Un Problema de Conjuntos en Computación Distribuida

Esteban Meneses y Francisco J. Torres-Rojas

Centro de Investigaciones en Computación

Resumen

En Matemática existen muchos problemas que involucran conjuntos. Generalmente, estos problemas están relacionados con un grupo de elementos que deben cumplir una cierta propiedad. Por ejemplo, los *conjuntos pitagóricos* son aquellos de la forma $\{x,y,z\}$, con $x < y < z$ tales que conforman una terna pitagórica: $x^2 + y^2 = z^2$. Sin embargo, el algoritmo para determinar si un conjunto de cardinalidad 3 es pitagórico o no, es muy eficiente. En Computación Distribuida existen también problemas de conjuntos. Uno de ellos es el *problema de los conjuntos imposibles de relojes vectoriales* ([9]), que no se ha determinado si posee un algoritmo eficiente que lo resuelva.

1. Introducción

La historia de las matemáticas recuerda con respeto a muchos hombres, uno de ellos es Georg Ferdinand Ludwig Philipp Cantor, creador de la Teoría de Conjuntos. Alrededor de 1873, Cantor descubrió esta apasionante rama de estudio, formalizando de paso la noción de *infinito*, que hasta el momento había sido solamente una forma de hablar.

La Teoría de Conjuntos sirve como base para muchas otras ramas matemáticas, como Probabilidades, Álgebra, Análisis y Ecuaciones Diferenciales. Mas aún, junto con el Cálculo de Predicados, constituye el fundamento de las Matemáticas. En Ciencias de la Computación, la Teoría de Conjuntos también juega un papel preponderante en la Matemática Discreta, Teoría de Computabilidad, Algoritmos, Programación y Especificación Formal de Software.

La Teoría de Conjuntos es la responsable de teoremas elegantes y poderosos, como el *Axioma de Elección*. Empero, en Computación Distribuida existen también problemas interesantes de conjuntos. Tal es el caso de los *Conjuntos Imposibles de Relojes Vectoriales* ([9]). Este artículo ofrecerá un reto con respecto a este problema.

El objeto de estudio de la Teoría de Conjuntos son los *conjuntos*. Estas entidades matemáticas son objetos que ayudan a representar conceptos. Por ejemplo, si las longitudes de los lados de un triángulo rectángulo son enteras, entonces, esas longitudes se convierten en una *terna pitagórica*, una especie particular de conjunto.

Una vasta cantidad de problemas en Teoría de Conjuntos se fundamentan en determinar si el conjunto cumple o no con una *propiedad*. Tal es el problema de los *conjuntos balanceados*. Un conjunto de números enteros se dice balanceado si la suma de los elementos es igual a su producto. De esta forma, $\{1,2,3\}$ es balanceado, pero $\{2,3,4\}$ no lo es.

En Computación Distribuida, cualquier ejecución produce una historia distribuida de eventos. Si se coleccionan las etiquetas vectoriales de esos eventos, se obtiene un conjunto de vectores de enteros, al que se le denominará conjunto *posible*. Sin embargo, no cualquier conjunto de vectores de enteros es el producto de los eventos de una historia distribuida. A estos conjuntos se les llama *imposibles*. Descubrir si un conjunto de vectores de enteros es posible o no, es un problema interesante. Existe un algoritmo que resuelve este problema, con el bemol de ser ineficiente. ¿Existirá entonces algún algoritmo eficiente que lo resuelva?

En la sección 2 se presenta un modelo de computación distribuida. Los relojes vectoriales, como mecanismo de temporización, se analizan en la sección 3 y en la sección 4 se explica el problema de los conjuntos imposibles. Un mecanismo para clasificar la eficiencia de los algoritmos se ofrece en la sección 5, para dejar las conclusiones al final.

2. Computación Distribuida

La Computación Distribuida es la rama de las Ciencias Computacionales que estudia la ejecución de un algoritmo por medio de varios sitios, dispersos geográficamente, que comparten recursos.

El modelo que utilizaremos en este artículo ([7]) supondrá varios elementos:

- *Sitio*: localidad donde se está ejecutando un programa.
- *Evento*: mínima unidad en la ejecución de un programa.
- *Mensaje*: envío de información de un sitio a otro.

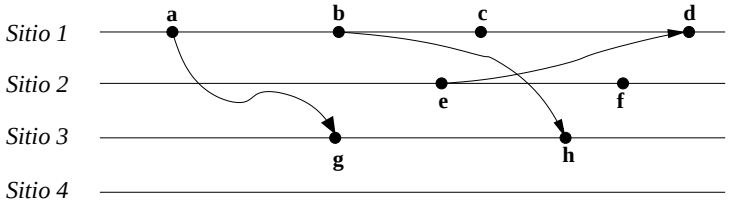

Figura 1: Historia distribuida

En la figura 1 se muestra la representación gráfica de una historia distribuida. En la misma se puede apreciar un sistema distribuido con 4 sitios, donde cada línea representa la ejecución en el tiempo de un programa compuestos por varios (inclusive ningún) eventos. Los círculos negros son los eventos que ocurren en cada uno de los sitios. Así pues, el evento *a* es el primero del sitio 1. Finalmente, las flechas indican la propagación de un mensaje de un evento a otro. Entonces, existe un mensaje que parte del evento *a* y llega al evento *g*.

La *Historia Local* del sitio i es una secuencia de eventos $\mathcal{H}_i = e_1 e_2 \ldots$ que son ejecutados en ese sitio. Hay un orden total en los eventos locales en cada sitio. La

Historia Global $\mathcal{H}$ del sistema distribuido (o simplemente, historia distribuida) es el conjunto de todos los eventos que ocurren en todos los sitios del sistema.

Existen solo 3 clases de eventos: **ENVIO**, **RECEPCION** e **INTERNO**. Dado esto, en la figura 1, el evento *a* es un ENVIO, mientras que *g* es un RECEPCION y aquellos que no sean ninguno de los anteriores se dicen INTERNO, como el evento *c*.

Una de las relaciones más importantes entre los eventos es la *causalidad*. Por causalidad se entiende que un evento *a* previo a un evento *g* podría ser la causa del último. Entonces, la causalidad que se trata de determinar es *potencial*. En la figura 1, el evento *a* es una causa potencial del evento *g*, porque precisamente *a* es el envío de un mensaje que es recibido posteriormente por *g*. Además, *a* es una causa potencial de *b*, dado que ocurre antes en el sitio 1. El tema de causalidad supone un ordenamiento de los eventos en una historia distribuida.

Para poder detectar este ordenamiento, se podrían utilizar relojes físicos y comparar el tiempo en que fue ejecutado cada evento para así determinar cuál es el orden de causalidad entre ellos. Empero, esta estrategia adolece de muchas complicaciones, debido a la sincronización de tales relojes.

Entonces, en el seminal artículo ([6]), Leslie Lamport describe los relojes lógicos como un mapeo entre eventos en una historia distribuida y el conjunto de los números enteros, de manera que se captura el orden causal.

Lamport ([6]) definió la relación de causalidad "$\rightarrow$" (*happens before*), como la relación más pequeña, tal que:

- si e_{ij} y $e_{ik} \in \mathcal{H}_i$ y $j < k$ entonces $e_{ij} \rightarrow e_{ik}$.
- si e_{im} es el evento **ENVIO**($\mathcal{M}$), e_{jn} es el evento **RECEPCION**($\mathcal{M}$) y $\mathcal{M}$ es el mismo mensaje en ambos casos, con i, j, m y n arbitrarios, se tiene que $e_{im} \rightarrow e_{jn}$.
- si a, b y $c \in \mathcal{H}$, si $a \rightarrow b$ y $b \rightarrow c$ entonces $a \rightarrow c$.

Si entre dos eventos *a* y *b*, no sucede ni $a \rightarrow b$, ni $b \rightarrow a$, entonces se dice que *a* y *b* son *concurrentes*. Denotaremos esta situación como $a \parallel b$. Como la relación de causalidad $\rightarrow$ es irreflexiva y transitiva, entonces define un orden parcial estricto $\langle H, \rightarrow \rangle$ sobre los eventos del sistema. Como orden parcial, tiene su correspondiente diagrama de Hasse ([3]).

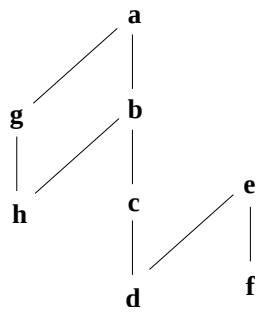

Figura 2: Diagrama de Hasse

La figura 2 muestra el diagrama de Hasse correspondiente a la figura 1. La relación utilizada corresponde a "$\rightarrow$" (*happens before*), definida por Lamport([6]).

En la figura 2 se aprecian las relaciones de causalidad que aparecen entre los eventos de la historia distribuida. Para ejemplificar, el evento *a* es causalmente antes que *h*, lo cual es cierto, dado que *a* es el **ENVIO** de un mensaje que luego recibe *g* y además *g* ocurre localmente antes que *h*. Sin embargo, no se puede determinar la relación de causalidad que existe entre *b* y *f*, lo que los hace eventos concurrentes.

3. Relojes Vectoriales

Los relojes vectoriales son un tipo de reloj lógico propuesto independientemente por Colin Fidge ([4]) y Friedemann Mattern ([8]). Interesantes recorridos sobre temas y aplicaciones de los mismos se puede encontrar en [1, 10, 11, 12].

Esta técnica consiste en un mapeo entre eventos en una historia distribuida y vectores de enteros. Cada sitio i mantiene un vector V_i con N entradas, donde N es la cantidad de sitios en el sistema distribuido. Este vector [1] se inicializa en ceros. La entrada $V_i[i]$ del sitio i mantiene el reloj local, mientras que la entrada $V_i[j]$ mantiene el conocimiento actual que el sitio i tiene de la actividad en el sitio j. Estos vectores se incrementan en el reloj local cada vez que ocurren eventos en el sitio. También, todo mensaje debe incluir el reloj vectorial que tenía el sitio emisor cuando el mensaje fue enviado, y al recibir mensajes, se debe actualizar el vector local, tomando en cuenta la etiqueta de tiempo incluida en cada mensaje.

El sitio i actualiza su vector V_i de acuerdo con las siguientes reglas:

V0: $1 \leq j \leq N$: $V_i[j] = 0$

V1: Cada vez que ocurre un evento local: $V_i[i] = V_i[i] + \Delta$, donde Δ es un valor entero positivo.

V2: Al recibir un mensaje con etiqueta de tiempo T: $1 \leq j \leq N$: $V_i[j] = max(V_i[j], T[j])$ $V_i[i] = V_i[i] + \Delta$.

Dado un evento $a \in \mathcal{H}$ se dice que la etiqueta de tiempo o el reloj vectorial asociado a a es el reloj vectorial que tenía el sitio particular cuando a fue ejecutado. Se denotará a este vector como $V(a)$.

Dados dos relojes vectoriales v y w, se definen las siguientes comparaciones:

- $v = w \Leftrightarrow 1 \leq j \leq N$: $v[j] = w[j]$
- $v \leq w \Leftrightarrow 1 \leq j \leq N$: $v[j] \leq w[j]$
- $v < w \Leftrightarrow v \leq w$ y $\exists j$ tal que $v[j] < w[j]$
- $v \parallel w \Leftrightarrow \exists k$ tal que $v[k] < w[k]$ y $\exists j$ tal que $v[j] > w[j]$

Mattern demostró en [8] que existe un isomorfismo entre los tiempos que se obtienen de un reloj vectorial cuando los eventos son ejecutados y la relación de causalidad entre los eventos de $\mathcal{H}$. Eso significa que $\forall a, b \in \mathcal{H}$:

- $a = b \Leftrightarrow V(a) = V(b)$
- $a \rightarrow b \Leftrightarrow V(a) < V(b)$
- $a \parallel b \Leftrightarrow V(a) \parallel V(b)$

Esta característica es conocida como la condición fuerte de reloj (*strong clock condition*, [8, 10, 12]).

[1] Indexado de 1 hasta N

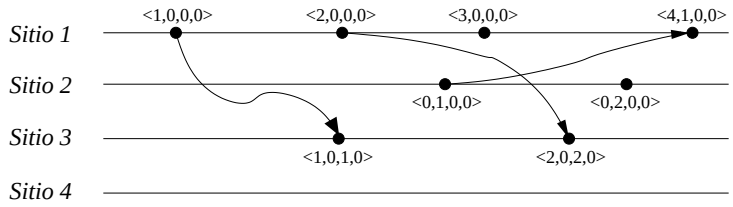

Figura 3: Etiquetas vectoriales de la figura 1

En la figura 3 se pueden observar las etiquetas vectoriales para los eventos de la historia distribuida de la figura 1. Estas etiquetas se obtuvieron al aplicar las reglas de generación de los relojes vectoriales (V0,V1 y V2), utilizando $\Delta = 1$.

Por otra parte, si se recolectan las etiquetas vectoriales de esta historia distribuida se obtiene un conjunto de vectores: $\{< 1,0,0,0 >, < 2,0,0,0 >, < 3,0,0,0 >, < 4,1,0,0 >, < 0,1,0,0 >, < 0,2,0,0 >, < 1,0,1,0 >, < 2,0,2,0 >\}$. A este conjunto se le denominará *conjunto posible*, dado que existe una historia distribuida que puede generar todas las etiquetas de ese conjunto (aquella de la figura 1). También son conjuntos posibles: $\{< 1,0,0,0 >, < 0,1,0,0 >, < 1,0,1,0 >\}$ y $\{< 1,0,0,0 >\}$.

Contrariamente, un *conjunto imposible* es aquel conjunto de etiquetas vectoriales para el cual es imposible construir una historia distribuida que contenga todos esos vectores. Hay muchos de ellos y obtenerlos es sencillo ([9, 11]).

4. El Problema de los Conjuntos Imposibles

El problema de los Conjuntos Imposibles establece que:

Dado un conjunto con n etiquetas vectoriales de k entradas cada una, decidir si es posible o no

De esta forma, dado un conjunto de vectores se quiere determinar si este conjunto cumple la propiedad de ser el reflejo de una historia distribuida. Un punto importante de mencionar es que la historia distribuida que justifica un conjunto puede tener más etiquetas que las que están en el conjunto. El asunto es que todos los vectores que están en el conjunto sean etiquetas de algún evento en la historia distribuida.

Este problema claramente tiene solución, dado que existe una cantidad finita de historias distribuidas que podrían servir de *testigo* para declarar a un conjunto como posible.

Los conjuntos posibles son aquellos resultantes de aplicar las reglas V0, V1 y V2 a una historia distribuida particular. Los imposibles aparecen porque al aplicar las reglas V0, V1 y V2 a cualquier historia distribuida no se pueden generar todos los elementos del conjunto.

Para ilustrar, considérense los siguientes dos conjuntos:

$$\mathcal{A} = \{< 2,3,1 >, < 3,0,0 >\}$$
$$\mathcal{B} = \{< 1,1,1 >\}$$

En el primer caso, el conjunto $\mathcal{A}$ contiene dos elementos (o sea, $n = 2$) y $k = 3$ (el número de sitios). Para determinar si $\mathcal{A}$ es posible o no, basta con aplicar las

reglas V0, V1 y V2 a todas las combinaciones posibles de mensajes entre eventos. Ahora, nótese que si se maximiza, entrada por entrada, los vectores de $\mathcal{A}$ se obtiene el número máximo de eventos en cada sitio ([11]). En este caso, $< 3, 3, 1 >$ es ese máximo y entonces la historia distribuida tendrá 3 eventos en el sitio 1, 3 en el sitio 2 y 1 en el sitio 3.

Entonces, cada uno de esos 7 eventos debe tener alguno de los tipos: **INTERNO**, **ENVIO** o **RECEPCION**. Analizando las posibles combinaciones que se pueden genererar al hacer que cada uno de estos eventos tenga uno de los tipos anteriores y aplicando las reglas V0, V1 y V2 se puede determinar si el conjunto es posible o no.

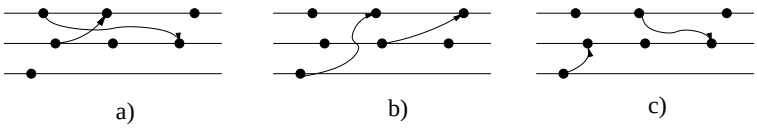

Figura 4: Varios candidatos para la historia testigo

En la figura 4 se presentan varias historias que podrían ser la historia testigo que se anda buscando. Particularmente, al aplicar las reglas V0, V1 y V2 se descubre que las historias en a) y b) no son testigos del conjunto $\mathcal{A}$, mientras que la historia c) sí lo es. Como se tiene una historia testigo, entonces el conjunto $\mathcal{A}$ es posible.

En el otro extremo, se puede aplicar esta misma estrategia para el conjunto $\mathcal{B}$, pero rápidamente se topa con la limitante de que es imposible lograr el objetivo de construir una historia testigo para este conjunto ([9]).

Esta estrategia funciona, pero a un costo muy elevado: probar todas las posibles alternativas de mensajes entre los eventos de una historia. Este algoritmo exhaustivo agota posibilidad por posibilidad hasta dar con una historia testigo o bien, darse por vencido.

Sería muy provechoso encontrar un algoritmo más eficiente, que no tuviera que lidiar con todas las alternativas.

5. Las clases P y NP

Los problemas se resuelven por medio de algoritmos, algunos de los cuales pueden resultar más costosos que otros. En Teoría de la Complejidad, se trata de analizar el costo de un algoritmo. Típicamente el costo es el tiempo que tarda el algoritmo en resolver el problema.

El algoritmo presentado en la sección 4 resuelve el problema de los conjuntos imposibles, pero de una manera muy ineficiente. Incluso, si el número de elementos del conjunto a analizar crece linealmente, entonces el tiempo de respuesta de ese algoritmo también crece, pero exponencialmente.

Existen muchas familias de problemas, pero dos clases muy populares son: P y NP-completo ([2, 5]). En la clase P están todos los problemas que se pueden resolver por un algoritmo en tiempo polinomial, con respecto a las entradas. Esto quiere decir que existe un polinomio $f(n)$ que acota el tiempo que tarda el algoritmo en resolver un problema de tamaño n. Por su parte, los problemas NP-completo son aquellos donde no es posible encontrar un algoritmo que resuelva el problema en un tiempo polinomial.

Dado que no se ha encontrado una solución eficiente al problema de los conjuntos imposibles, es importante intentar caracterizar este problema, ya sea como P (encontrando un algoritmo que resuelva el problema en tiempo polinomial) o bien clasificándolo como NP-completo (mostrando que es equivalente a otro problema NP-completo).

Los autores quieren heredarle este reto a los lectores.

6. Conclusiones

Los problemas de conjuntos han maravillado al mundo de los matemáticos. Sin embargo, también en Computación Distribuida existen problemas de conjuntos.

Uno de ellos es el Problema de los Conjuntos Imposibles. Este problema se relaciona con un sistema para detectar la causalidad en una historia distribuida. El sistema asigna vectores de enteros a eventos y utilizando ciertas propiedades se puede averiguar las relaciones causales entre los eventos respectivos.

Si se recolectan todas las etiquetas vectoriales se obtiene un conjunto posible. Cualquier sobconjunto de él también es posible. Aún así, cualquier conjunto de vectores no necesariamente corresponde a alguna historia distribuida.

Los conjuntos imposibles son aquellos para los cuales es imposible hallar una historia distribuida que los contenga. El problema de los conjuntos imposibles consta en determinar si un conjunto de vectores es posible o no. Resolver este problema es posible, pero por medio de un algoritmo que revisa exhaustivamente todas las historias candidatas.

El problema ingenieril es encontrar un algoritmo eficiente que resuelva el problema, o bien, demostrar que tal algoritmo no existe.

Referencias

[1] Baldoni, Roberto y Michel Raynal. *Fundamentals of Distributed Systems: A practical tour of Vector Clocks Systems*. IEEE Distributed Systems Online, Febrero, 2002.

[2] Cormen, Thomas et al. *Introduction to Algorithms*. 2nda edición. The MIT Press: Estados Unidos, 2001.

[3] Davey, B.A. y H.A. Prestley. *Introduction to Lattices and Order*. Cambridge University Press: Gran Bretaña, 1990.

[4] Fidge, Colin. *Logical Time in Distributed Computing Systems*. Computer. Vol 24(8): 28-33. Agosto, 1991.

[5] Garey, Michael y David Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company: Estados Unidos, 1979.

[6] Lamport, Leslie. *Time, Clocks, and the Ordering of Events in a Distributed System* . Communications of the ACM. Vol 21(7):558-565. Julio, 1978.

[7] Lynch, Nancy. *Distributed Algorithms*. Morgan-Kaufmann Publishers, 1996.

[8] Mattern, Friedemann. *Virtual Time and Global States of Distributed Systems*. Proceedings of the International Workshop on Parallel and Distributed Algorithms, 215-226, 1989.

[9] Meneses, Esteban y Francisco J. Torres-Rojas. *Possible and Impossible Vector Clock Sets*. *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA '04*. Junio, 2004.

[10] Schwarz, R. y Friedemann Mattern. *Detecting Causal Relationships in Distributed Systems: In Search of the Holy Grail*. Distributed Computing, 1994.

[11] Torres-Rojas, Francisco J. y Esteban Meneses. *Algunas Propiedades Interesantes de los Relojes Vectoriales*. Jornadas Chilenas de Computación, 2003.

[12] Yang, Z. y T.A. Marsland. (editores) *Global States and Time in Distributed Systems*. IEEE Computer Society Press, Los Alamos California, 1994.