Título

El objetivo de este trabajo es paralelizar un algoritmo existente, Progressive Hedging, que resuelve problemas estocásticos.

TOC

- Primero explicaremos conceptos necesarios para el entorno en el que trabajaremos.
- Luego veremos cómo se gestionó el la realización del proyecto.
- La parte principal será explicar el análisis, diseño e implementación del proyecto.
- Por último las pruebas de rendimiento y conclusiones.

Programación estocástica

La programación estocástica resuelve problemas estocásticos. Los problemas estocásticos son problemas de optimización, donde debemos maximizar o minimizar uno o más valores.

Lo que los hace diferentes a los problemas de optimización convencionales es que existe un nivel de incertidumbre. Alguna de las variables de las que depende la optimización no tiene un valor concreto. Este valor estará acotado entre un valor superior, que sería una estimación positiva, y un valor inferior que sería una estimación negativa.

En función de esta incertidumbre se generan varios posibles escenarios.

Programación estocástica (dibujo)

Por ejemplo, si la incertidumbre es el beneficio que aportará un producto, podemos hacer tres estimaciones en función de un histórico.

Ejemplos

Los problemas estocásticos son comunes en el mundo real y estos son 3 ejemplos:

1. El problema de rutas de vehículos es una variante del problema del viajante pero tenemos varios vehículos y cada uno tiene una capacidad concreta. Para estas rutas la incertidumbre puede venir en función del tráfico, la cantidad de elementos que puede llevar cada vehículo (tamaño variable).

En los otros, se busca optimizar el beneficio en función de la producción de energía que es variable.

Pyomo

Pyomo es el proyecto sobre el que trabajaremos. Es un programa de código abierto que se usa para solucionar problemas de optimización de todo tipo. Permite escribir un script en python que defina tu problema y lo soluciona.

Usa solvers de terceros. Esto son programas que realizan realmente los cálculos en función de un problema de entrada. Una interfaz común de entrada para este tipo de programas es AMPL, que también está soportada por Pyomo.

Nos centraremos en el módulo PySP que soluciona problemas estocásticos.

Objetivos

Analizar el funcionamiento del algoritmo.

Ver alternativas para la implementación paralela.

Diseñar e implementar la nueva implementación.

Una vez hecho, ver las ganancias de rendimiento.

Alcance

Adaptar el módulo existente, integrando la nueva solución con Pyomo.

La nueva implementación paralela deberá mostrar escalabilidad.

Entregable principal el informe de rendimiento.

Entregables

Código

Rendimiento

Memoria

Resto de documentación. (Diseño, Registros de progreso, etc).

Metodología

Tendremos las siguientes precondiciones:

Entonces elegimos una metodología en cascada. De esta forma será más sencillo hacer un seguimiento del avance y acogernos a la fecha de fin establecida.

EDT

Eso

Linea base

Tenemos las tareas anteriores ordenadas en cascada, con una división de tiempo más o menos igual para análisis, diseño e implementación.

Primer retraso

Pero no se cumple esta planificación. El proyecto es bastante grande, nunca había hecho nada con Python y la teoría de problemas estocásticos también es algo nuevo.

Por estas razones el análisis se alargó más de lo planeado y es necesario hacer ajustes. En principio podemos reducir el tiempo de diseño y mantener los tiempos iniciales.

Prototipos

En el punto anterior dejamos poco tiempo para el diseño y era probable tener más retrasos en cascada por las mismas razones iniciales.

Entonces modificamos el cronograma a un ciclo de vida por prototipos. Esto es más adecuado para probar herramientas con las que nunca se ha trabajado antes.

Línea base final

A pesar de todo esto, la implementación principal lleva más de lo esperado y es necesario mover la fecha de finalización del proyecto. Este retraso fue por un error escondido y derivado de que, al ser un proyecto de código abierto, mucha gente colabora en lugares distintos del programa. Por mala suerte alguien decidió borrar un parámetro que era necesario si los workers funcionaban en paralelo y se tenían que serializar.

Progressive Hedging

Algoritmo iterativo. Si recordamos el árbol de escenarios anterior, podemos descomponer el problema en varios subproblemas que se resuelven normalmente. Una vez resueltos los subproblemas hay que hacer que converjan a una solución única.

Pseudocódigo

En la primera iteración se calcula la solución de cada escenario, en este caso una minimización. Se calcula un valor común (x_hat), teniendo en cuenta las probabilidades (Pr) de cada escenario.

En las siguientes iteraciones se va ajustando este valor y se usa rho (el penalizador) para llegar a la convergencia.

Implementación en pyomo

Primero importa el modelo (definido para pyomo).

Genera los objetos necesarios para la ejecución. Son configurables por el usuario.

Cada iteración son 3 fases:

1. Prepara los escenarios para adaptarse a la interfaz de entrada del solver.
2. Ejecuta dicho solver.

3. Recupera la salida, la adapta a pyomo y hace los cálculos de pesos y convergencia para la siguiente iteración.

Implementación en pyomo

Es una arquitectura basada en tareas. Cada acción se empaqueta con un identificador junto a los datos necesarios para ejecutarla.

Esto nos ayuda a intercambiar el módulo de solución, simplemente tenemos que interceptar esas tareas y ejecutarlas de forma paralela.

Herramientas

Para la implementación se valorarán spark o mpi

Spark

Orientado al big data y de código abierto.

Ofrece una capa de muy alto nivel para ejecutar tareas de forma distribuida. Se encarga de la división del trabajo y gestión del clúster.

Trabaja sobre RDD. Estos son los objetos que permiten la paralelización en spark. Estos objetos contienen datos y se distribuyen automáticamente entre los nodos del clúster. Hay dos métodos de interacción con los RDD, transformaciones y acciones.

Adicionalmente hay otros módulos que no trabajan directamente con RDD. Colecciones ordenadas, machine learning, análisis de datos en tiempo real. Pero no nos interesan para este proyecto.

MPI

Interfaz de más bajo nivel. Es un estándar entonces podemos desacoplar totalmente los trabajadores, incluso implementarlos en otros lenguajes. Tiene métodos para distribuir datos. Hay que tener cuidado sincronizando el envío y recibimiento de mensajes.

Diseño

Esto es la implementación actual en pyomo. Ya existe una implementación paralela que usa Pyro, una librería de objetos distribuidos en python.

Tenemos un entorno principal que ejecuta el bucle del algoritmo. Un solver_manager que recibe las tareas y los phsolverserver que serán los trabajadores que se ejecutan en paralelo. Cada worker ejecutará un escenario concreto, es decir una de las ramas que definía un subproblema.

Proceso

Empezamos la implementación.

Vamos a generar 3 prototipos.

Primero uno que será independiente de Pyomo. Esto nos servirá para ver cómo funciona Spark y su integración con Python.

Luego usamos lo aprendido en este prototipo y hacemos una implementación inicial integrada con pyomo.

Por último finalizamos la funcionalidad buscada.

Funcionamiento

Concretamente con spark veremos como funciona.

Lo que hicimos fue implementar un nuevo solver_manager que reciba las tareas y las distribuya en spark.

Podemos reutilizar los workers anteriores usando un wrapper para las nuevas funciones que necesitemos.

Lo primero que hacemos será inicializar los workers necesarios (que era el número de escenarios) y esta lista de workers formará el rdd (el cuadro verde).

Se les envía una tarea a los workers con un identificador para el worker que la tiene que ejecutar (en función de nuevo del escenario que tenga cada uno). Esto es una transformación de spark, modifica el rdd generando uno nuevo (que tiene las tareas en cola) pero el runtime de spark todavía no la ejecuta.

Esto nos permite encolar múltiples tareas con un tiempo de ejecución de Spark 0.

Cuando se solicita un resultado, se ejecutará otra transformación con los workers en un nuevo estado y los resultados guardados en un buffer. En este punto se realiza un collect, que es una acción de spark y se obtienen los resultados. Hasta que se ejecuta esta acción, spark no realiza las transformaciones que habíamos encolado antes.

Pruebas

Se planean pruebas funcionales para las entradas de configuración posibles. Para conseguir una mayor cobertura de pruebas sería necesario crear problemas estocásticos específicos y no contamos con el tiempo necesario.

Para realizar pruebas más realistas se ejecutan los ejemplos que existen en el repositorio de pyomo.

En esta presentación vamos a ver las pruebas de rendimiento que es lo interesante.

Recursos

Los ejemplos escogidos para probar el rendimiento fueron Finance y Hydro.

Probaremos escalabilidad entre 1 y 8 hilos. No se pudo usar un clúster para pruebas así que será un ordenador normal.

Mediremos el tiempo total para comparar entre ejecuciones distintas. Y el tiempo medio por iteración para ver el overhead de spark.

Hydro

Aquí vemos el tiempo con 1-8 hilos y modificando la memoria disponible para spark.

De 1 a 2 cores vemos una escalabilidad casi perfecta, tardando aprox. la mitad. Pero el rendimiento se estanca ahí.

En cuanto a la memoria, vemos que gestionar un mayor número de hilos es más intensivo en la memoria y con sólo 2g vemos una penalización de rendimiento.

Tiempo medio

Aquí se puede ver más claramente el problema de estas pruebas. La escalabilidad de 1 a 2 hilos vuelve a ser muy buena. Sin embargo, el tiempo por iteración es muy bajo y casi no hay diferencia entre estos dos problemas. La cuestión es que el problema Finance no es capaz de finalizar en el portátil de pruebas pero Hydro si. Aunque tenemos buena escalabilidad, estamos con problemas pequeños y el tiempo por iteración no se puede reducir mucho más por el overhead de spark. A pesar de ser "problemas pequeños" ejecutándose en spark es demasiado intensivo a nivel de memoria para ejecutarlo en el portátil de pruebas.

Con un clúster más adecuado y un problema mayor es probable que nos encontremos con números mucho más prometedores.

Conclusiones

Spark es una gran herramienta. Aunque esté enfocada al Big Data es lo suficientemente flexible como para usarse en un entorno científico.

La arquitectura que estaba implementada nos permitió integrar el nuevo módulo de forma muy satisfactoria. Simplemente cambiar un parámetro en la ejecución y el algoritmo se ejecuta con spark.

Aunque no tenemos los recursos necesarios para mostrar mejoras con respecto a la implementación secuencial, los valores de escalabilidad son muy prometedores.

Lecciones aprendidas

Es importante dedicarle un poco más de tiempo a la planificación o al final se perderá más tiempo haciendo ajustes.

Spark es capaz de ejecutar las tareas que hicimos. Pero es más adecuado para un modelo de programación funcional. Hubo problemas de dependencias en la serialización, referencias desde dentro de Spark hacia fuera, etc.

Gestión de la configuración muy importante para realizar pruebas. Cuando se busca un error podemos cambiar grandes cantidades de código y tener un seguimiento de los cambios así como poder volver a una versión funcional en cualquier momento, agiliza el proceso de pruebas.

Futuro

Pyomo es open source, puede seguir evolucionando por cualquiera.

Repetir las pruebas con recursos adecuados para evaluar el rendimiento real conseguido y posibles rutas de optimización.