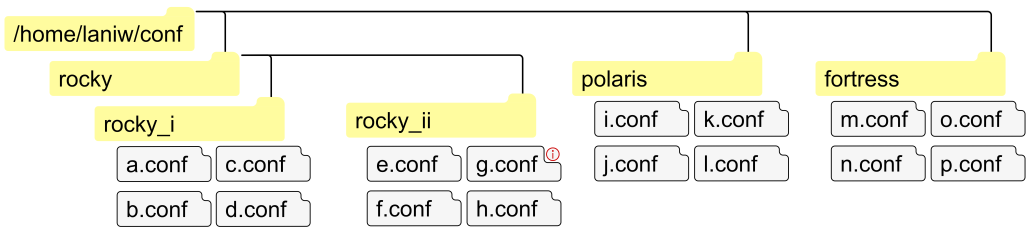
Rotating Index Load Balancer

Documentation



Fachhochschule Nordwestschweiz
Hochschule für Technik

Lani Wagner

2022-05-15 11:57:21

Introduction

This document serves as a documentation for the written implementation of a rotating index load balancer as described in the pre-production specification found at https://github.com/laniwfhnw/engw_specification/blob/main/engw_specification.pdf.

Contents

1 Rotating Index Load Balancer Concept

To explain what the idea of a Rotating Index Load Balancer is, it's easiest to first explain what problem it solves with a simple example. Imagine a filesystem like the one shown in Figure 1. We want to provide a service that accepts analysis requests. These analysis requests are run over the system to calculate a result. The result is then returned to the requester. You can imagine this service being an API, so we don't know when or how many of the requests we're going to get.

The simplest solution, once we get a request, would be to go through all of the files for one request and complete the analysis. Once all of the files have been included in the result we return the result. This works fine when we don't have many requests coming in and the analyses don't need a long time to complete. We are doing a lot of the same operations multiple times, though. Every time we read a file for one analysis and then another and then another we are doing the same operation. If they come in at a similar time we can read the file once to avoid expensive I/O operations.

The idea of reducing the amount of read operations $m \cdot n$, where m is the amount of analyses that need to read n files, to n read operations is the main idea of a Rotating Index Load Balancer . Imagine an index that points to a file at all times and rotates around pointing to all files in the filesystem. Every time it points to a file it reads that file and passes that data on to the analyses. Once a request reaches the service, it keeps track of the first file that service received. That way the service knows the analysis has looked at all files when the index points to the file that the analysis started with.

Let me me walk you through an example execution using the filesystem in Figure 1. We are working with 16 different configuration files. We want to analyze these files in three different ways. The first analysis A_1 would like to know the average length of the files in lines. The second analysis A_2 would like to know what the longest line in any of the files is. Finally, the third A_3 analysis would like to know the size of the files in bytes that are two levels deeper than the root. At first the service isn't doing anything because there are no active analyses running. Once the first analysis A_1 is requested, the index starts rotating and the filecontents are analysed per analysis A_1. The files `a.conf`, `b.conf`, `c.conf`, and `d.conf` are analysed without any other requests coming in. Analysis A_2 arrives right before the index points to `e.conf`, so `e.conf` is the first file A_2 analyses. A_3 comes in when the index points to `g.conf`. The contents of `g.conf` and `h.conf` are sent to all three analyses. The files `i.conf` through `p.conf` are only sent to analyses A_1 and A_2, since analysis A_3 doesn't care about those files. As soon as the index points to `a.conf` the second time the service knows that analysis A_1 has been completed and it returns the result. The rotation continues without interruption until the index points to `e.conf`, at which point analysis A_2 is returned. Now only analysis A_3 remains. It is still around the receive the file contents of `e.conf` and `f.conf`. The result of analysis A_3 is returned as soon as the index points to `g.conf`. Now there are no more analyses in the queue, so the index rests at `g.conf` until another request arrives. The final state of the service is depicted in Figure 1.

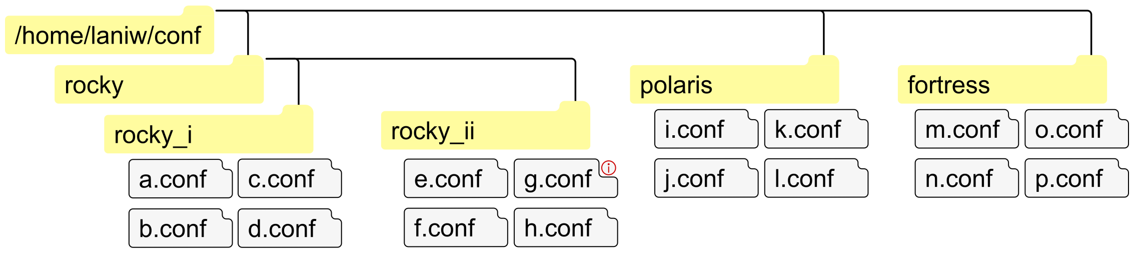


Figure 1: Visualization of a Rotating Index Load Balancer traversion through an example filesystem.

This example obviously doesn't illustrate what happens in certain special cases. For example we didn't consider the case that `g.conf` could get deleted during the rotation of A_3, so with our current approach we would not know when to stop. For more detail on the exact implementation and how special cases are handled please take a look at the pre-production specification of the library[1].

2 Library Use

2.1 Environment

2.2 Use Cases

[1]https://github.com/laniwfhnw/engw_specification/blob/main/engw_specification.pdf

3 Glossary

Term	Definition
Rotation	A single go around the list of files considered for analysis.

Table 1: Glossary definitions.

List of Figures

List of Tables