Universidad Nacional del Litoral

Proyecto Final de Carrera

Diseño de un sistema de detección de anomalías en redes de computadoras.

Informe de avance 3

Pineda Leandro
Córdoba
17 de enero de 2018

1. Introducción

2. Comunicación entre componentes

La arquitectura de un sistema es la descripción de su estructura en términos de componentes específicos y sus se interrelaciones. De esta manera, podemos asegurarnos que dicha estructura satisface las demandas actuales y puede ser adaptada para satisfacer demandas futuras. Cuando se diseñan sistemas distribuidos, es decir, aquellos compuestos por varios componentes que no comporten el mismo espacio de memoria, es necesario considerar lo siguiente:

- ¿Cuales son las entidades que se comunican entre si?
- ¿Cómo van a comunicarse, o para ser mas específicos, que paradigma de comunicación va a usarse?

Estas preguntas son centrales para entender los sistemas distribuidos; qué se está comunicando y cómo esas entidades se comunican entre si, definen una gran cantidad de variables a ser consideradas por quienes construyen estos sistemas.

Las entidades que se comunican en un sistema distribuido son típicamente procesos, lo que nos permite entender a los sistemas distribuidos como procesos que se relacionan mediante los paradigmas de comunicación *entre procesos* apropiados. Podemos nombrar tres paradigmas de comunicación:

- Comunicación *entre procesos*.
- Invocación remota.
- Comunicación indirecta.

La comunicación *entre procesos* refiere al soporte de bajo nivel para comunicarse entre procesos en sistemas distribuidos, incluyendo primitivas para manejo de mensajes, acceso directo a las API provistas por protocolos de Internet (esto es, usando Sockets), y soporte para comunicación *multicast*.

La invocación remota representa el paradigma de comunicación más común en sistemas distribuidos. El intercambio de mensajes entre las entidades comunicantes es bidireccional, de forma que operaciones remotas, procedimientos y métodos, pueden ser invocados como se define a continuación:

- Protocolos *request-reply*: estos protocolos involucran el intercambio de mensajes desde el cliente al servidor y luego del servidor al cliente, donde el primer mensaje representa la operación que será ejecutada en el servidor (con los parámetros necesarios) y el segundo contiene cualquier resultado de dicha operación. Este paradigma es mas bien primitivo, y es utilizado generalmente en sistemas embebidos donde la *performance* es de suma importancia.
- *Remote procedure calls* (RPC) o llamadas a procedimientos remotos: este concepto, atribuido inicialmente a Birrel and Nelson [1984], representó un gran cambio en los paradigmas de computación distribuida. En RPC, los procedimientos de los procesos en computadoras remotas pueden ser invocados como si fuesen invocados desde el espacio local de memoria. De esta manera, el sistema abstrae aspectos acerca de la distribución, como la codificación de los parámetros y resultados, el paso de mensajes, entre otros. Este esquema soporta comunicación cliente-servidor pero depende de servidores que ofrezcan un conjunto de operaciones a través de una interfaz de servicio para que los clientes puedan llamar esas operaciones como si estuviesen disponibles localmente.
- *Remote method invocation* (RMI) o invocación remota de métodos: RMI es similar a RPC pero utiliza objetos distribuidos. Bajo este paradigma, un objeto cliente puede invocar métodos de un objeto remoto. De la misma forma que con RPC, ciertos detalles de como se implementa la comunicación quedan ocultos al usuario. Algunas implementaciones de RMI pueden incluir, además, soporte para darle a los objetos identidad y la habilidad de usar esos identificadores de objetos en llamadas remotas.

Las técnicas discutidas hasta aquí tienen una cosa en común: la comunicación representa una relación en ambos sentidos entre el emisor y el receptor, con los emisores enviando explícitamente mensajes/invocaciones a los receptores asociados. Los receptores, generalmente deben saber sobre la identidad de los emisores y, en la mayoría de los casos, ambas partes deben existir al mismo tiempo para que la comunicación sea exitosa. En contraste, surgieron numerosas técnicas donde la comunicación es indirecta, a través de una tercera entidad, permitiendo un gran grado de desacople entre emisores y receptores. En particular:

- Los emisores no necesitan saber a quien le están enviando datos.
- Emisores y receptores no necesitan existir al mismo tiempo.

Las técnicas más usadas para comunicación indirecta incluyen:

- Sistemas *publish-suscribe*: en estos sistemas, un gran número de productores (o *publishers*) distribuyen elementos de información de interés (eventos) a un número similar de consumidores (o *suscribers*). Usar cualquier de los paradigmas discutidos anteriormente hubiera sido complejo e ineficiente y por lo tanto los sistemas *publish-suscribe* (a veces llamados sistemas basados en eventos) surgieron para cubrir esta demanda[1]. Todos los sistemas *publish-suscribe* comparten la característica crucial de proveer un servicio intermedio que asegura que la información generada por los productores es enrutada eficientemente a los consumidores que deseen dicha información.
- Colas de mensajes: de la misma forma que los sistemas *publish-suscribe* proveen un estilo de comunicación uno a muchos, las colas de mensajes ofrecen un servicio punto a punto mediante el cual los procesos de los productores pueden enviar mensajes a una cola especifica y los procesos consumidores pueden recibir los mensajes o ser notificados de la llegada de nuevos mensajes a la cola. Las colas, entonces, ofrecen una indirección entre los procesos productores y consumidores.

Referencias

[1] G. Mühl, L. Fiege y P. Pietzuch, *Distributed Event-Based Systems*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006, ISBN: 3540326510.

[2] G. Cormode y M. Hadjieleftheriou, «Finding Frequent Items in Data Streams», *Proc. VLDB Endow.*, vol. 1, n.º 2, págs. 1530-1541, ago. de 2008, ISSN: 2150-8097. DOI: 10.14778/1454159.1454225. dirección: http://dx.doi.org/10.14778/1454159.1454225.

[3] S. Muthukrishnan, «Data Streams: Algorithms and Applications», *Found. Trends Theor. Comput. Sci.*, vol. 1, n.º 2, págs. 117-236, ago. de 2005, ISSN: 1551-305X. DOI: 10.1561/0400000002. dirección: http://dx.doi.org/10.1561/0400000002.

[4] D. Terry, D. Goldberg, D. Nichols y B. Oki, «Continuous Queries over Append-only Databases», *SIGMOD Rec.*, vol. 21, n.º 2, págs. 321-330, jun. de 1992, ISSN: 0163-5808. DOI: 10.1145/141484.130333. dirección: http://doi.acm.org/10.1145/141484.130333.

[5] D. Tong y V. Prasanna, «High Throughput Sketch Based Online Heavy Hitter Detection on FPGA», *SIGARCH Comput. Archit. News*, vol. 43, n.º 4, págs. 70-75, abr. de 2016, ISSN: 0163-5964. DOI: 10.1145/2927964.2927977. dirección: http://doi.acm.org/10.1145/2927964.2927977.

[6] G. Cormode y M. Hadjieleftheriou, «Methods for Finding Frequent Items in Data Streams», *The VLDB Journal*, vol. 19, n.º 1, págs. 3-20, feb. de 2010, ISSN: 1066-8888. DOI: 10.1007/s00778-009-0172-z. dirección: http://dx.doi.org/10.1007/s00778-009-0172-z.

[7] L. J. Guibas, «Problems», *Journal of Algorithms*, vol. 2, n.º 2, págs. 208 -210, 1981, ISSN: 0196-6774. DOI: http://dx.doi.org/10.1016/0196-6774(81)90022-5. dirección: http://www.sciencedirect.com/science/article/pii/0196677481900225.

[8] R. M. Karp, S. Shenker y C. H. Papadimitriou, «A Simple Algorithm for Finding Frequent Elements in Streams and Bags», *ACM Trans. Database Syst.*, vol. 28, n.º 1, págs. 51-55, mar. de 2003, ISSN: 0362-5915. DOI: 10.1145/762471.762473. dirección: http://doi.acm.org/10.1145/762471.762473.

[9] E. Kranakis, P. Morin e Y. Tang, «Bounds for Frequency Estimation of Packet Streams», en *In SIROCCO*, 2003, págs. 33-42.

[10] E. M. Hutchins, M. J. Cloppert y R. M. Amin, «Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains», *Leading Issues in Information Warfare & Security Research*, vol. 1, pág. 80, 2011.

[11] A. Metwally, D. Agrawal y A. El Abbadi, «Efficient Computation of Frequent and Top-k Elements in Data Streams», en *Proceedings of the 10th International Conference on Database Theory*, ép. ICDT'05, Edinburgh, UK: Springer-Verlag, 2005, págs. 398-412, ISBN: 3-540-24288-0, 978-3-540-24288-8. DOI: 10.1007/978-3-540-30570-5_27. dirección: http://dx.doi.org/10.1007/978-3-540-30570-5_27.

[12] G. Cormode y S. Muthukrishnan, «An Improved Data Stream Summary: The Count-min Sketch and Its Applications», *J. Algorithms*, vol. 55, n.º 1, págs. 58-75, abr. de 2005, ISSN: 0196-6774. DOI: 10.1016/j.jalgor.2003.12.001. dirección: http://dx.doi.org/10.1016/j.jalgor.2003.12.001.

[13] F. Putze, P. Sanders y J. Singler, «Cache-, Hash-, and Space-efficient Bloom Filters», *J. Exp. Algorithmics*, vol. 14, 4:4.4-4:4.18, ene. de 2010, ISSN: 1084-6654. DOI: 10.1145/1498698.1594230. dirección: http://doi.acm.org/10.1145/1498698.1594230.

[14] B. H. Bloom, «Space/Time Trade-offs in Hash Coding with Allowable Errors», *Commun. ACM*, vol. 13, n.º 7, págs. 422-426, jul. de 1970, ISSN: 0001-0782. DOI: 10.1145/362686.362692. dirección: http://doi.acm.org/10.1145/362686.362692.

[15] D. Crockford, *The application/json Media Type for JavaScript Object Notation (JSON)*, RFC 4627, jul. de 2006. DOI: 10.17487/RFC4627. dirección: https://rfc-editor.org/rfc/rfc4627.txt.