Investigating the streamability of XProc pipelines

Norman Walsh

Principal Technologist
Mark Logic Corporation

What is XProc?

- An extensible vocabulary for describing a sequence of operations
  - Over an XML document or sequence of documents
- XProc has the expected control structures
  - Loops
  - Conditionals
  - Exception handling
- Under development at the W3C. Will be done *real soon now!*

Canonical XProc pipeline

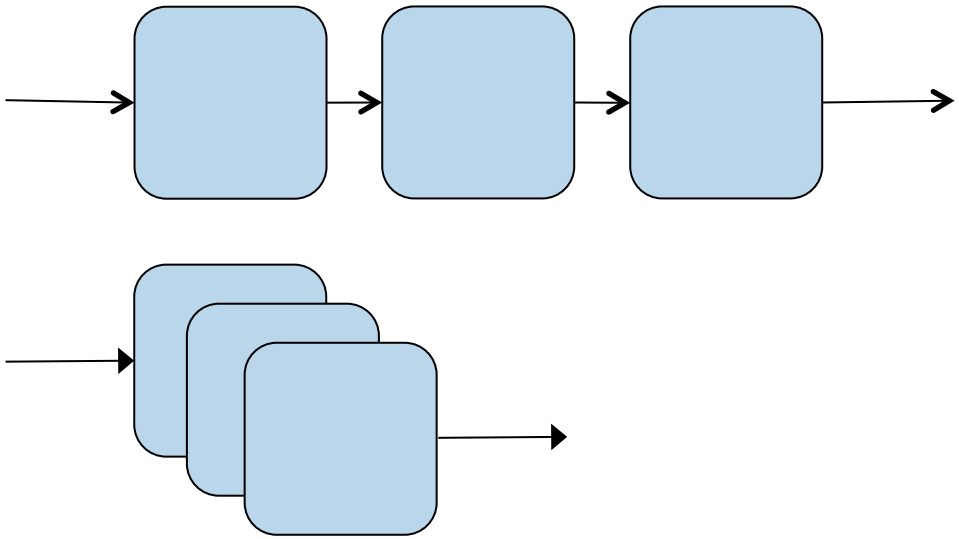
```
<p:pipeline xmlns:p="http://www.w3.org/ns/xproc">
  <p:xinclude/>
  <p:validate-with-xml-schema>
    <p:input port="schema">
      <p:document href="/uri/of/schema.xsd"/>
    </p:input>
  </p:validate-with-xml-schema>
  <p:xslt>
    <p:input port="stylesheet">
      <p:document href="/uri/of/stylesheet.xsl"/>
    </p:input>
  </p:xslt>
</p:pipeline>
```

What do we mean by “streaming”?

- A streaming process can perform its task in a single pass over only a sliding window of input, possibly never having simultaneous access to all parts of the input
  - Copy the input
  - Delete all attributes named “foo” on elements named “ex:bar”
  - Change the name of all “h:p” elements that occur as children of “h:td” elements to “h:div”.
  - Count all elements named “h:div”
  - ~~Add an attribute to the root element which contains a count of the number of text nodes in the document~~
- A streaming process may be able to begin producing output before it has seen all of the input
  - ~~Count all elements named “h:div”~~

Why stream?

- To process documents of arbitrary, perhaps unbounded size
  - No matter how much memory you have, I can construct a document that's bigger
- To produce results in parallel with the input
  - When synchrony is part of the larger goal
  - When it will achieve improved "end-to-end" performance



XProc steps

p:add-attribute
p:add-xml-base
p:compare
p:count
p:delete
p:directory-list
p:error
p:escape-markup
p:exec*
p:filter
p:hash*
p:http-request
p:identity
p:insert
p:label-elements
p:load
p:make-absolute-uris
p:namespace-rename
p:pack
p:parameters
p:rename
p:replace
p:set-attributes
p:sink
p:split-sequence
p:store
p:string-replace
p:unescape-markup
p:unwrap
p:uuid*
p:validate-with-relax-ng*
p:validate-with-schematron*
p:validate-with-xml-schema*
p:wrap
p:wrap-sequence
p:www-form-urldecode*
p:www-form-urlencode*
p:xinclude
p:xquery*
p:xsl-formatter*
p:xslt

*optional step

Streamability

All implementations must stream

Problem

We’re writing a specification,
not an implementation

Streamability

All steps must be streamable by definition

Problems

What about XSLT, XQuery, etc.?

Streaming in XProc

- The specification doesn't actually say much about streaming
- The WG tried to make steps "amenable to streaming"
- We highlighted a couple of places where it can be problematic, at least in the general case
  - References to context position and size
  - Particularly with respect to sequences of documents
- There's no requirement for steps or implementations to stream

So…

- We don't mandate streaming
- But we expect, in theory, streaming to improve performance
- Except, of course, where pipelines use steps that can't stream
- So:

To what extent does streaming matter in the XProc pipelines that people actually write?

What's streamable?

- Some steps aren't streamable, p:xslt
- Some steps are always streamable, p:count
- Some steps are often streamable, p:delete
- Let's be optimistic: assume if a step can be streamable, it will stream
  - 15,503 unique XPath expressions
  - 12,477 (~80%) are trivially streamable
    - trivially = identified by a dozen or so regular expressions

What's not streamable?

- What won't stream?
  - p:exec
  - p:http-request
  - p:validate-with-relaxng,
  - p:validate-with-schematron
  - p:validate-with-xml-schema
  - p:xquery
  - p:xslt

Exploring the question

- XML Calabash (http://xmlcalabash.com) is the author's XProc implementation
- It's written in Java, built on top of Saxon 9
- It evaluates all pipelines in a non-streaming, single-threaded fashion
- Motivations for XML Calabash:
  - Helping the author understand the specification
  - Completeness and correctness: satisfying CR exit criteria so we can be *done*!
- In aid of answering the question this paper asks, XML Calabash has a "phone home" feature
  - Opt-out, so unknown subset of total use
  - Doesn't include any of *my* pipelines

Static pipeline information


```
<pipeline-run-report xmlns='http://xmlcalabash.com/ns/phonehome'>
<datetime>2009-06-05T03:47:12Z</datetime>
<ipaddr>1.2.3.4</ipaddr>
<pipeline-report xmlns='http://xmlcalabash.com/ns/phonehome'>
<version>1.0</version>
<product-name>XML Calabash</product-name>
<product-version>0.9.7</product-version>
<episode>CBb05f50b9afddf1324ce18ce892c310</episode>
<step type='{http://www.w3.org/ns/xproc}declare-step' declared-
   type='{http://xmlcalabash.com/ns/extensions}anonymousType_153'
   name='pipeline'>
  <input port='source'/>
  <input port='parameters' kind='parameter'>
    <no-binding/>
  </input>
  …
```

Pipeline trace

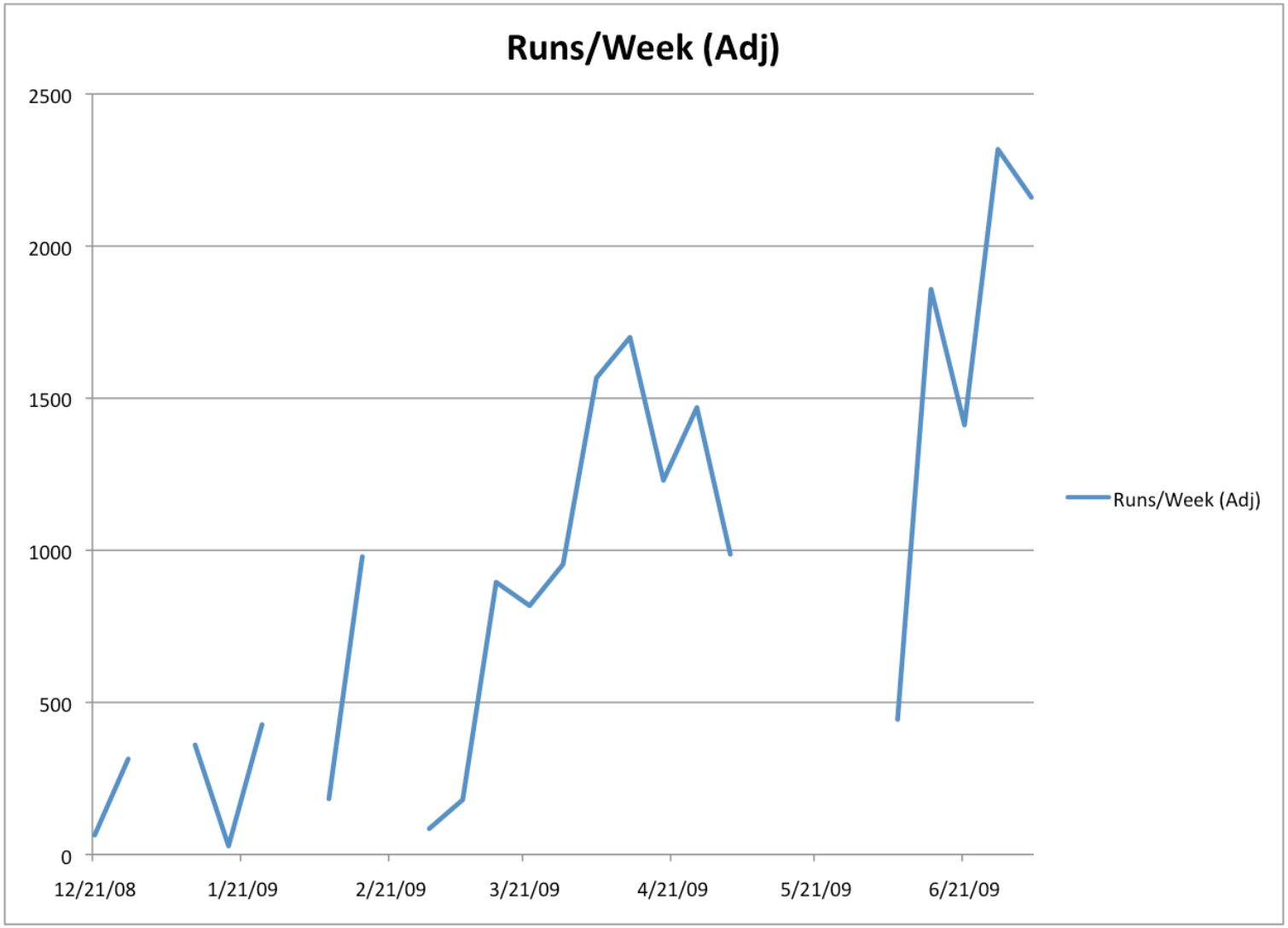
```
<pipeline-run-report xmlns='http://xmlcalabash.com/ns/phonehome'>
<datetime>2009-06-05T03:47:14Z</datetime>
<ipaddr>1.2.3.4</ipaddr>
<runtime-report xmlns='http://xmlcalabash.com/ns/phonehome'>
<version>1.0</version>
<product-name>XML Calabash</product-name>
<product-version>0.9.7</product-version>
<episode>CBb05f50b9afddf1324ce18ce892c310</episode>
<steps>
  <step type='{http://example.com/xproc}user-defined'/>
  <step type='{http://www.w3.org/ns/xproc}xslt'/>
</steps>
</runtime-report>
</pipeline-run-report>
```

Data collected

Collected data from 21 Dec 2008 to 5 July 2009

Week	Number of runs
21 Dec 2008	64
28 Dec 2008	314
4 Jan 2009	2539
11 Jan 2009	360
…	…
24 May 2009	152968
…	…
14 Jun 2009	1858
21 Jun 2009	1412
28 Jun 2009	2318
5 July 2009	2160

Data collected



Pipelines collected

- Collected 294,892 runs
  - 369 static errors
  - 7,778 dynamic errors
- Many pipelines run a single step
- A few ran more than 50,000 steps
- Only 3,994 “distinct” pipelines
  - Same IP address
  - Same steps
  - Same order

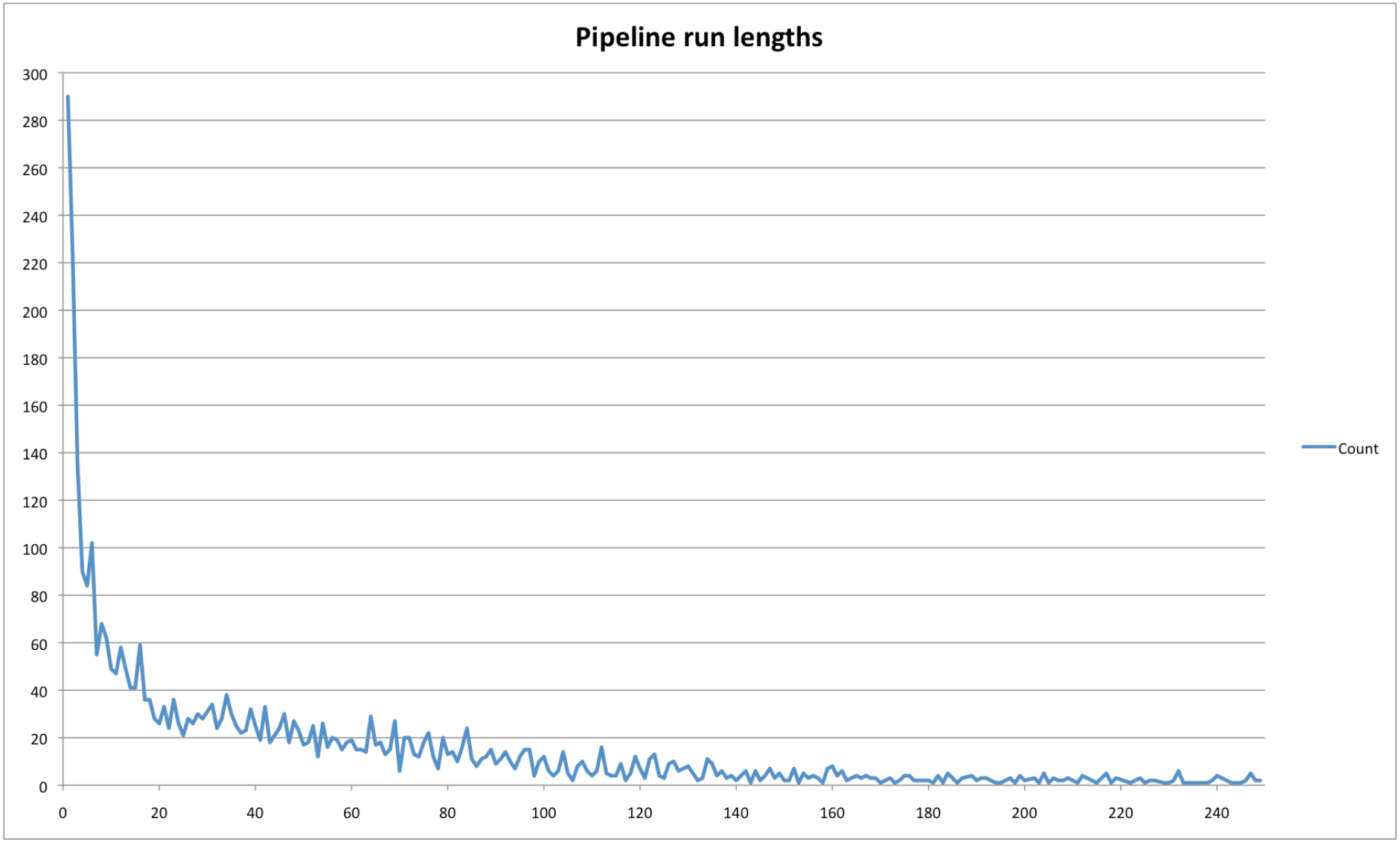
What's this "distinct" business?

- Suppose there were 1,000 runs and 900 were the same?
- If you treat that as 1,000 pipelines…
  - The pipeline run 900 times will overwhelm the others
- If you treat that as 101 distinct pipelines…
  - The *importance* of the pipeline run 900 times is ignored
- For the purpose of this paper, we're interested in distinct pipelines

Steps used

Count	%	Step name
284162	11.4382	xslt
275201	11.0775	identity
274397	11.0451	delete
272178	10.9558	string-replace
296066	10.8306	rename
263898	10.6225	unwrap
254872	10.2592	sink
245156	9.8681	add-attribute
206669	8.3189	pack
56046	2.2560	store
20389	0.8207	wrap
18284	0.7576	http-request
6225	0.7360	set-attribute

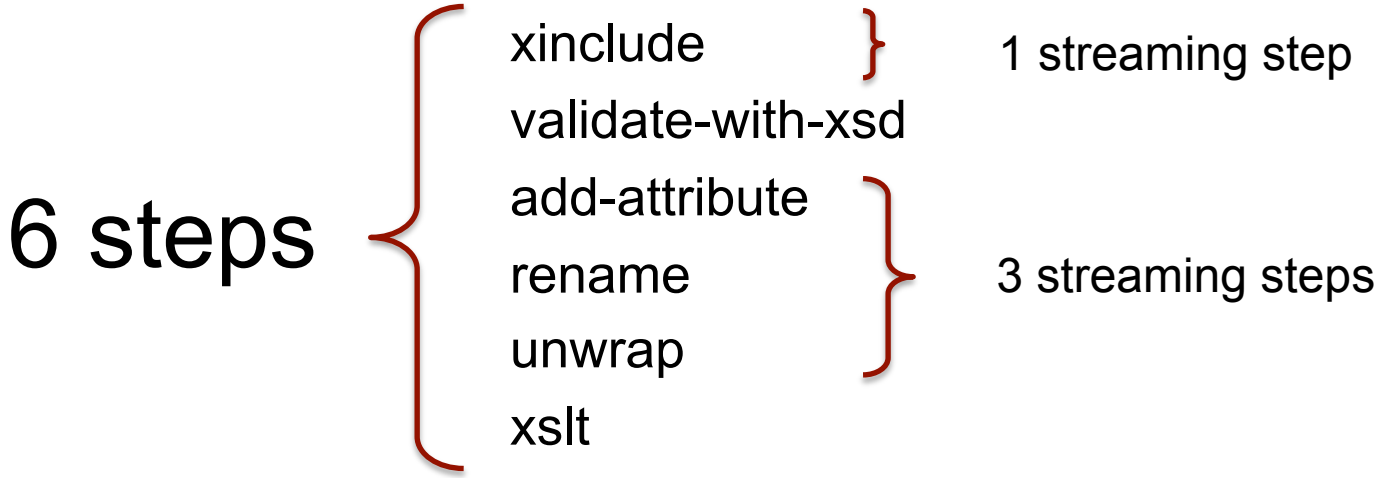
Absolute run length



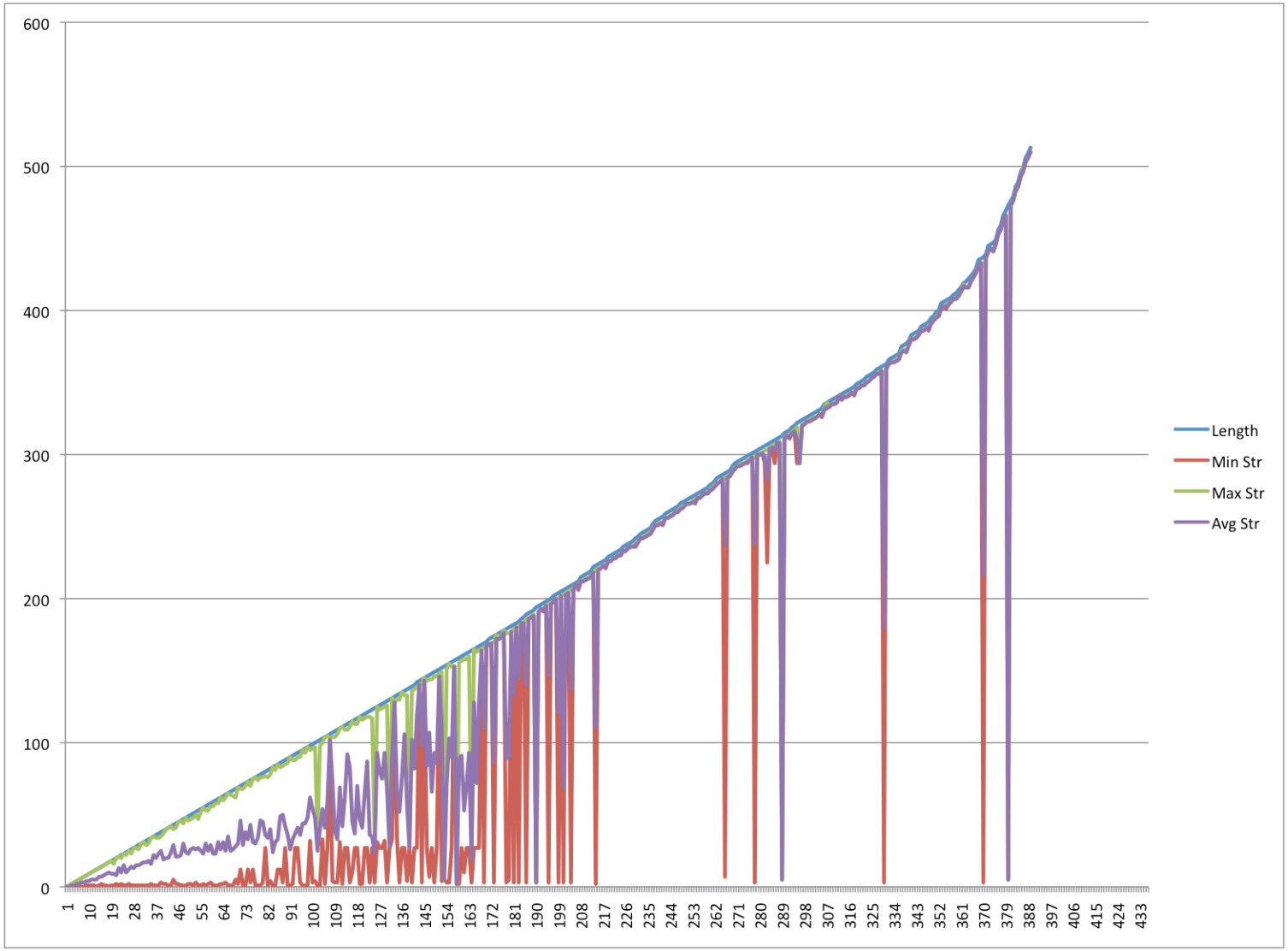
What about streaming?

- For a given pipeline we can look at
  - Its absolute length
  - The longest run of streamable steps
  - The shortest run of streamable steps
  - The average length of streamable steps

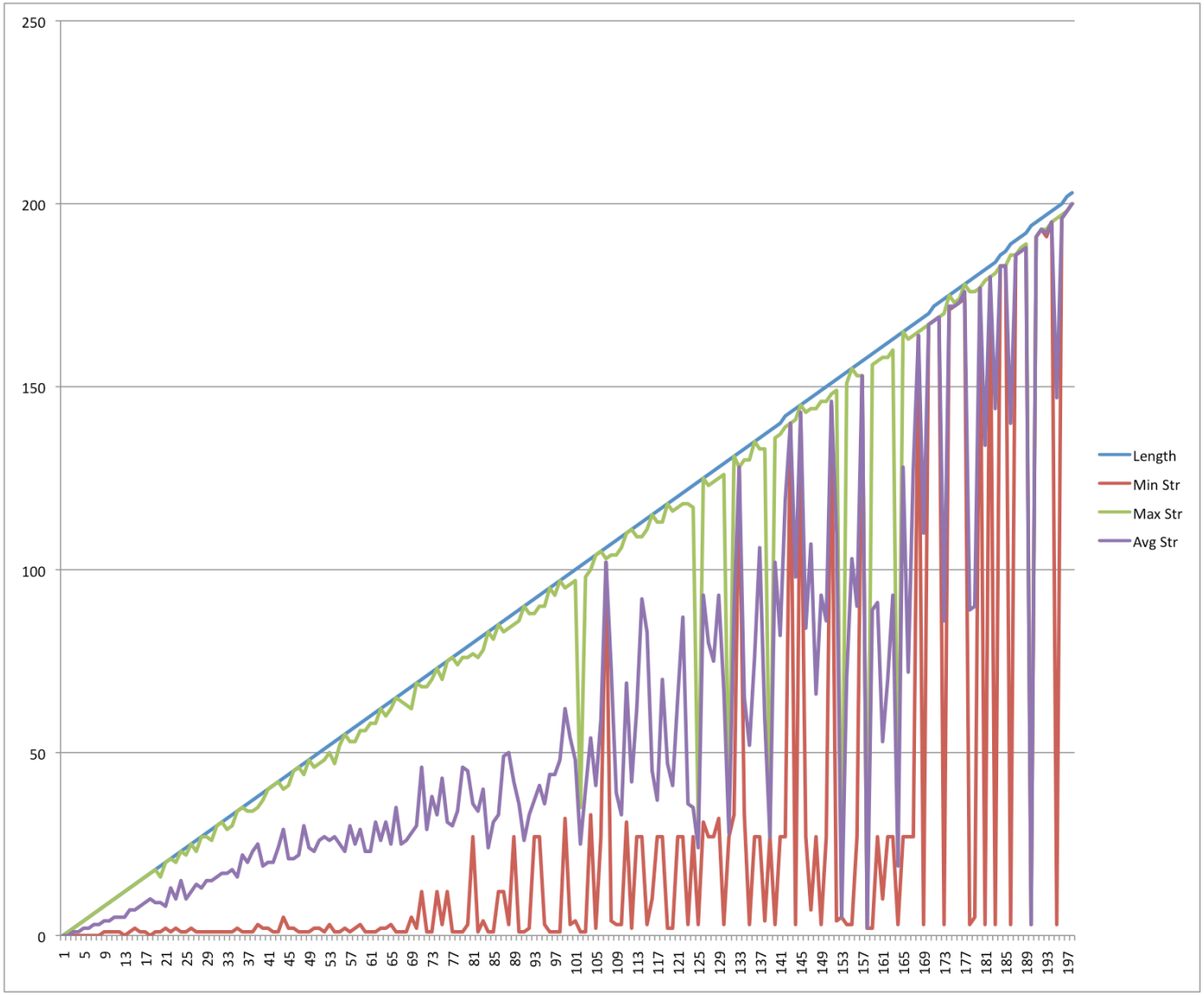
Run lengths



Streamable subpipeline length



Streamable subpipeline length (<200)



Conclusions?

- Most pipelines are short, so maybe streaming doesn't matter?
- On average, half of a pipeline can be streamed, so maybe it does?