High Speed Parsing

Massive XML in Ruby

Be faster than Python sample code!

Dogrun Inc. Tetsuya Hirota

RubyWorld Conference 2023

Introduction

Name: Tetsuya Hirota

Job Title: Chief Technology Officer (CTO) at Dogrun Inc.

Location: Shizuoka-city Japan

My Job:

- Bioinformatics database development.
- Developing applications using deep learning for our clients (which has been quite interesting **:-**)

Volunteer:

Programming courses mainly for children in Makinohara**-c**ity.

Dogrun's Expertise

Expertise:

Following in the field of life sciences.

- Database construction
- Development of robust data retrieval systems
- Creating powerful data visualization tools

Today's Focus:

High-speed parsing massive XML
as part of bioinformatics database development

BioProject

Public database for Bio science

- Such as genomes and RNA data are registered.
- Along with related metadata such as paper information, experimental conditions, and samples.

BioProject – a part of public database

- Metadata about sequence analysis projects such as genomes and RNA.
- Single 2GB XML file containing 700,000 records.
  (It’s getting bigger day by day.)

Problems

- Limited search patterns provided by the database.
- Complex searches on XML takes a lot of time.

Parsing XML Normally

Parsing BioProject XML and, output the extracted JSONL.

- **ReXML**
  - **Frozen somewhere and no response...**
- **Nokogiri**
  - **It took 35 minutes and used over 10GB of memory.**
  - **It is considered slow because it loads all objects into memory and searches for XPaths.**
- **Python sample (using iterparse())**
  - **It took 8+ minutes.**
  - **Each element in the first layer is lazily loaded.**

Make something like iterparse() in Ruby

- Nothing in Ruby like iterparse() → I make it!
- Design (Consists of 3 parts)
  - Split by 1st layer element lazily

    Each time a 1st layer element is read, that element is passed to the next process.
  - Parse splitted XML

    Normal parsing for splitted XML.
  - Output extracted JSONL

    Search XPath from splitted XML objects and output extracted JSONL
- Named “enumparse”

Making enumparse Using SAX

- String operations
  - Extract the range from the start tag to the end tag of an element using string operations. ⇒ Slow
- Using SAX
  - SAX?

    SAX (The Simple API for XML). Unlike DOM, SAX reads a document sequentially from the beginning and transmits information to the application via events.
  - SAX included in Nokogiri
  - Extract a 1st layer element using SAX. ⇒ Practical speed (13 min.)

Ox seems Fast

Couldn't be faster the splitted XML parsing part?

- Ox
  - Found a library that can parse faster.
  - https://github.com/ohler55/ox

Gem	Parse	to_s
Nokogiri	3.56 seconds	7.03 seconds
LibXML	3.67 seconds	0.67 seconds
Ox	1.90 seconds	0.33 seconds

http://www.ohler.com/dev/xml_with_ruby/xml_with_ruby.html

  - Change to parsing part to Ox. ⇒ Fast (about 8 min.) (enumparse + Ox)

- Now it can be in the same processing time as Python code. Completed. That's not it?

enumparse + Nokogiri + Ractor

- Ruby users are got used to Nokogiri, I think.
  And I want to enumparse + Nokogiri to be faster.
- There are Ractor in Ruby.
  - Parallelize reading, parsing, and writing for faster processing
  - Parsing part Nokogiri can’t parallelize by Ractor.
  - Parallelized only writing, but it to be faster. (less than 8min)
- If you are using Nokogiri, please use this.

enumparse + OX + Ractor

- Focusing on speed, I tried a similar configuration on Ox.
- Using Ractor
  - Parallelize reading, parsing, and writing
- Ox can parallelize by Ractor
- Very fast (about 6min)
- Naturally, CPU usage will increase.

Tried change to Ox's SAX, but ...

- Parsing by Ox is very fast.
- It may be fast reading by Ox's SAX?
- Unfortunately, it was slow
  - Parallelize reading, parsing, and writing
  - Slow... (8min over)

Fastest is enumparse (using libxml’s SAX) + OX + Ractor

- Nokogiri may not be installed due to gem dependencies. Therefore, I created enumparse using libxml's SAX version.
- It can parallelize reading, parsing, and writing
- It’s fastest! (5min34sec)
- If you are not using Nokogiri and focusing on speed, please use this.

Sharing this result enumparse

I released nokogiri-enumparse, and will release libxml_enumparse.

- **nokogiri-enumparse**
  - **GitHub: https://github.com/dogrun-inc/nokogiri-enumparse**
  - **RubyGems: https://rubygems.org/gems/nokogiri-enumparse**
- **libxml_enumparse**
  - **In preparation.**

Thank You, and More Details

If you wish to know more details, I will do the presentation of this contents and details in RubyConf Taiwan 2023.