CWL-Runner performance assessment report

Document Information

Reference Number	POP2_AR_007
Author	Andres CHARIF RUBIAL (UVSQ)
Contributor(s)	
Date	June 12th, 2019

Notices:
The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No "676553".

Table of Contents

1. Background

Applicants Name: Michael R. Crusoe
Applicants Affiliation: Darbo eigos
Application Name: cwltool, the reference CWL runner
Programming Language: Python
Programming Model: PThreads
Source Code Available: Yes
Input data: Yes (provided by user)
Performance study: Performance check (audit)

The user installed the performance tools on its local machine but was not able to generate a trace by himself because his code is a pure Python application. POP Partners' tools do not support properly pure Python applications. Usually Python applications are used for pre/post processing of calls to C/C++/Fortran/etc. libraries. Even though Score-P have preliminary support for Python application, the profiling overhead still requires to be improved and cannot be used in a reasonable amount of time (too many overhead). We then installed and profiled the application on one of our machines (UVSQ) using Python specific profilers.
The user provided a real worst-case scenario input set. He complains about "Slow startup processing and validating of large CWL documents with linked data attributes".
The user's ideal performance target would be to achieve a gain between 10 to 60x.

The performance issue is described on the user's tool website: https://github.com/common-workflow-language/cwltool/issues/1063

2. Application structure

The application uses pthreads to enable parallelism. However, the provided use case calls a part of the program, called validation phase, which does not benefit from pthreads. This part is purely sequential as shown in figure 1.

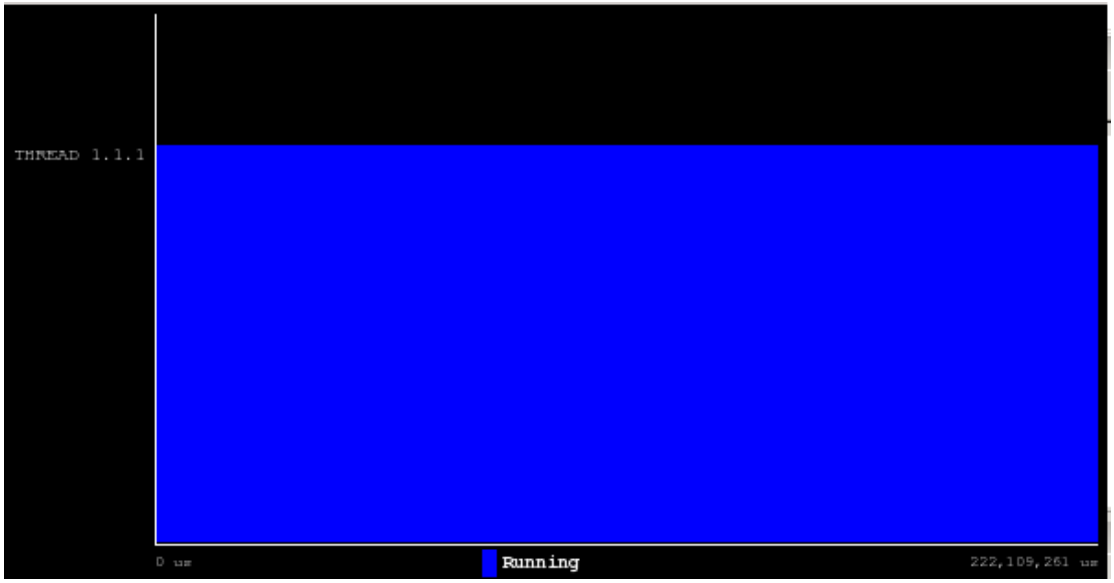

Figure 1: Temporal view of the application (Extrae/Paraver).

3. Focus of Analysis

After confirming with the user that the studies validation phase is purely sequential, we started analysing the outputs.

Figure 2 presents the inclusive time tree view representing the time per call for each invoked function. Few times is spent in the intermediate calls. Most of it is concentrated in the leaves of this view (not terminal leaves). Figure 3 shows that theses leaves hide recursive calls (to the same function). Looking at the source code we can see that some of these recursive functions contain loops which encompasses recursive calls? This means that it may be possible to introduce parallelism. However, this would imply using locks.

After exchanging with the user, it appears that the backtracking approach used, implying recursivity, is maybe not a good option (very expensive). Figure 4 also shows a higher number of calls than expected to the leaves. The provided input case encompasses multiple files. It appears that it is more expansive to process the packed file that each file separately. This will be investigated by the user.

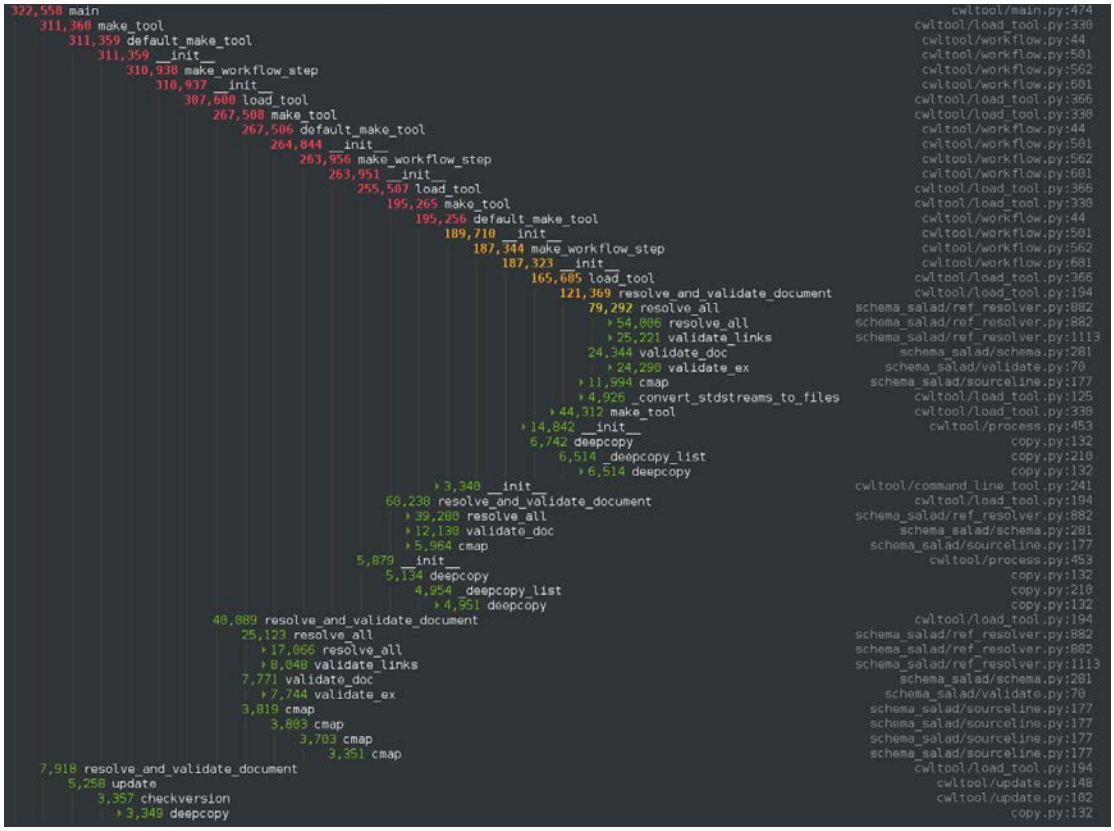

Figure 2: inclusive time tree view – global view (pyinstrument)

Figure 3: inclusive time tree view - zooming on leaves (pyinstrument)

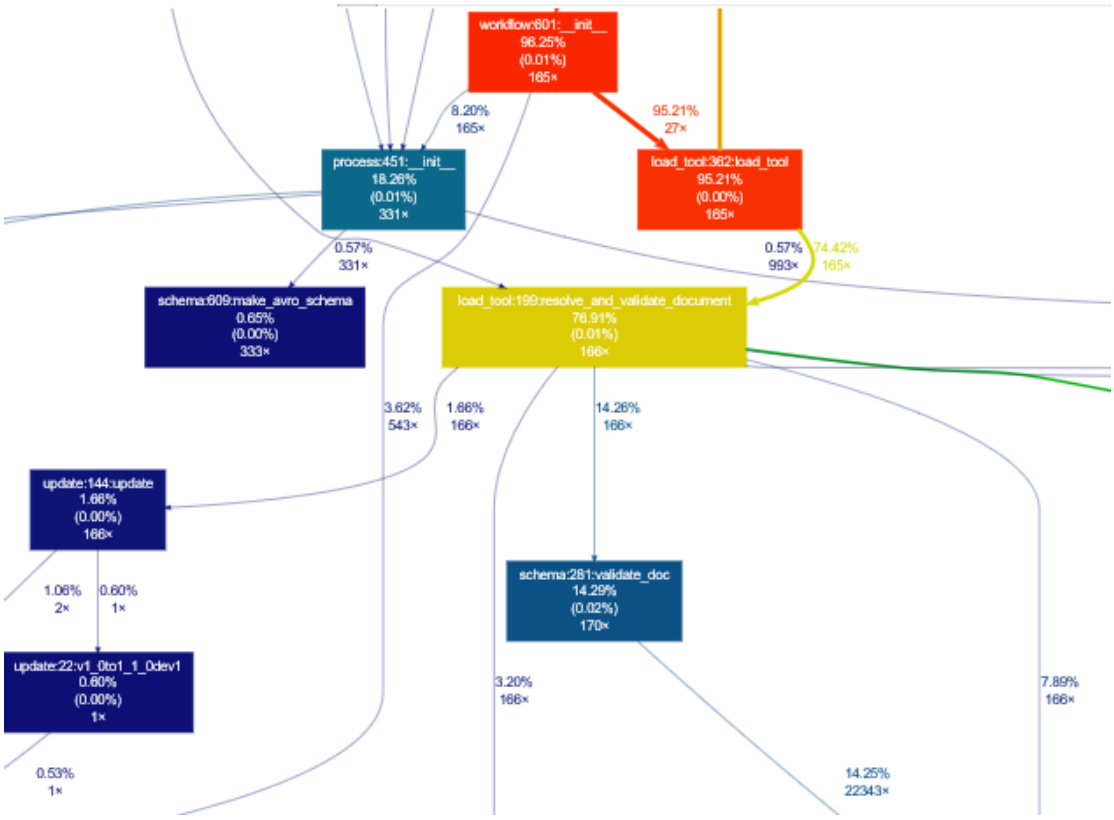

Figure 4: Selected part of the dynamic call tree showing a call to one of the recursive functions

4. Parallel and computation metrics

Since the studied part of the application is purely sequential parallel metrics are not relevant.

Being a pure Python application, hardware counters cannot be used to obtain POP computation metrics.

5. Summary of observations

The part of the application which was analyzed is purely sequential. The only way to reach the target gain expected by the user is to parallelize the hotspots.

We found that there are only a few hotspots which a recursive call. Some of them contain loops that could be parallelized if no dependencies prevent it. However, a better approach would be to change the recursive algorithm to avoid the cost of recursivity.

While checking with the calls count of these hotspots, it seems that there are more calls than expected. This may explain why the provided input case takes more time to process the files it encompasses when compared to the sum of the processing of each file separately.