Garantizando la consistencia de JavaScript en un contexto de memoria compartida

Matías Teragni, Gonzalo Zabala, Ricardo Morán, Sebastián Blanco.
Centro de Altos Estudios en Tecnología Informática
Facultad de Tecnología Informática
Universidad Abierta Interamericana
Av. Montes de Oca 745, Ciudad Autónoma de Buenos Aires, República Argentina
(+54 11) 4301-5323; 4301-5240; 4301-5248
{ Matias.Teragni, Gonzalo.Zabala, Ricardo.Moran, Sebastian.Blanco}@uai.edu.ar

Resumen

El propósito de este proyecto es diseñar e implementar un conjunto de librerías, técnicas y funcionalidades que permitan, dado un sistema distribuido de memoria compartida, generar código javascript que ejecute haciendo uso de los diversos nodos presentes prestando garantías sobre la consistencia del programa ejecutado, y brindando las herramientas necesarias para poder garantizar la semántica del código escrito por el programador, y su predictibilidad a la hora de ejecutar.

Palabras clave: Programación Distribuida, Consistencia, Javascript

Contexto

El presente proyecto será radicado en el Centro de Altos Estudios en Tecnología Informática, dependiente de la Facultad de Tecnología Informática de la Universidad Abierta Interamericana. El mismo se encuentra inserto en la línea de investigación “Algoritmos y software”. El financiamiento está compartido entre el CONICET y la misma Universidad por partes iguales.

Introducción

Se entiende a un sistema distribuido como un conjunto de computadoras independientes, cada una con su propia memoria y capacidad de procesamiento, que se encuentran interconectadas por medio de una red, y sobre las cuales opera un conjunto de software que colabora para lograr un comportamiento complejo como un todo.

Los sistemas distribuidos pueden proveer, dada su naturaleza, un conjunto de beneficios, desde resistencia a fallos mediante replicación, hasta la capacidad de ejecutar procesos sumamente complejos en computadoras económicas mediante la partición del problema en partes más simples.

A la hora de construir sistemas distribuidos existen múltiples paradigmas que se pueden aplicar en función de las necesidades particulares del contexto. Estos varían en nivel de abstracción, y las interfaces y lenguajes que le permiten utilizar a los desarrolladores para construirlos.

Particularmente uno de los mecanismos

que tuvo un resurgimiento últimamente, gracias a los avances tecnológicos que compensan algunas de sus falencias, fue la idea de una memoria compartida entre los nodos.

La idea no es novedosa, pero las limitaciones en velocidades de red y capacidad de memoria de los distintos nodos volvieron a este modelo poco práctico en comparación a algunas de las alternativas.

Este modelo es muy utilizado en los servicios provistos en la "nube" ya que para garantizar tiempos de acceso, fault tolerance, y service level assurance, los servidores normalmente se replican en varios datacenters a lo largo del mundo.

Esta replicación trae consigo claros beneficios pero no está libre de riesgo, ya que la consistencia de la información presente en dichos nodos puede generar comportamientos indebidos en las aplicaciones de los clientes.

Estos problemas son de interés actual, y muchos de los esfuerzos se centran en tratarlos exclusivamente a nivel de datos, sin considerar las particularidades que le pueden generar estas inconsistencias a un programa que ejecute en función a esta información que puede ser inconsistente.

El presente trabajo tiene como objetivo desarrollar un conjunto de librerías que permitan mantener las ventajas de las arquitecturas distribuidas pudiendo escribir código cuya ejecución sea predecible y garantizable.

Líneas de Investigación, Desarrollo e Innovación

Para poder abordar este problema existen dos conflictos en particular que deben tratarse. Por un lado está el problema de la replicación de los datos entre los datos (memoria) entre los distintos nodos, y por otro la construcción de herramientas que permitan garantizar la semántica del código escrito por un programador.

En cuanto a la replicación respecta, es un problema que ya fue abordado tanto por la comunidad como por los autores de este artículo con diversos grados de éxito. Con respecto a garantizar la semántica de un código ejecutado de manera distribuida existen algunas restricciones que se deben poder garantizar sobre este proceso.

- Consistencia Eventual: Se refiere a la capacidad que tiene un sistema distribuido de que todos los nodos que lo integran tengan una convergencia sobre el estado de la información compartida. El nombre hace referencia a que dicha convergencia puede no ser alcanzada de forma instantánea, pero dado tiempo infinito es demostrable que la misma existe.

- Consistencia Causal: Se refiere a la capacidad que tiene un sistema distribuido de preservar órdenes parciales de lecturas/modificaciones ante dependencia funcional de las acciones realizadas. Esto es fundamental para garantizar la semántica de cualquier programa que deba ejecutarse.

Dado un sistema de replicación que pueda garantizar ambas características consideramos que puede construirse sobre

el mismo un conjunto mínimo de operaciones que permitan a un programador tener un control fino de su modelo de ejecución, permitiéndole generar secciones críticas, mutex, y semáforos cuando sea necesario, y aprovechar las características concurrentes inherente a un sistema distribuido cuando no lo sea. Preservando así la simplicidad del código, obteniendo algunos de los beneficios de operar en un espacio distribuido pero sin sacrificar la semántica del código, evitando la generación de race conditions, ejecuciones secuencialmente inválidas, ni comportamientos impredecibles en un código que no lo amerite.

Resultados y objetivos

El objetivo de este proyecto es diseñar e implementar un esquema control que permita a un programador definir inequívocamente las necesidades de sincronismo que tiene el código que genera.

Para los prototipos se propone utilizar javascript dada su gran popularidad, y su naturaleza multiplataforma. Un requisito adicional es no imponer en el motor de ejecución del lenguaje ninguna modificación, permitiendo que estos beneficios sean aplicables as-is en cualquier contexto donde código javascript pueda ejecutar.

Una vez demostrados formalmente estos mecanismos, y planteados prototipos de su implementación el siguiente paso será evaluar extender estas funcionalidades a otros lenguajes de programación que implementan más restricciones, como podría ser Java, C# o C.

Formación de Recursos Humanos

El equipo de trabajo está conformado por un investigador adjunto del Centro de Altos Estudios en Tecnología Informática (CAETI) quien ejerce el rol de director del proyecto, dos doctorandos, y un ayudante alumno de la Facultad de Tecnología Informática de la Universidad Abierta Interamericana.

Referencias

- LeBlanc,T.J; Markatos,E.P. (1992). Shared memory vs. message passing in shared-memory multiprocessors. Proceedings of the Fourth IEEE Symposium on Parallel and Distributed Processing
- Coulouris, George; Jean Dollimore; Tim Kindberg; Gordon Blair (2011). Distributed Systems: Concepts and Design (5th Edition). Boston: Addison-Wesley.
- Ngo,T.A; Snyder,L. (1992). On the influence of programming models on shared memory computer performance. Scalable High Performance Computing Conference, 1992. SHPCC-92, Proceedings.
- LeLann, G. (1977). "Distributed systems - toward a formal approach,". Information Processing. 77: 155·160. – via Elsevier.
- Andrews, Gregory R. (2000), Foundations of Multithreaded, Parallel, and Distributed Programming, Addison–Wesley, ISBN 0-201-35752-6.
- Lamport, L. (1977). Proving the Correctness of Multiprocess Programs. IEEE Transactions on Software Engineering (Volume: SE-3, Issue: 2, March 1977)