Implementación de un sistema de cómputo distribudo Map-Reduce sobre AMQP

Juan Bautista Cabral [1],
[1] IATE-OAC;

El advenimiento de grandes volúmenes de datos (o Big Data) esta generando una necesidad de productos que sirvan para la manipulación y resumen de los mismos. Big Data puede considerarse de manera mas o menos precisa (no es mas que un numero comercial) como una cantidad de información tal que no puede procesarse ni almacenarse en un único ordenador.

Las dificultades más habituales vinculadas a la gestión de estas cantidades de datos se centran en la captura, el alma- cenamiento, búsqueda, compartición, análisis y visualización. La tendencia a manipular ingentes cantidades de datos se debe a la necesidad en muchos casos de incluir los diferentes conjuntos de datos relacionados.

La tendencia actual es el almacenamiento y el procesamiento a través de nodos distribuidos en una red de la manera mas transparente posible para el programador, haciéndolo parecer que esta ejecutando todo localmente; despegándose un poco del modelo propuesto por el ya tradicional modelo distribuido de MPI (del inglés Interfaz de Paso de Mensaje) de hacer evidente la no localidad del computo.

Por el dado del análisis de datos; en la presencia de una cantidad ingente de información como la que planteamos hace necesario el disponer de mecanismos automáticos para el procesamiento de estos volúmenes. Es en este campo donde una herramienta como el aprendizaje automático (o ML) obtiene un valor de piedra angular. El aprendizaje automático es una rama de la de la Inteligencia Artificial que consiste en crear programas que buscan de manera autónoma patrones en la información a partir de ejemplos.

En la actualidad la herramienta mas popular de computo distribuido es sin dudas Hadoop. Hadoop es una implementación del algoritmo MapReduce realizada íntegramente en Java, que ha servido de inspiración y guía a lo en todo este trabajo.

Modelos Computo distribuido

AMQP: (del ingles *Advanced Message Queuing Protocol* - Protocolo avanzado de colas de mensajes) es un estandar en el nivel de aplicación para middlewares oriantados a mensajes. Sus características principales son:

- Orientación a mensajes
- Colas
- Enrutamientos (punto a punto y publicador-suscripción)
- Exactitud y seguridad

MapReduce: es un modelo de cómputo que simplifica el uso de clusters; el cual computa una función que recibe como parámetros un conjunto de elementos *llave-valor*, y lo convierte en un nuevo conjunto el cual cada elemento es una *llave* y una *lista de valores*.

$$f(\{(k_{in},v_{in})\}) \rightarrow \{(k_{out},list(v_{in})\}$$

Para lograr esto divide la operación en dos etapas: **Etapa de Map:** Transforma el conjunto de entrada en un lista intermedia de elementos *llave de salida* y un *valores intermedios* para esa llave. **Etapa reduce:** Agrupa cada elemento de la lista intermedia según la llave final a la que pertenece y genera una nueva lista de salida para cada par *llave lista intermedia* de entrada

MapReduce se emplea en la resolución práctica de algunos algoritmos susceptibles de ser paralelizados. Cabe aclarar que MapReduce si bien es poderoso no sirve para cualquier problema, así como no es la forma mas eficiente para todos los problemas que si son solucionables por esta técnica.

Antecedentes

Hadoop esta programado y desarrollado sobre Java. Java actualmente, es la tecnología que esta acaparando la mayoría de los sistemas modernos para el almacenamiento y procesamiento de grandes volúmenes de información. Su elección radica principalmente en su plataforma subyacente: la JVM. que posee un mecanismo de concurrencia que hace útil para la escritura de sistemas que requieran alta disponibilidad y escalabilidad.

Por otro lado Python es un lenguaje de mas alto nivel que que Java, esta soportado una plataforma mucho menos robusta que la JVM; pero aun así se esta destacando de manera notable en el ámbito científico. Frente a la robustez de la JVM, Python ofrece un lenguaje con una sintaxis mas clara y limpia, además de un set de librerías propias y de terceros con un stack científico muy robusto y extensible.

Python, Hadoop y AMQP

Se dispone actualmente e varias implementaciones para interactuar entre Python y Haddop, de las cuales la mas populares son Pydoop , Hadoopy y MrJob. En todos los casos no se disminuye el esfuerzo de despliegue de los nodos Hadoop.

Como interfaces a otros sistemas de computo distribuido Python encuentra varias librerías para el trabajo contra AMQP de las cuales caben destacar Pika que ofrece acceso a las primitivas de los mensajes, y Celery (de un poco mas alto nivel)

Motivación

El proyecto en un principio nació con la necesidad de comprender las implicatorias de el diseño de un sistema MapReduce sobre AMQP tratando de lograr las mismas garantías que ofrecidas por Hadoop. Este planteo fue realizado en el segundo práctico de la materia **Aprendizaje automático sobre grandes volúmenes de datos** dictada en la *Facultad de Matemática, Astronomía y Fisica* de la *Univesidad Nacional de Córdoba* en el 2do. Semestre del 2014

Actualmente se planteo evolución del proyecto evolución y se volvió un protipo funcional de lo planteado, aprovechando la expresividad de Python; tratando de facilitar el depliegue de los nodos para hacerlo de manera trivial para lograr utilizar de manera distribuida Scikit-Learn.

El proyecto fue finalmente llamado Poopy

Implementacón

Si bien Python posee varias alternativas para el acceso a AMQP, en este trabajo se ha optado por uno de los mas simples: Pika. Pika tiene la particularidad de ser una implementación robusta, probada y bien documentada, ya que el broker que hemos usado para el desarrollo, rabbitMQ, da los ejemplos en Python con Pika y las conexciones Bloqueantes en un esquema multiproceso comunicados.

Poopy actualmente tiene 3 funcionalidades básicas divididas cada en una serie de colas diferentes implementadas con el patrón publicador subscriptor

Las funcionalidades son:

- **Anunciar la existencia de un nodo cliente al central:** Esto cociste en el anuncio al nodo principal conectado al broker cada cierta cantidad de segundos de la existencia y el estado de la configuración de un nodo de procesamiento. Por el lado del nodo central cociste en un proceso que registra cuando fue el ultimo momento que un nodo de comunico y verificar su disponibilidad.
- **Distribuir los archivos en en el sistema de archivos distribuido**
- **Ejecución de una tarea MapReduce.**

Ejemplo: Random Forest Sobre *Iris.arff*

```
# imports de poopy random, sklearn, numpy y scipy

class Script(script.ScriptBase):

    def map(self, k, v, ctx):
        attrs = ['sepallength', 'sepalwidth',
                 'petallength', 'petalwidth']
        random.shuffle(attrs)
        attrs.pop()

        data, meta = v

        target = np.array(data['class'])
        train = np.array(data[attrs][:75])
        X = np.asarray(train.tolist(), dtype=np.float32)
        dt = tree.DecisionTreeClassifier(
            criterion='entropy', max_features="auto",
            min_samples_leaf=10)
        ctx.emit(None, dt)

    def reduce(self, k, v, ctx):
        for vi in v:
            ctx.emit("iris", vi)

    def setup(self, job):
        job.name = "Random Forest"
        job.input_path.append(
            ["poopyFS://iris.arff", self.readers.ARFFReader])
```

Conclusiones

Si bien el proyecto en su estado actual es utlizable y posee bastantes característica útiles, es conveniente realizar cambios si se desea usar Poopy en un entorno real. Por mencionar algunos:

- **Crear una cola para el pasaje de errores:** Esto es importante para que el nodo central sepa por que fallo un nodo de cómputo.
- **Crear un sistema distribuido de archivos real:** Actualmente lo único que hace Poopy es dejar todos los archivos en todos los nodos. Esto implicaría guardar en el nodo de aviso de vida que pedazos de archivos tiene que nodo para distribuir correctamente las tareas.
- **Mejorar los formatos de salida:** Actualmente solo se genera un formato binario bastante inútil si lo que se quiere es generar reportes legibles por humanos.
- **Implementar herramientas para navegar el contenido de poopyFS.**

Cabe aclarar que AMQP para lograr una robustez similar a la de Hadoop es necesario desplegar mas un broker; ya que una caída de un único nodo central implicaría que todo el sistema falle, al punto de que se volvería inaccesible los datos almacenados en poopyFS.

Contacto: jbc.develop@gmail.com