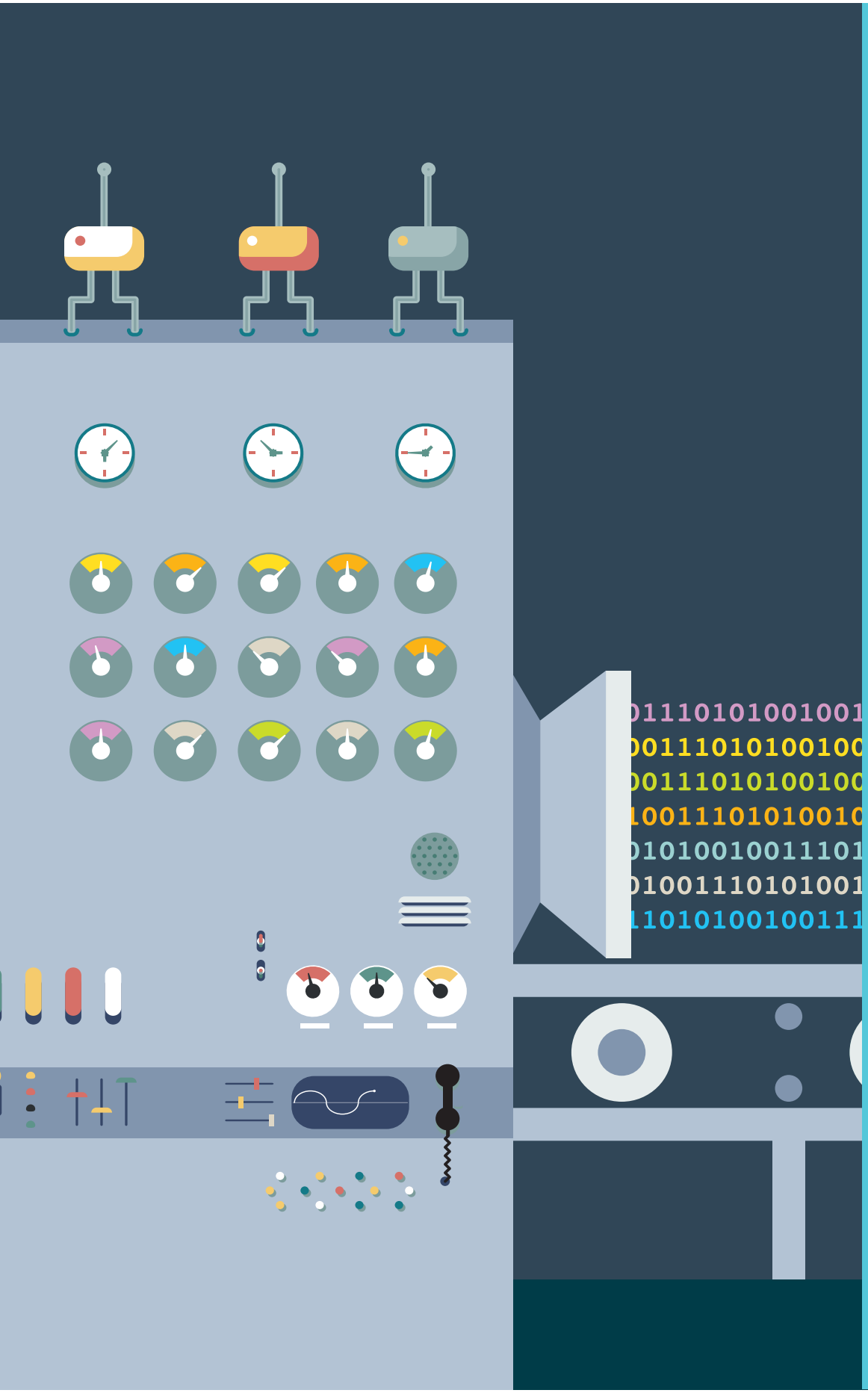
NAVEGANDO SOBRE CORRIENTES DE DATOS Y CÓMO NO AHOGARSE EN EL INTENTO



ALEJANDRO GREZ

Alumno de Doctorado en Ciencias de la Computación de la Pontificia Universidad Católica de Chile. Ingeniero Civil en Computación y Magíster en Ciencias de la Ingeniería, Pontificia Universidad Católica de Chile.

Líneas de investigación: manejo de datos streaming, complex event processing, teoría de autómatas, lógica computacional.

ajgrez@uc.cl

CRISTIÁN RIVEROS

Profesor Asistente del Departamento de Ciencia de la Computación de la Pontificia Universidad Católica de Chile. Investigador del Instituto Milenio Fundamentos de los Datos. DPhil in Computer Science, Oxford University.

Líneas de investigación: manejo de datos, sistema de base de datos, datos streaming, extracción de información, fundamentos de la computación, modelos de computación.

cristian.riveros@uc.cl

MANEJO DE DATOS STREAMING

Redes de sensores, redes sociales, tráfico en Internet, mediciones meteorológicas, observaciones astronómicas, son algunos escenarios en donde se generan gigabytes de datos continuamente y a gran velocidad. Pero, ¿de cuánta velocidad estamos hablando exactamente? Actualmente, un auto promedio cuenta con aproximadamente 100 sensores que miden datos relevantes para su funcionamiento, como el estado del motor o el nivel de bencina. A través de estos sensores, un auto genera alrededor de 10 gigabytes de datos cada hora. Otro ejemplo más cotidiano es el caso de Twitter. Si juntamos todos los tweets generados por los usuarios de esta red social, a cada segundo se twittean alrededor de 6 mil tweets, lo que equivale a 20 gigabytes generados por cada hora. Estos escenarios, conocidos como streams de datos, son ejemplos del fenómeno del Big Data y, en particular, aluden a la dimensión de velocidad: el problema de ser capaces de procesar grandes cantidades de datos en tiempo real.

El gran desafío que plantean estos escenarios es ¿cómo procesar de manera eficiente estos datos para obtener el mayor valor posible de ellos? Para responder esta pregunta, primero debemos notar que en muchos casos no es factible ni siquiera almacenar todos estos datos para procesarlos más tarde. En el caso de un auto, en un día habrá generado más de 240 gigabytes de datos, superando por lejos sus capacidades de almacenamiento. Lo segundo que debemos notar es que, incluso si logramos almacenar esta cantidad de datos, no tendremos quizás el tiempo suficiente para procesarlos. Hay que pensar que solo leer 240 gigabytes, y sin hacer ningún cálculo, nos tardaría alrededor de dos minutos en un procesador convencional. Por último, los datos generados por Twitter o en un auto son de importancia en el segundo que se generan, desvalorizándose rápidamente a medida que avanza el tiempo. En otras palabras, queremos saber sobre un "trending topic" de nuestro interés en Twitter, o de una posible falla del auto apenas suceda.

¿Cómo, entonces, procesar de manera eficiente estos streams de datos? Los algoritmos streaming aparecen a principios de este siglo como una respuesta para obtener información valiosa de estos grandes flujos de información. Dado que no tenemos ni tiempo ni espacio para acordarnos del pasado, un algoritmo streaming busca mantener una representación muy pequeña de los datos que ha visto (el pasado) que contenga solo la información que es de interés. Cada vez que recibe un dato nuevo, el algoritmo streaming actualiza su representación compacta con el dato (sin aumentar su tamaño considerablemente) y tomando tiempo que es proporcional solo a este nuevo dato. En otras palabras, no importa cuántos datos ha leído en el pasado, el tiempo de actualización por cada dato nuevo será el mismo. Para ilustrar la idea de un algoritmo streaming, suponga que usted está recibiendo el siguiente stream, en donde cada evento es un número:

$$3, 4, 8, 10, 12, 34, 12, 54, \underset{\downarrow}{65}, 23, \ldots$$

La condición es que, en cualquier momento, usted debe ser capaz de entregar el promedio aritmético de todos los números que han llegado hasta ese momento. Una posibilidad es que usted almacene toda la secuencia que ha recibido, y cuando le pregunten por el promedio, sume todos los números anteriores y divida por la cantidad de números. Por ejemplo, si estamos en el momento indicado por la flecha y se nos pregunta por el promedio, podemos sumar los números y dividir por la cantidad de números (9 hasta el 65) dando:

$$\frac{(3 + 4 + 8 + 10 + 12 + 34 + 12 + 54 + 65)}{9} = 22.44$$

Esto cumple con entregar el promedio de los números; sin embargo, es muy ineficiente ya que tengo que almacenar todos los números anteriores (usando mucho espacio) y sumarlos todos cada vez que necesito el promedio (mucho tiempo). En cambio, un algoritmo streaming muy sencillo puede hacer esto de una manera muy eficiente almacenando la suma parcial de los datos que hemos visto, **sum**, junto con la cantidad de números, **cant**. Luego, para cada número nuevo *n* actualiza los valores como:

```
sum  := sum + n
cant := cant + 1
```

Ahora, cada vez que se nos solicite el promedio de los números anteriores, solo necesitamos hacer el cálculo **sum/cant**, entregando el promedio esperado. Note que ahora lo único que debemos almacenar son estos dos números, y que actualizar estos valores solo nos toma dos operaciones por cada nuevo número.

Lamentablemente, no todos los problemas en procesamiento de datos streaming tienen una solución tan sencilla como la anterior. En esta área se han estudiado diversos problemas de procesamiento streaming como, por ejemplo, encontrar los datos más frecuentes o contar la cantidad de datos distintos que han ocurrido en un stream. Estos problemas muchas veces no tienen un algoritmo que pueda calcular, con poco espacio y tiempo, el resultado exacto. Por esto, muchos algoritmos streaming usan aleatoriedad para calcular un valor aproximado a la solución, cambiando eficiencia por precisión. Ésta es una área muy desafiante del diseño de algoritmos que ha logrado grandes avances en los últimos años.

SISTEMAS DE BASES DE DATOS STREAMING

A simple vista, los algoritmos streaming parecen ser una buena herramienta a la hora de manejar grandes volúmenes de datos, pero en la práctica resultan demasiado difíciles de utilizar o de programar. Por otro lado, tampoco queremos tener que programar un algoritmo desde cero cada vez que estamos interesados en extraer cierta información valiosa de un stream. Para hacer esto, el uso de un sistema de manejo de datos streaming (similar a lo que se conoce como un sistema de bases de datos) parece el correcto: tener un sistema (software) que se encargue de leer y gestionar nuestro stream de datos, mientras nosotros solo debemos escribir nuestra pregunta con un lenguaje declarativo (como SQL) y luego dejar que el sistema se encargue de responder la pregunta eficientemente. Ésta es la dirección que se ha tomado en el área de manejo de eventos complejos (CEP, del inglés Complex Event Processing). Un sistema de CEP considera al stream de datos simplemente como una secuencia de eventos, donde cada dato es visto como un

FIGURA 1
EJEMPLO DE UN STREAM DE DATOS EN TWITTER.

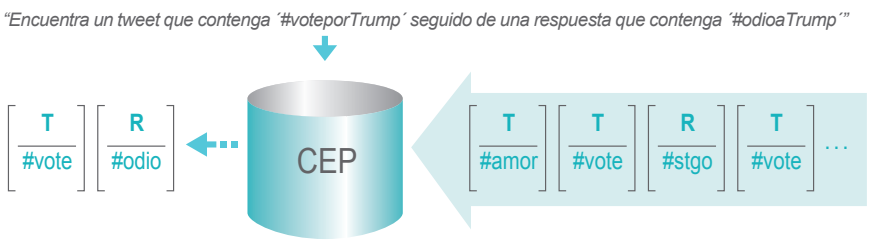


FIGURA 2
DIAGRAMA DEL FUNCIONAMIENTO DE UN SISTEMA DE MANEJO DE EVENTOS COMPLEJOS (CEP: COMPLEX EVENT PROCESSING).

evento o suceso. Por ejemplo, un tweet en Twitter puede ser visto como un evento **T**(`post`) donde **T** me dice que el evento es un tweet y `post` es su contenido. Otro evento podría ser **R**(`reply`), donde la **R** indica que es un evento de respuesta, en este caso con un mensaje de respuesta `reply`[1]. En la **Figura 1** podemos ver un posible stream de eventos **T** y **R** de Twitter, donde los eventos van llegando de izquierda a derecha (por presentación, los mensajes de cada tweet o respuesta han sido acortados a un hashtag). Ahora que tenemos un esquema que nos dice la estructura de los datos, podemos consultar al sistema CEP por una pregunta de interés o, en la nomenclatura de CEP, por un evento complejo. Situémonos en el caso que somos un periodista interesado en debates políticos y tenemos acceso al stream de eventos de tweets y respuestas de Twitter. Una pregunta de nuestro interés puede ser, por ejemplo, saber cuándo ocurre un debate en Twitter sobre Trump. Esta pregunta se podría enunciar de manera precisa como:

" Encuentra un tweet que contenga '#voteporTrump' seguido de una respuesta que contenga '#odioaTrump' ".

Si vemos nuestro stream de la **Figura 1**, una respuesta posible a esta pregunta es el par de eventos en las posiciones 1 y 5, que contienen exactamente los datos que buscamos. En otras palabras, un evento complejo no es más que un conjunto de posiciones que certifican la ocurrencia del evento.

Formalizando las ideas anteriores, el funcionamiento de un sistema CEP se puede entender con el diagrama en la **Figura 2**. El sistema lee el stream de datos (llamados eventos) y los usuarios envían consultas de interés a la base de datos. Apenas el sistema verifica que este evento complejo ha ocurrido (por ejemplo llega una respuesta **R** con '#odioaTrump' después de un tweet **T** con '#voteporTrump'), el sistema le entrega una notificación o alerta al usuario junto con los eventos que respaldan el evento complejo. Una vez que la alerta fue entregada al usuario, el sistema CEP sigue procesando los eventos en busca de nuevos eventos complejos que satisfagan la pregunta. Es importante notar aquí la diferencia de los sistemas CEP con un sistema de bases de datos tradicional (como Oracle o PostgreSQL). En un sistema tradicional los datos son estáticos (esto es, con pocas modificaciones) y son las consultas las que cambian continuamente. En cambio, en un sistema CEP las consultas son estáticas y son los datos los que cambian continuamente.

A pesar de que los sistemas CEP parecen una buena solución para gestionar datos streaming, hasta el momento estos sistemas no han tenido el protagonismo que uno espera en la práctica. Actualmente existen algunos sistemas ya asentados en la industria (como EsperTech) y algunos prototipos en la academia que son usados como referentes. Lamentablemente, estos sistemas carecen de una base sólida, con lenguajes de consultas poco claros y técnicas de evaluación deficientes. De hecho, es mencionado continuamente en la literatura que los lenguajes de consultas para CEP carecen de una semántica clara, siendo muchas veces confusos y difíciles de entender. Nuestra experiencia también nos ha mostrado que muchos de estos sistemas son difíciles de utilizar y configurar, funcionando correctamente solo para grupos particulares de consultas.

En nuestro proyecto (FONDECYT 11150653) buscamos dar formalidad a estos sistemas CEP. Para ello, proponemos empezar desde cero, dando respuesta a las preguntas más básicas (¿qué es un evento?, ¿qué es un evento complejo?, etc.) y reconociendo cuáles son las componentes fundamentales de estos sistemas. Ya con esta base definida, desarrollamos algoritmos que aseguran una ejecución eficiente y eficaz del sistema. A continuación, presentamos una breve reseña de nuestra investigación junto con algunos de los resultados que hemos obtenido hasta el momento.

NUESTRA PROPUESTA

La primera tarea en esta investigación fue formalizar el lenguaje de consultas para definir los patrones que el usuario desea utilizar. En la literatura se han propuesto más de 20 lenguajes de consulta distintos, con variados operadores y semánticas. De todos estos operadores, reconocimos los más recurrentes y que son fundamentales para la detección de eventos complejos. Estos operadores incluyen el de secuenciación (;), selección, disyunción (or), e iteración (similar a una clausura de Kleene), entre otros. Para dar una idea general de nuestro lenguaje, veamos el siguiente ejemplo. Recuerden la pregunta del periodista que buscaba debates sobre Trump en el stream de tweets (**T**) y respuestas (**R**). Esta pregunta

1. Twitter maneja una estructura muchísimo más compleja de datos que incluye identificadores, fechas (timestamp), retweets, follow y unfollow, entre otros. Aquí usamos una visión más sencilla de los datos de Twitter para simplificar la presentación.

se podría definir con nuestro lenguaje teórico[2] de la siguiente manera:

(**T** ; **R**) FILTER (**T**.post = '#vote'
AND **R**.reply = '#odio')

Para simplificar la presentación, en la expresión anterior abreviamos '#voteporTrump' por '#vote' y '#odioaTrump' por '#odio'. Esta expresión se puede leer como: quiero un tweet **T** seguido de una respuesta **R**, donde el mensaje de **T** contiene '#voteporTrump' y el mensaje de **R** contiene '#odioaTrump'. Si preguntamos por esta consulta en nuestro sistema **CEP**, sobre el stream de la **Figura 1**, entregaría el par de datos en las posiciones 1 y 5, dado que cumplen con la especificación anterior. Aquí el operador de secuenciación (; en símbolos) permite que entre **T** y **R** haya una cantidad arbitraria de eventos. Éste es un comportamiento esperado de un sistema **CEP** debido a que los streams de datos usualmente contienen mucho ruido, y los eventos nunca suceden continuamente como uno esperaría. Parte de nuestra investigación consistió en dar un significado claro a cada uno de estos operadores, lo que resulta en que el usuario tenga claro lo que espera recibir en respuesta a consultas como la anterior.

Una vez que definimos y formalizamos nuestro lenguaje de consultas, el siguiente paso fue entender cómo evaluarlas. En general, para la evaluación de consultas en sistemas de bases de datos uno busca una representación "intermedia", esto es, un objeto abstracto que define la consulta pero que sea más fácil de entender, manejar y optimizar. Por ejemplo, en una base de datos relacional, toda consulta SQL hecha por un usuario es convertida por el sistema a una expresión en álgebra relacional, la cual es mucho más sencilla y fácil de trabajar computacionalmente. En nuestro caso, utilizamos un modelo de autómatas como representación interna, el cual llamamos "autómatas de eventos complejos" (CEA, del inglés Complex Event Automata). Un CEA es muy parecido a un autómata finito no-determinista, con la diferencia que (1) lee streams en vez de palabras, (2) sus transiciones son predicados en vez de letras, y (3) sus transiciones pueden producir

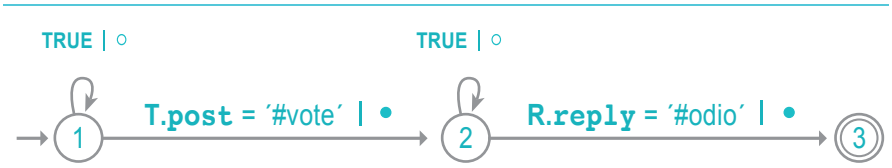


FIGURA 3
CEA QUE REPRESENTA LA PREGUNTA DE DEBATES EN TWITTER.

resultados (eventos complejos) además de leer eventos. Para tener una idea de lo que es un CEA, veamos el siguiente ejemplo. En la **Figura 3** podemos ver un CEA con tres estados, representados por círculos, y sus transiciones representadas por las flechas entre estados. La idea de estas flechas etiquetadas como (P | •) o (P | ○) es muy sencilla: si el dato d que leo cumple la condición P entonces cambia al siguiente estado y, si es de tipo •, marca d como parte del evento complejo y, si no, omite d. Como ejemplo, la flecha que nos lleva del estado 1 al estado 2 representa que si el dato d es de tipo tweet **T** y su mensaje **(post)** contiene '#voteporTrump', entonces cambiamos al estado 2 y marcamos d como parte del output. Es fácil ver que el CEA de la **Figura 3** representa, de hecho, nuestra consulta de debates, pero ahora con un modelo abstracto y procedural. En nuestra investigación estudiamos CEA formalmente y lo proponemos como el corazón de la evaluación de consultas en CEP. Es importante mencionar que en la literatura ya se habían propuesto modelos de autómatas similares para evaluar este tipo de consultas; nuestra contribución fue formalizar estos modelos matemáticamente, entendiendo sus propiedades fundamentales.

Una vez que ya tenemos una manera de representar las consultas con nuestro modelo intermedio de autómatas, la pregunta es ¿cómo evaluamos ahora estas máquinas eficientemente? Un desafío intrínseco de los lenguajes de CEP, por su capacidad de omitir datos intermedios, es que muchos eventos complejos pueden satisfacer una consulta al mismo tiempo. Por ejemplo, para nuestra consulta de debates no solo 1 y 5 cumplen con las condiciones de un debate al llegar el evento 5, si no también 3 y 5. De hecho, es fácil ver que una respuesta '#odioaTrump' puede emparejar con varios tweets de '#voteporTrump'. Por lo tanto, un sistema CEP debe llevar de manera streaming (o sea, con poco espacio y tiempo), varios potenciales eventos complejos durante su ejecución.

Una de las contribuciones más relevantes de nuestro trabajo fue definir formalmente garantías de eficiencia que los algoritmos streaming deben satisfacer para ser considerados "eficientes". Proponemos utilizar lo que llamamos "algoritmos de enumeración con demora constante" (en inglés *constant delay enumeration algorithms*). ¿Qué hacen estos algoritmos específicamente? En pocas palabras, estos algoritmos mantienen una representación compacta de todas las posibles respuestas parciales y, cada vez que un nuevo dato es recibido, actualizan esta representación compacta en tiempo proporcional al nuevo dato. Por otra parte, cuando en un momento cualquiera se requiere obtener las respuestas a la consulta, otro proceso independiente se encarga de construir todos los resultados desde la estructura compacta y enumerarlos, tomando un tiempo constante entre cada par de resultados entregados. Cabe destacar que ésta es la forma más eficiente posible de enumerar los resultados, ya que el tiempo que tarda depende solo del tamaño de la respuesta. La gracia de estos algoritmos es que separan el tiempo de actualización del sistema del tiempo de enumeración, siendo eficientes tanto en la evaluación como en la enumeración de los resultados.

En nuestro trabajo, diseñamos y proponemos algoritmos de enumeración con demora constante para un subgrupo importante de consultas en nuestro lenguaje. En particular, estos algoritmos aplican para consultas del tipo anterior, como nuestra consulta sobre debates políticos en Twitter. De hecho, estos algoritmos funcionan

2. Presentamos aquí una simplificación del lenguaje. En nuestro trabajo, también proponemos un lenguaje de usuario con sintaxis SQL.

órdenes de magnitud más rápido que los mejores sistemas desarrollados actualmente en la academia e industria. Para ver esta diferencia en la práctica, realizamos un experimento muy sencillo con las siguientes dos consultas:

$$Q_1 = A ; B ; C$$
$$Q_2 = A ; B ; C; D$$

La consulta Q_1 busca tres eventos de tipos *A*, *B* y *C*, en ese orden. Por otra parte, la consulta Q_2 además solicita que después del tercer evento C debe ocurrir un evento de tipo *D*. Con estas consultas realizamos un experimento que llamamos de "stress": cada consulta es evaluada sobre un stream donde todos los eventos son generados con distribución uniforme exceptuando el último, que ocurre solo al final (el último evento es *C* para Q_1 y *D* para Q_2). Esto implica que un sistema CEP no encontrará ningún resultado hasta que el último evento sea visto y, necesariamente, el sistema deberá "recordar" todos los resultados parciales hasta el último evento. En los gráficos de la **Figura 4** podemos ver la eficiencia de nuestra propuesta comparada con SASE y EsperTech, los dos referentes en la academia e industria, respectivamente. En el eje horizontal se muestra el número de datos procesados antes de ver el último evento, y en el eje vertical el uso de recursos en tiempo (gráficos de la izquierda) y memoria RAM (gráficos de la derecha). Como podemos ver, leyendo menos de dos mil datos (un número bastante bajo si pensamos en la cantidad de datos en un contexto real), los otros sistemas toman del orden de segundos para procesar Q_1 y del orden de minutos para procesar Q_2. En cambio, nuestra propuesta, basada en algoritmos de enumeración con demora constante, toma tiempos del orden de milisegundos en ambas consultas, muy por debajo del tiempo de los otros sistemas. Si vemos la cantidad de memoria usada durante los experimentos es incluso más dramático: los sistemas toman del orden de cientos a miles de megabytes de memoria, mientras que nuestra propuesta no alcanza los diez megabytes de memoria. De este experimento se pueden comprobar los beneficios de utilizar algoritmos de enumeración con demora constante. Los sistemas clásicos tratan de mantener todos los resultados parciales (que puede ser una cantidad exponencial) y tienen que actualizarlos todos cada vez que llega un nuevo evento. En cambio, nuestra representación compacta no necesita mucho espacio y, por otro lado, se puede actualizar rápidamente cada vez que llega un nuevo dato.

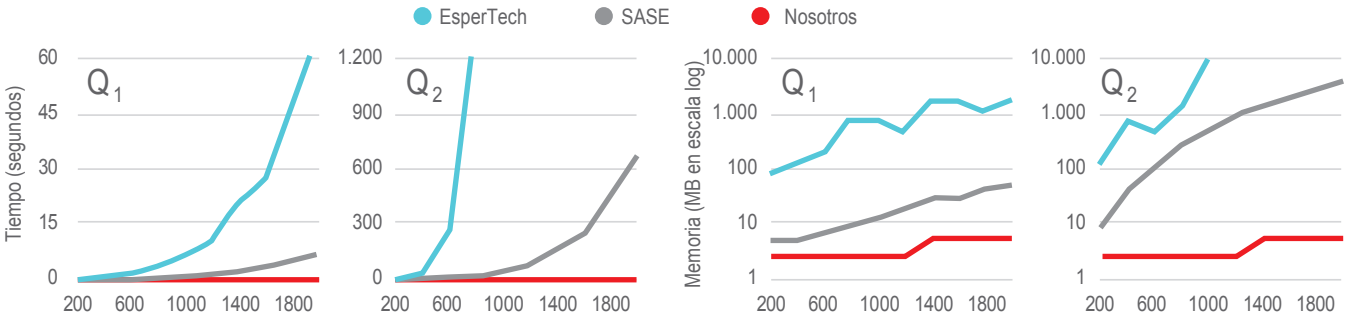


FIGURA 4
RESULTADOS DEL EXPERIMENTO DE STRESS EN NUESTRO SISTEMA, COMPARADO CON SISTEMAS DE LA INDUSTRIA (ESPERTECH) Y LA ACADEMIA (SASE).

CONCLUSIONES

COMO PUDIMOS OBSERVAR, EL MANEJO DE EVENTOS COMPLEJOS Y, MÁS GENERAL, EL MANEJO DE STREAMS DE DATOS, PROPONE GRANDES DESAFÍOS PARA LA CIENCIA DE LA COMPUTACIÓN, EN LA BÚSQUEDA DE ALGORITMOS EFICIENTES CAPACES DE PROCESAR GRANDES VOLÚMENES DE DATOS. NUESTRA PROPUESTA BUSCA SER UN PRIMER ACERCAMIENTO HACIA LA FORMALIZACIÓN DE SISTEMAS CEP, DEFINIENDO LAS BASES SOLIDAS PARA EL ESTUDIO DE ESTA ÁREA, Y DEJANDO VARIOS PROBLEMAS AÚN POR RESOLVER. ACTUALMENTE, NOS ENCONTRAMOS INVESTIGANDO NUEVOS ALGORITMOS DE ENUMERACIÓN CON DEMORA CONSTANTE PARA INCLUIR OTRAS CARACTERÍSTICAS DE ESTOS SISTEMAS, COMO MANEJO DE VENTANAS DE TIEMPO, CORRELACIÓN ENTRE LOS DATOS, Y CÁLCULO DE ESTADÍSTICAS SOBRE LOS DATOS. POR ÚLTIMO, TAMBIÉN NOS ENCONTRAMOS IMPLEMENTANDO, JUNTO A UN GRUPO DE ALUMNOS, UN PRIMER PROTOTIPO DE NUESTRA PROPUESTA, EL CUAL ESPERAMOS QUE ESTÉ LISTO EN EL CORTO PLAZO PARA SER PROBADO EN ESCENARIOS REALES. ■

▶ **La investigación de este artículo fue financiada por el proyecto FONDECYT Nr. 11150653.**