfacer la demanda de disponibilidad continua. La capacidad de volver a levantarse en caso de que se encuentre con un problema es posiblemente aún más importante cuando se trata de sistemas escalables y distribuidos, debido a que aumenta la probabilidad de fallas en el hardware o la red.

Por esto, algunos autores afirman que la arquitectura del sistema IoT es considerada la base del paradigma de programación reactiva [11].

Modelo de Programación de Actor

El modelo Actor es una teoría matemática de la computación que trata a los "Actores" como primitivas universales de la computación digital concurrente [12]. El modelo ha sido utilizado tanto como un framework para una comprensión teórica de la concurrencia, y como la base teórica para varias implementaciones prácticas de sistemas concurrentes.

A diferencia de los modelos de computación anteriores, el Modelo de actor se inspiró en las leyes físicas. También estuvo influenciado por los lenguajes de programación Lisp, Simula-67 nd Smalltalk72, así como por fundamentos de redes de Petri, sistemas de capacidad y packet switching. El advenimiento de la concurrencia masiva a través de la computación en la nube y las arquitecturas de computadora multi núcleo ha estimulado el interés en el Modelo de actor.

Dicho modelo de programación fue definido por Carl Hewitt en 1973, y presenta ventajas considerables para el desarrollo de aplicaciones para IoT [13].

En IoT los dispositivos pueden dejar de tener conexión momentáneamente por diversas

razones, por ejemplo la intensidad de la señal, razones climáticas, descarga de baterías (baterías solares) y posiblemente tiempo posterior vuelvan a conectarse, debido a la recarga de la batería solar o al alcance de la señal, si es un dispositivo móvil. Esta conexión y desconexión produce fallos que pueden propagar otros fallos en el resto de la red. En el modelo de programación de actor, cada actor se comunica de forma asincrónica con otro lo cual facilita la comunicación cuando hay conexión - desconexión de los dispositivos. Por otro lado, los actores presentan independencia en la ejecución unos de otros, lo cual en caso de falla no la propagaría al resto de la red. Además son muy livianos, por ejemplo en algunos toolkits como Akka ocupan solamente 600 Bytes, lo cual lo hace muy adecuado para entornos donde los dispositivos tienen poca capacidad de procesamiento y memoria

El modelo de programación de actor permite la concurrencia, está basado en memoria distribuida y puede ser aplicado al edge. [14].

El modelo de programación de actor presenta mínimas diferencias con la programación reactiva.

Los objetos reactivos tienen las siguientes ventajas sobre los actores:

- Los mensajes a métodos no definidos son simplemente puestos en cola.
- El modelo de actor carece de mensajes síncronos.
- La entrega asíncrona de mensajes no preserva el orden

Sin embargo, la sincronización puede ser emulada por pares de mensajes asíncronos, y la mayoría de las implementaciones modernas presentan mejoras en estos los tres puntos.

Líneas de Investigación, Desarrollo e Innovación

El desarrollo de la investigación se centrarán en el análisis de los lenguajes de programación, frameworks y otras herramientas que incorporan el modelo de programación de actor y la programación reactiva, en arquitecturas de IoT que permitan desarrollar aplicaciones en el fog o en los dispositivos (mist), tendientes a lograr la continuidad del Edge al Cloud.

Este análisis implica un cuidadoso estudio de las características de cada uno de ellos, evaluando en cada caso la eficiencia para las arquitecturas IoT, la posibilidad de orquestar recursos y de alcanzar una fluida continuidad al Cloud desde las cosas.

Resultados y Objetivos

Resultados Obtenidos

Durante los últimos dieciocho años se trabajó en una variedad de sistemas distribuidos y paralelos. Ya sea en Computación de Altas Prestaciones, en particular sobre análisis de diversas arquitecturas paralelas y distribuidas, tales como: Cloud Computing (públicos, híbridos y privados), Cluster de commodity, arquitecturas distribuidas de bajo costo y arquitecturas paralelas, como así también con arquitecturas de menores prestaciones (computación móvil y Edge Computing). Dicha experiencia motivó la elaboración del proyecto de investigación del que forma parte la presente propuesta. El grupo ha realizado publicaciones en el área durante los últimos años: dieciocho trabajos de investigación en diferentes Congresos y Jornadas, se realizaron tres publicaciones en revistas científicas y se transfirieron los resultados mediante seis conferencias en eventos científicos y encuentros de divulgación.

Se han aprobado diecisiete tesinas de grado y un trabajo de especialización y se incorporaron dos becarios de investigación categoría alumno y se encuentran en desarrollo tesinas de grado y de maestría.

Objetivos

El objetivo del grupo de investigación es analizar las ventajas de la programación reactiva y del modelo de programación de actor, investigando distintas arquitecturas y evaluando caso donde la resiliencia y la comunicación asíncrona permitan que los sistemas estén exentos de errores.

Formación de Recursos Humanos

El equipo de trabajo está compuesto por los docentes-investigadores de la línea de investigación presentada que figuran en este trabajo, de las universidades de San Juan y Champagnat (Mendoza) y cuatro alumnos de la UNSJ.

Cabe destacar que el proyecto marco de la presente línea de investigación incluye investigadores de UNSL y UNLaR y además se estarían realizando dos tesis doctorales, dos maestrías y varias tesinas de grado.

Recientemente se ha aprobado una tesina de grado sobre Evaluación del Modelo de programación de Actor sobre el entorno de Internet de las Cosas y se espera realizar una tesis de maestría sobre Control Topológico para reducción de interferencia en redes IoT. Se espera iniciar otras tesinas más en el área motivo de la presente propuesta de investigación.

Además se espera aumentar el número de publicaciones y también se prevé la divulgación de varios temas investigados por medio de cursos de postgrado y actualización o actividades de divulgación y asesoramiento a empresas y organismos del estado.

Referencias

[1] C. Savaglio, G. Fortino, M. Zhou (2016), "Towards interoperable, cognitive and autonomic IoT systems: an agent-based approach, in: Internet of Things (WF-IoT)", IEEE 3rd World Forum on IEEE, 2016, pp. 58–63.

[2] Hewitt C., Bishop P., Steiger R. (1973). "A Universal Modular Actor Formalism for Artificial Intelligence". IJCAI.

[3] Biron J., Follett J. (2016), "Foundational Elements of an IoT Solution The Edge, The Cloud, and Application Development", O'Reilly Media, Inc.

[4] Garnet: "Gartner Says 6.4 Billion Connected Things Will Be in Use in 2016, Up 30 Percent From 2015". http://www.gartner.com/newsroom/id/3165317.

[5] Varghese B., Wang N., Nikolopoulos D., Buyya R., "Feasibility of Fog Computing", arXiv:1701.05451v1cs.DC

[6] Bonomi F. et al (2012), "Fog Computing and Its Role in the Internet of Things", Proc. 1st Edition MCC Workshop Mobile Cloud Computing (MCC12), 2012, pp.13–15.

[7] Bainomugisha E., Carreton A. L., van Cutsem T., Mostinckx, S., de Meuter W. (2013), "A Survey on Reactive Programming", Journal ACM Computing Surveys (CSUR), Vol 45 N. 4, Aug. 2013, https://doi.org/10.1145/2501654.2501666

[8] Shibanai K., Watanabe T. (2018), "Distributed Functional Reactive Programming on Actor-Based Runtime", AGERE '18, November 5, 2018, Boston, USA.

[9] Mytera F., Scholliersb C., De Meutera W. (2019),"Distributed Reactive Programming for Reactive Distributed Systems", The Art, Science, and Engineering of Programming, vol. 3, no. 3, article 5; 52 pages.

[10] https://www.reactivemanifesto.org/

[11] Romanov E., Troshina G.(2017), "The IoT-architecture on the principles of reactive programming", 2017 International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON).

[12] Hewitt C. (2015), "Actor Model of Computation: Scalable Robust Information Systems", Cornell University, https://arxiv.org/abs/1008.1459

[13] Patil S., Nair S., Sruthi , Narkhede, B. E. , Pendam D. V. (2016). A 20/20 vision of Internet of things. IOSR Journal of Business and Management. 18. 76-80. 10.9790/487X-1808027680.

[14] Hewitt C., Bishop P. and Steiger R. (1973), "A Universal Modular ACTOR Formalism for Artificial Intelligence". Proceedings of the 3rd international joint conference on Artificial