Buscador de tweets distribuido

Cárdenas Rodrigo; Iribarne Gabriel

Resumen

El objetivo de este trabajo es implementar un sistema distribuido con **NodeJS** [1], que permita recopilar tweets y los almacene una base de datos no relacional.

Con NodeJS se paralelizará la recopilación de información, ejecutándose distintas instancias de la aplicación configuradas con diferentes parámetros. Cada una de las instancias almacenará los datos recolectados en una base de datos centralizada, con el objetivo de armar corpus de documentos voluminosos para su posterior procesamiento. Para la persistencia de datos se utilizará **MongoDB**, que presenta ventajas en términos de manipulación de datos, en este caso, los tweets obtenidos en formato **JSON**.

Contexto

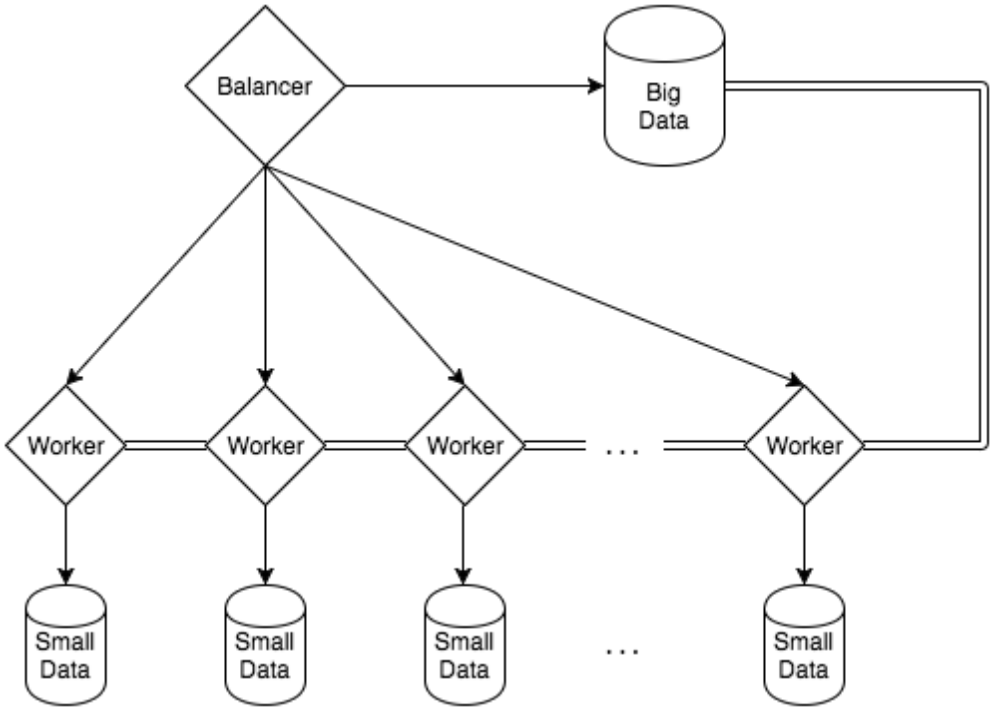
El servicio de micro-blogging de Twitter incluye dos RESTful APIs. Los métodos de la **API de Twitter** [2] permiten a los desarrolladores acceder al core de los datos de Twitter, esto incluye timelines completos, datos de tweets, e información de usuarios. Por otro lado, la **Search API** [3] presenta métodos que permiten interactuar con el módulo de búsquedas y *trends data* de Twitter. La API soporta distintos formatos de datos, pero en este caso nos enfocaremos en los resultados en formato JSON, ya que es JavaScript.

Al ser Twitter una red social masiva, el volumen de información que produce diariamente es muy grande, se generan cerca de 10000 tweets (documentos) por segundo. Y estos documentos son ampliamente utilizados en distintos campos de investigación, como por ejemplo: análisis de sentimientos, predicciones, análisis de opiniones, análisis demográficos, etc.
Como toda API pública, tiene una limitación en el servicio que brinda, por ejemplo, en el caso de la obtención de timelines de usuarios, solo permite obtener los últimos 3200 tweets y, en una ventana de tiempo de 15 minutos, solo se pueden realizar 300 requests a la API, superado este número las credenciales de conexión quedan inhabilitadas hasta que pase la ventana de tiempo. Por otra parte, la Search API permite obtener los tweets realizados en los últimos 7 días, con un máximo de 450 requests en una ventana de tiempo de 15 minutos; y, la búsqueda no se hace sobre la totalidad de tweets, sino que la Search API trabaja sobre un subconjunto del total de tweets en Twitter.

Para facilitar la obtención de documentos en Twitter, se ideó un sistema que permita utilizar la API, para la obtención de timelines, y la Search API, para la ejecución de queries de forma más rápida, resolviendo las limitaciones mediante la paralelización de las búsquedas y contando con una colección de credenciales intercambiables para que los nodos, quienes realizan las requests, no queden ociosos a la espera de la reactivación de la ventana de tiempo de la API.

Arquitectura

La arquitectura del sistema se puede visualizar en el siguiente diagrama:



El nodo superior (nodo de configuración/balancer) es el que se encargará de dialogar con los nodos de nivel inferior (workers), quienes serán los responsables de trackear la información. Estos nodos serán creados utilizando datos de parametrización de apps de Twitter, ubicados en una colección de una base de datos y, también, se conocerá cuáles son los nodos disponibles por medio de otra colección, que almacenará los datos de conexión del nodo. Por otra parte, también será utilizado como balanceador de request, ya que él recibirá las búsquedas y según de qué tipo sea, aplicará un algoritmo para dividir la búsqueda en los demás nodos worker.

Cada nodo worker será invocado con instrucciones para ejecutar una búsqueda determinada. Estos nodos abrirán una conexión con la API de Twitter y comenzarán a almacenar los Tweets obtenidos en una base de datos MongoDB, local a cada nodo. Luego de haber completado una búsqueda de forma exitosa, los nodos enviarán la data recolectada a la base de datos central.

Si el Nodo balanceador detecta inactividad en un nodo de nivel inferior, le podrá asignar nuevos request para satisfacer las búsquedas que están en cola.

En el caso de que todos los nodos lleguen al límite de request por hora definidos por Twitter, el sistema cuenta con un algoritmo de rotación de credenciales, lo que permitiría que los nodos puedan seguir realizando su labor, evitando tiempos muertos durante la espera de la recuperación de las credenciales agotadas.

La herramienta fue desarrollada íntegramente utilizando NodeJS, que funciona con un modelo de evaluación de un único hilo de ejecución, usando entradas y salidas asíncronas las cuales pueden ejecutarse concurrentemente en un número de hasta cientos de miles sin incurrir en costos asociados al cambio de contexto [4]**.** Esta característica de NodeJS es uno de los puntos claves utilizados para construir esta herramienta altamente concurrente, en la que todas las operaciones de entrada y salida, tienen una función *callback*. Tanto los nodos *balancer* como los *workers* son servidores web, y la comunicación entre esta red de nodos se hace mediante GET requests. Para realizar este desarrollo se recurrió a distintas librerías; obtenidas mediante el gestor de paquetes NPM [5], que funciona de manera conjunta con NodeJS. Esto permitió acelerar los tiempos de desarrollo. Algunas de las librerías utilizadas son:

- bluebird: que facilita el desarrollo de métodos asincrónicos y *callbacks*,
- body-parser: simplifica el manejo de requests,
- express: se utilizó para simplificar el desarrollo de los servidores web y para el manejo de requests entre *balancer* y nodo *worker*,
- moment: facilita la manipulación de variables tipo *date*,
- mongoose: utilizada para el manejo de conexiones con MongoDB,
- request: ayuda a manejar las requests HTTP,
- twitter: para facilitar la recolección de tweets.

Los documentos recuperados se formatean de la siguiente forma:

```
{
  "twid": "250075927172759552",
  "screen_name": "bullcityrecords",
  "text": "Aggressive Ponytail #freebandnames",
  "date": "Tue Apr 07 19:05:07 +0000 2009",
}
```

Donde “twid”, corresponde al ID del documento dentro de Twitter; “screen_name”, es el nombre del usuario que publicó el tweet en cuestión; “text”, es el contenido del documento; y “date”, es la fecha de publicación del documento. Se optó por esta estructura minimalista para no sobrecargar cada documento, aunque puede ser fácilmente modificable mediante programación, pudiendo adicionar más datos dentro de los que ofrece Twitter como resultado.

El sistema es capaz de optimizar las búsquedas, contando con una funcionalidad de caché, lo cual resulta útil para búsquedas frecuentes. Por cada búsqueda realizada y finalizada exitosamente, se almacena dicha búsqueda en una tabla de la base de datos, junto con información del último ID de Twitter recuperado para dicha búsqueda. De esta forma, si se repite una búsqueda ya realizada, se hará una request a Twitter indicando que los ID de los resultados deben ser mayor que el valor de ID que se pasó por parámetro. Como consecuencia, se aceleran los tiempos y se evitan datos repetidos.

El sistema, además, cuenta con una funcionalidad de recuperación de nodos. Siendo que el nodo balanceador puede administrar N nodos workers; si, por algún motivo, uno de los

nodos worker queda fuera de servicio, la búsqueda que está realizando no se perdería, sino que se recupera. Mediante un chequeo periódico de estados, el nodo balanceador puede conocer si un nodo quedó fuera de servicio mientras estaba ejecutando una búsqueda, y volver a colocar dicha búsqueda en una cola para que sea ejecutada por otro nodo.

Casos de estudio y resultados

Se realizaron distintas pruebas con diferentes configuraciones de nodos y distintos tipos de búsquedas, con el objetivo de establecer una relación entre tipo de búsqueda, tiempos de búsqueda, cantidad de nodos activos, y comprobar la escalabilidad del sistema.

Por ejemplo, pudimos observar que se puede obtener un *timeline completo* (tomando en consideración las limitaciones de la API) de un usuario random con 3200 tweets o más, para lo cual se realizan 17 requests a la API de Twitter y se recuperan 3200 documentos en 42 segundos.

En una nueva instancia escalamos el ejemplo anterior, y decidimos recuperar timelines de 200 usuarios de Twitter, la búsqueda demoró 140 minutos, aproximadamente. Luego, al activar un nuevo nodo, el tiempo se vió reducido a la mitad. El sistema se hace más eficiente conforme más nodos se agregan a la configuración, por lo tanto, el tiempo de recuperación de documentos para una búsqueda de un volumen considerable de usuarios, es inversamente proporcional a la cantidad de nodos activos.

En el siguiente caso, se comparó la recuperación de distinta cantidad de timelines, utilizando dos configuraciones de nodos distintas:

Timelines	Nodos	Tiempo	Documentos obtenidos
10	4	125 seg	32K
50	4	550 seg	160K
100	4	1113 seg	320K
10	8	66 seg	32K
50	8	234 seg	160K
100	8	563 seg	320K

Como se puede observar, la relación entre cantidad de nodos y tiempo de búsqueda es inversamente proporcional, es decir, a mayor número de nodos, el tiempo de búsqueda va en decremento, siempre y cuando se busquen más de un timeline a la vez.

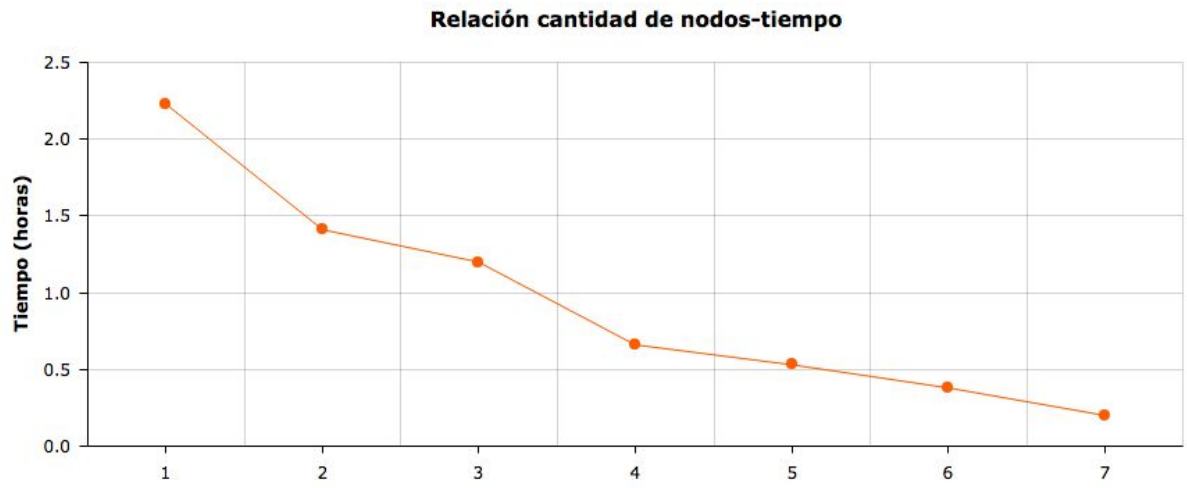
En otro caso, contamos con una búsqueda realizada en el sistema utilizando el término “arroz”, simplemente para elegir un término que, dentro del rango de búsqueda de 7 días, muestre una distribución regular. En la base de datos se cuentan con documentos con

fecha desde el 14/10/2016 hasta el 21/10/2016, y el tiempo de búsqueda utilizando 1 nodo fue de 2 horas 14 minutos. Se repitió la búsqueda el día 22/10/2016, esperando que se completen los documentos solo del día restante. La búsqueda se ejecutó utilizando 1 nodo (no tiene sentido usar más de 1 ya que solo se debe recuperar 1 día) y el tiempo de recuperación fue de 11 minutos 56 segundos.

La búsqueda se realizó sobre la Search API y la query utilizada fue el término “arroz”, nuevamente. La búsqueda arrojó casi 95000 documentos (tweets). En la siguiente tabla se ven reflejados los tiempos de recuperación de cada configuración probada.

Configuración	Tiempo	Documentos obtenidos
1 nodo	02:14:48	94.7K
2 nodos	01:25:31	94.7K
3 nodos	01:12:17	94.7K
4 nodos	00:40:39	94.7K
5 nodos	00:32:56	94.7K
6 nodos	00:23:07	94.7K
7 nodos	00:12:28	94.7K

Se puede observar un decremento en los tiempos de recuperación, comparando con la cantidad de nodos activos en el sistema.



Si se agregara un octavo nodo, el tiempo no mejoraría para esta búsqueda, ya que, como se mencionó anteriormente, la Search API solo permite recuperar tweets dentro de los últimos siete días. Tendría sentido agregar más nodos, cuando el objetivo es recuperar tweets de más de una búsqueda, en este caso, los tiempos de búsqueda seguirán disminuyendo a medida que se añadan más nodos al sistema.

Conclusiones

Con las bondades que brinda NodeJS, se puede paralelizar el proceso de búsqueda con una infraestructura mínima, pudiendo crear una cantidad razonable de nodos y, de esta forma, reducir considerablemente los tiempos de búsqueda sobre la API de Twitter. Siempre y cuando la cantidad de nodos activos y la cantidad de credenciales de Twitter sea sustanciosa.

Como bien se dijo previamente, es una herramienta que permite salvar las limitaciones de las API de Twitter, pudiendo aprovecharla de mejor forma.

Una vez configurado, no requiere monitoreo ni trabajo de mantenimiento, ya que los requests se encolan en el nodo balanceador y este distribuye de forma autónoma las queries entre los nodos disponibles.

Los datos son obtenidos y almacenados en formato JSON, lo cual minimiza el impacto en los recursos de almacenamiento y la estructura de datos es compacta y sencilla de procesar.

Referencias

[1] NodeJS - https://nodejs.org/es/
[2] API - https://dev.twitter.com/rest/public
[3] Search API - https://dev.twitter.com/rest/public/search
[4] http://blog.caustik.com/2012/08/19/node-js-w1m-concurrent-connections/
[5] NPM - https://www.npmjs.com/